Java

TRÅDAR OCH GRAFIK

Exempel

FADIL GALJIĆ

Studentlitteratur

Innehållsförteckning

Kapitel 1
Trådar

Ett program med flera trådar
Definiera, skapa och starta en tråd • Flera trådar av samma typ
En subklass till klassen Thread • Definiera en tråd lokalt

Operationer på en tråd
Identifiera en tråd • Uppehåll i en tråd
En demontråd • En tråds prioritet

Kontrollera en tråds aktivitet
Avbryta en tråd • Signal till en tråd • Signal till en grupp trådar

Synkroniserad användning av ett objekt
Osynkroniserad användning av ett objekt • Helt synkroniserade objekt
Sammansatta synkroniserade operationer • Optimera synkroniseringen
Synkronisering på klientsidan • Synkroniserade behållare

Synkroniserad användning av flera objekt
Ett dödläge

Synkronisering av tillståndsberoende operationer
Tillståndsberoende operationer • Synkronisera tillståndsberoende operationer
Synkronisering på klientsidan

Synkroniseringsobjekt
Synkroniseringslås • Läslås och skrivlås • Villkorsobjekt

Kommunikation mellan trådar
Objektöverföring via en kanal • Respons från mottagaren

Ett program med flera trådar

Definiera, skapa och starta en tråd

Skrivare1.java

En klass som definierar en tråd

En tråd (en aktivitet som kan exekveras parallellt med andra aktiviteter i ett program) kan definieras i en klass som implementerar gränssnittet `java.lang.Runnable`. Metoden `run` implementeras i den klassen, och i denna metod definieras den aktivitet som ska utföras parallellt med andra aktiviteter.

```
// Skrivare1 definierar en tråd, som skriver ut ett meddelande
// ett antal gånger
class Skrivare1 implements Runnable
{
    // en aktivitet som kan utföras
    // parallellt med andra aktiviteter
    public void run ()
    {
        for (int i = 0; i < 5; i++)
            System.out.println ("gyllene");
    }
}
```

Skrivare2.java

En klass som definierar en tråd

Klassen `Skrivare2` implementerar gränssnittet `java.lang.Runnable`, och definierar på så sätt en tråd.

```
// Skrivare2 definierar en tråd, som skriver ut ett meddelande
// ett antal gånger
class Skrivare2 implements Runnable
{
    public void run ()
    {
        for (int i = 0; i < 5; i++)
            System.out.println ("gryningen");
    }
}
```

DefinieraSkapaOchStartaTradar.java

Ett program med flera trådar

En tråd (ett objekt av typen `java.lang.Thread`) kan skapas utifrån ett objekt av en klass som implementerar gränssnittet `java.lang.Runnable`. Denna tråd kan sedan startas. Tråden blir körbar, och får så småningom möjlighet att exekvera koden i klassens `run`-metod.

Ett Javaprogram kan innehålla flera trådar. I detta fall får varje tråd möjlighet att exekvera motsvarande kod. Vanligtvis växlar operativsystemet snabbt mellan olika trådar, så att flera aktiviteter kan utföras samtidigt.

På en dator med flera processorer kan varje tråd exekveras i sin egen processor. Om datorn har en enda processor, växlar operativsystemet mellan olika trådar. Varje tråd får möjlighet att exekvera under en viss tid. Det går inte att förutsäga när en viss tråd ska få möjlighet att exekvera, och när tråden ska avbrytas. Det är operativsystemet på värddatorn som bestämmer hur de olika trådarna i ett program ska schemaläggas.

```
// DefinieraSkapaOchStartaTradar skapar och startar två trådar
// av olika typer
class DefinieraSkapaOchStartaTradar
{
    public static void main (String[] args)
    {
        System.out.println ("DEFINIERA, SKAPA OCH STARTA TRÅDAR");
        System.out.println ();

        // skapa två trådar av olika typer

        // skapa en tråd som representerar den aktivitet som
        // definieras i run-metoden i klassen Skrivare1
        Skrivare1    s1 = new Skrivare1 ();
        Thread       t1 = new Thread (s1);
        // det går att skriva kortare:
        // Thread    t1 = new Thread (new Sktivare1 ());

        // skapa en tråd som representerar den aktivitet som
        // definieras i run-metoden i klassen Skrivare2
        Skrivare2    s2 = new Skrivare2 ();
        Thread       t2 = new Thread (s2);

        // starta trådarna
        t1.start ();
        t2.start ();
        // Trådarna t1 och t2 är nu körbara, och kan när som helst
        // få möjlighet att exekvera motsvarande kod.
```

```
        // En tråd exekverar den kod som anges i metoden run i
        // definitionsklassen för den tråden.

        // main-tråden avslutas, men programmet
        // fortlever i trådarna t1 och t2.
        // En tråd avslutas när motsvarande run-metod avslutas.

        // en tråd kan även skapas och startas så här:
        // new Thread (new Skrivare1 ()).start ();
    }
}
```

Programmets utmatning vid en exekvering

```
DEFINIERA, SKAPA OCH STARTA TRÅDAR

gryningen
gryningen
gryningen
gryningen
gryningen
gyllene
gyllene
gyllene
gyllene
gyllene
```

Programmets utmatning vid en andra exekvering

```
DEFINIERA, SKAPA OCH STARTA TRÅDAR

gyllene
gyllene
gyllene
gyllene
gyllene
gryningen
gryningen
gryningen
gryningen
gryningen
```

Programmets utmatning vid en tredje exekvering

```
DEFINIERA, SKAPA OCH STARTA TRÅDAR

gyllene
gryningen
gyllene
gryningen
gyllene
gryningen
```

```
gyllene
gryningen
gyllene
gryningen
```

Programmets utmatning vid en fjärde exekvering

```
DEFINIERA, SKAPA OCH STARTA TRÅDAR

gryningen
gryningen
gryningen
gryningen
gyllene
gryningen
gyllene
gyllene
gyllene
gyllene
```

Flera trådar av samma typ

Skrivare.java

En klass som definierar en typ av trådar

En klass som implementerar gränssnittet `java.lang.Runnable` kan erhålla vissa uppgifter utifrån. Man kan därmed påverka beteendet hos en tråd som skapas utifrån denna klass.

```
// En tråd av typen Skrivare skriver ut ett givet meddelande ett
// angivet antal gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    // den aktivitet som utförs när en tråd skapas utifrån
    // ett objekt av typen Skrivare och den tråden startas
    public void run ()
    {
```

```
        for (int i = 0; i < 4; i++)
            System.out.println (meddelande);
    }

    // En klass som definierar en tråd kan även ha andra metoder
    // förutom metoden run. Dessa metoder kan vara hjälp-metoder,
    // som anropas via metoden run.

    // För att i en annan klass kunna anropa eventuella
    // andra instansmetoder i den här klassen, behövs det
    // ett objekt av den här klassen i den andra klassen
}
```

FleraTradarAvSammaTyp.java

Ett program med flera trådar av samma typ

Det är möjligt att skapa flera trådar utifrån en klass som definierar en typ av trådar.

```
// FleraTradarAvSammaTyp skapar och startar
// flera trådar av samma typ
class FleraTradarAvSammaTyp
{
    public static void main (String[] args)
    {
        System.out.println ("FLERA TRÅDAR AV SAMMA TYP");
        System.out.println ();

        // skapa en vektor av trådar av samma typ
        Thread[]    t = new Thread[4];
        t[0] = new Thread (new Skrivare ("öst"));
        t[1] = new Thread (new Skrivare ("väst"));
        t[2] = new Thread (new Skrivare ("syd"));
        t[3] = new Thread (new Skrivare ("nord"));

        // starta trådarna
        for (int i = 0; i < t.length; i++)
            t[i].start ();

        // meddelandet från main-tråden
        for (int i = 0; i < 4; i++)
            System.out.println ("i mitten");
        // main-tråden är nu en av flera trådar i programmet,
        // och exekverar parallellt med andra trådar
    }
}
```

Programmets utmatning vid en exekvering

```
FLERA TRÅDAR AV SAMMA TYP

i mitten
i mitten
väst
syd
nord
i mitten
öst
väst
syd
nord
i mitten
öst
väst
syd
nord
öst
väst
syd
nord
öst
```

En subklass till klassen Thread

Skrivare.java

En klass som definierar en tråd genom att ärva från klassen Thread

En klass som definierar en typ av trådar kan skapas antingen genom att implementera gränssnittet `java.lang.Runnable`, eller genom att ärva från klassen `java.lang.Thread` och omdefiniera metoden `run`.

En klass kan inte ärva från klassen `Thread`, om den redan ärver från en annan klass. Gränssnittet `Runnable` måste implementeras, om en typ av trådar ska definieras i klassen.

```
// Skrivare definierar en tråd som skriver ut ett givet meddelande
// ett angivet antal gånger
class Skrivare extends Thread
// en subklass till klassen java.lang.Thread
{
    // ett meddelande
    private String    meddelande;
```

```
    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    // omdefiniera metoden run
    public void run ()
    {
        for (int i = 0; i < 5; i++)
            System.out.println (meddelande);
    }
}
```

EnSubklassTillKlassenThread.java

Ett program som skapar flera trådar av en subklass till klassen Thread

En tråd kan skapas genom att ett objekt av en subklass till klassen `java.lang.Thread` skapas (en tråd kan dessutom skapas utifrån en klass som implementerar gränssnittet `java.lang.Runnable`).

```
// EnSubklassTillKlassenThread skapar och startar flera trådar
class EnSubklassTillKlassenThread
{
    public static void main (String[] args)
    {
        System.out.println ("EN SUBKLASS TILL KLASSEN Thread");
        System.out.println ();

        // skapa två trådar av typen Skrivare
        // (ett objekt av typen Skrivare är en tråd, ett objekt av
        // typen Thread)
        Skrivare    s1 = new Skrivare ("gyllene");
        Skrivare    s2 = new Skrivare ("gryningen");

        // starta trådarna
        // (klassen Skrivare ärver metoden start från
        // klassen Thread)
        s1.start ();
        s2.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
EN SUBKLASS TILL KLASSEN Thread
```

```
gyllene
gyllene
gyllene
gyllene
gyllene
gryningen
gryningen
gryningen
gryningen
gryningen
```

Programmets utmatning vid en annan exekvering

```
EN SUBKLASS TILL KLASSEN Thread

gryningen
gryningen
gryningen
gryningen
gyllene
gryningen
gyllene
gyllene
gyllene
gyllene
```

Definiera en tråd lokalt

DefinieraEnTradLokalt.java

Ett program som definierar trådar lokalt

En trådtyp kan definieras i en klass, och en eller flera trådar kan skapas utifrån denna klass i en annan klass.

I vissa fall är definitionsklassen för en tråd relativt enkel och tänkt att användas bara en gång (för att skapa en konkret tråd i en konkret klass). Den klassen kan i så fall definieras lokalt, på den plats där ett objekt av klassen skapas.

En tråd kan definieras i en lokal, anonym (namnlös) klass.

```
// DefinieraEnTradLokalt definierar, skapar och startar två trådar
class DefinieraEnTradLokalt
{
    public static void main (String[] args)
    {
```

```
        System.out.println ("DEFINIERA EN TRÅD LOKALT");
        System.out.println ();

        // definiera en klass som implementerar
        // gränssnittet Runnable, och skapa ett
        // objekt av den klassen
        // (klassen definieras som en lokal, anonym klass)
        Runnable    r = new Runnable ()
            {
                public void run ()
                {
                    for (int i = 0; i < 5; i++)
                        System.out.println ("gyllene");
                }
            };

        // skapa en tråd utifrån ett Runnable-objekt
        Thread    t1 = new Thread (r);

        // definiera en klass som implementerar
        // gränssnittet Runnable, skapa ett objekt
        // av den klassen, och skapa en tråd utifrån objektet
        // (klassen definieras som en lokal, anonym klass)
        Thread    t2 = new Thread (new Runnable ()
            {
                public void run ()
                {
                    for (int i = 0; i < 5; i++)
                        System.out.println ("gryningen");
                }
            } );

        // starta trådarna
        t1.start ();
        t2.start ();

/*************************************************************

        // en subklass till klassen Thread
        // kan även skapas lokalt:
        Thread    t = new Thread ()
            {
                public void run ()
                {
                    for (int i = 0; i < 16; i++)
                        System.out.println ("gyllene");
                }
            };

        t.start ();
```

```
        // definiera, skapa och starta en tråd i en enda sats
        (new Thread ()
            {
                public void run ()
                {
                    for (int i = 0; i < 16; i++)
                        System.out.println ("gryningen");
                }
            }).start ();

************************************************************/
    }
}
```

Programmets utmatning vid en exekvering

```
DEFINIERA EN TRÅD LOKALT

gyllene
gyllene
gyllene
gyllene
gyllene
gryningen
gryningen
gryningen
gryningen
gryningen
```

Programmets utmatning vid en annan exekvering

```
DEFINIERA EN TRÅD LOKALT

gyllene
gyllene
gyllene
gryningen
gyllene
gryningen
gyllene
gryningen
gryningen
gryningen
```

Operationer på en tråd

Identifiera en tråd

IdentifieraEnTrad.java

Ett program som visar hur en tråd identifieras

Flera trådar kan exekvera koden i en och samma `run`-metod. Varje tråd skapar och använder sina egna variabler i metoden. De kan inte förväxlas på något sätt.

Man kan erhålla en referens till den tråd som vid ett visst tillfälle exekverar koden i `run`-metoden. Denna referens kan sedan användas för att anropa en metod i samband med den exekverande tråden.

```
// Identifierare definierar en tråd, som skriver ut sitt namn till
// standardutmatningsenheten
class Identifierare implements Runnable
{
    public void run ()
    {
        // referens till den tråd som exekverar den här koden
        Thread    t = Thread.currentThread ();

        // trådens namn
        String    namn = t.getName ();
        // metodanrop kan bindas:
        // String    namn = Thread.currentThread ().getName ();

        // visa namnet
        System.out.println (namn);

        // t och namn är lokala variabler i den här metoden.
        // Varje tråd som exekverar (passerar genom) den här koden
        // skapar sin egen uppsättning av dessa variabler.
        // Dessa variabler kan inte förväxlas, oavsett på vilken
        // plats en tråd tillfälligt avbryts.
    }
}

// IdentifieraEnTrad skapar två trådar av typen Identifierare,
// och startar dessa trådar
class IdentifieraEnTrad
{
```

```
    public static void main (String[] args)
    {
        System.out.println ("IDENTIFIERA EN TRÅD");
        System.out.println ();

        // skapa en tråd med ett givet namn
        Thread  t1 = new Thread (new Identifierare (), "tråd t1");

        // skapa en tråd, och sätt dess namn efteråt
        Thread    t2 = new Thread (new Identifierare ());
        t2.setName ("tråd t2");

        // starta trådarna
        t1.start ();
        t2.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
IDENTIFIERA EN TRÅD
tråd t2
tråd t1
```

Programmets utmatning vid en annan exekvering

```
IDENTIFIERA EN TRÅD

tråd t1
tråd t2
```

Uppehåll i en tråd

VantaEnAngivenTid.java

Ett program som illustrerar hur en tråd väntar en angiven tid

Ett uppehåll kan införas i en tråds exekvering. Detta uppehåll anges när tråden definieras.

Ett uppehåll i en tråd kan åstadkommas med metoden `sleep` i klassen `Thread` (en statisk metod). Uppehållstiden anges som argument till denna metod. Den tråd som anropar metoden `sleep` blockeras under den angivna tiden. Den får möjlighet att exekvera först efter att denna tid har gått.

Metoden `sleep` används av två anledningar:

1. För att justera en tråds hastighet

2. För att ge även andra trådar möjlighet att exekvera (så att de inte blir helt beroende av operativsystemets schemaläggning)

```
// Skrivare definierar en tråd, som skriver ut
// ett givet meddelande exakt 4 gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // skriv ut meddelandet
            System.out.println (meddelande);

            // gör ett uppehåll i exekveringen
            try
            {
                Thread.sleep (500);
                // Den tråd som utför den här satsen blockeras
                // under 500 millisekunder. Efter den tiden blir
                // tråden körbar igen (tråden kan när som helst
                // få möjlighet att exekvera).
            }
            catch (InterruptedException e)
            // ett checked undantag
            {
                // undantaget kastas inte om metoden interrupt
                // inte anropas i samband med den här tråden
            }
        }
    }
}

// VantaEnAngivenTid skapar fyra trådar av typen Skrivare,
// och startar dessa trådar
class VantaEnAngivenTid
{
    public static void main (String[] args)
    {
        System.out.println ("VÄNTA EN ANGIVEN TID");
```

```
        System.out.println ();

        // skapa fyra trådar
        Thread    t1 = new Thread (new Skrivare ("öst"));
        Thread    t2 = new Thread (new Skrivare ("väst"));
        Thread    t3 = new Thread (new Skrivare ("syd"));
        Thread    t4 = new Thread (new Skrivare ("nord"));

        // starta trådarna
        t1.start ();
        t2.start ();
        t3.start ();
        t4.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
VÄNTA EN ANGIVEN TID

öst
väst
syd
nord
väst
öst
syd
nord
öst
väst
syd
nord
väst
öst
nord
syd
```

Programmets utmatning vid en annan exekvering

```
VÄNTA EN ANGIVEN TID

öst
nord
väst
syd
väst
nord
öst
syd
väst
öst
nord
```

```
syd
väst
syd
nord
öst
```

VantaPaEnTrad.java

Ett program som illustrerar hur en tråd väntar på en tråd

En tråd kan vänta tills en annan tråd avslutar sin exekvering.

```
// Skrivare definierar en tråd, som skriver ut ett givet
// meddelande exakt 4 gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // skriv ut meddelandet
            System.out.println (meddelande);

            // gör ett uppehåll
            try
            {
                Thread.sleep (500);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// VantaPaEnTrad skapar en tråd, och väntar tills den avslutar
// sin exekvering
class VantaPaEnTrad
{
    public static void main (String[] args)
```

```
    {
        System.out.println ("VÄNTA PÅ EN TRÅD");
        System.out.println ();

        System.out.println ("start\n");

        // skapa och starta en tråd
        Thread    t = new Thread (new Skrivare ("0 1 2 3 4"));
        t.start ();

        // vänta tills tråden t avslutar sin aktivitet
        try
        {
            t.join ();
            // den tråd som anropar metoden join (main-tråden)
            // blockeras, och förblir blockerad så länge tråden t
            // existerar
        }
        catch (InterruptedException e)
        {
            // väntandet avbryts om metoden interrupt anropas i
            // samband med den väntande tråden
        }

        System.out.println ("\nslut");
    }
}
```

Programmets utmatning

```
VÄNTA PÄ EN TRÅD

start

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4

slut
```

En demontråd

EnDemonTrad.java

Ett program som illustrerar demontrådar

En tråd kan göras till en demontråd. En demontråd är en tråd som på något sätt betjänar andra trådar i programmet. När dessa andra trådar avslutas, är det inte någon mening med att demontråden fortsätter exekvera. Tråden avslutas automatiskt, och hela programmet avslutas.

Det kan finnas flera demontrådar i ett och samma program.

```
// Understrykare definierar en tråd, som skriver ut en
// understrykningslinje i en oändlig loop
class Understrykare implements Runnable
{
    public void run ()
    {
        while (true)
        {
            // understryk
            System.out.println ("-----------------");

            // gör ett uppehåll
            try
            {
                Thread.sleep (500);

            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Skrivare definierar en tråd, som skriver ut
// ett givet meddelande exakt 5 gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }
```

```
    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // skriv ut meddelandet
            System.out.println (meddelande);

            // gör ett uppehåll
            try
            {
                Thread.sleep (500);

            }
            catch (InterruptedException e)
            {}
        }
    }
}

// EnDemonTrad skapar en användartråd och en demontråd,
// och startar dessa trådar
class EnDemonTrad
{
    public static void main (String[] args)
    {
        System.out.println ("EN DEMONTRÅD");
        System.out.println ();

        // skapa en tråd (en användartråd)
        Thread    t1 =
               new Thread (new Skrivare ("gyllene gryningen"));

        // skapa en demontråd
        Thread    t2 = new Thread (new Understrykare ());
        t2.setDaemon (true);
        // t2 blir en demontråd. Den avslutas ungefär
        // samtidigt som alla användartrådar (icke-demontrådar)
        // i programmet avslutas (när main och t1 avslutas).

        // setDaemon måste komma innan motsvarande tråd startas,
        // annars kastar metoden start ett undantag av typen
        // java.lang.IllegalThreadStateException

        // starta trådarna
        t1.start ();
        t2.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
EN DEMONTRÅD

gyllene gryningen
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
-----------------
```

Programmets utmatning vid en annan exekvering

```
EN DEMONTRÅD

gyllene gryningen
-----------------
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
gyllene gryningen
-----------------
```

En tråds prioritet

EnTradsPrioritet.java

Ett program som illustrerar prioriteter för olika trådar

Olika trådar i ett program kan ha olika prioritet. Den lägsta prioriteten för en tråd är `1`, den förvalda prioriteten är `5` och den högsta prioriteten är `10`. Om prioriteten för en tråd inte anges explicit, blir den automatiskt densamma som prioriteten för trådens skapare (som är också en tråd).

Om det finns flera körbara trådar vid ett tillfälle, får de trådar som har högst prioritet möjlighet att exekvera. De andra trådarna måste vänta. En tråd med en mindre prioritet kan få möjlighet att exekvera när alla trådar med högre prioriteter är blockerade.

Vissa operativsystem stödjer inte 10 olika prioriteter. I så fall avbildas de prioriteter som anges i ett program till de prioriteter som finns på motsvarande plattform. På så sätt kan två trådar med olika prioriteter i källkoden ha samma prioritet vid exekveringen. Istället för att fastställa prioriteter till konkreta heltalsvärden, kan motsvarande konstanter i klassen `Thread` användas. Dessa konstanter är `MIN_PRIORITY` (för den lägsta prioriteten), `NORM_PRIORITY` (för den förvalda prioriteten), och `MAX_PRIORITY` (för den högsta prioriteten),

Vissa operativsystem stödjer inte alls olika prioriteter. Därför ska man inte stödja sig på prioriteter vid utformningen av olika program.

```
// Skrivare definierar en tråd, som skriver ut
// ett givet meddelande exakt 4 gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet.
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // skriv ut meddelandet
            System.out.println (meddelande);

            // gör ett uppehåll - i så fall kan även en tråd som
            // har en mindre prioritet få möjlighet att exekvera
            // (om alla trådar med högre prioriteter är
            // blockerade)
            // try
            // {
            //    Thread.sleep (500);
            //
            // }
            // catch (InterruptedException e)
            // {}
        }
    }
}

// EnTradsPrioritet skapar flera trådar, tilldelar dem
// olika prioriteter, och startar trådarna
```

```
class EnTradsPrioritet
{
    public static void main (String[] args)
    {
        System.out.println ("EN TRÅDS PRIORITET");
        System.out.println ();

        // skapa flera trådar
        Thread    t1 = new Thread (new Skrivare ("gryningen"));
        Thread    t2 = new Thread (new Skrivare ("middag"));
        Thread    t3 = new Thread (new Skrivare ("skymningen"));

        // tilldela prioriteter till trådarna
        t1.setPriority (Thread.MAX_PRIORITY);
        t2.setPriority (Thread.NORM_PRIORITY);
        t3.setPriority (Thread.MIN_PRIORITY);
        // metoden setPriority tar emot ett argument
        // mellan 1 och 10.
        // Prioriteten kan anges även via motsvarande
        // konstant från klassen Thread.

        // starta trådarna
        t1.start ();
        t2.start ();
        t3.start ();
    }
}
```

Programmets utmatning på en plattform

```
EN TRÅDS PRIORITET

gryningen
gryningen
gryningen
gryningen
middag
middag
middag
middag
skymningen
skymningen
skymningen
skymningen
```

Programmets utmatning vid en exekvering om ett uppehåll görs i `run`-metoden

```
EN TRÅDS PRIORITET

gryningen
middag
```

```
skymningen
gryningen
middag
skymningen
gryningen
middag
skymningen
gryningen
middag
skymningen
```

Kontrollera en tråds aktivitet

Avbryta en tråd

AvbrytaEnTrad.java

Ett program som illustrerar hur en tråd kan avbrytas på ett kontrollerat sätt

När en tråd stoppas utifrån (från en annan tråd), kan oönskade konsekvenser uppstå. Tråden kan vara mitt i något viktigt jobb när den avbryts. Det kan hända att tråden påbörjat ändringen av ett objekt, och att tråden avbryts mitt i ändringen (objektet lämnas i ett inkonsistent tillstånd).

En tråd kan avbrytas på ett kontrollerat sätt med metoden `interrupt` i klassen `Thread`.

Om metoden `interrupt` anropas i samband med en tråd, sätts en flagga (en boolesk variabel som varje tråd har) i tråden till `true`. Statusen för denna flagga kan kontrolleras i den exekverande tråden med den statiska metoden `interrupted` i klassen `Thread` (metoden returnerar flaggans värde, och återställer den till `false`). Om flaggan är satt, kan den exekverande tråden bestämma sig för att avsluta exekveringen. Men tråden kan själv välja när exakt den ska avluta sin exekverering. På så sätt går det att undvika att en tråd avslutas mitt i någon viktigt uppgift.

Om en tråd är blockerad i metoden `sleep` när en annan tråd skickar `interrupt`-signalen till den, avslutas metoden `sleep` genom att ett undantag av typen `InterruptedException` kastas (istället för att motsvarande avbrottsflagga sätts i tråden). Därför behövs en lämplig reaktion även i detta fall (i `catch`-blocket för detta undantag). Man ska då antingen avsluta exekveringen, eller på något sätt signalera (till exempel genom att sätta en boolesk variabel) att `interrupt`-signalen kommit, så att det blir möjligt att på något annat ställe i `run`-metoden fatta beslut om att avsluta exekveringen.

```
// Raknare definierar en tråd, som simulerar räkningen av enheter
// av en viss typ
class Raknare implements Runnable
{
    // antalet enheter
    private int    antal = 0;

    public void run ()
    {
```

```
        // en hjälpvariabel
        int    v;

        // räkna antalet enheter
        while (true)
        {
            // vänta på enheten
            try
            {
                v = (int) (4 * Math.random () + 1);
                Thread.sleep (v * 500);
            }
            catch (InterruptedException e)
            {
                // Kontrolltråden begär att aktiviteten avslutas.
                // Aktiviteten kan avslutas här, eftersom det inte
                // orsakar några negativa effekter.
                System.out.println ();
                // (vissa nödvändiga operationer utförs innan
                // exekveringen avslutas - en ny rad påbörjas)
                break;
            }

            // en enhet har kommit
            antal++;

            // det skulle inte vara bra att avbryta tråden här
            // (om en enhet har kommit, ska detta rapporteras)

            // rapportera ändringen
            System.out.print (antal + " ");

            // nu går det att kontrollera om kontrolltråden begär
            // att aktiviteten avslutas, och i så fall avsluta den
            // (tråden kan avbrytas på den här platsen
            // - inga negativa effekter kan uppstå)
            if (Thread.interrupted ())
            {
                System.out.println ();
                // (vissa nödvändiga operationer utförs innan
                // exekveringen avslutas - en ny rad påbörjas)
                break;
            }
        }
    }
}

// Kontrollor definierar en tråd som skapar en räknartråd, och
// håller denna räknare aktiv medan en aktivitet pågår
class Kontrollor implements Runnable
{
```

```
    // en räknartråd
    private Thread    raknare;

    public void run ()
    {
        // skapa räknaren
        raknare = new Thread (new Raknare ());

        // vänta tills aktiviteten startar
        try
        {
            int    v = (int) (4 * Math.random () + 1);
            Thread.sleep (v * 1000);
        }
        catch (InterruptedException e)
        {}

        // aktiviteten är påbörjad, starta räknaren
        raknare.start ();

        // aktivitet pågår, räknaren räknar
        try
        {
            int    p = (int) (20 * Math.random () + 1);
            Thread.sleep (p * 1000);
        }
        catch (InterruptedException e)
        {}

        // aktiviteten avslutats, avbryt räknaren
        raknare.interrupt ();
    }
}

// AvbrytaEnTrad skapar och startar en kontrolltråd
class AvbrytaEnTrad
{
    public static void main (String[] args)
    {
        System.out.println ("AVBRYTA EN TRÅD");
        System.out.println ();

        // skapa och starta en kontrolltråd (som i sin tur
        // skapar och startar en räknare, och avbryter den sedan)
        Thread    kontrollor = new Thread (new Kontrollor ());
        kontrollor.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
AVBRYTA EN TRÅD

1 2 3 4 5 6 7 8 9 10
```

Programmets utmatning vid en annan exekvering

```
AVBRYTA EN TRÅD

1 2 3 4
```

Signaler till en tråd

SignalTillEnTrad.java

Ett program som illustrerar hur en signal skickas från en tråd till en annan tråd

Metoden `interrupt` i klassen `Thread` kan användas för att avbryta en tråd från en annan tråd. Metoden kan även användas för att på ett annat sätt kontrollera en tråd. Med metoden `interrupt` kan en signal skickas från en tråd till en annan tråd. Mottagartråden kan sedan tolka signalen på olika sätt.

Tråden som anropar metoden `interrupt` kan ange ett värde för en statusvariabel i mottagartråden före anropet. Mottagartråden kan analysera statusvariabeln när den registrerar att `interrupt`-signalen kommit, och beroende av variabelns värde fortsätta med en av flera möjliga operationer (den kan även avslutas om statusvariabeln indikerar det).

```
// Raknare definierar en tråd som simulerar räkningen av enheter
// av en viss typ
class Raknare implements Runnable
{
    // antalet enheter
    private int    antal = 0;

    // en statusvariabel
    private int    status = 0;

    public void run ()
    {
        // en hjälpvariabel
        int    v;

        // räkna antalet enheter
        while (true)
```

```
        {
            // om signal från kontrolltråden kommit
            if (Thread.interrupted ())
            {
                System.out.println ();

                if (status == -1)
                    break;              // avsluta
                else
                    antal = status;     // återställ räknaren
            }

            // vänta på enheten
            try
            {
                v = (int) (4 * Math.random () + 1);
                Thread.sleep (v * 500);
            }
            catch (InterruptedException e)
            // kontrolltråden skickar en interrupt-signal.
            {
                // sätt interrupt-flaggan i den exekverande tråden
                Thread.currentThread ().interrupt ();

                // hoppa över resten av loopen,
                // och gå och analysera signalen.
                continue;
            }

            // en enhet har kommit
            antal++;

            // rapportera ändringen
            System.out.print (antal + " ");
        }
    }

    // setStatus tilldelar statusvariabeln ett givet heltalsvärde
    public void setStatus (int status)
    {
        this.status = status;
    }
}

// Kontrollor definierar en tråd som skapar en räknartråd,
// återställer räknaren vid vissa tillfällen, och avbryter
// räknartråden när den inte vidare behövs
class Kontrollor implements Runnable
{
    // en räknare
    private Raknare    raknare;
```

```
    // en räknartråd
    private Thread    raknartrad;

    public void run ()
    {
        // skapa och starta räknartråden
        raknare = new Raknare ();
        raknartrad = new Thread (raknare);
        raknartrad.start ();

        while (true)
        {
            // aktivitet pågår, räknaren räknar
            try
            {
                int    p = (int) (20 * Math.random () + 1);
                Thread.sleep (p * 1000);
            }
            catch (InterruptedException e)
            {
                // sätt interrupt-flaggan i den exekverande tråden
                // (hanteringen kommer direkt efter det här
                // catch-blocket)
                Thread.currentThread ().interrupt ();
            }

            if (Thread.interrupted ())
            {
                // avbryt räknartråden
                raknare.setStatus (-1);
                raknartrad.interrupt ();

                // avbryt exekveringen
                break;
            }

            // en cykel har avslutats, återställ räknaren
            // och fortsätt på samma sätt
            raknare.setStatus (0);
            raknartrad.interrupt ();
        }
    }
}

// SignalTillEnTrad skapar och startar en kontrolltråd,
// och avbryter denna efter en viss tid
class SignalTillEnTrad
{
    public static void main (String[] args)
    {
```

```
        System.out.println ("SIGNAL TILL EN TRÅD");
        System.out.println ();

        // skapa och starta en kontrolltråd
        Thread    kontrollor = new Thread (new Kontrollor ());
        kontrollor.start ();

        // vänta, aktiviteten pågår
        try
        {
            int    n = (int) (20 * Math.random () + 11);
            Thread.sleep (n * 1000);
        }
        catch (InterruptedException e)
        {}

        // avsluta kontrolltråden
        kontrollor.interrupt ();
    }
}
```

Programmets utmatning vid en exekvering

```
SIGNAL TILL EN TRÅD

1 2 3 4 5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 2
```

Programmets utmatning vid en annan exekvering

```
SIGNAL TILL EN TRÅD

1
1 2 3 4
1 2 3 4 5

1 2 3
```

Signal till en grupp trådar

SignalTillEnGruppTradar.java

Ett program som skickar en signal till en grupp trådar

För att lättare manipulera ett antal trådar, kan trådarna grupperas i olika grupper.

En `interrupt`-signal kan skickas till en grupp trådar. I så fall vidarebefordras signalen till varje tråd i gruppen.

```
// Skrivare definierar en tråd, som skriver ut
// ett givet meddelande precis 4 gånger
class Skrivare implements Runnable
{
    // ett meddelande
    private String    meddelande;

    // Skrivare initierar meddelandet
    public Skrivare (String meddelande)
    {
        this.meddelande = meddelande;
    }

    // skriv ut ett meddelande ett antal gånger
    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            System.out.println (meddelande);

            // avbryt exekveringen om en interrupt-signal kommer
            if (Thread.interrupted ())
                break;
        }
    }
}

// SignalTillEnGruppTradar skapar en grupp trådar
// och en tråd utanför gruppen. Trådarna startas,
// och trådar i gruppen avbryts sedan.
class SignalTillEnGruppTradar
{
    public static void main (String[] args)
    {
        System.out.println ("SIGN AL TILL EN GRUPP TRÅDAR");
        System.out.println ();

        // skapa en grupp trådar
        ThreadGroup    tg = new ThreadGroup ("riktningar");
        Thread    t1 = new Thread (tg, new Skrivare ("öst"));
        Thread    t2 = new Thread (tg, new Skrivare ("väst"));
        Thread    t3 = new Thread (tg, new Skrivare ("syd"));
        Thread    t4 = new Thread (tg, new Skrivare ("nord"));

        // starta trådarna i gruppen
        t1.start ();
        t2.start ();
        t3.start ();
```

```
        t4.start ();

        // skapa en tråd och starta den
        Thread    t = new Thread (new Skrivare ("0 1 2 3 4"));
        t.start ();

        // vänta ett ögonblick
        try
        {
            Thread.sleep (1);
        }
        catch (InterruptedException e)
        {}

        // avbryt alla trådar i gruppen
        // (skicka en interrupt-signal till alla trådar i gruppen)
        tg.interrupt ();
        // tråden t avbryts inte - den fortsätter med sin
        // aktivitet (om den aktiviteten inte ännu avslutats)
    }
}
```

Programmets utmatning vid en exekvering

```
SIGNAL TILL EN GRUPP TRÅDAR

väst
väst
syd
nord
0 1 2 3 4
öst
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
```

Synkroniserad användning av ett objekt

Osynkroniserad användning av ett objekt

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` representerar ett ensiffrigt heltal.

Klassen har en förvald konstruktor, en metod som returnerar ett heltals strängrepresentation och en metod som sätter ett heltals numeriska värde och namn utifrån ett givet heltalsvärde.

Klassen `EnsiffrigtHeltal` är inte säker för användning i en miljö med flera trådar.

```
// EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        String    s = "";

        s = "[" + varde + ", " + namn + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
    // ett givet heltalsvärde
    public void set (int v)
```

```
    {
        // sätt heltalets numeriska värde
        this.varde = v;

        // bestäm heltalets namn
        String    n = "";
        int    absv = Math.abs (v);
        switch (absv)
        {
        case 0:
            n = "noll";
            break;
        case 1:
            n = "ett";
            break;
        case 2:
            n = "två";
            break;
        case 3:
            n = "tre";
            break;
        case 4:
            n = "fyra";
            break;
        case 5:
            n = "fem";
            break;
        case 6:
            n = "sex";
            break;
        case 7:
            n = "sju";
            break;
        case 8:
            n = "åtta";
            break;
        case 9:
            n = "nio";
            break;
        default:
            n = "inte ensiffrigt heltal";
            break;
       }

       if (v < 0)
           n = "minus " + n;

       // sätt heltalets namn
       this.namn = n;
    }
}
```

EnsiffrigaHeltal.java

Ett program som visar alla ensiffriga heltal

Det här programmet definierar en tråd (`main`-tråden), som skapar och använder ett ensiffrigt heltal. Tråden ändrar heltalet flera gånger och visar det efter varje ändring.

```
class EnsiffrigaHeltal
{
    public static void main (String[] args)
    {
        System.out.println ("ENSIFFRIGA HELTAL");
        System.out.println ();

        // skapa ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        for (int i = -9; i <= 9; i++)
        {
            // ändra heltalet
            n.set (i);

            // visa heltalet
            System.out.println (n);
        }
    }
}
```

Programmets utmatning

```
ENSIFFRIGA HELTAL

[-9, minus nio]
[-8, minus åtta]
[-7, minus sju]
[-6, minus sex]
[-5, minus fem]
[-4, minus fyra]
[-3, minus tre]
[-2, minus två]
[-1, minus ett]
[0, noll]
[1, ett]
[2, två]
[3, tre]
[4, fyra]
[5, fem]
[6, sex]
[7, sju]
```

```
[8, åtta]
[9, nio]
```

OsynkroniseradeAndringarAvEttObjekt.java

Ett program som illustrerar de problem som kan uppstå när flera trådar samtidigt ändrar ett och samma objekt

När flera trådar samtidigt ändrar ett och samma objekt, kan objektet hamna i ett inkonsistent tillstånd.

En tråd kan delvis ändra ett objekt. Samtidigt kan en annan tråd få tillfälle att exekvera. Den tråden kan ändra objektet på ett annat sätt. När den första tråden igen får tillfälle att exekvera, fortsätter den med den ändring som den redan påbörjat. Som resultat av dessa successiva osynkroniserade ändringar, kan objektet hamna i ett otillåtet tillstånd (den första tråden vill ändra objektet på ett sätt, den andra tråden vill ändra samma objekt på ett annat sätt, men objektet blir ändrat delvis av den första tråden och delvis av den andra tråden - dessa ändringar kan vara motsägelsefulla).

```
// Anvandare1 definierar en tråd som använder ett
// ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan 0 och 9
            int    v = (int) (10 * Math.random ());

            // ändra heltalet
            heltal.set (v);

            // visa heltalet
            String    s = heltal.toString ();
            System.out.println (s);
        }
    }
```

```
}

// Anvandare2 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan -9 och -1
            int    v = (int) (-9 * Math.random () - 1);

            // ändra heltalet
            heltal.set (v);

            // visa heltalet
            String    s = heltal.toString ();
            System.out.println (s);
        }
    }
}

// OsynkroniseradeAndringarAvEttObjekt skapar ett ensiffrigt
// heltal, och två trådar som använder detta heltal.
class OsynkroniseradeAndringarAvEttObjekt
{
    public static void main (String[] args)
    {
        System.out.println (
             "OSYNKRONISERADE ÄNDRINGAR AV ETT OBJEKT");
        System.out.println ();

        // ett ensiffrigt heltal (ett objekt, en resurs)
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));

        // starta trådarna
        t1.start ();
        t2.start ();
        // eftersom två trådar samtidigt ändrar
```

```
        // ett och samma objekt på ett osynkroniserat sätt,
        // kan objektet hamna i ett otillåtet tillsånd
    }
}
```

Programmets utmatning vid en exekvering

```
OSYNKRONISERADE ÄNDRINGAR AV ETT OBJEKT

[6, sex]
[9, nio]
[0, noll]
[7, sju]
[0, noll]
[-4, minus fyra]
[3, tre]
[-9, minus nio]
[9, nio]
[-8, minus åtta]
[7, sju]
[-3, minus tre]
[0, noll]
[-2, minus två]
[4, fyra]
[-8, minus åtta]
[-1, minus ett]
[-2, minus två]
[-3, minus tre]
[-4, minus fyra]
```

Programmets utmatning vid en annan exekvering

```
OSYNKRONISERADE ÄNDRINGAR AV ETT OBJEKT

[-9, sex]
[1, ett]
[9, nio]
[2, två]
[5, fem]
[5, minus nio]
[5, fem]
[-2, minus två]
[9, nio]
[-1, minus ett]
[2, två]
[-1, minus ett]
[8, åtta]
[-3, minus tre]
[2, två]
[-6, minus sex]
[-5, minus fem]
[-1, minus ett]
```

```
[-4, minus fyra]
[-9, minus nio]
```

Programmets utmatning vid en tredje exekvering

```
OSYNKRONISERADE ÄNDRINGAR AV ETT OBJEKT

[-1, minus ett]
[4, fyra]
[8, åtta]
[1, ett]
[1, minus ett]
[9, nio]
[-7, minus sju]
[2, två]
[-8, minus åtta]
[3, tre]
[-5, minus fem]
[3, tre]
[-1, minus ett]
[7, sju]
[-7, minus sju]
[6, sex]
[-7, minus sju]
[7, sju]
[-9, minus nio]
[-3, minus tre]
```

OsynkroniseradeAndringarOchAvlasningar.java

Ett program som illustrerar de problem som kan uppstå när ett objekt ändras och avläses på ett osynkroniserat sätt

En tråd (eller flera trådar) kan avläsa ett objekts tillstånd, medan en annan tråd (eller flera andra trådar) ändrar objektet. Som resultat av dessa osynkroniserade aktiviteter, kan motsägelsefull information erhållas vid avläsningen.

```
// Anvandare1 definierar en tråd som ändrar ett ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }
```

```
    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            int    v = (int) (10 * Math.random ());
            heltal.set (v);
        }
    }
}

// Anvandare2 definierar en tråd som avläser ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            String    s = heltal.toString ();
            System.out.println (s);
        }
    }
}

// OsynkroniseradeAndringarOchAvlasningar skapar
// ett ensiffrigt heltal, en tråd som ändrar heltalet,
// och en tråd som avläser detta heltal
class OsynkroniseradeAndringarOchAvlasningar
{
    public static void main (String[] args)
    {
        System.out.println ("OSYNKRONISERADE ÄNDRINGAR OCH"
                            + " AVLÄSNINGAR");
        System.out.println ();

        // ett ensiffrigt heltal (ett objekt, en resurs)
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        // en tråd som ändrar heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));

        // en tråd som avläser heltalet
        Thread    t2 = new Thread (new Anvandare2 (n));
```

```
        // starta trådarna
        t1.start ();
        t2.start ();
        // Den ena tråden ändrar heltalet och den andra tråden
        // avläser det. Dessa två aktiviteter är inte
        // synkroniserade på något sätt. Därför kan motsägelsefull
        // information erhållas vid avläsningen.
    }
}
```

Programmets utmatning vid en exekvering

```
OSYNKRONISERADE ÄNDRINGAR OCH AVLÄSNINGAR

[0, noll]
[0, noll]
[0, noll]
[0, noll]
[0, noll]
[5, fem]
[5, fem]
[5, fem]
[5, fem]
[5, fem]
```

Programmets utmatning vid en annan exekvering

```
OSYNKRONISERADE ÄNDRINGAR OCH AVLÄSNINGAR

[3, ett]
[3, ett]
[3, ett]
[3, ett]
[3, ett]
[0, noll]
[0, noll]
[0, noll]
[0, noll]
[0, noll]
```

Helt synkroniserade objekt

EnsiffrigtHeltal.java

En klass som definierar helt synkroniserade objekt

Objekt av klassen `EnsiffrigtHeltal` är helt synkroniserade objekt, eftersom:

1. Alla metoder i denna klass är synkroniserade

2. Alla metoder lämnar det aktuella objektet i ett konsistent tillstånd när de avslutas

3. Alla metoder är ändliga, och en tråd kan inte hålla ett objekts lås för alltid

4. Klassens variabler är privata, och det går inte att direkt komma åt dessa variabler medan en metod exekveras

5. Klassens konstruktorer initierar objekt av klassen till ett konsistent tillstånd

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public synchronized String toString ()
    // en synkroniserad metod - medan den här metoden exekveras
    // i samband med ett objekt, kan inte
    // någon annan synkroniserad metod i klassen exekveras
    // i samband med samma objekt- först när metoden avslutas
    // kan någon annan synkroniserad metod exekveras
    // i samband med detta objekt
    {
        String    s = "";

        s = "[" + varde + ", " + namn + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
    // ett givet heltalsvärde
    public synchronized void set (int v)
    {
```

```
        // sätt heltalets numeriska värde
        this.varde = v;

        // bestäm heltalets namn
        String    n = "";
        int    absv = Math.abs (v);
        switch (absv)
        {
        case 0:
            n = "noll";
            break;
        case 1:
            n = "ett";
            break;
        case 2:
            n = "två";
            break;
        case 3:
            n = "tre";
            break;
        case 4:
            n = "fyra";
            break;
        case 5:
            n = "fem";
            break;
        case 6:
            n = "sex";
            break;
        case 7:
            n = "sju";
            break;
        case 8:
            n = "åtta";
            break;
        case 9:
            n = "nio";
            break;
        default:
            n = "inte ensiffrigt heltal";
            break;
       }

       if (v < 0)
           n = "minus " + n;

       // sätt heltalets namn
       this.namn = n;
    }
}
```

HeltSynkroniseradeObjekt.java

Ett program som använder ett helt synkroniserat objekt

En metod i en klass kan deklareras som `synchronized` och på så sätt bli en synkroniserad metod.

När en tråd anropar en synkroniserad (`synchronized`) metod i samband med ett objekt, erhåller tråden objektets lås som sin exklusiva egendom. Denna tråd kan avbrytas tillfälligt, men den behåller låset tills den avslutar exekveringen av metoden (antingen normalt, eller genom ett undantag). Om metoden anropar andra metoder (`synchronized` eller ej) i samma klass, kan tråden exekvera även dessa metoder.

När en tråd håller ett objekts lås, kan inte andra trådar få tillfälle att exekvera någon synkroniserad metod (de kan dock få tillfälle att exekvera de metoder som inte är synkroniserade) i samband med objektet. De måste vänta tills låset blir ledigt igen (tills metoden avslutas). När låset blir ledigt, kan en av de väntande trådarna få låset (det går inte att förutsäga vilken tråd det blir), och anropa en synkroniserad metod i samband med objektet.

Genom att deklarera metoder i en klass som `synchronized`, förhindrar man att dessa metoder exekveras samtidigt från flera trådar. En klass vars samtliga metoder är synkroniserade och som uppfyller flera ytterligare krav, definierar helt synkroniserade objekt. Ett helt synkroniserat objekt kan inte hamna i ett inkonsistent tillstånd när flera trådar använder det.

```
// Anvandare1 definierar en tråd som använder ett
// ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan 0 och 9
            int    v = (int) (10 * Math.random ());

            // ändra heltalet
```

```
            heltal.set (v);

            // visa heltalet
            String    s = heltal.toString ();
            System.out.println (s);
        }
    }
}

// Anvandare2 definierar en tråd som använder ett
// ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan -9 och -1
            int    v = (int) (-9 * Math.random () - 1);

            // ändra heltalet
            heltal.set (v);

            // visa heltalet
            String    s = heltal.toString ();
            System.out.println (s);
        }
    }
}

// HeltSynkroniseradeObjekt skapar ett ensiffrigt heltal
// och två trådar som använder detta heltal.
class HeltSynkroniseradeObjekt
{
    public static void main (String[] args)
    {
        System.out.println ("HELT SYNKRONISERADE OBJEKT");
        System.out.println ();

        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
```

```
        Thread    t2 = new Thread (new Anvandare2 (n));

        // starta trådarna
        t1.start ();
        t2.start ();
        // trots att två trådar samtidigt använder
        // ett och samma objekt, kan objektet inte hamna
        // i ett inkonsistent tillstånd
        // (eftersom objekt av typen EnsiffrigtHeltal är helt
        // synkroniserade objekt)
    }
}
```

Programmets utmatning vid en exekvering

```
HELT SYNKRONISERADE OBJEKT

[-7, minus sju]
[-3, minus tre]
[-3, minus tre]
[0, noll]
[-4, minus fyra]
[7, sju]
[-5, minus fem]
[4, fyra]
[-2, minus två]
[0, noll]
[-3, minus tre]
[0, noll]
[-9, minus nio]
[0, noll]
[-3, minus tre]
[9, nio]
[-3, minus tre]
[5, fem]
[7, sju]
[1, ett]
```

Programmets utmatning vid en annan exekvering

```
HELT SYNKRONISERADE OBJEKT

[7, sju]
[4, fyra]
[4, fyra]
[2, två]
[2, två]
[4, fyra]
[4, fyra]
[1, ett]
[1, ett]
[6, sex]
```

```
[6, sex]
[7, sju]
[7, sju]
[2, två]
[2, två]
[9, nio]
[9, nio]
[6, sex]
[6, sex]
[-4, minus fyra]
```

Sammansatta synkroniserade operationer

EnsiffrigtHeltal.java

En klass som definierar helt synkroniserade objekt

Klassen `EnsiffrigtHeltal` definierar helt synkroniserade objekt.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public synchronized String toString ()
    {
        String    s = "";

        s = "[" + varde + ", " + namn + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
    // ett givet heltalsvärde
```

```
    public synchronized void set (int v)
    {
        // sätt heltalets numeriska värde
        this.varde = v;

        // bestäm heltalets namn
        String    n = "";
        int    absv = Math.abs (v);
        switch (absv)
        {
        case 0:
            n = "noll";
            break;
        case 1:
            n = "ett";
            break;
        case 2:
            n = "två";
            break;
        case 3:
            n = "tre";
            break;
        case 4:
            n = "fyra";
            break;
        case 5:
            n = "fem";
            break;
        case 6:
            n = "sex";
            break;
        case 7:
            n = "sju";
            break;
        case 8:
            n = "åtta";
            break;
        case 9:
            n = "nio";
            break;
        default:
            n = "inte ensiffrigt heltal";
            break;
       }

       if (v < 0)
           n = "minus " + n;

       // sätt heltalets namn
       this.namn = n;
    }
}
```

SammansattaSynkroniseradeOperationer.java

Ett program som illustrerar sammansatta synkroniserade operationer

Det går att skapa en klass som definierar helt synkroniserade objekt. Enskilda operationer i klassen är synkroniserade, och det kan inte inträffa att ett objekt av klassen hamnar i ett inkonsistent tillstånd.

I vissa fall vill man kombinera flera operationer (metoder) från en klass, och bilda en sammansatt synkroniserad operation. Alla dessa operationer behöver utföras som en odelbar operation för att ett konsistent resultat ska erhållas. Ett objekts lås ska inte släppas så länge operationerna utförs.

Flera operationer från en klass kan kombineras i en sammansatt synkroniserad operation med ett synkroniserat block. Operationerna placeras i ett block, som deklareras som `synchronized`. Det objekt vars lås ska hållas medan operationerna utförs anges också.

```
// Anvandare1 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // ett synkroniserat block
            synchronized (heltal)
            // heltalets lås ska hållas medan operationer i
            // det här blocket utförs
            {
                // värden mellan 0 och 9
                int    v = (int) (10 * Math.random ());

               // ändra heltalet
                heltal.set (v);

                // visa heltalet
                String    s = heltal.toString ();
                System.out.println (s);
```

```
                // ändringen och utskriften av heltalet utgör
                // en sammansatt synkroniserad operation -
                // det är bara när den exekverande tråden
                // utför både ändringen och utskriften
                // som någon annan tråd kan få heltalets lås
            }
        }
    }
}

// Anvandare2 definierar en tråd som använder ett
// ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            synchronized (heltal)
            {
                // värden mellan -9 och -1
                int    v = (int) (-9 * Math.random () - 1);

                // ändra heltalet
                heltal.set (v);

                // visa heltalet
                String    s = heltal.toString ();
                System.out.println (s);

                // ändringen och utskriften av heltalet utgör
                // en sammansatt synkroniserad operation -
                // det är bara när den exekverande tråden
                // utför både ändringen och utskriften
                // som någon annan tråd kan få heltalets lås
            }
        }
    }
}

// SammansattaSynkroniseradeOperationer skapar ett
// ensiffrigt heltal och två trådar som använder detta heltal.
class SammansattaSynkroniseradeOperationer
```

```
{
    public static void main (String[] args)
    {
        System.out.println (
              "SAMMANSATTA SYNKRONISERADE OPERATIONER");
        System.out.println ();

        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));
        t1.start ();
        t2.start ();
        // ändringen av heltalet och utskriften av heltalet bildar
        // en synkroniserad operation - det kan inte inträffa att
        // en tråd ändrar heltalet, och den andra tråden visar
        // heltalet direkt efter ändringen
    }
}
```

Programmets utmatning vid en exekvering

```
SAMMANSATTA SYNKRONISERADE OPERATIONER

[1, ett]
[-8, minus åtta]
[8, åtta]
[-2, minus två]
[3, tre]
[-8, minus åtta]
[8, åtta]
[-6, minus sex]
[4, fyra]
[-2, minus två]
[1, ett]
[-5, minus fem]
[4, fyra]
[-4, minus fyra]
[3, tre]
[-3, minus tre]
[3, tre]
[-4, minus fyra]
[6, sex]
[-6, minus sex]
```

Programmets utmatning vid en annan exekvering

```
SAMMANSATTA SYNKRONISERADE OPERATIONER

[3, tre]
```

```
[9, nio]
[4, fyra]
[3, tre]
[0, noll]
[-9, minus nio]
[5, fem]
[-4, minus fyra]
[1, ett]
[-4, minus fyra]
[1, ett]
[-3, minus tre]
[5, fem]
[-7, minus sju]
[0, noll]
[-5, minus fem]
[-8, minus åtta]
[-9, minus nio]
[-9, minus nio]
[-6, minus sex]
```

Optimera synkroniseringen

EnsiffrigtHeltal.java

En definitionsklass som optimerar synkroniseringen

Klassen `EnsiffrigtHeltal` optimerar synkroniseringen, så att endast kritiska sektioner synkroniseras (istället för hela metoder).

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
```

```
{
    int       v = 0;
    String    n = "";
    synchronized (this)
    // det här blocket utgör en kritisk sektion
    {
        v = this.varde;
        n = this.namn;
    }

    String    s = "";

    s = "[" + v + ", " + n + "]";

    return s;
}

// set bestämmer heltalets numeriska värde och namn utifrån
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
    case 7:
        n = "sju";
        break;
    case 8:
```

```
            n = "åtta";
            break;
         case 9:
            n = "nio";
            break;
         default:
            n = "inte ensiffrigt heltal";
            break;
        }

        if (v < 0)
            n = "minus " + n;

         synchronized (this)
         // det här blocket utgör en kritisk sektion
         {
            // sätt heltalets numeriska värde
            this.varde = v;

            // sätt heltalets namn
            this.namn = n;
         }
    }
}
```

OptimeraSynkroniseringen.java

Ett program som optimerar synkroniseringen

När en tråd utför en synkroniserad kodsektion, håller tråden ett objekts lås. Under tiden kan ingen annan tråd använda objektets synkroniserade metoder. Det går inte heller att exekvera de block som är synkroniserade på objektet. På så sätt bevaras objektet i ett konsistent tillstånd.

Synkroniseringen har även negativa konsekvenser: den minskar programmets hastighet. Det tar tid att låsa ett objekt och att låsa upp det, och det tar tid att vänta på ett låst objekt. Därför behöver synkroniseringen optimeras. Endast de kodsektioner där ett objekts variabler används ska synkroniseras. Dessa kodsektioner kallas för kritiska sektioner.

```
// Anvandare1 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;
```

```
    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan 0 och 9
            int    v = (int) (10 * Math.random ());

            String   s = "";

            synchronized (heltal)
            // det här blocket utgör en kritisk sektion
            {
                // ändra heltalet
                heltal.set (v);

                // bestäm heltalets strängrepresentation
                s = heltal.toString ();
            }
            // Endast den kod som använder heltalet placeras i ett
            // synkroniserat block. Resten placeras utanför detta
            // block. På så sätt optimeras synkroniseringen.
            // Endast en kritisk sektion synkroniseras.

            System.out.println (s);
        }
    }
}

// Anvandare2 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // värden mellan -9 och -1
            int    v = (int) (-9 * Math.random () - 1);
```

```
            String    s = "";

            synchronized (heltal)
            // det här blocket utgör en kritisk sektion
            {
                // ändra heltalet
                heltal.set (v);

                // bestäm heltalets strängrepresentation
                s = heltal.toString ();
            }
            // Endast koden som använder heltalet placeras i ett
            // synkroniserat block. Resten placeras utanför detta
            // block. På så sätt optimeras synkroniseringen.
            // Endast en kritisk sektion synkroniseras.

            System.out.println (s);
        }
    }
}

// OptimeraSynkroniseringen skapar ett ensiffrigt heltal,
// och två trådar som använder detta heltal.
class OptimeraSynkroniseringen
{
    public static void main (String[] args)
    {
        System.out.println ("OPTIMERA SYNKRONISERINGEN");
        System.out.println ();

        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));
        t1.start ();
        t2.start ();
    }
}
```

Programmets utmatning vid en exekvering

```
OPTIMERA SYNKRONISERINGEN

[-4, minus fyra]
[1, ett]
[-8, minus åtta]
[1, ett]
[-6, minus sex]
[6, sex]
```

```
[-9, minus nio]
[5, fem]
[-9, minus nio]
[3, tre]
[-6, minus sex]
[8, åtta]
[-8, minus åtta]
[4, fyra]
[-5, minus fem]
[2, två]
[-6, minus sex]
[7, sju]
[-6, minus sex]
[8, åtta]
```

Programmets utmatning vid en annan exekvering

```
OPTIMERA SYNKRONISERINGEN

[8, åtta]
[4, fyra]
[8, åtta]
[4, fyra]
[2, två]
[-2, minus två]
[0, noll]
[-3, minus tre]
[8, åtta]
[-5, minus fem]
[8, åtta]
[-9, minus nio]
[2, två]
[-5, minus fem]
[7, sju]
[-9, minus nio]
[-2, minus två]
[-2, minus två]
[-4, minus fyra]
[-3, minus tre]
```

Synkronisering på klientsidan

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` definierar helt synkroniserade objekt. Klassen `EnsiffrigtHeltal` definierar sorterbara objekt (den implementerar gränssnittet `java.lang.Comparable`).

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal implements Comparable<EnsiffrigtHeltal>
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";

        s = "[" + v + ", " + n + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
    // ett givet heltalsvärde
    public void set (int v)
    {
```

```
 // bestäm heltalets namn
 String    n = "";
 int    absv = Math.abs (v);
 switch (absv)
 {
 case 0:
     n = "noll";
     break;
 case 1:
     n = "ett";
     break;
 case 2:
     n = "två";
     break;
 case 3:
     n = "tre";
     break;
 case 4:
     n = "fyra";
     break;
 case 5:
     n = "fem";
     break;
 case 6:
     n = "sex";
     break;
 case 7:
     n = "sju";
     break;
 case 8:
     n = "åtta";
     break;
 case 9:
     n = "nio";
     break;
 default:
     n = "inte ensiffrigt heltal";
     break;
}

if (v < 0)
    n = "minus " + n;

 synchronized (this)
 {
     // sätt heltalets numeriska värde
     this.varde = v;

     // sätt heltalets namn
     this.namn = n;
 }
```

```
    }

    // compareTo jämför det här heltalet med ett givet ensiffrigt
    // heltal, och returnerar ett heltalsvärde.
    // Metoden returnerar 0 om dessa två heltal är lika,
    // -1 om det här heltalet är mindre, och 1 om det här heltalet
    // är större.
    public int compareTo (EnsiffrigtHeltal heltal)
    {
        int    v1 = 0;
        int    v2 = 0;
        synchronized (this)
        {
            v1 = this.varde;

            synchronized (heltal)
            {
                v2 = heltal.varde;
            }
        }

        int    p = 0;
        if (v1 < v2)
            p = -1;
        else if (v1 > v2)
            p = 1;

        return p;
    }
}
```

SynkroniseringPaKlientSidan.java

Ett program som illustrerar synkroniseringen på klientsidan

Om objekt som inte är helt synkroniserade används, kan användningen av dessa objekt behöva synkroniseras på klientsidan (i den klass som använder objekten).

Om flera trådar samtidigt använder en vektor av den inbyggda typen, måste användningen synkroniseras (eftersom det inte finns några inbyggda mekanismer som synkroniserar användningen av en sådan vektor).

```
// Sorterare definierar en tråd som sorterar
// en vektor med ensiffriga heltal.
class Sorterare implements Runnable
{
```

```
    private EnsiffrigtHeltal[]    heltal;

    public Sorterare (EnsiffrigtHeltal[] heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        synchronized (heltal)
        // synkroniseringen på klientsidan (den här klassen är
        // en klientklass, som använder en vektor)
        {
            // sortera heltal
            java.util.Arrays.sort (heltal);
        }
    }
}

// SynkroniseringPaKlientSidan skapar en vektor med
// ensiffriga heltal, och en tråd som sorterar dessa heltal
class SynkroniseringPaKlientSidan
{
    public static void main (String[] args)
    {
        System.out.println ("SYNKRONISERING PÅ KLIENTSIDAN");
        System.out.println ();

        // en vektor med ensiffriga heltal
        EnsiffrigtHeltal[]    v = new EnsiffrigtHeltal[10];
        for (int i = 0; i < 10; i++)
            v[i] = new EnsiffrigtHeltal ();
        for (int i = 0; i < 10; i++)
            v[i].set (9 - i);

        // en tråd som sorterar heltalen
        Thread    t = new Thread (new Sorterare (v));
        t.start ();

        // vänta en stund
        // try
        // {
        //     Thread.sleep (1);
        // }
        // catch (InterruptedException e)
        // {}
        // se vad som händer om kommentarsmarkeringarna tas bort

        // visa vektorn
        synchronized (v)
        {
```

```
            for (int i = 0; i < 10; i++)
                System.out.println (v[i]);
        }

        // Tråden t och main-tråden använder vektorn v
        // samtidigt. Denna användning synkroniseras.
        // Vektorn kan visas före sorteringen eller
        // efter sorteringen. Det går inte att växla
        // mellan sorteringen och utskriften (ta bort synchronized
        // och se vad som händer i så fall).
    }
}
```

Programmets utmatning vid en exekvering

```
SYNKRONISERING PÅ KLIENTSIDAN

[9, nio]
[8, åtta]
[7, sju]
[6, sex]
[5, fem]
[4, fyra]
[3, tre]
[2, två]
[1, ett]
[0, noll]
```

Programmets utmatning vid en exekvering om man inte använder `synchronized`-block

```
SYNKRONISERING PÅ KLIENTSIDAN

[9, nio]
[8, åtta]
[7, sju]
[6, sex]
[5, fem]
[5, fem]
[6, sex]
[7, sju]
[8, åtta]
[9, nio]
```

Synkroniserade behållare

SynkroniseradeBehallare.java

Ett program som illustrerar synkroniserade behållare

Javas standardbibliotek innehåller (i paketet `java.util`) ett antal klasser som representerar osynkroniserade behållare (`ArrayList`, `LinkedList`, `HashMap`, `HashSet` och andra). När dessa behållare används i en flertrådad miljö måste deras användning synkroniseras.

Istället för att synkronisera användningen av olika osynkroniserade behållare, kan en annan strategi användas. Utifrån en osynkroniserad behållare kan motsvarande synkroniserad behållare skapas (alla metoder görs till synkroniserade metoder). För detta ändamål används lämpliga statiska metoder från klassen `java.util.Collections`.

```
import java.util.*;  // List, ArrayList, Collections

// Adder definierar en tråd som placerar ett antal heltal i en
// behållare (med index)
class Adder implements Runnable
{
    // behållaren för heltal
    private List<Integer>    list;

    public Adder (List<Integer> list)
    {
        this.list = list;
    }

    public void run ()
    {
        // lägg till tusen heltal i behållaren
        for (int i = 1; i <= 1000; i++)
            list.add (i);
    }
}

// Remover definierar en tråd som tar bort ett antal heltal
// från en behållare (med index)
class Remover implements Runnable
{
    // behållaren med heltal
    private List<Integer>    list;

    public Remover (List<Integer> list)
```

```
    {
        this.list = list;
    }

    public void run ()
    {
        // ta bort tusen heltal från behållaren
        int i = 0;
        while (i < 1000)
            if (list.size () > 0)
            {
                list.remove (0);
                i++;
            }
    }
}

// SynkroniseradeBehallare skapar och startar två trådar
// som använder en och samma behållare
class SynkroniseradeBehallare
{
    public static void main (String[] args)
    {
        System.out.println ("SYNKRONISERADE BEHÅLLARE");
        System.out.println ();

        // en osynkroniserad vektor för heltal
        List<Integer>    v = new ArrayList<Integer> ();

        // skapa en synkroniserad vektor utifrån en osynkroniserad
        // vektor (alla metoder i den nya vektorn
        // blir synkroniserade)
        // v = Collections.synchronizedList(v);
        // ta bort kommentarsmarkeringen för att använda den
        // synkroniserade vektorn

        // visa vektorn (vektorn är tom)
        System.out.println (v);

        // två trådar som använder vektorn
        Thread    t1 = new Thread (new Adder (v));
        Thread    t2 = new Thread (new Remover (v));
        t1.start ();
        t2.start ();
        // t1 placerar tusen heltal i vektorn, t2 tar bort tusen
        // heltal från vektorn. På slutet ska vektorn vara tom.

        // vänta tills trådarna avslutar sina uppgifter
        try
        {
            t1.join ();
```

```
            t2.join ();
        }
        catch (InterruptedException e)
        {}

        // vektorn efter bearbetningen
        System.out.println (v);

        // Vektorn ska vara tom. Men om den osynkroniserade
        // vektorn används, kan det hända att olika operationer
        // i olika trådar ändrar vektorn på ett olämpligt sätt
        // så att dess innehåll slutligen blir meningslöst.
        // Om den synkroniserade vektorn används, kan inte en tråd
        // använda den förrän den andra tråden avslutar en konkret
        // operation (metoderna add, size och remove blir i så
        // fall synkroniserade).
        // Därför blir vektorn tom på slutet.
    }
}
```

Programmets utmatning vid en exekvering

```
SYNKRONISERADE BEHÅLLARE

[]
[]
```

Programmets utmatning vid en annan exekvering

```
SYNKRONISERADE BEHÅLLARE

[]
[996, 996, 998, 999]
```

Programmets utmatning om kommentarmarkeringen tas bort

```
SYNKRONISERADE BEHÅLLARE

[]
[]
```

Synkroniserad användning av flera objekt

Ett dödläge

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` representerar ett ensiffrigt heltal.

Klassen `EnsiffrigtHeltal` är säker för användning i samband med flera trådar (kritiska kodsektioner är `synchronized`).

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";
        s = "[" + v + ", " + n + "]";

        return s;
    }
```

```
// set bestämmer heltalets numeriska värde och namn utifrån
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
    case 7:
        n = "sju";
        break;
    case 8:
        n = "åtta";
        break;
    case 9:
        n = "nio";
        break;
    default:
        n = "inte ensiffrigt heltal";
        break;
   }

   if (v < 0)
       n = "minus " + n;

    synchronized (this)
    {
        // sätt heltalets numeriska värde
        this.varde = v;
```

```
            // sätt heltalets namn
            this.namn = n;
        }
    }
}
```

Dodläge.java

Ett program som illustrerar ett dödläge

Om flera trådar i ett program samtidigt använder flera gemensamma objekt, kan programmet hamna i ett dödläge. En tråd kan låsa ett objekt och vänta på ett annat objekts lås. En annan tråd kan äga det andra objektets lås och vänta på något tredje objekts lås. En tredje tråd kan äga det tredje objektets lås, och vänta på det första objektets lås. På så sätt bildas en cirkulär kedja (en ring) av trådar som väntar på varandra. Ingen av dessa trådar kan fortsätta exekvera. Eventuella andra trådar i programmet kan fortsätta exekvera, men det kan hända att även dessa trådar snabbt låser sig.

Ett dödläge kan förhindras genom att de gemensamma objekten tilldelas i en förbestämd ordning. Dessa objekt numreras och tilldelas till en tråd i en stigande ordning. På så sätt förhindras skapandet av en cirkulär kedja av väntande trådar (en tråd som har ett objekt med ett nummer kan inte gå tillbaka och kräva ett objekt med ett mindre nummer - kedjan kan inte stängas).

Det finns även ett annat sätt att förhindra skapandet av en cirkulär kedja av väntande trådar. Den tråd som håller ett objekt och försöker få ett redan upptaget objekt, ska släppa det objekt som den äger. Tråden ska invänta en mer gynnsam situation för exekvering, men under tiden ska den inte hindra andra trådar från att exekvera.

```
// Anvandare1 definierar en tråd som använder
// två ensiffriga heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal1;
    private EnsiffrigtHeltal    heltal2;

    public Anvandare1 (EnsiffrigtHeltal heltal1,
                       EnsiffrigtHeltal heltal2)
    {
        this.heltal1 = heltal1;
```

```
        this.heltal2 = heltal2;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
            // två slumpmässiga värden mellan 0 och 9
            int    v1 = (int) (10 * Math.random ());
            int    v2 = (int) (10 * Math.random ());

            synchronized (heltal1)
            // nu låser tråden heltal1
            {
                // ändra första heltalet
                heltal1.set (v1);

                synchronized (heltal2)
                // nu låser tråden även heltal2
                {
                    // ändra andra heltalet
                    heltal2.set (v2);

                    // visa heltalen
                    System.out.println (heltal1 + " " + heltal2);
                }
                // heltal2 låses upp
            }
            // heltal1 låses upp
        }
    }
}

// Anvandare2 definierar en tråd som använder
// två ensiffriga heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal1;
    private EnsiffrigtHeltal    heltal2;

    public Anvandare2 (EnsiffrigtHeltal heltal1,
                       EnsiffrigtHeltal heltal2)
    {
        this.heltal1 = heltal1;
        this.heltal2 = heltal2;
    }

    public void run ()
    {
        for (int i = 0; i < 10; i++)
        {
```

```
            // två slumpmässiga värden mellan -9 och -1
            int    v1 = (int) (-9 * Math.random () - 1);
            int    v2 = (int) (-9 * Math.random () - 1);

            synchronized (heltal1)
            {
                // ändra första heltalet
                heltal1.set (v1);

                synchronized (heltal2)
                {
                    // ändra andra heltalet
                    heltal2.set (v2);

                    // visa heltalen
                    System.out.println (heltal1 + " " + heltal2);
                }
            }
        }
    }
}

// Dodlage skapar två ensiffriga heltal och två trådar som
// samtidigt använder dessa heltal.
class Dodlage
{
    public static void main (String[] args)
    {
        System.out.println ("DÖDLÄGE");
        System.out.println ();

        // två ensiffriga heltal
        EnsiffrigtHeltal    n1 = new EnsiffrigtHeltal ();
        EnsiffrigtHeltal    n2 = new EnsiffrigtHeltal ();

        // två trådar som använder dessa heltal
        Thread    t1 = new Thread (new Anvandare1 (n1, n2));
        Thread    t2 = new Thread (new Anvandare2 (n2, n1));
        // t1 har n1 och n2, och t2 har n2 och n1
        t1.start ();
        t2.start ();
        // En tråd kan låsa n1, och den andra tråden kan låsa n2.
        // Den tråd som håller låset för n1 kan vänta på låset
        // för n2, och tråden som håller låset för n2 kan
        // vänta på låset för n1. Ingen tråd kan fortsätta
        // exekvera, och programmet hamnar i ett dödläge.

        // Ett dödläge kan inte uppstå om trådarna låser
        // heltalen i samma ordning, till exempel
        // först n1 och sedan n2. Det kan åstadkommas
        // genom att den andra tråden definieras med en annan
```

```
        // ordning på argumenten. Så här ska det göras:
        // Thread    t2 = new Thread (new Anvandare2 (n1, n2));
    }
}
```

Programmets utmatning vid en exekvering

```
DÖDLÄGE

[0, noll] [9, nio]
[5, fem] [9, nio]
[8, åtta] [6, sex]
[9, nio] [7, sju]
[1, ett] [0, noll]
```

Programmets utmatning vid en annan exekvering

```
DÖDLÄGE
```

Programmets utmatning vid en tredje exekvering

```
DÖDLÄGE

[3, tre] [3, tre]
[0, noll] [6, sex]
[4, fyra] [3, tre]
[7, sju] [5, fem]
[6, sex] [8, åtta]
[6, sex] [1, ett]
[6, sex] [6, sex]
[9, nio] [5, fem]
[2, två] [8, åtta]
[6, sex] [8, åtta]
[-7, minus sju] [-9, minus nio]
[-2, minus två] [-3, minus tre]
[-3, minus tre] [-8, minus åtta]
[-5, minus fem] [-6, minus sex]
[-1, minus ett] [-1, minus ett]
[-2, minus två] [-2, minus två]
[-8, minus åtta] [-4, minus fyra]
[-8, minus åtta] [-2, minus två]
[-9, minus nio] [-6, minus sex]
[-6, minus sex] [-5, minus fem]
```

Synkronisering av tillståndsberoende operationer

Tillståndsberoende operationer

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` har en metod (operation) som bara kan utföras om det aktuella objektet är i ett lämpligt tillstånd.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";

        s = "[" + v + ", " + n + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
```

```
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
    case 7:
        n = "sju";
        break;
    case 8:
        n = "åtta";
        break;
    case 9:
        n = "nio";
        break;
    default:
        n = "inte ensiffrigt heltal";
        break;
   }

   if (v < 0)
       n = "minus " + n;

    synchronized (this)
    {
        // sätt heltalets numeriska värde
        this.varde = v;
```

```
            // sätt heltalets namn
            this.namn = n;
        }
    }

    // add lägger till ett givet heltalsvärde till heltalet
    // (ett objekt kan hamna i ett otillåtet tillstånd som
    // ett resultat av denna operation)
    public synchronized void add (int p)
    {
        int    v = this.varde + p;

        this.set (v);
    }
}
```

TillstandsberoendeOperationer.java

Ett program som illustrerar tillståndsberoende operationer

Det kan inträffa att vissa operationer i en klass bara kan utföras om det aktuella objektet är i ett lämpligt tillstånd. I så fall måste det aktuella objektets tillstånd undersökas innan dessa operationer utförs. Om detta inte görs, kan objektet hamna i ett otillåtet tillstånd.

```
// Anvandare1 definierar en tråd som använder ett
// ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // 0, 1 eller 2
            int    p = (int) (3 * Math.random ());

            synchronized (heltal)
            {
                // öka heltalet
                heltal.add (p);
```

```
                // en tillståndsberoende operation
                // (det går till exempel inte att lägga 2 till 9,
                // eftersom heltal måste förbli
                // ett ensiffrigt heltal)

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// Anvandare2 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // -1 eller -2
            int    p = - (int) (2 * Math.random () + 1);

            synchronized (heltal)
            {
                // minska heltalet
                heltal.add (p);
                // en tillståndsberoende operation

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// TillstandsberoendeOperationer skapar ett ensiffrigt heltal,
// och två trådar som använder detta heltal.
class TillstandsberoendeOperationer
{
    public static void main (String[] args)
    {
        System.out.println ("TILLSTÅNDSBEROENDE OPERATIONER");
        System.out.println ();
```

```
        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();
        n.set (5);

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));

        // starta trådarna
        t1.start ();
        t2.start ();
        // Trådarna utför även operationen add,
        // som bara kan utföras om objektet n
        // är i ett lämpligt tillstånd.
        // Eftersom det inte finns någon kontroll
        // av detta tillstånd, kan objektet n hamna
        // i ett otillåtet tillstånd.
    }
}
```

Programmets utmatning vid en exekvering

```
TILLSTÅNDSBEROENDE OPERATIONER

[6, sex]
[5, fem]
[7, sju]
[5, fem]
[7, sju]
[5, fem]
[6, sex]
[5, fem]
```

Programmets utmatning vid en andra exekvering

```
TILLSTÅNDSBEROENDE OPERATIONER

[5, fem]
[7, sju]
[8, åtta]
[10, inte ensiffrigt heltal]
[9, nio]
[7, sju]
[5, fem]
[3, tre]
```

Programmets utmatning vid en tredje exekvering

```
TILLSTÅNDSBEROENDE OPERATIONER

[7, sju]
```

```
[9, nio]
[11, inte ensiffrigt heltal]
[9, nio]
[10, inte ensiffrigt heltal]
[8, åtta]
[7, sju]
[5, fem]
```

Synkronisera tillståndsberoende operationer

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` har en operation (en metod) som bara kan utföras om det aktuella objektet är i ett lämpligt tillstånd.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";
```

```
        s = "[" + v + ", " + n + "]";

        return s;
    }

    // set bestämmer heltalets numeriska värde och namn utifrån
    // ett givet heltalsvärde
    public void set (int v)
    {
        // bestäm heltalets namn
        String    n = "";
        int    absv = Math.abs (v);
        switch (absv)
        {
        case 0:
            n = "noll";
            break;
        case 1:
            n = "ett";
            break;
        case 2:
            n = "två";
            break;
        case 3:
            n = "tre";
            break;
        case 4:
            n = "fyra";
            break;
        case 5:
            n = "fem";
            break;
        case 6:
            n = "sex";
            break;
        case 7:
            n = "sju";
            break;
        case 8:
            n = "åtta";
            break;
        case 9:
            n = "nio";
            break;
        default:
            n = "inte ensiffrigt heltal";
            break;
       }

       if (v < 0)
```

```
            n = "minus " + n;

        synchronized (this)
        {
            // sätt heltalets numeriska värde
            this.varde = v;

            // sätt heltalets namn
            this.namn = n;
        }
    }

    // add lägger till ett givet heltalsvärde till heltalet
    public synchronized void add (int p)
                                throws InterruptedException
    {
        // vänta med ändringen, om ändringen skulle orsaka
        // ett otillåtet tillstånd
        while (this.varde + p < - 9  ||  this.varde + p > 9)
            this.wait ();

        // lägg till heltalsvärdet
        this.set (this.varde + p);

        // informera alla trådar som väntar (i det här objektets
        // wait-mängd) om att objektet ändrats
        this.notifyAll ();
    }
}
```

SynkroniseraTillstandsberoendeOperationer.java

Ett program som illustrerar användning av synkroniserade tillståndsberoende operationer

Det kan hända att en operation i en klass bara kan utföras om det aktuella objektet är i ett lämpligt tillstånd. I så fall kontrolleras objektets tillstånd innan operationen utförs. Om objektet är i ett lämpligt tillstånd, utförs operationen. Eftersom operationen nödvändigtvis ändrar det aktuella objektet, håller den exekverande tråden objektets lås medan den utför denna operation.

Om det aktuella objektet inte är i ett lämpligt tillstånd, släpper den exekverande tråden objektets lås, och blockerar sig (med anropet till metoden `wait` i klassen `Object`). En annan tråd kan få möjlighet att ändra objektet. Denna ändring kan eventuellt skapa en situation som gör det möjligt för

den blockerade tråden att fortsätta med sin operation. Därför informerar den exekverande tråden (med anropet till metoden `notifyAll` i klassen `Object`) alla de trådar som väntar på att fortsätta sina operationer i samband med objektet (dessa trådar bildar en `wait`-mängd hos objektet) om ändringen. Alla dessa trådar aktiveras igen (om `notify` används istället för `notifyAll` aktiveras bara en slumpmässigt vald tråd), och de kan få möjlighet att exekvera. Den första av trådarna som får möjlighet att exekvera låser det aktuella objektet (om det inte är redan låst av en annan tråd), och fortsätter från den punkten där den avbröts. Tråden fortsätter med sin operation om de nödvändiga villkoren är uppfyllda, annars blockeras den igen (med anropet till metoden `wait` - detta anrop måste därför vara placerat i en loop).

```
// Anvandare1 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // 0, 1 eller 2
            int    p = (int) (3 * Math.random ());

            synchronized (heltal)
            {
                // öka heltalet
                try
                {
                    heltal.add (p);
                }
                catch (InterruptedException e)
                {}

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// Anvandare2 definierar en tråd som använder
```

```
// ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // -1 eller -2
            int    p = - (int) (2 * Math.random () + 1);

            synchronized (heltal)
            {
                // minska heltalet
                try
                {
                    heltal.add (p);
                }
                catch (InterruptedException e)
                {}

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// SynkroniseraTillstandsberoendeOperationer skapar ett ensiffrigt
// heltal och två trådar som använder detta heltal.
class SynkroniseraTillstandsberoendeOperationer
{
    public static void main (String[] args)
    {
        System.out.println ("SYNKRONISERA TILLSTÅNDSBEROENDE"
                            + " OPERATIONER");
        System.out.println ();

        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();
        n.set (5);
        // Prova med att sätta heltalets värde till 7,
        // istället för 5 (kör programmet flera gånger).
        // Prova också med -7 (nära nedre gränsen).
```

```
        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));

        // starta trådarna
        t1.start ();
        t2.start ();
        // Trådarna utför även operationen add,
        // som bara kan utföras om det aktuella objektet
        // är i ett lämpligt tillstånd.
        // Eftersom tillståndet kontrolleras (inuti metoden add
        // i klassen EnsiffrigtHeltal), kan det aktuella objektet
        // inte hamna i ett otillåtet tillstånd.
    }
}
```

Programmets utmatning vid en exekvering

```
SYNKRONISERA TILLSTÅNDSBEROENDE OPERATIONER

[5, fem]
[4, fyra]
[5, fem]
[4, fyra]
[6, sex]
[5, fem]
[6, sex]
[4, fyra]
```

Programmets utmatning vid en andra exekvering

```
SYNKRONISERA TILLSTÅNDSBEROENDE OPERATIONER

[6, sex]
[8, åtta]
[9, nio]
[8, åtta]
[6, sex]
[8, åtta]
[7, sju]
[6, sex]
```

Programmets utmatning vid en tredje exekvering

```
SYNKRONISERA TILLSTÅNDSBEROENDE OPERATIONER

[7, sju]
[9, nio]
[8, åtta]
[9, nio]
[7, sju]
[9, nio]
```

```
[8, åtta]
[7, sju]
```

Synkronisering på klientsidan

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` har en metod (en operation) som bara kan utföras om det aktuella objektet är i ett lämpligt tillstånd.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // getVarde returnerar heltalets numeriska värde
    public synchronized int getVarde ()
    {
        return this.varde;
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";
```

```
    s = "[" + v + ", " + n + "]";

    return s;
}

// set bestämmer heltalets numeriska värde och namn utifrån
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
    case 7:
        n = "sju";
        break;
    case 8:
        n = "åtta";
        break;
    case 9:
        n = "nio";
        break;
    default:
        n = "inte ensiffrigt heltal";
        break;
   }

   if (v < 0)
       n = "minus " + n;
```

```
        synchronized (this)
        {
            // sätt heltalets numeriska värde
            this.varde = v;

            // sätt heltalets namn
            this.namn = n;
        }
    }

    // add lägger till ett givet heltalsvärde till heltalet
    public synchronized void add (int p)
    {
        int    v = this.varde + p;

        this.set (v);
    }
}
```

SynkroniseraTillstandsberoendeOperationerPaKlientsidan.java

Ett program som illustrerar synkronisering av tillståndsberoende operationer på klientsidan

Tillståndsberoende operationer kan synkroniseras på klientsidan, om de inte är synkroniserade i definitionsklasserna för de objekt som används. Det går även att kombinera flera operationer, och i klientklassen ange de villkor som måste vara uppfyllda för att det ska bli möjligt att utföra motsvarande kombination av operationer (en sammansatt operation).

```
// Anvandare1 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare1 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare1 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
```

```
        {
            // 0, 1 eller 2
            int    p = (int) (3 * Math.random ());

            synchronized (heltal)
            {
                // vänta med ändringen, om den skulle orsaka ett
                // otillåtet tillstånd
                try
                {
                    while (heltal.getVarde () + p < -9  ||
                           heltal.getVarde () + p > 9)
                        heltal.wait ();
                }
                catch (InterruptedException e)
                {}

                // ändra heltalet
                heltal.add (p);

                // informera alla de trådar som väntar på
                // att heltalet ändras om ändringen
                heltal.notifyAll ();

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// Anvandare2 definierar en tråd som använder
// ett ensiffrigt heltal
class Anvandare2 implements Runnable
{
    private EnsiffrigtHeltal    heltal;

    public Anvandare2 (EnsiffrigtHeltal heltal)
    {
        this.heltal = heltal;
    }

    public void run ()
    {
        for (int i = 0; i < 4; i++)
        {
            // -1 eller -2
            int    p = - (int) (2 * Math.random () + 1);

            synchronized (heltal)
            {
```

```
                // vänta med ändringen, om den skulle orsaka ett
                // otillåtet tillstånd
                try
                {
                    while (heltal.getVarde () + p < -9  ||
                           heltal.getVarde () + p > 9)
                        heltal.wait ();
                }
                catch (InterruptedException e)
                {}

                // ändra heltalet
                heltal.add (p);

                // informera alla de trådar som väntar på
                // att heltalet ändras om ändringen
                heltal.notifyAll ();

                // visa heltalet
                System.out.println (heltal);
            }
        }
    }
}

// SynkroniseraTillstandsberoendeOperationerPaKlientsidan
// skapar ett ensiffrigt heltal och två trådar
// som använder detta heltal.
class SynkroniseraTillstandsberoendeOperationerPaKlientsidan
{
    public static void main (String[] args)
    {
        System.out.println ("SYNKRONISERA TILLSTÅNDSBEROENDE"
                          + " OPERATIONER PÅ KLIENTSIDAN\n");

        // ett ensiffrigt heltal
        EnsiffrigtHeltal    n = new EnsiffrigtHeltal ();
        n.set (5);

        // två trådar som använder heltalet
        Thread    t1 = new Thread (new Anvandare1 (n));
        Thread    t2 = new Thread (new Anvandare2 (n));
        t1.start ();
        t2.start ();
        // Trådarna utför även add-operationen,
        // som bara kan utföras om det aktuella objektet
        // är i ett lämpligt tillstånd.
        // Eftersom det aktuella objektets tillstånd kontrolleras
        // innan operationen utförs (i klasserna Anvandare1
        // och Anvandare2), kan objektet inte hamna
        // i ett otillåtet tillstånd.
```

```
    }
}
```

Programmets utmatning vid en exekvering

```
SYNKRONISERA TILLSTÅNDSBEROENDE OPERATIONER PÅ KLIENTSIDAN

[7, sju]
[7, sju]
[9, nio]
[7, sju]
[5, fem]
[3, tre]
[2, två]
[3, tre]
```

Synkroniseringsobjekt

Synkroniseringslås

Rectangle.java

En klass som utför synkroniseringen med särskilda synkroniseringslås

Ett särskilt synkroniseringslås kan skapas och användas för att synkronisera en kritisk sektion.

```
import java.awt.Dimension;
import java.util.concurrent.locks.*;  // Lock, ReentrantLock

// Klassen Rectangle representerar en rektangel
class Rectangle
{
    // rektangelns bredd
    private int    width = 5;

    // rektangelns höjd
    private int    height = 5;

    // namn på rektangelns hörn
    private String[]    name = {"A", "B", "C", "D"};

    // rektangelns synkroniseringslås
    private Lock    sizeLock = new ReentrantLock ();
    private Lock    nameLock = new ReentrantLock ();

    // Två skilda lås ska användas. Det ena låset ska användas för
    // att synkronisera ändringen och avläsningen av rektangelns
    // dimensioner. Det andra låset ska användas
    // för att synkronisera ändringen och avläsningen
    // av rektangelns namn. Rektangelns namn och dimensioner
    // är oberoende av varandra, och därför behöver man inte
    // använda samma lås. När man till exempel ändrar
    // rektangelns dimensioner, kan samtidigt rektangelns namn
    // bearbetas. På så sätt optimeras synkroniseringen (en tråd
    // ska inte vänta om den inte behöver göra det).

    // Rectangle skapar en förvald rektangel
    public Rectangle ()
    {}

    // setSize tilldelar rektangelns dimensioner
```

```
    public void setSize (int width, int height)
    {
        sizeLock.lock ();
        this.width = width;
        this.height = height;
        sizeLock.unlock ();
    }

    // getSize returnerar rektangelns dimensioner
    public Dimension getSize ()
    {
        sizeLock.lock ();
        Dimension    d = new Dimension (this.width, this.height);
        sizeLock.unlock ();

        return d;
    }

    // setName tilldelar rektangelns namn
    public void setName (String[] name)
    {
        nameLock.lock ();
        for (int i = 0; i < name.length; i++)
            this.name[i] = name[i];
        nameLock.unlock ();
    }

    // getName returnerar rektangelns namn
    public String getName ()
    {
        nameLock.lock ();
        String    rectName = "";
        for (int i = 0; i < name.length; i++)
            rectName += name[i];
        nameLock.unlock ();

        return rectName;
    }
}
```

SynkroniseringsLas.java

Ett program som använder en klass som använder särskilda synkroniseringslås

Förutom de inbyggda låsen (som varje objekt i Java har), kan även särskilda objekt som representerar lås skapas och användas. Ett lås kan låsas och

låsas upp. Endast en tråd åt gången kan äga ett sådant lås. Ett lås kan därför användas för att synkronisera användningen av kritiska kodsektioner i ett program.

Synkroniseringen kan optimeras genom att olika lås används för olika kodsektioner. Ett lås används för att synkronisera en uppsättning kritiska sektioner (som på något sätt hör ihop) och ett annat lås för att synkronisera en annan uppsättning kritiska sektioner (som är oberoende av den första uppsättningen kritiska sektioner).

```
// Anvandare1 definierar en tråd som modifierar en rektangels
// dimensioner
class Anvandare1 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare1 (Rectangle rektangel)
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // heltal mellan 1 och 10
            int    v = (int) (10 * Math.random () + 1);

            // ändra rektangelns dimensioner
            rektangel.setSize (v, v);

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Anvandare2 definierar en tråd som avläser en rektangels
// dimensioner
class Anvandare2 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare2 (Rectangle rektangel)
    {
```

```
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // rektangelns dimensioner
            java.awt.Dimension    d = rektangel.getSize ();
            System.out.println (
                    "[" + d.width + ", " + d.height + "]");

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Anvandare3 definierar en tråd som modifierar en rektangels
// namn
class Anvandare3 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare3 (Rectangle rektangel)
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        final String[][]    namn = { {"A", "B", "C", "D"},
                                     {"F", "G", "H", "I"},
                                     {"P", "Q", "R", "S"} };

        for (int i = 0; i < 5; i++)
        {
            // ett heltal mellan 0 och 2
            int    v = (int) (3 * Math.random ());

            // ändra rektangelns namn
            rektangel.setName (namn[v]);

            // vänta en stund
            try
            {
```

```
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Anvandare4 definierar en tråd som avläser en rektangels namn
class Anvandare4 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare4 (Rectangle rektangel)
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // rektangelns namn
            String    namn = rektangel.getName ();
            System.out.println (namn);

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// SynkroniseringsLas skapar en rektangel
// och flera trådar som använder denna rektangel.
class SynkroniseringsLas
{
    public static void main (String[] args)
    {
        System.out.println ("SYNKRONISERINGSLÅS\n");

        // en rektangel
        Rectangle    rektangel = new Rectangle ();

        // flera trådar som använder rektangeln
        Thread    t1 = new Thread (new Anvandare1 (rektangel));
        Thread    t2 = new Thread (new Anvandare2 (rektangel));
```

```
        Thread    t3 = new Thread (new Anvandare3 (rektangel));
        Thread    t4 = new Thread (new Anvandare4 (rektangel));
        t1.start ();
        t2.start ();
        t3.start ();
        t4.start ();
        // flera trådar använder rektangeln
        // på ett synkroniserat sätt
    }
}
```

Programmets utmatning vid en exekvering

```
SYNKRONISERINGSLÅS

ABCD
FGHI
FGHI
FGHI
[8, 8]
PQRS
[7, 7]
[7, 7]
[5, 5]
[5, 5]
```

Programmets utmatning vid en exekvering om synkroniseringen i klassen `Rectangle` tas bort

```
SYNKRONISERINGSLÅS

PBCD
PQCD
PQRD
[8, 4]
PQRS
[8, 8]
PQRS
[9, 8]
[9, 9]
[3, 9]
```

Läslås och skrivlås

Rectangle.java

En klass som utför synkroniseringen med särskilda läslås och skrivlås

För att optimera synkroniseringen kan separata läslås och skrivlås användas. Ett skrivlås kan bara ägas av en tråd i taget. Alla andra trådar utesluts. Ett skrivlås används för att synkronisera ändringar av ett objekt.

Ett läslås kan ägas samtidigt av flera lästrådar (de trådar som avläser olika uppgifter i samband med ett objekt, men som inte ändrar objektet). Det utesluter alla skrivtrådar (de trådar som ändrar objektet).

```
import java.awt.Dimension;
import java.util.concurrent.locks.*;  // Lock,
                                      // ReentrantReadWriteLock

// Klassen Rectangle representerar en rektangel
class Rectangle
{
    // rektangelns bredd
    private int    width = 5;

    // rektangelns höjd
    private int    height = 5;

    // rektangelns synkroniseringslås
    private ReentrantReadWriteLock    rwl =
                                new ReentrantReadWriteLock ();
    private Lock    readLock = rwl.readLock ();
    private Lock    writeLock = rwl.writeLock ();

    // Två skilda lås ska användas. Det ena låset (skrivlåset) ska
    // användas för att synkronisera rektangelns ändringar.
    // Det andra låset (läslåset) ska användas för att
    // synkronisera de metoder som avläser olika uppgifter
    // om rektangeln, men som inte ändrar den.
    // Flera trådar kan använda rektangeln samtidigt, så
    // länge de inte ändrar den. En tråd ska inte vänta,
    // om den inte behöver göra det.

    // Rectangle skapar en förvald rektangel
    public Rectangle ()
    {}

    // setSize tilldelar rektangelns dimensioner
```

```
    public void setSize (int width, int height)
    {
        writeLock.lock ();
        this.width = width;
        this.height = height;
        writeLock.unlock ();
    }

    // getSize returnerar rektangelns dimensioner
    public Dimension getSize ()
    {
        readLock.lock ();
        Dimension    d = new Dimension (this.width, this.height);
        readLock.unlock ();

        return d;
    }

    // getDiagonal returnerar rektangelns diagonal
    public double getDiagonal ()
    {
        readLock.lock ();
        double    d = Math.sqrt (this.width * this.width +
                                 this.height * this.height);
        readLock.unlock ();

        return d;
    }
}
```

LaslasOchSkrivlas.java

Ett program som använder en klass som använder särskilda läslås och skrivlås

För att optimera synkroniseringen, kan separata läslås och skrivlås användas. Ett läslås kan ägas samtidigt av flera lästrådar, men utesluter alla skrivtrådar. Ett skrivlås kan bara ägas av ett skrivtråd och utesluter alla andra trådar.

```
// Anvandare1 definierar en tråd som modifierar en rektangels
// dimensioner
class Anvandare1 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare1 (Rectangle rektangel)
```

```
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // heltal mellan 1 och 10
            int    v = (int) (10 * Math.random () + 1);

            // ändra rektangelns dimensioner
            rektangel.setSize (v, v);

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Anvandare2 definierar en tråd som avläser en rektangels
// dimensioner
class Anvandare2 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare2 (Rectangle rektangel)
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // rektangelns dimensioner
            java.awt.Dimension    d = rektangel.getSize ();
            System.out.println (
                    "[" + d.width + ", " + d.height + "]");

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
```

```
            {}
        }
    }
}

// Anvandare3 definierar en tråd som avläser en rektangels
// diagonal
class Anvandare3 implements Runnable
{
    private Rectangle    rektangel;

    public Anvandare3 (Rectangle rektangel)
    {
        this.rektangel = rektangel;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // rektangelns diagonal
            double    diagonal = rektangel.getDiagonal ();
            System.out.println (diagonal);

            // vänta en stund
            try
            {
                Thread.sleep (100);
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// LaslasOchSkrivlas skapar en rektangel,
// och flera trådar som använder denna rektangel.
class LaslasOchSkrivlas
{
    public static void main (String[] args)
    {
        System.out.println ("LÄSLÅS OCH SKRIVLÅS");
        System.out.println ();

        // en rektangel
        Rectangle    rektangel = new Rectangle ();

        // flera trådar som använder rektangeln
        Thread    t1 = new Thread (new Anvandare1 (rektangel));
        Thread    t2 = new Thread (new Anvandare2 (rektangel));
        Thread    t3 = new Thread (new Anvandare3 (rektangel));
```

```
        t1.start ();
        t2.start ();
        t3.start ();
        // flera trådar använder rektangeln på
        // ett synkroniserat sätt
    }
}
```

Programmets utmatning vid en exekvering

```
LÄSLÅS OCH SKRIVLÅS

12.727922061357855
12.727922061357855
[9, 9]
4.242640687119285
[3, 3]
1.4142135623730951
[1, 1]
1.4142135623730951
[1, 1]
[8, 8]
```

Programmets utmatning vid en exekvering om synkroniseringen i klassen `Rectangle` tas bort

```
LÄSLÅS OCH SKRIVLÅS

5.830951894845301
4.242640687119285
[4, 3]
5.0
[4, 4]
5.656854249492381
[7, 4]
9.899494936611665
[7, 7]
[6, 7]
```

Villkorsobjekt

Square.java

En klass som utför synkroniseringen av olika tillståndsberoende operationer med särskilda villkorsobjekt

Ett villkor i en klass kan representeras med ett särskilt villkorsobjekt. Ett villkorsobjekt binds alltid till ett konkret synkroniseringslås. Det kan finnas flera villkorsobjekt som är bundna till ett och samma lås.

Genom att olika villkorsobjekt används för olika villkor, kan synkroniseringen optimeras. En tråd väntar på ett exakt formulerat villkor, och så snart detta villkor är uppfyllt aktiveras tråden. Tråden behöver inte vänta på de villkor som inte är intressanta för den operation som den utför.

```
import java.util.concurrent.locks.*; // Lock, ReentrantLock,
                                     // Condition

// Klassen Square representerar en kvadrat vars storlek kan
// ändras inom givna gränser.
class Square
{
    // största och minsta längd av kvadratens sida
    public static final int    MIN = 1;
    public static final int    MAX = 10;

    // längden av kvadratens sida
    private int    length = 5;

    // kvadratens synkroniseringslås
    private Lock    lock = new ReentrantLock ();

    // kvadratens villkorsobjekt
    private Condition    canBeIncreased = lock.newCondition ();
    private Condition    canBeDecreased = lock.newCondition ();

    // Två skilda villkorsobjekt ska användas.
    // Under vissa villkor kan kvadraten ökas, och dessa
    // villkor representeras med ett av de två villkorsobjekten.
    // Det andra villkorsobjektet representerar de villkor
    // som behövs för att kunna minska kvadraten.

    // Square skapar en förvald kvadrat
    public Square ()
    {}

    // getLength returnerar längden av kvadratens sida
```

```
public int getLength ()
{
    lock.lock ();
    int    len = this.length;;
    lock.unlock ();

    return len;
}

// increase ökar kvadratens sida med ett givet värde
public void increase (int increment)
                        throws InterruptedException
{
    lock.lock ();
    increment = Math.abs (increment);

    try
    {
        // vänta så länge ökningen inte kan utföras
        while (this.length + increment > MAX)
            canBeIncreased.await ();
        this.length += increment;

        // Aktivera de trådar som väntar på
        // att minska kvadraten
        // (efter den senaste ökningen kan eventuellt
        // minskningsoperationen utföras).
        // Dessa trådar blir körbara, och kan eventuellt få
        // möjlighet att exekvera.
        canBeDecreased.signalAll ();
        // de trådar som väntar på ökningen aktiveras inte
        // (villkoren för ökningsoperationen
        // har inte förbättrats)
    }
    finally
    {
        lock.unlock ();
        // låset ska släppas oavsett om metoden avslutas
        // normalt eller genom undantag
    }
}

// decrease minskar kvadratens sida med ett givet värde
public void decrease (int decrement)
                           throws InterruptedException
{
    lock.lock ();
    decrement = Math.abs (decrement);

    try
    {
```

```
            // vänta så länge minskningen inte kan utföras
            while (this.length - decrement < MIN)
                canBeDecreased.await ();
            this.length -= decrement;

            // aktivera de trådar som väntar på att öka kvadraten
            // (efter den senaste minskningen kan eventuellt
            // ökningsoperationen utföras)
            canBeIncreased.signalAll ();
            // de trådar som väntar på minskningen aktiveras inte
            // (villkor för minskningsoperationen har inte
            // förbättrats)
        }
        finally
        {
            lock.unlock ();
        }
    }
}
```

Villkorsobjekt.java

Ett program som använder en klass med flera villkorsobjekt

För att optimera synkroniseringen, kan flera villkorsobjekt användas. Ett villkorsobjekt representerar ett exakt formulerat villkor i en klass. Så snart det villkoret är uppfyllt, kan de trådar som väntar på villkoret återaktiveras. De behöver inte vänta på de villkor som inte är intressanta för deras operationer.

```
// Anvandare1 definierar en tråd som ökar en kvadrat
class Anvandare1 implements Runnable
{
    private Square    kvadrat;

    public Anvandare1 (Square kvadrat)
    {
        this.kvadrat = kvadrat;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // öka kvadraten
            try
```

```
            {
                kvadrat.increase (2);
                System.out.print (kvadrat.getLength () + " ");
                // om dessa två satser placeras
                // i ett synkroniserat block, och om detsamma
                // görs i klassen Anvandare2,
                // kan ett dödläge uppstå.
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Anvandare2 definierar en tråd som minskar en kvadrat
class Anvandare2 implements Runnable
{
    private Square    kvadrat;

    public Anvandare2 (Square kvadrat)
    {
        this.kvadrat = kvadrat;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // minska kvadraten
            try
            {
                kvadrat.decrease (2);
                System.out.print (kvadrat.getLength () + " ");
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// Villkorsobjekt skapar en kvadrat, och två trådar som använder
// denna kvadrat
class Villkorsobjekt
{
    public static void main (String[] args)
    {
        System.out.println ("VILLKORSOBJEKT");
        System.out.println ();

        // en kvadrat
        Square    kvadrat = new Square ();
```

```
        // två trådar som använder kvadraten
        Thread    t1 = new Thread (new Anvandare1 (kvadrat));
        Thread    t2 = new Thread (new Anvandare2 (kvadrat));
        t1.start ();
        t2.start ();
        // Två trådar använder kvadraten på
        // ett synkroniserat sätt.
        // Ökningstråden förbereder villkor för minskningstråden,
        // och tvärtom. Kvadratens storlek förblir inom
        // givna gränser.
    }
}
```

Programmets utmatning vid en exekvering

```
VILLKORSOBJEKT

7 9 7 5 3 1 3 5 7 5
```

Programmets utmatning vid en exekvering om väntandet på villkor i klassen `Square` tas bort

```
VILLKORSOBJEKT

7 9 11 13 15 13 11 9 7 5
```

Kommunikation mellan trådar

Objektöverföring via en kanal

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` representerar ett ensiffrigt heltal.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // getVarde returnerar heltalets numeriska värde
    public synchronized int getVarde ()
    {
        return this.varde;
    }

    // toString returnerar heltalets strängrepresentation
    public String toString ()
    {
        int       v = 0;
        String    n = "";
        synchronized (this)
        {
            v = this.varde;
            n = this.namn;
        }

        String    s = "";

        s = "[" + v + ", " + n + "]";

        return s;
```

```
}

// set bestämmer heltalets numeriska värde och namn utifrån
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
    case 7:
        n = "sju";
        break;
    case 8:
        n = "åtta";
        break;
    case 9:
        n = "nio";
        break;
    default:
        n = "inte ensiffrigt heltal";
        break;
   }

   if (v < 0)
       n = "minus " + n;

    synchronized (this)
    {
```

```
            // sätt heltalets numeriska värde
            this.varde = v;

            // sätt heltalets namn
            this.namn = n;
        }
    }

    // setNamn sätter heltalets namn till ett givet namn
    public synchronized void setNamn (String namn)
    {
        this.namn = namn;
    }
}
```

Channel.java

Ett gränssnitt som definierar en kanal för objektöverföring

Gränssnittet `Channel` definierar en kanal för objektöverföring. Ett objekt kan placeras i en kanal, och ett objekt kan tas från kanalen.

```
// Gränssnittet Channel definierar en kanal för objektöverföring
public interface Channel<T>
{
    // put placerar ett givet objekt i kanalen
    void put (T object) throws InterruptedException;

    // take tar ett objekt från kanalen och returnerar det
    T take () throws InterruptedException;
}
```

ObjectContainer.java

En klass som representerar en objektbehållare

Klassen `ObjectContainer` representerar en behållare för ett objekt. Ett objekt kan placeras i en objektbehållare, och objektet kan tas från objektbehållaren.

```
// Klassen ObjectContainer representerar en
// behållare för ett objekt
public class ObjectContainer<T> implements Channel<T>
{
```

```
    // objekt i objektbehållaren
    private T    object;

    // objektbehållarens status (full eller tom)
    private boolean    full;

    // ObjectContainer skapar en tom objektbehållare
    public ObjectContainer ()
    {
        object = null;
        full = false;
    }

    // put placerar ett givet objekt i objektbehållaren
    public synchronized void put (T object)
                      throws InterruptedException
    {
        // vänta om objektbehållaren full
        while (full)
            this.wait ();

        // placera objektet i objektbehållaren
        this.object = object;

        // nu är objektbehållaren full
        full = true;
        this.notifyAll ();
    }

    // take tar ett objekt från objektbehållaren
    // och returnerar det
    public synchronized T take ()
                      throws InterruptedException
    {
        // vänta om objektbehållaren tom
        while (!full)
            this.wait ();

        // ta objektet - objektbehållaren blir tom
        full = false;
        this.notifyAll ();

        return object;
    }
}
```

Buffer.java

En klass som representerar en buffert med obegränsad kapacitet

Klassen `Buffer` representerar en buffert med obegränsad kapacitet. En buffert av typen `Buffer` är en objektbehållare, som kan lagra flera objekt.

Ett objekt kan placeras i en buffert, och ett objekt kan tas från bufferten. Objekten tas från bufferten i den ordning som används vid placeringen (först i bufferten, först ur bufferten).

```
// Klassen Buffert representerar en buffert
// med obegränsad kapacitet
class Buffer<T> implements Channel<T>
{
    // lagringsplats för objekt
    java.util.ArrayList<T>    v;

    // Buffer skapar en tom buffert
    public Buffer ()
    {
        v = new java.util.ArrayList<T> ();
    }

    // put placerar ett givet objekt som sista objekt i bufferten
    public synchronized void put (T object)
    {
        // placera objekt som sista objekt i bufferten
        v.add (object);

        this.notifyAll ();
    }

    // take tar det första objektet från bufferten
    // och returnerar det
    public synchronized T take ()
                      throws InterruptedException
    {
        // vänta om bufferten tom
        while (v.isEmpty ())
            this.wait ();

        // ta första objektet från bufferten
        T    object = v.get (0);
        v.remove (0);

        return object;
    }
}
```

ObjektoverforingViaEnKanal.java

Ett program som illustrerar hur en tråd skickar objekt till en annan tråd via en kanal

Flera trådar kan kommunicera med varandra genom att skicka och ta emot olika objekt.

En kanal kan skapas, och användas för objektöverföring mellan olika trådar. En tråd kan placera ett objekt i kanalen, och en annan tråd kan ta detta objekt från kanalen. En kanal håller de objekt som är i transit.

En kanal kan implementeras på olika sätt. En kanal kan implementeras som en behållare för ett objekt, som en ändlig buffert, som en obegränsad buffert och på andra sätt.

```
// HeltalsGenerator definierar en tråd som genererar
// ensiffriga heltal, och skickar dessa heltal till en annan tråd
class HeltalsGenerator implements Runnable
{
    // det ensiffriga heltalet som genereras och skickas
    private EnsiffrigtHeltal    heltal;

    // kanal som används för att skicka de genererade heltalen
    private Channel<EnsiffrigtHeltal>    kanal;

    // HeltalsGenerator skapar en heltalsgenerator, och binder den
    // till en given kanal
    public HeltalsGenerator (Channel<EnsiffrigtHeltal> kanal)
    {
        this.kanal = kanal;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // generera ett slumpmässigt ensiffrigt heltal
            int    p = (int) (10 * Math.random ());
            heltal = new EnsiffrigtHeltal ();
            heltal.set (p);

            // visa heltalet
            System.out.println (heltal);

            // placera heltalet i kanalen
            try
            {
                kanal.put (heltal);
```

```
                // En referens till det genererade heltalet
                // skickas. Mottagartråden ska få en referens
                // till samma heltal, och kan påverka det
                // (till exempel översätta dess namn till ett
                // annat språk).
                // Om man inte vill att mottagartråden ska påverka
                // originalobjektet, kan en kopia av objektet
                // skapas och en referens till denna kopia skickas
            }
            catch (InterruptedException e)
            {}
        }
    }
}

// HeltalsOversattare definierar en tråd som tar emot
// ensiffriga heltal från en annan tråd, och
// översätter dessa heltal till engelska
class HeltalsOversattare implements Runnable
{
    // det ensiffriga heltalet som tas emot och översätts
    private EnsiffrigtHeltal    heltal;

    // den kanalen från vilken de ensiffriga heltalen tas
    private Channel<EnsiffrigtHeltal>    kanal;

    // HeltalsOversattare skapar en heltalsöversättare,
    // och binder den till en given kanal
    public HeltalsOversattare (Channel<EnsiffrigtHeltal> kanal)
    {
        this.kanal = kanal;
    }

    public void run ()
    {
        for (int i = 0; i < 5; i++)
        {
            // ta det ensiffriga heltalet från kanalen
            try
            {
                heltal = kanal.take ();

                // översättningstråden har nu tillgång till det
                // heltal som generatortråden genererat
                // (man får tillgång till samma objekt
                // via en annan referens)
            }
            catch (InterruptedException e)
            {}
```

```
// heltalets numeriska värde
int    v = heltal.getVarde ();

// översätt heltalets namn
String    n = "";  // heltalets nya namn
int    absv = Math.abs (v);
switch (absv)
{
case 0:
    n = "zero";
    break;
case 1:
    n = "one";
    break;
case 2:
    n = "two";
    break;
case 3:
    n = "three";
    break;
case 4:
    n = "four";
    break;
case 5:
    n = "five";
    break;
case 6:
    n = "six";
    break;
case 7:
    n = "seven";
    break;
case 8:
    n = "eight";
    break;
case 9:
    n = "nine";
    break;
default:
    n = "not correct value";
    break;
}

if (v < 0)
    n = "minus " + n;

// sätt heltalets nya namn
heltal.setNamn (n);
// (Originalheltalet ändras, det som heltalsgenerator
// skapat. Man kunde istället skapa ett nytt
// ensiffrigt heltal, som skulle representera
```

```
            // översättningen)

            // visa heltalet
            System.out.println (heltal);
        }
    }
}

// ObjektoverforingViaEnKanal skapar en tråd
// som genererar slumpmässiga ensiffriga heltal,
// och en tråd som översätter de genererade heltalen
// till engelska.
class ObjektoverforingViaEnKanal
{
    public static void main (String[] args)
    {
        System.out.println ("OBJEKTÖVERFÖRING VIA EN KANAL");
        System.out.println ();

        // en kanal
        // (kanalens typ väljs slumpmässigt)
        Channel<EnsiffrigtHeltal>    kanal = null;
        if (Math.random () < 0.5)
            kanal = new ObjectContainer<EnsiffrigtHeltal> ();
        else
            kanal = new Buffer<EnsiffrigtHeltal> ();

        // en generator av ensiffriga heltal
        Thread    t1 = new Thread (new HeltalsGenerator (kanal));

        // en översättare av ensiffriga heltal
        Thread   t2 = new Thread (new HeltalsOversattare (kanal));

        // starta trådarna
        t1.start ();
        t2.start ();
        // Tråden t1 genererar ensiffriga heltal, och placerar dem
        // i kanalen. Tråden t2 tar dessa heltal från kanalen, och
        // översätter dem.
    }
}
```

Programmets utmatning vid en exekvering

```
OBJEKTÖVERFÖRING VIA EN KANAL

[8, åtta]
[-8, minus åtta]
[5, fem]
[-1, minus ett]
[9, nio]
```

```
[8, eight]
[-8, minus eight]
[5, five]
[-1, minus one]
[9, nine]
```

Programmets utmatning vid en annan exekvering

```
OBJEKTÖVERFÖRING VIA EN KANAL

[1, ett]
[-4, minus fyra]
[1, one]
[7, sju]
[-4, minus four]
[6, sex]
[7, seven]
[3, tre]
[6, six]
[3, three]
```

Respons från mottagaren

EnsiffrigtHeltal.java

En klass som representerar ett ensiffrigt heltal

Klassen `EnsiffrigtHeltal` representerar ett ensiffrigt heltal.

```
// Klassen EnsiffrigtHeltal representerar ett ensiffrigt heltal.
class EnsiffrigtHeltal
{
    // heltalets numeriska värde
    private int    varde;

    // heltalets namn
    private String    namn;

    // EnsiffrigtHeltal skapar ett förvalt ensiffrigt heltal (0)
    public EnsiffrigtHeltal ()
    {
        varde = 0;
        namn = "noll";
    }

    // getVarde returnerar heltalets numeriska värde
    public synchronized int getVarde ()
    {
```

```
    return this.varde;
}

// toString returnerar heltalets strängrepresentation
public String toString ()
{
    int       v = 0;
    String    n = "";
    synchronized (this)
    {
        v = this.varde;
        n = this.namn;
    }

    String    s = "";

    s = "[" + v + ", " + n + "]";

    return s;
}

// set bestämmer heltalets numeriska värde och namn utifrån
// ett givet heltalsvärde
public void set (int v)
{
    // bestäm heltalets namn
    String    n = "";
    int    absv = Math.abs (v);
    switch (absv)
    {
    case 0:
        n = "noll";
        break;
    case 1:
        n = "ett";
        break;
    case 2:
        n = "två";
        break;
    case 3:
        n = "tre";
        break;
    case 4:
        n = "fyra";
        break;
    case 5:
        n = "fem";
        break;
    case 6:
        n = "sex";
        break;
```

```
            case 7:
                n = "sju";
                break;
            case 8:
                n = "åtta";
                break;
            case 9:
                n = "nio";
                break;
            default:
                n = "inte ensiffrigt heltal";
                break;
           }

           if (v < 0)
               n = "minus " + n;

            synchronized (this)
            {
                // sätt heltalets numeriska värde
                this.varde = v;

                // sätt heltalets namn
                this.namn = n;
            }
        }


        // setNamn sätter heltalets namn till ett givet namn
        public synchronized void setNamn (String namn)
        {
            this.namn = namn;
        }
}
```

Intermediary.java

En klass som representerar en behållare för ett meddelande och motsvarande svar

`Intermediary` representerar en behållare för ett par objekt: ett objekt som representerar ett meddelande och ett objekt som representerar svaret på detta meddelande.

En tråd kan använda ett objekt av typen `Intermediary` för att lämna ett meddelande till en annan tråd, och för att få svar från denna tråd.

```
// Klassen Intermediary representerar en tillfällig
// mellanplats mellan två trådar som kommunicerar - en tråd
// lämnar på denna plats ett meddelande till en annan tråd,
// och får här svar från den andra tråden
class Intermediary<T1, T2>
{
    // ett objekt som representerar ett meddelande
    private T1    message;

    // ett objekt som representerar ett svar
    private T2    reply;

    // om svaret är klart
    private boolean    replied;

    // Intermediary skapar en behållare för ett meddelande och
    // motsvarande svar utifrån ett givet meddelande
    public Intermediary (T1 message)
    {
        this.message = message;

        this.reply = null;
        this.replied = false;
    }

    // getMessage returnerar det objekt som
    // representerar meddelandet
    public synchronized T1 getMessage ()
    {
        return message;
    }

    // setReply bestämmer det objekt som representerar svaret
    public synchronized void setReply (T2 reply)
    {
        this.reply = reply;

        this.replied = true;
        this.notifyAll ();
    }

    // getReply returnerar det objekt som representerar svaret
    public synchronized T2 getReply ()
                      throws InterruptedException
    {
        while (!replied)
            this.wait ();

        return reply;
    }
}
```

ResponsFranMottagaren.java

Ett program som illustrerar hur en tråd skickar ett meddelande till en annan tråd, och väntar på svar från denna tråd

En tråd som skickar ett meddelande till en annan tråd kan av någon anledning behöva svar från den andra tråden. Detta svar kan vara ett resultat av meddelandets hantering eller bara en bekräftelse på att meddelandet har hanterats.

En tråd kan vänta på mottagarens svar, och fortsätta exekvera efter att svaret kommit. På så sätt synkroniseras sändarens och mottagarens aktiviteter. Ett alternativ till detta är att sändaren skickar meddelandet och fortsätter med sina aktiviteter, och vid ett senare tillfälle kontrollerar mottagarens svar.

```
// HeltalsGenerator definierar en tråd som genererar
// ensiffriga heltal, skickar dem till en annan tråd,
// och tar heltalets översättningar från den tråden
class HeltalsGenerator implements Runnable
{
    // det ensiffriga heltal som genereras
    private EnsiffrigtHeltal    heltal;

    // kanal för de objekt som innehåller de ensiffriga
    // heltalen som genereras och deras översättningar
    private Channel<
        Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>>   kanal;

    // HeltalsGenerator skapar en heltalsgenerator, och binder den
    // till en given kanal
    public HeltalsGenerator (Channel<
        Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>> kanal)
    {
        this.kanal = kanal;
    }


    public void run ()
    {
        // behållare för meddelandet och motsvarande svar
        Intermediary<EnsiffrigtHeltal,EnsiffrigtHeltal> im = null;

        // det heltal som representerar översättningen
        EnsiffrigtHeltal    oversattning = null;

        for (int i = 0; i < 5; i++)
```

```
        {
            // generera ett slumpmässigt ensiffrigt heltal
            int    p = (int) (10 * Math.random ());
            heltal = new EnsiffrigtHeltal ();
            heltal.set (p);

            // visa heltalet
            System.out.println (heltal);

            // förbered en behållare för översättningen (svaret),
            // och packa heltalet (meddelandet) i behållaren
            im = new Intermediary<
                     EnsiffrigtHeltal, EnsiffrigtHeltal> (heltal);

            // placera behållaren i kanalen,
           // och ta sedan översättningen från behållaren
            try
            {
                kanal.put (im);

                // här väntar tråden på svar
                oversattning = im.getReply ();
            }
            catch (InterruptedException e)
            {}

            // visa översättningen
            System.out.println (oversattning);
        }
    }
}

// HeltalsOversattare definierar en tråd som tar emot
// ensiffriga heltal från en annan tråd, översätter
// dessa heltal till ett annat språk och skickar
// tillbaka översättningarna
class HeltalsOversattare implements Runnable
{
    // det ensiffriga heltalet som översätts
    private EnsiffrigtHeltal    heltal;

    // den kanal från vilken de packade ensiffriga heltalen tas
    private Channel<
        Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>>  kanal;

    // HeltalsOversattare skapar en heltalsöversättare,
    // och binder den till en given kanal
    public HeltalsOversattare (Channel<
        Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>> kanal)
    {
        this.kanal = kanal;
```

```
}

public void run ()
{
    // behållare för meddelandet och motsvarande svar
    Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>    im =
                                                          null;

    // det heltal som representerar översättningen
    EnsiffrigtHeltal    oversattning = null;

    for (int i = 0; i < 5; i++)
    {
        // ta det ensiffriga heltalet från kanalen
        try
        {
            // ta behållaren (som innehåller heltalet)
            im = kanal.take ();

            // erhåll heltalet (meddelandet) från behållaren
            heltal = im.getMessage ();
        }
        catch (InterruptedException e)
        {}

        // heltalets numeriska värde
        int    v = heltal.getVarde ();

        // översätt heltalets namn
        String    n = "";  // heltalets nya namn
        int    absv = Math.abs (v);
        switch (absv)
        {
        case 0:
            n = "zero";
            break;
        case 1:
            n = "one";
            break;
        case 2:
            n = "two";
            break;
        case 3:
            n = "three";
            break;
        case 4:
            n = "four";
            break;
        case 5:
            n = "five";
            break;
```

```
            case 6:
                n = "six";
                break;
            case 7:
                n = "seven";
                break;
            case 8:
                n = "eight";
                break;
            case 9:
                n = "nine";
                break;
            default:
                n = "not correct value";
                break;
            }

            if (v < 0)
                n = "minus " + n;

            // skapa ett heltal som representerar översättningen

            // skapa översättningen (svaret)
            oversattning = new EnsiffrigtHeltal ();
            oversattning.set (v);
            oversattning.setNamn (n);
            // Ett nytt heltal, som representerar översättningen,
            // skapas. Det skickade heltalets namn ändras inte.

            // packa översättningen i behållaren (svara)
            im.setReply (oversattning);
        }
    }
}

// ResponsFranMottagaren skapar en tråd som genererar slumpmässiga
// ensiffriga heltal, och en tråd som översätter
// de genererade heltalen till ett annat språk
// och skickar översättningar tillbaka till generatorn
class ResponsFranMottagaren
{
    public static void main (String[] args)
    {
        System.out.println ("RESPONS FRÅN MOTTAGAREN\n");

        // en kanal
        Channel<
         Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>> kanal =
            new Buffer<
             Intermediary<EnsiffrigtHeltal, EnsiffrigtHeltal>> ();
```

```
        // en generator av ensiffriga heltal
        Thread    t1 = new Thread (new HeltalsGenerator (kanal));

        // en översättare av ensiffriga heltal
        Thread   t2 = new Thread (new HeltalsOversattare (kanal));

        // starta trådarna
        t1.start ();
        t2.start ();
        // Tråden t1 genererar ensiffriga heltal, och placerar dem
        // i kanalen. Tråden t2 tar dessa heltal från kanalen,
        // översätter dem till ett annat språk och skickar
        // översättningar tillbaka till tråden t1. Kommunikation
        // mellan trådarna sker via ett objekt, som innehåller
        // både ett heltal och dess översättning.
    }
}
```

Programmets utmatning vid en exekvering

```
RESPONS FRÅN MOTTAGAREN

[2, två]
[2, two]
[6, sex]
[6, six]
[-2, minus två]
[-2, minus two]
[0, noll]
[0, zero]
[1, ett]
[1, one]
```

Programmets utmatning vid en annan exekvering

```
RESPONS FRÅN MOTTAGAREN

[-6, minus sex]
[-6, minus six]
[3, tre]
[3, three]
[1, ett]
[1, one]
[-5, minus fem]
[-5, minus five]
[5, fem]
[5, five]
```

Kapitel 2

Kommunikation mellan program

Kommunikation mellan två program

Ett nät av datorer
Ansluta ett Javaprogram till en server
Kommunikation mellan två Javaprogram

Ett serverprogram

En tråd per klient
En pool av trådar
Kommunikation via en server

Kommunikation mellan två program

Ett nät av datorer

IdentifieraEnDatorIInternet.java

Ett program som visar hur en dator i Internet identifieras

Varje dator i Internet (värddator, host) har sin adress. Tack vare dessa adresser kan olika datorer kommunicera med varandra. För att underlätta identifieringen av olika datorer i Internet, tilldelas dessa datorer även olika namn (förutom adresser).

```
import java.net.*;    // InetAddress
import static java.lang.System.out;

// IdentifieraEnDatorIInternet bestämmer adresser och namn
// för några datorer i Internet
class IdentifieraEnDatorIInternet
{
    public static void main (String[] args)
    {
        out.println ("IDENTIFIERA EN DATOR I INTERNET");
        out.println ();

        try
        {
            // Internetadressen för en dator vars namn anges
            InetAddress    host =
                        InetAddress.getByName ("www.kth.se");

            // Internetadressen
            out.println ("en dator i Internet: " + host);

            // datorns namn
            String    namn = host.getHostName ();
            out.println ("datorns namn: " + namn);

            // datorns adress på strängform
            String    adress = host.getHostAddress ();
            out.println ("datorns adress: " + adress);
            out.println ();

            // Internet-adressen för den lokala datorn
            // (den dator där detta program körs)
```

```
            InetAddress   lokalHost = InetAddress.getLocalHost ();

            // namn och adress av den lokala datorn
            out.println ("namnet på den lokala datorn: "
                                    + lokalHost.getHostName ());
            out.println ("adressen för den lokala datorn: "
                                  + lokalHost.getHostAddress ());
            out.println ();

            // ett gemensamt namn för (eventuellt) flera datorer
            // (som uppfyller samma funktion)
            InetAddress[]    a = InetAddress.getAllByName (
                                                 "java.sun.com");
            out.println ("namn: java.sun.com");
            out.println ("datorer bakom namnet:");
            for (int i = 0; i < a.length; i++)
                out.println (a[i]);
        }
        // om ett olämpligt namn anges för en dator
        catch (UnknownHostException e)
        {
            e.printStackTrace ();
        }
    }
}
```

Programmets utmatning på en viss dator i Internet

```
IDENTIFIERA EN DATOR I INTERNET

en dator i Internet: www.kth.se/130.237.32.51
datorns namn: www.kth.se
datorns adress: 130.237.32.51

namnet på den lokala datorn: oemcomputer
adressen för den lokala datorn: 213.112.169.137

namn: java.sun.com
datorer bakom namnet:
jjava.sun.com/209.249.116.141
```

Programmets utmatning om programmet körs på en dator som inte är kopplad till Internet

```
IDENTIFIERA EN DATOR I INTERNET

java.net.UnknownHostException: www.kth.se: www.kth.se
    at java.net.InetAddress.getAllByName0(InetAddress.java:1128)
    at java.net.InetAddress.getAllByName0(InetAddress.java:1098)
    at java.net.InetAddress.getAllByName(InetAddress.java:1061)
    at java.net.InetAddress.getByName(InetAddress.java:958)
    at IdentifieraEnDatorIInternet.main
                      (IdentifieraEnDatorIInternet.java:33)
```

Ansluta ett Javaprogram till en server

WebbKlient.java

Ett program som ansluter sig till en webbserver

Ett Javaprogram kan skapa en förbindelse till ett program som körs i en annan dator, och kommunicera med detta program. Ett klientprogram kan ansluta sig till ett serverprogram, som körs i en dator i Internet. Klienten kan sedan kommunicera med denna server enligt ett i förväg fastställt protokoll. Det går till exempel att ansluta sig till en webbserver och begära en html-fil från den. Filen som servern skickar tas sedan emot och används på något sätt.

```
import java.net.*;  // Socket
import java.io.*;   // OutputStream, PrintWriter,
                    // InputStream, InputStreamReader,
                    // BufferedReader, IOException

// WebbKlient är ett program som ansluter sig till en webbserver,
// begär en fil från denna server och tar emot och visar filen
class WebbKlient
{
    public static void main (String[] args)
    {
        System.out.println ("WEBBKLIENT");
        System.out.println ();

        // en förbindelse till en fjärrdator
        Socket    s = null;

        try
        {
            // skapa förbindelse

            // skapa en förbindelse till en angiven webbserver
            s = new Socket ("www.kth.se", 80);
            // man ansluter sig till datorn www.kth.se
            // på port 80 (en standardport, används för
            // http-kommunikation -
            // på den porten väntar serverprogrammet)

            // skapa ett lämpligt kommunikationsverktyg

            // extrahera utströmmen och skapa
            // en lämplig teckenström utifrån den
            // (för att kunna skicka begäran)
```

```
        OutputStream    os = s.getOutputStream ();
        PrintWriter    out = new PrintWriter (os, true);
        // argumentet true gör att bufferten töms vid varje
        // anrop till metoden println (prova utan true)

        // extrahera inströmmen, och skapa
        // en lämplig teckenström utifrån den
        // (för att kunna ta emot svar)
        InputStream    is = s.getInputStream ();
        BufferedReader    in = new BufferedReader (
                                new InputStreamReader (is));

        // kommunicera

        // skicka begäran till webbservern
        out.println ("GET /index.html HTTP/1.0");
        out.println ();
        // en tom rad måste ingå, annars förstår servern inte
        // begäran

        // out.close ();
        // det orsakar ett undantag senare i programmet:
        // java.net.SocketException: socket closed
        // Om en av strömmarna stängs, stängs även socketen
        // och alla andra strömmar i den –
        // de kan inte användas senare.

        // alternativt kan begäran skickas så här:
        // out.print ("GET /index.html HTTP/1.0\n\n");
        // out.flush ();
        // eftersom utmatningsbufferten inte töms vid anropet
        // till metoden print, måste metoden flush anropas för
        // att skicka det som finns i utmatningsbufferten
        // till webbservern

        // läs in och visa det som servern skickar
        // (eftersom servern skickar en text i flera rader,
        // sker läsningen radvis)
        String    rad = in.readLine ();
        while (rad != null)
        {
            System.out.println (rad);
            rad = in.readLine ();
        }
    }
    // om det inte går att upprätta en förbindelse
    // till servern, eller om något fel uppstår
    // vid dataöverföringen
    catch (IOException e)
    {
        e.printStackTrace ();
```

```
        }
        // stäng förbindelsen oavsett om programmet
        // avslutas normalt, eller via ett undantag
        // (motsvarande systemresurser och
        // nätverksresurser frigörs)
        finally
        {
            try  // close kan kasta ett undantag
            {
                s.close ();
                // när socketen stängs, stängs även
                // alla strömmar i den
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}
```

Programmets utmatning

en `html`-fil läses in och visas på standardutmatningsenheten

Om programmet körs på en dator som inte är kopplad till Internet, blir utmatningen den följande:

```
WEBBKLIENT

java.net.UnknownHostException: www.kth.se
   at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:177)
   at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:364)
        at java.net.Socket.connect(Socket.java:505)
        at java.net.Socket.connect(Socket.java:455)
        at java.net.Socket.<init>(Socket.java:363)
        at java.net.Socket.<init>(Socket.java:178)
        at WebbKlient.main(WebbKlient.java:42)
Exception in thread "main" java.lang.NullPointerException
        at WebbKlient.main(WebbKlient.java:103)
```

Kommunikation mellan två Javaprogram

AnslutandePart.java

Ett program som ansluter sig till ett annat program och kommunicerar med det

Ett Javaprogram kan ansluta sig till ett annat Javaprogram, som väntar på en viss port på en bestämd dator. Efter det kan de två programmen (det anslutande och det väntande) kommunicera med varandra.

För att kommunikationen mellan två program ska bli möjlig, måste programmen följa ett i förväg bestämt sätt att kommunicera (ett givet applikationsprotokoll).

```
import java.net.*;  // Socket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
                    // InputStreamReader, BufferedReader,
                    // IOException

// AnslutandePart är ett program som ansluter sig till
// ett väntande program, och kommunicerar med det
class AnslutandePart
{
    public static void main (String[] args)
    {
        System.out.println ("ANSLUTANDE PART");
        System.out.println ();

        // det väntande parten
        Socket    vantandePart = null;

        try
        {
            // förbered kommunikation

            // skapa en förbindelse till ett program,
            // som väntar på en given port på en given dator
            vantandePart = new Socket ("localhost", 1201);
            // man ansluter sig till ett program som väntar på
            // porten 1201, på den lokala maskinen (localhost)
            // (istället för localhost, kan namn på en annan dator
            // anges - om det väntande programmet körs där)

            // skapa ett lämpligt kommunikationsverktyg
            OutputStream    os = vantandePart.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
```

```
            InputStream    is = vantandePart.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));
            // eller:
            // java.util.Scanner  in = new java.util.Scanner (is);

            // data som ska skickas, och plats för de data
            // som ska tas emot

            // data som ska skickas till det väntande programmet
            String[]    u = {"noll", "ett", "två", "tre", "fyra"};

            // lagringsstruktur för de data som tas emot från det
            // väntande programmet
            String[]    v = new String[u.length];

            // kommunicera

            // kommunicera på ett i förväg bestämt sätt
            for (int i = 0; i < u.length; i++)
            {
                // skicka ett meddelande
                out.println (u[i]);

                // ta emot svaret
                v[i] = in.readLine ();
                // om ett objekt av typen java.util.Scanner
                // används som inmatningsverktyg,
                // ska metoden nextLine användas här

                // visa svaret
                System.out.print (v[i] + " ");
            }
            System.out.println ();
        }
        catch (IOException e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
            {
                vantandePart.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
```

```
}
```

Programmets utmatning (det här programmet startas efter det att programmet `VantandePart` startats)

```
ANSLUTANDE PART

zero one two three four
```

VantandePart.java

Ett program som väntar på ett annat program för att kommunicera med det

Ett Javaprogram kan vänta på ett annat program. När en begäran för anslutning kommer, skapar det väntande programmet en förbindelse till det anslutande programmet. Därefter kan de två programmen komunicera via denna förbindelse.

För att kommunikationen mellan två program ska bli möjlig, måste programmen respektera ett i förväg bestämt sätt att kommunicera (ett givet applikationsprotokoll).

```
import java.net.*;  // Socket, ServerSocket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
                    // InputStreamReader, BufferedReader,
                    // IOException

// VantandePart är ett program som väntar på
// ett anslutande program.
// När en begäran för anslutning kommer, skapar programmet
// en förbindelse till det anslutande programmet. Därefter
// kommunicerar de två programmen.
class VantandePart
{
    public static void main (String[] args)
    {
        System.out.println ("VÄNTANDE PART");
        System.out.println ();

        // ett väntande verktyg
        ServerSocket    ss = null;

        // den anslutande parten
        Socket    anslutandePart = null;

        try
```

```
{
    // upprätta förbindelse

    // ett objekt som väntar på en given port
    ss = new ServerSocket (1201);
    // portnummer större än 1024 ska väljas
    // för icke-standardtjänster

    // acceptera anslutningsbegäran
    anslutandePart = ss.accept ();

    // anslutningen skapad, stäng det väntande objektet
    // (frigör motsvarande resurser)
    ss.close ();

    // ett lämpligt kommunikationsverktyg

    // skapa ett lämpligt kommunikationsverktyg för att
    // kunna kommunicera med det anslutande programmet
    OutputStream   os = anslutandePart.getOutputStream ();
    PrintWriter    out = new PrintWriter (os, true);

    InputStream    is = anslutandePart.getInputStream ();
    BufferedReader    in = new BufferedReader (
                              newInputStreamReader (is));
    // eller:
    // java.util.Scanner  in = new java.util.Scanner (is);

    // data som ska skickas, och plats för de data
    // som ska tas emot

    // data som ska skickas till det anslutande programmet
    String[]  u = {"zero", "one", "two", "three", "four"};

    // lagringsstruktur för de data som tas emot från det
    // anslutande programmet
    String[]    v = new String[u.length];

    // kommunicera

    // kommunicera på ett i förväg bestämt sätt
    for (int i = 0; i < u.length; i++)
    {
        // ta emot meddelandet
        v[i] = in.readLine ();
        // om ett objekt av typen java.util.Scanner
        // används som inmatningsverktyg, ska
        // metoden nextLine användas här

        // visa meddelandet
        System.out.print (v[i] + " ");
```

```
                // vänta lite
                try
                {
                    Thread.sleep (4000);
                }
                catch (InterruptedException e)
                {}

                // skicka svaret
                out.println (u[i]);
            }
            System.out.println ();
        }
        catch (IOException e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
            {
                ss.close ();
                anslutandePart.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}
```

Programmets utmatning (det här programmet startas före programmet `AnslutandePart`)

```
VÄNTANDE PART

noll ett två tre fyra
```

Ett serverprogram

En tråd per klient

HeltalsNamn.java

En klass som definierar namn för vissa heltal

`HeltalsNamn` är en klass som definierar namn för heltalen mellan `0` (inklusive) och `10` (inklusive). Heltalens namn anges på två olika språk.

```
// HeltalsNamn definierar namn för heltalen mellan 0 och 10
class HeltalsNamn
{
    public static final String[]    namn = {
                "noll", "ett", "två", "tre", "fyra",
                "fem", "sex", "sju", "åtta", "nio", "tio"
                };

    public static final String[]    name = {
                "zero", "one", "two", "three", "four",
                "five", "six", "seven", "eight", "nine", "ten"
                };
}
```

NamnServer.java

Ett program som representerar en multitrådad server

Man kan skapa en multitrådad server, som skapar en särskild tråd för varje klient som ansluter sig. Huvudtråden lyssnar och skapar en förbindelse till den anslutande klienten. När en förbindelse har upprättats, skapas en särskild tråd som hanterar denna klient. Huvudtråden fortsätter lyssna efter andra klienter, medan den skapade tråden betjänar sin klient. Olika trådar betjänar olika klienter. På så sätt kan flera klienter betjänas samtidigt (en klient behöver inte vänta tills kommunikationen med en annan klient avslutas).

```
import java.net.*;  // ServerSocket, Socket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
                    // InputStreamReader, BufferedReader,
                    // IOException
```

```
// KlientHanterare tar emot ett heltal från klienten,
// och returnerar heltalets namn.
// Detta upprepas tills klienten skickar -1.
class KlientHanterare implements Runnable
{
    // klient som ska hanteras
    private Socket    klient;

    // KlientHanterare initierar klienten
    public KlientHanterare (Socket klient)
    {
        this.klient = klient;
    }

    public void run ()
    {
        try
        {
            // skapa ett lämpligt kommunikationsverktyg för att
            // kunna kommunicera med klienten
            OutputStream    os = klient.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
            InputStream    is = klient.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));

            // betjäna klienten

            // ta emot klientens heltal
            String    namn = "noll";
            int    heltal = Integer.parseInt (in.readLine ());
            while (heltal != -1)  // kontrollera heltalet
            {
                // bestäm motsvarande namn
                namn = HeltalsNamn.namn[heltal] + " " +
                       HeltalsNamn.name[heltal];

                // skicka namnet till klienten
                out.println (namn);

                // ta emot nästa heltal
                heltal = Integer.parseInt (in.readLine ());
            }
        }
        catch (IOException e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
```

```
            {
                klient.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}

// NamnServer lysnar efter klienter, skapar förbindelser
// till dessa och skapar och startar trådar som
// hanterar klienterna.
class NamnServer
{
    public static void main (String[] args)
    {
        ServerSocket    ss = null;

        try
        {
            // väntande objekt
            ss = new ServerSocket (1201);
            System.out.println (
               "namnservern startad, och lyssnar på"
             + " porten " + ss.getLocalPort () + "\n");

            while (true)
            {
                // lyssna efter en klient, och skapa
                // en förbindelse till den
                Socket    klient = ss.accept ();

                // skapa och starta en tråd som
                // hanterar kommunikation med denna klient
                Thread    t =
                    new Thread (new KlientHanterare (klient));
                t.start ();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace ();

            try
            {
                ss.close ();
            }
            catch (IOException ex)
            {
```

```
                ex.printStackTrace ();
            }
        }
    }
}
```

Programmets utmatning (det här programmet ska startas innan klienterna försöker ansluta sig)

```
namnservern startad, och lyssnar på porten 1201
```

NamnKlient.java

Ett program som ansluter sig till en namnserver och utnyttjar denna servers tjänst

Man kan ansluta sig till ett serverprogram (en server), och utnyttja detta programs tjänster.

```
import java.net.*;  // Socket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
            // InputStreamReader, BufferedReader, IOException

// NamnKlient är ett program som ansluter sig till en namnserver
// och skickar flera heltal till denna. För varje heltal som
// skickas, erhålls motsvarande namn.
class NamnKlient
{
    public static void main (String[] args)
    {
        System.out.println ("NAMNKLIENT\n");

        // servern
        Socket    server = null;

        try
        {
            // skapa en förbindelse till namnservern
            server = new Socket ("localhost", 1201);

            // skapa ett lämpligt kommunikationsverktyg
            OutputStream    os = server.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
            InputStream    is = server.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));

            // skapa flera heltal, skicka dem till namnservern,
```

```
            // och ta emot deras namn
            int    heltal = 0;
            String    namn = "noll";
            for (int i = 0; i < 5; i++)
            {
                // ett slumpmässigt heltal
                heltal = (int) (11 * Math.random ());

                // skicka heltalet till namnservern
                out.println (heltal);

                // ta emot heltalets namn från namnservern
                namn = in.readLine ();

                // visa heltalet och dess namn
                System.out.println (heltal + " " + namn);

                // vänta lite
                Thread.sleep (7000);
            }

            // avsluta kommunikationen
            out.println (-1);
            // motsvarande tråd på serversidan avslutas
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
            {
                server.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}
```

Exempel på programmets utmatning (det här programmet kan startas flera gånger - på så sätt skapas flera klienter som samtidigt kommunicerar med ett och samma server)

```
NAMNKLIENT

5 fem five
7 sju seven
```

```
6 sex six
0 noll zero
9 nio nine
```

En pool av trådar

HeltalsNamn.java

En klass som definierar namn för vissa heltal

`HeltalsNamn` är en klass som definierar namn på heltalen mellan `0` (inklusive) och `10` (inklusive). Heltalens namn anges på två olika språk.

```
// HeltalsNamn definierar namn för heltalen mellan 0 och 10
class HeltalsNamn
{
    public static final String[]    namn = {
                "noll", "ett", "två", "tre", "fyra",
                "fem", "sex", "sju", "åtta", "nio", "tio"
                };

    public static final String[]    name = {
                "zero", "one", "two", "three", "four",
                "five", "six", "seven", "eight", "nine", "ten"
                };
}
```

NamnServer1.java

Ett program som representerar en multitrådad server med en pool av trådar

Ett serverprogram, som har en pool av trådar, kan skapas. När en förbindelse med en klient upprättats, hämtas en ledig tråd från poolen och tilldelas en kommunikationsuppgift (i form av ett `Runnable`-objekt). Denna tråd utför sedan uppgiften (exekverar `run`-metoden i samband med det objekt som den fått). Därefter blir tråden åter ledig och kan utnyttjas för en annan uppgift.

```
import java.net.*;  // ServerSocket, Socket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
        // InputStreamReader, BufferedReader, IOException
import java.util.concurrent.*;// BlockingQueue, ArrayBlockingQueue
```

```
// KlientHanterare tar emot ett heltal från klienten, och skickar
// tillbaka heltalets namn. Detta upprepas tills klienten skickar
// heltalet -1.
class KlientHanterare implements Runnable
{
    // klienten som ska hanteras
    private Socket    klient;

    // KlientHanterare initierar klienten
    public KlientHanterare (Socket klient)
    {
        this.klient = klient;
    }

    public void run ()
    {
        try
        {
            // skapa ett lämpligt kommunikationsverktyg för att
            // kunna kommunicera med klienten
            OutputStream    os = klient.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
            InputStream    is = klient.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));

            // betjäna klienten

            // ta emot klientens heltal
            String    namn = "noll";
            int    heltal = Integer.parseInt (in.readLine ());
            while (heltal != -1) // kontrollera heltalet
            {
                // bestäm motsvarande namn
                namn = HeltalsNamn.namn[heltal] + " " +
                       HeltalsNamn.name[heltal];

                // skicka namnet till klienten
                out.println (namn);

                // ta emot nästa heltal
                heltal = Integer.parseInt (in.readLine ());
            }
        }
        catch (IOException e)
        {
            e.printStackTrace ();
        }
        finally
        {
```

```
            try
            {
                klient.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}

// ArbetarTrad är en tråd som väntar på en plats för lediga
// trådar. Tråden kan tas från platsen och tilldelas en uppgift i
// form av ett Runnable-objekt. Tråden utför då den kod som finns
// i objektets run-metod. Därefter placerar sig tråden åter på
// platsen för lediga trådar, och väntar där.
class ArbetarTrad extends Thread
{
    // en plats (kö) för lediga trådar
    // (tråden kan placera sig på denna plats och vänta
    // på arbetsuppgiften där)
    private BlockingQueue<ArbetarTrad>    ledigaTradar;

    // en plats (kö) för ett Runnable-objekt
    // (som representerar en uppgift som tråden behöver utföra)
    private BlockingQueue<Runnable>    uppgiftBehallare;

    // initiera platserna
    public ArbetarTrad (BlockingQueue<ArbetarTrad> ledigaTradar)
    {
        this.ledigaTradar = ledigaTradar;
        // gemensam resurs för alla lediga trådar

        uppgiftBehallare = new ArrayBlockingQueue<Runnable> (1);
        // trådens resurs - plats för en uppgift
        // (i form av ett Runnable-objekt)
    }

    public void run ()
    {
        while (true)
        {
            try
            {
                // placera sig på den plats där lediga trådar
                // väntar
                // (en annan tråd ska ta den här tråden från denna
                // plats, och tilldela den en uppgift)
                ledigaTradar.put (this);
                // tråden placeras som sista tråd i kön
```

```
                // ta den tilldelade uppgiften
                Runnable    uppgift = uppgiftBehallare.take ();
                // vänta (i metoden take) tills uppgiften lämnas
                // på en förbestämd plats
                // (i trådens uppgiftbehållare)

                // utför uppgiften
                uppgift.run ();
                // metoden run anropas direkt – det skapas inte
                // en ny tråd
                // (systemet belastas inte med nya trådar)
            }
            catch (InterruptedException e)
            {}
        }
    }

    // utfor placerar en given uppgift (representerad via ett
    // Runnable-objekt) i trådens behållare, så att uppgiften
    // sedan kan tas (och utföras) av tråden
    // (metoden representerar trådens mottagningfunktion)
    public void utfor (Runnable uppgift)
                            throws InterruptedException
    {
        uppgiftBehallare.put (uppgift);
    }
}

// NamnServer1 är ett serverprogram, som har en pool av trådar.
class NamnServer1
{
    public static void main (String[] args) throws Exception
    {
        // förbered en pool av trådar

        // en pool av trådar
        ArbetarTrad[]    pool = new ArbetarTrad[10];

        // plats för lediga trådar
        BlockingQueue<ArbetarTrad>    ledigaTradar =
            new ArrayBlockingQueue<ArbetarTrad> (pool.length);

        // skapa och starta trådarna i poolen
        for (int i = 0; i < pool.length; i++)
        {
            pool[i] = new ArbetarTrad (ledigaTradar);
            pool[i].start ();
        }

        // ett lyssnarobjekt
```

```
        // skapa ett objekt som lyssnar på en given port
        ServerSocket    ss = new ServerSocket (1201);
        System.out.println ("namnservern startad, och lyssnar på"
                            + " porten " + ss.getLocalPort ());
        System.out.println ();

        // lyssna efter klienterna och betjäna dem

        // skapa förbindelser till klienterna och kommunicera
        // med dem
        while (true)
        {
            // skapa förbindelse till en klient
            Socket    klient = ss.accept ();

            // skapa en klienthanterare
            Runnable    r = new KlientHanterare (klient);

            // ta den första lediga tråden
            ArbetarTrad    ledigTrad = ledigaTradar.take ();

            // skicka klienthanteraren till den tråden
            // (tråden ska utföra klienthanterarens run-metod)
            ledigTrad.utfor (r);
        }
    }
}
```

Programmets utmatning (det här programmet startas innan klienterna försöker ansluta sig)

```
namnservern startad, och lyssnar på porten 1201
```

NamnServer2.java

Ett program som representerar en multitrådad server med en pool av trådar

Ett serverprogram, som har en pool av trådar, kan skapas. När en förbindelse med en klient upprättas, hämtas en ledig tråd från poolen och tilldelas en kommunikationsuppgift (i form av ett `Runnable`-objekt). Denna tråd utför sedan uppgiften (kör `run`-metoden i samband med det objektet som den fått). Tråden blir därefter ledig igen, och kan utnyttjas för en annan uppgift.

```
import java.net.*;  // ServerSocket, Socket
```

```
import java.io.*;   // OutputStream, PrintWriter, InputStream,
            // InputStreamReader, BufferedReader, IOException
import java.util.concurrent.*; // Executors, ExecutorService

// KlientHanterare tar emot ett heltal från klienten, och skickar
// tillbaka detta heltals namn. Detta upprepas tills
// klienten skickar heltalet -1.
class KlientHanterare implements Runnable
{
    // klienten som ska hanteras
    private Socket    klient;

    // KlientHanterare initierar klienten
    public KlientHanterare (Socket klient)
    {
        this.klient = klient;
    }

    public void run ()
    {
        try
        {
            // skapa ett lämpligt kommunikationsverktyg, för att
            // kunna kommunicera med klienten
            OutputStream    os = klient.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
            InputStream    is = klient.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));

            // betjäna klienten

            // ta emot klientens heltal
            String    namn = "noll";
            int    heltal = Integer.parseInt (in.readLine ());
            while (heltal != -1) // kontrollera heltalet
            {
                // bestäm motsvarande namn
                namn = HeltalsNamn.namn[heltal] + " " +
                       HeltalsNamn.name[heltal];

                // skicka namnet till klienten
                out.println (namn);

                // ta emot nästa heltal
                heltal = Integer.parseInt (in.readLine ());
            }
        }
        catch (IOException e)
        {
            e.printStackTrace ();
```

```
        }
        finally
        {
            try
            {
                klient.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}

// NamnServer2 är ett serverprogram, som har en pool av trådar.
class NamnServer2
{
    public static void main (String[] args) throws Exception
    {
        // en pool av trådar
        ExecutorService  pool = Executors.newFixedThreadPool (10);
        // pool är en referens av gränssnittet
        // ExecutorService, som refererar till ett
        // objekt av typen ThreadPoolExecutor
        // (i paketet java.util.concurrent).
        // Poolen har 10 arbetstrådar.

        // ett lyssnarobjekt

        // skapa ett objekt som lyssnar på en given port
        ServerSocket    ss = new ServerSocket (1201);
        System.out.println ("namnservern startad, och lyssnar på"
                        + " porten " + ss.getLocalPort () + "\n");

        // lyssna efter klienterna och betjäna dem

        // skapa förbindelser till klienterna,
        // och kommunicera med dem
        while (true)
        {
            // skapa förbindelse till en klient
            Socket    klient = ss.accept ();

            // skapa en klienthanterare
            Runnable    r = new KlientHanterare (klient);

            // skicka klienthanteraren till poolen
            // (en ledig tråd i poolen ska utföra
            // klienthanterarens run-metod)
            pool.submit (r);
```

```
        }
    }
}
```

Programmets utmatning (det här programmet startas innan klienterna försöker ansluta sig)

```
namnservern startad, och lyssnar på porten 1201
```

NamnKlient.java

Ett program som ansluter sig till en namnserver och utnyttjar denna servers tjänst

Man kan ansluta sig till ett serverprogram (en server), och utnyttja detta programs tjänster.

```
import java.net.*;  // Socket
import java.io.*;   // OutputStream, PrintWriter, InputStream,
                    // InputStreamReader, BufferedReader,
                    // IOException

// NamnKlient är ett program som ansluter sig till en namnserver,
// och skickar flera heltal till denna server.
// För varje heltal som skickas, får klienten motsvarande namn.
class NamnKlient
{
    public static void main (String[] args)
    {
        System.out.println ("NAMNKLIENT");
        System.out.println ();

        // servern
        Socket    server = null;
        try
        {
            // skapa en förbindelse till namnservern
            server = new Socket ("localhost", 1201);

            // skapa ett lämpligt kommunikationsverktyg
            OutputStream    os = server.getOutputStream ();
            PrintWriter    out = new PrintWriter (os, true);
            InputStream    is = server.getInputStream ();
            BufferedReader    in = new BufferedReader (
                                    new InputStreamReader (is));

            // skapa flera heltal, skicka dem till namnservern,
```

```
            // och ta emot deras namn
            int    heltal = 0;
            String    namn = "noll";
            for (int i = 0; i < 5; i++)
            {
                // ett slumpmässigt heltal
                heltal = (int) (11 * Math.random ());

                // skicka heltalet till namnservern
                out.println (heltal);

                // ta emot heltalets namn från namnservern
                namn = in.readLine ();

                // visa heltalet och dess namn
                System.out.println (heltal + " " + namn);

                // vänta lite
                Thread.sleep (7000);
            }

            // avsluta kommunikationen
            out.println (-1);
            // motsvarande tråd på serversidan avslutas
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
            {
                server.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}
```

Exempel på programmets utmatning (det här programmet kan startas flera gånger - på så sätt skapas flera klienter som samtidigt kommunicerar med ett och samma serverprogram)

```
NAMNKLIENT

5 fem five
7 sju seven
```

```
6 sex six
0 noll zero
9 nio nine
```

Kommunikation via en server

IntermediaryServer.java

En server som kan användas för kommunikation mellan olika klienter

En server kan ta emot ett meddelande från en klient, och skicka meddelandet till alla andra klienter som är anslutna. Flera klienter kan ansluta sig till en sådan server och kommunicera med varandra via denna server.

```
import java.net.*;   // ServerSocket, Socket
import java.io.*;    // ObjectOutputStream, ObjectInputStream,
                     // IOException
import java.util.*;  // ArrayList

// KlientHanterare definierar en tråd, som tar emot ett meddelande
// från en klient, och skickar det till alla andra klienter
// som är anslutna till servern
class KlientHanterare implements Runnable
{
    // objektströmmar till alla anslutna klienter
    // (en statisk resurs, tillgänglig till alla klienthanterare)
    private static ArrayList<ObjectOutputStream>    outStreams =
                            new ArrayList<ObjectOutputStream> ();

    // klient som ska hanteras
    private Socket    klient;

    // KlientHanterare initierar klienten
    public KlientHanterare (Socket klient)
    {
        this.klient = klient;
    }

    public void run ()
    {
        // strömmar
        ObjectInputStream     in  = null;
        ObjectOutputStream    out = null;

        try
        {
```

```
    // skapa en lämplig inström
    in = new ObjectInputStream (klient.getInputStream ());

    // skapa en lämplig utström och placera den i vektorn
    // för utströmmar
    out =
       new ObjectOutputStream (klient.getOutputStream ());
    outStreams.add (out);

    // kommunicera

    // ta emot meddelandet från klienten och skicka
    // det till alla andra klienter
    Object    obj = null;
    int    size  = 0;
    int    index = 0;
    ObjectOutputStream    oos = null;
    while (true)
    {
        // ta emot klientens meddelande
        obj = in.readObject ();

        // skicka meddelandet till alla andra klienter
        synchronized (outStreams)
        // vektorn ska inte ändras medan operationen pågår
        {
            // stega genom vektorn med strömmar
            size = outStreams.size ();
            for (index = 0; index < size; index++)
            {
                try
                {
                    oos = outStreams.get (index);
                    if (oos != out)
                    // skicka inte meddelandet till
                    // sig själv
                        oos.writeObject (obj);
                        // skicka meddelandet
                        // längs strömmen
                }
                catch (IOException e)
                {}
            }
        }
    }
}
catch (Exception e)
{
    // om det inte går att kommunicera,
    // ta bort klienten från vektorn
    // (meddelanden ska inte skickas till den)
```

```
            outStreams.remove (out);

            try
            {
                klient.close ();
            }
            catch (IOException ex)
            {}
        }
    }
}

// IntermediaryServer lyssnar efter klienterna,
// skapar förbindelser till dem, och skapar och startar
// trådar som hanterar klienterna.
class IntermediaryServer
{
    public static void main (String[] args)
    {
        // ett objekt som lyssnar efter klienterna
        ServerSocket    ss = null;

        try
        {
            ss = new ServerSocket (1201);
            System.out.println("intermediary-servern startad, och"
                    + " lyssnar på porten " + ss.getLocalPort ());
            System.out.println ();

            while (true)
            {
                // skapa förbindelse till en klient
                Socket    klient = ss.accept ();
                System.out.println ("klienten "
                                + klient.getInetAddress ()
                                + " accepterad");

                // skapa och starta en tråd som hanterar klienten
                Thread    t =
                        new Thread (new KlientHanterare (klient));
                t.start ();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace ();

            try
            {
                ss.close ();
            }
```

```
            catch (IOException ex)
            {
                ex.printStackTrace ();
            }
        }
    }
}
```

Exempel på programmets utmatning (det här programmet ska startas innan klienterna försöker ansluta sig)

```
intermediary-servern startad, och lyssnar på porten 1201

klienten 127.0.0.1/127.0.0.1 accepterad
klienten 127.0.0.1/127.0.0.1 accepterad
klienten 127.0.0.1/127.0.0.1 accepterad
```

Klient.java

En klient som kommunicerar med andra klienter via en gemensam server

Flera klienter kan kommunicera med varandra via en gemensam server. Denna server tar emot ett meddelande från en klient och skickar det till alla andra klienter.

```
import java.net.*;   // Socket
import java.io.*;    // ObjectOutputStream, ObjectInputStream,
                     // IOException
import java.util.*;  // Scanner

// Mottagare definierar en tråd som tar emot meddelanden från en
// server och visar dessa meddelanden
class Mottagare implements Runnable
{
    // en inström
    private ObjectInputStream    in;

    // Mottagare initierar inströmmen
    public Mottagare (ObjectInputStream in)
    {
        this.in = in;
    }

    public void run ()
    {
        // ett meddelande
```

```
        Object    obj = null;

        try
        {
            while (true)
            {
                // ta emot meddelandet, och visa det
                obj = in.readObject ();
                System.out.println (obj);
            }
        }
        catch (Exception e)
        {}
    }
}

// Klient kommunicerar med andra klienter via en gemensam server
class Klient
{
    public static void main (String[] args)
    {
        System.out.println ("KLIENT\n");

        // server
        Socket    server = null;

        try
        {
            // förbindelse

            // skapa en förbindelse till en server
            server = new Socket ("localhost", 1201);

            // kommunikationsverktyg

            // skapa ett lämpligt kommunikationsverktyg
            ObjectOutputStream out = new ObjectOutputStream (
                                        server.getOutputStream ());
            ObjectInputStream    in = new ObjectInputStream (
                                         server.getInputStream ());
            // skapa ett lämpligt verktyg för standardinmatning
            Scanner    sin = new Scanner (System.in);

            // visa de meddelanden som kommer från olika klienter
            // (via servern)

            // skapa och starta en tråd som tar emot
            // de meddelanden som kommer från olika klienter
            // (via servern), och visar dessa meddelanden
            new Thread (new Mottagare (in)).start ();
            // den här operationen måste utföras i
```

```
            // en särskild tråd
            // - meddelandet kan komma när som helst

            // skicka meddelanden till olika klienter
            // (via servern)

            // Mata in meddelanden från standardinmatningsenheten
            // och skicka dessa meddelanden till servern (och via
            // denna server till alla andra klienter som är
            // anslutna till servern).
            String    med = sin.nextLine ();
            while (!med.equals (""))
            {
                // skicka meddelandet
                out.writeObject (med);

                // mata in nästa meddelande
                med = sin.nextLine ();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
        finally
        {
            try
            {
                server.close ();
            }
            catch (IOException e)
            {
                e.printStackTrace ();
            }
        }
    }
}
```

Exempel på programmets inmatning och utmatning i en klient (det här programmet kan startas flera gånger - på så sätt skapas flera klienter som kan kommunicera med varandra)

```
KLIENT

ett  // input, skickas till andra klienter
två  // output, kommer från en annan klient
tre  // input, skickas till andra klienter
fyra // output, kommer från en annan klient
fem  // output, kommer från en annan klient
```

Kapitel 3
Fönster

En ram
Skapa och hantera en ram
Definiera en egen ram
En allmän ram
Ett fönster utan dekorationer

Dialoger
En dialog
Standarddialoger
Genvägar till standarddialoger
En fildialog
En färgdialog

En ram

Skapa och hantera en ram

SkapaVisaEnRam.java

Ett program som skapar en ram och visar den

En ram (frame) är ett toppnivåfönster (ett fönster som inte kan placeras i ett annat fönster), som kan innehålla olika grafiska komponenter.

En ram har kanter, en yta där olika grafiska komponenter kan placeras, en titelrad, tre knappar (för att kunna minimera, maximera och stänga ramen) och en ikon (en meny kan öppnas via ikonen).

En ram kan manipuleras manuellt via dess kanter och hörn, och via dess knappar, meny och titelrad.

En ram döljs när den stängs (förvalt beteende). Den förstörs inte, och motsvarande program avslutas inte.

```
import javax.swing.*;    // JFrame

class SkapaVisaEnRam
{
    public static void main (String[] args)
    {
        // en ram
        JFrame    frame = new JFrame ();

        // ramens dimensioner
        frame.setSize (600, 400);
        // (förvalda dimensioner är 0 x 0 pixlar)

        // visa ramen
        frame.setVisible (true);

        // main-tråden avslutas, men programmet fortlever
        // i en särskild tråd som hanterar GUI (Graphic User
        // Interface, grafiskt användargränssnitt)

        // avsluta programmet utifrån, på ett systemberoende sätt
        // (tryck Ctrl+C i Unix, Ctrl+Alt+Delete i Windows)
    }
}
```

Programmets utmatning

En ram (ett fönster) i vänstra övre hörnet av skärmen visas.

AvslutaProgrammetViaEnRam.java

Ett program som avslutas när en ram stängs

När en ram stängs blir den osynlig, men den förstörs inte. Ramen fortsätter existerra och ta upp systemresurser. Detta förvalda beteende kan ändras genom att en av följande konstanter (från klassen `JFrame`) används:
`DO_NOTHING_ON_CLOSE` (gör ingenting)
`HIDE_ON_CLOSE` (dölj ramen, förvalt beteende)
`DISPOSE_ON_CLOSE` (förstör ramen)
`EXIT_ON_CLOSE` (förstör ramen, avsluta programmet)

Om konstanten `EXIT_ON_CLOSE` väljs för en ram, avslutas programmet när ramen stängs.

```
import javax.swing.*;    // JFrame

class AvslutaProgrammetViaEnRam
{
    public static void main (String args[])
    {
        // en ram med en given titel
        JFrame    frame = new JFrame (" En ram");

        // sätt ramens storlek
        frame.setSize (600, 400);
```

```
        // vad ska hända när ramen stängs
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ram med titel visas.

När ramen stängs, avslutas programmet.

EnRamsIkon.java

Ett program som skapar en ram med en vald ikon

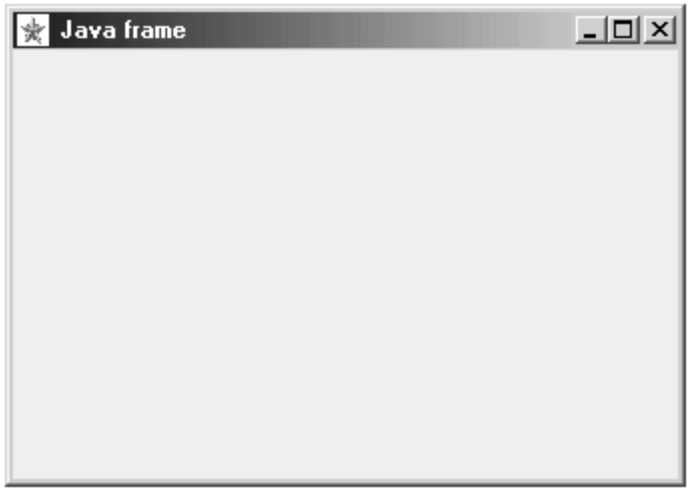
En ram i Java har en förvald ikon (Javas kaffekopp). I Windows visas denna ikon längst upp till vänster i ramen (en meny kan öppnas via den). Istället för den förvalda ikonen, kan man använda en egen ikon. En ikon kan skapas utifrån en fil som innehåller en bild.

```
import java.awt.*;       // Toolkit, Image
import javax.swing.*;    // JFrame
import java.io.*;        // File

class EnRamsIkon
{
    public static void main (String[] args)
    {
        // en ram
        JFrame    frame = new JFrame (" Java frame");
        frame.setSize (600, 400);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en fil som innehåller en bild
        String    fil = System.getProperty ("java.home")
                        + File.separatorChar +"classes"
                        + File.separatorChar +"fjava"
                        + File.separatorChar +"edu"
                        + File.separatorChar + "TheStar.gif";

        // sätt ramens ikon till den angivna bilden
        Toolkit    tk = Toolkit.getDefaultToolkit ();
        Image    bild = tk.getImage (fil);
        frame.setIconImage (bild);
```

```
        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ram med en vald ikon visas.



EnRamsDimensionerOchPlacering.java

Ett program som skapar en ram, vars storlek och placering anges relativt den aktuella skärmen

Man kan ange en rams dimensioner (de förvalda dimensionerna är 0 x 0 pixlar) och placering på skärmen (den förvalda placeringen är i det övre vänstra hörnet). Både ramens dimensioner och placering (placering av ramens övre vänstra hörn) anges i antalet pixlar. Olika skärmar har olika upplösning (totala antalet pixlar). En ram med dimensioner 600 x 400 (pixlar) kan verka stor på en skärm, och liten på en annan skärm. Därför är det bättre att precisera en rams storlek och placering relativt den aktuella skärmen.

```
import java.awt.*;              // Toolkit, Dimension
import javax.swing.*;           // JFrame
```

```
class EnRamsDimensionerOchPlacering
{
    public static void main (String[] args)
    {
        // en ram
        JFrame    frame = new JFrame (" Java frame");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // skärmens dimensioner i antalet pixlar
        Toolkit     tk = Toolkit.getDefaultToolkit ();
        Dimension    d = tk.getScreenSize ();
        // width och height - publika instansvariabler
        int    skarmBredd = d.width;   // skärmens bredd
        int    skarmHojd  = d.height;  // skärmens höjd

        // ange ramens dimensioner relativt skärmen
        int    frameBredd = 6 * skarmBredd / 10;
        // 60% av skärmens bredd
        int    frameHojd  = 4 * skarmHojd  / 10;
        // 40% av skärmens höjd
        frame.setSize (frameBredd, frameHojd);

        // ange ramens placering (placering av övre vänstra
        // hörnet) relativt skärmen, så att ramen hamnar i
        // mitten av skärmen
        int    xHorn = (skarmBredd - frameBredd) / 2;
        int    yHorn = (skarmHojd  - frameHojd ) / 2;
        frame.setLocation (xHorn, yHorn);

        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ram i mitten av skärmen visas. Ramens dimensioner är 60 % av skärmens bredd och 40 % av skärmens höjd (oavsett på vilken skärm den visas).

EnRamSomBehallare.java

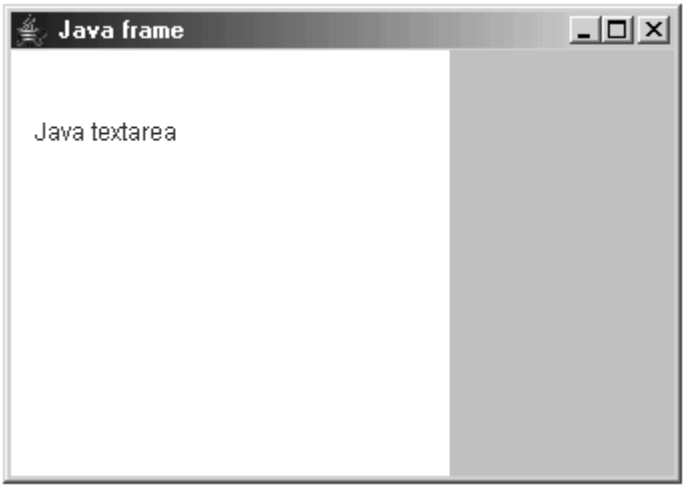
Ett program som skapar en ram, och placerar andra komponenter i denna ram

En ram är en behållare. Olika grafiska komponenter kan placeras i en ram. Det går inte att placera en ram i en annan ram (en ram är en toppnivåbehållare).

```
import java.awt.*;       // Color
import javax.swing.*;    // JFrame, JPanel, JTextArea

class EnRamSomBehallare
{
    public static void main (String[] args)
    {
        // en ram
        JFrame    frame = new JFrame (" Java frame");
        frame.setSize (600, 400);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // andra grafiska komponenter

        // en panel
        JPanel    panel = new JPanel ();
        // ange panelens bakgrundsfärg
        panel.setBackground (Color.GRAY);

        // en textarea
        JTextArea    textArea = new JTextArea (30, 20);
        // placera en text i textarean
        textArea.append ("\n\n  Java textarea");

        // placera komponenterna i ramen

        // placera panelen i mitten
        frame.add (panel, "Center");
        // placera textarean till vänster
        frame.add (textArea, "West");

        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ram visas. Ramen innehåller en textarea (med en viss text) till vänster, och en grå panel i resten av den yta där olika komponenter placeras.



Definiera en egen ram

EnEgenRam.java

En klass som definierar en typ av ramar, och ett program som använder en ram av den typen

En egen typ av ramar kan definieras. En subklass till klassen `JFrame` skapas, och i den subklassen definieras typen. Subklassen definierar hur en ram (av denna klass) ser ut, och hur den beter sig.

```
import javax.swing.*;    // JFrame

// SpecialFrame definierar en typ av ramar
class SpecialFrame extends JFrame
{
    public SpecialFrame (String titel)
    {
        super (titel);

        this.setSize (600, 400);
        this.setLocation (100, 100);
        this.setDefaultCloseOperation (EXIT_ON_CLOSE);
    }
}
```

```
// EnEgenRam skapar och visar en ram av klassen SpecialFrame
class EnEgenRam
{
    public static void main (String[] args)
    {
        // en ram av en egen typ
        SpecialFrame  frame = new SpecialFrame (" En egen frame");

        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ram med titel visas.

En allmän ram

RFrame.java

En klass som definierar en ram, vars dimensioner och placering kan anges relativt skärmen

NAMN

`RFrame` -- en klass som definierar en typ av ram

SYNOPSIS

Klassen `RFrame` representerar en ram vars dimensioner och placering kan anges relativt skärmen.

`public class RFrame extends javax.swing.JFrame.`

FILER OCH PAKET

Klassen `RFrame` är definierad och implementerad i filen `RFrame.java`.

Klassen `RFrame` finns i paketet `fjava.edu`.

KONSTRUKTORER

Man kan skapa en ram av typen `RFrame` med titel (en argumentsträng), eller utan titel (utan argument). En sådan ram har följande egenskaper:
ramen har en stjärna som ikon
ramens bredd är `80` % av skärmens bredd

ramens höjd är 70 % av skärmens höjd
ramen ligger i mitten av skärmen
ramen förstörs och programmet avslutas vid stängningen

EGNA METODER

`public void setSize (double relativeWidth, double relativeHeight)`

sätter ramens dimensioner relativt skärmen, som andel av skärmens bredd och höjd.

`public void setLocation (double relativeX, double relativeY)`

sätter ramens placering relativt skärmen - koordinaterna för ramens övre vänstra hörn anges som andel av skärmens bredd och höjd

ÄRVDA METODER och KONSTANTER

Klassen `RFrame` ärver metoder och konstanter från följande klasser:

`java.lang.Object`, `java.awt.Component`, `java.awt.Container`, `java.awt.Window`, `java.awt.Frame`, `javax.swing.JFrame`.

Se motsvarande dokumentation.

FÖRFATTARE

Fadil Galjic <fadil@kth.se>

VERSION

1.0

```
// package fjava.edu;     // utbildningspaketet

import java.awt.*;     // Toolkit, Dimension, Container, Image
import javax.swing.*;  // JFrame
import java.io.*;      // File

public class RFrame extends JFrame
{
    // RFrame skapar en förvald ram
    public RFrame ()
    {
        super ();

        initiateFrame ();
    }

    // RFrame skapar en förvald ram med en given titel
    public RFrame (String title)
    {
```

```
    super (title);

    initiateFrame ();
}

// setSize bestämmer ramens dimensioner relativt skärmen,
// som andel av skärmens bredd och höjd
public void setSize (double relativeWidth,
                     double relativeHeight)
// relativeWidth - ramens bredd som andel av skärmens bredd
// relativeHeight - ramens höjd som andel av skärmens höjd
{
    // bestäm skärmens dimensioner i pixlar
    Toolkit    tk = Toolkit.getDefaultToolkit ();
    Dimension    d = tk.getScreenSize ();
    int    screenWidth   = d.width;
    int    screenHeight  = d.height;

    // bestäm ramens dimensioner i pixlar
    int  frameWidth   = (int) (relativeWidth * screenWidth);
    int  frameHeight  = (int) (relativeHeight * screenHeight);

    // ange ramens dimensioner
    super.setSize (frameWidth, frameHeight);
}

// setLocation anger ramens placering relativt skärmen -
// koordinater för ramens övre vänstra hörn anges som andel
// av skärmens bredd och höjd
public void setLocation (double relativeX, double relativeY)
// relativeX - x-koordinat för övre vänstra hörnet som andel
// av skärmens bredd
// relativeY - y-koordinat för övre vänstra hörnet som andel
// av skärmens höjd
{
    // bestäm skärmens dimensioner i pixlar
    Toolkit    tk = Toolkit.getDefaultToolkit ();
    Dimension    d = tk.getScreenSize ();
    int    screenWidth = d.width;
    int    screenHeight  = d.height;

    // bestäm placering av ramens övre vänstra hörn i pixlar
    int    x = (int) (relativeX * screenWidth);
    int    y = (int) (relativeY * screenHeight);

    // ange ramens placering
    super.setLocation (x, y);
}

// en hjälpmetod
```

```
    // initiateFrame anger en rams ikon och förvalda
    // dimensioner och placering, och preciserar
    // dess förvalda beteende vid stängningen
    private void initiateFrame ()
    {
        // ange ramens ikon
        Toolkit    tk = Toolkit.getDefaultToolkit ();
        String    filNamn = System.getProperty ("java.home")
                          + File.separatorChar +"classes"
                          + File.separatorChar +"fjava"
                          + File.separatorChar +"edu"
                          + File.separatorChar +
                          "TheStar.gif";
        Image    image = tk.getImage (filNamn);
        this.setIconImage (image);

        // ange relativa dimensioner
        this.setSize (0.8, 0.7);

        // bestäm relativ placering
        this.setLocation (0.1, 0.15);

        // avsluta programmet vid stängningen
        this.setDefaultCloseOperation (EXIT_ON_CLOSE);
    }
}
```

AnvandRFrame.java

Ett program som skapar två ramar av typen RFrame

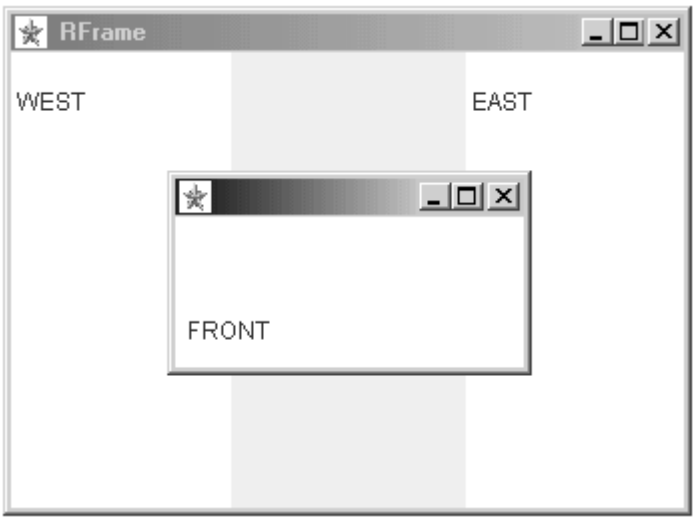
Olika ramar av typen `RFrame` kan skapas och användas. Dimensioner och placering av en sådan ram kan anges som andel av skärmens dimensioner.

```
import java.awt.*;     // JTextArea, Color
import javax.swing.*;  // JPanel

// import fjava.edu.*;    // RFrame
// Klassen RFrame ska göras tillgänglig på något sätt

class AnvandRFrame
{
    public static void main (String[] args)
    {
        // skapa två ramar av typen RFrame
        RFrame    f1 = new RFrame (" RFrame");
        RFrame    f2 = new RFrame ();
```

```
        // skapa två textareor och placera dem i den första ramen
        JTextArea    ta1 = new JTextArea (20, 10);
        ta1.append ("\n WEST");
        f1.add (ta1, "West");

        JTextArea    ta2 = new JTextArea (20, 10);
        ta2.append ("\n EAST");
        f1.add (ta2, "East");

        // bestäm relativa dimensioner och relativ placering
        // för den andra ramen
        f2.setSize (0.4, 0.3);
        f2.setLocation (0.3, 0.35);

        // ändra det förvalda beteendet vid stängningen
        // för den andra ramen
        f2.setDefaultCloseOperation (RFrame.DISPOSE_ON_CLOSE);

        // skapa en textarea och placera den i den andra ramen
        JTextArea    ta = new JTextArea (20, 10);
        ta.append ("\n\n\n  FRONT");
        f2.add (ta);

        // visa ramarna så att den andra ramen hamnar längst fram
        // på skärmen
        f1.setVisible (true);
        // f1.toBack ();
        f2.setVisible (true);
        f2.toFront ();
        // Metodena toBack och toFront anropas efter att metoden
        // setVisible anropats med true som argument.
    }
}
```

Programmets utmatning

En större ram visas i mitten av skärmen, och en mindre ram över den första ramen. Den större ramen innehåller en textarea i sin vänstra del och en textarea i sin högra del. Den mindre ramen innehåller bara en textarea. Programmet avslutas via den större ramen.



Ett fönster utan dekorationer

EttFonsterUtanDekorationer.java

Ett program som hanterar fönster utan ram, titeltrad, ikon och knappar

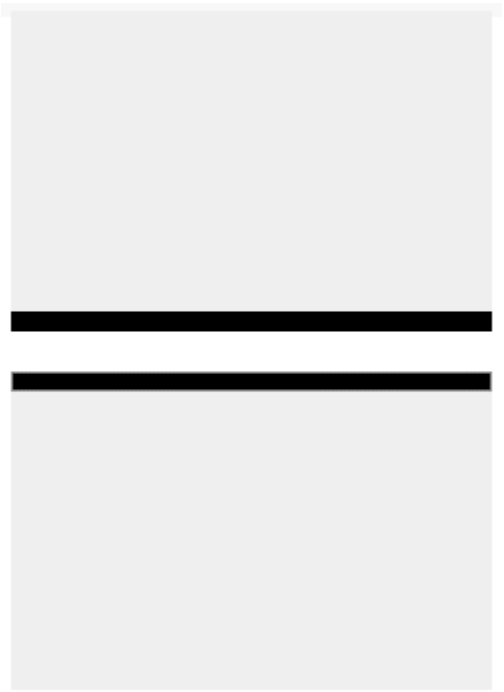
Ett fönster utan ram, titelrad, ikon och knappar kan skapas. Ett sådant fönster kan användas som behållare för andra komponenter.

```
import java.awt.*;       // Color
import javax.swing.*;    // JFrame, JPanel, JWindow

class EttFonsterUtanDekorationer
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame ();

        // fönstret ska inte ha några dekorationer
        frame.setUndecorated (true);

        // bestäm fönstrets storlek och position på skärmen
        frame.setSize (240, 160);
        frame.setLocation (100, 80);
```

```
        // en panel
        JPanel    panel = new JPanel ();
        panel.setBackground (Color.BLACK);

        // lägg till panelen i fönstret
        frame.add (panel, "South");

        // visa fönstret
        frame.setVisible (true);


        // ett fönster utan ram, titelrad, ikon och knappar
        JWindow    window = new JWindow ();
        window.setSize (240, 160);
        window.setLocation (100, 260);

        // en knapp
        JButton    knapp = new JButton ();
        knapp.setBackground (Color.BLACK);

        // placera knappen i fönstret
        window.add (knapp, "North");

        // visa fönstret
        window.setVisible (true);

        // avsluta programmet utifrån
    }
}
```

Programmets utmatning

Två fönster utan ram, titelrad, ikon och knappar visas. Det övre fönstret innehåller en panel i sin nedre del. Det nedre fönstret innehåller en knapp i sin övre del.



Dialoger

En dialog

EnDialog.java

Ett program som visar modala och icke-modala dialoger

Ett Javaprogram kan skapa och visa en dialog (ett dialogfönster). En dialog används för en kort kommunikation med användaren (visa ett meddelande, välja ett alternativ, mata in vissa uppgifter).

En dialog kan vara modal eller icke-modal. När en modal dialog visas, blockeras programmet och väntar på användaren. Exekveringen kan fortsätta först när dialogen stängts.

```
import javax.swing.*;  // JDialog

class EnDialog
{
    public static void main (String[] args)
    {
        // en icke-modal dialog
        JDialog    dialog1 = new JDialog ();
        dialog1.setTitle (" Not modal dialog");  // titel
        dialog1.setModal (false);  // en icke-modal dialog
        dialog1.setSize (200, 150);
        dialog1.setLocation (100, 100);

        // en modal dialog
        JDialog    dialog2 = new JDialog ();
        dialog2.setTitle (" Modal dialog");
        dialog2.setModal (true);  // en modal dialog
        dialog2.setSize (200, 150);
        dialog2.setLocation (100, 270);

        // en modal dialog
        JDialog    dialog3 = new JDialog ();
        dialog3.setTitle (" Modal dialog");
        dialog3.setModal (true);
        dialog3.setSize (200, 150);
        dialog3.setLocation (100, 300);

        // visa den första dialogen - programmet kan fortsätta
        // (dialogen är icke-modal)
        dialog1.setVisible (true);
        // visa den andra dialogen - programmet kan inte
```

```
        // fortsätta förrän användaren stänger dialogen
        // (dialogen är modal)
        dialog2.setVisible (true);
        // visa den tredje dialogen – programmet
        // kan inte fortsätta förrän
        // användaren stänger dialogen (modal dialog)
        dialog3.setVisible (true);

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

Två dialoger visas först. Programmet kan fortsätta först när den nedre dialogen har stängts. När exekveringen fortsätter, visas en dialog till på skärmen. Programmet kan inte fortsätta förrän denna dialog stängs.

EnEgenDialog.java

En klass som definierar en egen typ av dialoger, och ett program som använder en dialog av denna klass.

En egen typ av dialoger kan definieras. Detta görs genom att en subklass till klassen `JDialog` skapas. I denna subklass anpassas en dialogs utseende och beteende till konkreta behov.

```
import javax.swing.*;    // JDialog

// InputDialog definierar en egen typ av dialoger
class InputDialog extends JDialog
{
    public InputDialog (String meddelande)
    {
        // dialogens rubrik
        this.setTitle (" Input dialog");

        // dialogen ska vara en modal dialog
        this.setModal (true);

        // dialogens storlek
        this.setSize (200, 150);

        // dialogens placering
        this.setLocation (100, 80);

        // dialogen förstörs vid stängningen
        this.setDefaultCloseOperation (DISPOSE_ON_CLOSE);

        // dialogens innehåll
        JLabel label = new JLabel (meddelande);  // ett meddelande
        JTextField  tf = new JTextField (20);    // ett textfält

        // placera komponenterna i dialogen
        this.add (label, "Center");
        this.add (tf, "South");
    }

    // klassen är inte fullständig, programmet kan inte läsa
    // det som användaren skriver i textfältet
}

// EnEgenDialog skapar och visar en dialog av klassen InputDialog
class EnEgenDialog
{
    public static void main (String[] args)
    {
```

```
        // skapa en dialog
        InputDialog   dialog = new InputDialog ("    Födelseår:");

        // visa dialogen
        dialog.setVisible (true);

        // här ska programmet läsa det som användaren skriver,
        // och använda det på något sätt

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En dialog med titel visas. Dialogen innehåller ett textmeddelande och ett textfält.

Standarddialoger

EnStandardDialogruta.java

Ett program som skapar och använder en standarddialogruta

Man kan skapa och utforma en standarddialogruta, och visa denna dialogruta via ett dialogfönster.

```
import javax.swing.*;  // JFrame, JOptionPane, JDialog

class EnStandardDialogruta
{
    public static void main (String[] args)
    {
```

```
        // en ram
        JFrame    frame = new JFrame ("En frame");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (200, 200);
        frame.setVisible (true);

        // en dialogruta, som innehåller ett meddelande
        JOptionPane    ruta =
                      new JOptionPane ("Förbindelse upprättad!");

        // skapa en dialog, som ska innehålla dialogrutan som sin
        // enda komponent
        JDialog    dialog = ruta.createDialog (
             frame,  // förälderkomponent
                     // - dialogen hamnar ovanpå den
             "En dialog");  // titel

        // man ska inte göra så här:
        // JDialog    dialog = new JDialog ();
        // dialog.setTitle (" En dialog");
        // dialog.add (ruta);
        // om man gör så här, stängs inte dialogen
        // när användaren klickar på OK-knappen
        // i dialogrutan - dialogfönstret och dialogrutan
        // är inte integrerade i en enhet

        // bestäm dialogens beteende
        dialog.setModal (true);

        // visa dialogen
        dialog.setVisible (true);
    }
}
```

Programmets utmatning

En ram och en dialog (i mitten av ramen, ovanpå den) visas. Dialogen innehåller en standarddialogruta, med ett meddelande och en `OK`-knapp. Dialogen stängs antingen via dess stängningsknapp, eller via `OK`-knappen i dialogrutan.

Först när dialogen stängts, kan programmet fortsätta. Programmet avslutas när ramen stängs.

EnMessageDialog.java

Ett program som skapar och visar en standarddialog, som innehåller ett meddelande

En standarddialog kan innehålla ett meddelande. Meddelanden av olika typer kan visas: ett vanligt meddelande, ett informationsmeddelande, ett varningsmeddelande, ett felmeddelande, eller en fråga.

```
import javax.swing.*;  // JOptionPane, JDialog

class EnMessageDialog
{
    public static void main (String[] args)
    {
        // en dialogruta som innehåller ett meddelande
        JOptionPane    ruta =
                   new JOptionPane ("Förbindelse upprättad!");

        // bestäm meddelandets typ
        ruta.setMessageType (JOptionPane.INFORMATION_MESSAGE);
        // En av följande meddelandetyp kan väljas:
        // PLAIN_MESSAGE, INFORMATION_MESSAGE, QUESTION_MESSAGE,
        // WARNING_MESSAGE eller ERROR_MESSAGE.
        // En ikon visas inuti dialogrutan, och denna ikon beror
        // på meddelandets typ. Det förvalda valet är
        // PLAIN_MESSAGE, och i detta fall visas ingen ikon.
```

```
        // Man kan också ange en egen ikon.
        // En ikon kan skapas utifrån en fil som
        // innehåller en bild:
        // java.awt.ImageIcon ikon = new java.awt.ImageIcon (fil);
        // ruta.setIcon (ikon);

        // skapa en dialog utifrån dialogrutan
        JDialog    dialog = ruta.createDialog (null,
                                               "En Message Dialog");
        dialog.setModal (true);
        dialog.setSize (200, 150);
        dialog.setLocation (50, 100);

        // visa dialogen
        dialog.setVisible (true);

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En standarddialog visas. Dialogen innehåller ett meddelande, en ikon (i form av bokstaven i) och en OK-knapp.

EnConfirmDialog.java

Ett program som skapar och visar en standarddialog, som innehåller en standarduppsättning alternativ

En standarddialog kan innehålla ett meddelande och en standarduppsättning alternativ. Användaren kan välja ett av dessa alternativ eller stänga dialogen utan att välja någonting.

```
import javax.swing.*;  // JOptionPane, JDialog

class EnConfirmDialog
{
    public static void main (String[] args)
    {
        // en dialogruta, som innehåller ett meddelande
        JOptionPane    ruta =
                    new JOptionPane ("Vill du fortsätta?");
        ruta.setMessageType (JOptionPane.QUESTION_MESSAGE);

        // välj en standarduppsättning alternativ
        ruta.setOptionType (JOptionPane.YES_NO_OPTION);
        // Dialogrutan innehåller ett meddelande, en ikon,
        // en Ja-knapp (markerad) och en Nej-knapp.
        // Det går att välja en av följande
        // alternativuppsättningaar:
        // DEFAULT_OPTION (bara en OK knapp), YES_NO_OPTION,
        // YES_NO_CANCEL_OPTION eller OK_CANCEL_OPTION

        // skapa en dialog utifrån dialogrutan
        JDialog    dialog = ruta.createDialog (null,
                                           "En Confirm Dialog");
        dialog.setModal (true);
        dialog.setSize (200, 150);
        dialog.setLocation (50, 100);

        // visa dialogen
        dialog.setVisible (true);

        // det som användaren valt via dialogen
        Object    val = ruta.getValue ();

        if (val == null)
        // om användaren stängt dialogen utan att välja någonting
            System.out.println ("Du stängde dialogen!");
        else
        // om användaren klickat på någon av knapparna
        // i dialogrutan
        {
            // knappens ordningsnummer, från vänster till höger
            int    knappNummer = (Integer) val;
            if (knappNummer == 0)  // första knappen
                System.out.println ("Du valde Ja!");
            else                   // andra knappen
                System.out.println ("Du valde Nej!");
        }
        // om dialogen är en icke-modal dialog, returnerar metoden
        // getValue värdet JOptionPane.UNINITIALIZED_VALUE om
        // rutan ännu inte är stängd
```

```
        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En standarddialog visas. Dialogen innehåller ett meddelande, en ikon (i form av tecknet ?), en `Ja`-knapp och en `Nej`-knapp. Det valda alternativet visas på standardutmatningsenheten.

EnOptionDialog.java

Ett program som skapar och visar en standarddialog, som innehåller en egen uppsättning alternativ

En standarddialog kan innehålla ett meddelande och en egen uppsättning alternativ. Användaren kan välja något av alternativen, eller stänga dialogen utan att välja någonting.

```
import javax.swing.*;  // JOptionPane, JDialog

class EnOptionDialog
{
    public static void main (String[] args)
    {
        // en dialogruta, som innehåller ett meddelande
        JOptionPane   ruta = new JOptionPane ("Välj nästa steg:");

        // en egen uppsättning alternativ i dialogrutan
        Object[]    alternativ = { new String ("Avbryt"),
                                   new String ("Tillbaka"),
                                   new String ("Nästa") };
```

```
        ruta.setOptions (alternativ);

        // markera ett av alternativen
        ruta.setInitialValue (alternativ[2]);

        // skapa en dialog utifrån dialogrutan
        JDialog    dialog = ruta.createDialog (null,
                                               "En Option Dialog");
        dialog.setModal (true);
        dialog.setSize (240, 160);
        dialog.setLocation (50, 100);

        // visa dialogen
        dialog.setVisible (true);

        // det som användaren valt via dialogen
        Object    val = ruta.getValue ();

        if (val == null)
        // om användaren stängt dialogen utan att välja någonting
            System.out.println ("Du stängde dialogen!");
        else
        // om användaren klickat på någon av knapparna
        // inuti dialogrutan
        {
            // texten på den valda knappen
            String    text = (String) val;

            if (text.equals ("Nästa"))
                System.out.println ("Du valde Nästa!");
            else if (text.equals ("Tillbaka"))
                System.out.println ("Du valde Tillbaka!");
            else
                System.out.println ("Du valde Avbryt!");
        }
        // om dialogen är en icke-modal dialog, returnerar metoden
        // getValue värdet JOptionPane.UNINITIALIZED_VALUE om
        // rutan ännu inte är stängd

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En standarddialog visas. Dialogen innehåller ett meddelande och tre knappar. Varje knapp representerar ett av tre givna alternativ (`Nästa`, `Tillbaka`, `Avbryt`).

Det som användaren väljer visas på standardutmatningsenheten.

EnInputDialog.java

Ett program som skapar och visar en standarddialog som innehåller ett textfält

En standarddialog kan innehålla ett textfält. Via detta textfält kan en teckensträng matas in.

```
import javax.swing.*;  // JOptionPane, JDialog

class EnInputDialog
{
    public static void main (String[] args)
    {
        // en dialogruta, som innehåller ett meddelande
        JOptionPane    ruta = new JOptionPane ("Ditt namn:");

        // ett textfält i dialogrutan
        ruta.setWantsInput (true);

        // skapa en dialog utifrån dialogrutan
        JDialog    dialog = ruta.createDialog (null,
                                                "En Input Dialog");
        dialog.setModal (true);
        dialog.setSize (240, 160);
        dialog.setLocation (50, 100);

        // visa dialogen
        dialog.setVisible (true);

        // det som användaren valt via dialogen
        Object    val = ruta.getValue ();
```

```
        if (val == null)
        // om användaren stängt dialogen utan att välja någonting
            System.out.println ("Du stängde dialogen!");
        else
        // om användaren klickat på OK-knappen inuti dialogrutan
        {
            // det som användaren matat in
            String    input = (String) ruta.getInputValue ();

            System.out.println ("Du matade in: " + input);
        }
        // om dialogen är en icke-modal dialog, returnerar metoden
        // getValue värdet JOptionPane.UNINITIALIZED_VALUE om
        // rutan ännu inte är stängd

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En standarddialog visas. Dialogen innehåller ett meddelande, en OK-knapp och och ett textfält.

Om användaren matar in en text via textfältet och klickar på OK-knappen, visas den inmatade texten på standardutmatningsenheten.

Genvägar till standarddialoger

GenvagarTillStandardDialoger.java

Ett program som visar genvägar till standarddialoger

Klassen `JOptionPane` har ett antal statiska metoder, som underlättar skapandet och användningen av standarddialoger. En sådan metod skapar en standarddialogruta och motsvarande dialog, och visar denna dialog. Om användaren väljer ett alternativ eller matar in ett värde, returnerar motsvarande metod det valda alternativet eller det värdet som matats in.

```
import javax.swing.*;  // JOptionPane
import java.io.*;      // PrintWriter

class GenvagarTillStandardDialoger
{
    public static void main (String[] args)
    {
        // utmatningsverktyg
        PrintWriter    out = new PrintWriter (System.out, true);

        // en dialog, som innehåller ett meddelande
        JOptionPane.showMessageDialog (
            null,              // förälderkomponent
            "Vi startar!",     // meddelandet
            "MessageDialog",   // dialogens titel
            JOptionPane.INFORMATION_MESSAGE);  // meddelandets typ
        // man kan ha en egen ikon som femte argument

        // en dialog, som innehåller en
        // standarduppsättning alternativ
        int    val = JOptionPane.showConfirmDialog (
            null,                           // förälderkomponent
            "Vill du fortsätta?",           // meddelandet
            "ConfirmDialog",                // dialogens titel
            JOptionPane.YES_NO_OPTION,      // typ av alternativ
            JOptionPane.QUESTION_MESSAGE);  // meddelandets typ
        // Det femte argumentet kan utelämnas.
        // Man kan ha en egen ikon som sjätte argument.

        // val är ordningsnumret för den valda knappen,
        // eller -1 om dialogen stängdes via dess stängningsknapp
        // För att kunna skilja mellan olika alternativ,
        // kan även olika konstanter från klassen
        // JOptionPane användas. Dessa konstanter är:
        // CLOSED_OPTION, YES_OPTION, NO_OPTION, OK_OPTION och
        // CANCEL_OPTION
```

```
if (val == JOptionPane.YES_OPTION)
    out.println ("Vi fortsätter!");
else
    System.exit (0);


// en dialog med en egen uppsättning alternativ
Object[]    alternativ = { new String ("Jag vill!"),
                           new String ("Jag vill inte!") };
val = JOptionPane.showOptionDialog (
    null,                              // förälderkomponent
    "Vill du verkligen fortsätta?", // meddelandet
    "OptionDialog",                 // dialogens titel
    JOptionPane.DEFAULT_OPTION,     // typ av alternativ
    JOptionPane.QUESTION_MESSAGE,   // meddelandets typ
    null,                           // en egen ikon
    alternativ,          // en egen uppsättning alternativ
    alternativ[0]);                 // förvalt alternativ

// val är ordningsnumret för den valda knappen,
// eller -1 om dialogen stängs via dess stängningsknapp

if (val != 0)
    System.exit (0);
else
    out.println ("Då fortsätter vi!");

// en dialog, som innehåller ett inmatningsfält
Object    input = JOptionPane.showInputDialog (
    null,                              // förälderkomponent
    "Ditt namn?",                   // meddelandet
    "InputDialog",                  // dialogens titel
    JOptionPane.QUESTION_MESSAGE);  // meddelandets typ

// Man matar in en teckensträng,
// och klickar på OK-knappen. I detta fall
// representerar input den inmatade teckensträngen.
// I andra fall blir input null.

if (input == null)
    System.exit (0);
else
{
    String    namn = (String) input;
    out.println ("Du heter alltså: " + namn);
}

// en dialog, som innehåller en valruta med olika
// alternativ
Object[]    farger = { new String ("gul"),
```

```
                            new String ("rosa"),
                            new String ("röd"),
                            new String ("blå") };
        input = JOptionPane.showInputDialog (
            null,                               // förälderkomponent
            "Din färg?",                        // meddelandet
            "InputDialog",                      // dialogens titel
            JOptionPane.QUESTION_MESSAGE,       // meddelandets typ
            null,                               // egen ikon
            farger,                             // alternativ
            farger[0]);                         // förvalt alternativ

        // Om ett alternativ från valrutan väljs och valet
        // bekräftas genom en klickning på OK-knappen, blir input
        // det valda objektet. I andra fall blir input null.

        if (input == null)
            System.exit (0);
        else
        {
            String    farg = (String) input;
            out.println ("Din färg är alltså: " + farg);
        }

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

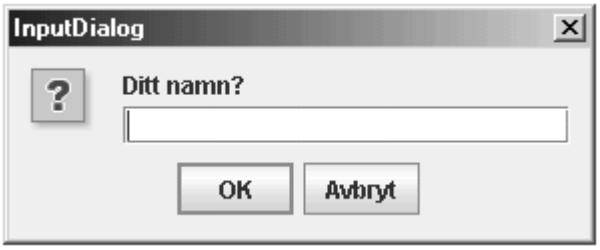
Fem dialoger (med titel) visas i en följd.

Först visas en dialog med ett meddelande, en ikon och en `OK`-knapp. Programmet fortsätter när användaren stänger dialogen (antingen via dess stängningsknapp, eller via `OK`-knappen).

Därefter visas en dialog med ett meddelande, en ikon, en `Ja`-knapp och en `Nej`-knapp. Om användaren väljer `Ja` fortsätter programmet (och ett lämpligt meddelande skrivs på standardutmatningsenheten), annars avslutas programmet.

Sedan visas en dialog med ett meddelande, en ikon, en `Jag vill!`-knapp och en `Jag vill inte!`-knapp. Om användaren väljer `Jag vill!` fortsätter programmet (och ett lämpligt meddelande skrivs på standardutmatningsenheten), annars avslutas programmet.

Därefter visas en dialog med ett meddelande, en ikon, ett textfält, en `OK`-knapp och en `Avbryt`-knapp. Om användaren väljer `OK`-knappen, läses det som skrivits inuti textfältet (och skrivs på standardutmatningsenheten), annars avslutas programmet.



Slutligen visas en dialog med ett meddelande, en ikon, en `OK`-knapp, en `Avbryt`-knapp, och en valruta med olika alternativ (4 färger), där ett alternativ (gul) är förvalt. Användaren kan öppna valrutan, och välja något annat alternativ (en annan färg). Om användaren klickar på `OK`-knappen, hämtas det valda alternativet (och visas på standardutmatningsenheten), annars avslutas programmet.

En fildialog

EnFilDialog.java

Ett program som illustrerar fildialoger

En fildialog kan användas för att välja filer och kataloger i det aktuella filsystemet.

En fildialog öppnar och sparar inte filer. Den används bara för att navigera och välja i filsystemet. För att kunna öppna eller spara en fil, måste man precisera hur filen ska öppnas eller sparas.

```
import javax.swing.*;     // JFileChooser
import java.io.*;         // File, IOException,
                          // FileWriter, PrintWriter,
                          // FileReader, BufferedReader

class EnFilDialog
{
    public static void main (String[] args) throws IOException
    {
        // en fildialogruta, som startar i den aktuella katalogen
        File    startKatalog = new File (".");
        JFileChooser filValjare = new JFileChooser (startKatalog);
        // alternativt:
        // JFileChooser    filValjare = new JFileChooser ();
        // filValjare.setCurrentDirectory (startKatalog);
        // man kan när som helst ändra den förvalda katalogen

        // visa en Spara-dialog (utan förälderkomponent)
        int    alternativ = filValjare.showSaveDialog (null);
        // en dialog skapas, fildialogrutan placeras i den,
        // och dialogen visas

        // avsluta programmet om användaren inte valt "Spara"
```

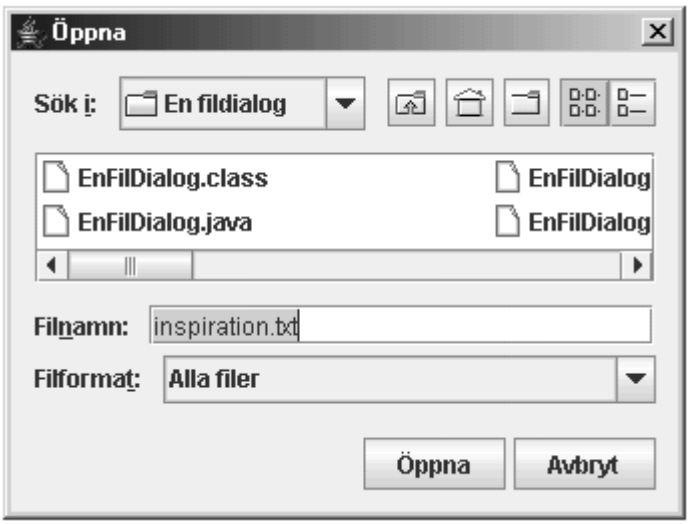
```
if (alternativ != JFileChooser.APPROVE_OPTION)
    System.exit (0);

// den filen som användaren valt
File    fil = filValjare.getSelectedFile ();

// sökvägen till den valda filen
String     sokVag = fil.getAbsolutePath ();

// avsluta programmet om användaren inte angivit ett
// .txt-fil
if (!sokVag.endsWith (".txt"))
    System.exit (0);

// spara information i den valda filen
PrintWriter  out = new PrintWriter (new FileWriter (fil));
out.println ("sanningen");
out.println ("friden");
out.println ("ljuset");
out.println ("kärleken");
out.close ();


// en fildialog med en "Öppna"-knapp
alternativ = filValjare.showOpenDialog (null);

// avsluta programmet om användaren inte valt "Öppna"
if (alternativ != JFileChooser.APPROVE_OPTION)
    System.exit (0);

// den valda filen
fil = filValjare.getSelectedFile ();

 // avsluta programmet om användaren inte angivit ett
// .txt-fil
sokVag = fil.getAbsolutePath ();
if (!sokVag.endsWith (".txt"))
    System.exit (0);

// öppna filen (läs filen, och visa dess innehåll på
// standardutmatningsenheten)
BufferedReader in =
           new BufferedReader (new FileReader (fil));
String    rad = in.readLine ();
while (rad != null)
{
    System.out.println (rad);
    rad = in.readLine ();
}
in.close ();
```

```
            // avsluta programmet (avsluta GUI-tråden)
            System.exit (0);
        }
}
```

Programmets inmatning och utmatning:

En Spara-fildialog visas först. Användaren väljer plats och namn för den fil, som informationen ska sparas i. Därefter sparas informationen i den valda filen.

Sedan visas en Öppna-fildialog. Användaren väljer den fil som ska läsas in. Därefter läses denna fil in och visas på standardutmatningsenheten.



Om användaren väljer någon fil som inte är en `.txt`-fil, avslutas programmet. Programmet avslutas även om användaren stänger fildialogen via knappen `Avbryt`, eller via dess stängningsknapp.

SuffixFileFilter.java

En klass som representerar en filfilter

Klassen `SuffixFileFilter` definierar ett filfilter, som filtrerar filer efter deras suffix.

```
import java.io.*;     // File

class SuffixFileFilter extends javax.swing.filechooser.FileFilter
{
    // filtyper som det här filtret accepterar
    private String    typAvFiler;

    // acceptabla suffix
    private String[]    suffix;

    // initierar acceptabla filtyper och suffix
    public SuffixFileFilter (String typAvFiler, String suffix)
    {
        this.typAvFiler = typAvFiler;
        this.suffix = suffix.split ("\\s");
```

```
    }

    // om en given fil är acceptabel eller inte
    public boolean accept (File fil)
    {
        boolean    acceptabelFil = false;

        // en katalog är acceptabel
        if (fil.isDirectory ())
            acceptabelFil = true;
        // om en fil
        else
        {
            // om filens namn avslutas med ett av filtrets suffix,
            // så är den acceptabel
            String     namn = fil.getName ();
            for (int i = 0; i < suffix.length; i++)
                if (namn.endsWith (suffix[i]))
                {
                    acceptabelFil = true;
                    break;
                }
        }

        return acceptabelFil;
    }

    // returnerar beskrivning av de filtyper som det här filtret
    // accepterar
    public String getDescription ()
    {
        String    beskrivning = typAvFiler + " [*" + suffix[0];

        for (int i = 1; i < suffix.length; i++)
            beskrivning += ", *" + suffix[i];
        beskrivning += "]";

        return beskrivning;
    }
}
```

EnFilDialogMedOlikaFilter.java

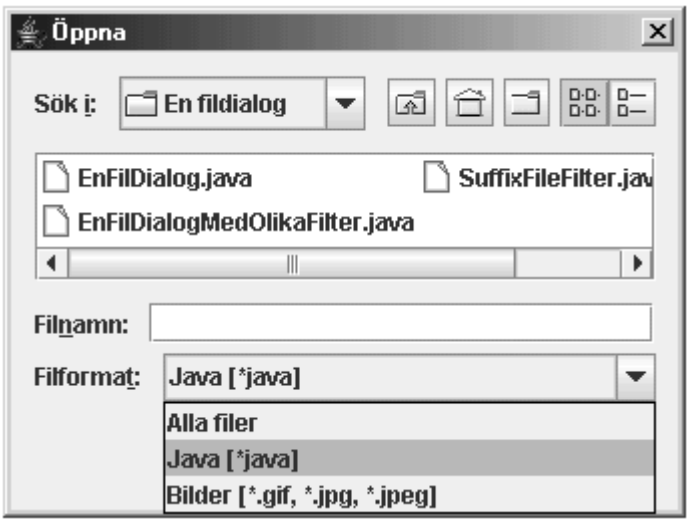
Ett program som illustrerar hur olika filter kan byggas in i en fildialog

Olika typer filter kan byggas in i en fildialog. Man kan välja ett av flera tillgängliga filter, och på så sätt begränsa de kataloger och filer som visas. Bara de kataloger och filer som det valda filtret accepterar visas.

```
import javax.swing.*;    // JFileChooser

class EnFilDialogMedOlikaFilter
{
    public static void main (String[] args)
    {
        // en fildialogruta
        JFileChooser    filValjare = new JFileChooser (".");

        // två olika filter (av samma typ)
        SuffixFileFilter    filter1 =
            new SuffixFileFilter ("Java", "java");
        SuffixFileFilter    filter2 =
            new SuffixFileFilter ("Bilder", ".gif .jpg .jpeg");

        // lägg till filtren till fildialogrutan
        filValjare.addChoosableFileFilter (filter1);
        filValjare.addChoosableFileFilter (filter2);

        // förvalt filter
        filValjare.setFileFilter (filter1);

        // filtret "Alla filer" tas bort så här:
        // filValjare.setAcceptAllFileFilterUsed (false);

        // fildialogen
        int    val = filValjare.showOpenDialog (null);

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En fildialog med olika filfilter visas. Genom att välja ett filter, begränsar man de filer som visas till en viss typ av filer.



En färgdialog

EnFargDialog.java

Ett program som illustrerar färgdialoger

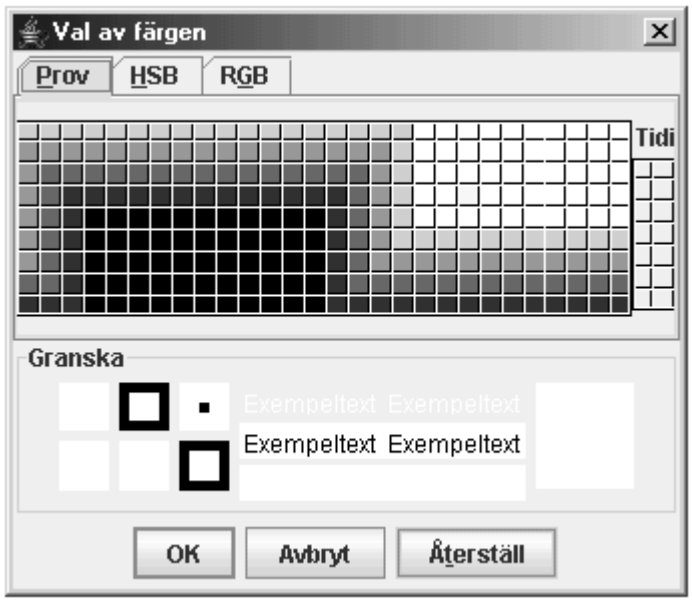
Via en färgdialog kan en färg väljas under programmets gång.

```
import java.awt.*;       // Color
import javax.swing.*;    // JColorChooser, JOptionPane, JDialog

class EnFargDialog
{
    public static void main (String[] args)
    {
        // en färgdialog
        Color    farg = JColorChooser.showDialog (
                  null,            // dialogens förälderkomponent
                  "Val av färgen",  // dialogens rubrik
                  Color.WHITE);     // förvald färg

        // om användaren stängt dialogen via dess stängningsknapp,
        // eller via Avbryt
        if (farg == null)
            System.exit (0);
```

```
        // en standarddialogruta
        JOptionPane    ruta = new JOptionPane ("Titta FÄRGEN!!!");
        ruta.setMessageType (JOptionPane.INFORMATION_MESSAGE);
        ruta.setBackground (farg);

        // skapa en dialog utifrån dialogrutan
        JDialog    dialog = ruta.createDialog (null,
                                "En färgs presentation");
        dialog.setModal (true);
        dialog.setSize (200, 150);
        dialog.setLocation (50, 100);

        // visa dialogen
        dialog.setVisible (true);

        // avsluta programmet (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En färgdialog visas. Användaren kan välja en av många färger. När användaren väljer en färg och trycker på OK-knappen, visas en standarddialog i den färgen.



Kapitel 4
Grafik

Geometriska figurer
En punkt • Linjer och kurvor • Rektanglar, ellipser och bågar
• Areor • Sammansatta figurer

Rita i en panel
Rita figurer • Rita tecken • Rita figurer av olika typer

Hantera bilder
Lagra en bild • Spara en bild • Definiera en bilds pixlar
Bearbeta en bild • Skriva ut en bild

Olika rittekniker
Olika typer av linjer • Olika fyllnadsmönster • Rita inuti en figur
Förbättra en bilds kvalitet • Sammanslagning av två bilder

Rörliga figurer
Flytta en figur • En figur som rör sig • Definiera en rörlig figur
Flera rörliga figurer

Koordinatbyte
En egen skala • Förflytta en figur • Rotera en figur
Skjuva en figur • Komposition av transformationer

Geometriska figurer

En punkt

Punkter.java

Ett program som illustrerar hur punkter i ett plan hanteras

Ett antal punkter i ett plan kan skapas och hanteras på olika sätt.

```
import java.awt.geom.*;  // Point2D, Point2D.Float, Point2D.Double
import java.io.*;        // PrintWriter

class Punkter
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("PUNKTER");
        out.println ();

        // punkter i planet
        Point2D.Float     p1 = new Point2D.Float (1.0F, 3.0F);
        Point2D.Double    p2 = new Point2D.Double (2.0, 4.0);
        out.println ("punkterna:");
        out.println (p1);
        out.println (p2);
        out.println ();

        // referera till olika punkter via superklassreferenser
        Point2D    p3 = new Point2D.Double ();
        Point2D    p4 = new Point2D.Double (3.0, 4.0);
        out.println ("punkterna:");
        out.println (p3);
        out.println (p4);

        // punkternas koordinater
        double    x3 = p3.getX ();
        double    y3 = p3.getY ();
        double    x4 = p4.getX ();
        double    y4 = p4.getY ();
        out.println ("punkternas koordinater:");
        out.println (x3 + " " + y3 + "    " + x4 + " " + y4);

        // avståndet mellan punkterna
        double    d = p3.distance (p4);
        out.println ("avståndet mellan punkterna: " + d);
```

```
        // jämför punkterna
        boolean    b = p3.equals (p4);
        out.println ("punkterna lika: " + b);
        out.println ();

        // ändra punkternas platser i planet
        p3.setLocation (4, 5);
        p4.setLocation (p3);
        out.println ("punkterna efter ändringen");
        out.println (p3);
        out.println (p4);
    }
}
```

Programmets utmatning

```
PUNKTER

punkterna:
Point2D.Float[1.0, 3.0]
Point2D.Double[2.0, 4.0]

punkterna:
Point2D.Double[0.0, 0.0]
Point2D.Double[3.0, 4.0]
punkternas koordinater:
0.0 0.0    3.0 4.0
avståndet mellan punkterna: 5.0
punkterna lika: false

punkterna efter ändringen
Point2D.Double[4.0, 5.0]
Point2D.Double[4.0, 5.0]
```

Linjer och kurvor

Linjer.java

Ett program som illustrerar hur ett linjesegment hanteras

Ett linjesegment kan skapas, och hanteras på olika sätt.

Ett linjesegment preciseras genom att dess ändpunkter anges.

```
import java.awt.geom.*;  // Point2D, Line2D, Line2D.Double
import java.io.*;        // PrintWriter
```

```
class Linjer
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("LINJER");
        out.println ();

        // ett linjesegment och dess ändpunkter
        Line2D    linje = new Line2D.Double (2, 5, 6, 5);
        out.println ("ett linjesegment och dess ändpunkter:");
        Point2D    p1 = linje.getP1 ();
        Point2D    p2 = linje.getP2 ();
        System.out.println (p1 + " " + p2);
        double    x1 = linje.getX1 ();
        double    y1 = linje.getY1 ();
        double    x2 = linje.getX2 ();
        double    y2 = linje.getY2 ();
        out.println (x1 + ", " + y1 + "    " + x2 + ", " + y2);
        out.println ();

        // ett linjesegment till
        Point2D    p3 = new Point2D.Double (4, 2);
        Point2D    p4 = new Point2D.Double (4, 8);
        Line2D    linje1 = new Line2D.Double (p3, p4);
        out.println ("ett linjesegment till:");
        out.println (linje1.getX1 () + ", " + linje1.getY1 ()
           + "    " + linje1.getX2 () + ", " + linje1.getY2 ());
        out.println ();

        // om linjesegmenten skär varandra
        boolean    b = linje.intersectsLine (linje1);
        out.println ("linjesegmenten skär varandra: " + b);
        out.println ();

        // en punkt och en linje
        Point2D    p = new Point2D.Double (7, 4);
        out.println ("punkt: " + p);
        out.print ("linjesegment: ");
        out.println (linje.getX1 () + ", " + linje.getY1 ()
           + "    " + linje.getX2 () + ", " + linje.getY2 ());
        out.print ("punkten ligger inuti linjesegmentet: ");
        out.println (linje.contains (p));
        // punkten kan inte ligga INUTI en linje
        out.print("avståndet från punkten till linjesegmentet: ");
        out.println (linje.ptSegDist (p));
        out.print ("avståndet från punkten till linjen: ");
        out.println (linje.ptLineDist (p));
        out.println ();
```

```
        // ändra ett linjesegment
        linje.setLine (0, 0, 10, 10);
        out.print ("linjesegmentet efter ändringen: ");
        out.println (linje.getX1 () + ", " + linje.getY1 ()
             + "    " + linje.getX2 () + ", " + linje.getY2 ());
    }
}
```

Programmets utmatning

```
LINJER

ett linjesegment och dess ändpunkter:
Point2D.Double[2.0, 5.0] Point2D.Double[6.0, 5.0]
2.0, 5.0    6.0, 5.0

ett linjesegment till:
4.0, 2.0    4.0, 8.0

linjesegmenten skär varandra: true

punkt: Point2D.Double[7.0, 4.0]
linjesegment: 2.0, 5.0    6.0, 5.0
punkten ligger inuti linjesegmentet: false
avståndet från punkten till linjesegmentet: 1.4142135623730951
avståndet från punkten till linjen: 1.0

linjesegmentet efter ändringen: 0.0, 0.0    10.0, 10.0
```

KvadratiskaKurvor.java

Ett program som illustrerar hur ett kvadratiskt kurvsegment hanteras

Ett segment av en kvadratisk kurva kan skapas och hanteras på olika sätt. Ett segment av en kvadratisk kurva kan preciseras genom att dess ändpunkter och en kontrollpunkt anges. Kontrollpunkten bestämmer kurvsegmentets tangenter i dess ändpunkter (den ena tangenten går från kontrollpunkten genom startpunkten, och den andra tangenten går från kontrollpunkten genom slutpunkten). Genom att ändra kontrollpunktens position, justeras kurvsegmentets utseende.

```
import java.awt.geom.*;  // Point2D, Point2D.Double,
                         // DuadCurve2D, QuadCurve2D.Double
import java.io.*;        // PrintWriter

class KvadratiskaKurvor
```

```
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("KVADRATISKA KURVOR");
        out.println ();

        // ett kvadratiskt kurvsegment
        QuadCurve2D    kurva = new QuadCurve2D.Double (
                                    1, 1,    // startpunkten
                                    2, 4,    // kontrollpunkten
                                    5, 1);   // slutpunkten
        out.println ("ett kvadratiskt kurvsegment:");
        out.print ("startpunkten:    ");
        out.println (kurva.getP1 () + "    " +
                     kurva.getX1 () + ", " + kurva.getY1 ());
        out.print ("slutpunkten:     ");
        out.println (kurva.getP2 () + "    " +
                     kurva.getX2 () + ", " + kurva.getY2 ());
        out.print ("kontrollpunkten: ");
        out.println (kurva.getCtrlPt () + "    " +
                    kurva.getCtrlX () + ", " + kurva.getCtrlY ());
        out.println ();

        // ändra kurvsegmentet
        Point2D    p1 = new Point2D.Double (2, 2);
        Point2D    p2 = new Point2D.Double (2, 5);
        Point2D    pk = new Point2D.Double (4, 3);
        kurva.setCurve (p1, pk, p2);
        // även punkternas koordinater kan anges som argument
        out.println ("kurvsegmentet efter ändringen:");
        out.print ("startpunkten:    ");
        out.println (kurva.getX1 () + ", " + kurva.getY1 ());
        out.print ("slutpunkten:     ");
        out.println (kurva.getX2 () + ", " + kurva.getY2 ());
        out.print ("kontrollpunkten: ");
        out.println(kurva.getCtrlX () + ", " + kurva.getCtrlY ());
    }
}
```

Programmets utmatning

```
KVADRATISKA KURVOR

ett kvadratiskt kurvsegment:
startpunkten:    Point2D.Double[1.0, 1.0]    1.0, 1.0
slutpunkten:     Point2D.Double[5.0, 1.0]    5.0, 1.0
kontrollpunkten: Point2D.Double[2.0, 4.0]    2.0, 4.0

kurvsegmentet efter ändringen:
```

```
startpunkten:    2.0, 2.0
slutpunkten:     2.0, 5.0
kontrollpunkten: 4.0, 3.0
```

KubiskaKurvor.java

Ett program som illustrerar hur ett kubiskt kurvsegment hanteras

Ett segment av en kubisk kurva kan skapas och hanteras på olika sätt.

Ett segment av en kubisk kurva kan preciseras genom dess ändpunkter och två kontrollpunkter. Kontrollpunkterna bestämmer kurvsegmentets tangenter i dess ändpunkter (den ena tangenten går från den första kontrollpunkten genom startpunkten, och den andra tangenten går från den andra kontrollpunkten genom slutpunkten). Genom att kontrollpunkternas positioner ändras, justeras kurvsegmentets utseende.

```
import java.awt.geom.*;  // Point2D, Point2D.Double,
                         // CubicCurve2D, CubicCurve2D.Double
import java.io.*;        // PrintWriter

class KubiskaKurvor
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("KUBISKA KURVOR");
        out.println ();

        // ett kubiskt kurvsegment
        CubicCurve2D    kurva = new CubicCurve2D.Double (
                                        1, 1,  // startpunkten
                                        2, 4,  // kontrollpunkten
                                        3, 3,  // kontrollpunkten
                                        5, 1); // slutpunkten
        out.println ("ett kubiskt kurvsegment:");
        out.print ("startpunkten:    ");
        out.println (kurva.getP1 () + "    " +
                     kurva.getX1 () + ", " + kurva.getY1 ());
        out.print ("slutpunkten:     ");
        out.println (kurva.getP2 () + "    " +
                     kurva.getX2 () + ", " + kurva.getY2 ());
        out.print ("kontrollpunkten: ");
        out.println (kurva.getCtrlP1 () + "    " +
                  kurva.getCtrlX1 () + ", " + kurva.getCtrlY1 ());
```

```
        out.print ("kontrollpunkten: ");
        out.println (kurva.getCtrlP2 () + "    " +
                  kurva.getCtrlX2 () + ", " + kurva.getCtrlY2 ());
        out.println ();

        // ändra kurvsegmentet
        Point2D    p1 = new Point2D.Double (2, 2);
        Point2D    p2 = new Point2D.Double (2, 5);
        Point2D    pk1 = new Point2D.Double (4, 3);
        Point2D    pk2 = new Point2D.Double (4, 4);
        kurva.setCurve (p1, pk1, pk2, p2);
        // även punkternas koordinater kan anges som argument
        out.println ("kurvsegmentet efter ändringen:");
        out.print ("startpunkten:    ");
        out.println (kurva.getX1 () + ", " + kurva.getY1 ());
        out.print ("slutpunkten:     ");
        out.println (kurva.getX2 () + ", " + kurva.getY2 ());
        out.print ("kontrollpunkten: ");
        out.println (kurva.getCtrlX1 () + ", "
                   + kurva.getCtrlY1 ());
        out.print ("kontrollpunkten: ");
        out.println (kurva.getCtrlX2 () + ", "
                   + kurva.getCtrlY2 ());
    }
}
```

Programmets utmatning

```
KUBISKA KURVOR

ett kubiskt kurvsegment:
startpunkten:    Point2D.Double[1.0, 1.0]    1.0, 1.0
slutpunkten:     Point2D.Double[5.0, 1.0]    5.0, 1.0
kontrollpunkten: Point2D.Double[2.0, 4.0]    2.0, 4.0
kontrollpunkten: Point2D.Double[3.0, 3.0]    3.0, 3.0

kurvsegmentet efter ändringen:
startpunkten:    2.0, 2.0
slutpunkten:     2.0, 5.0
kontrollpunkten: 4.0, 3.0
kontrollpunkten: 4.0, 4.0
```

SammansattaLinjer.java

Ett program som illustrerar hur sammansatta linjer hanteras

En sammansatt linje, som består av ett antal segment av olika typer, kan definieras. Olika linjesegment, kvadratiska kurvsegment och kubiska kurvsegment kan användas som komponenter i en sammansatt linje.

En sammansatt linje kan vara sluten eller öppen.

En sammansatt linje kan vara en sammanhängande linje (där ett segment startar på samma plats som föregående segment avslutas), eller kan bestå av flera separata delar (med flera oberoende startpunkter).

```
import java.awt.geom.*;  // Point2D, GeneralPath
import java.io.*;        // PrintWriter

class SammansattaLinjer
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("SAMMANSATTA LINJER");
        out.println ();

        // en sammansatt linje - en triangel

        out.println ("en triangel:");
        GeneralPath    path = new GeneralPath ();
        // startpunkt
        path.moveTo (2.0f, 2.0f);
        Point2D    p = path.getCurrentPoint ();
        out.print (p.getX () + ", " + p.getY ());
         // ett linjesegment till nästa punkt
        path.lineTo (6.0f, 2.0f);
        p = path.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
         // ett linjesegment till nästa punkt
        path.lineTo (4.0f, 4.0f);
        p = path.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        // slut på den sammansatta linjen
        // (linjen kan även lämnas öppen)
        path.closePath ();
        p = path.getCurrentPoint ();
        out.println ("  " + p.getX () + ", " + p.getY ());
        out.println ();
```

```
        // en osammanhängande sammansatt linje

        out.println ("en osammanhängande sammansatt linje:");
        GeneralPath    path1 = new GeneralPath ();
        // första del av den sammansatta linjen
        // (en sluten del)
        // en startpunkt
        path1.moveTo (2.0f, 2.0f);
        p = path1.getCurrentPoint ();
        out.print ("[");
        out.print (p.getX () + ", " + p.getY ());
        // ett linjesegment
        path1.lineTo (6.0f, 2.0f);
        p = path1.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        // ett kvadratiskt kurvsegment
        path1.quadTo (7.0f, 3.0f, 4.0f, 4.0f);
        p = path1.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        // ett kubiskt kurvsegment
        path1.curveTo (1.0f, 3.0f, 1.0f, 3.5f, 2.0f, 2.0f);
        p = path1.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        out.println ("]");
        // starta en separat del av den sammansatta linjen
        // (en öppen del)
        // en ny startpunkt
        path1.moveTo (8.0f, 2.0f);
        p = path1.getCurrentPoint ();
        out.print ("[");
        out.print (p.getX () + ", " + p.getY ());
        // ett linjesegment
        path1.lineTo (12.0f, 2.0f);
        p = path1.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        // ett kvadratiskt kurvsegment
        path1.quadTo (10.0f, 3.0f, 12.0f, 4.0f);
        p = path1.getCurrentPoint ();
        out.print ("  " + p.getX () + ", " + p.getY ());
        out.println ("]");

    }
}
```

Programmets utmatning

```
SAMMANSATTA LINJER

en triangel:
2.0, 2.0  6.0, 2.0  4.0, 4.0  2.0, 2.0
```

```
en osammanhängande sammansatt linje:
[2.0, 2.0  6.0, 2.0  4.0, 4.0  2.0, 2.0]
[8.0, 2.0  12.0, 2.0  12.0, 4.0]
```

Rektanglar, ellipser och bågar

Rektanglar.java

Ett program som illustrerar hur olika rektanglar hanteras

En rektangel kan skapas, och hanteras på olika sätt.

En rektangel kan preciseras genom dess övre vänstra hörn, bredd och höjd. Rektangeln kan även bestämmas via dess mittpunkt och vänstra övre hörn, eller via dess vänstra övre hörn och högra nedre hörn.

```
import java.awt.geom.*;  // Point2D, Point2D.Double,
                         // Line2D, Line2D.Double,
                         // Rectangle2D, Rectangle2D.Double
import java.io.*;        // PrintWriter

class Rektanglar
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("REKTANGLAR");
        out.println ();

        // två punkter
        Point2D    p1 = new Point2D.Double (10, 20);
        Point2D    p2 = new Point2D.Double (9, 20);
        out.println ("två punkter:");
        out.println (p1);
        out.println (p2);
        out.println ();

        // två linjesegment
        Line2D    lin1 = new Line2D.Double (70, 20, 100, 20);
        Line2D    lin2 = new Line2D.Double (71, 20, 100, 20);
        out.println ("två linjesegment:");
        out.println (lin1.getX1 () + ", " + lin1.getY1 () + "    "
                   + lin1.getX2 () + ", " + lin1.getY2 ());
        out.println (lin2.getX1 () + ", " + lin2.getY1 () + "    "
                   + lin2.getX2 () + ", " + lin2.getY2 ());
```

```
        out.println ();

        // en rektangel
        Rectangle2D    rek = new Rectangle2D.Double (
                              10.0, 20.0,   // vänstra övre hörn
                              60.0,         // bredden
                              40.0);        // höjden
        out.println ("en rektangel:");
        out.println (rek);
        out.println ("det vänstra övre hörnet: " + rek.getX ()
                                  + ", " + rek.getY ());
        out.println ("bredden: " + rek.getWidth ());
        out.println ("höjden: " + rek.getHeight ());
        out.println ("mittpunkten: " + rek.getCenterX ()
                             + ", " + rek.getCenterY ());
        out.println ("minsta x: " + rek.getMinX ());
        out.println ("minsta y: " + rek.getMinY ());
        out.println ("största x: " + rek.getMaxX ());
        out.println ("största y: " + rek.getMaxY ());
        out.println ("rektangeln innehåller den första punkten: "
                                  + rek.contains (p1));
        out.println ("rektangeln innehåller den andra punkten: "
                                  + rek.contains (p2));
        out.println ("rektangeln skär det första linjesegmentet: "
                         + rek.intersectsLine (lin1));
        out.println ("rektangeln skär det andra linjesegmentet: "
                         + rek.intersectsLine (lin2));
        out.println ();

        // ändra rektangeln
        rek.setRect (15, 30, 70, 50);
        out.println ("rektangeln efter ändringen:");
        out.println (rek);
        out.println ();

        // två nya rektanglar
        Rectangle2D    rek1 = new Rectangle2D.Double ();
        rek1.setFrameFromCenter (40, 40, 10, 20);
        Rectangle2D    rek2 = new Rectangle2D.Double ();
        rek2.setFrameFromDiagonal (10, 20, 70, 60);
        out.println ("två nya rektanglar:");
        out.println (rek1);
        out.println (rek2);
    }
}
```

Programmets utmatning

```
REKTANGLAR

två punkter:
```

```
Point2D.Double[10.0, 20.0]
Point2D.Double[9.0, 20.0]

två linjesegment:
70.0, 20.0    100.0, 20.0
71.0, 20.0    100.0, 20.0

en rektangel:
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]
det vänstra övre hörnet: 10.0, 20.0
bredden: 60.0
höjden: 40.0
mittpunkten: 40.0, 40.0
minsta x: 10.0
minsta y: 20.0
största x: 70.0
största y: 60.0
rektangeln innehåller den första punkten: true
rektangeln innehåller den andra punkten: false
rektangeln skär det första linjesegmentet: true
rektangeln skär det andra linjesegmentet: false

rektangeln efter ändringen:
java.awt.geom.Rectangle2D$Double[x=15.0,y=30.0,w=70.0,h=50.0]

två nya rektanglar:
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]
```

RundadeRektanglar.java

Ett program som illustrerar hur rektanglar med rundade hörn hanteras

En rektangel med rundade hörn kan skapas, och hanteras på olika sätt.

En rektangel med rundade hörn kan preciseras genom dess omslutande rektangel (som anges via dess vänstra övre hörn, bredd och höjd) och bredden och höjden för rundningen (bredden och höjden för den rektangel som ligger i ett hörn, och som omsluter den ellips varav en del utgör rundningen).

```
import java.awt.geom.*;  // RoundRectangle2D,
                         // RoundRectangle2D.Double
import java.io.*;        // PrintWriter

class RundadeRektanglar
```

```
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("RUNDADE REKTANGLAR");
        out.println ();

        // en rektangel med runda hörn
        RoundRectangle2D    rek = new RoundRectangle2D.Double (
                                10.0, 20.0, // vänstra övre hörnet
                                60.0,       // bredden
                                40.0,       // höjden
                                6.0,        // rundningens bredd
                                4.0);       // rundningens höjd
        out.println ("en rektangel med runda hörn:");
        out.println ("vänstra övre hörnet: " + rek.getX ()
                                    + ", " + rek.getY ());
        out.println ("bredden: " + rek.getWidth ());
        out.println ("höjden: " + rek.getHeight ());
        out.println ("rundningens bredd: " + rek.getArcWidth ());
        out.println ("rundningens höjd: " + rek.getArcHeight ());
        out.println ("mittpunkten: " + rek.getCenterX ()
                            + ", " + rek.getCenterY ());
        out.println ("minsta x: " + rek.getMinX ());
        out.println ("minsta y: " + rek.getMinY ());
        out.println ("största x: " + rek.getMaxX ());
        out.println ("största y: " + rek.getMaxY ());
        out.println ("omslutande rektangeln:");
        out.println (rek.getFrame ());
        out.println ();

        // ändra rektangeln flera gånger
        out.println ("rektangeln ändras:");
        rek.setRoundRect (15, 30, 70, 50, 5, 3);
        out.println (rek.getX () + ", " + rek.getY ()
         + "  " + rek.getWidth () + " " + rek.getHeight ()
         + "  " + rek.getArcWidth () + " " + rek.getArcHeight ());
        rek.setFrame (10, 20, 60, 40);
        out.println (rek.getX () + ", " + rek.getY ()
         + "  " + rek.getWidth () + " " + rek.getHeight ()
         + "  " + rek.getArcWidth () + " " + rek.getArcHeight ());
        rek.setFrameFromCenter (40, 40, 10, 20);
        out.println (rek.getX () + ", " + rek.getY ()
         + "  " + rek.getWidth () + " " + rek.getHeight ()
         + "  " + rek.getArcWidth () + " " + rek.getArcHeight ());
        rek.setFrameFromDiagonal (10, 20, 70, 60);
        out.println (rek.getX () + ", " + rek.getY ()
         + "  " + rek.getWidth () + " " + rek.getHeight ()
         + "  " + rek.getArcWidth () + " " + rek.getArcHeight ());
    }
```

```
}
```

`Programmets` utmatning

```
RUNDADE REKTANGLAR

en rektangel med runda hörn:
vänstra övre hörnet: 10.0, 20.0
bredden: 60.0
höjden: 40.0
rundningens bredd: 6.0
rundningens höjd: 4.0
mittpunkten: 40.0, 40.0
minsta x: 10.0
minsta y: 20.0
största x: 70.0
största y: 60.0
omslutande rektangeln:
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]

rektangeln ändras:
15.0, 30.0  70.0 50.0  5.0 3.0
10.0, 20.0  60.0 40.0  5.0 3.0
10.0, 20.0  60.0 40.0  5.0 3.0
10.0, 20.0  60.0 40.0  5.0 3.0
```

Ellipser.java

Ett program som illustrerar hur olika ellipser hanteras

En ellips kan skapas och hanteras på olika sätt. En ellips kan preciseras genom dess omslutande rektangel.

```
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import java.io.*;        // PrintWriter

class Ellipser
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("ELLIPSER");
        out.println ();

        // en ellips
        Ellipse2D    el = new Ellipse2D.Double (
                          10.0, 20.0,   // vänstra övre hörnet av
```

```
                                        // omslutande rektangeln
                        60.0,           // bredden
                        40.0);          // höjden
        out.println ("en ellips:");
        out.println ("vänstra övre hörnet: " + el.getX ()
                                    + ", " + el.getY ());
        out.println ("bredden: " + el.getWidth ());
        out.println ("höjden: " + el.getHeight ());
        out.println ("mittpunkten: " + el.getCenterX ()
                             + ", " + el.getCenterY ());
        out.println ("omslutande rektangeln:");
        out.println (el.getFrame ());
        out.println ();

        // ändra ellipsen flera gånger
        out.println ("ellipsen ändras:");
        el.setFrame (15, 30, 70, 50);
        out.println (el.getX () + ", " + el.getY ()
            + "  " + el.getWidth () + " " + el.getHeight ());
        el.setFrameFromCenter (40, 40, 10, 20);
        out.println (el.getX () + ", " + el.getY ()
            + "  " + el.getWidth () + " " + el.getHeight ());
        el.setFrameFromDiagonal (10, 20, 70, 60);
        out.println (el.getX () + ", " + el.getY ()
            + "  " + el.getWidth () + " " + el.getHeight ());
    }
}
```

Programmets utmatning

```
ELLIPSER

en ellips:
vänstra övre hörnet: 10.0, 20.0
bredden: 60.0
höjden: 40.0
mittpunkten: 40.0, 40.0
omslutande rektangeln:
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]

ellipsen ändras:
15.0, 30.0  70.0 50.0
10.0, 20.0  60.0 40.0
10.0, 20.0  60.0 40.0
```

Bagar.java

Ett program som illustrerar hur olika bågar hanteras

En båge kan skapas, och hanteras på olika sätt.

En båge kan preciseras genom den ellips där den ligger, dess startvinkel (som bestämmer den punkt på ellipsen där bågen börjar) och dess vinkel (som bestämmer bågens slutpunkt). Bågens ändpunkter kan bindas (direkt eller via dess mittpunkt), eller inte.

```
import java.awt.geom.*;  // Arc2D, Arc2D.Double
import java.io.*;        // PrintWriter

class Bagar
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("BÅGAR");
        out.println ();

        // en båge
        Arc2D    b = new Arc2D.Double (
                         10.0, 20.0,  // vänstra övre hörnet av
                                      // omslutande rektangeln
                         60.0,        // bredden
                         40.0,        // höjden
                         30.0,        // startvinkeln
                         60.0,        // bågens vinkel
                         Arc2D.PIE);  // ändpunkterna bundna
                                     // via rektangelns mittpunkt
        // även konstanterna Arc2D.CHORD (ändpunkterna bundna
        // direkt) och Arc2D.OPEN (ändpunkterna inte bundna) kan
        // användas

        out.println ("en båge:");
        out.println ("vänstra övre hörnet: " + b.getX ()
                                      + ", " + b.getY ());
        out.println ("bredden: " + b.getWidth ());
        out.println ("höjden: " + b.getHeight ());
        out.println ("startvinkeln: " + b.getAngleStart ());
        out.println ("bågens vinkel: " + b.getAngleExtent ());
        out.println ("bågens typ: " + b.getArcType ());
        out.println ("startpunkten: " + b.getStartPoint ());
        out.println ("slutpunkten: " + b.getEndPoint ());
        out.println ("mittpunkten: " + b.getCenterX ()
                            + ", " + b.getCenterY ());
```

```
        out.println ("omfattar vinkeln 60: "
                    + b.containsAngle(60));
        out.println ("omslutande rektangeln:");
        out.println (b.getFrame ());
        out.println ();

        // ändra bågen flera gånger
        out.println ("bågen ändras:");
        b.setArc (15, 30, 70, 50, -30, 60, Arc2D.CHORD);
        out.println (b.getX () + ", " + b.getY ()
           + "  " + b.getWidth () + " " + b.getHeight ()
           + "  " + b.getAngleStart () + " " + b.getAngleExtent ()
           + "  " + b.getArcType ());
        b.setFrame (10, 20, 60, 40);
        b.setAngleStart (90);
        b.setAngleExtent (30);
        b.setArcType (Arc2D.OPEN);
        out.println (b.getX () + ", " + b.getY ()
           + "  " + b.getWidth () + " " + b.getHeight ()
           + "  " + b.getAngleStart () + " " + b.getAngleExtent ()
           + "  " + b.getArcType ());
        b.setAngles (70, 40, 40, 20); // ändpunkterna anges
        out.println (b.getX () + ", " + b.getY ()
           + "  " + b.getWidth () + " " + b.getHeight ()
           + "  " + b.getAngleStart () + " " + b.getAngleExtent ()
           + "  " + b.getArcType ());
        b.setArcByCenter (60.0, 40.0, 20.0, 0, 90, Arc2D.PIE);
        // cirkelns mittpunkt och radie anges först
        out.println (b.getX () + ", " + b.getY ()
           + "  " + b.getWidth () + " " + b.getHeight ()
           + "  " + b.getAngleStart () + " " + b.getAngleExtent ()
           + "  " + b.getArcType ());
    }
}
```

Programmets utmatning

```
BÅGAR

en båge:
vänstra övre hörnet: 10.0, 20.0
bredden: 60.0
höjden: 40.0
startvinkeln: 30.0
bågens vinkel: 60.0
bågens typ: 2
startpunkten: Point2D.Double[65.98076211353316, 30.0]
slutpunkten: Point2D.Double[40.0, 20.0]
mittpunkten: 40.0, 40.0
omfattar vinkeln 60: true
omslutande rektangeln:
```

```
java.awt.geom.Rectangle2D$Double[x=10.0,y=20.0,w=60.0,h=40.0]

bågen ändras:
15.0, 30.0  70.0 50.0  -30.0 60.0  1
10.0, 20.0  60.0 40.0  90.0 30.0  0
10.0, 20.0  60.0 40.0  0.0 90.0  0
40.0, 20.0  40.0 40.0  0.0 90.0  2
```

Areor

Areor.java

Ett program som illustrerar hur olika areor hanteras

En area kan skapas utifrån en figur (ett objekt som implementerar gränssnittet `Shape`).

Två areor kan kombineras på olika sätt. En area kan utökas med en annan area. Den del av en area som även hör till en annan area kan tas bort. Även den del som inte är gemensam kan tas bort. En area kan utökas med en annan area och den gemensamma delen kan sedan tas bort.

```
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
                         // Area
import java.io.*;        // PrintWriter

class Areor
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("AREOR");
        out.println ();

        // två areor (skapade utifrån två figurer)
        Ellipse2D    ellips1 =
                         new Ellipse2D.Double (100, 50, 80, 50);
        Ellipse2D    ellips2 =
                         new Ellipse2D.Double (160, 50, 80, 50);
        Area    area1 = new Area (ellips1);
        Area    area2 = new Area (ellips2);

        // utöka en area med en annan area
        area1.add (area2);
        out.println (area1.getBounds2D ());
```

```
        // ta bort från en area den del som även tillhör
        // en annan area (behåll det som tillhör exklusivt)
        area1 = new Area (ellips1);
        area1.subtract (area2);
        out.println (area1.getBounds2D ());

        // ta bort från en area den del som inte också tillhör
        // en annan area (behåll den gemensamma delen)
        area1 = new Area (ellips1);
        area1.intersect (area2);
        out.println (area1.getBounds2D ());

        // utöka en area med en annan area, och ta sedan bort den
        // gemensamma delen
        area1 = new Area (ellips1);
        area1.exclusiveOr (area2);
        out.println (area1.getBounds2D ());
        out.println ();

        // undersök en area
        out.println (area1.isRectangular ());
        out.println (area1.isPolygonal ());
        out.println (area1.isSingular ());
        out.println (area1.isEmpty ());
        out.println ();

        // töm en area
        area1.reset ();
        out.println (area1.getBounds2D ());
    }
}
```

Programmets utmatning

```
AREOR

java.awt.geom.Rectangle2D$Double[x=100.0,y=50.0,w=140.0,h=50.0]
java.awt.geom.Rectangle2D$Double[x=100.0,y=50.0,
w=70.00000579268837,h=50.0]
java.awt.geom.Rectangle2D$Double[x=160.0,y=58.46360162881345
w=20.0,h=33.07280084244172]
java.awt.geom.Rectangle2D$Double[x=100.0,y=50.0,w=140.0,h=50.0]

false
false
true
false

java.awt.geom.Rectangle2D$Double[x=0.0,y=0.0,w=0.0,h=0.0]
```

Sammansatta figurer

EnSammansattFigur.java

Ett program som illustrerar sammansatta figurer

En sammansatt figur kan bildas utifrån flera oberoende figurer (`Shape`-s).

```
import java.awt.geom.*;  // GeneralPath, Ellipse2D,
                         // Ellipse2D.Double, Rectangle2D

class EnSammansattFigur
{
    public static void main (String[] args)
    {
        // en triangel
        GeneralPath    triangel = new GeneralPath ();
        triangel.moveTo (20, 40);
        triangel.lineTo (50, 40);
        triangel.lineTo (35, 10);
        triangel.closePath ();

        // en cirkel
        Ellipse2D  cirkel = new Ellipse2D.Double (20, 50, 30, 30);

        // en sammansatt figur
        // (som består av triangeln och cirkeln)
        GeneralPath    figur = new GeneralPath ();
        figur.append (triangel, false);
        figur.append (cirkel, false);

        // figurens omslutande rektangel
        Rectangle2D    rek = figur.getBounds2D ();
        System.out.println (rek);

        // töm figuren
        figur.reset ();
        System.out.println (figur.getBounds2D ());
    }
}
```

Programmets utmatning

```
java.awt.geom.Rectangle2D$Float[x=20.0,y=10.0,w=30.0,h=70.0]
java.awt.geom.Rectangle2D$Float[x=0.0,y=0.0,w=0.0,h=0.0]
```

Rita i en panel

Rita figurer

RepresenteraEnFarg.java

Ett program som visar hur olika färger representeras

I ett Javaprogram kan olika färger användas.

En färg preciseras genom dess komponenter. Färgens röda komponent, gröna komponent, blå komponent och alfakomponent anges. Färgen skapas som en blandning av dess röda, gröna och blå komponent. Alfakomponenten anger färgens genomskinlighet.

En färgs komponenter kan anges antigen som heltal mellan `0` och `255` (inklusive), eller som flyttal (av typen `float`) mellan `0.0` och `1.0` (inklusive).

```
import java.io.*;     // PrintWriter
import java.awt.*;    // Color

class RepresenteraEnFarg
{
    public static void main (String[] args)
    {
        PrintWriter    out = new PrintWriter (System.out, true);

        out.println ("REPRESENTERA EN FÄRG\n");

        // skapa en färg
        Color    c = new Color (200, 100, 0);
        out.println (c);

        // färgens komponenter
        // (heltal mellan 0 och 255, inklusive)
        int    r = c.getRed ();    // röda komponenten
        int    g = c.getGreen ();  // gröna komponenten
        int    b = c.getBlue ();   // blåa komponenten
        int    a = c.getAlpha ();  // alfakomponenten
                                   // (färgens genomskinlighet)
        out.println (
                "[" + r + ", " + g + ", " + b + ", " + a + "]");
        out.println ();

        // en genomskinlig färg (alfa < 255)
        Color    c1 = new Color (200, 100, 0, 180);
```

```
out.println (c1);
r = c1.getRed ();
g = c1.getGreen ();
b = c1.getBlue ();
a = c1.getAlpha ();
out.println (
       "[" + r + ", " + g + ", " + b + ", " + a + "]");
out.println ();

// ett heltal med en färgs komponenter
// (bitarna 24-31 representerar alfakomponenten,
// bitarna 16-23 representerar den röda komponenten,
// bitarna 8-15 representerar en gröna komponenten
// och bitarna 0-7 representerar den blå komponenten)
int    argb = c1.getRGB ();
Color    c2 = new Color (argb, true);
// det andra argumentet betyder att alfabitarna
// ska räknas med;
// om false anges är alfa 255
r = c2.getRed ();
g = c2.getGreen ();
b = c2.getBlue ();
a = c2.getAlpha ();
out.println (
    "[" + r + ", " + g + ", " + b + ", " + a + "]");
out.println ();

// representera en färgs komponenter med flyttal
// (mellan 0.0 och 1.0, inklusive)
Color    c3 = new Color (0.75f, 0.25f, 0.8f);
out.println (c3);
r = c3.getRed ();
g = c3.getGreen ();
b = c3.getBlue ();
a = c3.getAlpha ();
out.println (
   "[" + r + ", " + g + ", " + b + ", " + a + "]");
out.println ();

// en genomskinlig färg (alfa < 1.0)
Color    c4 = new Color (0.2f, 0.6f, 0.8f, 0.85f);
out.println (c4);
r = c4.getRed ();
g = c4.getGreen ();
b = c4.getBlue ();
a = c4.getAlpha ();
out.println (
   "[" + r + ", " + g + ", " + b + ", " + a + "]");
out.println ();

// en fördefinierad färg
```

```
        Color    c5 = Color.YELLOW;
        out.println (c5);
        r = c5.getRed ();
        g = c5.getGreen ();
        b = c5.getBlue ();
        a = c5.getAlpha ();
        out.println (
          "[" + r + ", " + g + ", " + b + ", " + a + "]");
        out.println ();

        // vit och svart
        out.println (Color.WHITE);
        out.println (Color.BLACK);

        // fördefinierade färger i klassen Color:
        // BLACK, DARK_GRAY, GRAY, LIGHT_GRAY, WHITE,
        // RED, GREEN, BLUE, YELLOW, ORANGE, MAGENTA
        // PINK, CYAN
    }
}
```

Programmets utmatning

```
REPRESENTERA EN FÄRG

java.awt.Color[r=200,g=100,b=0]
[200, 100, 0, 255]

java.awt.Color[r=200,g=100,b=0]
[200, 100, 0, 180]

[200, 100, 0, 180]

java.awt.Color[r=191,g=64,b=204]
[191, 64, 204, 255]

java.awt.Color[r=51,g=153,b=204]
[51, 153, 204, 217]

java.awt.Color[r=255,g=255,b=0]
[255, 255, 0, 255]

java.awt.Color[r=255,g=255,b=255]
java.awt.Color[r=0,g=0,b=0]
```

RitaFigurer.java

Ett program som ritar olika figurer i en panel

Olika figurer (`Shape`-s) kan ritas och målas i en panel, eller i någon annan grafisk komponent (som är ett objekt av någon subklass till klassen `javax.swing.JComponent`).

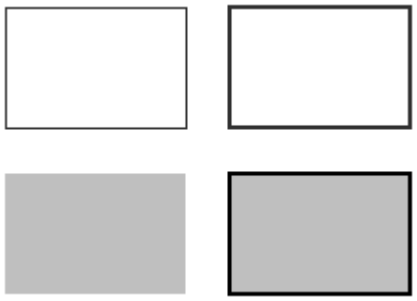
```
import java.awt.*;       // Graphics, Graphics2D, Color
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller flera figurer
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rita en rektangel
        Rectangle2D    rektangel1 = new Rectangle2D.Double (
                                        100, 50, 120, 80);
        g.draw (rektangel1);

        // rita en rektangel med en linje av en angiven tjocklek
        Rectangle2D    rektangel2 = new Rectangle2D.Double (
                                        250, 50, 120, 80);
        BasicStroke    stroke = new BasicStroke (2.0F);
        g.setStroke (stroke);
        g.draw (rektangel2);

        // måla en rektangel med en given färg
        Rectangle2D    rektangel3 = new Rectangle2D.Double (
                                        100, 160, 120, 80);
        g.setPaint (Color.LIGHT_GRAY);
        // prova att bortkommentera den övre satsen
        g.fill (rektangel3);

        // måla och rita en rektangel
        Rectangle2D    rektangel4 = new Rectangle2D.Double (
                                        250, 160, 120, 80);
        g.setPaint (Color.YELLOW);
        g.fill (rektangel4);
        g.setPaint (Color.BLACK);
        g.draw (rektangel4);
    }
```

```
}


// RitaFigurer är ett program som visar en ram,
// som innehåller en panel med flera figurer:
class RitaFigurer
{
    public static void main (String[] args)
    {
        // skapa en ram
        JFrame    frame = new JFrame (" Figurer");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // skapa en panel, och lägg den i ramen
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa ramen (och därmed panelen med figurer)
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En ritning med två rektanglar i den övre delen av en panel, och två rektanglar i den nedre delen av panelen, skapas.

Rektanglarna i den övre delen är ritade (en rektangels kontur syns) med linjer av olika tjocklek. Rektanglarna i nedre delen är målade med olika färger.



RitaMedRelativPositionOchDimensioner.java

Ett program som anger en figurs position och dimensioner relativt den panel där den ritas

Dimensionerna för den panel som olika figurer ritas i kan bestämmas. En figurs position och dimensioner kan sedan anges relativt denna panel.

```
import java.awt.*;       // Graphics, Graphics2D, Color
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
                         // Rectangle2D, Rectangle2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel med en cirkel i mitten
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // den här panelens dimensioner
        int    w = this.getWidth ();
        int    h = this.getHeight ();

        // koordinaterna för den här panelens mittpunkt
        int    xc = w / 2;
        int    yc = h / 2;

        // bestäm radien för den cirkel som ska ritas
        int    d = (h < w)? h : w;
        int    radie = 3 * d / 8;

        // rita cirkeln mitt i panelen
        Ellipse2D    cirkel = new Ellipse2D.Double ();
        cirkel.setFrameFromCenter(xc, yc, xc - radie, yc - radie);

        // måla och rita cirkeln
        g.setPaint (Color.LIGHT_GRAY);
        g.fill (cirkel);
        g.setPaint (Color.BLACK);
        g.setStroke (new BasicStroke (2));
        g.draw (cirkel);

        // rita en rektangel längs panelens kanter
        Rectangle2D    rek = new Rectangle2D.Double (0, 0, w, h);
        g.setStroke (new BasicStroke (1));
```

```
        g.draw (rek);
    }
}

// RitaMedRelativPositionOchDimensioner är ett program som visar
// ett fönster som innehåller en panel med en cirkel i mitten
class RitaMedRelativPositionOchDimensioner
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En cirkel i mitten");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En panel med en cirkel i mitten visas. Cirkelns radie är `3/8` av panelens kortare dimension. En rektangel, som går längs panelens kanter, visas.

Rita tecken

RitaTecken.java

Ett program som ritar tecken

Olika tecken kan ritas i en panel. Ett antal tecken organiseras i en teckensträng, och denna teckensträng ritas sedan.

```
import java.awt.*;       // Graphics, Graphics2D, Color, Font
import java.awt.geom.*;  // Rectangle2D
import java.awt.font.*;  // FontRenderContext
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som innehåller flera teckensträngar
class RitPanel extends JPanel
{
```

```
public void paintComponent (Graphics gr)
{
    // förberedande handlingar
    super.paintComponent (gr);
    this.setBackground (Color.WHITE);
    Graphics2D    g = (Graphics2D) gr;

    // rita en teckensträng
    String    s = "Morgonstund har guld i mun.";
    g.drawString (s, 50, 20);

    // representera olika fonter
    // (en fonts namn, stil och storlek anges)
    Font f1 = new Font ("SansSerif", Font.PLAIN, 16);// vanlig
    Font f2 = new Font ("SansSerif", Font.BOLD, 20);   // fet
    Font f3 = new Font ("SansSerif", Font.ITALIC,14);// kursiv
    Font f4 = new Font ("SansSerif",
               Font.BOLD + Font.ITALIC, 14); // fet och kursiv

    // rita teckensträngen med olika fonter
    g.setFont (f1);
    g.drawString (s, 50, 60);
    g.setFont (f2);
    g.drawString (s, 50, 90);
    g.setFont (f3);
    g.drawString (s, 50, 120);
    g.setFont (f4);
    g.drawString (s, 50, 150);

    // representera olika fonter
    // (logiska fonter används - dessa fonter avbildas till
    // konkreta fonter i den aktuella datorn)
    Font   f5 = new Font ("Serif", Font.PLAIN, 16);
    Font   f6 = new Font ("Monospaced", Font.PLAIN, 16);
    Font   f7 = new Font ("Dialog", Font.PLAIN, 16);
    Font   f8 = new Font ("DialogInput", Font.PLAIN, 16);

    // rita teckensträngen med olika fonter
    g.setFont (f5);
    g.drawString (s, 50, 190);
    g.setFont (f6);
    g.drawString (s, 50, 220);
    g.setFont (f7);
    g.drawString (s, 50, 250);
    g.setFont (f8);
    g.drawString (s, 50, 280);

    // bestäm teckensträngens bredd och höjd

    String    s1 = "Morgonstund har  ";
    String    s2 = "g u l d  ";
```

```
        String    s3 = "i mun.";

        // ett objekt som innehåller olika uppgifter, som
        // är nödvändiga för olika mätningar i samband
        // med teckensträngar
        FontRenderContext    kontext = g.getFontRenderContext ();

        // den omslutande rektangeln för den första strängen,
        // vid en vald font
        Rectangle2D    rek1 = f1.getStringBounds (s1, kontext);
        int    w1 = (int) rek1.getWidth ();
        // den omslutande rektangeln för den andra strängen,
        // vid en vald font
        Rectangle2D    rek2 = f2.getStringBounds (s2, kontext);
        int    w2 = (int) rek2.getWidth ();
        int    h2 = (int) rek2.getHeight ();

        // rita strängarna på givna platser
        g.setFont (f1);
        g.drawString (s1, 50, 330);
        g.setFont (f2);
        g.setPaint (Color.YELLOW);
        g.drawString (s2, 50 + w1, 330);
        g.setFont (f1);
        g.setPaint (Color.BLACK);
        g.drawString (s3, 50 + w1 + w2, 330);
        g.drawString (s, 50, 330 + h2);
    }
}

// RitaTecken är ett program som visar ett fönster som innehåller
// en panel med teckensträngar
class RitaTecken
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Tecken");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

EnTeckenstrangSomShape.java

Ett program som hanterar teckensträngar som figurer

En figur (en `Shape`) kan skapas utifrån en given teckensträng (en teckensträng kan tolkas som en `Shape`). Figuren kan sedan hanteras på samma sätt som alla andra figurer (`Shape`-s).

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke, Shape
import java.awt.geom.*;  // AffineTransform
import java.awt.font.*;  // FontRenderContext, TextLayout
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som innehåller flera teckensträngar
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
```

```
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rita en teckensträng
        String    s = "L o v e";
        Font font = new Font("Serif",Font.BOLD + Font.ITALIC, 60);
        g.setFont (font);
        g.drawString (s, 80, 60);

        // skapa en figur (en Shape) utifrån en teckensträng

        FontRenderContext    kontext = g.getFontRenderContext ();
        TextLayout    layout = new TextLayout (s, font, kontext);
        // ett objekt av typen TextLayout representerar en
        // teckensträng som ett grafiskt objekt
        AffineTransform    transform =
                   AffineTransform.getTranslateInstance (80, 160);
        // objektet transform bestämmer figurens/teckensträngens
        // position (baslinjens vänstra ändpunkt)
        Shape    figur1 = layout.getOutline (transform);

        // rita figuren (som alla andra Shape-s)
        g.setPaint (Color.LIGHT_GRAY);
        g.fill (figur1);
        g.setPaint (Color.BLACK);
        g.draw (figur1);

        // skapa en figur som motsvarar samma teckensträng,
        // men med en annan font

        font = new Font ("Serif", Font.BOLD + Font.ITALIC, 100);
        layout = new TextLayout (s, font, kontext);
        transform = AffineTransform.getTranslateInstance(80, 290);
        Shape    figur2 = layout.getOutline (transform);

        // rita även denna figur
        g.setFont (font);
        g.setStroke (new BasicStroke (2));
        g.setPaint (Color.LIGHT_GRAY);
        g.fill (figur2);
        g.setPaint (Color.BLACK);
        g.draw (figur2);
    }
}

// EnTeckenstrangSomShape är ett program som visar ett fönster som
// innehåller en panel med teckensträngar
class EnTeckenstrangSomShape
{
    public static void main (String[] args)
```

```
    {
        // ett fönster och en panel i den
        JFrame  frame = new JFrame (" En teckenstrang som Shape");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Rita figurer av olika typ

RitaLinjer.java

Ett program som ritar olika linjer

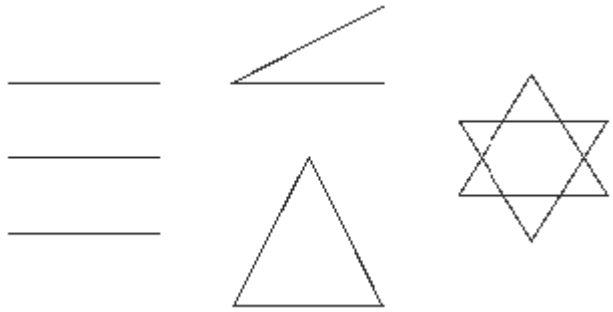
Ett linjesegment och en sammansatt linje kan ritas.

```
import java.awt.*;       // Graphics, Graphics2D
import java.awt.geom.*;  // Point2D, Point2D.Double,
                         // Linje2D, Linje2D.Double,
                         // GeneralPath
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller olika linjer
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);

        // en lämplig grafisk kontext
        Graphics2D    g = (Graphics2D) gr;

        // ett linjesegment
        Point2D    p1 = new Point2D.Double (100, 100);
        Point2D    p2 = new Point2D.Double (200, 100);
        Line2D    linje1 = new Line2D.Double (p1, p2);

        // rita linjesegmentet
        g.draw (linje1);

        // ett linjesegment
        Line2D    linje2 = new Line2D.Double (
                               new Point2D.Double (100, 150),
                               new Point2D.Double (200, 150));
        g.draw (linje2);

        // ett linjesegment
        Line2D    linje3 = new Line2D.Double (100, 200, 200, 200);
        g.draw (linje3);

        // en sammansatt linje
        GeneralPath    path1 = new GeneralPath ();
        path1.moveTo (350, 100);  // startpunkten
        path1.lineTo (250, 100);  // linjesegment till nästa punkt
        path1.lineTo (350, 50);   // linjesegment till nästa punkt

        // rita en sammansatt linje
        g.draw (path1);

        // en sluten linje - en triangel
        GeneralPath    path2 = new GeneralPath ();
        path2.moveTo (250, 250);
        path2.lineTo (300, 150);
        path2.lineTo (350, 250);
        path2.closePath ();       // slut linjen
```

```
        g.draw (path2);

        // en osammanhängande figur
        GeneralPath    path3 = new GeneralPath ();
        path3.moveTo (400, 175);  // startpunkten
        path3.lineTo (450, 95);
        path3.lineTo (500, 175);
        path3.closePath ();       // slut på en del av figuren
        path3.moveTo (400, 125);  // en ny startpunkt
        path3.lineTo (500, 125);
        path3.lineTo (450, 205);
        path3.closePath ();       // slut på en del av figuren
        g.draw (path3);
    }
}

// RitaLinjer är ett program som visar ett fönster som innehåller
// en panel med olika linjer
class RitaLinjer
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Linjer");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Tre linjer visas (under varandra) till vänster, två sammansatta linjer visas i mitten (en spetsig vinkel och en triangel), och en sammansatt linje (bestående av två trianglar) visas till höger.



RitaKurvor.java

Ett program som ritar olika kurvor

En kvadratisk kurva, en kubisk kurva och en sammansatt kurva kan ritas.

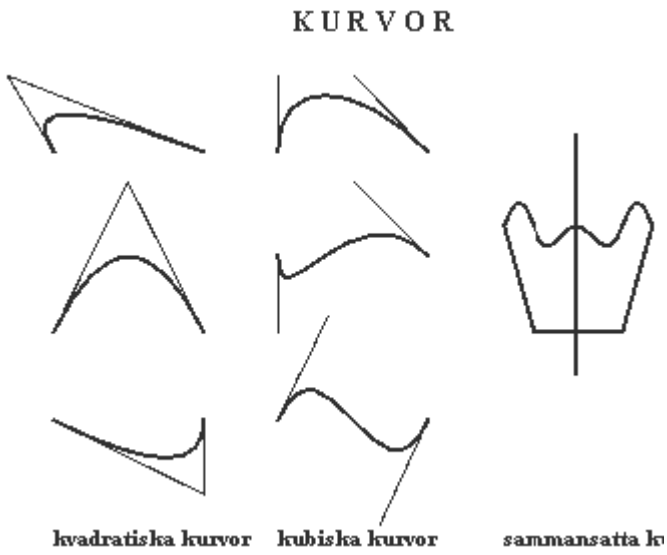
```
import java.awt.*;       // Graphics, Graphics2D, Color,
                         // Font, BasicStroke
import java.awt.geom.*;  // Linje2D, Linje2D.Double,
                         // QuadCurve2D, QuadCurve2D.Double,
                         // CubicCurve2D, CubicCurve2D.Double,
                         // GeneralPath
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller olika kurvor
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("K U R V O R", 260, 20);

        // linjetjocklek att rita kurvor med
        BasicStroke    stroke1 = new BasicStroke (2.0F);

        // linjetjocklek att rita tangenter med
```

```
BasicStroke    stroke2 = new BasicStroke (1.0F);

// en kvadratisk kurva
QuadCurve2D    kurva1 = new QuadCurve2D.Double (
                            100, 100,   // startpunkten
                             70, 50,    // kontrollpunkten
                            200, 100);  // slutpunkten

// rita kurvan
g.setStroke (stroke1);
g.draw (kurva1);

// rita kurvans tangenter
Line2D    tangent1 = new Line2D.Double (70, 50, 100, 100);
Line2D    tangent2 = new Line2D.Double (70, 50, 200, 100);
g.setStroke (stroke2);
g.draw (tangent1);
g.draw (tangent2);

// en kvadratisk kurva
QuadCurve2D    kurva2 = new QuadCurve2D.Double (
                           100, 220, 150, 120, 200, 220);
g.setStroke (stroke1);
g.draw (kurva2);

// kurvans tangenter
Line2D  tangent3 = new Line2D.Double (150, 120, 100, 220);
Line2D  tangent4 = new Line2D.Double (150, 120, 200, 220);
g.setStroke (stroke2);
g.draw (tangent3);
g.draw (tangent4);

// en kvadratisk kurva
QuadCurve2D    kurva3 = new QuadCurve2D.Double (
                           100, 280, 200, 330, 200, 280);
g.setStroke (stroke1);
g.draw (kurva3);

// kurvans tangenter
Line2D  tangent5 = new Line2D.Double (200, 330, 100, 280);
Line2D  tangent6 = new Line2D.Double (200, 330, 200, 280);
g.setStroke (stroke2);
g.draw (tangent5);
g.draw (tangent6);

// en förklarande text
Font    f2 = new Font ("Serif", Font.BOLD, 14);
g.setFont (f2);
g.drawString ("kvadratiska kurvor", 100, 365);

// en kubisk kurva
```

```
CubicCurve2D    kurva4 = new CubicCurve2D.Double (
                            250, 100,   // startpunkten
                            250, 50,    // kontrollpunkten
                            300, 50,    // kontrollpunkten
                            350, 100);  // slutpunkten
g.setStroke (stroke1);
g.draw (kurva4);

// kurvans tangenter
Line2D   tangent7 = new Line2D.Double (250, 50, 250, 100);
Line2D   tangent8 = new Line2D.Double (300, 50, 350, 100);
g.setStroke (stroke2);
g.draw (tangent7);
g.draw (tangent8);

// en kubisk kurva
CubicCurve2D    kurva5 = new CubicCurve2D.Double (
                  250, 170, 250, 220, 300, 120, 350, 170);
g.setStroke (stroke1);
g.draw (kurva5);

// kurvans tangenter
Line2D  tangent9 = new Line2D.Double (250, 220, 250, 170);
Line2D tangent10 = new Line2D.Double (300, 120, 350, 170);
g.setStroke (stroke2);
g.draw (tangent9);
g.draw (tangent10);

// en kubisk kurva
CubicCurve2D    kurva6 = new CubicCurve2D.Double (
                  250, 280, 283, 210, 317, 350, 350, 280);
g.setStroke (stroke1);
g.draw (kurva6);

// kurvans tangenter
Line2D tangent11 = new Line2D.Double (283, 210, 250, 280);
Line2D tangent12 = new Line2D.Double (317, 350, 350, 280);
g.setStroke (stroke2);
g.draw (tangent11);
g.draw (tangent12);

// en förklarande text
g.drawString ("kubiska kurvor", 250, 365);

// en sammansatt kurva
GeneralPath    kurva = new GeneralPath ();
kurva.moveTo (420, 220);  // startpunkt
kurva.lineTo (400, 150);  // en linje
kurva.quadTo (            // en kvadratisk kurva
           410, 120,      // kontrollpunkt
           420, 150);     // slutpunkt
```

```
        kurva.curveTo (           // en kubisk kurva
                  430, 180,       // kontrollpunkt
                  440, 150,       // kontrollpunkt
                  450, 150);      // slutpunkt
        kurva.curveTo (460, 150, 470, 180, 480, 150);
        kurva.quadTo (490, 120, 500, 150);
        kurva.lineTo (480, 220);
        kurva.closePath ();
        kurva.moveTo (450, 90);
        kurva.lineTo (450, 250);

        g.setStroke (stroke1);
        g.draw (kurva);

        // en förklarande text
        g.drawString ("sammansatta kurvor", 400, 365);
    }
}

// RitaKurvor är ett program som visar ett fönster som innehåller
// en panel med olika kurvor
class RitaKurvor
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Kurvor");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        frame.setVisible (true);
    }
}
```

Programmets utmatning



RitaRektanglarEllipserBagar.java

Ett program som ritar rektanglar, rundade rektanglar, ellipser och bågar

En rektangel, en rundad rektangel, en ellips och en båge kan ritas.

```
import java.awt.*;       // Graphics, Graphics2D, Color,
                         // Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
               // RoundRectangle2D, RoundRectangle2D.Double,
               // Ellipse2D, Ellipse2D.Double,
               // Arc2D, Arc2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller rektanglar,
// rundade rektanglar, ellipser och bågar
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;
```

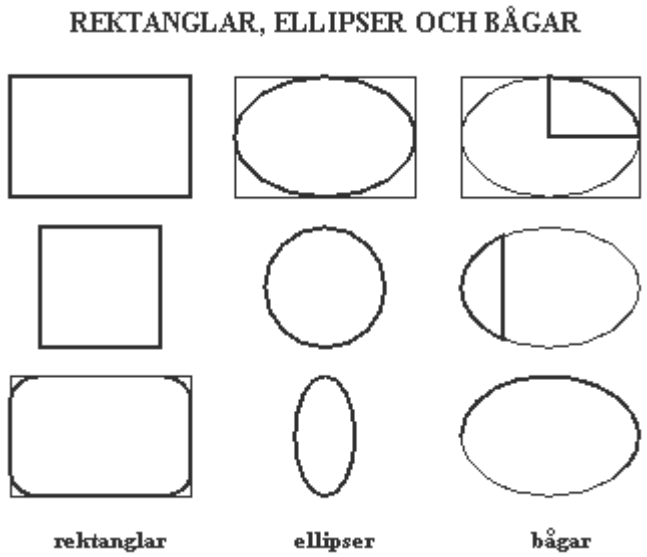
```
// rubrik
Font    f1 = new Font ("Serif", Font.BOLD, 18);
g.setFont (f1);
g.drawString ("REKTANGLAR, ELLIPSER OCH BÅGAR", 140, 20);

// linjetjocklek att rita kurvor med
BasicStroke    stroke1 = new BasicStroke (2.0F);

// linjetjocklek att rita hjälpfigurer med
BasicStroke    stroke2 = new BasicStroke (1.0F);

// rektanglar

// en rektangel
Rectangle2D    rektangel = new Rectangle2D.Double (
                   100, 50,  // övre vänstra hörnet
                   120, 80); // rektangelns bredd och höjd

// rita rektangeln
g.setStroke (stroke1);
g.draw (rektangel);

// en kvadrat
Rectangle2D    kvadrat = new Rectangle2D.Double (120, 150,
                                                 80, 80);
g.draw (kvadrat);

// en rundad rektangel
RoundRectangle2D  rundRekt = new RoundRectangle2D.Double (
                  100, 250,  // övre vänstra hörnet
                  120, 80,   // rektangelns bredd och höjd
                   40, 30);  // hörnets bredd och höjd
g.draw (rundRekt);

// den omslutande rektangeln
g.setStroke (stroke2);
g.draw (new Rectangle2D.Double (100, 250, 120, 80));

// en förklarande text
Font    f2 = new Font ("Serif", Font.BOLD, 14);
g.setFont (f2);
g.drawString ("rektanglar", 130, 365);

// ellipser

// en ellips
Ellipse2D    ellips1 = new Ellipse2D.Double (
    250, 50,  // den omslutande rektangelns
              // övre vänstra hörn
    120, 80); // den omslutande rektangelns bredd och höjd
```

```
// rita ellipsen
g.setStroke (stroke1);
g.draw (ellips1);

// den omslutande rektangeln
g.setStroke (stroke2);
g.draw (new Rectangle2D.Double (250, 50, 120, 80));

// en cirkel
Ellipse2D    cirkel =
           new Ellipse2D.Double (270, 150, 80, 80);
g.setStroke (stroke1);
g.draw (cirkel);

// en ellips till
Ellipse2D    ellips2 =
           new Ellipse2D.Double (290, 250, 40, 80);
g.draw (ellips2);

// en förklarande text
g.drawString ("ellipser", 290, 365);

// bågar

// en båge
Arc2D    bage1 = new Arc2D.Double (
    400, 50, // den omslutande rektangelns
             // vänstra övre hörn
    120, 80, // den omslutande rektangelns bredd och höjd
    0,       // startvinkeln, relativt x-axeln
    90,      // vinkeln som bestämmer bågens längd
    Arc2D.PIE); // binda bågens änder
                // med rektangelns mittpunkt

// rita bågen
g.draw (bage1);

// den omslutande rektangeln
g.setStroke (stroke2);
g.draw (new Rectangle2D.Double (400, 50, 120, 80));

// ellipsen som bågen ligger i
g.draw (new Ellipse2D.Double (400, 50, 120, 80));

// en båge
Arc2D    bage2 =
      new Arc2D.Double (400, 150, 120, 80, 120, 120,
          Arc2D.CHORD); // binda bågens änder med varandra
g.setStroke (stroke1);
g.draw (bage2);
```

```
        // ellipsen som bågen ligger i
        g.setStroke (stroke2);
        g.draw (new Ellipse2D.Double (400, 150, 120, 80));

        // en båge
        Arc2D    bage3 =
              new Arc2D.Double (400, 250, 120, 80, -30, 210,
                    Arc2D.OPEN);  // en öppen båge
        g.setStroke (stroke1);
        g.draw (bage3);

        // ellipsen som bågen ligger i
        g.setStroke (stroke2);
        g.draw (new Ellipse2D.Double (400, 250, 120, 80));

        // en förklarande text
        g.drawString ("bågar", 450, 365);
    }
}

// RitaRektanglarEllipserBagar är ett program som visar
// ett fönster, som innehåller en panel med rektanglar,
// rundade rektanglar, ellipser och bågar
class RitaRektanglarEllipserBagar
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (
                              " Rektanglar, ellipser och bågar");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        frame.setVisible (true);
    }
}
```

Programmets utmatning

Det visas två rektanglar och en rundad rektangel (och dess omslutande rektangel) till vänster, tre ellipser i mitten (för den första ellipsen även omslutande rektangel), och tre bågar (med motsvarande ellipser, för den första bågen även motsvarande rektangel) till höger. Flera förklarande texter finns också.



RitaAreor.java

Ett program som ritar olika areor

En area kan ritas i en panel.

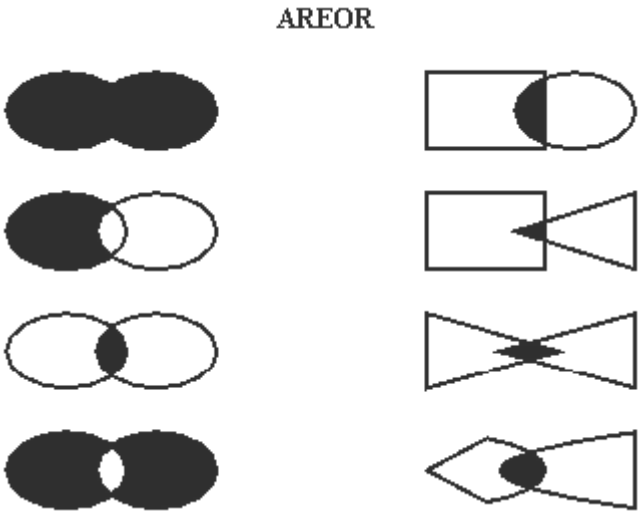
```
import java.awt.*;       // Graphics, Graphics2D,
                         // Font, BasicStroke, Color
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double,
                         // Rectangle2D, Rectangle2D.Double,
                         // Arc2D, Arc2D.Double,
                         // GeneralPath, Area
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som olika areor ritas i
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
```

```
Font    f = new Font ("Serif", Font.BOLD, 18);
g.setFont (f);
g.drawString ("AREOR", 280, 20);

// linjetjocklek att rita figurer med
BasicStroke    stroke = new BasicStroke (2.0F);
g.setStroke (stroke);

// union av två areor

// två ellipser
Ellipse2D    ellips1 =
                 new Ellipse2D.Double (100, 50, 80, 50);
Ellipse2D    ellips2 =
                 new Ellipse2D.Double (160, 50, 80, 50);

// två areor
Area    area1 = new Area (ellips1);
Area    area2 = new Area (ellips2);
g.draw (area1);
g.draw (area2);

// lägg till area2 till area1
area1.add (area2);
g.fill (area1);

// differens av två areor

Area    area3 =
    new Area (new Ellipse2D.Double (100, 130, 80, 50));
Area    area4 =
    new Area (new Ellipse2D.Double (160, 130, 80, 50));
g.draw (area3);
g.draw (area4);

// ta bort från area3 den del som även tillhör area4
area3.subtract (area4);
g.fill (area3);

// snitt av två areor

Area    area5 =
   new Area (new Ellipse2D.Double (100, 210, 80, 50));
Area    area6 =
   new Area (new Ellipse2D.Double (160, 210, 80, 50));
g.draw (area5);
g.draw (area6);

// ta bort från area5 den del som inte också tillhör area6
area5.intersect (area6);
g.fill (area5);
```

```
// det som inte är gemensamt för två areor

Area    area7 =
   new Area (new Ellipse2D.Double (100, 290, 80, 50));
Area    area8 =
   new Area (new Ellipse2D.Double (160, 290, 80, 50));
g.draw (area7);
g.draw (area8);

// lägg till area8 till area7, och ta bort
// den gemensamma delen
area7.exclusiveOr (area8);
g.fill (area7);

// snittet av en rektangel och en ellips
Area    area9 =
   new Area (new Rectangle2D.Double (380, 50, 80, 50));
Area    area10 =
   new Area (new Ellipse2D.Double (440, 50, 80, 50));
g.draw (area9);
g.draw (area10);

area9.intersect (area10);
g.fill (area9);

// snittet av en rektangel och en triangel
Area    area11 = new Area (new Rectangle2D.Double (
                                        380, 130, 80, 50));
GeneralPath    path1 = new GeneralPath ();
path1.moveTo (440, 155);
path1.lineTo (520, 130);
path1.lineTo (520, 180);
path1.closePath ();
Area    area12 = new Area (path1);
g.draw (area11);
g.draw (area12);

area11.intersect (area12);
g.fill (area11);

// snittet av två trianglar
GeneralPath    path2 = new GeneralPath ();
path2.moveTo (470, 235);
path2.lineTo (380, 210);
path2.lineTo (380, 260);
path2.closePath ();
Area    area13 = new Area (path2);

GeneralPath    path3 = new GeneralPath ();
path3.moveTo (430, 235);
```

```
        path3.lineTo (520, 210);
        path3.lineTo (520, 260);
        path3.closePath ();
        Area    area14 = new Area (path3);
        g.draw (area13);
        g.draw (area14);

        area13.intersect (area14);
        g.fill (area13);

        // snittet av två areor
        Area    area15 = new Area (new Arc2D.Double (
                  300, 290, 160, 50, -60, 120, Arc2D.PIE));

        GeneralPath    path5 = new GeneralPath ();
        path5.moveTo (520, 290);
        path5.quadTo (340, 315, 520, 340);
        path5.closePath ();
        Area    area16 = new Area (path5);
        g.draw (area15);
        g.draw (area16);

        area15.intersect (area16);
        g.fill (area15);
    }
}

// RitaAreor är ett program som visar ett fönster som
// innehåller en panel med olika areor
class RitaAreor
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Areor");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        frame.setVisible (true);
    }
}
```

Programmets utmatning



RitaSammansattaFigurer.java

Ett program som ritar sammansatta figurer

En sammansatt figur kan ritas i en panel.

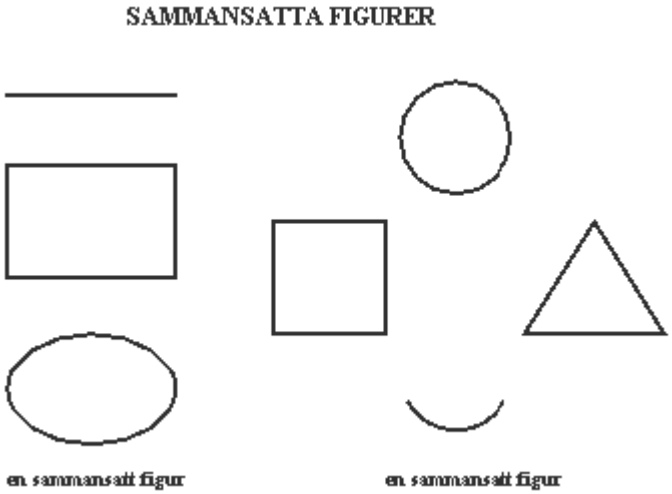
```
import java.awt.*;       // Graphics, Graphics2D,
                         // Font, BasicStroke, Color
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double,
                         // Rectangle2D, Rectangle2D.Double,
                         // Arc2D, Arc2D.Double, GeneralPath
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, där olika sammansatta figurer ritas
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;
        BasicStroke    stroke = new BasicStroke (2.0F);
        g.setStroke (stroke);

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("SAMMANSATTA FIGURER", 185, 20);
```

```
        // en sammansatt figur

        // en rektangel
        Rectangle2D    rektangel =
                  new Rectangle2D.Double (100, 120, 120, 80);
        // en ellips
        Ellipse2D    ellips =
                    new Ellipse2D.Double (100, 240, 120, 80);

        // skapa och rita en sammansatt figur
        GeneralPath    figur1 = new GeneralPath ();
        figur1.moveTo (100, 70);
        figur1.lineTo (220, 70);
        figur1.append (rektangel, false);
        figur1.append (ellips, false);
        g.draw (figur1);

        Font    f2 = new Font ("Serif", Font.BOLD, 14);
        g.setFont (f2);
        g.drawString ("en sammansatt figur", 100, 350);

        // en sammansatt figur

        // en cirkel
        Ellipse2D cirkel = new Ellipse2D.Double (380, 60, 80, 80);

        // en triangel
        GeneralPath    triangel = new GeneralPath ();
        triangel.moveTo (520, 160);
        triangel.lineTo (570, 240);
        triangel.lineTo (470, 240);
        triangel.closePath ();

        // en båge
        Arc2D    bage = new Arc2D.Double (380, 230, 80, 80,
                                          -150, 120, Arc2D.OPEN);

        // en kvadrat
         Rectangle2D  kvadrat = new Rectangle2D.Double (290, 160,
                                                        80, 80);
        // skapa och rita en sammansatt figur
        GeneralPath    figur2 = new GeneralPath ();
        figur2.append (cirkel, false);
        figur2.append (triangel, false);
        figur2.append (bage, false);
        figur2.append (kvadrat, false);
        g.draw (figur2);

        g.drawString ("en sammansatt figur", 370, 350);
    }
```

```
}

// RitaSammansattaFigurer är ett program som visar ett fönster som
// innehåller en panel med sammansatta figurer
class RitaSammansattaFigurer
{
    public static void main (String[] args)
    {
        JFrame    frame = new JFrame (" Sammansatta figurer");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.add (new RitPanel ());
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En sammansatt figur visas (bestående av en linje, en rektangel och en ellips) till vänster, och en sammansatt figur (bestående av en cirkel, en triangel, en båge och en kvadrat) till höger.



Hantera bilder

Lagra en bild

LagraVisaEnBild.java

Ett program som skapar en bild och visar den på flera olika ställen

En bild kan skapas och användas. Bilden lagras genom att dess pixlar lagras i datorns minne. En bild kan visas i en panel (eller i någon annan grafisk komponent). En delbild av en given bild kan bestämmas och användas.

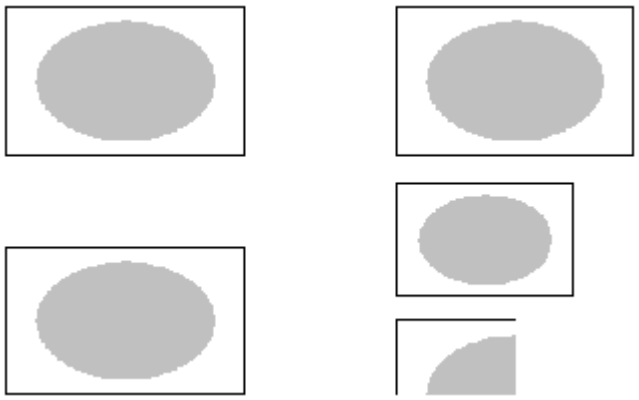
```
import java.awt.*;       // Graphics, Graphics2D,
                         // BasicStroke, Color
import java.awt.image.*; // BufferedImage
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som visar en bild på flera olika ställen
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // skapa en bild, och visa den i panelen

        // skapa en bild
        BufferedImage    bild = new BufferedImage (
                        160,  // bildens bredd
                        100,  // bildens höjd
                    BufferedImage.TYPE_INT_ARGB); // bildens typ

        // den grafiska kontexten för att rita bilden
        Graphics2D    gbi = bild.createGraphics ();
        gbi.setStroke (new BasicStroke (2.0f));

        // rita bildens ram
        // (angivna positioner och dimensioner gäller relativt
        // bilden - bildens vänstra övre hörn har
        // koordinaterna 0 och 0)
```

```
Rectangle2D ram = new Rectangle2D.Double (0, 0, 160, 100);
gbi.setPaint (Color.WHITE);
gbi.fill (ram);
gbi.setColor (Color.BLACK);
gbi.draw(ram);

// rita en ellips
Ellipse2D    el = new Ellipse2D.Double (20, 10, 120, 80);
// (angivna positioner och dimensioner gäller relativt
// bilden)
gbi.setPaint (Color.LIGHT_GRAY);
gbi.fill (el);

// frigör de resurser som den grafiska kontexten tar upp
// (bilden är färdigritad)
gbi.dispose ();

// visa bilden i den här panelen
g.drawImage (bild, // bilden
      null,        // filter för att först bearbeta bilden
                   // (filtret är av typen
                   // java.awt.image.BufferedImageOp)
      60, 40);     // bildens vänstra övre hörn
                   // relativt panelen

// visa bilden i panelen
g.drawImage (bild,   // bilden
   60,200, // bildens vänstra övre hörn
   null);  // ett objekt av typen
           // java.awt.image.ImageObserver,
           // som informeras om progressen
           // när bilden laddas

// visa bilden i panelen
g.drawImage (bild,      // bilden
      320,40, 160, 100, // rektangulärt område att rita i
      null);            // ImageObserver

// visa bilden i panelen
g.drawImage (bild,       // bilden
      320, 158, 120, 75, // området mindre än bild
                         // - förminska bilden
      null);             // ImageObserver

// en delbild
BufferedImage    delBild = bild.getSubimage (
             0, 0,     // områdets vänstra övre hörn
             80, 50);  // områdets bredd och höjd

// rita delbilden
g.drawImage (delBild, null, 320, 250);
```

```
    }
}

// LagraVisaEnBild är ett program som visar ett fönster,
// som innehåller en panel med bilder
class LagraVisaEnBild
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Lagra och visa en bild");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

En bild (en vit rektangel, med en svart ram och en grå ellips i mitten) visas i ett fönster. Bilden visas på flera olika ställen.



FImage.java

En klass som definierar en typ av bilder

En klass, som definierar en typ av bilder, kan skapas.

```
import java.awt.image.*; // BufferedImage
import java.awt.*;       // Graphics2D, BasicStroke, Color
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // Ellipse2D, Ellipse2D.Double

// en klass som definierar en typ av bilder
class FImage extends BufferedImage
{
    public FImage (
        int      w,              // bildens bredd
        int      h,              // bildens höjd
        Color    ellipseColor)   // färg för ellipsen
    {
        // skicka uppgifter om bildens dimensioner och typ
        // till superklassens konstruktor
        super (w, h, BufferedImage.TYPE_INT_ARGB);

        // verktyg att rita och måla den här bilden
        Graphics2D    g = this.createGraphics ();
        g.setStroke (new BasicStroke (2.0f));

        // måla hela ytan och rita ram för den här bilden
        Rectangle2D    ram = new Rectangle2D.Double (0, 0, w, h);
        g.setPaint (Color.WHITE);
        g.fill (ram);
        g.setColor (Color.BLACK);
        g.draw(ram);

        // måla en ellips i mitten
        Ellipse2D    el = new Ellipse2D.Double ();
        el.setFrameFromDiagonal (w / 10, h / 10,
                                 9 * w / 10, 9 * h / 10);
        g.setPaint (ellipseColor);
        g.fill (el);

        // frigör de resurser som den grafiska kontexten tar upp
        g.dispose ();
    }
}
```

DefinieraEnTypBilder.java

Ett program som skapar bilder av en given typ, och visar dessa bilder

En typ av bilder kan definieras. Sedan kan olika bilder av typen skapas och användas på olika sätt.

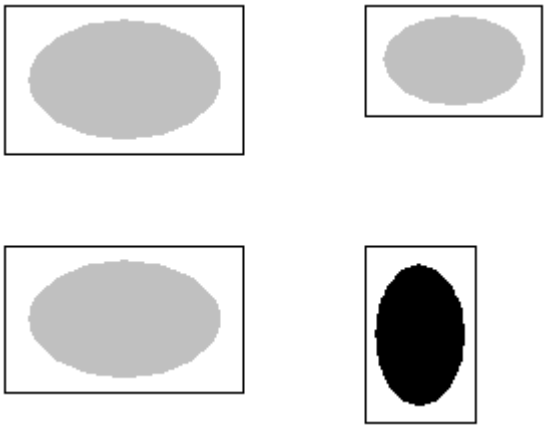
```
import java.awt.*;       // Graphics, Graphics2D, Color
```

```
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller bilder av en viss typ
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;


        // skapa en bild av en viss typ
        FImage    bild1 = new FImage (
                          160,   // bildens bredd
                          100,   // bildens höjd
                          Color.LIGHT_GRAY); // färg för ellipsen

        // visa bilden i panelen
        g.drawImage (bild1, null, 60, 40);

        // visa samma bild på en annan plats
        g.drawImage (bild1, null, 60,200);

       // skapa en bild av en viss typ, och visa den
        FImage    bild2 = new FImage (120, 75, Color.LIGHT_GRAY);
        g.drawImage (bild2, null, 320, 40);

       // skapa en bild av en viss typ, och visa den
        FImage    bild3 = new FImage (75, 120, Color.BLACK);
        g.drawImage (bild3, null, 320, 200);
    }
}

// DefinieraEnTypBilder är ett program som visar ett fönster som
// innehåller en panel med bilder
class DefinieraEnTypBilder
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame frame = new JFrame (" Definiera en typ av bilder");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
```

```
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Flera bilder visas i ett fönster. Bilderna är av samma typ: en vit rektangel med en svart ram, och en färgad ellips i mitten.



Spara en bild

SparaEnBild.java

Ett program som sparar en bild i en fil

En bild kan sparas i en fil.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // BasicStroke, Color
import java.awt.image.*; // BufferedImage
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // Ellipse2D, Ellipse2D.Double
import java.io.*;        // File, IOException
import javax.imageio.*;  // ImageIO
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller en bild
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
```

```
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // skapa en bild
        BufferedImage    bild = new BufferedImage (160, 100,
                                BufferedImage.TYPE_INT_ARGB);

        // den grafiska kontexten för att rita bilden
        Graphics2D    gbi = bild.createGraphics ();
        gbi.setStroke (new BasicStroke (2.0f));

        // rita bilden
        Rectangle2D ram = new Rectangle2D.Double (0, 0, 160, 100);
        gbi.setPaint (Color.WHITE);
        gbi.fill (ram);
        gbi.setColor (Color.BLACK);
        gbi.draw(ram);

        Ellipse2D    el = new Ellipse2D.Double (20, 10, 120, 80);
        gbi.setPaint (Color.LIGHT_GRAY);
        gbi.fill (el);

        gbi.dispose ();

        // visa bilden i panelen
        g.drawImage (bild, null, 60, 40);

        // spara bilden i en fil
        try
        {
            File    fil = new File ("bild.png");
            ImageIO.write (bild, "PNG", fil);
            // PNG är filens format
            // (även formatet JPEG kan användas)
            // Om filen redan finns och innehåller en bild,
            // skrivs den bilden över.
        }
        catch (IOException e)
        {
            System.out.println (e);
        }
    }
}

// SparaEnBild är ett program som visar ett fönster, som
// innehåller en panel med en bild
class SparaEnBild
{
```

```
    public static void main (String[] args)
    {
        JFrame    frame = new JFrame (" Spara en bild");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        frame.setVisible (true);
    }
}
```

Programmets utmatning

En bild visas i ett fönster. Bilden sparas i en fil.

LasInEnBild.java

Ett program som läser in en bild från en fil

En bild kan läsas in från en fil.

```
import java.awt.*;       // Graphics, Graphics2D, Color
import java.awt.image.*; // BufferedImage
import java.io.*;        // File, IOException
import javax.imageio.*;  // ImageIO
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som innehåller en bild
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // läs in en bild från en fil
        BufferedImage    bild = null;
```

```
        try
        {
            File    fil = new File ("bild.png");
            bild = ImageIO.read (fil);
            // PNG är filens format
            // (även formaten JPEG och GIF kan användas)
        }
        catch (IOException e)
        {
            System.out.println (e);
        }

        // visa bilden i panelen
        g.drawImage (bild, null, 60, 40);
    }
}

// LasInEnBild är ett program som visar ett fönster som innehåller
// en panel med en bild
class LasInEnBild
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
        JFrame    frame = new JFrame (" Läs in en bild");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        RitPanel    panel = new RitPanel ();
        frame.add (panel);
        frame.setVisible (true);
    }
}
```

Programmets inmatning/utmatning

En bild läses in från en fil, och visas i ett fönster.

Definiera en bilds pixlar

PixelBild.java

En klass som definierar en typ av bilder

En typ av bilder kan definieras genom att pixlarna i en bild av typen specificeras.

```
// en klass som definierar en typ av bilder
import java.awt.image.*; // BufferedImage, WritableRaster
```

```
class PixelBild extends BufferedImage
{
    public PixelBild (
                  int w)    // bildens bredd och höjd
    {
        // skicka uppgifter om bildens dimensioner och typ
        // till superklassens konstruktor
        super (w, w, BufferedImage.TYPE_INT_ARGB);

        // bildens raster
        WritableRaster    raster = this.getRaster ();

        // färger som ska användas
        int[]    svart = {0, 0, 0, 255};
        int[]    vit = {255, 255, 255, 255};
        int[]    startFarg = vit;
        int[]    farg = vit;

        // antalet pixlar längs en linje ska vara delbart med 8
        while (!(w % 8 == 0))
            w--;

        // antalet pixlar längs en linje inuti ett schackfält
        int    antal = w / 8;

        // rita ett schackbräde
        for (int i = 0; i < w; i++)
        {
            // ändra färg vid övergång till nästa fält
            if (i % antal == 0)
                if (startFarg == vit)
                    startFarg = svart;
                else
                    startFarg = vit;

            farg = startFarg;
            for (int j = 0; j < w; j++)
            {
                // ändra färg vid övergången till nästa fält
                if (j % antal == 0  &&  j != 0)
                    if (farg == vit)
                        farg = svart;
                    else
                        farg = vit;

                // rita en pixel
                if (i == 0  ||  i == w - 1  ||
                    j == 0  ||  j == w -1)   // på konturen
                    raster.setPixel (i, j, svart);
                else                         // inuti schackbrädet
```

```
                    raster.setPixel (i, j, farg);
            }
        }
    }
}
```

SkapaEnBildPixelEfterPixel.java

Ett program som skapar och visar en bild av en viss typ

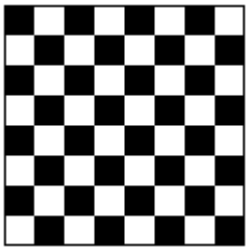
En typ av bilder kan definieras genom att specificera pixlar i en bild av typen. Sedan kan olika bilder av denna typ skapas och användas.

```
import java.awt.*;       // Graphics, Graphics2D, Color
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller en bild
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // skapa en bild som definieras (i klassen PixelBild)
        // via dess pixlar
        PixelBild    bild = new PixelBild (160);

        // visa bilden
        g.drawImage (bild, null, 100, 60);
    }
}

// SkapaEnBildPixelEfterPixel är ett program som visar
// ett fönster, som innehåller en panel med en bild
class SkapaEnBildPixelEfterPixel
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (
                            "Skapa en bild pixel efter pixel");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
```

```
        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Ett schackbräde visas i ett fönster.



Bearbeta en bild

BearbetaEnBild.java

Ett program som bearbetar en bild

En bilds enskilda pixlar kan bearbetas på olika sätt. På så sätt kan en bild, eller ett område inuti en bild, bearbetas. Vid en bilds bearbetning ändras en pixels färg eller inte, beroende på pixelns aktuella färg och position. En bild av typen `TYPE_INT_ARGB` kan bearbetas på ett sätt som inte kan användas i samband med andra typer av bilder.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // BasicStroke, Color
import java.awt.image.*; // BufferedImage, WritableRaster
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller flera bilder
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
```

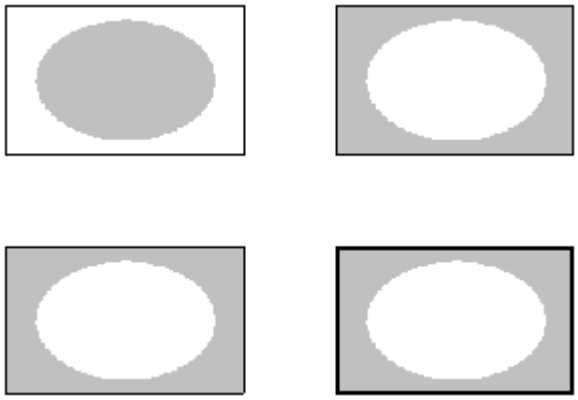
```
// förberedande handlingar
super.paintComponent (gr);
this.setBackground (Color.WHITE);
Graphics2D    g = (Graphics2D) gr;

// skapa en bild och visa den

BufferedImage    bild = new BufferedImage (160, 100,
                            BufferedImage.TYPE_INT_ARGB);
Graphics2D    gbi = bild.createGraphics ();
gbi.setStroke (new BasicStroke (2.0f));
gbi.setPaint (Color.BLACK);

Rectangle2D rek = new Rectangle2D.Double (0, 0, 160, 100);
gbi.setPaint (Color.WHITE);
gbi.fill (rek);
gbi.setColor (Color.BLACK);
gbi.draw(rek);
Ellipse2D    el = new Ellipse2D.Double (20, 10, 120, 80);
gbi.setPaint (Color.LIGHT_GRAY);
gbi.fill (el);

gbi.dispose ();

g.drawImage (bild, null, 60, 40);

// olika uppgifter om bilden

int    w = bild.getWidth ();
int    h = bild.getHeight ();
int    t = bild.getType ();
WritableRaster    raster = bild.getRaster ();
// uppgifter om bildens pixlar
int[]    pixels = new int[4 * w * h];
raster.getPixels (0, 0, w, h, pixels);
// uppgifter om bildens pixlar finns i vektorn pixels
// (varje pixel representeras med 4 int-komponenter -
// röd, grön, blå, alfa)

// bearbeta bilden

// lagra uppgifter om en pixel
int[]    pixel = new int[4];
Color    farg = null;

// stega genom alla pixlar
for (int x = 0; x < w; x++)
    for (int y = 0; y < h; y++)
    {
        // erhålla uppgifter om en pixel
        raster.getPixel (x, y, pixel);
```

```
            farg = new Color (
                pixel[0], pixel[1], pixel[2], pixel[3]);

            // analysera färgen och ändra den eventuellt
            if (farg.equals (Color.WHITE))
                farg = Color.LIGHT_GRAY;
            else if (farg.equals (Color.LIGHT_GRAY))
                farg = Color.WHITE;

            // tilldela pixelns färg
            pixel[0] = farg.getRed ();
            pixel[1] = farg.getGreen ();
            pixel[2] = farg.getBlue ();
            pixel[3] = farg.getAlpha ();
            raster.setPixel (x, y, pixel);
        }

    g.drawImage (bild, null, 280, 40);

    // skapa en ny bild utifrån den första bilden
    // (före ändringen)

    BufferedImage    bild1 = new BufferedImage (w, h, t);
    WritableRaster    raster1 = bild1.getRaster ();
    // uppgifter om alla pixlar
    int[]    pixels1 = new int[4 * w * h];
    for (int i = 0; i < pixels.length - 4; i = i + 4)
    {
        // bestäm en pixels färg och ändra den eventuellt
        farg = new Color (pixels[i], pixels[i + 1],
                          pixels[i + 2], pixels[i + 3]);
        if (farg.equals (Color.WHITE))
            farg = Color.LIGHT_GRAY;
        else if (farg.equals (Color.LIGHT_GRAY))
            farg = Color.WHITE;

        // ange en pixels färg
        pixels1[i] = farg.getRed ();
        pixels1[i + 1] = farg.getGreen ();
        pixels1[i + 2] = farg.getBlue ();
        pixels1[i + 3] = farg.getAlpha ();
    }
    // ange bildens pixlar
    raster1.setPixels (0, 0, w, h, pixels1);

    g.drawImage (bild1, null, 60, 200);

    // skapa en ny bild utifrån den första bilden
    // (efter ändringen)

    BufferedImage    bild2 = new BufferedImage (w, h, t);
```

```
        WritableRaster    raster2 = bild2.getRaster ();
        for (int x = 0; x < w; x++)
            for (int y = 0; y < h; y++)
            {
                raster.getPixel (x, y, pixel);
                farg = new Color (
                         pixel[0], pixel[1], pixel[2], pixel[3]);

                // ändra färg på pixlarna vid bildens
                // kanter
                if (x == 1  ||  x == w - 2  ||
                    y == 1  ||  y == h - 2 )
                    farg = Color.BLACK;

                pixel[0] = farg.getRed ();
                pixel[1] = farg.getGreen ();
                pixel[2] = farg.getBlue ();
                pixel[3] = farg.getAlpha ();
                raster2.setPixel (x, y, pixel);
            }

        g.drawImage (bild2, null, 280, 200);
    }
}

// BearbetaEnBild är ett program som visar ett fönster
// som innehåller en panel med olika bilder
class BearbetaEnBild
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Bearbeta en bild");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Fyra bilder visas i ett fönster. Den första bilden (en grå ellips på en vit bakgrund med en svart ram) bearbetas och visas efter varje bearbetning.



BearbetaEnBildAvEnGodtyckligTyp.java

Ett program som bearbetar en bild av en godtycklig typ

En bild av en godtycklig typ (som inte behöver vara `TYPE_INT_ARGB`) kan bearbetas.

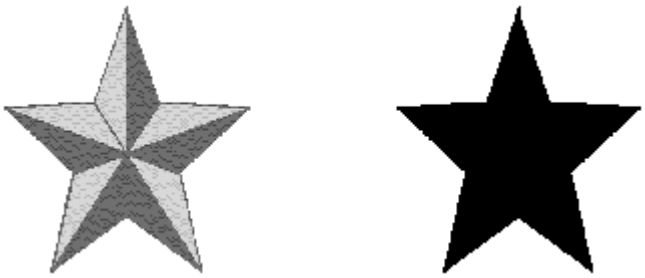
```
import java.awt.*;       // Graphics, Graphics2D, Color
import java.awt.image.*; // BufferedImage, WritableRaster,
                         // ColorModel
import java.io.*;        // File, IOException
import javax.imageio.*;  // ImageIO
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som visar en bild före och efter
// bearbetningen
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.YELLOW);
        Graphics2D    g = (Graphics2D) gr;

        // läs in en bild
        BufferedImage    bild = null;
        try
        {
            String   filNamn = System.getProperty ("java.home")
                             + File.separatorChar +"classes"
```

```
                    + File.separatorChar +"fjava"
                    + File.separatorChar +"edu"
                    + File.separatorChar + "TheStar.gif";
    File    fil = new File (filNamn);
    bild = ImageIO.read (fil);
}
catch (IOException e)
{
    System.out.println (e);
}

// visa den inlästa bilden
g.drawImage (bild, 70, 50, 200, 200, null);

// olika uppgifter om bilden
int    w = bild.getWidth ();
int    h = bild.getHeight ();
WritableRaster    raster = bild.getRaster ();
ColorModel    model = bild.getColorModel ();
// model kan översätta mellan en
// typspecifik färgrepresentation
// och en ARGB-representation (alfa, röd, grön, blå)

// bearbeta bilden
Object    pixel = null;  // typspecifik färgrepresentation
int    argb = 0;         // ARGB-färgrepresentation
Color    farg = null;    // en pixels färg
for (int x = 0; x < w; x++)
    for (int y = 0; y < h; y++)
    {
        // få uppgifter om pixelns färg
        pixel = raster.getDataElements (x, y, null);

        // översätt till motsvarande ARGB-representation
        argb = model.getRGB (pixel);
        farg = new Color (argb, true);

        // om pixeln inte är vit, ändra dess färg till blå
        if (!farg.equals (Color.WHITE))
        {
            // ändra färgen
            farg = new Color (0, 0, 255, 255);  // blå

            // översätt till en
            // typspecifik färgrepresentation
            argb = farg.getRGB ();
            pixel = model.getDataElements (argb, null);

            // ändra pixelns färg
            raster.setDataElements (x, y, pixel);
        }
```

```
            }

        // visa bilden efter bearbetningen
        g.drawImage (bild, 350, 50, 200, 200, null);
    }
}

// BearbetaEnBildAvEnGodtyckligTyp är ett program som
// visar ett fönster som innehåller en panel med två bilder
class BearbetaEnBildAvEnGodtyckligTyp
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (
                    " Bearbeta en bild av en godtycklig typ");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Två bilder visas i ett fönster; en stjärna före bearbetningen (en färgad stjärna på en vit bakgrund) och efter bearbetningen (en blå stjärna på en vit bakgrund).



Skriva ut en bild

SkrivUtEnBild.java

Ett program som skriver ut en bild

En bild kan skrivas ut.

Först samlas de nödvändiga uppgifterna om utskriften via en dialogruta. Via dialogrutan kan olika inställningar som gäller utskriftssidan och skrivaren justeras.

```
import java.awt.*;        // Graphics, Graphics2D, Color,
                          // BasicStroke, Font
import java.awt.image.*;  // BufferedImage
import java.awt.geom.*;   // Rectangle2D, Rectangle2D.Double,
                          // Ellipse2D, Ellipse2D.Double
import java.awt.print.*;  // Printable, PageFormat,
                          // PrinterException, PrinterJob
import javax.print.attribute.*;  // PrintRequestAttributeSet,
                                 // HashPrintRequestAttributeSet

// definiera en bild
class EllipseImage extends BufferedImage
{
    public EllipseImage (int w, int h)
    // w och h: bildens bredd och höjd
    {
        // skicka uppgifter om bildens bredd, höjd och typ
        // till superklassens konstruktor
        super (w, h, TYPE_INT_ARGB);

        // bildens grafiska kontext
        Graphics2D    g = this.createGraphics ();

        // ritarean och en ram
        Rectangle2D    r = new Rectangle2D.Double (0, 0, w, h);
        g.setPaint (Color.WHITE);
        g.fill (r);
        g.setColor (Color.BLACK);
        g.setStroke (new BasicStroke (2.0f));
        g.draw(r);

        // en ellips i mitten
        Ellipse2D    el = new Ellipse2D.Double ();
        el.setFrameFromDiagonal (w / 10, h / 10,
                                 9 * w / 10, 9 * h / 10);
        g.setPaint (Color.LIGHT_GRAY);
```

```
        g.fill (el);

        // frigör de resurser som använts vid ritningen
        g.dispose ();
    }
}

// definiera en bild som kan skrivas ut
class PrintableImage implements Printable
{
    // metoden som anropas vid utskriften
    public int print (Graphics gr,    // grafisk kontext
                      PageFormat pf,  // hur utskriftssidan ska
                                      // formateras
                      int page)       // sidnummer, börjar med 0
                throws PrinterException
    {
        // sluta anropa den här metoden när sidan skrivits ut
        if (page >= 1)
            return Printable.NO_SUCH_PAGE;

        // få en lämplig grafisk kontext
        Graphics2D    g = (Graphics2D) gr;

        // uppgifter om utskriftssidan

        // koordinater för det vänstra övre hörnet av den area
        // som det går att rita i (marginalerna exkluderas)
        double    x = pf.getImageableX ();
        double    y = pf.getImageableY ();

        // bredden och höjden för den area som det går att rita i
        double    w = pf.getImageableWidth ();
        double    h = pf.getImageableHeight ();

        // bestäm bildens position och dimensioner inuti sidan

        // flytta koordinatsystemets origo till det
        // vänstra övre hörnet av den area som det går att rita i
        g.translate (x, y);

        // bestäm bildens position
        int    xi = (int) (w / 10);
        int    yi = (int) (h / 10);

        // bestäm bildens dimensioner
        int    wi = (int) (8 * w / 10);
        int    hi = (int) (8 * h / 10);

        // rita bilden
```

```
        // skriv rubrik
        Font    f = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f);
        g.drawString ("En ellips", 0, 20);

        // bilden som ska ritas
        EllipseImage    image = new EllipseImage (wi, hi);

        // rita bilden
        g.drawImage (image, xi, yi, wi, hi, null);
        // Bilden ritas med den givna grafiska kontexten, inuti
        // det givna rektangulära området.
        // Skrivarens grafiska kontext använder punkter för att
        // mäta längder. Den använder inte pixlar.
        // (1 punkt = 1/72 inch, 1 inch = 25.4 mm,
        //  A4-format är ungefär 595 x 842 punkter).

        // fortsätt med utskriftsjobbet
        return Printable.PAGE_EXISTS;
    }
}

// SkrivUtEnBild är ett program som skriver ut en bild
class SkrivUtEnBild
{
    public static void main (String[] args)
    {
        // bild att skrivas ut
        PrintableImage    bild = new PrintableImage ();

        // ett objekt som lagrar olika utskriftsinställningar
        // (som hämtas via motsvarande dialogruta)
        PrintRequestAttributeSet    attribut =
                      new HashPrintRequestAttributeSet ();
        try
        {
            // ett objekt som kan skriva ut
            PrinterJob    skrivare = PrinterJob.getPrinterJob ();

            // ange utskriftsbilden
            skrivare.setPrintable (bild);

            // ange olika inställningar för skrivaren,
            // utskriftsformatet och sidans utseende
            boolean    okAttSkrivaUt =
                             skrivare.printDialog (attribut);

            // skriv ut bilden, enligt samlade attribut
            if (okAttSkrivaUt)
                skrivare.print (attribut);
            // metoden print av Printable-objektet anropas
```

```
            // (beroende av skrivaren, kan den anropas
            // flera gånger)
        }
        catch (PrinterException e)
        {
            System.out.println (e);
        }

        // avsluta programmet
        // (avsluta GUI-tråden)
        System.exit (0);
    }
}
```

Programmets utmatning

En sida skrivs ut på en skrivare. Överst på sidan visas en rubrik (En ellips). Större delen av sidan tas upp av en centrerad bild (en ellips med ram).

Olika rittekniker

Olika typer av linjer

LinjeTyper.java

Ett program som använder linjer av olika typer

Linjer av olika typer kan användas i olika sammanhang.

Linjer av olika tjocklek kan användas när en figur ritas. Olika dekorationer för en öppen linjes ändpunkter kan användas. Olika dekorationer för de punkter där två segment inuti en öppen linje möter varandra kan användas. Streckade linjer (med olika faser och olika streckmönster) kan också användas.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // BasicStroke, Color, Font
import java.awt.geom.*;  // Line2D, Line2D.Double
                         // GeneralPath
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som innehåller linjer av olika typer
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("L I N J E R  A V  O L I K A  T Y P E R",
                      150, 20);

        // linjer av olika tjocklek

        // en linje med förvald (default) tjocklek (1.0)
        Line2D    linje1 = new Line2D.Double (100, 50, 200, 50);
        g.draw (linje1);

        // medföljande förklaring
        Font    f2 = new Font ("Serif", Font.PLAIN, 12);
        g.setFont (f2);
```

```
g.drawString ("1", 145, 64);

// en linje med en given tjocklek
Line2D    linje2 = new Line2D.Double (100, 80, 200, 80);
BasicStroke    stroke = new BasicStroke (
                              4.0F);  // linjens tjocklek
g.setStroke (stroke);
g.draw (linje2);
g.drawString ("4", 145, 94);

// en linje med en given tjocklek
Line2D    linje3 = new Line2D.Double (100, 110, 200, 110);
stroke = new BasicStroke (8.0F);
g.setStroke (stroke);
g.draw (linje3);
g.drawString ("8", 145, 125);

// en förklarande text
Font    f3 = new Font ("Serif", Font.BOLD, 14);
g.setFont (f3);
g.drawString ("Tjocklek", 125, 145);

// dekoration för en öppen linjes ändpunkter

// en linje utan dekoration
Line2D    linje4 = new Line2D.Double (100, 200, 200, 200);
stroke = new BasicStroke (
                10.0F,                // tjockleck
                BasicStroke.CAP_BUTT, // ingen dekoration
                BasicStroke.JOIN_MITER);
g.setStroke (stroke);
g.draw (linje4);
g.setFont (f2);
g.drawString ("CAP_BUTT", 114, 217);

// en linje med en halvcirkel som dekoration
Line2D    linje5 = new Line2D.Double (100, 240, 200, 240);
stroke = new BasicStroke (
                   10.0F,
                   BasicStroke.CAP_ROUND, // halvcirkel
                   BasicStroke.JOIN_MITER);
g.setStroke (stroke);
g.draw (linje5);
g.drawString ("CAP_ROUND", 114, 257);

// en linje med en halvkvadrat som dekoration
Line2D    linje6 = new Line2D.Double (100, 280, 200, 280);
stroke = new BasicStroke (
                  10.0F,
                  BasicStroke.CAP_SQUARE, // halvkvadrat
                  BasicStroke.JOIN_MITER);
```

```
g.setStroke (stroke);
g.draw (linje6);
g.drawString ("CAP_SQUARE", 114, 297);

// en förklarande text
g.setFont (f3);
g.drawString ("CAP stil", 114, 320);

// dekorationer för de punkter där två segment inuti
// en öppen linje möts

// en sammansatt linje med runda mötpunkter
GeneralPath    path1 = new GeneralPath ();
path1.moveTo (350, 50);
path1.lineTo (250, 100);
path1.lineTo (350, 100);

stroke = new BasicStroke (
               10.0F,
               BasicStroke.CAP_BUTT,
               BasicStroke.JOIN_ROUND);  // rund
g.setStroke (stroke);
g.draw (path1);

g.setFont (f2);
g.drawString ("JOIN_ROUND", 255, 118);

// en sammansatt linje med raka mötpunkter
GeneralPath    path2 = new GeneralPath ();
path2.moveTo (350, 150);
path2.lineTo (250, 200);
path2.lineTo (350, 200);

stroke = new BasicStroke (
               10.0F,
               BasicStroke.CAP_BUTT,
               BasicStroke.JOIN_BEVEL);  // rak
g.setStroke (stroke);
g.draw (path2);
g.drawString ("JOIN_BEVEL", 255, 218);

// en sammansatt linje med spetsiga mötpunkter
GeneralPath    path3 = new GeneralPath ();
path3.moveTo (350, 250);
path3.lineTo (250, 300);
path3.lineTo (350, 300);

stroke = new BasicStroke (
               10.0F,
               BasicStroke.CAP_BUTT,
               BasicStroke.JOIN_MITER);  // spets
```

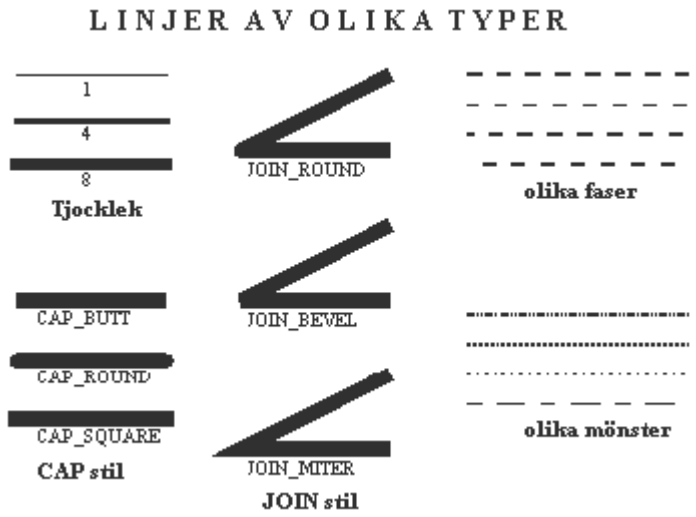
```
g.setStroke (stroke);
g.draw (path3);
g.drawString ("JOIN_MITER", 255, 318);

g.setFont (f3);
g.drawString ("JOIN stil", 264, 340);

// streckade linjer

// ett mönster som definierar längden på enskilda streck
// i en streckad linje, och längden på mellanrummen
// mellan dessa streck
float[]    monster1 = {10, 10};
// (streck med längden 10 ska följas av
// mellanrum med längden 10)
// stroke för att kunna rita streckade linjer
stroke = new BasicStroke (
       2.0F,
       BasicStroke.CAP_BUTT, // appliceras på varje streck
       BasicStroke.JOIN_MITER,
       10.0F,     // MITER-limit, om två segment möts med
                  // en vinkel som är mindre än 11
                  // appliceras BEVEL-stil
       monster1,  // streckmönster
       0); // fas, förskjutningen av det första strecket
        // till vänster från linjens början (den
        // förskjutna delen ritas inte)

// en streckad linje
Line2D    linje7 = new Line2D.Double (400, 50, 550, 50);
g.setStroke (stroke);
g.draw (linje7);

// en streckad linje
stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
     BasicStroke.JOIN_MITER, 10.0F, monster1, 2.0F);
Line2D    linje8 = new Line2D.Double (400, 70, 550, 70);
g.setStroke (stroke);
g.draw (linje8);

// en streckad linje
stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
     BasicStroke.JOIN_MITER, 10.0F, monster1, 5.0F);
Line2D    linje9 = new Line2D.Double (400, 90, 550, 90);
g.setStroke (stroke);
g.draw (linje9);

// en streckad linje
stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
     BasicStroke.JOIN_MITER, 10.0F, monster1, 10.0F);
```

```
        Line2D   linje10 = new Line2D.Double (400, 110, 550, 110);
        g.setStroke (stroke);
        g.draw (linje10);

        // en förklarande text
        g.drawString ("olika faser", 440, 132);

        // en streckad linje
        float[]    monster2 = {2, 1};
        stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
             BasicStroke.JOIN_MITER, 10.0F, monster2, 0.0F);
        Line2D   linje11 = new Line2D.Double (400, 210, 550, 210);
        g.setStroke (stroke);
        g.draw (linje11);

        // en streckad linje
        float[]    monster3 = {2, 2};
        stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
             BasicStroke.JOIN_MITER, 10.0F, monster3, 0.0F);
        Line2D   linje12 = new Line2D.Double (400, 230, 550, 230);
        g.setStroke (stroke);
        g.draw (linje12);

        // en streckad linje
        float[]    monster4 = {2, 4};
        stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
             BasicStroke.JOIN_MITER, 10.0F, monster4, 0.0F);
        Line2D   linje13 = new Line2D.Double (400, 250, 550, 250);
        g.setStroke (stroke);
        g.draw (linje13);

        // en streckad linje
        float[]    monster5 = {10, 10, 20, 10};
        stroke = new BasicStroke (2.0F, BasicStroke.CAP_BUTT,
             BasicStroke.JOIN_MITER, 10.0F, monster5, 0.0F);
        Line2D   linje14 = new Line2D.Double (400, 270, 550, 270);
        g.setStroke (stroke);
        g.draw (linje14);

        // en förklarande text
        g.drawString ("olika mönster", 440, 292);

        // förvald linje:
        // stroke = new BasicStroke (
        //                     1.0F,
        //                     BasicStroke.CAP_SQUARE,
        //                     BasicStroke.JOIN_MITER,
        //                     10.0F);
    }
}
```

```
// LinjeTyper är ett program som visar ett fönster som innehåller
// en panel med linjer av olika typer
class LinjeTyper
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
        JFrame    frame = new JFrame (" Linjetyper");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        frame.setVisible (true);
    }
}
```

Programmets GUI

Linjer av olika typer med förklarande text på flera ställen visas. Det visas linjer av olika tjocklek, linjer med olika dekorationer på sina ändar, sammansatta linjer med olika dekorationer för de punkter där enskilda segment möter varandra och streckade linjer (med olika faser och streckmönster).



Olika fyllnadsmönster

OlikaFyllnadsmonster.java

Ett program som visar olika fyllnadsmönster

En figur kan fyllas med olika mönster. En figur kan fyllas med en enhetlig färg, med en övertoning eller med en bild.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // BasicStroke
                         // Color, GradientPaint, TexturePaint
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel, JFrame
import java.awt.image.*; // BufferedImage
import java.io.*;        // File
import javax.imageio.*;  // ImageIO

// RitPanel är en panel, som innehåller figurer fyllda med
// olika fyllnadsmönster
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // fyll en figur med en given färg

        // en rektangel
        Rectangle2D  rektangel1 = new Rectangle2D.Double (100, 50,
                                                          120, 80);
        // rita rektangeln
        BasicStroke    stroke = new BasicStroke (2.0F);
        g.setStroke (stroke);
        g.draw (rektangel1);

        // fyll rektangeln
        g.setPaint (Color.ORANGE);
        g.fill (rektangel1);

        // en rektangel
        Rectangle2D rektangel2 = new Rectangle2D.Double (100, 150,
                                                         120, 80);
        g.fill (rektangel2);
```

```
g.setPaint (Color.BLACK);
g.draw (rektangel2);

// en ellips
Ellipse2D    ellips1 = new Ellipse2D.Double (100, 250,
                                             120, 80);
g.setPaint (Color.ORANGE);
g.fill (ellips1);

// fyll en figur med en övertoning

// en rektangel
Rectangle2D  rektangel3 = new Rectangle2D.Double (250, 50,
                                                  120, 80);
// rita rektangeln
g.draw (rektangel3);

// en övertoning
GradientPaint    gp1 = new GradientPaint (
                               250, 50,       // startpunkt
                               Color.ORANGE,  // startfärg
                               370, 50,       // slutpunkt
                               Color.RED);    // slutfärg

// fyll rektangeln med en övertoning
g.setPaint (gp1);
g.fill (rektangel3);

// en ellips
Ellipse2D    ellips2 = new Ellipse2D.Double (250, 150,
                                             120, 80);
GradientPaint    gp2 = new GradientPaint (
                    290, 190,  // en punkt inuti ellipsen
                    Color.ORANGE,
                    330, 190,  // en punkt inuti ellipsen
                    Color.RED);
g.setPaint (gp2);
g.fill (ellips2);

// en ellips
Ellipse2D    ellips3 = new Ellipse2D.Double (250, 250,
                                             120, 80);
GradientPaint    gp3 = new GradientPaint (
                          290, 290,
                          Color.ORANGE,
                          330, 290, Color.RED,
                          true);  // övertoningen upprepas
                                  // bortom punkterna
g.setPaint (gp3);
g.fill (ellips3);
```

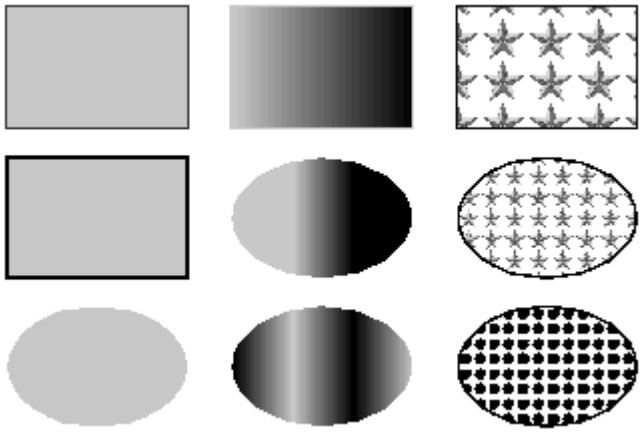
```
// fyll en figur med en given bild

// en rektangel
Rectangle2D  rektangel4 = new Rectangle2D.Double (400, 50,
                                                  120, 80);
g.setPaint (Color.BLACK);
g.draw (rektangel4);

// läs in en bild, som ska användas som ett
// fyllnadsmönster (bilden finns i en fil)
BufferedImage    bild = null;
try
{
    String   filNamn = System.getProperty ("java.home")
                     + File.separatorChar +"classes"
                     + File.separatorChar +"fjava"
                     + File.separatorChar +"edu"
                     + File.separatorChar + "TheStar.gif";
    File    fil = new File (filNamn);
    bild = ImageIO.read (fil);
}
catch (IOException e)
{
    System.out.println (e);
}

// en rektangel som representerar en elementär fyllnadsyta
Rectangle2D    ankare1 = new Rectangle2D.Double (0, 0,
         0.1 * bild.getWidth (), 0.1 * bild.getHeight ());

// ett fyllnadsmönster
TexturePaint    tp1 = new TexturePaint (bild, ankare1);

// applicera fyllnadsmönstret och fyll rektangeln
g.setPaint (tp1);
g.fill (rektangel4);

// en ellips
Ellipse2D    ellips4 = new Ellipse2D.Double (400, 150,
                                             120, 80);
Rectangle2D    ankare2 = new Rectangle2D.Double (0, 0,
       0.05 * bild.getWidth (), 0.05 * bild.getHeight ());
TexturePaint    tp2 = new TexturePaint (bild, ankare2);
g.setPaint (tp2);
g.fill (ellips4);
g.setPaint (Color.BLACK);
g.draw (ellips4);

// en ellips
Ellipse2D    ellips5 = new Ellipse2D.Double (400, 250,
                                             120, 80);
```

```
        // skapa en bild (en röd cirkel) som ska användas som ett
        // fyllnadsmönster
        BufferedImage    bild2 = new BufferedImage (10, 10,
                                    BufferedImage.TYPE_INT_ARGB);
        Graphics2D    g2 = bild2.createGraphics ();
        Ellipse2D     cirkel = new Ellipse2D.Double (0, 0, 8, 8);
        g2.setPaint (Color.RED);
        g2.fill (cirkel);
        g2.dispose ();

        // definiera ett fyllnadsmönster utifrån
        // den nyskapade bilden
        Rectangle2D    ankare3 = new Rectangle2D.Double (0, 0,
                  1 * bild2.getWidth (), 1 * bild2.getHeight ());
        TexturePaint    tp3 = new TexturePaint (bild2, ankare3);

        // applicera fyllnadsmönstret på ellipsen
        g.setPaint (tp3);
        g.fill (ellips5);

        // rita ellipsen
        g.setPaint (Color.BLACK);
        g.draw (ellips5);
    }
}

// OlikaFyllnadsmonster är ett program som visar ett fönster som
// innehåller en panel med figurer som är fyllda med olika mönster
class OlikaFyllnadsmonster
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Olika fyllnadsmönster");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Flera figurer, fyllda med olika mönster, visas i ett fönster. Till vänster visas två rektanglar och en ellips, fyllda med en enhetlig färg. I mitten visas en

rektangel och två ellipser, fyllda med en övertoning. Till höger visas en rektangel och två ellipser, fyllda med en given bild.



Rita inuti en figur

RitaInutiEnFigur.java

Ett program som visar hur ritområdet begränsas

Ritoperationer kan (eventuellt tillfälligt) begränsas till en viss figur (`Shape`).

Flera begränsningar efter varandra kan utföras. I så fall begränsas ritområdet till den area som är gemensam för de figurer som definierar dessa begränsningar.

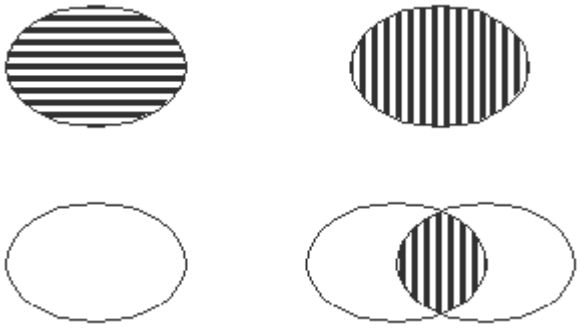
```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, BasicStroke
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller flera figurer
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;
```

```
// en ellips
Ellipse2D    ellips1 = new Ellipse2D.Double (100, 50,
                                             120, 80);
BasicStroke    stroke0 = new BasicStroke (1);
g.setStroke (stroke0);
g.draw (ellips1);

// begränsa ritoperationer till ellipsen
Shape    ritArea = g.getClip ();  // aktuella ritområdet
                                  // (hela panelen)
g.clip (ellips1);  // begränsa ritområdet till ellipsen
// ritområdet blir snittet mellan det aktuella
// ritområdet (hela panelen) och ellipsen (ritområdet
// blir själva ellipsen)

// rita - endast det som hamnar inuti ellipsen ritas
BasicStroke    stroke = new BasicStroke (4);
g.setStroke (stroke);
int    h = 50;
while (h <= 130)
{
    g.draw (new Line2D.Double (100, h, 220, h));
    h = h + 8;
}

// återgå till det ursprungliga ritområdet (hela panelen)
g.setClip (ritArea);

// en ellips
Ellipse2D    ellips2 = new Ellipse2D.Double (100, 180,
                                             120, 80);
g.setStroke (stroke0);
g.draw (ellips2);

// en ellips
Ellipse2D    ellips3 = new Ellipse2D.Double (330, 50,
                                             120, 80);
g.draw (ellips3);

// begränsa ritområdet till denna ellips
g.clip (ellips3);

// rita - det ritas bara inuti ellipsen
g.setStroke (stroke);
int    w = 330;
while (w <= 450)
{
    g.draw (new Line2D.Double (w, 50, w, 130));
    w = w + 8;
}
```

```
        // återgå till det ursprungliga ritområdet (hela panelen)
        g.setClip (ritArea);

        // en ellips
        Ellipse2D    ellips4 = new Ellipse2D.Double (300, 180,
                                                     120, 80);
        g.setStroke (stroke0);
        g.draw (ellips4);

        // en ellips
        Ellipse2D    ellips5 = new Ellipse2D.Double (360, 180,
                                                     120, 80);
        g.draw (ellips5);

        // begränsa ritområdet till snittet av två ellipser
        g.clip (ellips4);
        g.clip (ellips5);
        // begränsningarna kombineras

        // rita - det ritas bara inuti ellipsernas snitt
        g.setStroke (stroke);
        w = 300;
        while (w <= 480)
        {
            g.draw (new Line2D.Double (w, 180, w, 260));
            w = w + 8;
        }
    }
}

// RitaInutiEnFigur är ett program som visar ett fönster som
// innehåller en panel med flera figurer
class RitaInutiEnFigur
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Rita inuti en figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Fem ellipser visas i ett fönster. Två ellipser visas till vänster. I den övre ellipsen ritas horisontella linjer. Tre ellipser visas till höger. I den övre ellipsen ritas vertikala linjer. Vertikala linjer ritas även i den gemensamma delen av de två andra ellipserna.



Förbättra en bilds kvalitet

ForbattraEnBildsKvalitet.java

Ett program som visar hur en bilds kvalitet kan förbättras

En bilds kvalitet kan förbättras på bekostnad av rithastigheten (och tvärtom).

```
import java.awt.*;       // Graphics, Graphics2D,
                         // RenderingHints, BasicStroke, Font,
                         // Color, TexturePaint, AffineTransform
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
                         // Rectangle2D, Rectangle2D.Double
import java.awt.image.*; // BufferedImage
import java.awt.font.*;  // FontRenderContext, TextLayout
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som innehåller flera figurer
// av olika kvalitet
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
```

```
this.setBackground (Color.WHITE);
Graphics2D    g = (Graphics2D) gr;

// en ellips
Ellipse2D    ellips1 = new Ellipse2D.Double (70, 50,
                                             120, 80);
BasicStroke    stroke0 = new BasicStroke (2);
g.setStroke (stroke0);
g.setPaint (Color.LIGHT_GRAY);
g.fill (ellips1);
g.setPaint (Color.BLACK);
g.draw (ellips1);

// en ellips till
Ellipse2D    ellips2 = new Ellipse2D.Double (70, 150,
                                             120, 80);
// justera inställningar i den grafiska kontexten
// - förbättra ellipsens ytterlinje (på bekostnad av
// rithastigheten)
g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
// om rithastigheten ska ökas på bekostnad av
// bildens kvalitet, skriver man så här:
// g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
//                     RenderingHints.VALUE_ANTIALIAS_OFF);
g.setPaint (Color.LIGHT_GRAY);
g.fill (ellips2);
g.setPaint (Color.BLACK);
g.draw (ellips2);

// en tredje ellips
Ellipse2D    ellips3 = new Ellipse2D.Double (70, 250,
                                             120, 80);
// försök att ytterligare förbättra bildens kvalitet
g.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
            RenderingHints.VALUE_COLOR_RENDER_QUALITY);
g.setPaint (Color.LIGHT_GRAY);
g.fill (ellips3);
g.setPaint (Color.BLACK);
g.draw (ellips3);

// tillbaka till de förvalda inställningarna
g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
               RenderingHints.VALUE_ANTIALIAS_DEFAULT);
g.setRenderingHint (RenderingHints.KEY_COLOR_RENDERING,
            RenderingHints.VALUE_COLOR_RENDER_DEFAULT);

// rita en teckensträng
String    s = "Sanningen";
Font font =
         new Font ("Serif", Font.BOLD + Font.ITALIC, 30);
```

```
g.setFont (font);
g.setPaint (Color.LIGHT_GRAY);
g.drawString (s, 270, 80);

// rita samma teckensträng med bättre kvalitet
// (på bekostnad av rithastigheten)
g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
g.setRenderingHint (RenderingHints.KEY_TEXT_ANTIALIASING,
                  RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
g.setPaint (Color.LIGHT_GRAY);
g.drawString (s, 270, 110);

// definiera ett fyllnadsmönster
BufferedImage    bild = new BufferedImage (10, 10,
                            BufferedImage.TYPE_INT_ARGB);
Graphics2D    gb = bild.createGraphics ();
Ellipse2D     cirkel = new Ellipse2D.Double (0, 0, 8, 8);
gb.setPaint (Color.LIGHT_GRAY);
gb.fill (cirkel);
gb.dispose ();
Rectangle2D    ankare = new Rectangle2D.Double (0, 0,
         0.8 * bild.getWidth (), 0.8 * bild.getHeight ());
TexturePaint    tp = new TexturePaint (bild, ankare);

// tolka en teckensträng som en figur
font = new Font ("Serif", Font.BOLD + Font.ITALIC, 70);
FontRenderContext    kontext = g.getFontRenderContext ();
TextLayout    layout = new TextLayout (s, font, kontext);
AffineTransform    transform =
        AffineTransform.getTranslateInstance (270, 250);
Shape    figur1 = layout.getOutline (transform);

// rita teckensträngen (som en figur)
g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_DEFAULT);
g.setPaint (tp);
g.fill (figur1);
g.setPaint (Color.BLACK);
g.draw (figur1);

// en figur till utifrån samma teckensträng
transform =
    AffineTransform.getTranslateInstance (270, 320);
Shape    figur2 = layout.getOutline (transform);

// rita teckensträngen med bättre kvalitet
// (på bekostnad av rithastigheten)
g.setRenderingHint (RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
g.setPaint (tp);
```

```
        g.fill (figur2);
        g.setPaint (Color.BLACK);
        g.draw (figur2);
    }
}

// ForbattraEnBildsKvalitet är ett program som visar ett fönster
// som innehåller en panel med flera figurer av olika kvalitet
class ForbattraEnBildsKvalitet
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame =
             new JFrame (" Förbättra en bilds kvalitet");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Tre ellipser visas till vänster och fyra teckensträngar till höger. Dessa ellipser och teckensträngar är ritade med olika kvalitet.

Komposition av två bilder

KompositionAvTvaBilder.java

Ett program som visar hur två bilder kan kombineras

Två bilder kan kombineras på olika sätt när de ritas (en komposition av bilderna skapas). Det går att precisera vilken av bilderna som ska hamna överst, om den andra bilden ska synas, hur den ska synas, och så vidare.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, BasicStroke, AlphaComposite
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double
import java.awt.image.*; // BufferedImage
import javax.swing.*;    // JPanel, JFrame

// definiera en bildtyp
class RImage extends BufferedImage
{
    public RImage (
              int      w,  // bildens bredd
              int      h,  // bildens höjd
              Color    c)  // rektangelns färg
    {
        // skicka uppgifter om bildens dimensioner och typ
        // till superklassens konstruktor
        super (w, h, BufferedImage.TYPE_INT_ARGB);

        // verktyg att rita den här bilden med
        Graphics2D    g = this.createGraphics ();
        g.setStroke (new BasicStroke (2.0f));

        // rita en rektangel
        Rectangle2D    rek = new Rectangle2D.Double (0, 0, w, h);
        g.setPaint (c);
        g.fill (rek);
        g.setPaint (Color.BLACK);
        g.draw (rek);

        // frigör de resurser som använts vid ritningen
        g.dispose ();
    }
}

// definiera en bild som en komposition av två bilder
class CompositeImage extends BufferedImage
{
    public CompositeImage (
```

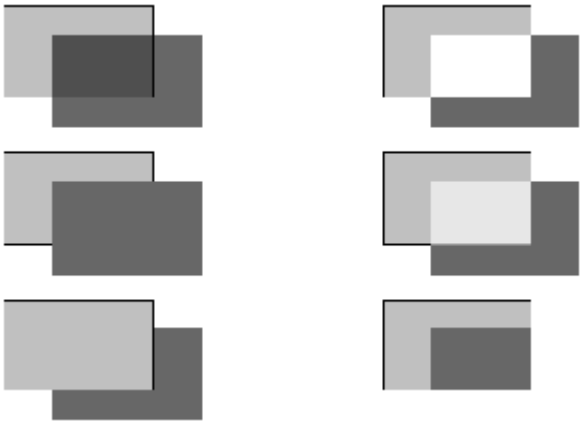
```
                    int    w,  // bildens bredd
                    int    h,  // bildens höjd
                    int    rule)  // kompositionsregel
    {
        // skicka uppgifter om bildens dimensioner och typ
        // till superklassens konstruktor
        super (w, h, BufferedImage.TYPE_INT_ARGB);

        // verktyg att rita den här bilden med
        Graphics2D    g = this.createGraphics ();

        // destinationsbilden - som ska ritas över
        RImage    im1 = new RImage (7 * w / 11, 7 * h / 11,
                                    Color.YELLOW);

        // källbilden - bilden som ska ritas överst
        RImage    im2 = new RImage (7 * w / 11, 7 * h / 11,
                                    Color.RED);

        // rita destinationsbilden
        g.drawImage (im1, null, 1 * w / 10, 1 * h / 10);

        // definiera ett sätt att kombinera bilder
        float    alfa = 0.6f;  // alfa för de pixlar
                               // som kommer över
        AlphaComposite    composite =
                 AlphaComposite.getInstance(rule, alfa);
        g.setComposite (composite);

        // rita källbilden
        g.drawImage (im2, null, 3 * w / 10, 3 * h / 10);

        // frigör de resurser som den grafiska kontexten tar upp
        g.dispose ();
    }
}

// RitPanel är en panel som innehåller olika kompositioner
// av två givna bilder
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // den här panelens dimensioner
        int    w = this.getWidth ();
        int    h = this.getHeight ();
```

```
        // flera bilder, med olika kompositionsregler
        CompositeImage    bild1 = new CompositeImage (
                           w / 4, h / 4, AlphaComposite.SRC_OVER);
        g.drawImage (bild1, null, w / 10, h / 10);

        CompositeImage    bild2 = new CompositeImage (
                             w / 4, h / 4, AlphaComposite.SRC);
        g.drawImage (bild2, null, w / 10, 35 * h / 100);

        CompositeImage    bild3 = new CompositeImage (
                           w / 4, h / 4, AlphaComposite.DST_OVER);
        g.drawImage (bild3, null, w / 10, 6 * h / 10);

        CompositeImage    bild4 = new CompositeImage (
                             w / 4, h / 4, AlphaComposite.SRC_OUT);
        g.drawImage (bild4, null, w / 2, h / 10);

        CompositeImage    bild5 = new CompositeImage (
                             w / 4, h / 4, AlphaComposite.XOR);
        g.drawImage (bild5, null, w / 2, 35 * h / 100);

        CompositeImage    bild6 = new CompositeImage (
                             w / 4, h / 4, AlphaComposite.SRC_IN);
        g.drawImage (bild6, null, w / 2, 6 * h / 10);

        // En kompositionsregel anges genom en lämplig
        // konstant i klassen AlphaComposite.
        // Det går att välja mellan följande konstanter:
        // SRC, SRC_OVER, SRC_IN, SRC_OUT, SRC_ATOP,
        // DST, DST_OVER, DST_IN, DST_OUT, DST_ATOP, XOR, CLEAR.

        // default regel för Graphics2D: SRC_OVER
    }
}

// KompositionAvTvaBilder är ett program som visar ett fönster,
// som innehåller en panel med olika kompositioner av två givna
// bilder
class KompositionAvTvaBilder
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame frame = new JFrame("Komposition av två bilder");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);
```

```
        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Sex olika bilder visas i ett fönster. Varje bild representerar en komposition av två överlappande rektanglar. Dessa rektanglar kombineras på olika sätt i olika bilder.



Rörliga figurer

Flytta en figur

FlyttaEnFigur.java

Ett program som visar hur en figur kan flyttas

En figur kan flyttas i en panel. Detta kan åstadkommas genom att panelen ritas om flera gånger, med en tidsfördröjning. Vid varje omritning ritas figuren på ett nytt ställe.

```
import java.awt.*;       // Graphics, Graphics2D
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel, JFrame

// en klass som definierar ett rum för en figur
class Rum extends JPanel
{
    // en figur
    private Shape    figur = null;

    // figurens position i rummet
    private int    x = 0;
    private int    y = 100;

    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        // rita figuren på den aktuella positionen
        figur = new Ellipse2D.Double (x, y, 12, 8);
        g.fill (figur);

        // förbered position för nästa omritning
        // (för nästa anrop till den här metoden - den här metoden
        // anropas indirekt via metoden repaint)
        x += 100;
    }
}


// FlyttaEnFigur är ett program som visar ett fönster
// som innehåller en panel med en hoppande figur
class FlyttaEnFigur
{
```

```
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Flytta en figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // ett rum där en figur rör sig
        Rum    rum = new Rum ();
        frame.add (rum);

        // visa fönstret
        frame.setVisible (true);

        // rita om rummet flera gånger
        // (det resulterar i att figuren flyttas inuti rummet)
        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep (2000);
            }
            catch (InterruptedException e)
            {}

            // begäran om att rummet ritas om
            rum.repaint ();
            // om ett anrop till repaint kommer innan föregående
            // anrop bearbetats, resulterar dessa två anrop i en
            // omritning
            // (Det behöver inte vara lika många anrop
            // till paintComponent som till repaint)
        }
    }
}
```

Programmets utmatning

Ett rum (en panel), med en partikel som hoppar i rummet, visas. Partikeln hoppar från rummets vänstra kant till rummets högra kant. Efter ett antal hopp lämnar partikeln rummet.

En figur som rör sig

Rum.java

En klass som representerar ett rum med en figur

Klassen `Rum` representerar ett rum med en figur. Figurens position i rummet kan ändras.

```
import java.awt.*;       // Color, Graphics, Graphics2D, Shape
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JPanel

class Rum extends JPanel
{
    // en figur
    Shape    figur;

    // figurens position
    private int    x;
    private int    y;

    // figurens steg
    private int    dx;
    private int    dy;

    // initiera rummets färg och figurens position och steg
    public Rum ()
    {
        this.setBackground (Color.PINK);

        this.x = -12;  // 12 är partikelns bredd
        this.y = 200;

        dx = 1;
        dy = 1;
    }

    // rita figuren
    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        figur = new Ellipse2D.Double (x, y, 12, 8);
        g.setPaint (Color.BLUE);
        g.fill (figur);
    }
```

```
    // ändra figurens position
    public boolean move ()
    {
        // modifiera partikelns position
        x += dx;
        y += dy;

        // rummets dimensioner
        int    w = this.getWidth ();
        int    h = this.getHeight ();

        // ändra riktningen vid rummets nedre och övre kant
        if (y == h - 8  ||  y == 0)  // (8 är partikelns höjd)
            dy = -dy;

        // partikeln lämnar rummet vid rummets högra kant
        boolean    inutiRummet = true;
        if (x > w + 4)  // (4 är säkerhetsmarginal)
            inutiRummet = false;

        return inutiRummet;
    }
}
```

Mover.java

En klass som flyttar en figur inuti ett givet rum

En figur kan ritas på flera successiva platser i ett givet rum. Närliggande positioner och ett kort tidsintervall mellan successiva ritningar kan väljas. På så sätt skapas ett intryck av en figur som rör sig.

```
class Mover
{
    // ett rum
    private Rum    rum;

    // initiera rummet
    public Mover (Rum rum)
    {
        this.rum = rum;
    }

     // flytta en figur inom rummet
    public void move ()
    {
        // flytta figuren så länge den inte når sin destination
```

```
        boolean    inteNattDestination = rum.move ();
        while (inteNattDestination)
        {
            // visa (rita) figuren på den aktuella platsen
            // i rummet
            rum.repaint ();

             // vänta en stund
            try
            {
                Thread.sleep (4);
            }
            catch (InterruptedException e)
            {}

             // flytta figuren till en ny position, och erhåll
             // information om figuren nått sin destination
            inteNattDestination = rum.move ();
        }
    }
}
```

EnFigurSomRorSig.java

Ett program som visar hur en figur kan röra sig

Ett intryck av en figur som rör sig inuti ett rum kan skapas.

```
import javax.swing.*;  // JFrame

// EnFigurSomRorSig är ett program som visar ett fönster
// som innehåller ett rum med en figur som rör sig
class EnFigurSomRorSig
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En figur som rör sig");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // ett rum där en partikel kan röra sig
        Rum    rum = new Rum ();
        frame.add (rum);

        // visa fönstret
        frame.setVisible (true);
```

```
        // rörelseskapare - ett objekt som kan flytta en figur
        // i det givna rummet
        Mover    mover = new Mover (rum);

        // flytta partikeln inuti rummet
        mover.move ();
    }
}
```

Programmets utmatning

Ett rum, och en partikel som rör sig i rummet, visas. Partikeln kommer in i rummet vid rummets vänstra kant och rör sig mot rummets högra kant. Partikeln lämnar rummet vid dess högra kant.

Definiera en rörlig figur

Drawable.java

Ett gränssnitt som representerar en figur som kan ritas

Gränssnittet `Drawable` representerar en figur som kan ritas med en given grafisk kontext.

```
import java.awt.*;  // Graphics2D

public interface Drawable
{
    // rita figuren med den givna grafiska kontexten
    void draw (Graphics2D g);
}
```

Movable.java

Ett gränssnitt som representerar en rörlig figur

Gränssnittet `Movable` representerar en figur som kan röra sig.

```
public interface Movable
{
    // Låt figuren röra sig till en ny position.
    // Metoden returnerar true om figuren ännu inte nått sin
    // destination, annars false.
    boolean move ();
}
```

Partikel.java

En klass som representerar en partikel som kan visa (rita) sig och som kan röra sig i ett givet rum

Klassen `Partikel` representerar en partikel i ett givet rum.

En partikel kan visa (rita) sig på den aktuella platsen. En partikel kan röra sig (linjärt) till en annan plats, så länge den hamnar inuti det givna rumet.

```
import java.awt.*;       // Graphics2D, Paint
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JComponent

class Partikel implements Drawable, Movable
{
    // rummets dimensioner
    private int    w;  // rummets bredd
    private int    h;  // rummets höjd

    // partikelns fyllnadsmönster
    private Paint    fyllnadsmonster;

    // partikelns dimensioner
    private int    xsize;  // partikelns bredd
    private int    ysize;  // partikelns höjd

    // partikelns position i rummet
    private int    x;  // x-koordinat
    private int    y;  // y-koordinat

    // partikelns steg
    private int    dx;  // partikelns steg i x-riktning
    private int    dy;  // partikelns steg i y-riktning

    // initiera rummets dimensioner och partikelns fyllnadsmönster
    // och storlek, partikelns plats i rummet och partikelns steg
    public Partikel (JComponent rum, Paint fyllnadsmonster)
    {
        this.w = rum.getWidth ();
        this.h = rum.getHeight ();

        this.fyllnadsmonster = fyllnadsmonster;
        this.xsize = 12;
        this.ysize = 8;

        this.x = -xsize;
        this.y = (int) (9 * h * Math.random () / 10) + 1;
```

```
        this.dx = 1;
        this.dy = 1;
    }

    // ange partikelns bredd och höjd
    public void setSize (int xsize, int ysize)
    {
        this.xsize = xsize;
        this.ysize = ysize;
    }

    // rita partikeln
    public void draw (Graphics2D g)
    {
        Ellipse2D    figur =
                new Ellipse2D.Double (x, y, xsize, ysize);
        g.setPaint (fyllnadsmonster);
        g.fill (figur);
    }

    // Flytta partikeln till en ny plats.
    // Returnera true om partikeln fortfarande
    // finns i rummet, annars false.
    public boolean move ()
    {
        // modifiera partikelns position
        x += dx;
        y += dy;

        // ändra riktning vid rummets nedre och övre kant
        if (y == h - ysize  ||  y == 0)
            dy = -dy;

        // partikeln lämnar rummet vid rummets högra kant
        boolean    inutiRummet = true;
        if (x > w)
            inutiRummet = false;

        return inutiRummet;
    }
}
```

Rum.java

En klass som representerar ett rum för en figur

Klassen `Rum` representerar ett rum som en figur kan röra sig i. Olika figurer kan placeras i rummet vid olika tidpunkter.

```
import java.awt.*;       // Color, Graphics, Graphics2D
import javax.swing.*;    // JPanel

class Rum extends JPanel
{
    // en figur som kan ritas
    private Drawable    figur;

    // initiera rummet
    public Rum ()
    {
        this.setBackground (Color.PINK);
        figur = null;
    }

    // rita figuren
    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        if (figur != null)
            figur.draw (g);
    }

    // placera en figur i rummet
    public void setDrawable (Drawable figur)
    {
        this.figur = figur;
    }
}
```

Mover.java

En klass som flyttar en given figur i ett givet rum

En figur kan ritas på flera successiva positioner i ett givet rum, Närliggande positioner och ett kort tidsintervall mellan successiva ritningar kan väljas. På så sätt skapas ett intryck av en figur som rör sig.

```
class Mover
{
     // ett rum där en figur kan röra sig
    private Rum    rum;

    // initiera rummet
    public Mover (Rum rum)
    {
        this.rum = rum;
    }

    // flytta den givna figuren i rummet
    public void move (Movable figur)
    {
        // flytta figuren så länge den inte når sin destination
        boolean    inteNattDestination = figur.move ();
        while (inteNattDestination)
        {
            // visa (rita) figuren på den aktuella platsen
            // i rummet
            rum.repaint ();

            // vänta en stund
            try
            {
                Thread.sleep (4);
            }
            catch (InterruptedException e)
            {}

            // flytta figuren till en ny position,
            // och erhåll information om figuren
            // nått sin destination
            inteNattDestination = figur.move ();
        }
    }
}
```

EnRorligFigur.java

Ett program som visar hur en rörlig figur definieras och används

En figur, som kan röra sig inuti ett givet rum, kan definieras och användas.

```
import java.awt.*;     // Color
```

```
import javax.swing.*;  // JFrame

// EnRorligFigur är ett program som visar ett fönster
// som innehåller ett rum med en rörlig figur.
class EnRorligFigur
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En rörlig figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // ett rum där en figur kan röra sig,
        Rum    rum = new Rum ();
        frame.add (rum);

        // visa fönstret
        frame.setVisible (true);

        // rörelseskapare - ett objekt som kan flytta
        // olika figurer i det givna rummet, en figur i taget
        Mover    mover = new Mover (rum);

        // färger att måla figurer med
        Color[]   farger = {Color.BLACK, Color.WHITE,
                            Color.BLUE, Color.RED};
        int    k = 0;  // index i vektorn

        // skapa och flytta olika partiklar, en partikel i taget
        while (true)
        {
            // en partikel som kan röra sig i det givna rummet
            Partikel    partikel = new Partikel (rum, farger[k]);

            // placera partikeln i rummet
            rum.setDrawable (partikel);

            // en annan färg för nästa partikel
            k = (k + 1) % farger.length;

            // flytta partikeln inuti rummet
            mover.move (partikel);
        }
    }
}
```

Programmets utmatning

Ett rum, och en partikel som rör sig i rummet, visas.

En partikel kommer in i rummet vid rummets vänstra kant och rör sig mot rummets högra kant. En partikel lämnar rummet vid dess högra kant. När en partikel lämnar rummet, kommer en ny partikel in i rummet.

Flera rörliga figurer

Drawable.java

Ett gränssnitt som representerar en figur som kan ritas

Gränssnittet `Drawable` representerar en figur som kan ritas med en given grafisk kontext.

```
import java.awt.*;  // Graphics2D

public interface Drawable
{
    // rita figuren med den givna grafiska kontexten
    void draw (Graphics2D g);
}
```

Movable.java

Ett gränssnitt som representerar en rörlig figur

Gränssnittet `Movable` representerar en figur som kan röra sig.

```
public interface Movable
{
    // Låt figuren röra sig till en ny position.
    // Metoden returnerar true om figuren ännu inte nått sin
    // destination, annars false.
    boolean move ();
}
```

Partikel.java

En klass som representerar en partikel som kan visa (rita) sig och som kan röra sig inuti ett givet rum

Klassen `Partikel` representerar en partikel i ett givet rum.

En partikel kan visa (rita) sig på den aktuella platsen. En partikel kan röra sig (linjärt) till en annan plats, så länge den förblir i det givna rummet.

```
import java.awt.*;       // Graphics2D, Paint
import java.awt.geom.*;  // Ellipse2D, Ellipse2D.Double
import javax.swing.*;    // JComponent

class Partikel implements Drawable, Movable
{
    // rummets dimensioner
    private int    w;  // rummets bredd
    private int    h;  // rummets höjd

    // partikelns fyllnadsmönster
    private Paint    fyllnadsmonster;

    // partikelns dimensioner
    private int    xsize;  // partikelns bredd
    private int    ysize;  // partikelns höjd

    // partikelns position i rummet
    private int    x;  // x-koordinat
    private int    y;  // y-koordinat

    // partikelns steg
    private int    dx;  // partikelns steg i x-riktning
    private int    dy;  // partikelns steg i y-riktning

    // initiera rummets dimensioneroch partikelns fyllnadsmönster
    // och storlek, partikelns plats i rummet och partikelns steg
    public Partikel (JComponent rum, Paint fyllnadsmonster)
    {
        this.w = rum.getWidth ();
        this.h = rum.getHeight ();

        this.fyllnadsmonster = fyllnadsmonster;
        this.xsize = 12;
        this.ysize = 8;

        this.x = -xsize;
        this.y = (int) (9 * h * Math.random () / 10) + 1;
```

```
        this.dx = 1;
        this.dy = 1;
    }

    // ange partikelns bredd och höjd
    public void setSize (int xsize, int ysize)
    {
        this.xsize = xsize;
        this.ysize = ysize;
    }

    // rita partikeln
    public void draw (Graphics2D g)
    {
        Ellipse2D  figur =
               new Ellipse2D.Double (x, y, xsize, ysize);
        g.setPaint (fyllnadsmonster);
        g.fill (figur);
    }

    // Flytta partikeln till en ny plats.
    // Returnera true om partikeln fortfarande
    // finns i rummet, annars false.
    public boolean move ()
    {
        // modifiera partikelns position
        x += dx;
        y += dy;

        // ändra riktning vid rummets nedre och övre kant
        if (y == h - ysize  ||  y == 0)
            dy = -dy;

        // partikeln lämnar rummet vid rummets högra kant
        boolean    inutiRummet = true;
        if (x > w)
            inutiRummet = false;

        return inutiRummet;
    }
}
```

Rum.java

En klass som representerar ett rum för olika figurer

Klassen `Rum` representerar ett rum som olika figurerer kan röra sig i. En figur kan läggas till i rummet eller tas bort från rummet. Rummet kan också rensas från figurerna.

```
import java.util.*;      // ArrayList
import java.awt.*;       // Color, Graphics, Graphics2D
import javax.swing.*;    // JPanel

class Rum extends JPanel
{
    // rummets figurer
    private ArrayList<Drawable>    figurer;

    // skapa ett rum
    public Rum ()
    {
        this.setBackground (Color.PINK);
        figurer = new ArrayList<Drawable> ();
    }

    // visa figurer i rummet
    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        // rita alla figurer - en figur kan inte läggas till eller
        // tas bort under tiden (blocket är synkroniserat)
        synchronized (this)
        {
            int    antalFigurer = figurer.size ();
            Drawable    figur = null;
            for (int i = 0; i < antalFigurer; i++)
            {
                figur = figurer.get (i);
                figur.draw (g);
            }
        }
    }

    // lägg till en figur i rummet
    public synchronized void addDrawable (Drawable figur)
    {
        figurer.add (figur);
    }
```

```
    // ta bort en figur från rummet
    public synchronized void removeDrawable (Drawable figur)
    {
        figurer.remove (figur);
    }


    // rensa rummet
    public synchronized void clearDrawables ()
    {
        figurer.clear ();
    }
```

Mover.java

En klass som flyttar en given figur inuti ett givet rum

En figur kan ritas på flera successiva positioner inuti ett givet rum, Närliggande positioner och ett kort tidsintervall mellan successiva ritningar kan väljas. På så sätt skapas ett intryck av en figur som rör sig.

```
class Mover<T extends Drawable & Movable> implements Runnable
{
     // ett rum där en figur kan röra sig
    private Rum    rum;

    // en figur som kan röra sig
    private T    figur;

    // initiera rummet och figuren
    public Mover (Rum rum, T figur)
    {
        this.rum = rum;
        this.figur = figur;
    }

    // flytta den givna figuren inuti rummet
    public void move ()
    {
        // flytta figuren så länge den inte når sin destination
        boolean    inteNattDestination = figur.move ();
        while (inteNattDestination)
        {
            // visa (rita) figuren på den aktuella positionen
            // i rummet
```

```
            rum.repaint ();

            // vänta en stund
            try
            {
                Thread.sleep (4);
            }
            catch (InterruptedException e)
            {}

            // flytta figuren till en ny position,
            // och erhåll information om figuren nått
            // sin destination
            inteNattDestination = figur.move ();
        }

        // figuren har lämnat rummet
        rum.removeDrawable (figur);
    }

    // att exekveras som en separat tråd
    public void run ()
    {
        this.move ();
    }
}
```

RorligaFigurer.java

Ett program med flera rörliga figurer

Flera figurer kan röra sig samtidigt i ett och samma rum.

```
import java.awt.*;     // Color
import javax.swing.*;  // JFrame

// RorligaFigurer är ett program som visar ett fönster
// som innehåller ett rum med flera figurer som rör sig
class RorligaFigurer
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Rörliga figurer");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // ett rum där olika figurer kan röra sig
```

```
        Rum    rum = new Rum ();
        frame.add (rum);

        // visa fönstret
        frame.setVisible (true);

        // färger att måla partiklar med
        Color[]   farger = {Color.BLACK, Color.WHITE,
                            Color.BLUE, Color.RED};
        int    k = 0;  // index i vektorn
        while (true)
        {
            // skapa en partikel och placera den i rummet
            Partikel    partikel = new Partikel (rum, farger[k]);
            rum.addDrawable (partikel);

            // en annan färg för nästa partikel
            k = (k + 1) % farger.length;

            // rör partikeln i rumet
            Mover<Partikel>    mover =
                     new Mover<Partikel> (rum, partikel);
            new Thread (mover).start ();

            // vänta en stund
            try
            {
                Thread.sleep (2000);
            }
            catch (InterruptedException e)
            {}
        }
    }
}
```

Programmets utmatning

Ett rum, och flera partiklar som rör sig i rummet, visas.

En partikel kommer in i rummet vid rummets vänstra kant och rör sig mot rummets högra kant. En partikel lämnar rummet vid dess högra kant. Partiklarna kommer in i rummet med regelbundna tidsfördröjningar.

Koordinatbyte

Komponentkoordinater och användarkoordinater

EnEgenSkala.java

Ett program som använder en egen skala för koordinatsystemet

Man kan definiera sina figurer i ett koordinatsystem, som använder en egen skala (egna enheter används - till exempel centimeter, istället för antalet pixlar). När en sådan figur sedan ritas, måste två skalfaktorer anges som preciserar hur de egna enheterna avbildas till antalet pixlar (hur många pixlar det blir per egen enhet).

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // AffineTransform
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som innehåller rektanglar
// definierade enligt en egen skala
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("EN EGEN SKALA", 240, 20);

        // använd pixlar först, därefter egna enheter och sedan
        // pixlar på nytt

        // använd den ursprungliga skalan (med enheter i antalet
        // pixlar)
        Rectangle2D    rekt1 = new Rectangle2D.Double (100, 50,
                                                       120, 80);
        BasicStroke    stroke0 = new BasicStroke (2.0F);
```

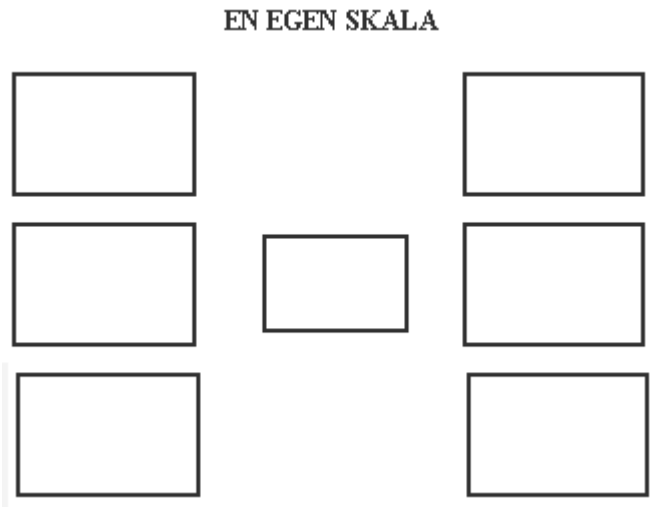
```
g.setStroke (stroke0);
g.draw (rekt1);

// använd en egen skala för att definiera en rektangel,
// och motsvarande linjetjocklek
Rectangle2D    rekt2 = new Rectangle2D.Double (10, 15,
                                               12, 8);
// 10, 15, 12 och 8 egna enheter (till exempel centimeter)
BasicStroke    stroke = new BasicStroke (0.2F);
// 0.2 egna enheter (till exempel centimeter)
g.setStroke (stroke);

// bestäm hur de egna enheterna översätts  till antalet
// pixlar vid ritningen
g.scale (10,   // skalfaktorn för x-koordinater
         10);  // skalfaktorn för y-koordinater
// dessa skalfaktorer gäller nu

// rita rektangeln
g.draw (rekt2);
// en rektangel vars övre vänstra hörn ligger
// i punkten (10 * 10 pixlar, 15 * 10 pixlar), och vars
// dimensioner är 12 * 10 pixlar och 8 * 10 pixlar

// definiera en rektangel i det ursprungliga koordinat-
// systemet
Rectangle2D    rekt3 = new Rectangle2D.Double (100, 250,
                                               120, 80);
g.setStroke (stroke0);

// återgå till de ursprungliga skalfaktorerna
g.scale (1.0 / 10, 1.0 / 10);
g.draw (rekt3);

// använd egna enheter
// (tänk på panelens storlek vid transformationen)

// bestäm skalfaktorerna relativt dimensionerna för
// den komponent som det ritas i (den här panelen)
int    w = this.getWidth ();   // panelens bredd i pixlar
int    h = this.getHeight ();  // panelens höjd i pixlar
int    minDimension = Math.min (w, h);
// definiera en rektangel enligt en egen skala
Rectangle2D    rekt4 = new Rectangle2D.Double (34, 20,
                                               12, 8);
g.setStroke (stroke);
// ange skalfaktorerna
g.scale (minDimension / 50.0, minDimension / 50.0);
g.draw (rekt4);

// återgå till de gamla skalfaktorerna
```

```
        g.scale (50.0 / minDimension, 50.0 / minDimension);


        // använd egna enheter först, därefter pixlar och
        // sedan åter egna enheter

        // lagra den uppsättning av transformationsfaktorer
        // som gäller just nu (den gällande transformationen),
        // så att det blir lätt att återgå till den
        AffineTransform    at1 = g.getTransform ();

        // definiera en rektangel enligt en egen skala
        Rectangle2D    rekt5 = new Rectangle2D.Double (40, 5,
                                                       12, 8);
        g.setStroke (stroke);

        // definiera en ny uppsättning skalfaktorer
        // (en elementär transformation)
        AffineTransform  at2 =
                   AffineTransform.getScaleInstance (10, 10);
        // lägg till den nya uppsättningen skalfaktorer till
        // den gällande transformationen
        g.transform (at2);
        // rita rektangeln (applicera de angivna skalfaktorerna)
        g.draw (rekt5);

        // definiera en rektangel enligt den ursprungliga skalan
        Rectangle2D    rekt6 = new Rectangle2D.Double (400, 150,
                                                       120, 80);
        g.setStroke (stroke0);

        // återgå till de ursprungliga skalfaktorerna
        g.setTransform (at1);
        g.draw (rekt6);

        // definiera en rektangel enligt en egen skala
        Rectangle2D    rekt7 = new Rectangle2D.Double (40, 25,
                                                       12, 8);
        g.setStroke (stroke);

        // ändra skalfaktorerna en gång till
        g.setTransform (at2);
        g.draw (rekt7);
    }
}

// EnEgenSkala är ett program som visar ett fönster som innehåller
// en panel med rektanglar definierade enligt en egen skala.
class EnEgenSkala
{
    public static void main (String[] args)
```

```
    {
        // ett fönster
        JFrame    frame = new JFrame (" En egen skala");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Det visas tre likadana rektanglar (i vertikal ordning) till vänster, en rektangel i mitten och tre likadana rektanglar (i vertikal ordning) till höger.



Förflytta en figur

ForflyttaEnFigur.java

Ett program som förflyttar flera figurer

Man kan definiera en figur i ett koordinatsystem, och sedan förflytta detta koordinatsystem inuti den komponent där figuren ritas. På så sätt förflyttas även figuren inuti komponenten.

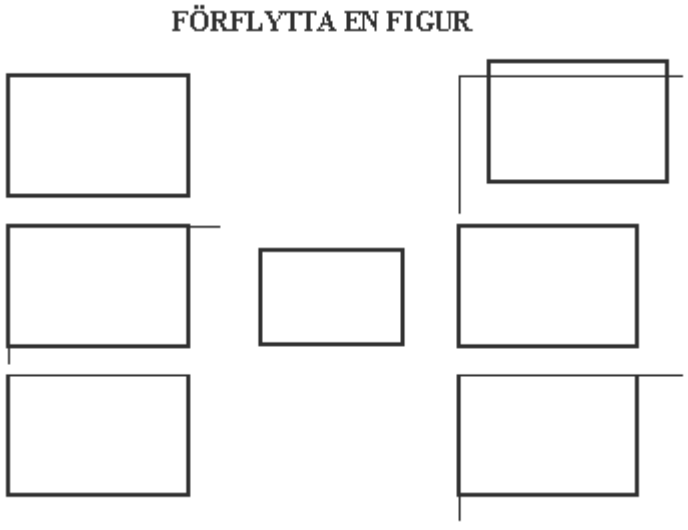
```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // AffineTransform
import javax.swing.*;    // JPanel, JFrame

// ForflyttaEnFigur är en panel, som flera figurer förflyttas i
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("FÖRFLYTTA EN FIGUR", 210, 20);

        // linje att rita figurer med
        BasicStroke    stroke0 = new BasicStroke (2.0F);

        // linje för att rita koordinatsystem
        BasicStroke    stroke1 = new BasicStroke (1.0F);

        // rita ett eget koordinatsystem
        g.setStroke (stroke1);
        g.draw (new Line2D.Double (100, 150, 240, 150));
        g.draw (new Line2D.Double (100, 150, 100, 240));

        g.draw (new Line2D.Double (400, 50, 550, 50));
        g.draw (new Line2D.Double (400, 50, 400, 140));

        g.draw (new Line2D.Double (400, 250, 550, 250));
        g.draw (new Line2D.Double (400, 250, 400, 345));
```

```
// använd det ursprungliga koordinatsystemet,
// förflytta sedan koordinatsystemet till en punkt
// i panelen och återgå därefter till det
// ursprungliga koordinatsystemet

// använd det ursprungliga koordinatsystemet
Rectangle2D    rekt1 = new Rectangle2D.Double (100, 50,
                                               120, 80);
g.setStroke (stroke0);
g.draw (rekt1);

// definiera en rektangel
Rectangle2D    rekt2 = new Rectangle2D.Double (0, 0,
                                               120, 80);
// punkten (0, 0) gäller i egna koordinatsystemet

// bestäm den riktiga platsen för origo (i antalet pixlar)
// vid ritningen
g.translate (100,   // x-koordinat för egen origo blir 100
             150);  // y-koordinat för egen origo blir 150
// dessa förflyttningsparametrar gäller nu

// rita rektangeln
g.draw (rekt2);
// en rektangel ritas, vars övre vänstra hörn ligger
// i punkten (0 + 100 pixlar, 0 + 150 pixlar), och vars
// dimensioner är 120 pixlar x 80 pixlar

// definiera en rektangel
Rectangle2D    rekt3 = new Rectangle2D.Double (100, 250,
                                               120, 80);
// återgå till de ursprungliga förflyttningsparametrarna
// (till den ursprungliga transformationen)
g.translate (-100, -150);
g.draw (rekt3);

// flytta koordinatsystemet och använd en egen skala
// (tänk på panelens storlek)

// positionera origo relativt den komponent
// där figurer ritas (relativt den här panelen)
int    w = this.getWidth ();   // panelens bredd i pixlar
int    h = this.getHeight ();  // panelens höjd i pixlar
int    minDimension = Math.min (w, h);
// definiera en rektangel i ett koordinatsystem
// med en egen skala
Rectangle2D    rekt4 = new Rectangle2D.Double (0, 0,
                                               12, 8);
BasicStroke    stroke = new BasicStroke (0.2F);
g.setStroke (stroke);
// ange förflyttningsparametrarna (positionera origo)
```

```
// (det måste göras innan nya skalfaktorer anges,
// eftersom förflyttningen anges i pixlar)
g.translate (w / 2.0 - 6 * minDimension / 50.0,
             h / 2.0 - 4 * minDimension / 50.0);
// tilldela skalfaktorerna
g.scale (minDimension / 50.0, minDimension / 50.0);
g.draw (rekt4);

// återgå till de gamla skalfaktorerna
g.scale (50.0 / minDimension, 50.0 / minDimension);
// återgå till de gamla förflyttningsparametrarna
g.translate (-(w / 2.0 - 6 * minDimension / 50.0),
             -(h / 2.0 - 4 * minDimension / 50.0));


// förflytta koordinatsystemet till en punkt
// inuti panelen, återgå därefter till det
// ursprungliga koordinatsystemet och
// förflytta koordinatsystemet en gång till

// lagra den uppsättning av transformationsparametrar
// (där finns även skalfaktorerna
// och förflyttningsparametrarna)
// som gäller just nu (så att det går lätt att återgå
// till den)
AffineTransform    at0 = g.getTransform ();

// definiera en rektangel
Rectangle2D    rekt5 = new Rectangle2D.Double (20, -10,
                                               120, 80);
g.setStroke (stroke0);

// definiera en uppsättning av förflyttningsparametrar
AffineTransform    at1 =
          AffineTransform.getTranslateInstance(400, 50);
// lägg till den nya uppsättningen
// av förflyttningsparametrar
g.transform (at1);
// rita rektangeln (applicera de angivna
// förflyttningsparametrarna)
g.draw (rekt5);

// definiera en rektangel
Rectangle2D    rekt6 = new Rectangle2D.Double (400, 150,
                                               120, 80);
// återgå till de ursprungliga transformationsparametrarna
// (till den ursprungliga transformationen)
g.setTransform (at0);
g.draw (rekt6);

// definiera en rektangel
```

```
        Rectangle2D    rekt7 = new Rectangle2D.Double (0, 0,
                                                       120, 80);
        // ändra transformationsparametrarna en gång till
        AffineTransform    at2 =
              AffineTransform.getTranslateInstance (400, 250);
        g.transform (at2);
        g.draw (rekt7);
    }
}

// ForflyttaEnFigur är ett program som visar ett fönster
// som innehåller en panel där flera figurer förflyttas
class ForflyttaEnFigur
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
        JFrame    frame = new JFrame (" Förflytta en figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        RitPanel    panel = new RitPanel ();
        frame.add (panel);
        frame.setVisible (true);
    }
}
```

Programmets utmatning



Rotera en figur

RoteraEnFigur.java

Ett program som roterar flera figurer

Ett koordinatsystem (och därmed alla figurer som är definierade i det) kan roteras med en given vinkel.

En figur kan roteras kring en given punkt, med en given vinkel.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // AffineTransform
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som flera figurer roteras i
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("ROTERA EN FIGUR", 225, 20);

        // linjetjocklek att rita figurer med
        BasicStroke    stroke0 = new BasicStroke (2.0F);
        g.setStroke (stroke0);

        // använd det ursprungliga koordinatsystemet, rotera sedan
        // koordinatsystemet med en given vinkel och
        // återgå därefter till det ursprungliga koordinatsystemet

        // en rektangel
        Rectangle2D    rekt1 = new Rectangle2D.Double (100, 50,
                                                       120, 80);
        g.draw (rekt1);

        // definiera en rektangel
        Rectangle2D    rekt2 = new Rectangle2D.Double (100, 150,
                                                       120, 80);
        // bestäm hur axlarna placeras vid ritningen - rotera
```

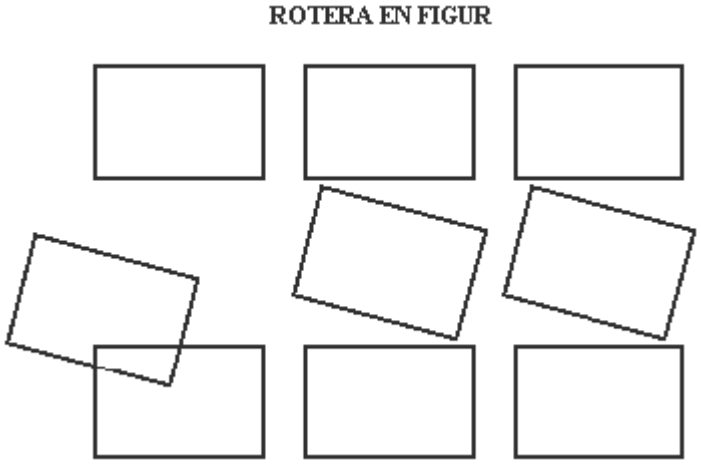
```
// dem kring punkten (0, 0)
g.rotate (
    Math.PI / 12);  // rotationsvinkel
// en ny transformation läggs till till den gällande
// transformationen (en komposition av
// transformationer bildas)

// den nya transformationen gäller nu

// rita rektangeln i det roterade koordinatsystemet
// (rektangeln roteras relativt det ursprungliga
// koordinatsystemet)
g.draw (rekt2);

// definiera en rektangel
Rectangle2D    rekt3 = new Rectangle2D.Double (100, 250,
                                               120, 80);
// återgå till den ursprungliga transformationen
g.rotate (-Math.PI / 12);
g.draw (rekt3);


// använd det ursprungliga koordinatsystemet, rotera sedan
// en figur kring en given punkt och återgå därefter
// till det ursprungliga koordinatsystemet

// en rektangel
Rectangle2D    rekt4 = new Rectangle2D.Double (250, 50,
                                               120, 80);
g.draw (rekt4);

// definiera en rektangel
Rectangle2D    rekt5 = new Rectangle2D.Double (250, 150,
                                               120, 80);
// rotera en figur kring en given punkt
// (förflytta figuren så att en given punkt
// sammanfaller med origo, rotera koordinatsystemet
// och flytta tillbaka figuren)

g.rotate (Math.PI / 12,  // rotationsvinkel (i radianer)
          310, 190);     // rotationspunkt
                         // (rektangelns mittpunkt)
// en ny transformation läggs till till den gällande
// transformationen (en komposition av
// transformationer bildas)

// den nya transformationen gäller nu

// rita rektangeln
g.draw (rekt5);
```

```
        // definiera en rektangel
        Rectangle2D    rekt6 = new Rectangle2D.Double (250, 250,
                                                       120, 80);
        // återgå till den ursprungliga transformationen
        g.rotate (-Math.PI / 12, 310, 190);
        g.draw (rekt6);


        // använd det ursprungliga koordinatsystemet, rotera
        // sedan en figur kring en given punkt och återgå
        // därefter till det ursprungliga koordinatsystemet

        // en rektangel
        Rectangle2D    rekt7 = new Rectangle2D.Double (400, 50,
                                                       120, 80);
        g.draw (rekt7);


        // definiera en rektangel
        Rectangle2D    rekt8 = new Rectangle2D.Double (400, 150,
                                                       120, 80);

        // lagra den gällande transformationen
        // (så att det går lätt att komma tillbaka till den)
        AffineTransform    at0 = g.getTransform ();

        // rotera en figur kring en given punkt
        // (förflytta figuren så att en given punkt
        // sammanfaller med origo, rotera koordinatsystemet
        // och flytta tillbaka figuren)

        AffineTransform  at1 = AffineTransform.getRotateInstance (
                                        Math.PI / 12, 460, 190);
        g.transform (at1);
        // transformationen at1 läggs till till den gällande
        // transformationen at0
        // (en komposition av transformationer bildas)

        // den nya transformationen gäller nu

        // rita rektangeln
        g.draw (rekt8);

        // definiera en rektangel
        Rectangle2D    rekt9 = new Rectangle2D.Double (400, 250,
                                                       120, 80);
        // återgå till den ursprungliga transformationen
        g.setTransform (at0);
        g.draw (rekt9);
    }
}
```

```
// RoteraEnFigur är ett program som visar ett fönster
// som innehåller en panel som flera figurer roteras i
class RoteraEnFigur
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Rotera en figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Tre rektanglar (vertikalt ordnade, den i mitten är roterad) visas till vänster, tre rektanglar (vertikalt ordnade, den i mitten är roterad) i mitten och tre rektanglar (vertikalt ordnade, den i mitten är roterad) till höger.



Skjuva en figur

SkjuvaEnFigur.java

Ett program som skjuvar flera figurer

En figur kan skjuvas, i antingen x-riktningen, y-riktningen eller i båda riktningarna.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // AffineTransform
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel som flera figurer skjuvas i
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("SKJUVA EN FIGUR", 230, 20);

        // linjetjocklek att rita figurer med
        BasicStroke    stroke0 = new BasicStroke (2.0F);
        g.setStroke (stroke0);

        // använd det ursprungliga koordinatsystemet, skjuva sedan
        // en figur i x-riktningen och återgå därefter till det
        // ursprungliga koordinatsystemet

        // en rektangel
        Rectangle2D    rekt1 = new Rectangle2D.Double (100, 50,
                                                       120, 80);
        g.draw (rekt1);

        // definiera en rektangel
        Rectangle2D    rekt2 = new Rectangle2D.Double (100, 150,
                                                       120, 80);
        // lägg till en skjuvning
        g.shear (
```

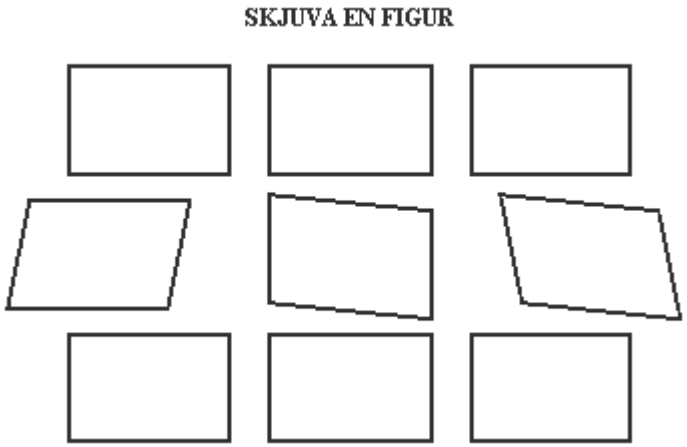
```
    -0.2,  // skjuvning i x-riktningen
           // (x avbildas till x - 0.2 * y)
    0);    // skjuvning i y-riktningen
           // (y avbildas till y + 0 * x);
// en ny transformation läggs till till den gällande
// transformationen (en komposition av
// transformationer bildas)

// den nya transformationen gäller nu

// rita rektangeln
g.draw (rekt2);

// definiera en rektangel
Rectangle2D    rekt3 = new Rectangle2D.Double (100, 250,
                                               120, 80);
// återgå till den ursprungliga transformationen
g.shear (0.2, 0);
g.draw (rekt3);


// använd det ursprungliga koordinatsystemet, skjuva sedan
// en figur i y-riktningen, och återgå därefter till det
// ursprungliga koordinatsystemet

// en rektangel
Rectangle2D    rekt4 = new Rectangle2D.Double (250, 50,
                                               120, 80);
g.draw (rekt4);

// definiera en rektangel
Rectangle2D    rekt5 = new Rectangle2D.Double (250, 120,
                                               120, 80);
// inför en skjuvning
g.shear (0, 0.1);
// en ny transformation läggs till till den gällande
// transformationen (en komposition av transformationer
// bildas)

// den nya transformationen gäller nu

// rita rektangeln
g.draw (rekt5);

// definiera en rektangel
Rectangle2D    rekt6 = new Rectangle2D.Double (250, 250,
                                               120, 80);
// återgå till den ursprungliga transformationen
g.shear (0, -0.1);
g.draw (rekt6);
```

```
        // använd det ursprungliga koordinatsystemet, skjuva sedan
        // en figur både i x-riktningen och y-riktningen och
        // återgå därefter till det ursprungliga koordinatsystemet

        // en rektangel
        Rectangle2D    rekt7 = new Rectangle2D.Double (400, 50,
                                                       120, 80);
        g.draw (rekt7);

        // definiera en rektangel
        Rectangle2D    rekt8 = new Rectangle2D.Double (400, 105,
                                                      120, 80);
        // lagra den gällande transformationsen
        // (så att det lätt går att komma tillbaka till den)
        AffineTransform    at0 = g.getTransform ();

        // en skjuvningstransformation
        AffineTransform   at1 = AffineTransform.getShearInstance (
                                                         0.2, 0.1);
        g.transform (at1);
        // transformationen at1 läggs till till den gällande
        // transformationen at0
        // (en komposition av transformationer bildas)

        // den nya transformationen gäller nu

        // rita rektangeln
        g.draw (rekt8);
        // definiera en rektangel
        Rectangle2D    rekt9 = new Rectangle2D.Double (400, 250,
                                                       120, 80);
        // återgå till den ursprungliga transformationen
        g.setTransform (at0);
        g.draw (rekt9);
    }
}

// SkjuvaEnFigur är ett program som visar ett fönster
// som innehåller en panel där flera figurer skjuvas
class SkjuvaEnFigur
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Skjuva en figur");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
```

```
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Tre rektanglar visas (vertikalt ordnade, den i mitten är skjuvad) till vänster, tre rektanglar (vertialt ordnade, den i mitten är skjuvad) i mitten och tre rektanglar (vertikalt ordnade, den i mitten är skjuvad) till höger.



Komposition av transformationer

KompositionAvTransformationer.java

Ett program som binder flera koordinattransformationer

Det finns en standardtransformation, som appliceras när en figur ritas (användarkoordinater transformeras till pixlar i ritkomponenten).

En elementär transformation av koordinater kan definieras, och läggas till till den gällande transformationen. Detta kan upprepas flera gånger. På så sätt bildas en komposition av transformationer.

När en figur ritas, appliceras alla de elementära transformationer som man lagt till före ritningen. Transformationerna appliceras i omvänd ordning relativt den som de anges i (den sist angivna transformationen appliceras först.

```
import java.awt.*;       // Graphics, Graphics2D,
                         // Color, Font, BasicStroke
import java.awt.geom.*;  // Rectangle2D, Rectangle2D.Double,
                         // AffineTransform
import javax.swing.*;    // JPanel, JFrame

// RitPanel är en panel, som flera koordinattransformationer
// kombineras i
class RitPanel extends JPanel
{
    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // rubrik
        Font    f1 = new Font ("Serif", Font.BOLD, 18);
        g.setFont (f1);
        g.drawString ("KOMPOSITION AV TRANSFORMATIONER", 140, 20);

        // linje att rita figurer med
        BasicStroke    stroke = null;

        // standardtransformation
        AffineTransform    at0 = g.getTransform ();

        // en komposition av transformationer

        // en rektangel
        Rectangle2D    rekt1 = new Rectangle2D.Double (50, 50,
                                                       60, 40);
        stroke = new BasicStroke (1.0F);
        g.setStroke (stroke);
        g.draw (rekt1);

        // förflytta rektangeln
        g.translate (50, 80);  // lägg till en elementär
                               // transformation
        g.draw (rekt1);
```

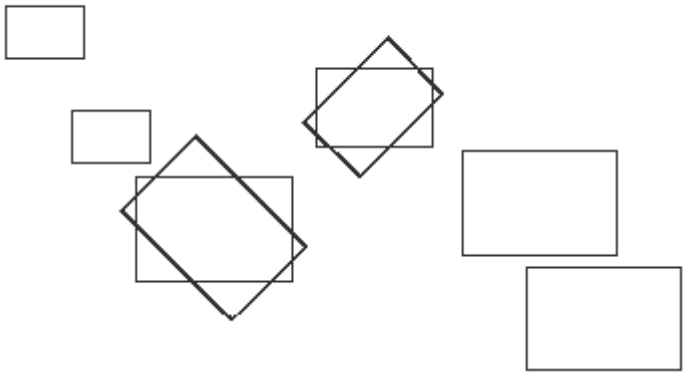
```
// förstora och förflytta rektangeln
g.scale (2, 2);        // lägg till en elementär
                       // transformation
stroke = new BasicStroke (0.5F);
g.setStroke (stroke);
g.draw (rekt1);

// rotera, förstora och förflytta  rektangeln
g.rotate (             // lägg till en elementär
                       // transformation
          Math.PI / 4,
          80, 70);  // den ursprungliga rektangelns
                    // mittpunkt (rotationen utförs först)
stroke = new BasicStroke (1.0F);
g.setStroke (stroke);
g.draw (rekt1);
// För att rita rektangeln, appliceras en transformation
// som erhålls genom att alla angivna transformationer
// länkas samman. Transformationerna appliceras i omvänd
// ordning - sist angiven transformation utförs först

// återgå till standardtransformationen
g.setTransform (at0);

// en komposition av transformationer

// en rektangel
Rectangle2D    rekt4 = new Rectangle2D.Double (450, 250,
                                                120, 80);
stroke = new BasicStroke (1.0F);
g.setStroke (stroke);
g.draw (rekt4);

// förflytta rektangeln
AffineTransform    at1 =
        AffineTransform.getTranslateInstance (-50, -90);
g.transform (at1);     // lägg till en elementär
                       // transformation
g.draw (rekt4);

// förminska och förflytta rektangeln
AffineTransform    at2 =
          AffineTransform.getScaleInstance (0.75, 0.75);
g.transform (at2);     // lägg till en elementär
                       // transformation
stroke = new BasicStroke (1F / 0.75F);
g.setStroke (stroke);
g.draw (rekt4);

// rotera, förminska och förflytta rektangeln
```

```
        AffineTransform  at3 = AffineTransform.getRotateInstance (
                                        -Math.PI / 4, 510, 290);
        g.transform (at3);     // lägg till en elementär
                               // transformation

        stroke = new BasicStroke (2F / 0.75F);
        g.setStroke (stroke);
        g.draw (rekt4);

        // En sammansatt transformation kan skapas och användas.
        // Man kan börja så här:
        // AffineTransform    at =
        //       AffineTransform.getTranslateInstance (-50, -90);
        // Objektet at kan sedan justeras med metoderna
        // setToScale, setToTranslation, setToRotation och
        // setToShear.
        // Metoden setToScale, till exempel, används så här:
        // at.setToScale (0.75, 0.75);
    }
}

// KompositionAvTransformationer är ett program som visar
// ett fönster som innehåller en panel där flera
// koordinattransformationer kombineras
class KompositionAvTransformationer
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (
                     " Komposition av transformationer");
        frame.setBounds (80, 90, 640, 420);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // en panel
        RitPanel    panel = new RitPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets utmatning

Ett antal rektanglar visas till vänster och ett antal rektanglar till höger. Först ritas en rektangel. Flera elementära transformationer (förflyttning, skalning och rotation) läggs sedan till. Rektangeln ritas på nytt efter varje ändring av transformationen. Detta upprepas på två olika sätt.

KOMPOSITION AV TRANSFORMATIONER



Kapitel 5

Grafiska användargränssnitt

Ett grafiskt användargränssnitt
Ett konsolprogram
Ett program med grafiskt användargränssnitt

Grafiska komponenter
Standardkomponenter
Komponenternas utseende
Ramar runt komponenter

Ordna komponenter i en behållare
Vanliga layoutstrategier
En flexibel layoutstrategi
Layoutkomponenter
Interna fönster

Hantera händelser
Komponenter, händelser och lyssnare
Olika sätt att implementera en lyssnarklass
Mushändelser
Tangentbordshändelser

Ett grafiskt användargränssnitt

Ett konsolprogram

HeltalsKvadraterKonsol.java

Ett program som kommunicerar med användaren via ett konsolfönster

En plattform har ett terminalfönster (konsolfönster), där användaren kan skriva olika kommandon och få olika slags information.

En plattforms konsolfönster kan användas för kommunikation mellan ett Javaprogram och dess användare. Ett program som på så sätt kommunicerar med användaren kallas för ett konsolprogram.

```
import java.io.*;    // PrintWriter
import java.util.*;  // Scanner

class HeltalsKvadraterKonsol
{
    public static void main (String[] args)
    {
        // verktyg för kommunikation med användaren via
        // plattformens konsolfönster
        PrintWriter    out = new PrintWriter (System.out, true);
        Scanner    in = new Scanner (System.in);

        // kommunicera med användaren via plattformens
        // konsolfönster
        int    heltal = 0;
        out.print ("ett heltal: ");
        out.flush ();
        String    input = in.nextLine ();
        while (!input.equals (""))
        {
            heltal = Integer.parseInt (input);
            out.println ("heltalet: " + heltal);
            out.println ("dess kvadrat: " + heltal * heltal);
            out.println ();

            out.print ("ett heltal: ");
            out.flush ();
            input = in.nextLine ();
        }

        // Vid felaktig inmatning kastas ett undantag av typen
```

```
        // java.lang.NumberFormatException. Undantaget skrivs ut
        // till standardutmatningsenheten och programmet avslutas.
    }
}
```

Programmets inmatning och utmatning

Användaren anger ett heltal via standardinmatningsenheten. Programmet hämtar heltalet, och skriver ut både heltalet och dess kvadrat till standardutmatningsenheten.

Detta upprepas tills användaren trycker på returtangenten utan att ange något annat.

Ett program med grafiskt användargränssnitt

HeltalsKvadraterGUI.java

Ett program som kommunicerar med användaren via ett grafiskt gränssnitt

Ett Javaprogram kan skapa ett grafiskt gränssnitt (GUI - Graphical User Interface) mot användaren. Ett fönster och olika grafiska komponenter skapas, och komponenterna placeras i fönstret. Komponenternas beteende definieras också (olika händelser hanteras). Ett programs grafiska gränssnitt används för kommunikation mellan användaren och programmet.

```
import javax.swing.*;       // JLabel, JTextField,
                            // JPanel, JTextArea,
                            // JFrame
import java.awt.event.*;    // ActionListener, ActionEvent

// ett objekt av den här klassen kan användas som
// en händelsehanterare
class HandelseHanterare implements ActionListener
{
    // komponenter som är nödvändiga för att kunna reagera på en
    // händelse i ett textfält
    private JTextField    textFalt;
    private JTextArea     textArea;

    // initiera textfältet och textarean
    public HandelseHanterare (JTextField textFalt,
```

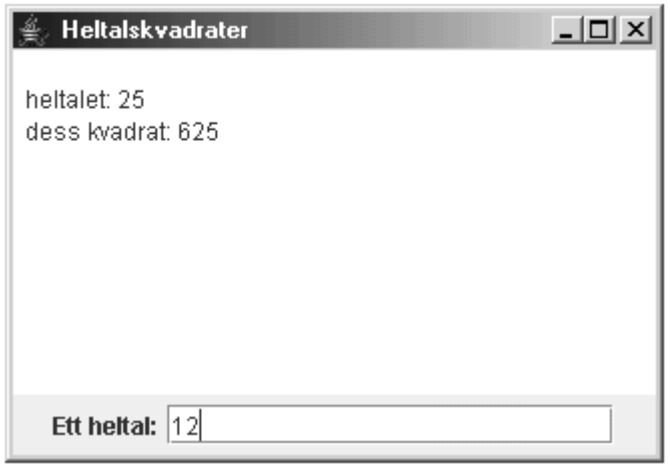
```
                                JTextArea textArea)
    {
        this.textFalt = textFalt;
        this.textArea = textArea;
    }

    // det som ska hända när användaren trycker på
    // returtangenten i textfältet
    public void actionPerformed (ActionEvent e)
    // actionPerformed är den enda metoden
    // i gränssnittet ActionListener
    {
        // hämta texten från textfältet
        String    input = textFalt.getText ();

        // rensa textfältet
        textFalt.setText ("");

        // motsvarande heltal
        int    heltal = Integer.parseInt (input);

        // rensa textarean
        textArea.setText ("");

        // skriv ut heltalet och dess kvadrat till textarean
        textArea.append ("\n  heltalet: " + heltal + "\n");
        textArea.append ("  dess kvadrat: " +
                          heltal * heltal + "\n");
    }
}

// huvudprogrammet
class HeltalsKvadraterGUI
{
    public static void main (String[] args)
    {
        // inmatningskomponenter

        // en etikett
        JLabel    label = new JLabel ("Ett heltal:");

        // ett textfält
        JTextField    textFalt = new JTextField (20);

        // en panel
        JPanel    panel = new JPanel ();

        // placera etiketten och textfältet i panelen
        // (gruppera komponenter - skapa en grafisk enhet)
        panel.add (label);
        panel.add (textFalt);
```

```
        // en utmatningskomponent (en display)

        // en textarea
        JTextArea    textArea = new JTextArea (40, 20);

        // definiera textfältets beteende (hantera textfältets
        // händelser) - skapa ett objekt som ska reagera på
        // ett lämpligt sätt när användaren trycker
        // på returtangenten i textfältet
        HandelseHanterare    hanterare =
                   new HandelseHanterare (textFalt, textArea);
        textFalt.addActionListener (hanterare);


        // ett fönster

        // ett fönster
        JFrame    frame = new JFrame (" Heltalskvadrater");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 80);

        // placera de olika komponenterna i fönstret
        frame.add (panel, "South");
        frame.add (textArea, "Center");

        // visa fönstret och dess komponenter
        frame.setVisible (true);

        // Vid felaktig inmatning kastas ett undantag av typen
        // java.lang.NumberFormatException. Undantaget skrivs ut
        // till standardutmatningsenheten (även en dialog med
        // ett felmeddelande kan användas), men programmet
        // avslutas inte (till skillnad mot ett konsolprogram).
    }
}
```

Programmets inmatning och utmatning

Användaren anger ett heltal i ett textfält och trycker på returtangenten. Programmet hämtar heltalet, och skriver i en textarea ut både heltalet och dess kvadrat.

Den angivna proceduren kan upprepas. Programmet avslutas när dess fönster stängs.



Grafiska komponenter

Standardkomponenter

GrafiskaKomponenter.java

Ett program som illustrerar flera grafiska komponenter

För att utforma ett grafiskt användargränssnitt används olika grafiska komponenter. Normalt används så kallade Swing-komponenter.

De Swing-komponenter som kan läggas i ett fönster definieras via olika subklasser till klassen `JComponent`. Klassen `JComponent` är en subklass till klassen `Container`, som i sin tur är en subklass till klassen `Component`. Tack vare en sådan organisation har alla grafiska komponenter som kan läggas i ett fönster ett antal gemensamma metoder.

```
import java.awt.*;          // Component, Container, Color, Font
import javax.swing.*;       // JComponent, JLabel,
                            // JTextField, JButton,
                            // JTextArea, JPanel, JFrame

// ett program som illustrerar flera grafiska komponenter
class GrafiskaKomponenter
{
    public static void main (String[] args)
    {
        // flera grafiska komponenter som kan placeras
        // i ett fönster

        // en vektor för grafiska komponenter
        Component[]    komponenter = new Component[5];

        // en etikett
        komponenter[0] = new JLabel ("Helheten");

        // ett textfält
        komponenter[1] = new JTextField (25);

        // en knapp
        komponenter[2] = new JButton ("Sanningen");

        // en textarea
        komponenter[3] = new JTextArea (10, 25);

        // en panel
        komponenter[4] = new JPanel ();
```

```
// ange komponenternas olika egenskaper

// ange komponenternas bakgrundsfärg och förgrundsfärg
for (int i = 0; i < komponenter.length; i++)
{
    komponenter[i].setBackground (Color.WHITE);
    komponenter[i].setForeground (Color.BLACK);
    // svart är förvalt förgrundsfärg
}

// ange komponenternas font
Font    font = new Font ("Serif", Font.BOLD, 12);
for (int i = 0; i < komponenter.length; i++)
    komponenter[i].setFont (font);

// användaren ska kunna skriva i textfältet från början
// (det förvalda beteendet - om false anges som argument
// kan användaren inte skriva i textfältet från början)
komponenter[1].setEnabled (true);

// ange panelens önskade storlek
Dimension    d = new Dimension (300, 50);
JComponent    jKomponent = (JComponent) komponenter[4];
jKomponent.setPreferredSize (d);

// låt panelen blir ogenomskinlig
// (det förvalda beteendet - om false anges som argument,
// blir panelen genomskinlig)
jKomponent.setOpaque (true);

// ange en förklarande text för panelen
// (texten visas när musen placeras i panelen)
jKomponent.setToolTipText ("en panel");

// ange textareans text
JTextArea    textArea = (JTextArea) komponenter [3];
textArea.setText ("\n\n    Balansen");

// placera komponenterna i en panel
// (gruppera komponenterna - skapa en grafisk enhet)

Container    kontainer = new JPanel ();
for (int i = 0; i < komponenter.length; i++)
    kontainer.add (komponenter[i]);

// ett fönster

JFrame    frame = new JFrame (" Grafiska komponenter");
frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
frame.setSize (400, 300);
```

```
        frame.setLocation (120, 80);

        // placera panelen (tillsammans med de komponenter
        // som finns i den) i fönstret
        frame.add (kontainer);

        // visa fönstret och dess komponenter
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller ett antal grafiska komponenter (en etikett, ett textfält, en knapp, en textarea och en panel) visas.

Komponenternas utseende

KomponenternasUtseende.java

Ett program som illustrerar komponenternas utseende (Look & Feel)

Självständiga fönster (`JWindow`, `JFrame`, `JDialog` - tungviktiga komponenter) ritas normalt av det underliggande systemet. Därför ser de ut som alla andra fönster på den aktuella plattformen. De har en plattformsberoende Look & Feel.

De Swing-komponenter som inte representerar självständiga fönster (lättviktiga komponenter) ritas av Java. En förvald Look & Feel används. Därför har dessa komponenter ett speciellt utseende - Java-utseende.

De olika komponenternas Look & Feel i ett program kan justeras (detta kan även göras dynamiskt). Den aktuella plattformens Look & Feel kan väljas för alla komponenter i ett grafiskt användargränssnitt. Även Javas förvalda Look & Feel kan väljas för alla komponenter (även för olika fönster - ramar och dialoger).

```
import java.awt.*;          // Font
import javax.swing.*;       // JLabel, JTextField,
                            // JButton, JPanel,
                            // JTextArea, JFrame, UIManager

// ett program som slumpmässigt bestämmer komponenternas utseende
// (Look & Feel)
class KomponenternasUtseende
{
    public static void main (String[] args)
    {
        // slumpmässigt utseende (Look & Feel)

        // ett slumpmässigt heltal (0, 1 eller 2)
        int    n = (int) (3 * Math.random ());

        // komponenternas utseende (Look & Feel) som
        // normalt används på den aktuella plattformen
        String   utseende =
                    UIManager.getSystemLookAndFeelClassName ();

        // om n == 0: förvalt beteende
        // (fönstret ritas enligt plattformsberoende Look & Feel,
        // andra komponenter enligt Javas förvalda Look & Feel)
```

```
if (n == 1)
// plattformsberoende Look & Feel för alla komponenter
{
    try
    {
        UIManager.setLookAndFeel (utseende);
    }
    catch (Exception e)
    {
        e.printStackTrace ();
    }
}
else if (n == 2)
// Javas förvalda Look & Feel för alla komponenter,
// även för fönstret
    JFrame.setDefaultLookAndFeelDecorated (true);

// flera grafiska komponenter
JLabel    label = new JLabel ("Ett heltal:");
JTextField    textFalt = new JTextField (20);
textFalt.setText ("" + n);
JButton    knapp = new JButton ("Exit");

// placera komponenterna i en panel
JPanel    panel = new JPanel ();
panel.add (label);
panel.add (textFalt);
panel.add (knapp);

// en textarea
JTextArea    textArea = new JTextArea (40, 20);
Font   font =
         new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
textArea.setFont (font);

// förklarande text i textarean
if (n == 0)
    textArea.setText("\n Plattform frame,\n resten Java");
else if (n == 1)
    textArea.setText ("\n    Plattform");
else
    textArea.setText ("\n    Java");


// ett fönster
JFrame    frame = new JFrame (" Komponenternas utseende");
frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
frame.setSize (400, 300);
frame.setLocation (120, 80);

// placera de olika komponenterna i fönstret
```

```
        frame.add (panel, "South");
        frame.add (textArea, "Center");

        // visa fönstret och dess komponenter
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller ett antal grafiska komponenter visas. Komponenternas utseende (Look & Feel) bestäms slumpmässigt. Kanske ritas fönstret med plattformsberoende Look & Feel, och andra komponenter med Javas förvalda Look & Feel. Det kan också hända att alla komponenter ritas med plattformsberoende Look & Feel, eller att alla komponenter (inklusive fönstret) ritas med Javas förvalda Look & Feel.

Komponenternas utseende:

eller så här:

eller så här:

Ramar runt komponenter

RamarRuntKomponenter.java

Ett program som illustrerar hur olika ramar kan placeras runt en grafisk komponent

En ram kan placeras runt en grafisk komponent. En av flera möjliga ramtyper kan väljas.

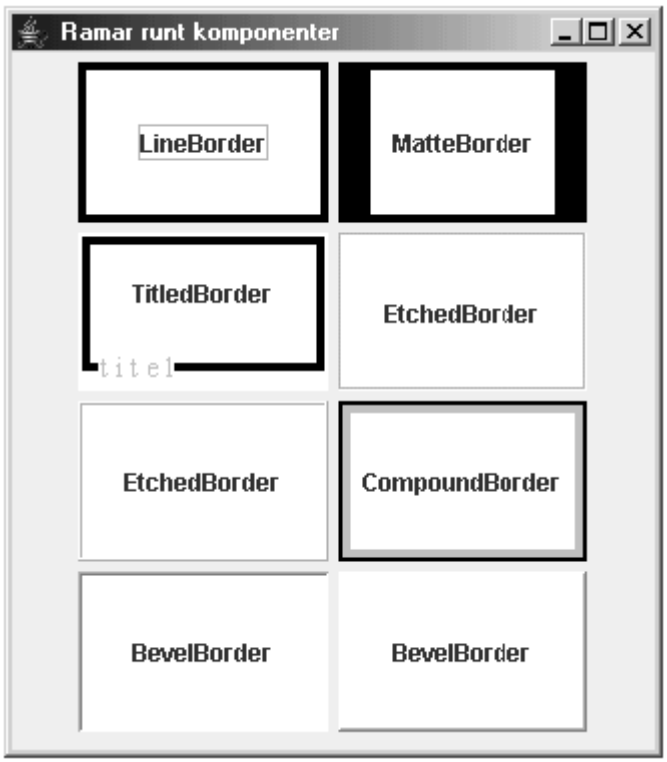
```
import java.awt.*;            // Dimension, Color, Font

import javax.swing.*;         // JButton, JPanel, JFrame
import javax.swing.border.*;  // Border, LineBorder,
                              // MatteBorder, TitledBorder,
                              // EtchedBorder, CompoundBorder,
                              // BevelBorder

// en panel som innehåller ett antal knappar med ramar
class KnappPanel extends JPanel
{
    public KnappPanel ()
    {
        // knappar
        JButton[]    knappar = new JButton[8];
        Dimension    d = new Dimension (125, 80);
        for (int i = 0; i < knappar.length; i++)
        {
            knappar[i] = new JButton ();
            knappar[i].setBackground (Color.WHITE);
            knappar[i].setPreferredSize (d);
        }

        // ramar
        Border[]    ramar = new Border[knappar.length];

        // en linjeram
        ramar[0] = new LineBorder (Color.BLACK,  // färg
                                   4);  // tjocklek i pixlar
        // om true anges som tredje argument, får ramen runda hörn

        // en färgad ram
        ramar[1] = new MatteBorder (
                           4, 16, 4, 16,  // sidornas tjocklekar
                           Color.BLACK);  // färg
        // istället för färg kan en ikon anges

        // en ram med text
        ramar[2] = new TitledBorder (
```

```
    new LineBorder (Color.BLACK, 4),     // ramens typ
    "t i t e l",                         // text i ramen
    TitledBorder.LEFT,                // textens justering
    TitledBorder.BOTTOM,              // textens position
    new Font ("Serif", Font.PLAIN, 14),  // textens font
    Color.LIGHT_GRAY);                   // textens färg
// Alla dessa argument behöver inte anges.
// Möjliga justeringar: LEFT, CENTER, RIGHT.
// Möjliga positioner:
// ABOVE_TOP, TOP, BELOW_TOP,
// ABOVE_BOTTOM, BOTTOM, BELLOW_BOTTOM

// en ram med nedsänkta sidor
ramar[3] = new EtchedBorder (EtchedBorder.LOWERED);

// en ram med upphöjda sidor
ramar[4] = new EtchedBorder (EtchedBorder.RAISED);

// en sammansatt ram
ramar[5] = new CompoundBorder (
 new LineBorder (Color.BLACK, 2),              // inre ram
new MatteBorder(4, 4, 4, 4, Color.LIGHT_GRAY));//yttre ram

// en ram som gör komponenten till en nedsänkt komponent
ramar[6] = new BevelBorder (BevelBorder.LOWERED);

// en ram som gör komponenten till en upphöjd komponent
ramar[7] = new BevelBorder (BevelBorder.RAISED);

// Istället för att använda konstruktorer för att
// skapa olika ramar, kan man använda statiska metoder
// i klassen javax.swing.BorderFactory. Dessa metoder är
// createLineBorder, createMatteBorder,
// createEtchedBorder, och så vidare.
// Så här kan detta göras:
// ramar[0] =
//       BorderFactory.createLineBorder (Color.BLACK, 4);

// ange knapparnas ramar och texter
String    klassNamn = null;
String    text = null;
for (int i = 0; i < knappar.length; i++)
{
    // ange ramen runt knappen
    knappar[i].setBorder (ramar[i]);

    // ange knappens text till ramens typ
    klassNamn = ramar[i].getClass ().getName ();
    text = klassNamn.substring (
                klassNamn.lastIndexOf ('.') + 1);
    knappar[i].setText (text);
```

```
        }

        // lägg till knapparna i panelen
        for (int i = 0; i < knappar.length; i++)
            this.add (knappar[i]);
    }
}

// ett program som placerar ramar runt grafiska komponenter
class RamarRuntKomponenter
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Ramar runt komponenter");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        KnappPanel    panel = new KnappPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller ett antal knappar visas. Runt varje knapp finns en ram. Olika typer av ramar används för olika knappar.



Ordna komponenter i en behållare

Vanliga layoutstrategier

OrdnaKomponenterIEnFoljd.java

Ett program som ordnar grafiska komponenter i en följd

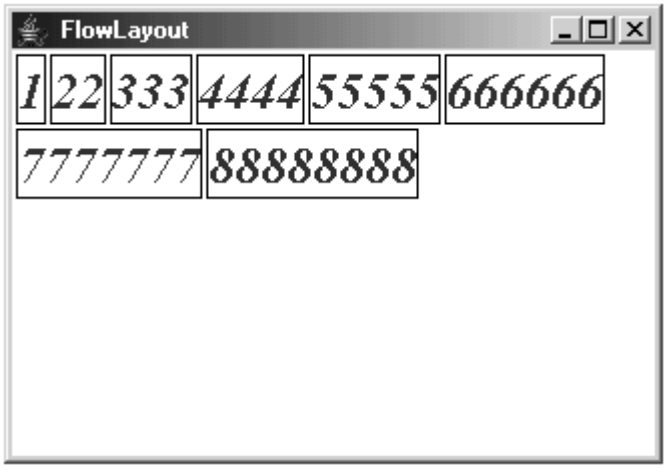
De olika grafiska komponenterna i en behållare kan ordnas i en följd. När det inte finns tillräckligt utrymme i en rad, fortsätter utplaceringen i nästa rad. Komponenterna i en rad kan justeras till vänster, i mitten (förvalt) eller till höger. När behållarens storlek ändras, omplaceras komponenterna i behållaren. Det kan därför hända att antalet rader med komponenter minskas eller ökas. Komponenternas storlek ändras inte vid ändringeen av behållarens storlek.

```
import java.awt.*;           // FlowLayout, Dimension, Color, Font
import javax.swing.*;        // JButton, JPanel,
                             // JFrame, BorderFactory
import javax.swing.border.*; // Border

// en panel som innehåller ett antal knappar ordnade i en följd
class FPanel extends JPanel
{
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // knappar

        JButton[]    knappar = new JButton[8];
        Border ram = BorderFactory.createLineBorder (Color.BLACK);
        Font   font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 25);
        // Dimension    d = new Dimension (125, 80);
        for (int i = 0; i < knappar.length; i++)
        {
            knappar[i] = new JButton ();
            knappar[i].setBackground (Color.WHITE);
            knappar[i].setFont (font);
            // knappar[i].setPreferredSize (d);
            knappar[i].setBorder (ram);
        }
        knappar[0].setText ("1");
```

```
        knappar[1].setText ("22");
        knappar[2].setText ("333");
        knappar[3].setText ("4444");
        knappar[4].setText ("55555");
        knappar[5].setText ("666666");
        knappar[6].setText ("7777777");
        knappar[7].setText ("88888888");

        // ange den här panelens layout
        FlowLayout    layout = new FlowLayout (
            FlowLayout.LEFT,  // komponenternas justering
                              // inuti en rad
            2, 2);            // avstånd mellan komponenterna
        // Avstånden kan utelämnas (i så fall blir de 0).
        // Om alla argument utelämnas, centreras komponenterna
        // i en rad

        this.setLayout (layout);
        // FlowLayout är förvald layout för en panel

        // placera knapparna i panelen
        for (int i = 0; i < knappar.length; i++)
            this.add (knappar[i]);
    }
}

// ett program som ordnar grafiska komponenter i en följd
class OrdnaKomponenterIEnFoljd
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" FlowLayout");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

I ett fönster finns en panel med ett antal knappar. Knapparna är ordnade i en följd.



OrdnaKomponenterEnligtVaderstreck.java

Ett program som ordnar grafiska komponenter efter väderstreck

De olika grafiska komponenterna i en behållare kan ordnas efter väderstreck. Komponenterna utplaceras längs behållarens fyra kanter (nord, syd, väst och öst) och i mitten (center). När en komponent placeras i en behållare, preciseras platsen för komponenten.

Man behöver inte använda alla fem positionerna. Om en av positionerna inte används, tas motsvarande utrymme av de komponenter som finns på andra positioner.

Om behållarens storlek ändras, ändras även storleken på motsvarande komponenter. Den översta, den mittersta och den nedersta komponenten ändras när behållarens bredd ändras. Den vänstra, den mittersta och den högra komponenten ändras när behållarens höjd ändras.

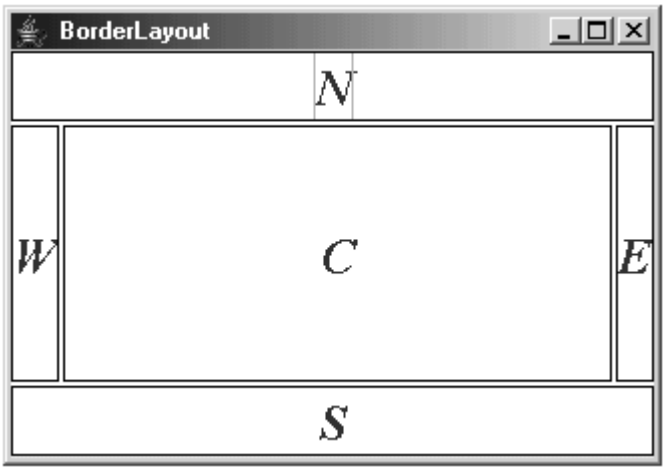
```
import java.awt.*;            // BorderLayout, Color, Font
import javax.swing.*;         // JButton, JPanel,
                              // JFrame, BorderFactory
import javax.swing.border.*;  // Border

// en panel som innehåller ett antal knappar
// ordnade efter väderstreck
class FPanel extends JPanel
```

```
{
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // knappar

        JButton[]    knappar = new JButton[5];
        Border ram = BorderFactory.createLineBorder (Color.BLACK);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 25);
        for (int i = 0; i < knappar.length; i++)
        {
            knappar[i] = new JButton ();
            knappar[i].setBackground (Color.WHITE);
            knappar[i].setFont (font);
            knappar[i].setBorder (ram);
        }
        knappar[0].setText ("N");
        knappar[1].setText ("E");
        knappar[2].setText ("S");
        knappar[3].setText ("W");
        knappar[4].setText ("C");

        // ange den här panelens layout
        BorderLayout    layout = new BorderLayout (
                           2, 2);  // avstånd mellan komponenterna
        // Avstånden kan utelämnas (i så fall blir de 0).
        // BorderLayout är förvald layout för självständiga
        // fönster.

        this.setLayout (layout);
        // lägg knapparna i panelen
        this.add (knappar[0], BorderLayout.NORTH);
        this.add (knappar[1], BorderLayout.EAST);
        this.add (knappar[2], BorderLayout.SOUTH);
        this.add (knappar[3], BorderLayout.WEST);
        this.add (knappar[4], BorderLayout.CENTER);

        // Det går även att använda konstanterna "North", "East",
        // "South", "West" och "Center" för att precisera
        // komponenternas platser. Man kan skriva så här:
        // this.add (knappar[0], "North");
    }
}

// ett program som ordnar grafiska komponenter efter väderstreck
class OrdnaKomponenterEnligtVaderstreck
{
    public static void main (String[] args)
    {
```

```
        // ett fönster
        JFrame    frame = new JFrame (" BorderLayout");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

I ett fönster visas en panel med fem knappar. Dessa knappar är ordnade efter väderstreck (nord, syd, väst, öst och mitten).



OrdnaKomponenterIEnMatris.java

Ett program som ordnar grafiska komponenter i en matris

De olika grafiska komponenterna i en behållare kan ordnas i en matris. En matris har ett antal rader och ett antal kolumner, som bildar ett antal fält. Komponenterna placeras i dessa fält i matrisen.

Komponenterna utplaceras från första till sista fält i en rad. Därefter fortsätter utplaceringen i nästa rad, och allt upprepas på samma sätt. Hela matrisen behöver inte fyllas.

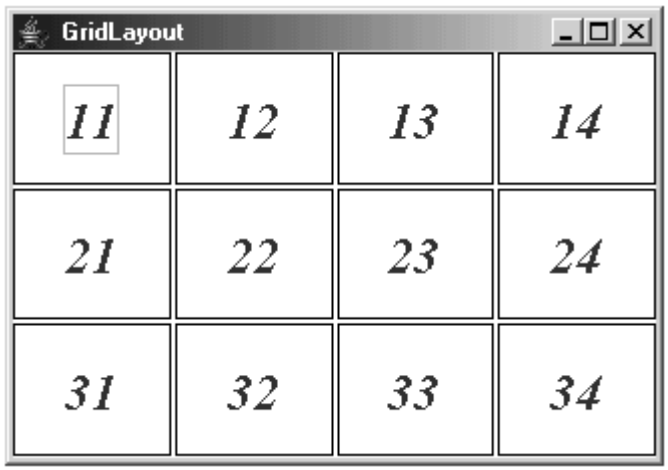
Alla komponenter i en matris har samma storlek. Det är storleken av ett fält i matrisen. Om behållarens storlek ändras, ändras även komponenternas storlek.

```
import java.awt.*;            // GridLayout, Color, Font
import javax.swing.*;         // JButton, JPanel,
                              // JFrame, BorderFactory
import javax.swing.border.*;  // Border

// en panel som innehåller ett antal knappar ordnade i en matris
class FPanel extends JPanel
{
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // knappar

        JButton[]    knappar = new JButton[12];
        Border ram = BorderFactory.createLineBorder (Color.BLACK);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 25);
        for (int i = 0; i < knappar.length; i++)
        {
            knappar[i] = new JButton ();
            knappar[i].setBackground (Color.WHITE);
            knappar[i].setFont (font);
            knappar[i].setBorder (ram);
        }
        knappar[0].setText ("11");
        knappar[1].setText ("12");
        knappar[2].setText ("13");
        knappar[3].setText ("14");
        knappar[4].setText ("21");
        knappar[5].setText ("22");
        knappar[6].setText ("23");
        knappar[7].setText ("24");
        knappar[8].setText ("31");
        knappar[9].setText ("32");
        knappar[10].setText ("33");
        knappar[11].setText ("34");

        // ange den här panelens layout
        GridLayout    layout = new GridLayout (
            3, 4,     // antalet rader och kolumner
            2, 2);    // avstånd mellan komponenterna
```

```
        // avstånden kan utelämnas (i så fall blir de 0)

        this.setLayout (layout);

        // placera knapparna i panelen
        for (int i = 0; i < knappar.length; i++)
            this.add (knappar[i]);
    }
}

// ett program som ordnar grafiska komponenter i en matris
class OrdnaKomponenterIEnMatris
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" GridLayout");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

I ett fönster finns en panel med ett antal knappar. Knapparna är ordnade i en matris med tre rader och tre kolumner.



KombineraOlikaLayoutStrategier.java

Ett program som ordnar grafiska komponenter enligt en kombinerad strategi

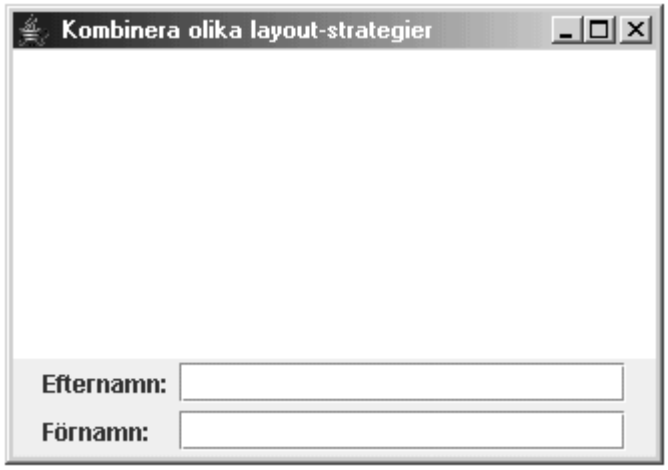
Olika strategier kan kombineras vid utplaceringen av de grafiska komponenterna i en behållare.

```
import java.awt.*;       // FlowLayot, GridLayout, BorderLayout,
                         // Color
import javax.swing.*;    // JLabel, JTextField, JPanel, JTextArea,
                         // JFrame

// en panel som innehåller ett antal grafiska komponenter, ordnade
// enligt en kombinerad strategi
class FPanel extends JPanel
{
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // en etikett och ett textfält ordnade i en panel
        JLabel   etikett1 = new JLabel ("Efternamn:");
        JTextField    falt1 = new JTextField (20);
        JPanel    panel1 = new JPanel ();
        panel1.setLayout (
                        new FlowLayout (FlowLayout.CENTER, 6, 2));
        panel1.add (etikett1);
```

```
        panel1.add (falt1);

        // en etikett och ett textfält ordnade i en panel
        JLabel   etikett2 = new JLabel ("Förnamn:");
        JTextField    falt2 = new JTextField (20);
        JPanel    panel2 = new JPanel ();
        panel2.setLayout (
                       new FlowLayout (FlowLayout.CENTER, 15, 2));
        panel2.add (etikett2);
        panel2.add (falt2);

        // två paneler med komponenter ordnade i en panel
        JPanel    panel = new JPanel ();
        panel.setLayout (new GridLayout (2, 1));
        panel.add (panel1);
        panel.add (panel2);

        // en textarea
        JTextArea    textArea = new JTextArea (10, 20);

        // placera textarean och panelen i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (textArea, "Center");
        this.add (panel, "South");
    }
}

// ett program som ordnar grafiska komponenter enligt
// en kombinerad strategi
class KombineraOlikaLayoutStrategier
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (
                        " Kombinera olika layoutstrategier");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

I ett fönster visas en panel med olika grafiska komponenter. I mitten finns en textarea. I nedre delen finns en panel, som innehåller två andra paneler. Dessa paneler är ordnade vertikalt. Varje panel innehåller en etikett och ett textfält.



En flexibel layoutstrategi

EnFlexibelLayoutStrategi.java

Ett program som ordnar grafiska komponenter enligt en flexibel strategi

Ett särskilt, beskrivande objekt kan skapas för varje komponent som placeras i en behållare, Ett sådant objekt preciserar en komponents position och storlek. Det anger också komponentens beteende vid ändringen av behållarens storlek.

För att kunna beskriva en komponents position och storlek, ritar man (på ett papper) en matris bestående av ett antal rader och ett antal kolumner. Raderna kan vara av olika höjd, och kolumnerna av olika bredd. Det betyder att matrisen kan innehålla fält av varierande storlek.

En komponents position preciseras genom att raden och kolumnen för komponentens vänstra övre hörn anges. En komponents storlek anges genom att antalet fält (i horisontell och vertikal riktning) som komponenten tar upp preciseras. Ett område för komponenten anges. Komponents

position i området kan sedan närmare preciseras (var i området som komponenten ska placeras, hur mycket fyllnad som ska omge komponenten och om komponentens minimala dimensioner ska utökas).

Komponenternas beteende vid ändringen av behållarens storlek definieras genom att olika vikter tilldelas till matrisens rader och kolumner (motsvarande komponenter tilldelas olika vikter - den största vikten i en rad blir radens vikt och den största vikten i en kolumn blir kolumnens vikt). Dessa vikter preciserar om och hur mycket enskilda rader och kolumner ska ändras vid ändringen av behållarens storlek.

```
import java.awt.*;           // GridBagLayout, GridBagConstraints,
                             // Color, Font
import javax.swing.*;    // JButton, JPanel, JFrame, BorderFactory
import javax.swing.border.*;  // Border

// en panel som innehåller ett antal knappar ordnade
// enligt en flexibel strategi
class FPanel extends JPanel
{
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // knappar

        JButton[]    knappar = new JButton[4];
        Border ram = BorderFactory.createLineBorder (Color.BLACK);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 25);
        for (int i = 0; i < knappar.length; i++)
        {
            knappar[i] = new JButton ("" + (i + 1));
            knappar[i].setBackground (Color.WHITE);
            knappar[i].setFont (font);
            knappar[i].setBorder (ram);
        }

        // ram runt den här panelen (behållaren)
        this.setBorder (ram);

        // en flexibel layoutstrategi

        GridBagLayout    layout = new GridBagLayout ();
        this.setLayout (layout);

        // ett objekt som beskriver en komponents
        // position, storlek och beteende vid
        // ändringen av behållarens storlek
        GridBagConstraints    con = new GridBagConstraints ();
```

```
// Rita en lämplig matris på ett papper, och placera
// komponenterna i matrisen. Utgå ifrån denna matris
// och beskriv komponenternas olika egenskaper
// genom motsvarande objekt.
// Utifrån dessa beskrivningar ska layouthanteraren förstå
// hur den ritade matrisen ser ut och hur komponenterna
// ska utplaceras.

// placera enskilda knappar i den här panelen

// Föreställ dig en matris med två rader och tre kolumner.
// Man behöver nu beskriva för layouthanteraren
// hur de olika knapparna ska placeras i matrisen.

// placera första knappen

// knappens vänstra hörn ligger i rad nummer 0 och kolumn
// nummer 0
con.gridx = 0;
con.gridy = 0;

// knappen ska sträcka sig ett fält horisontellt,
// och ett fält vertikalt
con.gridwidth = 1;
con.gridheight = 1;
// förvalt värde är 1, så dessa satser kan utelämnas

// fyllnad runt knappen - knappens minimala avstånd från
// områdets (fältets) kanter
con.insets = new Insets (8, 4, 8, 4);
// man anger fyllnad (i pixlar) som ska läggas upp, till
// vänster, ner och till höger i området (fältet)

// knappen ska inte fylla ut hela området (fältet),
// den ska bara fylla området horisontellt
con.fill = GridBagConstraints.HORIZONTAL;
// möjliga värden:
// NONE (default), HORIZONTAL, VERTICAL, BOTH.
// BOTH betyder att komponenten ska fylla ut området både
// vertikalt och horisontellt

// eftersom knappen inte fyller hela området, ska man
// precisera var knappen ska placeras inom området
con.anchor = GridBagConstraints.CENTER;
// möjliga värden:
// NORTH, EAST, SOUTH, WEST, CENTER (default),
// NORTHEAST, NORTHWEST, SOUTHEAST, SOUTHWEST

// Knappens vikter - den största vikten i en rad (kolumn)
// bestämmer radens (kolumnens) vikt.
```

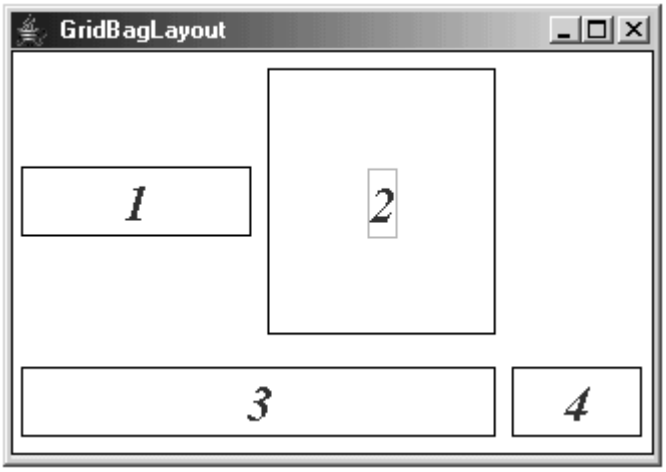
```
// En rads vikt bestämmer ändringen av radens höjd vid
// ändringen av behållarens storlek.
// En kolumns vikt bestämmer ändringen av kolumnens bredd
// vid ändringen av behållarens storlek.
con.weightx = 1;  // vikt i kolumnen
con.weighty = 1;  // vikt i raden
// förvalt värde är 0 - ingen ändring vid ändringen av
// behållarens storlek

// placera knappen i panelen
 this.add (knappar [0], con);

// placera den andra knappen

con = new GridBagConstraints ();
// Ett nytt objekt skapas med förvalda värden.
// Vissa värden justeras sedan.
// Det går även att använda det objekt som redan använts
// - objektet anpassas till den nya komponenten.
con.gridx = 1;
con.gridy = 0;
con.gridwidth = 1;
con.gridheight = 1;
con.insets = new Insets (8, 4, 8, 4);
con.fill = GridBagConstraints.BOTH;
con.weightx = 1;
con.weighty = 1;

this.add (knappar [1], con);

// placera den tredje knappen

con = new GridBagConstraints ();
con.gridx = 0;
con.gridy = 1;
con.gridwidth = 2;
con.gridheight = 1;
con.insets = new Insets (8, 4, 8, 4);
con.fill = GridBagConstraints.BOTH;

// ange knappens vikt - den vikten ska bestämma vikten
// för motsvarande rad
con.weightx = 0;
// Villkoret con.weight.x = 0 ignoreras, eftersom knappen
// ligger i de två första kolumnerna med vikten 1.
// Därför används även här vikten 1 - de första två
// kolumnerna blir bredare när behållarens bredd ökas.
con.weighty = 0;  // vikten för den andra raden
// knappen ska behålla sin höjd vid ändringar
// av behållarens höjd
```

```
        this.add (knappar [2], con);

        // placera den fjärde knappen

        con = new GridBagConstraints ();
        con.gridx = 2;
        con.gridy = 0;
        con.gridwidth = 1;
        con.gridheight = 2;
        con.insets = new Insets (8, 4, 8, 4);
        con.fill = GridBagConstraints.NONE;
        con.anchor = GridBagConstraints.SOUTH;

        // tilldela ett extrautrymme (i antalet pixlar)
        // till knappen, annars visas knappen med sin
        // minimala storlek
        con.ipadx = 50;
        con.ipady = 0;
        // förvalt värde är 0

        // ange knappens vikt - den vikten ska bestämma vikten
        // för motsvarande kolumn
        con.weightx = 0;  // vikten för tredje kolumnen
        // knappen ska behålla sin bredd vid ändringar
        // av behållarens bredd
        con.weighty = 0;
        // första raden ändras vid behållarens storlek
        // - andra raden ska inte ändras eftersom
        // den tredje knappens höjd ska behållas

       this.add (knappar [3], con);
    }
}

// ett program som ordnar grafiska komponenter enligt en flexibel
// strategi
class EnFlexibelLayoutStrategi
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" GridBagLayout");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (450, 320);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
```

```
        frame.setVisible (true);
    }
}
```

Programmets GUI

I ett fönster visas en panel med fyra knappar. Knapparna är ordnade i två rader och tre kolumner. De enskilda raderna (kolumnerna) har olika höjd (bredd).



Layoutkomponenter

BehallareForTvaKomponenter.java

Ett program som illustrerar en behållare för exakt två komponenter

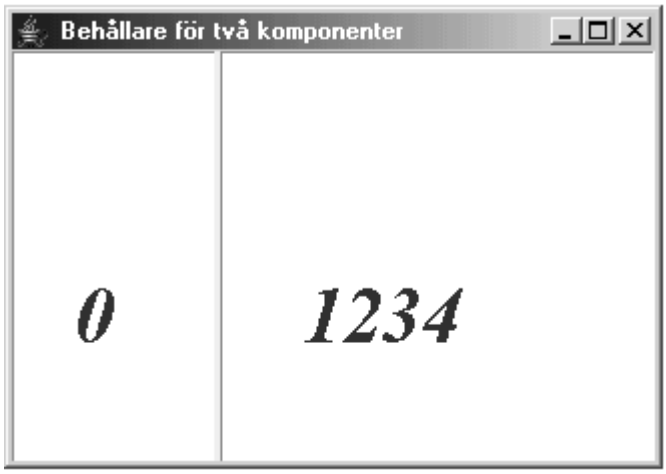
Två komponenter kan placeras i en speciell behållare, som har en avgränsningslinje mellan komponenterna. Användaren kan dra i linjen, och på så sätt välja hur behållarens area ska fördelas mellan komponenterna.

```
import java.awt.*;          // Font
import javax.swing.*;       // JTextArea, JSplitPane, JFrame

// ett program som illustrerar en behållare för exakt
// två komponenter
class BehallareForTvaKomponenter
```

```
{
    public static void main (String[] args)
    {
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 40);

        // en textarea
        JTextArea    ta1 = new JTextArea (12, 10);
        ta1.setFont (font);
        ta1.append ("\n\n   0");

        // en textarea till
        JTextArea    ta2 = new JTextArea (12, 10);
        ta2.setFont (font);
        ta2.append ("\n\n    1234");

        // en behållare för två komponenter
        JSplitPane    splitPane = new JSplitPane (
            JSplitPane.HORIZONTAL_SPLIT, // komponenterna i en rad
            ta1, ta2);  // komponenterna som ska
                        // placeras i behållaren

        // (flera komponenter kan packas i en panel, och denna
        // panel kan placeras i en behållare av typen JSplitPane)

        // komponenterna ska ständigt uppdateras när användaren
        // drar i avgränsningslinjen
        splitPane.setContinuousLayout (true);

        // bestäm avgränsningslinjens tjocklek
        splitPane.setDividerSize (5);

        // bestäm avgränsningslinjens placering
        splitPane.setDividerLocation (100);

        // ett fönster
        JFrame  frame =
                 new JFrame(" Behållare för två komponenter");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 80);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

En behållare med två textareor visas. Dessa textareor avgränsas från varandra med en linje. Användaren kan dra i linjen och utöka en textarea på bekostnad av den andra textarean.



SeEnForStorKomponent.java

Ett program som illustrerar hur för stora komponenter hanteras

En grafisk komponent (eller en uppsättning komponenter) kan vara för stor för det utrymme som den tilldelas. I så fall kan en speciell behållare användas, som tillåter användaren att se komponenten stegvis.

En komponent kan vara för stor i horisontell riktning, i vertikal riktning, eller i båda riktningarna. Så snart en komponent blir för stor i en riktning, visas en rullningslist i den riktningen. Användaren kan använda den för att granska komponenten stegvis.

```
import java.awt.*;          // Font
import javax.swing.*;       // JTextArea, JScrolPane, JFrame

// ett program som illustrerar hur för stora komponenter hanteras
class SeEnForStorKomponent
{
    public static void main (String[] args)
    {
        // en textarea
        JTextArea    textArea = new JTextArea (12, 10);
```

```
        Font    font =
                 new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        textArea.append ("\n\n  v i n t e r n");
        textArea.append ("\n\n  v å r e n");
        textArea.append ("\n\n  s o m m a r e n");
        textArea.append ("\n\n  h ö s t e n");

        // en behållare med en för stor komponent
        JScrollPane    scrollPane = new JScrollPane (textArea);
        // (flera komponenter kan packas i en panel, och panelen
        // kan placeras i en behållare av den här typen)

        // ett fönster
        JFrame frame = new JFrame (" Visa en för stor komponent");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 80);

        // placera behållaren i fönstret
        frame.add (scrollPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

En textarea placeras inuti en speciell behållare. Eftersom textarean är för stor för det utrymme som den tilldelats, ser användaren bara en del av den. Genom att använda en rullningslist kan hela textarean visas (en del i taget).

BladdraMellanOlikaKomponenter.java

Ett program som illustrerar hur användaren kan bläddra mellan olika komponenter i en behållare

Flera komponenter kan placeras bakom varandra i en behållare. En komponent placeras ut tillsammans med ett namn (eller ett namn och en ikon). Dessa namn visas överst i behållaren, så att användaren kan bläddra mellan komponenterna via namnen.

```
import java.awt.*;          // Font
import javax.swing.*;       // JTextArea, JTabbPane, JFrame

// ett program som illustrerar hur användaren kan bläddra
// mellan olika komponenter i en behållare
class BladdraMellanOlikaKomponenter
{
    public static void main (String[] args)
    {
        Font    font =
             new Font ("Serif", Font.BOLD + Font.ITALIC, 40);

        // en vektor med strängar
        String[]    arsTider = {"v i n t e r n", "v å r e n",
                                "s o m m a r e n", "h ö s t e n"};

        // en vektor med textareor
        JTextArea[]    ta = new JTextArea [4];
        for (int i = 0; i < ta.length; i++)
        {
            ta[i] = new JTextArea (12, 10);
            ta[i].setFont (font);
            ta[i].append ("\n  " + arsTider[i]);
        }

        // namn på de komponenter som ska lagras
        String[]    namn = {"ett", "två", "tre", "fyra"};

        // en behållare som placerar olika komponenter
        // bakom varandra
        JTabbedPane    tabbedPane = new JTabbedPane ();

        // placera komponenterna (med tillhörande namn)
        // i behållaren
        for (int i = 0; i < ta.length; i++)
            tabbedPane.addTab (namn[i], ta[i]);
        // (flera komponenter kan packas i en panel, som placeras
        // som en komponent i en behållare av den här typen)
```

```
        // Förutom namn, kan även en ikon tilldelas till varje
        // komponent:
        // tabbedPane.addTab (
        //          namn, new ImageIcon ("bild.gif"), komponent)):

        // Om för många komponenter ska lagras i behållaren,
        // tar komponenternas namn upp för mycket plats.
        // I så fall kan dessa namn placeras i en rad, som har
        // knappar med pilar. Med dessa pilar kan användaren
        // bläddra mellan olika namn. Man gör så här:
        // tabbedPane.setTabLayoutPolicy (
        //              JTabbedPane.SCROLL_TAB_LAYOUT);
        // Om alla namn sedan ska visas, gör man så här:
        // tabbedPane.setTabLayoutPolicy (
        //                JTabbedPane.WRAP_TAB_LAYOUT);

        // ett fönster
        JFrame    frame = new JFrame (
                        " Bläddra mellan olika komponenter");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 80);

        // placera behållaren i fönstret
        frame.add (tabbedPane);

        frame.setVisible (true);
    }
}
```

Programmets GUI

En behållare med flera komponenter bakom varandra visas. Överst i behållaren finns en rad med komponenternas namn. Användaren kan bläddra mellan komponenterna via dessa namn.

Interna fönster

InternaFonster.java

Ett program som illustrerar interna fönster

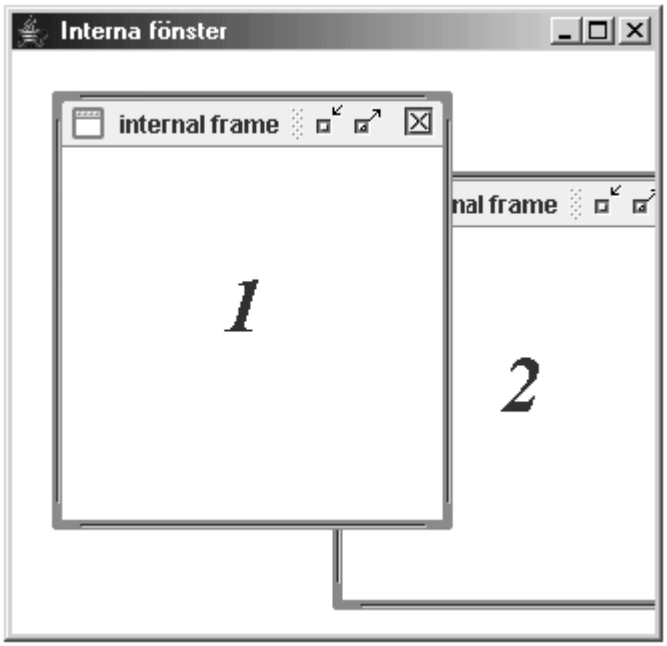
En ram kan innehålla flera interna fönster. Olika grafiska komponenter kan fördelas mellan dessa interna fönster. På så sätt skapas flera typiska enheter i ett grafiskt användargränssnitt.

```
import java.awt.*;          // Font
import javax.swing.*;       // JTextArea, JFrame,
                            // JInternalFrame, JDesktopPane,

// ett program som illustrerar interna fönster
class InternaFonster
{
    public static void main (String[] args)
    {
        Font    font =
              new Font ("Serif", Font.BOLD + Font.ITALIC, 40);

        // två textareor
        JTextArea    ta1 = new JTextArea (10, 10);
        ta1.setFont (font);
        ta1.append ("\n        1");
        JTextArea    ta2 = new JTextArea (10, 10);
        ta2.setFont (font);
        ta2.append ("\n        2");

        // ett internt fönster
        JInternalFrame   iframe1 = new JInternalFrame (
                    " internal frame",    // titel
                    true,  // storleken ska kunna ändras
                    true,  // fönstret ska kunna stängas
                    true,  // fönstret ska kunna maximeras
                    true); // fönstret ska kunna ikonifieras

        // fönstrets placering och storlek
        iframe1.reshape (
            20, 20, // koordinater för fönstrets vänstra övre hörn
            200, 220);   // fönstrets bredd och höjd

        // fönstret ska vara synligt
        iframe1.setVisible (true);

        // fönstrets beteende vid stängningen
        iframe1.setDefaultCloseOperation (
                              JInternalFrame.DISPOSE_ON_CLOSE);
```

```
        // placera en komponent i fönstret
        // (förvald layout är av typen BorderLayout)
        iframe1.add (ta1);

        // ett internt fönster till
        JInternalFrame    iframe2 = new JInternalFrame (
                     " internal frame", true, true, true, true);
        iframe2.reshape (160, 60, 200, 220);
        iframe2.setVisible (true);

        // placera en komponent i fönstret
        iframe2.add (ta2);

        // en speciell behållare för interna fönster
        JDesktopPane    desktop = new JDesktopPane ();
        desktop.add (iframe1);
        desktop.add (iframe2);

        // en ram
        JFrame    frame = new JFrame (" Interna fönster");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 340);
        frame.setLocation (120, 80);

        // ramens arbetsyta ska vara behållaren som innehåller
        // interna fönster
        frame.setContentPane (desktop);

        // visa ramen
        frame.setVisible (true);
    }
}
```

Programmets GUI

En ram med två interna fönster visas. Varje fönster innehåller en textarea.



Hantera händelser

Komponenter, händelser och lyssnare

EnGemensamLyssnare.java

Ett program som använder en gemensam lyssnare för flera grafiska komponenter

En klass kan hantera händelser från flera olika källor. I så fall kan källan till den händelse som inträffat bestämmas. När källan väl bestämts, kan programmet reagera på ett lämpligt sätt (som beror på källkomponenten) på händelsen. En gemensam lyssnare (ett gemensamt lyssnarobjekt) kan användas för flera olika komponenter i ett grafiskt användargränssnitt.

```
import java.awt.*;          // BorderLayout
import javax.swing.*;       // JLabel, JTextField,
                            // JButton, JPanel,
                            // JTextArea, JFrame
import java.awt.event.*;    // ActionListener, ActionEvent

// en klass som hanterar händelser för två olika komponenter
class HandelseHanterare implements ActionListener
{
    // komponenter som är nödvändiga för att kunna reagera på
    // olika händelser
    private JTextField    textFalt;
    private JTextArea     textArea;
    private JButton       knapp;

    // initiera textfältet, textarean och knappen
    public HandelseHanterare (JTextField textFalt,
                        JTextArea      textArea, JButton knapp)
    {
        this.textFalt = textFalt;
        this.textArea = textArea;
        this.knapp = knapp;
    }

    // det som ska hända när användaren trycker på returtangenten
    // i textfältet, eller klickar på knappen
    public void actionPerformed (ActionEvent e)
    {
        // bestäm källan för händelsen
        Object    kallan = e.getSource ();

        // gör olika saker för olika källor
```

```
        if (kallan == textFalt)
        // om händelsen inträffat i textfältet
        {
            // hämta texten från textfältet
            // och skriv den i textarean
            String    text = textFalt.getText ();
            textFalt.setText ("");
            textArea.setText ("");
            textArea.append ("\n\n  " + text + "\n");
        }
        else  // else if (kallan == knapp)
        // om användaren klickat på knappen
            System.exit (0);  // avsluta

        // alternativt kan händelsens källa bestämmas så här:
        // Object    kallan = e.getSource ();
        // if (kallan instanceof JTextField)
        // {
        //    JTextField    textFalt = (JTextField) kallan;
        //    reaktion här
        // }
        // else
        // {
        //    JButton    knapp = (JButton) kallan;
        //    reaktion här
        // }
        // Om denna strategi används, behövs det bara en textarea
        // som argument till konstruktorn (knappen och textfältet
        // erhålls med metoden getSource).
        // Denna strategi (med instanceof) kan inte användas om
        // alla källor är av samma klass
        // (till exempel om händelser för flera textfält
        // hanteras i en och samma klass)
    }
}

// en panel med flera grafiska komponenter
class FPanel extends JPanel
{
    public FPanel ()
    {
        // flera grafiska komponenter
        JLabel    label = new JLabel ("Ditt namn:");
        JTextField    textFalt = new JTextField (30);
        JButton    knapp = new JButton (" Avsluta");
        JTextArea    textArea = new JTextArea (40, 20);

        // skapa en lämplig lyssnare, och registrera den hos
        // textfältet och knappen
        ActionListener    hanterare =
             new HandelseHanterare (textFalt, textArea, knapp);
```

```
        textFalt.addActionListener (hanterare);
        knapp.addActionListener (hanterare);

        // en gemensam lyssnare för textfältet och knappen -
        // denna lyssnare bestämmer varifrån händelsen kommer,
        // och reagerar på ett lämpligt sätt


        // gruppera flera komponenter i en särskild panel
        JPanel    panel = new JPanel ();
        panel.add (label);
        panel.add (textFalt);
        panel.add (knapp);

        // placera komponenterna i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (textArea, "Center");
        this.add (panel, "South");
    }
}

// ett program som använder en gemensam lyssnare för
// flera grafiska komponenter
class EnGemensamLyssnare
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En gemensam lyssnare");
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret tillsammans med olika komponenter
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster visas som innehåller en textarea, en etikett, ett textfält och en knapp. Det som matas in via textfältet, visas i textarean. Programmet avslutas när användaren trycker på knappen.

OlikaTyperHandelser.java

Ett program som hanterar händelser av olika typer

Komponenter i ett grafiskt användargränssnitt kan generera händelser av olika typer. För att kunna hantera händelser av en viss typ, måste en klass skapas som implementerar det gränssnitt som motsvarar den typen av händelser. En och samma klass kan hantera händelser av olika typer. Denna klass måste i så fall implementera alla nödvändiga gränssnitt.

```
import java.awt.*;           // BorderLayout
import javax.swing.*;        // JSlider, JButton, JPanel, JFrame
import java.awt.event.*;     // ActionListener, ActionEvent
import javax.swing.event.*;  // ChangeListener, ChangeEvent

// en klass som hanterar händelser av olika typer
class HandelseHanterare implements ActionListener, ChangeListener
{
    // en panel
    private JPanel    panel;

    // initiera panelen
    public HandelseHanterare (JPanel panel)
    {
        this.panel = panel;
    }
```

```
    // det som ska hända när användaren trycker på en knapp
    public void actionPerformed (ActionEvent e)
    {
        System.exit (0);
    }

    // det som ska hända när användaren ändrar tillståndet
    // för en vallinjal
    public void stateChanged (ChangeEvent e)
    // stateChanged är den enda metod som definieras i
    // gränssnittet ChangeListener
    {
        // bestäm händelsens källa
        Object     kallan = e.getSource ();
        JSlider    slider = (JSlider) kallan;

        // bestäm vallinjalens värde
        int   varde = slider.getValue ();

        // ändra panelens färg
        Color    farg = new Color (0, 0, varde);
        panel.setBackground (farg);
        panel.repaint ();
    }
}

// en panel med flera grafiska komponenter
class FPanel extends JPanel
{
    public FPanel ()
    {
        // flera grafiska komponenter
        JSlider     slider = new JSlider (JSlider.HORIZONTAL,
                                          0,    // minsta värdet
                                          255,  // största värdet
                                          127); // förvalda värdet
        JButton    knapp = new JButton (" Avsluta");
        JPanel     panel = new JPanel ();
        panel.setBackground (new Color (255, 255, 255));

        // skapa lämpliga lyssnare och registrera dem hos
        // motsvarande komponenter
        ChangeListener hanterare1 = new HandelseHanterare (panel);
        slider.addChangeListener (hanterare1);

        ActionListener hanterare2 = new HandelseHanterare (panel);
        knapp.addActionListener (hanterare2);

        // gruppera flera komponenter
        JPanel    kontroll= new JPanel ();
        kontroll.add (slider);
```

```
        kontroll.add (knapp);

        // placera komponenterna i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (panel, "Center");
        this.add (kontroll, "South");
    }
}

// ett program som hanterar händelser av olika typer
class OlikaTyperHandelser
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame   frame = new JFrame (" Olika typer av händelser");
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

En vallinjal och en knapp visas i nedre delen av fönstret, och en panel i mitten. Användaren kan välja olika värden via vallinjalen, och på så sätt justera panelens färg.

Användaren avslutar programmet genom att klicka på knappen.

AnvandEnAdapterklass.java

Ett program som använder en adapterklass

Ett lyssnargränssnitt kan ha flera lyssnarmetoder. Vid olika händelser anropas olika metoder. En lyssnarklass som implementerar ett lyssnargränssnitt med flera metoder, måste implementera alla metoderna. Den måste även implementera de metoder som man inte är intresserad av. Dessa metoder kan implementeras som tomma metoder (metoder som inte gör någonting).

För att underlätta implementeringen av lyssnarklasser med flera metoder, har Java skapat ett antal adapterklasser. För var och ett av de lyssnargränssnitt som finns i paketet `java.awt.event` och som har flera metoder, finns motsvarande adapterklass. En adapterklass implementerar ett lyssnargränssnitt med flera metoder på så sätt att dessa metoder inte utför någonting (tomma metoder). En lyssnarklass kan definieras som en subklass till en adapterklass. I så fall behöver inte alla metoder från gränssnittet implementeras i klassen, eftersom den ärver metoderna. Endast de metoder vars beteende ska ändras (de metoder som reagerar på aktuella händelser i det grafiska användargränssnittet) omdefinieras.

```
import javax.swing.*;        // JFrame
import java.awt.event.*;     // WindowAdapter

// en klass som hanterar de händelser i ett fönster som genereras
```

```
// när användaren klickar på stängningsknappen i fönstret
class Avslutare extends WindowAdapter
// Klassen WindowAdapter implementerar gränssnittet
// WindowListener. På så sätt implementerar även klassen
// Avslutare detta gränssnitt (klassen ärver sju metoder
// som gör ingenting).
// I det här klassen omdefinieras bara metoden windowClosing.
{
    // det som händer när användaren klickar på ett fönsters
    // stängningsknapp
    public void windowClosing (WindowEvent e)
    {
        System.exit (0);
    }
}

/*************************************************************

Klassen Avslutare kan även implementeras så här:

    class Avslutare implements WindowListener
    {
        void windowClosing (WindowEvent e)
        {
            System.exit (0);
        }

        void windowOpened (WindowEvent e) {}
        void windowClosed (WindowEvent e) {}
        void windowIconified (WindowEvent e) {}
        void windowDeiconified (WindowEvent e) {}
        void windowActivated (WindowEvent e) {}
        void windowDeactivated (WindowEvent e) {}
    }

*************************************************************/

// ett program som använder en adapterklass
class AnvandEnAdapterklass
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Använd en adapterklass");
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // skapa en lyssnare och registrera den hos fönstret
        Avslutare    lyssnare = new Avslutare ();
        frame.addWindowListener (lyssnare);
        // det här är ekvivalent med:
```

```
        // frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster visas. När användaren klickar på stängningsknappen i fönstret avslutas programmet.

Olika sätt att implementera en lyssnarklass

EnInreLyssnarklass.java

Ett program som definierar en lyssnarklass som en inre klass

En lyssnarklass kan definieras som inre klass i en annan klass. Ett objekt av en inre klass konstrueras alltid i samband med ett objekt av den omgivande klassen. Det "inre" objektet kan fritt använda instansvariabler i det "omgivande" objektet.

En lyssnarklass kan definieras i en klass som innehåller de variabler som är nödvändiga för ett lyssnarobjekts funktion. I så fall kan den inre klassen fritt använda instansvariablerna i den omgivande klassen. Det underlättar implementeringen av en lyssnarklass, eftersom instansvariablerna inte behöver skickas som argument till lyssnarklassens konstruktorer och metoder. En inre klass kan även använda instansmetoder i den omgivande klassen.

En inre klass kan definieras i en annan (omgivande) klass, och ett objekt av klassen kan skapas i den omgivande klassen. Ett "inre" objekt kan skapas i den omgivande klassens konstruktor. I så fall skapas alltid ett objekt av den inre klassen när ett objekt av den omgivande klassen skapas. Det inre objektet skapas då i samband med det omgivande objektet, och kan använda dess instansvariabler och instansmetoder. Ett "inre" objekt kan även skapas i en av den omgivande klassens instansmetoder. I så fall skapas alltid ett "inre" objekt när metoden exekveras. Det "inre" objektet skapas i så fall i samband med det aktiverande objektet (som är av den omgivande klassen).

```
import java.awt.*;              // BorderLayout
import javax.swing.*;           // JLabel, JTextField,
                                // JTextArea, JPanel,
                                // JFrame
import java.awt.event.*;        // ActionListener, ActionEvent

// en panel med flera grafiska komponenter
class FPanel extends JPanel
{
    // instansvariabler - panelens grafiska komponenter
    private JLabel    label;
    private JTextField    textFalt;
    private JTextArea     textArea;

    // en inre lyssnarklass
    private class HandelseHanterare implements ActionListener
    {
        // hantera händelsen
        public void actionPerformed (ActionEvent e)
        {
            // bestäm den text som användaren skrivit i textfältet
            String    text = textFalt.getText ();
            textFalt.setText ("");

            // skriv den inmatade texten i textarean
            textArea.setText ("");
            textArea.append ("\n\n  " + text + "\n");

            // Eftersom den här klassen definieras som en
            // inre klass inuti klassen FPanel, kan den
            // fritt använda instansvariablerna i klassen
            // FPanel. Dessa variabler behöver inte anges
            // som parametrar i den här klassen.

            // Instansvariablerna textFalt och textArea i den
            // omgivande klassen FPanel används i den här klassen.
            // Det underlättar implementeringen av den här
            // klassen.

            // En inre klass kan även använda instansmetoder
            // i den omgivande klassen.
        }
    }

    public FPanel ()
    {
        // skapa panelens grafiska komponenter
        label = new JLabel ("Ditt namn:");
        textFalt = new JTextField (40);
        textArea = new JTextArea (40, 20);
```

```
        // skapa en lämplig lyssnare, och registrera den hos
        // textfältet
        ActionListener    hanterare = new HandelseHanterare ();
        textFalt.addActionListener (hanterare);

        // Objektet hanterare är ett objekt av en inre klass till
        // den här klassen. Varje gång när ett objekt
        // av den här klassen (av typen FPanel) skapas,
        // skapas även ett objekt av typen HandelseHanterare
        // (en lyssnare) i samband med objektet
        // (koden i den här konstruktorn utförs, och ett objekt
        // av typen HandelseHanterare skapas). Denna lyssnare
        // kan fritt använda instansvariabler och instansmetoder
        // i objektet (av typen FPanel).

        // gruppera flera komponenter i en särskild panel
        JPanel    panel = new JPanel ();
        panel.add (label);
        panel.add (textFalt);

        // placera komponenterna i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (textArea, "Center");
        this.add (panel, "South");
    }
}

// ett program som använder en inre lyssnarklass
class EnInreLyssnarklass
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En inre lyssnarklass");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster visas som innehåller en textarea i mitten och en etikett och ett textfält i den nedre delen. Det som användaren matar in via textfältet, visas i textarean.

EnLokalLyssnarklass.java

Ett program som definierar en lyssnarklass lokalt

En lyssnarklass kan definieras som en inre, anonym klass. En sådan klass definieras när ett objekt av klassen skapas. En inre, anonym klass kan definieras lokalt, när en lyssnare skapas och registreras (en lokal lyssnarklass).

En inre, anonym klass skapas i en konstruktor, eller i en metod i den omgivande klassen. Därför kan en anonym, inre klass använda även lokala `final`-variabler i denna konstruktor eller metod.

```
import java.awt.*;          // BorderLayout
import javax.swing.*;       // JLabel, JTextField,
                            // JTextArea, JPanel,
                            // JFrame
import java.awt.event.*;    // ActionListener, ActionEvent

// en panel med flera grafiska komponenter
class FPanel extends JPanel
{
    // instansvariabler - panelens grafiska komponenter
    private JLabel    label;
    private JTextField    textFalt;
    private JTextArea     textArea;

    public FPanel ()
    {
        // skapa panelens grafiska komponenter
        label = new JLabel ("Ditt namn:");
        textFalt = new JTextField (40);
        textArea = new JTextArea (40, 20);

       // definiera en lyssnarklass, skapa en lyssnare
       // av klassen, och registrera lyssnaren hos textfältet
        textFalt.addActionListener (new ActionListener ()
            // Lyssnarklassen implementerar gränssnittet
            // ActionListener (en lokal lyssnarklass kan även
            // ärva från en adapterklass).
            // Här definieras lyssnarklassens metoder.
            {
```

```
                // hantera händelsen
                public void actionPerformed (ActionEvent e)
                {
                    // hämta den text som användaren skrivit i
                    // textfältet
                    String    text = textFalt.getText ();
                    textFalt.setText ("");

                    // skriv den inmatade texten i textarean
                    textArea.setText ("");
                    textArea.append ("\n\n  " + text + "\n");
                }
            } );

        // Lyssnarklassen definieras lokalt, när ett objekt
        // av klassen skapas och registreras.

/*************************************************************

// Det går även att använda en annan strategi.
// Först definieras en inre, anonym lyssnarklass, och ett
// objekt av klassen skapas. Därefter registreras objektet
// som lyssnare.

        ActionListener    lyssnare = new ActionListener ()
             {
                public void actionPerformed (ActionEvent e)
                {
                    // hämta texten som användaren skrivit i
                    // textfältet
                    String    text = textFalt.getText ();
                    textFalt.setText ("");

                    // visa den inmatade texten i textarean
                    textArea.setText ("");
                    textArea.append ("\n\n  " + text + "\n");
                }
             };

        textFalt.addActionListener (lyssnare);

*************************************************************/

        // gruppera flera komponenter i en särskild panel
        JPanel    panel = new JPanel ();
        panel.add (label);
        panel.add (textFalt);

        // placera komponenterna i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (textArea, "Center");
```

```
        this.add (panel, "South");
    }
}

// ett program som använder en lokal lyssnarklass
class EnLokalLyssnarklass
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" En lokal lyssnarklass");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster visas, som innehåller en textarea i mitten och en etikett och ett textfält i den nedre delen. Det som användaren matar in via textfältet visas i textarean.

EnGrafiskKomponentSomLyssnare.java

Ett program som använder en grafisk komponent som lyssnare

En grafisk komponent kan användas som lyssnare för en annan grafisk komponent.

En behållare för grafiska komponenter (till exempel en panel), kan göras till en lyssnare för händelser som uppstår i samband med dessa komponenter. Behållaren är i så fall en intelligent komponent, som både innehåller olika komponenter och reagerar på deras händelser.

Genom att definiera lyssnarmetoder i behållaren, skapar man möjlighet för dessa metoder att använda instansvariablerna och instansmetoderna i behållaren. Det underlättar implementeringen av lyssnarmetoderna.

```
import java.awt.*;          // BorderLayout
import javax.swing.*;       // JLabel, JTextField,
                            // JTextArea, JPanel,
                            // JFrame
import java.awt.event.*;    // ActionListener, ActionEvent

 // en panel som innehåller olika grafiska komponenter, och som är
 // lyssnare för händelser som genereras av dessa komponenter
class FPanel extends JPanel implements ActionListener
// panelen är en lyssnare
{
    // instansvariabler - panelens grafiska komponenter
    private JLabel    label;
    private JTextField    textFalt;
    private JTextArea     textArea;

    public FPanel ()
    {
        // skapa panelens grafiska komponenter
        label = new JLabel ("Ditt namn:");
        textFalt = new JTextField (40);
        textArea = new JTextArea (40, 20);

        // registrera den här panelen (this-objektet) som lyssnare
        // hos textfältet
        textFalt.addActionListener (this);

        // gruppera flera komponenter i en särskild panel
        JPanel    panel = new JPanel ();
        panel.add (label);
        panel.add (textFalt);

        // placera komponenterna i den här panelen
        this.setLayout (new BorderLayout ());
        this.add (textArea, "Center");
        this.add (panel, "South");
    }

    // hantera händelser som uppstår i samband med olika
    // komponenter i panelen
    public void actionPerformed (ActionEvent e)
    {
        // bestäm den text som användaren skrivit i textfältet
        String    text = textFalt.getText ();
        textFalt.setText ("");
```

```
        // skriv den inmatade texten i textarean
        textArea.setText ("");
        textArea.append ("\n\n  " + text + "\n");

        // klassens instansvariabler och instansmetoder
        // kan användas i den här metoden
        // (eftersom metoden ligger i klassen)
    }
}

// ett program som använder en panel som lyssnare
class EnGrafiskKomponentSomLyssnare
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame =
            new JFrame (" En grafisk komponent som lyssnare");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();

        // placera panelen i fönstret
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster, som innehåller en textarea i mitten och en etikett och ett textfält i nedre delen, visas. Det som användaren matar in via textfältet visas i textarean.

Mushändelser

HanteraMusKlick.java

Ett program som hanterar mussklick

Olika mushändelser, som sker i en viss grafisk komponent, kan hanteras. En händelse skapas när en musknapp klickas, när den trycks ner eller när

en nedtryckt musknapp släpps. En händelse skapas även när musen flyttas in i en grafisk komponent (den del av komponenten som inte är täckt av en annan komponent), eller när musen lämnar komponenten.

Musändelserna skapas även när musen flyttas över en viss komponent, eller när en mussknapp trycks ner och musen sedan dras över komponenten.

```
import java.awt.*;          // Shape, Color, Graphics, Graphics2D
import java.awt.geom.*;     // Ellipse2D, Ellipse2D.Double,
                            // Rectangle2D, Rectangle2D.Double
import javax.swing.*;       // JPanel, JFrame
import java.awt.event.*;    // MouseEvent,
                            // MouseListener, MouseAdapter

// en panel som har en muslyssnare
class FPanel extends JPanel
{
    // en figur och dess färg
    private Shape    figur = null;
    private Color    farg = null;

    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // en muslyssnare
        MouseListener    lyssnare = new MouseAdapter ()
            {
                public void mouseClicked (MouseEvent e)
                {
                    // antal klick
                    int    antalKlick = e.getClickCount ();

                    // musknappen som använts (1, 2 eller 3,
                    // eller BUTTON1, BUTTON2, eller BUTTON3)
                    int    musKnapp = e.getButton ();

                    // koordinaterna för den punkt i den grafiska
                    // komponenten som musen klickats på
                    int    x = e.getX ();
                    int    y = e.getY ();

                    // skapa en figur
                    if (antalKlick == 1)
                      figur = new Ellipse2D.Double (x, y, 40, 40);
                    else if (antalKlick == 2)
                    // vid ett dubbelklick registreras även det
                    // första klicket som ett separat klick,
                    // vid ett trippel- klick registreras även
```

```
                    // ett enkelklick och ett dubbelklick,
                    // och så vidare
                        figur =
                            new Rectangle2D.Double (x, y, 40, 40);

                    // bestäm figurens färg
                    if (musKnapp == MouseEvent.BUTTON1)
                        farg = Color.RED;
                    else
                        farg = Color.BLUE;

                    // rita om panelen
                    repaint ();
                    // en metod i den omgivande klassen
                    // kan användas i en inre klass -
                    // men this-referensen till det
                    // omgivande objektet kan inte användas direkt
                    // (inte this.repaint (), this betyder det här
                    // objektet, av den här klassen).
                    // Referensen kan ändå användas, men den
                    // omgivande klassens namn måste anges:
                    // FPanel.this.repaint ();
                }

                // Förutom metoden mouseClicked, har gränssnittet
                // MouseListener (och adapterklassen MouseAdapter)
                // även metoderna mousePressed, mouseReleased,
                // mouseEntered och mouseExited
            };

        // registrera muslyssnaren hos den här panelen
        // (så att de mushändelser som sker i panelen kan
        // fångas och hanteras)
        this.addMouseListener (lyssnare);
    }

    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        // rita figuren
        g.setPaint (farg);
        if (figur != null)
            g.fill (figur);
    }
}

// ett program som hanterar musklick
class HanteraMusKlick
{
```

```
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Hantera musklick");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600, 400);
        frame.setLocation (120, 80);

        // en panel
        FPanel    panel = new FPanel ();
        frame.add (panel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en panel visas.

Genom att klicka eller dubbelklicka med musen inuti panelen, ritas en figur (eller två) i panelen. Vilken eller vilka figurer som ska ritas beror på om användaren klickat eller dubbelklickat. Figurens färg beror på den använda knappen.

HanteraMusDrag.java

Ett program som hanterar musdrag

Mushändelser kan genereras genom att användaren trycker ned en musknapp över en grafisk komponent och drar musen. Händelserna genereras så länge musen dras (musen flyttas med en nedtryckt knapp). Händelserna fortsätter att genereras även när musen lämnar komponenten.

```
import java.awt.*;          // Graphics, Graphics2D
import java.awt.geom.*;     // Ellipse2D, Ellipse2D.Double,
import javax.swing.*;       // JPanel, JFrame
import java.awt.event.*;    // MouseEvent, MouseMotionListener,
                            // MouseMotionAdapter,

// en panel som har en muslyssnare
class FPanel extends JPanel
{
    // en partikel
    private Ellipse2D    partikel = null;
```

```
    public FPanel ()
    {
        this.setBackground (Color.WHITE);

        // en muslyssnare
        MouseMotionListener   lyssnare = new MouseMotionAdapter ()
            {
                public void mouseDragged (MouseEvent e)
                {
                    // koordinater för den punkt
                    // där musen befinner
                    // sig när användaren drar musen
                    int    x = e.getX ();
                    int    y = e.getY ();

                    // skapa en partikel på den plats där musen
                    // befinner sig
                    partikel = new Ellipse2D.Double ();
                    partikel.setFrameFromCenter (
                                   x, y, x - 5, y - 5);

                    // rita om panelen
                    repaint ();
                }

                // Förutom metoden mouseDragged, har gränssnittet
                // MouseMotionListener (och adapterklassen
                // MouseMotionAdapter) även metoden mouseMoved
            };

        // registrera muslyssnaren hos den här panelen
        this.addMouseMotionListener (lyssnare);
    }

    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);
        Graphics2D    g = (Graphics2D) gr;

        // rita partikeln
        if (partikel != null)
            g.fill (partikel);
    }
}

// ett program som hanterar musdrag
class HanteraMusDrag
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
```

```
        JFrame    frame = new JFrame (" Hantera musdrag");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600, 400);
        frame.setLocation (120, 80);
        FPanel    panel = new FPanel ();
        frame.add (panel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en panel visas. Genom att trycka ner en musknapp i panelen och dra musen, ritar användaren en partikel som rör sig efter musen.

Tangentbordshändelser

HanteraEnNedtrycktTangent.java

Ett program som hanterar händelser som uppstår när användaren trycker ned en tangent på tangentbordet

Olika händelser kan skapas med datorns tangentbord. En händelse skapas när en tangent trycks ner, när en nertryckt tangent släpps upp, eller när användaren trycker på en tangent (eller en kombination av tangenter) för att producera ett tecken i ett textfönster.

En tangentbordshändelse kan bindas till en grafisk komponent, och fångas och hanteras. Tangentbordshändelsen kan bara bindas till en komponent som är i fokus (musen används för att sätta en komponent i fokus).

När en tangent på tangentbordet trycks ner, skapas en händelse. Händelserna skapas om och om igen, så länge tangenten hålls nertryckt.

```
import java.awt.*;          // Graphics, Graphics2D
import java.awt.geom.*;     // Ellipse2D, Ellipse2D.Double,
import javax.swing.*;       // JPanel, JFrame
import java.awt.event.*;    // KeyEvent, KeyListener, KeyAdapter,

// en panel som har en tangentbordslyssnare
class FPanel extends JPanel
{
    // en partikel
    private Ellipse2D    partikel = null;
```

```
// partikelns position
private int    x = 5;
private int    y = 5;

public FPanel ()
{
    this.setBackground (Color.WHITE);

    // panelen ska kunna vara i fokus, så att den kan ta emot
    // tangentbordshändelser (förvalt beteende för en panel:
    // en panel kan inte vara i fokus)
    this.setFocusable (true);

    // en tangentbordslyssnare
    KeyListener    lyssnare = new KeyAdapter ()
        {
            public void keyPressed (KeyEvent e)
            // medan en tangent hålls nedtryckt
            {
                // koden för den nedtryckta tangenten
                int    tangentKod = e.getKeyCode ();

                // modifiera partikelns position,
                // om en piltangent är nedtryckt
                if (tangentKod == KeyEvent.VK_RIGHT)
                // om högerpilen nedtryckt
                    x = x + 10;
                else if (tangentKod == KeyEvent.VK_LEFT)
                    x = x - 10;
                else if (tangentKod == KeyEvent.VK_DOWN)
                    y = y + 10;
                else if (tangentKod == KeyEvent.VK_UP)
                    y = y - 10;

                // rita om panelen
                repaint ();
            }

            // Förutom metoden keyPressed, har gränssnittet
            // KeyListener (och adapterklassen KeyAdapter)
            // även metoderna keyReleased och keyTyped.
        };

    // registrera tangentbordslyssnaren hos den här panelen
    this.addKeyListener (lyssnare);
}

public void paintComponent (Graphics gr)
{
    super.paintComponent (gr);
    Graphics2D    g = (Graphics2D) gr;
```

```
        // partikeln
        partikel = new Ellipse2D.Double ();
        partikel.setFrameFromCenter (x, y, x - 5, y - 5);
        g.fill (partikel);
    }
}

// ett program som hanterar en nedtryckt tangent
class HanteraEnNedtrycktTangent
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
        JFrame    frame =
                  new JFrame (" Hantera en nedtryckt tangent");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (600, 400, 120, 80);
        FPanel    panel = new FPanel ();
        frame.add (panel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en panel med en partikel visas. Användaren kan använda piltangenterna för att dra partikeln i olika riktningar. Detta kan bara göras när panelen sätts i fokus (genom att användaren klickar med musen i den).

HanteraEnTangenttryckning.java

Ett program som hanterar den händelse som uppstår när användaren trycker på en tangent på tangentbordet

När användaren trycker på en tangent eller en kombination av tangenter för att producera ett tecken, skapas en händelse. Händelsen kan bindas till en grafisk komponent, och fångas och hanteras.

```
import java.awt.*;          // BorderLayout, KeyboardFocusManager
import javax.swing.*;       // JPanel, JTextArea, JFrame
import java.awt.event.*;    // KeyEvent, KeyListener, KeyAdapter,

// en panel som har en tangentbordslyssnare
class FPanel extends JPanel
{
```

```
    // två textareor
    private JTextArea    textArea1 = null;
    private JTextArea    textArea2 = null;

    public FPanel ()
    {
        // skapa textareorna och placera dem i panelen
        textArea1 = new JTextArea (10, 100);
        textArea2 = new JTextArea (10, 100);
        this.setLayout (new BorderLayout ());
        this.add (textArea1, "North");
        this.add (textArea2, "South");

        // en tangentbordslyssnare
        KeyListener    lyssnare = new KeyAdapter ()
            {
                public void keyTyped (KeyEvent e)
                // när användaren trycker på en tangent eller en
                // kombination av tangenter för att producera ett
                // Unicode-tecken
                {
                    // det tecken som produceras
                    char    tecken = e.getKeyChar ();

                    // bestäm den textarea som för tillfället
                    // inte är i fokus
                    JTextArea    textArea = textArea1;
                    if (textArea1.isFocusOwner ())
                        textArea = textArea2;

/*****************************************************************

// den textarea som för tillfället är i fokus bestäms så här:
KeyboardFocusManager    fokusManager =
           KeyboardFocusManager.getCurrentKeyboardFocusManager ();
JTextArea    textArea = (JTextArea) fokusManager.getFocusOwner ();

*****************************************************************/

                    // lägg till tecknet i textarean som inte är
                    // i fokus
                    textArea.append ("" + tecken);
                }
            };

        // registrera tangentbordslyssnaren hos textareorna
        textArea1.addKeyListener (lyssnare);
        textArea2.addKeyListener (lyssnare);
    }
}
```

```
// ett program som hanterar en tangenttryckning
class HanteraEnTangenttryckning
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en panel
        JFrame    frame =
                  new JFrame (" Hantera en tangenttryckning");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (600, 400, 120, 80);
        FPanel    panel = new FPanel ();
        frame.add (panel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster med en panel visas. Panelen innehåller två textareor. När användaren skriver en text via tangentbordet, visas texten i den textarea som är i fokus. Men texten visas automatiskt även i den andra textarean. Användaren kan med musen välja den textarea som ska (för tillfället) vara i fokus (användaren kan växla mellan textareorna).

Kapitel 6

Användargränssnittets funktioner

Texthantering

Textutmatning
Textinmatning
Bevaka ändringar i ett dokument

Val mellan olika alternativ

Komponenter med två lägen
Symbolbehållare
Symbolgeneratorer

Strukturerade val

Ett verktygsfält
Menyer

Dialoger

Använda dialoger

Datahantering via tabeller

Presentera data via en tabell
Redigera data via en tabell
Val via en tabell

Texthantering

Textutmatning

MataUtTeckenstrangar.java

Ett program som visar hur olika teckensträngar matas ut

En teckensträng bestående av en rad kan matas ut till ett textfält i ett grafiskt användargränssnitt. En teckensträng bestående av ett godtyckligt antal rader kan matas ut till en textarea i ett grafiskt användargränssnitt.

```
import java.awt.*;           // Font, Color, BorderLayout
import javax.swing.*;        // JTextField, JTextArea, JLabel,
                             // JScrollPane, JPanel, JFrame

// en behållare med en utmatningsarea och ett utmatningsfält
class OutPanel extends JPanel
{
    // en utmatningsarea
    private JTextArea    outArea;

    // ett utmatningsfält
    private JTextField    outFalt;

    // initiera den här behållaren
    public OutPanel ()
    {
        // utmatningsarean
        outArea = new JTextArea (4, 6);
        // en textarea med 4 rader, varje rad har 6 positioner
        // (en position kan rymma minst ett tecken)

        // för långa rader ska brytas vid textareans kant
        outArea.setLineWrap (true);

        // enskilda ord ska inte brytas
        outArea.setWrapStyleWord (true);

        // användaren ska inte kunna redigera text i textarean
        outArea.setEditable (false);

        // font i textarean
        Font    font =
           new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        outArea.setFont (font);
```

```
// textareans förgrundsfärg
outArea.setForeground (Color.LIGHT_GRAY);

// en etikett för utmatningsarean
JLabel    etikett1 = new JLabel ("display 1");

// utmatningsfältet
outFalt = new JTextField (20);
// ett utmatningsfält med 20 synliga positioner
// (en position kan rymma minst ett tecken)

// användaren ska inte kunna redigera text i textfältet
outFalt.setEditable (false);

// en etikett för utmatningsfältet
JLabel    etikett2 = new JLabel ("display 2");

// en tråd som matar ut teckensträngar till textarean
Runnable    skrivare1 = new Runnable ()
    {
        public void run ()
        {
            while (true)
            {
                // lägg till teckensträngen i textarean
                outArea.append (
                        "Vi älskar varandra, vi älskar!");

                try
                {
                    Thread.sleep (1000);
                }
                catch (InterruptedException e)
                {}

                // lägg till teckensträngen i textarean
                outArea.append (
                           "\nKärleken är så vacker!");

                try
                {
                    Thread.sleep (2000);
                }
                catch (InterruptedException e)
                {}

                // infoga teckensträngen på den givna
                // positionen i textarean
                outArea.insert ("\n", 30);

                try
```

```
                {
                    Thread.sleep (4000);
                }
                catch (InterruptedException e)
                {}

                // byt ut tecken i det angivna området
                // mot den givna teckensträngen
                outArea.replaceRange ("", 0, 32);

                try
                {
                    Thread.sleep (1000);
                }
                catch (InterruptedException e)
                {}

                // ange texten i textarean till den givna
                // teckensträngen
                outArea.setText ("");

                try
                {
                    Thread.sleep (2000);
                }
                catch (InterruptedException e)
                {}
            }
        }
    };

new Thread (skrivare1).start ();


// en tråd som matar ut teckensträngar till textfältet
Runnable    skrivare2 = new Runnable ()
    {
        public void run ()
        {
            while (true)
            {
                // ange textfältets text
                outFalt.setText (
                   "Kärleken är så vacker!");

                try
                {
                    Thread.sleep (2000);
                }
                catch (InterruptedException e)
                {}
```

```
                        // ange textfältets text
                        outFalt.setText ("");

                        try
                        {
                            Thread.sleep (1000);
                        }
                        catch (InterruptedException e)
                        {}
                    }
                }
            };

        new Thread (skrivare2).start ();

        // placera komponenterna i den här behållaren
        JPanel    panel = new JPanel ();
        panel.add (etikett2);
        panel.add (outFalt);
        this.setLayout (new BorderLayout ());
        this.add (etikett1, "North");
        this.add (new JScrollPane (outArea), "Center");
        this.add (panel, "South");
    }
}

// ett program som visar hur olika teckensträngar matas ut
class MataUtTeckenstrangar
{
    public static void main (String[] args)
    {
        // en behållare med en utmatningsarea
        // och ett utmatningsfält
        OutPanel    outPanel = new OutPanel ();

        // ett fönster
        JFrame    frame = new JFrame (" Mata ut teckensträngar");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (outPanel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en utmatningsarea och ett utmatningsfält visas. Två olika trådar matar ut olika teckensträngar till dessa textkomponenter.

gryningen.html

```
<html>

<head>
<title>gryningen</title>
</head>
<body bgcolor="#ffffff">

<center>
<h1>gryningen</h1>
<h3>dagningen  ljusningen  dagbräckningen</h3>
</center>

<br> <br> <br> <br>

<ul>
<a href="./skymningen.html"> skymningen</a>
</ul>

</body>
</html>
```

skymningen.html

```
<html>

<head>
<title>skymningen</title>
</head>

<body bgcolor="#ffffff">

<center>
<h1>skymningen</h1>
<h3>kvällningen  kvällsljuset  halvmörkret</h3>
</center>

<br> <br> <br> <br>

<a href="./gryning<ul>
en.html"> gryningen</a>
</ul>

</body>

</html>
```

VisaEnHTMLSida.java

Ett program som visar en HTML-sida

En HTML-sida kan visas i ett Javafönster. En länk i en HTML-sida kan användas för att övergå till en annan HTML-sida.

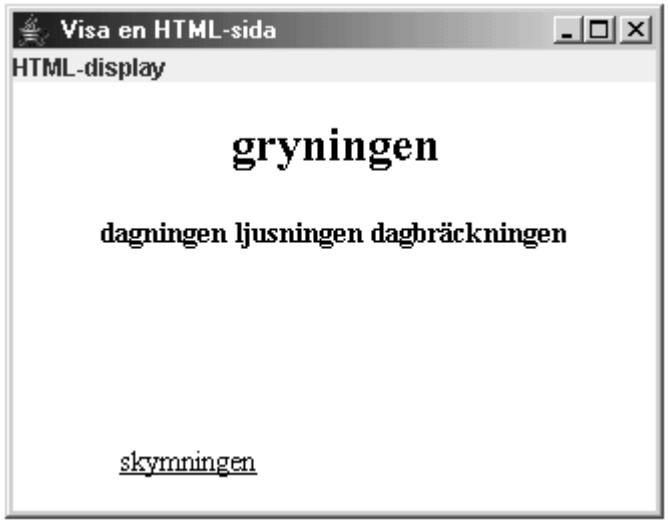
```
import java.awt.*;          // BorderLayout
import javax.swing.*;       // JEditorPane, JLabel, JPanel, JFrame
import java.net.*;          // URL
import javax.swing.event.*;  // HyperlinkEvent, HyperlinkListener,
                             // HyperlinkEvent.EventType
import java.io.*;            // IOException

// en behållare med en display för en HTML-sida
class HTMLPanel extends JPanel
{
    // en display för en HTML-sida
    private JEditorPane   htmlDisplay;

    // initiera den här behållaren
    public HTMLPanel ()
```

```
{
    // HTML-displayen
    htmlDisplay = new JEditorPane ();

    // texten i displayen ska inte redigeras
    htmlDisplay.setEditable (false);
    // (länkarna i en HTML-sida kan inte användas om man inte
    // gör så här)

    // etikett för displayen
    JLabel    etikett = new JLabel ("HTML-display");

    // startsidan
    try
    {
        // det HTML-dokument som ska visas i början
        URL    starturl = new URL ("file:./gryningen.html");
        // ett HTML-dokument definieras i en html-fil, som
        // finns i den aktuella datorn (i detta fall inleds
        // URL med file:), eller på någon plats på Internet
        // (i detta fall börjar URL med http:)

        // visa sidan
        htmlDisplay.setPage (starturl);
    }
    catch (IOException ex) // MalformedURLException,
                           // IOException
    // om en felaktig URL anges, eller om något problem
    // uppstår vid överföringen av motsvarande HTML-dokument
    {
        // visa undantag
        htmlDisplay.setText (ex.toString ());
    }

    // aktivera länkar
    HyperlinkListener  linkListener = new HyperlinkListener ()
        {
            // när användaren kommer in i en länk, klickar på
            // en länk eller lämnar en länk
            public void hyperlinkUpdate (HyperlinkEvent e)
            {
                // när användaren klickar på länken
                if (e.getEventType () ==
                          HyperlinkEvent.EventType.ACTIVATED)
                // (förutom konstanten ACTIVATED, finns också
                // konstanterna ENTERED och EXITED)
                {
                    try
                    {
                        // det valda HTML-dokumentet
                        URL    url = e.getURL ();
```

```
                            // visa motsvarande HTML-sida
                            htmlDisplay.setPage (url);
                        }
                        catch (IOException ex)
                        {
                            htmlDisplay.setText (ex.toString ());
                        }
                    }
                }
            };

        htmlDisplay.addHyperlinkListener (linkListener);

        // placera HTML-displayen i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (etikett, "North");
        this.add (htmlDisplay, "Center");
    }
}

// ett program som visar en HTML-sida
class VisaEnHTMLSida
{
    public static void main (String[] args)
    {
        // en behållare med en display för en HTML-sida
        HTMLPanel    htmlPanel = new HTMLPanel ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" Visa en HTML-sida");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);
        frame.add (htmlPanel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en display för en HTML-sida visas. När användaren klickar på länken i den visade HTML-sidan, övergår programmet till en annan HTML-sida.



Textinmatning

MataInEnRad.java

Ett program som visar hur en text bestående av en rad matas in

En teckensträng bestående av en rad kan matas in via ett inmatningsfält i ett grafiskt användargränssnitt.

```
import java.awt.*;           // BorderLayout
import javax.swing.*;        // JTextField, JTextArea, JLabel,
                             // JScrollPane, JPanel, JFrame
import java.awt.event.*;     // ActionEvent, ActionListener

// en behållare som innehåller ett inmatningsfält
// och en utmatningsarea
class IOPanel extends JPanel
{
    // ett inmatningsfält
    private JTextField    inFalt;

    // en utmatningsarea
    private JTextArea    outArea;

    // initiera den här behållaren
```

```
public IOPanel ()
{
    // inmatningsfältet
    inFalt = new JTextField (20);
    // inmatningsfältet med 20 synliga positioner
    // (användaren kan skriva in hur många tecken som helst,
    // men inmatningsfältets bredd avgör hur många tecken
    // som syns)

    // en etikett för inmatningsfältet
    // (så att användaren vet vad som ska matas in)
    JLabel    inEtikett = new JLabel ("Input");

    // utmatningsarean
    outArea = new JTextArea (4, 6);

    // en etikett för utmatningsarean
    JLabel    outEtikett = new JLabel ("Output");

    // en lyssnare som reagerar på användarens inmatning via
    // inmatningsfältet
    // (användaren skriver in en rad i inmatningsfältet,
    // och trycker på returtangenten med markören i
    // inmatningsfältet)
    ActionListener    inLyssnare = new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                // hämta raden från inmatningsfältet
                String    rad = inFalt.getText ();

                // töm inmatningsfältet
                inFalt.setText("");

                // visa den inmatade raden i utmatningsarean
                outArea.append ("\n" + rad);
            }
        };

    // registrera lyssnaren hos inmatningsfältet
    inFalt.addActionListener (inLyssnare);

    // placera komponenterna i den här behållaren
    JPanel    outPanel = new JPanel ();
    outPanel.setLayout (new BorderLayout ());
    outPanel.add (outEtikett, "North");
    outPanel.add (new JScrollPane (outArea), "Center");
    JPanel    inPanel = new JPanel ();
    inPanel.add (inEtikett);
    inPanel.add (inFalt);
    this.setLayout (new BorderLayout ());
```

```
        this.add (inPanel, "South");
        this.add (outPanel, "Center");
    }
}

// ett program som visar hur en text bestående av en rad matas in
class MataInEnRad
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en behållare
        JFrame    frame = new JFrame (" Mata in en rad");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        IOPanel    ioPanel = new IOPanel ();
        frame.add (ioPanel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller ett inmatningsfält och en utmatningsarea visas. Det som användaren matar in via inmatningsfältet visas i utmatningsarean.

MataInFleraRelateradeUppgifter.java

Ett program som illustrerar hur flera relaterade uppgifter matas in

Flera inmatningsfält kan användas för att mata in flera relaterade uppgifter. För att kunna hämta de olika uppgifterna samtidigt från dessa inmatningsfält, kan en knapp användas.

```
import java.awt.*;           // BorderLayout, GridLayout
import javax.swing.*;        // JTextField, JPasswordField,
                             // JButton, JLabel,
                             // JTextArea, JPanel, JFrame
import java.awt.event.*;     // ActionEvent, ActionListener

// en behållare som innehåller flera inmatningsfält, en knapp
// och en utmatningsarea
class IOPanel extends JPanel
{
    // två inmatningsfält
    private JTextField    namnFalt;
    private JPasswordField    losenordFalt;

    // en knapp
    private JButton    hamtaUppgifter;

    // en utmatningsarea
    private JTextArea    outArea;

    // initiera den här behållaren
    public IOPanel ()
    {
        // inmatningsfälten
        namnFalt = new JTextField (16);
        losenordFalt = new JPasswordField (16);

        // etiketter för inmatningsfälten
        JLabel    namnEtikett = new JLabel ("Namn");
        JLabel    losenordEtikett = new JLabel ("Lösenord");

        // knappen
        hamtaUppgifter = new JButton ("Hämta uppgifter");

        // utmatningsarean
        outArea = new JTextArea (4, 6);

        // etiketten för utmatningsarean
        JLabel    outEtikett = new JLabel ("Inmatade uppgifter");
```

```
    // en lyssnare som reagerar när användaren trycker
    // på knappen (användaren skriver in olika uppgifter
    // i inmatningsfälten, och trycker därefter på knappen)
    ActionListener    inLyssnare = new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                // hämta uppgifterna från inmatningsfälten
                String    namn = namnFalt.getText ();
                char[]    losenordChars =
                                losenordFalt.getPassword ();
                String  losenord = new String (losenordChars);

                // töm inmatningsfälten
                namnFalt.setText("");
                losenordFalt.setText("");

                // visa de inmatade uppgifterna
                // i utmatningsarean
                outArea.append ("\n" + namn);
                outArea.append ("\n" + losenord);
                outArea.append ("\n");
            }
        };

    // registrera lyssnaren hos knappen
    hamtaUppgifter.addActionListener (inLyssnare);

    // placera komponenterna i den här behållaren
    JPanel    outPanel = new JPanel ();
    outPanel.setLayout (new BorderLayout ());
    outPanel.add (outEtikett, "North");
    outPanel.add (outArea, "Center");
    JPanel    panel1 = new JPanel ();
    panel1.setLayout (new GridLayout (2, 1));
    panel1.add (namnEtikett);
    panel1.add (losenordEtikett);
    JPanel    panel2 = new JPanel ();
    panel2.setLayout (new GridLayout (2, 1));
    panel2.add (namnFalt);
    panel2.add (losenordFalt);
    JPanel    panel3 = new JPanel ();
    panel3.add (hamtaUppgifter);
    JPanel    inPanel = new JPanel ();
    inPanel.add (panel1);
    inPanel.add (panel2);
    inPanel.add (panel3);
    this.setLayout (new BorderLayout ());
    this.add (inPanel, "South");
    this.add (outPanel, "Center");
}
```

```
}

// ett program som illustrerar hur flera
// relaterade uppgifter matas in
class MataInFleraRelateradeUppgifter
{
    public static void main (String[] args)
    {
        // en behållare med flera inmatningsfält, en knapp och en
        // utmatningsarea
        IOPanel    ioPanel = new IOPanel ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (
                       " Mata in flera relaterade uppgifter");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (ioPanel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller två inmatningsfält, en knapp och en utmatningsarea visas. Det som användaren matar in via inmatningsfälten visas i utmatningsarean. De angivna uppgifterna hämtas från inmatningsfälten när användaren trycker på knappen.

MataInFleraRader.java

Ett program som illustrerar hur en text bestående av flera rader matas in

En teckensträng bestående av flera rader kan matas in via en inmatningsarea i ett grafiskt användargränssnitt.

```
import java.awt.*;           // BorderLayout
import javax.swing.*;        // JTextArea, JLabel, JButton,
                             // JScrollPane, JPanel, JFrame
import java.awt.event.*;     // ActionEvent, ActionListener

// en behållare som innehåller en inmatningsarea, en knapp och en
// utmatningsarea
class IOPanel extends JPanel
{
    // en inmatningsarea, en knapp och en utmatningsarea
    private JTextArea    inArea;
    private JButton    hamtaText;
    private JTextArea    outArea;

    // initiera den här behållaren
    public IOPanel ()
    {
        // inmatningsarean
        inArea = new JTextArea (6, 16);
        inArea.setLineWrap (true);

        // en etikett för inmatningsarean
        JLabel    inEtikett = new JLabel ("Input");

        // knappen
        hamtaText = new JButton ("Hämta text");

        // utmatningsarean
        outArea = new JTextArea (6, 16);
        outArea.setLineWrap (true);

        // en etikett för utmatningsarean
        JLabel    outEtikett = new JLabel ("Output");

        // en lyssnare som reagerar på användarens inmatning via
        // inmatningsarean
        // (användaren skriver in en text i inmatningsarean,
        // markerar eventuellt en del av texten, och trycker på
        // knappen efter det)
        ActionListener    inLyssnare = new ActionListener ()
            {
```

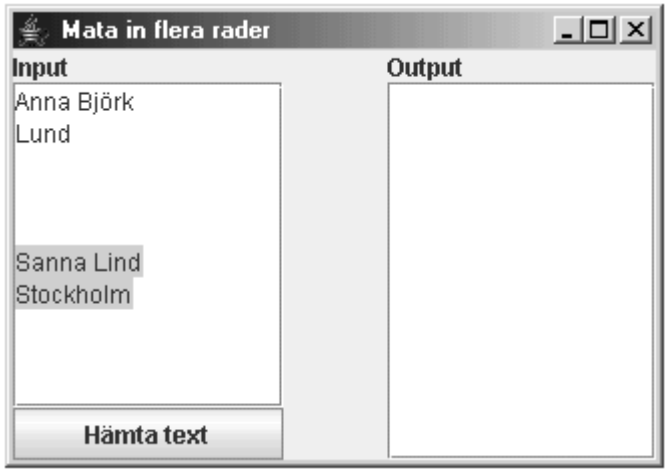
```
                public void actionPerformed (ActionEvent e)
                {
                    // hämta texten från inmatningsarean
                    String    inText = inArea.getText ();

                    // hämta den markerade texten
                    String    markeradText =
                                    inArea.getSelectedText ();

                    // den text som ska visas i utmatningsarean
                    // - om användaren markerat en text i
                    // inmatningsarean så blir det denna text,
                    // annars blir det hela texten
                    // i inmatningsarean
                    String    outText = "";
                    if (markeradText != null)
                        outText = markeradText;
                    else
                    {
                        outText = inText;

                       // töm inmatningsarean
                        inArea.setText("");
                    }

                    // visa den hämtade texten i utmatningsarean
                    outArea.setText (outText);
                }
            };

        // registrera lyssnaren hos knappen
        hamtaText.addActionListener (inLyssnare);

        // placera komponenterna i den här behållaren
        JPanel    inPanel = new JPanel ();
        inPanel.setLayout (new BorderLayout ());
        inPanel.add (inEtikett, "North");
        inPanel.add (new JScrollPane (inArea), "Center");
        inPanel.add (hamtaText, "South");
        JPanel    outPanel = new JPanel ();
        outPanel.setLayout (new BorderLayout ());
        outPanel.add (outEtikett, "North");
        outPanel.add (new JScrollPane (outArea), "Center");
        this.setLayout (new BorderLayout ());
        this.add (inPanel, "West");
        this.add (outPanel, "East");
    }
}

// ett program som illustrerar hur en text bestående av
// flera rader matas in
```

```
class MataInFleraRader
{
    public static void main (String[] args)
    {
        // en behållare med en inmatningsarea, en knapp och en
        // utmatningsarea
        IOPanel    ioPanel = new IOPanel ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" Mata in flera rader");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (ioPanel);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en inmatningsarea, en knapp och en utmatningsarea visas. Användaren skriver in en text i inmatningsarean, och trycker på knappen. Texten överförs då från inmatningsarean till utmatningsarean.

Om användaren markerar en text i inmatningsarean och trycker på knappen, visas den markerade texten i utmatningsarean.



Bevaka ändringar i ett dokument

BevakaEttDokumentsAndringar.java

Ett program som visar hur ändringar i ett dokument bevakas

En grafisk textkomponent används för att visa och för att redigera ett textdokument. Det är möjligt att bevaka ändringar i ett textdokument, och reagera på dessa ändringar.

```
import java.awt.*;           // BorderLayout
import javax.swing.*;        // JTextField, JTextArea, JLabel,
                             // JScrollPane, JPanel, JFrame
import javax.swing.text.*;   // Document
import javax.swing.event.*;  // DocumentEvent, DocumentListener

// en behållare som innehåller ett inmatningsfält
// och en utmatningsarea
class IOPanel extends JPanel
{
    // ett inmatningsfält
    private JTextField    inFalt;

    // en utmatningsarea
    private JTextArea    outArea;

    // initiera den här behållaren
    public IOPanel ()
    {
        // inmatningsfältet
        inFalt = new JTextField (20);
        JLabel    inEtikett = new JLabel ("Input");

        // utmatningsarean
        outArea = new JTextArea (4, 6);
        outArea.setLineWrap (true);
        outArea.setWrapStyleWord (true);
        JLabel    outEtikett = new JLabel ("Output");

        // en lyssnare som reagerar på ändringar av texten
        // (dokumentet) i textfältet
        DocumentListener    inLyssnare = new DocumentListener ()
            {
                // när någonting läggs till i dokumentet
                public void insertUpdate (DocumentEvent e)
                {
                    // texten i inmatningsfältet
```

```
                String    text = inFalt.getText ();

                // visa texten i utmatningsarean
                outArea.setText ("\n" + text);
            }

            // när någonting tas bort från dokumentet
            public void removeUpdate (DocumentEvent e)
            {
                // texten i inmatningsfältet
                String    text = inFalt.getText ();

                // visa texten i utmatningsarean
                outArea.setText ("\n" + text);
            }

            // när dokumentets attribut ändras
            // (detta kan inte ske i samband med ett textfält)
            public void changedUpdate (DocumentEvent e)
            {}
        };

        // registrera lyssnaren hos inmatningsfältets dokument
        Document    dokument = inFalt.getDocument ();
        dokument.addDocumentListener (inLyssnare);
        // kortare:
        // inFalt.getDocument ().addDocumentListener (inLyssnare);

        // placera komponenterna i den här behållaren
        JPanel    outPanel = new JPanel ();
        outPanel.setLayout (new BorderLayout ());
        outPanel.add (outEtikett, "North");
        outPanel.add (new JScrollPane (outArea), "Center");
        JPanel    inPanel = new JPanel ();
        inPanel.add (inEtikett);
        inPanel.add (inFalt);
        this.setLayout (new BorderLayout ());
        this.add (inPanel, "South");
        this.add (outPanel, "Center");
    }
}

// ett program som visar hur ett dokuments ändringar bevakas
class BevakaEttDokumentsAndringar
{
    public static void main (String[] args)
    {
        // en behållare med ett inmatningsfält
        // och en utmatningsarea
        IOPanel    ioPanel = new IOPanel ();
```

```
        // ett fönster
        JFrame    frame =
               new JFrame (" Bevaka ett dokuments ändringar");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (ioPanel);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller ett inmatningsfält och en utmatningsarea visas. Det som användaren skriver in i inmatningsfältet visas omedelbart i utmatningsarean.

Val mellan olika alternativ

Komponenter med två lägen

Knappar.java

Ett program som illustrerar knappar

En knapp kan bindas till ett alternativ i ett program. Detta alternativ kan i så fall väljas genom att knappen klickas.

```
import java.awt.*;           // Graphics, Graphics2D,
                             // Font, Dimension
import javax.swing.*;        // JButton, JTextArea,
                             // JSplitPane, JFrame
import java.awt.geom.*;      // Line2D, Line2D.Double
import java.awt.event.*;     // ActionListener

// en knapp med en pil på
class ArrowButton extends JButton
{
    // pilens möjliga riktningar
    public static final int    RIGHT = 1;
    public static final int    LEFT = 2;

    // den här pilens riktning
    private int    riktning;

    // initiera knappen
    public ArrowButton (int riktning)
    {
        this.riktning = riktning;
    }

    // rita en pil på knappen
    public void paintComponent (Graphics gr)
    {
        super.paintComponent (gr);

        // verktyg att rita på den här knappen
        Graphics2D    g = (Graphics2D) gr;

        // knappens dimensioner
        int    w = this.getWidth ();
        int    h = this.getHeight ();

        // en pil i mitten
        int    x1 = 25 * w / 100;
```

```
        int    y1 = h / 2;
        int    x2 = 75 * w / 100;
        int    y2 = h / 2;
        Line2D    line = new Line2D.Double (x1, y1, x2, y2);
        Line2D    linea = null;
        Line2D    lineb = null;

        if (riktning == RIGHT)
        {
            linea = new Line2D.Double (x2 - 6, y2 - 3, x2, y2);
            lineb = new Line2D.Double (x2 - 6, y2 + 3, x2, y2);
        }
        else if (riktning == LEFT)
        {
            linea = new Line2D.Double (x1 + 6, y2 - 3, x1, y1);
            lineb = new Line2D.Double (x1 + 6, y2 + 3, x1, y1);
        }

        // rita pilen
        g.draw (line);
        g.draw (linea);
        g.draw (lineb);
    }
}

// en behållare som innehåller två knappar och en textarea
class FSplitPane extends JSplitPane
{
    // två knappar
    private JButton    b1;
    private JButton    b2;

    // en textarea
    private JTextArea    textArea;

    // val
    private int    val = 1;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // knapparna
        b1 = new ArrowButton (ArrowButton.RIGHT);
        b2 = new ArrowButton (ArrowButton.LEFT);
        b1.setPreferredSize (new Dimension (70, 25));
```

```
        b2.setPreferredSize (new Dimension (70, 25));

        // På en knapp kan en text eller en ikon placeras,
        // eller både en text och en ikon.
        // Metoderna setText och setIcon används för detta
        // ändamål.

        // textarean
        final String[] alternativ = {"v i n t e r n", "v å r e n",
                                     "s o m m a r e n", "h ö s t e n"};
        // final för att kunna användas i en inre klass
        textArea = new JTextArea (4, 6);
        Font    font =
           new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        textArea.setText ("\n  " + alternativ[val]);


        // en lyssnare som reagerar på användarens val
        // via knapparna
        ActionListener    lyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    Object    source = e.getSource ();

                    // gå till nästa eller föregående alternativ,
                    // beroende på vilken knapp användaren tryckt
                    // på
                    if (source == b1)
                        val = ++val % alternativ.length;
                    else if (source == b2)
                        val =
                       (val > 0)? --val : (alternativ.length - 1);

                    textArea.setText ("\n  " + alternativ[val]);
                }
            };

        // registrera lyssnaren hos knapparna
        b1.addActionListener (lyssnare);
        b2.addActionListener (lyssnare);
        // olika lyssnare kan användas för olika knappar

        // placera komponenterna i den här behållaren
        this.setTopComponent (textArea);
        JPanel    panel = new JPanel ();
        panel.add (b1);
        panel.add (b2);
        this.setBottomComponent (panel);
    }
```

```
}

// ett program som illustrerar knappar
class Knappar
{
    public static void main (String[] args)
    {
        // en behållare med två knappar och en textarea
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" Knappar");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (splitPane);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller två knappar med piler och en textarea visas. Genom att klicka på en knapp, ändrar användaren det som visas i textarean.

Kryssrutor.java

Ett program som illustrerar kryssrutor

En kryssruta är en ruta, som kan vara markerad eller inte. En kryssruta markeras med en klickning med musen i den. Om användaren klickar i en markerad kryssruta, försvinner kryssrutans markering. En kryssruta kan användas för att symbolisera ett visst alternativ. Detta alternativ väljs genom att kryssrutan markeras.

En uppsättning kryssrutor kan användas för att symbolisera ett antal relaterade alternativ. Dessa kryssrutor är i så fall oberoende av varandra: om en av kryssrutorna markeras, påverkas inte de andra kryssrutorna på något sätt.

```
import java.awt.*;           // Font
import javax.swing.*;        // JCheckBox, JTextArea,
                             // JSplitPane, JFrame
import java.awt.event.*;     // ActionListener

// en behållare som innehåller en uppsättning kryssrutor
// och en textarea
class FSplitPane extends JSplitPane
{
    // kryssrutor
    private JCheckBox[]    cb;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // kryssrutorna
        String[]    symboler = {" en", " två", " tre", " fyra"};
        cb = new JCheckBox [4];
        for (int i = 0; i < cb.length; i++)
            cb[i] = new JCheckBox (symboler[i]);
        // även ikoner kan användas i samband med kryssrutor
        // (enbart ikoner, eller tillsammans med text)

        // markera en kryssruta redan från början
```

```
        cb[1].setSelected (true);

        // textarean
        final String[] alternativ = {"v i n t e r n", "v å r e n",
                                     "s o m m a r e n", "h ö s t e n"};
        textArea = new JTextArea (4, 6);
        Font   font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        textArea.setText ("\n  " + alternativ[1]);

        // en lyssnare som reagerar på användarens val
        // via kryssrutor
        ActionListener    lyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    // stega igenom kryssrutorna, kontrollera
                    // om de är markerade och bilda en
                    // teckensträng som avspeglar kryssrutornas
                    // tillstånd
                    String    s = "\n";
                    for (int i = 0; i < cb.length; i++)
                        if (cb[i].isSelected ())
                            s += "    " + alternativ[i];

                    // visa strängen
                    textArea.setText (s);
                }
            };

        // registrera lyssnaren hos kryssrutorna
        for (int i = 0; i < cb.length; i++)
            cb[i].addActionListener (lyssnare);

        // placera komponenterna i den här behållaren
        this.setTopComponent (new JScrollPane (textArea));
        JPanel    panel = new JPanel ();
        for (int i = 0; i < cb.length; i++)
            panel.add (cb[i]);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar kryssrutor
class Kryssrutor
{
    public static void main (String[] args)
    {
        // en behållare med en uppsättning kryssrutor
        // och en textarea
```

```
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
        JFrame    frame = new JFrame (" Kryssrutor");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (420, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en uppsättning kryssrutor och en textarea visas. Genom att markera ett antal kryssrutor, väljer användaren det som ska visas i textarean.

Radioknappar.java

Ett program som illustrerar radioknappar

En radioknapp är en knapp som kan vara antingen på eller av. En radioknapp markeras genom att användaren klickar på den. Om användaren klickar på en radiokapp som är på, övergår radioknappen till sitt av-läge.

En radioknapp kan bindas till ett alternativ i ett program. I så fall väljs detta alternativ genom att motsvarande radioknapp sätts i på-läge.

En uppsättning radioknappar kan användas, för att välja ett av flera tillgängliga alternativ. En grupp radioknappar kan skapas, där bara en knapp i taget kan vara på.

```
import java.awt.*;           // Font
import javax.swing.*;        // JRadioButton,
                             // ButtonGroup, JTextArea,
                             // JSplitPane, JFrame
import java.awt.event.*;     // ActionListener

// en behållare som innehåller en grupp radioknappar
// och en textarea
class FSplitPane extends JSplitPane
{
    // radioknappar
    private JRadioButton[]    rb;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // radioknapparna
        String[]    symboler = {" en", " två", " tre", " fyra"};
        rb = new JRadioButton [4];
        for (int i = 0; i < rb.length; i++)
            rb[i] = new JRadioButton (symboler[i]);
        // även ikoner kan användas i samband med radioknappar
        // (bara ikoner, eller tillsammans med text)

        // välj ett alternativ redan från början
        rb[1].setSelected (true);

        // gruppera radioknapparna
        // (så att bara en knapp i taget kan vara på)
        ButtonGroup    bg = new ButtonGroup ();
        for (int i = 0; i < rb.length; i++)
            bg.add (rb[i]);

        // textarean
```

```
        final String[] alternativ = {"v i n t e r n", "v å r e n",
                                     "s o m m a r e n", "h ö s t e n"};
        textArea = new JTextArea (4, 6);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        textArea.setText ("\n  " + alternativ[1]);

        // en lyssnare som reagerar på användarens val via
        // radioknapparna
        ActionListener    lyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    String    s = "";
                    if (rb[0].isSelected ())
                        s = alternativ[0];
                    else if (rb[1].isSelected ())
                        s = alternativ[1];
                    else if (rb[2].isSelected ())
                        s = alternativ[2];
                    else if (rb[3].isSelected ())
                        s = alternativ[3];

                    textArea.setText ("\n  " + s);
                }
            };

        // registrera lyssnaren hos radioknapparna
        for (int i = 0; i < rb.length; i++)
            rb[i].addActionListener (lyssnare);
        // olika lyssnare kan användas för olika radioknappar

        // placera komponenterna i den här behållaren
        this.setTopComponent (textArea);
        JPanel    panel = new JPanel ();
        for (int i = 0; i < rb.length; i++)
            panel.add (rb[i]);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar radioknappar
class Radioknappar
{
    public static void main (String[] args)
    {
        // en behållare med en grupp radioknappar och en textarea
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
```

```
        JFrame    frame = new JFrame (" Radioknappar");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en grupp radioknappar och en textarea visas. Genom att markera en radioknapp i gruppen, väljer användaren vad som ska visas i textarean.

Symbolbehållare

EnValruta.java

Ett program som illustrerar en valruta

En valruta är en ruta som innehåller en knapp och ett antal symboler. Dessa symboler representerar olika alternativ i ett program.

Användaren kan öppna en valruta via dess knapp och välja en av dess symboler. Efter valet stängs valrutan, och det alternativ i programmet som motsvarar den valda symbolen kan aktiveras.

```
import java.awt.*;           // Font
import javax.swing.*;        // JComboBox, JTextArea, JSplitPane,
                             // JFrame
import java.awt.event.*;     // ActionListener

// en behållare som innehåller en valruta och en textarea
class FSplitPane extends JSplitPane
{
    // en valruta
    private JComboBox    valRuta;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // valrutan
        String[]    symboler = {" en", " två", " tre", " fyra"};
        valRuta = new JComboBox (symboler);
        // teckensträngar eller/och ikoner kan användas
        // som symboler i en valruta

        // välj en symbol redan från början
        valRuta.setSelectedIndex (1);
        // valBox.setSelectedItem (symboler[1]);


        // textarean
        final String[] alternativ = {"v i n t e r n", "v å r e n",
                                     "s o m m a r e n", "h ö s t e n"};
        textArea = new JTextArea (4, 6);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        textArea.setText ("\n  " +
                        alternativ[valRuta.getSelectedIndex ()]);

        // en lyssnare, som reagerar på användarens val
        // via valrutan
```

```
        ActionListener    lyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    JComboBox source = (JComboBox) e.getSource ();

                    // det valda indexet
                    int valdaIndexet = source.getSelectedIndex ();

                    // aktiverar motsvarande alternativ
                    // i programmet
                    textArea.setText (
                        "\n  " + alternativ[valdaIndexet]);
                }
            };

        // registrera lyssnaren hos valrutan
        valRuta.addActionListener (lyssnare);

        // placera komponenterna i den här behållaren
        this.setTopComponent (textArea);
        JPanel    panel = new JPanel ();
        panel.add (valRuta);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar en valruta
class EnValruta
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en behållare
        JFrame    frame = new JFrame (" En valruta");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (new FSplitPane ());
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en valruta och en textarea visas. Genom att välja en symbol via valrutan, kan användaren välja vad som ska visas i textarean.

EnFlexibelValruta.java

Ett program som illustrerar en flexibel valruta

En valruta kan utformas så att det blir möjligt att mata in egna symboler via den. Den symbol som användaren matar in blir den valda symbolen. På så sätt skapas en valruta som kan användas både för att välja bland ett antal fördefinierade symboler och för att mata in egna symboler. En flexibel valruta skapas, som kombinerar val och inmatning.

Flexibiliteten hos en valruta kan utökas genom att man gör det möjligt för användaren att ändra valrutans innehåll. Användaren ska kunna lägga till nya symboler i valrutan och ta bort redan befintliga symboler från valrutan. Till exempel ska de symboler som användaren matar in via valrutan, kunna läggas till i valrutan.

```
import java.awt.*;           // Font
import javax.swing.*;        // JComboBox, JTextArea, JSplitPane,
                             // JFrame
import java.awt.event.*;     // ActionListener

// en behållare som innehåller en flexibel valruta och en textarea
class FSplitPane extends JSplitPane
{
    // en flexibel valruta
    private JComboBox    valRuta;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // den flexibla valrutan
        String[]    symboler = {" 30", " 35", " 40", " 45"};
        valRuta = new JComboBox (symboler);
        // låt användaren kunna mata in egna symboler
        valRuta.setEditable (true);

        // textarean
        textArea = new JTextArea (4, 6);
        Font  font = new Font ("Serif", Font.BOLD + Font.ITALIC,
```

```
                    Integer.parseInt (symboler[0].trim ()));
        textArea.setFont (font);
        textArea.setText ("\n  s a n n i n g e n");

        // en lyssnare som reagerar på användarens val via den
        // flexibla valrutan
        ActionListener    lyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    JComboBox source = (JComboBox) e.getSource ();

                    // den valda symbolen
                    String    symbol =
                               (String) source.getSelectedItem ();

                    // om symbolen inte finns i valrutan,
                    // lägg till den
                    int valdaIndexet = source.getSelectedIndex ();
                    if (valdaIndexet < 0)
                        source.addItem (symbol);

                    // aktivera motsvarande alternativ
                    // i programmet
                    int  size = Integer.parseInt (symbol.trim ());
                    textArea.setFont (new Font ("Serif",
                                 Font.BOLD + Font.ITALIC, size));
                }
            };

        // registrera lyssnaren hos den flexibla valrutan
        valRuta.addActionListener (lyssnare);

        // placera komponenterna i den här behållaren
        this.setTopComponent (textArea);
        JPanel    panel = new JPanel ();
        panel.add (valRuta);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar en flexibel valruta
class EnFlexibelValruta
{
    public static void main (String[] args)
    {
        // en behållare med en flexibel valruta och en textarea
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
        JFrame    frame = new JFrame (" En flexibel valruta");
```

```
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en flexibel valruta och en textarea visas. Genom att välja en symbol från valrutan kan användaren välja storlek på det som visas i textarean. Användaren kan också mata in en storlek via valrutan. Denna storlek läggs då till i valrutan.

EnLista.java

Ett program som illustrerar en lista

En lista är en ruta som innehåller ett antal symboler. När en lista visas, visas också de symboler som finns i listan. Användaren kan välja en eller flera av dessa symboler genom att klicka på dem. Olika symboler i en lista kan bindas till ett antal alternativ i ett program. Användaren kan då välja

ett eller flera av dessa alternativ genom att välja motsvarande symboler i listan.

```
import java.awt.*;           // Font
import javax.swing.*;        // JList, ListSelectionModel,
                             // JTextArea,
                             // JSplitPane, JScrollPane, JFrame
import javax.swing.event.*;  // ListSelectionEvent,
                             // ListSelectionListener

// en behållare som innehåller en lista och en textarea
class FSplitPane extends JSplitPane
{
    // en lista
    private JList    list;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.HORIZONTAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (100);

        // listan
        Object[]    symboler =
                        {" ett", " två", " tre", " fyra", " fem",
                         " sex", " sju", " åtta", " nio", " tio"};
        list = new JList (symboler);

        // Förvalt beteende: det går att välja hur många
        // symboler som helst.
        // För att välja flera symboler håller användaren
        // Ctrl-tangenten nedtryckt.
        // För att välja alla symboler i ett intervall, väljer
        // användaren den första symbolen i intervallet,
        // trycker ned Shift-tangenten, och väljer sedan den
        // sista symbolen i intervallet)
        // list.setSelectionMode (
        //        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);

        // Val kan begränsas till endast ett intervall, eller till
        // endast en symbol, genom att en av följande konstanter
        // väljs:
        // SINGLE_INTERVAL_SELECTION eller SINGLE_SELECTION.

        // start-val
```

```
        list.setSelectedIndex (4);

        // textarean
        textArea = new JTextArea (4, 6);
        Font    font =
                  new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);

        // visa den symbol som användaren valt i listan
        Object    valdaSymbolen = list.getSelectedValue ();
        textArea.setText (" " + valdaSymbolen);

        // en lyssnare som reagerar på användarens val i listan
        list.addListSelectionListener(new ListSelectionListener ()
           {
               public void valueChanged (ListSelectionEvent e)
               {
                   JList    source = (JList) e.getSource ();

                   // valda symboler
                   // (användaren kan välja hur många symboler
                   // som helst)
                   Object[]    valdaSymboler =
                                    list.getSelectedValues ();
                   // även metoderna getSelectedIndex och
                   // getSelectedIndices kan användas

                   // en sträng som innehåller valda symboler
                   String    val = "";
                   for (int i = 0; i < valdaSymboler.length; i++)
                       val = val + " " + valdaSymboler[i] + "\n";

                   // visa valda symboler
                   textArea.setText (val);
               }
           } );


        // placera komponenterna i den här behållaren
        this.setLeftComponent (new JScrollPane (list));
        this.setRightComponent (new JScrollPane (textArea));
    }
}

// ett program som illustrerar en lista
class EnLista
{
    public static void main (String[] args)
    {
        // en behållare med en lista och en textarea
        FSplitPane    splitPane = new FSplitPane ();
```

```
        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" En lista");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (splitPane);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en lista och en textarea visas. Användaren kan använda mus (och eventuellt `Ctrl`-tangenten eller `Shift`-tangenten), och välja en eller flera symboler från listan. Så snart en uppsättning symboler i listan väljs, visas dessa symboler i textarean.

EnForanderligLista.java

Ett program som illustrerar en föränderlig lista

En listas innehåll kan ändras under programmets gång.

```
import java.awt.*;           // Font
import javax.swing.*;        // JList, DefaultListModel,
                             // JTextArea, JTextField,
                             // JSplitPane, JScrollPane,
                             // JFrame
import java.awt.event.*;     // ActionListener
```

```
import javax.swing.event.*;  // ListSelectionEvent,
                             // ListSelectionListener

// en behållare som innehåller en föränderlig lista,
// en textarea och ett textfält
class FSplitPane extends JSplitPane
{
    // en lista
    private JList    list;

    // en textarea
    private JTextArea    textArea;

    // ett textfält
    JTextField    textFalt;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (6);

        // en listmodell - listans innehåll kan ändras via
        // den här modellen
        final DefaultListModel    listModel =
                                  new DefaultListModel ();
        // final för att kunna användas i en inre klass
        listModel.addElement (new String ("helheten"));
        listModel.addElement (new String ("sanningen"));
        listModel.addElement (new String ("balansen"));
        listModel.addElement (new String ("kärleken"));

        // listan
        list = new JList (listModel);
        list.setSelectedIndex (0);

        // textarean
        textArea = new JTextArea (4, 6);
        Font    font =
           new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
        textArea.setFont (font);
        Object    valdaSymbolen = list.getSelectedValue ();
        textArea.setText (" " + valdaSymbolen);

        // en lyssnare som reagerar på användarens val i listan
        list.addListSelectionListener(new ListSelectionListener ()
           {
               public void valueChanged (ListSelectionEvent e)
               {
```

```
            JList    source = (JList) e.getSource ();

            // valda symboler
            Object[]    valdaSymboler =
                           list.getSelectedValues ();

            // en sträng som innehåller valda symboler
            String    val = "";
            for (int i = 0; i < valdaSymboler.length; i++)
                val = val + " " + valdaSymboler[i] + "\n";

            // visa valda symboler
            textArea.setText (val);
        }
    });

// textfältet
JLabel    label = new JLabel ("Lägg till i listan:");
textFalt = new JTextField (20);
JPanel    panel = new JPanel ();
panel.add (label);
panel.add (textFalt);

// ändra listan efter inmatningen via textfältet
textFalt.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // hämta texten från textfältet
            String    symbol = textFalt.getText ();
            textFalt.setText ("");

            // ändra listan
            if (!symbol.equals (""))
            {
                // lägg till den inmatade symbolen
                // i listan
                listModel.addElement (symbol);

                // ta bort den första symbolen från listan
                // (om så önskas)
                listModel.remove (0);
            }
        }
    });


// placera komponenterna i den här behållaren
JSplitPane    splitPane = new JSplitPane (
      JSplitPane.HORIZONTAL_SPLIT,
      new JScrollPane (list), new JScrollPane (textArea));
```

```
        splitPane.setContinuousLayout (true);
        splitPane.setDividerSize (4);
        this.setTopComponent (splitPane);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar en föränderlig lista
class EnForanderligLista
{
    public static void main (String[] args)
    {
        // en behållare med en lista, en textarea och ett textfält
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
        JFrame    frame = new JFrame (" En föränderlig lista");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en lista, en textarea och ett textfält visas. Det som användaren väljer i listan, visas i textarean.

Det som användaren matar in via textfältet, läggs till i listan. Så snart en ny symbol läggs till i listan, tas listans första symbol bort. På så sätt förblir listans längd oförändrad.

EnListaMedBilder.java

Ett program som illustrerar en lista med bilder

En lista kan ha bilder som sina symboler.

```
import java.awt.*;           // Graphics2D, BasicStroke, Color,
                             // Font
import java.awt.image.*;     // BufferedImage
import java.awt.geom.*;      // Rectangle2D, Rectangle2D.Double,
                             // Ellipse2D, Ellipse2D.Double
import javax.swing.*;        // JList, ListSelectionModel, JPanel,
                             // JSplitPane, JFrame
import javax.swing.event.*;  // ListSelectionEvent,
                             // ListSelectionListener

// en klass som definierar en typ av bilder
class FImage extends BufferedImage
{
    public FImage (
        int      w,              // bildens bredd
        int      h,              // bildens höjd
        Color    ellipseColor)   // ellipsens färg
    {
        // skicka uppgifter om bildens dimensioner och typ
        // till superklassens konstruktor
        super (w, h, BufferedImage.TYPE_INT_ARGB);
```

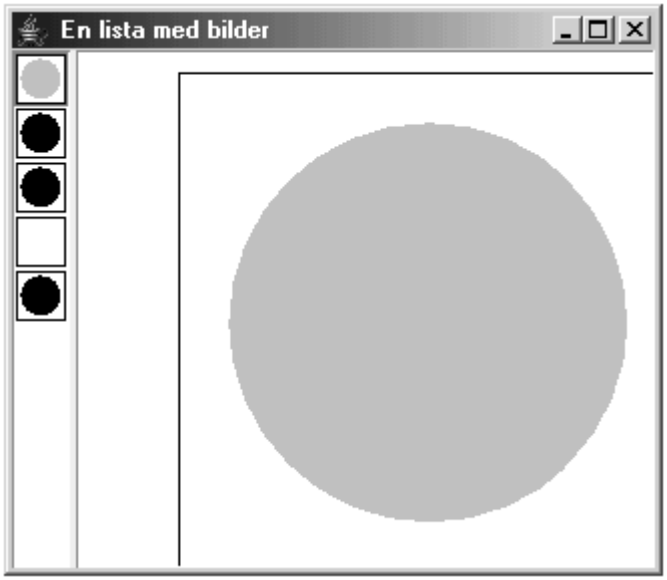
```
        // verktyg att rita och måla den här bilden
        Graphics2D    g = this.createGraphics ();
        g.setStroke (new BasicStroke (2.0f));

        // måla hela ytan och rita ram för den här bilden
        Rectangle2D    ram = new Rectangle2D.Double (0, 0, w, h);
        g.setPaint (Color.WHITE);
        g.fill (ram);
        g.setColor (Color.BLACK);
        g.draw(ram);

        // måla en ellips i mitten
        Ellipse2D    el = new Ellipse2D.Double ();
        el.setFrameFromDiagonal (w / 10, h / 10,
                                 9 * w / 10, 9 * h / 10);
        g.setPaint (ellipseColor);
        g.fill (el);

        // frigör de resurser som tas upp av
        // den grafiska kontexten
        g.dispose ();
    }
}

// en panel som visar en bild (en bild-display)
class FPanel extends JPanel
{
    // bild att visa på den här panelen
    private BufferedImage    image;

    public void paintComponent (Graphics gr)
    {
        // förberedande handlingar
        super.paintComponent (gr);
        this.setBackground (Color.WHITE);
        Graphics2D    g = (Graphics2D) gr;

        // visa bilden
        g.drawImage (image, null, 50, 10);

    }

    // ange panelens bild
    public void setImage (BufferedImage image)
    {
        this.image = image;

        this.repaint ();
    }
}
```

```
// en behållare som innehåller en lista och en panel
class FSplitPane extends JSplitPane
{
    // en lista
    private JList    list;

    // en panel
    private FPanel    panel;

    // bilder som ska visas i panelen
    private FImage[]    images;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.HORIZONTAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);


        // färger för bilderna
        Color[]    c = { Color.LIGHT_GRAY, Color.BLUE, Color.RED,
                         Color.YELLOW, Color.MAGENTA };

        // bilderna som ska visas
        images = new FImage[5];
        for (int i = 0; i < images.length; i++)
            images[i] = new FImage (250, 250, c[i]);

        // ikoner med bilder
        ImageIcon[]    symboler = new ImageIcon[images.length];
        for (int i = 0; i < symboler.length; i++)
            symboler[i] =
                     new ImageIcon (new FImage (25, 25, c[i]));

        // listan med ikoner som symboler
        list = new JList (symboler);
        list.setSelectedIndex (0);
        list.setSelectionMode (
                          ListSelectionModel.SINGLE_SELECTION);

        // panelen med bilden
        panel = new FPanel ();
        int    index = list.getSelectedIndex ();
        panel.setImage (images[index]);

        // en lyssnare som reagerar på användarens val i listan
        list.addListSelectionListener(new ListSelectionListener ()
           {
               public void valueChanged (ListSelectionEvent e)
```

```
                {
                    // det valda indexet
                    int    valdaIndexet = list.getSelectedIndex ();

                    // visa den valda bilden
                    panel.setImage (images[valdaIndexet]);
                }
            } );

        // placera komponenterna i den här behållaren
        this.setLeftComponent (list);
        this.setRightComponent (panel);
    }
}

// ett program som illustrerar en lista med bilder
class EnListaMedBilder
{
    public static void main (String[] args)
    {
        // en behållare med en lista och en panel
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
        JFrame    frame = new JFrame (" En lista med bilder");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en panel och en lista med flera bilder visas. Den bild som användaren väljer via listan, visas i panelen.



EnValLinjal.java

Ett program som illustrerar en vallinjal

En vallinjal är en linjal med en flyttbar "punkt". Genom att flytta denna punkt längs linjalen kan användaren ändra linjalens värde. På så sätt kan ett heltalsvärde i ett givet intervall väljas (valet kan utföras även genom att användaren klickar någonstans på linjalen, eller håller musen nedtryckt på en plats på linjalen).

Olika värden på en vallinjal kan bindas till olika alternativ i ett program. Ett alternativ kan då väljas genom att motsvarande värde på vallinjalen väljs.

```
import java.awt.*;           // Font, Color
import javax.swing.*;        // JSlider, JTextArea,
                             // JSplitPane, JFrame
import javax.swing.event.*;  // ChangeEvent, ChangeListener
import java.util.*;          // Hashtable

// en behållare som innehåller två vallinjaler och en textarea
class FSplitPane extends JSplitPane
{
```

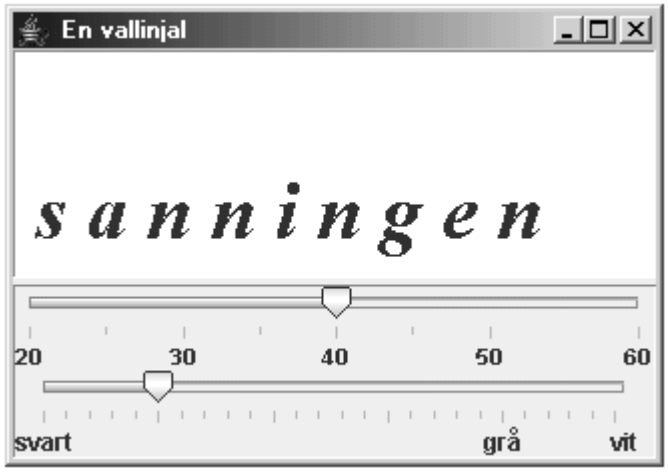
```
// två vallinjaler
private JSlider    fontLinjal;
private JSlider    fargLinjal;

// en textarea
private JTextArea    display;

// initiera den här behållaren
public FSplitPane ()
{
    // justera inställningar i den här behållaren
    super (JSplitPane.VERTICAL_SPLIT);
    this.setContinuousLayout (true);
    this.setDividerSize (5);
    this.setDividerLocation (200);

    // textarean
    display = new JTextArea (4, 6);
    display.setBackground (Color.YELLOW);
    Font    font =
             new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
    display.setFont (font);
    display.setText ("\n s a n n i n g e n");

    // vallinjalen för font
    fontLinjal = new JSlider (
                      20,    // minnimum
                      60,    // maximum
                      40);   // startval
    // placeringen av de korta strecken
    fontLinjal.setMinorTickSpacing (5);
    // placeringen av de långa strecken
    fontLinjal.setMajorTickSpacing (10);
    // strecken ska vara synliga
    fontLinjal.setPaintTicks (true);
    // märkena ska vara synliga
    fontLinjal.setPaintLabels (true);
    // reagera på val via vallinjalen
    fontLinjal.addChangeListener (new ChangeListener ()
        {
            public void stateChanged (ChangeEvent e)
            {
                // det valda värdet
                int    vardet = fontLinjal.getValue ();

                // justera font i textarean
                Font    font = new Font (
                    "Serif", Font.BOLD + Font.ITALIC, vardet);
                display.setFont (font);
            }
        } );
```

```
        // en vertikal vallinjal kan skapas så här:
        //fontLinjal = new JSlider (JSlider.VERTICAL, 20, 60, 40);

        // en hashtabell med etiketter
        Hashtable<Integer, JComponent>    etiketter =
                new Hashtable<Integer, JComponent> ();
        etiketter.put (new Integer (0), new JLabel ("svart"));
        etiketter.put (new Integer (200), new JLabel ("grå"));
        etiketter.put (new Integer (255), new JLabel ("vit"));

        // vallinjalen för färg
        fargLinjal = new JSlider (0, 255, 50);
        fargLinjal.setMinorTickSpacing (10);
        fargLinjal.setMajorTickSpacing (50);
        fargLinjal.setPaintTicks (true);
        fargLinjal.setPaintLabels (true);
        // sätt etiketter som märken på vallinjalen
        fargLinjal.setLabelTable (etiketter);
        fargLinjal.addChangeListener (new ChangeListener ()
            {
                public void stateChanged (ChangeEvent e)
                {
                    int    vardet = fargLinjal.getValue ();

                    // justera färgen
                    display.setForeground (
                         new Color (vardet, vardet, vardet));
                }
            } );

        // placera komponenterna i den här behållaren
        this.setTopComponent (display);
        JPanel    panel = new JPanel ();
        panel.setLayout (new GridLayout (2, 1));
        panel.add (fontLinjal);
        panel.add (fargLinjal);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar vallinjaler
class EnValLinjal
{
    public static void main (String[] args)
    {
        // en behållare med två vallinjaler och en textarea
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster
        JFrame    frame = new JFrame (" En vallinjal");
```

```
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (412, 340);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (splitPane);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller två vallinjaler och en textarea visas. Den ena vallinjalen kan användas för att justera storlek på texten i textarean. Den andra vallinjalen kan användas för att justera textens färg.



Symbolgeneratorer

EnSymbolGenerator.java

Ett program som illustrerar en symbolgenerator

En symbolgenerator är en grafisk komponent som kan generera en följd av symboler. Den genererar symboler enligt en algoritm som preciserar efterföljare och föregångare för varje symbol i följden. De genererade

symbolerna kan utnyttjas för att representera olika alternativ i ett program.

En symbolgenerator har två knappar med pilar. Varje gång användaren trycker på någon av knapparna, genereras en symbol. Användaren går ett steg fram eller tillbaka i följden. Den genererade symbolen visas i symbolgeneratorns display.

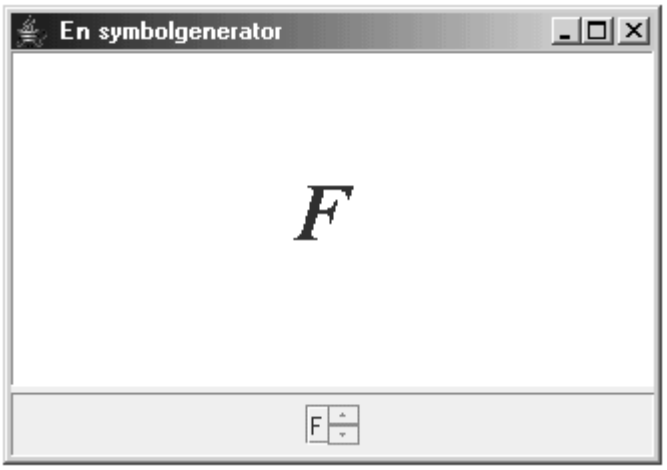
```
import java.awt.*;           // Font
import javax.swing.*;        // JSpinner, AbstractSpinnerModel,
                             // JTextArea, JSplitPane, JFrame
import javax.swing.event.*;  // ChangeEvent, ChangeListener

// en klass som definierar ett sätt att generera en följd tecken
// - en modell för symbolgenerering
class TeckenGeneratorModell extends AbstractSpinnerModel
{
    // den genererade symbolen (tecknet)
    private Character    symbol;

    // initiera den aktuella symbolen
    public TeckenGeneratorModell (Character symbol)
    {
        this.symbol = symbol;
    }

    // den aktuella symbolen
    public Object getValue ()
    {
        return symbol;
    }

    // ange den aktuella symbolen
    public void setValue (Object symbol)
    {
        // ett tecken ska anges
        if (!(symbol instanceof Character))
            throw new IllegalArgumentException ();

        this.symbol = (Character) symbol;

        // anropa metoden stateChanged i samband med alla lyssnare
        // av typen ChangeListener
        this.fireStateChanged ();
    }

    // nästa symbol i följden (relativt den aktuella symbolen)
    public Object getNextValue ()
    {
        Character    next = symbol;
```

```
        int    n = symbol.charValue ();
        if (n < Character.MAX_VALUE)
        // följdens övre gräns är tecknet MAX_VALUE
            next = new Character ((char) (n + 1));

        return next;
    }

    // föregående symbol i följden (relativt den aktuella
    // symbolen)
    public Object getPreviousValue ()
    {
        Character    previous = symbol;

        int    n = symbol.charValue ();
        if (n > 0)
        // följdens nedre gräns är det tecken som har koden 0
            previous = new Character ((char) (n - 1));

        return previous;
    }
}

// en behållare som innehåller en symbolgenerator och en textarea
class FSplitPane extends JSplitPane
{
    // en symbolgenerator
    private JSpinner    symbolGenerator;

    // en textarea
    private JTextArea    textArea;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (230);

        // symbolgeneratorn
        TeckenGeneratorModell    modell =
            new TeckenGeneratorModell (new Character ('F'));
        symbolGenerator = new JSpinner (modell);

        // textarean
        textArea = new JTextArea (4, 6);
        Font    font =
               new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
```

```
        textArea.setFont (font);
        textArea.setText ("\n                  " +
                          symbolGenerator.getValue ());

        // en lyssnare, som reagerar på användarens val via
        // symbolgeneratorn
        ChangeListener    lyssnare = new ChangeListener ()
            {
                public void stateChanged (ChangeEvent e)
                {
                    JSpinner   source = (JSpinner) e.getSource ();

                    // den genererade symbolen
                    Object    genereradeSymbolen =
                                                source.getValue ();

                    // aktivera motsvarande alternativ i
                    // programmet
                    textArea.setText ("\n                  "
                                      + genereradeSymbolen);
                }
            };

        // registrera lyssnaren hos symbolgeneratorn
        symbolGenerator.addChangeListener (lyssnare);

        // placera komponenterna i den här behållaren
        this.setTopComponent (textArea);
        JPanel    panel = new JPanel ();
        panel.add (symbolGenerator);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar en symbolgenerator
class EnSymbolGenerator
{
    public static void main (String[] args)
    {
        // en behållare med en symbolgenerator och en textarea
        FSplitPane    splitPane = new FSplitPane ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" En symbolgenerator");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);
        frame.add (splitPane);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en symbolgenerator och en textarea visas. Den symbol som genereras via symbolgeneratorn visas i textarean.



SymbolGeneratorerMedStandardModeller.java

Ett program som illustrerar symbolgeneratorer med standardmodeller

En symbolgenerator använder en modell, som definierar den följd av symboler som kan genereras. Det finns flera standardklasser som representerar olika modeller för symbolgenerering. Olika symbolgeneratorer kan skapas utifrån dessa standardmodeller.

```
import java.awt.*;           // Font
import javax.swing.*;        // SpinnerNumberModel,
          // SpinnerListModel, JSpinner, JTextArea,
          // JSplitPane, JFrame
import java.awt.event.*;     // ActionEvent, ActionListener
import java.util.*;          // GregorianCalendar, Calendar
import static java.util.Calendar.*;  // olika konstanter

// en behållare som innehåller tre symbolgeneratorer,
// en knapp och en textarea
class FSplitPane extends JSplitPane
{
    // tre symbolgeneratorer
```

```
private JSpinner    dagGenerator;
private JSpinner    manadGenerator;
private JSpinner    arGenerator;

// en knapp
private JButton    knapp;

// en textarea
private JTextArea    display;

// initiera den här behållaren
public FSplitPane ()
{
    // justera inställningar i den här behållaren
    super (JSplitPane.VERTICAL_SPLIT);
    this.setContinuousLayout (true);
    this.setDividerSize (5);
    this.setDividerLocation (230);

    // en standardmodell för symbolgenerering
    SpinnerNumberModel    dagModell =
            new SpinnerNumberModel (11, 1, 31, 1);
    // definierar talföljden fr o m 1 t o m 31, med steget 1,
    // och med startvärdet 11

    // den första symbolgeneratorn
    dagGenerator = new JSpinner (dagModell);
    // ett värde kan väljas antingen via knappar (upp och ner)
    // eller genom att användaren skriver in ett värde i
    // symbolgeneratorns textfält (som är både en display
    // och ett inmatningsfält)

    // en standardmodell för symbolgenerering
    final String[]   manader = {"Januari", "Februari", "Mars",
              "April", "Maj", "Juni", "Juli", "Augusti",
              "September", "Oktober", "November", "December"};
    SpinnerListModel   manadModell =
                               new SpinnerListModel (manader);
    // definierar en följd av strängar som representerar
    // namn på olika månader

    // andra symbolgeneratorn
    manadGenerator = new JSpinner (manadModell);

    // en standardmodell för symbolgenerering
    SpinnerNumberModel    arModell =
            new SpinnerNumberModel (1987, 1800, 2500, 1);
    // definierar talföljden fr o m 1800 t o m 2500,
    // med steget 1 och med startvärdet 1987

    // tredje symbolgeneratorn
```

```
arGenerator = new JSpinner (arModell);

// textarean
display = new JTextArea (4, 6);
Font  f = new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
display.setFont (f);
display.setText ("\n     söndag");

// knappen
knapp = new JButton ("bekräfta valet");

// en lyssnare som reagerar när användaren trycker
// på knappen
ActionListener    lyssnare = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // vald dag, månad och år
            Integer d =(Integer) dagGenerator.getValue ();
            String m =(String) manadGenerator.getValue ();
            Integer a = (Integer) arGenerator.getValue ();

            // motsvarande datum
            int    dag = d; int    manad = 0;
            while (!m.equals (manader[manad]))
                manad++;
            int    ar = a;
            GregorianCalendar    gc =
                   new GregorianCalendar (ar, manad, dag);

            // veckodag för det valda datumet
            int    vD = gc.get (DAY_OF_WEEK);
            String    veckoDag = "";
            switch (vD)
            {
            case SUNDAY:
                veckoDag = "Söndag";
                break;
            case MONDAY:
                veckoDag = "Måndag";
                break;
            case TUESDAY:
                veckoDag = "Tisdag";
                break;
            case WEDNESDAY:
                veckoDag = "Onsdag";
                break;
            case THURSDAY:
                veckoDag = "Torsdag";
                break;
            case FRIDAY:
```

```
                    veckoDag = "Fredag";
                    break;
                case SATURDAY:
                    veckoDag = "Lördag";
                }

                // visa veckodagen för det valda datumet
                display.setText ("\n          " + veckoDag);
            }
        };
        knapp.addActionListener (lyssnare);

        // placera komponenterna i den här behållaren
        this.setTopComponent (display);
        JPanel    panel = new JPanel ();
        panel.add (new JLabel ("Ett datum: "));
        panel.add (dagGenerator);
        panel.add (manadGenerator);
        panel.add (arGenerator);
        panel.add (knapp);
        this.setBottomComponent (panel);
    }
}

// ett program som illustrerar symbolgeneratorer
// med standardmodeller
class SymbolGeneratorerMedStandardModeller
{
    public static void main (String[] args)
    {
        // ett fönster som innehåller en behållare
        JFrame    frame = new JFrame (
                      " Symbolgeneratorer med standardmodeller");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (new FSplitPane ());
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller tre symbolgeneratorer, en knapp och en text-area visas. Användaren väljer ett datum via symbolgeneratorerna och trycker på knappen. Veckodagen för datumet visas då i textarean.

Strukturerade val

Ett verktygsfält

EttVerktygsfalt.java

Ett program som illustrerar ett verktygsfält

Flera valkomponenter kan placeras i ett fält. På så sätt skapas ett fält med olika verktyg - ett verktygsfält.

```
import java.awt.*;           // Font
import javax.swing.*;        // JToolBar, JTextArea, JComboBox,
                             // JToggleButton, JPanel, JFrame
import java.awt.event.*;     // ActionEvent, ActionListener

// en behållare som innehåller en textarea och ett verktygsfält
class Display extends JPanel
{
    // texten som ska visas
    private String    text;

    // en textarea - plats att visa texten
    private JTextArea    displayArea;

    // valrutor
    private JComboBox    fontNameChoice;
    private JComboBox    fontSizeChoice;

    // knappar med två stabila lägen
    private JToggleButton    boldChoice;
    private JToggleButton    italicChoice;

    // ett fält för olika verktyg (ett verktygsfält)
    private JToolBar    tools;

    // initiera den här behållaren
    public Display (String text)
    {
        // texten
        this.text = text;

        // textarean
        displayArea = new JTextArea (4, 6);
        Font    font = new Font ("Serif", Font.PLAIN, 40);
        displayArea.setFont (font);
        displayArea.setText ("\n " + text);
```

```
// valrutan för val av fontens logiska namn
String[]  fontNames = {"Serif", "SansSerif", "Monospaced",
                       "Dialog", "DialogInput"};
fontNameChoice = new JComboBox (fontNames);
fontNameChoice.setSelectedIndex (1);
fontNameChoice.setToolTipText ("Fontens namn");

// reagera på användarens val av fontens namn
fontNameChoice.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // valda namnet för fonten
            Object    val =
                   fontNameChoice.getSelectedItem ();
            String    namn = (String) val;

            // den nya fonten
            Font    font0 = displayArea.getFont ();
            Font    font = new Font (
               namn, font0.getStyle (), font0.getSize ());

            // ange fonten i displayen
            displayArea.setFont (font);
        }
    } );


// valrutan för val av fontens storlek
String[]    fontSizes = {"15", "20", "25", "30",
                         "35", "40", "45", "50"};
fontSizeChoice = new JComboBox (fontSizes);
fontSizeChoice.setSelectedIndex (5);
fontSizeChoice.setToolTipText ("Fontens storlek");

// reagera på användarens val av fontens storlek
fontSizeChoice.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // den valda storleken på fonten
            Object    val =
                       fontSizeChoice.getSelectedItem ();
            int    size = Integer.parseInt ((String) val);

            // den nya fonten
            Font    font0 = displayArea.getFont ();
            Font    font = new Font (
               font0.getName (), font0.getStyle (), size);

            // ange fonten i displayen
```

```
            displayArea.setFont (font);
        }
    } );

// knappen för fet stil
boldChoice = new JToggleButton (" F ");
Font    f0 = boldChoice.getFont ();
Font    f =
      new Font (f0.getName (), Font.BOLD, f0.getSize ());
boldChoice.setFont (f);
boldChoice.setToolTipText ("Fet stil");

// fonten för kursiv stil
italicChoice = new JToggleButton (" K ");
f0 = italicChoice.getFont ();
f = new Font (f0.getName (), Font.ITALIC, f0.getSize ());
italicChoice.setFont (f);
italicChoice.setToolTipText ("Kursiv stil");

// reagera på användarens val av fontens stil
ActionListener    lyssnare = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // valda stilen för fonten
            boolean    bold = boldChoice.isSelected ();
            boolean   italic = italicChoice.isSelected ();

            // den nya fonten
            Font    font0 = displayArea.getFont ();
            Font    font = null;
            if (bold  &&  italic)
                font = new Font (font0.getName (),
                                 Font.BOLD + Font.ITALIC,
                                 font0.getSize ());
            else if (bold)
                font = new Font (font0.getName (),
                                 Font.BOLD, font0.getSize ());
            else if (italic)
                font = new Font (font0.getName (),
                                 Font.ITALIC, font0.getSize ());
            else
                font = new Font (font0.getName (),
                                 Font.PLAIN, font0.getSize ());

            // ange fonten i displayen
            displayArea.setFont (font);
        }
    };
boldChoice.addActionListener (lyssnare);
italicChoice.addActionListener (lyssnare);
```

```
        // en fyllnad (så att andra komponenter
        // i verktygsfältet inte blir för stora)
        JPanel    fyllnad = new JPanel ();
        fyllnad.setPreferredSize (new Dimension (250,
                             fyllnad.getPreferredSize ().height));

        // verktygsfältet
        tools = new JToolBar ();
        tools.add (fontNameChoice);
        tools.add (fontSizeChoice);
        tools.addSeparator ();  // ett mellanrum
        tools.add (boldChoice);
        tools.add (italicChoice);
        tools.add (fyllnad);
        // fyllnaden kan även åstadkommas så här:
        // tools.add (Box.createGlue ());

        // ett vertikalt verktygsfält kan skapas så här:
        // tools = new JToolBar (JToolBar.VERTICAL);

        // placera komponenterna i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (tools, "North");
        this.add (displayArea, "Center");

        // Verktygsfältet placeras på en av fyra sidopositioner.
        // Verktygsfältet kan flyttas manuellt från en
        // position till en annan position. Denna möjlighet
        // kan sättas ur spel så här:
        // tools.setFloatable (false);
    }
}

// ett program som illustrerar ett verktygsfält
class EttVerktygsfalt
{
    public static void main (String[] args)
    {
        // en behållare med en textarea och ett verktygsfält
        Display    display = new Display ("S a n n i n g e n");

        // ett fönster
        JFrame    frame = new JFrame (" Ett verktygsfält");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (560, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (display);
```

```
        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea och ett verktygsfält visas. En teckensträng visas i textarean. Via verktygsfältet kan fonten för textarean väljas. Det går att välja fontens logiska namn, storlek och stil.

Menyer

Menyer.java

Ett program som illustrerar menyer

Ett alternativ i ett program kan symboliseras via en tvålägeskomponent. Flera tvålägeskomponenter kan användas för att representera flera relaterade alternativ. Dessa komponenter kan grupperas i en meny, som kan placeras i en menyrad. Flera menyer kan placeras i en och samma menyrad. Via dessa menyer kan olika val i ett program utföras.

```
import java.awt.*;          // Font, Color
import javax.swing.*;       // JMenuBar, JMenu, JMenuItem,
                            // JCheckBoxMenuItem,
                            // JRadioButtonMenuItem,
                            // ButtonGroup, JTextArea,
```

```
                            // JPanel, JFrame
import java.awt.event.*;    // ActionEvent, ActionListener

// en behållare som innehåller en textarea och en menyrad
class Display extends JPanel
{
    // texten som ska visas
    private String    text;

    // en textarea - plats att visa texten
    private JTextArea    displayArea;

    // olika menyer
    private JMenu    fontNameMenu;
    private JMenu    fontSizeMenu;
    private JMenu    fontStyleMenu;
    private JMenu    colorMenu;
    private JMenu    backgroundMenu;

    // en menyrad
    private JMenuBar    menus;

    // initiera den här behållaren
    public Display (String text)
    {
        // texten
        this.text = text;

        // textarean
        displayArea = new JTextArea (4, 6);
        Font    font = new Font ("Serif", Font.PLAIN, 40);
        displayArea.setFont (font);
        displayArea.setText ("\n " + text);

        // meny för val av fontens logiska namn
        String[]    fontNames = { "Serif", "SansSerif",
                                  "Monospaced",
                                  "Dialog", "DialogInput" };
        JMenuItem[]    fontNameItems =
                            new JMenuItem[fontNames.length];
        fontNameMenu = new JMenu ("Font");
        for (int i = 0; i < fontNameItems.length; i++)
        {
            fontNameItems[i] = new JMenuItem (fontNames[i]);
            fontNameMenu.add (fontNameItems[i]);

            if (i == 1  ||  i == 2)
                fontNameMenu.addSeparator ();
        }

        // reagera på användarens val av fontens namn
```

```
ActionListener    fontNameListener = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // det valda namnet för fonten
            JMenuItem    val = (JMenuItem) e.getSource ();
            String    namn = val.getText();

            // den nya fonten
            Font font0 = displayArea.getFont ();
            Font font = new Font (namn, font0.getStyle (),
                                  font0.getSize ());

            // ange fonten i displayen
            displayArea.setFont (font);
        }
    };
for (int i = 0; i < fontNameItems.length; i++)
    fontNameItems[i].addActionListener (fontNameListener);

// meny för val av fontens storlek
String[]    fontSizes = {"25", "30", "35", "40"};
JMenuItem[]    fontSizeItems =
                 new JMenuItem[fontSizes.length];
fontSizeMenu = new JMenu ("Storlek");
for (int i = 0; i < fontSizeItems.length; i++)
{
    fontSizeItems[i] = new JMenuItem (fontSizes[i]);
    fontSizeMenu.add (fontSizeItems[i]);
}

// reagera på användarens val av fontens storlek
ActionListener    fontSizeListener = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // den valda storleken för fonten
            JMenuItem    val = (JMenuItem) e.getSource ();
            int   size = Integer.parseInt (val.getText());

            // den nya fonten
            Font    font0 = displayArea.getFont ();
            Font    font = new Font (font0.getName (),
                                font0.getStyle (), size);

            // ange fonten i displayen
            displayArea.setFont (font);
        }
    };
for (int i = 0; i < fontSizeItems.length; i++)
    fontSizeItems[i].addActionListener (fontSizeListener);
```

```
// meny för val av fontens stil
final JCheckBoxMenuItem     boldStyle =
                            new JCheckBoxMenuItem ("Fet");
final JCheckBoxMenuItem     italicStyle =
                          new JCheckBoxMenuItem ("Kursiv");
fontStyleMenu = new JMenu ("Stil");
fontStyleMenu.add (boldStyle);
fontStyleMenu.add (italicStyle);

// reagera på användarens val av fontens stil
ActionListener   fontStyleListener = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // den valda stilen för fonten
            boolean    bold = boldStyle.isSelected ();
            boolean    italic = italicStyle.isSelected ();

            // den nya fonten
            Font    font0 = displayArea.getFont ();
            Font    font = null;
            if (bold  &&  italic)
                font = new Font (font0.getName (),
                           Font.BOLD + Font.ITALIC,
                                  font0.getSize ());
            else if (bold)
                font = new Font (font0.getName (),
                                 Font.BOLD, font0.getSize ());
            else if (italic)
                font = new Font (font0.getName (),
                               Font.ITALIC, font0.getSize ());
            else
                font = new Font (font0.getName (),
                                Font.PLAIN, font0.getSize ());

            // ange fonten i displayen
            displayArea.setFont (font);
        }
    };
boldStyle.addActionListener (fontStyleListener);
italicStyle.addActionListener (fontStyleListener);

// meny för val av textens färg
final JRadioButtonMenuItem    blackColor =
                        new JRadioButtonMenuItem ("Svart");
final JRadioButtonMenuItem    blueColor =
                          new JRadioButtonMenuItem ("Blå");
final JRadioButtonMenuItem    redColor =
                          new JRadioButtonMenuItem ("Röd");
ButtonGroup    colorGroup = new ButtonGroup ();
```

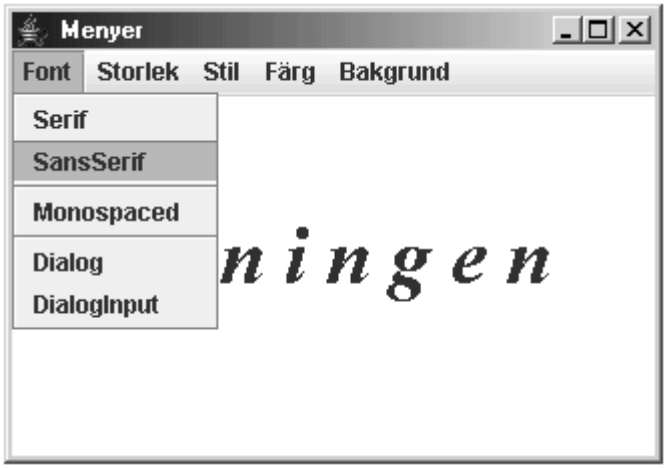
```
colorGroup.add (blackColor);
colorGroup.add (blueColor);
colorGroup.add (redColor);
colorMenu = new JMenu ("Färg");
colorMenu.add (blackColor);
colorMenu.add (blueColor);
colorMenu.add (redColor);

// reagera på användarens val av textens färg
ActionListener   colorListener = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // den valda färgen
            Object    source = e.getSource ();
            Color    color = null;
            if (source == blackColor)
                color = Color.BLACK;
            else if (source == blueColor)
                color = Color.BLUE;
            else if (source == redColor)
                color = Color.RED;

            // ange textens färg
            displayArea.setForeground (color);
        }
    };
blackColor.addActionListener (colorListener);
blueColor.addActionListener (colorListener);
redColor.addActionListener (colorListener);

// meny för val av bakgrundsfärgen

final JMenuItem   whiteBackground = new JMenuItem ("Vit");
final JMenuItem   lightgrayBackground =
                                  new JMenuItem ("Ljusgrå");
final JMenuItem   grayBackground =
                                   new JMenuItem ("Grå");
final JMenuItem   darkgrayBackground =
                                  new JMenuItem ("Mörkgrå");
// en submeny
JMenu    grayMenu = new JMenu ("Grå");
grayMenu.add (lightgrayBackground);
grayMenu.add (grayBackground);
grayMenu.add (darkgrayBackground);

backgroundMenu = new JMenu ("Bakgrund");
backgroundMenu.add (whiteBackground);
backgroundMenu.add (grayMenu);

// reagera på användarens val av bakgrundsfärgen
```

```
        ActionListener  backgroundListener = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    // den valda färgen
                    Object    source = e.getSource ();
                    Color    farg = null;
                    if (source == whiteBackground)
                        farg = Color.WHITE;
                    else if (source == lightgrayBackground)
                        farg = Color.LIGHT_GRAY;
                    else if (source == grayBackground)
                        farg = Color.GRAY;
                    else if (source == darkgrayBackground)
                        farg = Color.DARK_GRAY;

                    // ange bakgrundsfärgen i displayen
                    displayArea.setBackground (farg);
                }
            };
        whiteBackground.addActionListener (backgroundListener);
        lightgrayBackground.addActionListener (backgroundListener);
        grayBackground.addActionListener (backgroundListener);
        darkgrayBackground.addActionListener (backgroundListener);

        // menyraden
        menus = new JMenuBar ();
        menus.add (fontNameMenu);
        menus.add (fontSizeMenu);
        menus.add (fontStyleMenu);
        menus.add (colorMenu);
        menus.add (backgroundMenu);

        // placera komponenterna i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (menus, "North");
        this.add (displayArea, "Center");
    }
}

// ett program som illustrerar menyer
class Menyer
{
    public static void main (String[] args)
    {
        // en behållare med en textarea och en menyrad
        Display    display = new Display ("S a n n i n g e n");

        // ett fönster
        JFrame    frame = new JFrame (" Menyer");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
```

```
        frame.setSize (460, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (display);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea och en menyrad visas. I textarean visas en text. Menyraden innehåller flera menyer. Via dessa menyer kan användaren justera textareans font (typ, storlek och stil), bakgrundsfärg och förgrundsfärg.



BlockeraAktiveraEnMenysKomponenter.java

Ett program som illustrerar hur en menys komponenter blockeras och aktiveras

Under vissa omständigheter kan ett eller flera alternativ i ett program bli inaktuella. I så fall kan man blockera de komponenter i motsvarande meny som används för att välja dessa alternativ. Om ett alternativ på nytt blir aktuellt, måste motsvarande komponent i menyn aktiveras igen.

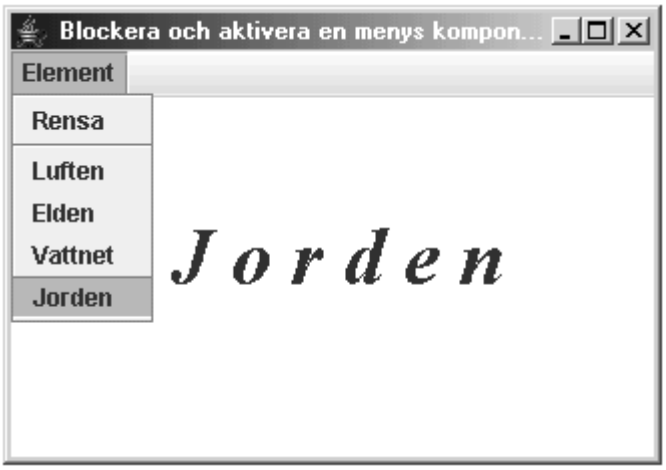
```
import java.awt.*;          // Font
import javax.swing.*;       // JMenuBar, JMenu, JMenuItem,
                            // JTextArea, JPanel, JFrame
import java.awt.event.*;    // ActionEvent, ActionListener

// en behållare som innehåller en textarea och en meny
class Display extends JPanel
{
    // en textarea
    private JTextArea    elementDisplay;

    // en meny - för att kunna välja det element som ska visas
    // i textarean
    private JMenu    elementMeny;

    // initiera den här behållaren
    public Display ()
    {
```

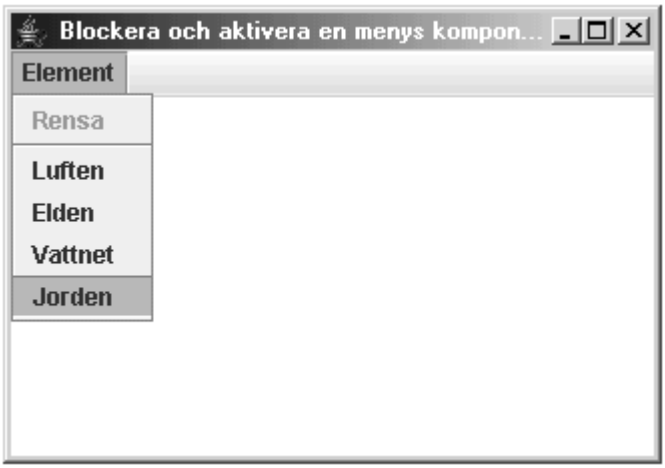
```
// element som ska visas i textarean
String[]    element =
              {"Luften", "Elden", "Vattnet", "Jorden"};

// textarean
elementDisplay = new JTextArea (4, 6);
Font    font =
       new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
elementDisplay.setFont (font);
elementDisplay.setText ("\n        l u f t e n");


// utforma elementmenyn

// komponenter i elementmenyn
final JMenuItem    rensa = new JMenuItem ("Rensa");
JMenuItem[]    elementValjare = new JMenuItem[4];
for (int i = 0; i < elementValjare.length; i++)
    elementValjare[i] = new JMenuItem (element[i]);

// elementmenyn
elementMeny = new JMenu ("Element");
elementMeny.add (rensa);
elementMeny.addSeparator ();
for (int i = 0; i < elementValjare.length; i++)
    elementMeny.add (elementValjare[i]);

// reagera på användarens val via elementmenyn
ActionListener    lyssnare = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // om en text ska visas, aktivera den
            // komponent i menyn som gör det möjligt
            // att texten sedan kan rensas
            if (!rensa.isEnabled ())
                rensa.setEnabled (true);

            // texten på motsvarande menykomponent
            JMenuItem source = (JMenuItem) e.getSource ();
            String    text = "";
            if (source != rensa)
                text = source.getText ();

            // texten som ska visas
            String    displayText = "";
            for (int i = 0; i < text.length (); i++)
                displayText += text.charAt (i) + " ";

            // visa texten
            elementDisplay.setText ("\n  " + displayText);
```

```
                    // om displayen är tom, blockera
                    // motsvarande komponent i menyn
                    // (en tom display ska inte rensas)
                    if (text.equals (""))
                        rensa.setEnabled (false);
                }
            };
        rensa.addActionListener (lyssnare);
        for (int i = 0; i < elementValjare.length; i++)
            elementValjare[i].addActionListener (lyssnare);

        // menyraden
        JMenuBar    menyer = new JMenuBar ();
        menyer.add (elementMeny);

        // placera komponenterna i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (elementDisplay, "Center");
        this.add (menyer, "North");
    }
}

// ett program som illustrerar hur komponenterna
// i en meny blockeras och aktiveras
class BlockeraAktiveraEnMenysKomponenter
{
    public static void main (String[] args)
    {
        // en behållare med en textarea och en menyrad
        Display    display = new Display ();

        // ett fönster
        JFrame    frame = new JFrame (
               " Blockera och aktivera en menys komponenter");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (display, "Center");

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea och en meny visas. Via menyn väljs den text som ska visas i textarean. Det går också att välja att rensa en befintlig text.





KortaKommandonMedMenyer.java

Ett program som illustrerar användningen av kortkommandon i samband med menyer

I samband med menyer kan olika kortkommandon användas. En tangent på tangentbordet (en så kallad mnemonic) kan bindas till en komponent i en meny. Denna komponent kan då aktiveras via tangentbordet, om menyn redan är öppen. Den valda tangenten används i kombination med `Alt`- tangenten.

En tangent kan även bindas till en meny. Denna meny kan i så fall öppnas genom att tangenten används i kombination med `Alt`-tangenten.

En tangent på tangentbordet (en så kallad accelerator) kan bindas till en komponent i en meny. Denna komponent kan i så fall aktiveras via tangentbordet även om menyn inte är öppen. Endast den valda tangenten används. Det går även att kombinera en tangent med en eller flera kontrolltangenter (`Alt`, `Ctrl`, `Shift`, `Meta`), och använda kombiniationen för att direkt aktivera en komponent i menyn.

```
import java.awt.*;          // Font, Color
import javax.swing.*;       // JMenuBar, JMenu,
                            // JMenuItem, KeyStroke,
                            // JTextArea, JPanel, JFrame
import java.awt.event.*;    // ActionEvent, ActionListener,
                            // KeyEvent, InputEvent

// en behållare som innehåller en textarea och en menyrad
class Display extends JPanel
{
    // en textarea
    private JTextArea    elementDisplay;

    // elementmeny - en meny för att välja det element
    // som ska visas i textarean
    private JMenu    elementMeny;

    // färgmeny - en meny för elementens färg
    private JMenu    fargMeny;

    // en menyrad
    private JMenuBar    menyer;

    // initiera den här behållaren
    public Display ()
    {
        // element som ska visas i textarean
```

```
final String[]    element = {"E t e r",
                             "L u f t e n",
                             "E l d e n",
                             "V a t t n e t",
                             "J o r d e n"};

// namn i elementmenyn
String[]   nummer = {"Noll", "En", "Två", "Tre", "Fyra"};

// färger som ska kunna väljas via färgmenyn
final Color[]    farger = { Color.MAGENTA,
                            Color.BLUE,
                            Color.RED,
                            new Color (230, 230, 230),
                            Color.YELLOW};

// namn i färgmenyn
String[]    fargNamn = {"Magenta", "Blue",
                        "Red", "Gray", "Yellow"};


// textarean
elementDisplay = new JTextArea (4, 6);
Font    font =
       new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
elementDisplay.setFont (font);
elementDisplay.setText ("\n  " + element[0]);
elementDisplay.setForeground (farger[0]);


// utforma elementmenyn

// komponenter i elementmenyn
final JMenuItem[]    elementValjare = new JMenuItem[5];
// den tangent som innehåller tecknet '0'
int    tangent = KeyEvent.VK_0;
for (int i = 0; i < 5; i++)
{
    elementValjare[i] = new JMenuItem (" " + i + " ");

    // ange accelerator för en komponent i menyn
    KeyStroke    accelerator = KeyStroke.getKeyStroke (
                  tangent + i,  // tangenten
                  InputEvent.CTRL_MASK); //Ctrl-tangenten
    elementValjare[i].setAccelerator (accelerator);
    // Kortkommandon: Ctrl-0, Ctrl-1, ...

    // Virtuella tangentkoder för de tangenter
    // som innehåller 0 - 9 är desammma som
    // motsvarande Unicode-koder.
```

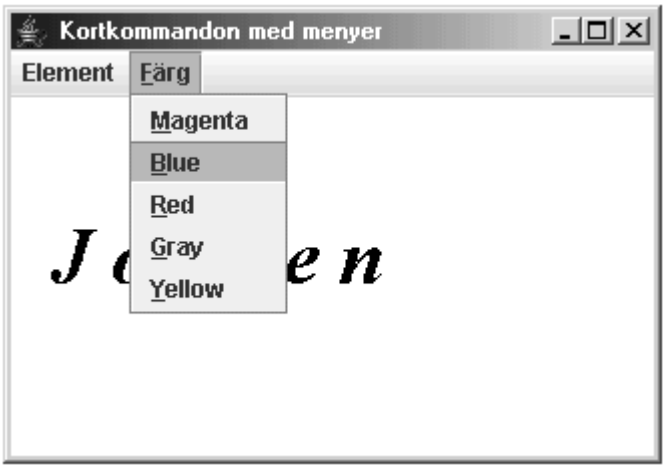
```
    // Det går att välja mellen kontrolltangenterna
    // SHIFT, CTRL, META och ALT,
    // Flera kontrolltangenter kan kombineras,
    // t ex så här:
    // InputEvent.CTRL_MASK + InputEvent.ALT_MASK
    // Om kontrolltangenterna inte ska användas,
    // anges 0 som andra argument.
}

// reagera på användarens val via elementmenyn
ActionListener    elementLyssnare = new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // identifiera källan
            Object    source = e.getSource ();
            int    val = 0;
            while (!(source == elementValjare[val]))
                val++;

            // visa motsvarande element
            elementDisplay.setText ("\n  " +
                                    element[val]);
        }
    };
for (int i = 0; i < elementValjare.length; i++)
    elementValjare[i].addActionListener (elementLyssnare);

// elementmenyn
elementMeny = new JMenu ("Element");
for (int i = 0; i < elementValjare.length; i++)
    elementMeny.add (elementValjare[i]);

// utforma färgmenyn

// komponenter i färgmenyn
final JMenuItem[]    fargValjare = new JMenuItem[5];
int    mnemonic = 0;
for (int i = 0; i < 5; i++)
{
    fargValjare[i] = new JMenuItem (fargNamn[i]);

    // ange mnemonic (mnemonic-tangenten) för en komponent
    // i menyn
    mnemonic = fargNamn[i].charAt (0);
    fargValjare[i].setMnemonic (mnemonic);

    // Kortkommandon Alt-M, Alt-B, Alt-R, Alt-G, Alt-Y när
    // färgmenyn är öppen

    // Virtuella tangentkoder för de tangenter
```

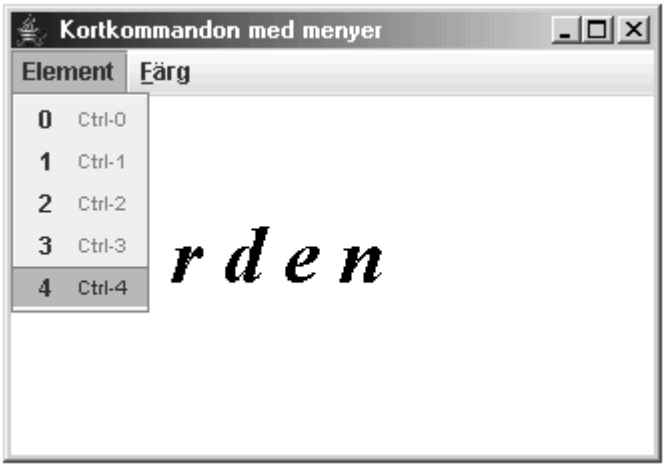
```
            // som innehåller A - Z är likadana som
            // motsvarande Unicode-koder.
        }

        // reagera på användarens val via färgmenyn
        ActionListener    fargLyssnare = new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    // identifiera källan
                    Object    source = e.getSource ();
                    int    val = 0;
                    while (!(source == fargValjare[val]))
                        val++;

                    // ange textens färg
                    elementDisplay.setForeground (farger[val]);
                }
            };
        for (int i = 0; i < fargValjare.length; i++)
            fargValjare[i].addActionListener (fargLyssnare);

        // färgmenyn
        fargMeny = new JMenu ("Färg");
        for (int i = 0; i < fargValjare.length; i++)
            fargMeny.add (fargValjare[i]);

        // ange mnemonic för färgmenyn
        fargMeny.setMnemonic (KeyEvent.VK_F);
        // menyn kan öppnas med Alt-F
        // (användaren kan till exempel välja den röda färgen
        // genom att hålla ned Alt-tangenten, sedan trycka på
        // F och därefter på R).

        // menyraden
        menyer = new JMenuBar ();
        menyer.add (elementMeny);
        menyer.add (fargMeny);

        // placera komponenterna i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (elementDisplay, "Center");
        this.add (menyer, "North");
    }
}

// ett program som illustrerar kortkommandon i samband med menyer
class KortkommandonMedMenyer
{
    public static void main (String[] args)
    {
```

```
        // en behållare med en textarea och en menyrad
        Display    display = new Display ();

        // ett fönster
        JFrame   frame = new JFrame (" Kortkommandon med menyer");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 300);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (display, "Center");

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea och en menyrad visas. I textarean visas en text. Menyraden innehåller två menyer. Via dessa menyer väljer användaren textareans text och förgrundsfärg. Olika kortkommandon kan användas i samband med menyerna.





PopupMenyer.java

Ett program som illustrerar popup-menyer

En meny kan bindas till en komponent i ett fönster. Denna meny visas över komponenten med hjälp av musen (motsvarande knapp måste användas). När menyn blir synlig, kan användaren välja ett alternativ via den. Efter valet blir menyn osynlig på nytt. En sådan meny kallas för popup-meny.

```
import java.awt.*;          // Font, Color, Component
import javax.swing.*;       // JPopupMenu, JMenuItem,
                            // JTextArea, JPanel, JFrame
import java.awt.event.*;    // ActionEvent, ActionListener,
                            // MouseEvent, MouseListener,
                            // MouseAdapter

// en behållare som innehåller en textarea
// som en popup-meny är knuten till
class Display extends JPanel
{
    // en textarea
    private JTextArea    displayArea;

    // en popup-meny
    private JPopupMenu    popupMeny;

    // initiera den här behållaren
    public Display ()
```

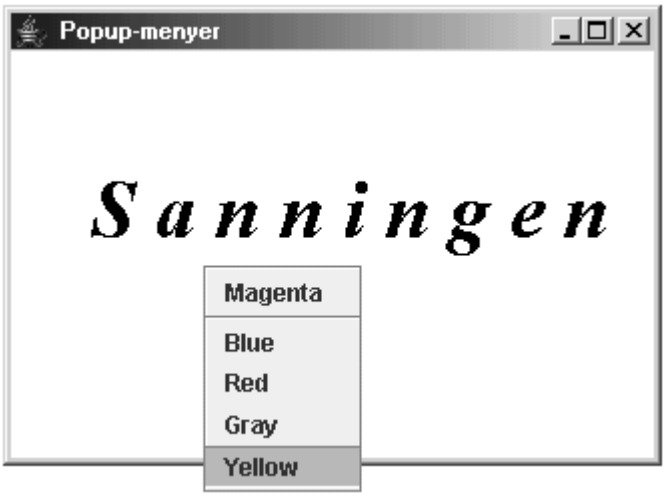
```
{
    // färger som ska kunna väljas via popup-menyn
    final Color[]    farger = { Color.MAGENTA,
                                Color.BLUE,
                                Color.RED,
                                new Color (230, 230, 230),
                                Color.YELLOW};

    // textarean
    displayArea = new JTextArea (4, 6);
    Font    font =
         new Font ("Serif", Font.BOLD + Font.ITALIC, 40);
    displayArea.setFont (font);
    displayArea.setForeground (farger[0]);
    displayArea.setText ("\n    S a n n i n g e n");

    // komponenter i popup-menyn
    String[]    fargNamn = {"Magenta", "Blue",
                            "Red", "Gray", "Yellow"};
    final JMenuItem[]    fargValjare = new JMenuItem[5];
    for (int i = 0; i < 5; i++)
        fargValjare[i] = new JMenuItem (fargNamn[i]);

    // reagera på användarens val via popup-menyn
    ActionListener    fargLyssnare = new ActionListener ()
        {
            public void actionPerformed (ActionEvent e)
            {
                // identifiera källan
                Object    source = e.getSource ();
                int    val = 0;
                while (!(source == fargValjare[val]))
                    val++;

                // ange textens färg
                displayArea.setForeground (farger[val]);
            }
        };
    for (int i = 0; i < fargValjare.length; i++)
        fargValjare[i].addActionListener (fargLyssnare);


    // popup-menyn
    popupMeny = new JPopupMenu ();
    for (int i = 0; i < fargValjare.length; i++)
    {
        popupMeny.add (fargValjare[i]);
        if (i == 0)
            popupMeny.addSeparator ();
    }
```

```
        // binda popup-menyn till textarean
        MouseListener    popupLyssnare = new MouseAdapter ()
            {
                // på vissa plattformar visas popup-menyer
                // när användaren trycker ner motsvarande musknapp
                public void mousePressed (MouseEvent e)
                {
                    // komponenten där användaren
                    // tryckt musknappen, och
                    // koordinaterna för motsvarande punkt
                    Component    komponent = e.getComponent ();
                    int    x = e.getX ();
                    int    y = e.getY ();

                    // om händelsen är en popupmeny-aktivering
                    if (e.isPopupTrigger ())
                        // visa popup-menyn på den plats där
                        // användaren tryckt ner musknappen
                        popupMeny.show (komponent, x, y);
                }

                // på visa plattformar visas popupmenyer
                // när användaren släpper motsvarande musknapp
                public void mouseReleased (MouseEvent e)
                {
                    // komponenten där användaren släppt
                    // musknappen, och koordinaterna
                    // för motsvarande punkt
                    Component    komponent = e.getComponent ();
                    int    x = e.getX ();
                    int    y = e.getY ();

                    // om händelsen är en popupmeny-aktivering
                    if (e.isPopupTrigger ())
                        // visa popup-menyn på den plats
                        // där användaren släppt musknappen
                        popupMeny.show (komponent, x, y);
                }
            };
        // textarean ska lyssna på mushändelser
        displayArea.addMouseListener (popupLyssnare);

        // placera textarean i den här behållaren
        this.setLayout (new BorderLayout ());
        this.add (displayArea, "Center");
    }
}

// ett program som illustrerar popup-menyer
class PopupMenyer
```

```
{
    public static void main (String[] args)
    {
        // en behållare med en textarea, till vilken är knuten
        // en popup-meny
        Display    display = new Display ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" Popup-menyer");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 300, 120, 120);
        frame.add (display);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea visas. En popup-meny kan aktiveras (med musen) på textarean. Via denna popup-meny kan färgen på texten i textarean väljas.



Dialoger

Använda dialoger

AnvandaEnDialog.java

Ett program som illustrerar hur dialoger används

En dialog kan aktiveras via en meny, eller via någon annan komponent i ett programs grafiska användargränssnitt. Via dialogen kan användaren få information av olika slag och/eller mata in olika uppgifter.

```
import java.awt.*;          // Font
import javax.swing.*;       // JDialog, JMenuBar,
                            // JMenu, JMenuItem,
                            // JTextArea, JScrollPane,
                            // JPanel, JFrame
import java.awt.event.*;    // ActionEvent, ActionListener

// en behållare som innehåller en textarea och en meny
class Display extends JPanel
{
    // ett ordspråk
    private String    ordsprak;

    // en textarea - plats att visa ordspråket
    private JTextArea    displayArea;

    // en meny - att välja ordspråket
    private JMenu    ordsprakMeny;

    // en dialog - att mata in ordspråket
    JDialog    ordsprakDialog = null;

    // en klass som definierar en typ av dialoger
    private class OrdsprakDialog extends JDialog
    {
        // en etikett
        private JLabel    etikett;

        // ett textfält
        private JTextField    textFalt;

        // initiera dialogen
        public OrdsprakDialog ()
        {
            // bestäm dialogens rubrik, storlek och placering
```

```
        this.setTitle ("Ordspråk-dialog");
        this.setSize (300, 160);
        this.setLocation (200, 280);

        // dialogen ska vara en modal dialog
        this.setModal (true);

        // etiketten
        etikett = new JLabel ("Ett ordspråk:");

        // textfältet
        textFalt = new JTextField (20);
        textFalt.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent e)
                {
                    // hämta ordspråket från textfältet
                    ordsprak = textFalt.getText ();
                    textFalt.setText ("");

                    // visa ordspråket i textarean
                    displayArea.setText ("\n  " + ordsprak);

                    // dölj dialogen
                    // (samma dialog kan även användas senare)
                    setVisible (false);
                    // eller:
                    // OrdsprakDialog.this.setVisible (false);
                }
            } );

        // placera etiketten och textfältet i den här dialogen
        JPanel    panel1 = new JPanel ();
        panel1.add (etikett);
        JPanel    panel2 = new JPanel ();
        panel2.add (textFalt);
        this.setLayout (new GridLayout (2, 1));
        this.add (panel1);
        this.add (panel2);
    }
}

// initiera den här behållaren
public Display ()
{
    // ordspråket
    this.ordsprak = "Morgonstund har guld i mun!";

    // textarean
    displayArea = new JTextArea (4, 6);
    Font   font =
```

```
   new Font ("Serif", Font.BOLD + Font.ITALIC, 30);
displayArea.setFont (font);
displayArea.setText ("\n  " + ordsprak);

// menyn
JMenuItem    item1 = new JMenuItem ("Förvalt");
JMenuItem    item2 = new JMenuItem ("Välj");
ordsprakMeny = new JMenu ("Ordspråk");
ordsprakMeny.add (item1);
ordsprakMeny.add (item2);

// hantera användarens val

// reagera på användarens val via
// den första komponenten i menyn
item1.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // ett förvalt ordspråk
            ordsprak = "Morgonstund har guld i mun!";
            displayArea.setText ("\n  " + ordsprak);
        }
    } );

// reagera på användarens val via
// den andra komponenten i menyn
item2.addActionListener (new ActionListener ()
    {
        public void actionPerformed (ActionEvent e)
        {
            // om dialogen inte ännu är skapad, skapa den
            if (ordsprakDialog == null)
                ordsprakDialog = new OrdsprakDialog ();
                // den här utförs bara när användaren
                // första gången väljer att mata in
                // ordspråket via dialogen –
                // längre fram används samma dialog

            // visa dialogen
            ordsprakDialog.setVisible (true);
        }
    } );

// menyraden
JMenuBar    menyrad = new JMenuBar ();
menyrad.add (ordsprakMeny);

// placera komponenterna i den här behållaren
this.setLayout (new BorderLayout ());
this.add (menyrad, "North");
```

```
        this.add (new JScrollPane (displayArea), "Center");
    }
}

// ett program som illustrerar hur dialoger används
class AnvandaEnDialog
{
    public static void main (String[] args)
    {
        // ett fönster
        JFrame    frame = new JFrame (" Använd en dialog");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (460, 300);
        frame.setLocation (120, 120);

        // en behållare med en textarea och en meny
        Display    display = new Display ();

        // placera behållaren i fönstret
        frame.add (display);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en textarea och en meny visas. I textarean visas ett ordspråk. Via menyn kan ett ordspråk väljas. Det går också att aktivera en dialog via menyn. Via dialogen kan ett ordspråk matas in.

Datahantering via tabeller

Presentera data via en tabell

PresenteraDataViaEnTabell.java

Ett program som visar hur olika data presenteras via en tabell

En mängd data kan presenteras via en tabell. Först skapas en modell som definierar en tabells data. Utifrån denna modell skapas sedan motsvarande tabell.

```
import javax.swing.*;        // JTable, JTextField, JScrollPane,
                             // JFrame
import javax.swing.table.*;  // AbstractTableModel
import java.awt.event.*;     // ActionListener

// en modell för en tabell
class EnkelModell extends AbstractTableModel
{
    // antalet rader och kolumner i tabellen
    private int       antalRader;
    private int       antalKolumner;

    // namn på tabellens kolumner
    private String[]    kolumnNamn;

    // tabellens data
    private Object    data = " frid";

    // initiera antalet rader och kolumner i tabellen, och
    // kolumnernas namn
    public EnkelModell (int antalRader, int antalKolumner,
                        String[] kolumnNamn)
    {
        this.antalRader = antalRader;
        this.antalKolumner = antalKolumner;
        this.kolumnNamn = kolumnNamn;
    }


    // tabellens data
    // (tabellen behöver fyllas med data på något sätt:
    // data kan beräknas, hämtas från en vektor, läsas in från
    // en fil, matas in via användargränssnittet, ...)
    public void setData (Object data)
```

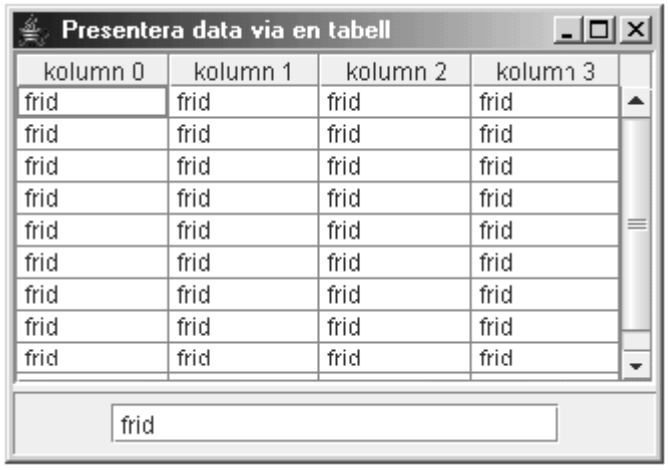
```
    {
        this.data = data;

        // uppdatera tabellen efter att underliggande data ändrats
        this.fireTableDataChanged ();
    }

    // antalet rader i tabellen
    public int getRowCount ()
    {
        return antalRader;
    }

    // antalet kolumner i tabellen
    public int getColumnCount ()
    {
        return antalKolumner;
    }

    // data som ska visas i en bestämd cell i tabellen
    // (den här metoden anropas automatiskt i samband med
    // varje cell när tabellen ritas eller ritas om – härifrån
    // får tabellen data som ska visas i olika celler)
    public Object getValueAt (int rad, int kolumn)
    {
        Object    element = data;
        // olika data kan finnas i olika celler

        return element;
    }

    // namn på en kolumn i tabellen
    public String getColumnName (int kolumn)
    {
        return kolumnNamn[kolumn];
    }
}

// en behållare som innehåller en tabell och ett textfält
class FSplitPane extends JSplitPane
{
    // en tabell
    private JTable    tabell;

    // ett textfält
    private JTextField    textFalt;


    // initiera den här behållaren
    public FSplitPane ()
    {
```

```
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (200);

        // textfältet
        textFalt = new JTextField (20);
        textFalt.setText (" frid");

        // en modell för en tabell
        String[]    kolumnNamn = {"kolumn 0", "kolumn 1",
                                  "kolumn 2","kolumn 3"};
        final EnkelModell    modell =
                            new EnkelModell (10, 4, kolumnNamn);

        // tabellen
        tabell = new JTable (modell);

        // en lyssnare som reagerar på användarens inmatning via
        // textfältet
        textFalt.addActionListener (new ActionListener ()
           {
               public void actionPerformed (ActionEvent e)
               {
                   // den inmatade strängen
                   String    text = textFalt.getText ();

                   // ändra tabellens data (genom att ändra
                   // motsvarande modell)
                   modell.setData (text);
               }
           } );

        // placera komponenterna i den här behållaren
        this.setTopComponent (new JScrollPane (tabell));
        JPanel    panel = new JPanel ();
        panel.add (textFalt);
        this.setBottomComponent (panel);
    }
}

// ett program som visar hur olika data presenteras via en tabell
class PresenteraDataViaEnTabell
{
    public static void main (String[] args)
    {
        // en behållare med en tabell och ett textfält
        FSplitPane    behallare = new FSplitPane ();

        // ett fönster
```

```
        JFrame    frame =
                new JFrame (" Presentera data via en tabell");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (400, 280);
        frame.setLocation (120, 120);

        // placera behållaren i fönstret
        frame.add (behallare);

        // visa fönstret
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en tabell och ett textfält visas. Det som användaren matar in via textfältet, visas i alla celler i tabellen.



Redigera data via en tabell

RedigeraDataViaEnTabell.java

Ett program som visar hur olika data kan redigeras via en tabell

Olika data kan redigeras via en tabell. Nya uppgifter kan matas in, och redan befintliga uppgifter kan modifieras.

```
import javax.swing.*;        // JTable, JTextArea, JScrollPane,
                             // JFrame

// en modell för en tabell
class TabellModell extends javax.swing.table.AbstractTableModel
{
    // antalet rader och kolumner i tabellen
    private int       antalRader;
    private int       antalKolumner;

    // namn på tabellens kolumner
    private String[]    kolumnNamn;

    // tabellens data
    private Object[][]    data = null;

    // display för data i modellen
    JTextArea    display = null;

    // initiera antalet rader och kolumner i tabellen, och
    // kolumnernas namn
    public TabellModell (int antalRader, int antalKolumner,
                         String[] kolumnNamn)
    {
        this.antalRader = antalRader;
        this.antalKolumner = antalKolumner;
        this.kolumnNamn = kolumnNamn;
    }

    // tabellens data
    public void setData (Object[][] data)
    {
        this.data = data;

        // uppdatera tabellen efter att underliggande data ändrats
        this.fireTableDataChanged ();
    }

    // displayen
    public void setDisplay (JTextArea display)
    {
        this.display = display;
    }

    // antalet rader i tabellen
    public int getRowCount ()
    {
        return antalRader;
    }

    // antalet kolumner i tabellen
```

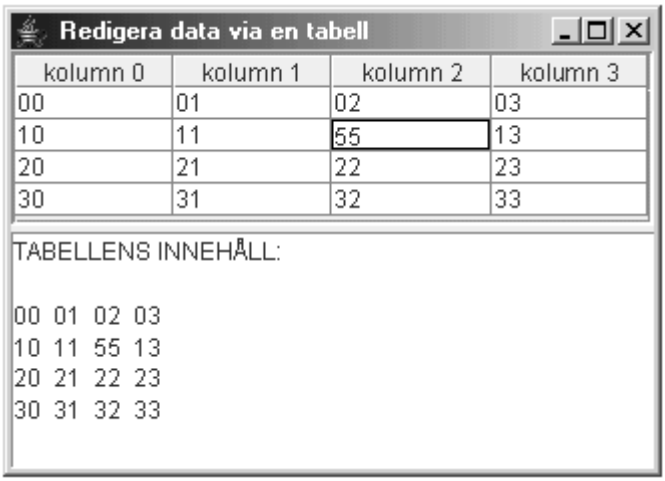
```
public int getColumnCount ()
{
    return antalKolumner;
}

// data som ska visas i en bestämd cell i tabellen
public Object getValueAt (int rad, int kolumn)
{
    Object    element = data[rad][kolumn];

    return element;
}

// namn på en kolumn i tabellen
public String getColumnName (int kolumn)
{
    return kolumnNamn[kolumn];
}

// tabellens innehåll ska kunna redigeras
public boolean isCellEditable (int rad, int kolumn)
{
    // låt en godtycklig cell vara redigerbar
    // (det är möjligt att tillåta redigeringen bara för
    // vissa celler)
    return true;
}

// ändra underliggande data när data i motsvarande cell ändras
// (den här metoden anropas automatiskt när redigeringen
// i en cell avslutas)
public void setValueAt (Object inputVarde, int rad,
                        int kolumn)
{
    // ändra data i modellen
    data[rad][kolumn] = inputVarde;
    // Det värde som matas in via en cell i tabellen
    // (inputVarde) kan modifieras, utan att den
    // underliggande modellen modifieras. Men i så fall
    // blir detta värde inte synligt i tabellen
    // i fortsättningen. Tabellen ritas om efter
    // varje ändring, och omritningen baseras på den
    // aktuella modellen.
    // Bara de ändringar som når modellen är varaktiga.

    // visa data i modellen på displayen (för att se
    // att ändringen i en cell medför ändringen i modellen)
    String    text = "TABELLENS INNEHÅLL:\n\n";
    for (int i = 0; i < antalRader; i++)
    {
        for (int j = 0; j < antalKolumner; j++)
```

```
                text = text + data[i][j] + "  ";
            text = text + "\n";
        }
        display.setText (text);
    }
}

// en behållare som innehåller en tabell och en textarea
class FSplitPane extends JSplitPane
{
    // en tabell
    private JTable    tabell;

    // en textarea
    private JTextArea    display;

    // initiera den här behållaren
    public FSplitPane ()
    {
        // justera inställningar i den här behållaren
        super (JSplitPane.VERTICAL_SPLIT);
        this.setContinuousLayout (true);
        this.setDividerSize (5);
        this.setDividerLocation (140);

        // en modell för en tabell
        int    antalRader = 4;
        int    antalKolumner = 4;

        String[]    kolumnNamn = new String[antalKolumner];
        for (int i = 0; i < antalKolumner; i++)
            kolumnNamn[i] = "kolumn " + i;

        String[][]   data = new String[antalRader][antalKolumner];
        for (int i = 0; i < antalRader; i++)
            for (int j = 0; j < antalKolumner; j++)
                data[i][j] = new String (i + "" + j);

        TabellModell    modell = new TabellModell (antalRader,
                                        antalKolumner, kolumnNamn);
        modell.setData (data);

        // tabellen
        tabell = new JTable (modell);
        // en bestämd cell i tabellen ska kunna väljas
        tabell.setCellSelectionEnabled (true);

        // textarean
        display = new JTextArea (6, 10);

        // visa tabellens startdata i textarean
```

```
        String    text = "TABELLENS INNEHÅLL:\n\n";
        for (int i = 0; i < antalRader; i++)
        {
            for (int j = 0; j < antalKolumner; j++)
                text = text + data[i][j] + "  ";
            text = text + "\n";
        }
        display.setText (text);

        // binda modellen till textarean, så att ändringar i
        // modellen blir synliga i textarean
        modell.setDisplay (display);

        // placera komponenterna i den här behållaren
        this.setTopComponent (new JScrollPane (tabell));
        this.setBottomComponent (display);
    }
}

// ett program som visar hur olika data redigeras via en tabell
class RedigeraDataViaEnTabell
{
    public static void main (String[] args)
    {
        // en behållare med en tabell och en textarea
        FSplitPane    behallare = new FSplitPane ();

        // ett fönster som innehåller behållaren
        JFrame   frame =
                 new JFrame (" Redigera data via en tabell");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 280, 120, 120);
        frame.add (behallare);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en tabell och en textarea visas. Användaren kan komma åt en cell i tabellen och ändra motsvarande uppgift. Textarean avspeglar tabellens aktuella innehåll.



Val via en tabell

ValViaEnTabell.java

Ett program som visar hur olika val görs via en tabell

En uppsättning symboler i en tabell kan väljas. Olika symboler i en tabell kan bindas till olika alternativ i ett program. I så fall väljs ett bestämt alternativ genom att en eller flera symboler i tabellen väljs.

```
import javax.swing.*;        // JTable, ListSelectionModel,
                             // JTextArea, JScrollPane, JFrame
import javax.swing.event.*;  // ListSelectionEvent,
                             // ListSelectionListener

// en modell för en tabell
class TabellModell extends javax.swing.table.AbstractTableModel
{
    // antalet rader och kolumner i tabellen
    private int       antalRader;
    private int       antalKolumner;

    // namn på tabellens kolumner
    private String[]    kolumnNamn;

    // tabellens data
    private Object[][]    data = null;
```

```
    // initiera antalet rader och kolumner i tabellen, och
    // kolumnernas namn
    public TabellModell (int antalRader, int antalKolumner,
                         String[] kolumnNamn)
    {
        this.antalRader = antalRader;
        this.antalKolumner = antalKolumner;
        this.kolumnNamn = kolumnNamn;
    }

    // tabellens data
    public void setData (Object[][] data)
    {
        this.data = data;

        // uppdatera tabellen efter att underliggande data ändrats
        this.fireTableDataChanged ();
    }

    // antalet rader i tabellen
    public int getRowCount ()
    {
        return antalRader;
    }

    // antalet kolumner i tabellen
    public int getColumnCount ()
    {
        return antalKolumner;
    }

    // data som ska visas i en bestämd cell i tabellen
    public Object getValueAt (int rad, int kolumn)
    {
        Object    element = data[rad][kolumn];

        return element;
    }

    // namn på en kolumn i tabellen
    public String getColumnName (int kolumn)
    {
        return kolumnNamn[kolumn];
    }
}

// en behållare som innehåller en tabell och en textarea
class FSplitPane extends JSplitPane
{
    // en tabell
```

```
private JTable    tabell;

// en textarea
private JTextArea    display;

// initiera den här behållaren
public FSplitPane ()
{
    // justera inställningar i den här behållaren
    super (JSplitPane.VERTICAL_SPLIT);
    this.setContinuousLayout (true);
    this.setDividerSize (5);
    this.setDividerLocation (140);

    // en modell för en tabell
    int    antalRader = 7;
    int    antalKolumner = 4;

    String[]    kolumnNamn = new String[antalKolumner];
    for (int i = 0; i < antalKolumner; i++)
        kolumnNamn[i] = "" + i;

    String[][]   data = new String[antalRader][antalKolumner];
    for (int i = 0; i < antalRader; i++)
        for (int j = 0; j < antalKolumner; j++)
          data[i][j] = new String ((char) ('A' + i) + "" + j);

    TabellModell    modell = new TabellModell (antalRader,
                                  antalKolumner, kolumnNamn);
    modell.setData (data);

    // tabellen
    tabell = new JTable (modell);
    // en bestämd cell i tabellen ska kunna väljas (markeras)
    tabell.setCellSelectionEnabled (true);

    // textarean
    display = new JTextArea (6, 10);
    display.setText ("VALDA SYMBOLER:");

    // hantera val genom tabellen

    // en modell som gör det möjligt att välja olika
    // symboler i tabellen (en modell som baseras på val
    // via rader)
    ListSelectionModel    valModell =
                            tabell.getSelectionModel ();
    // om olika val ska baseras på tabellens kolumner, skapas
    // motsvarande modell så här:
    // ListSelectionModel    valModell =
    //          tabell.getColumnModel ().getSelectionModel ();
```

```
        // Valet kan begränsas till celler i enstaka rader:
        // valModell.setSelectionMode (
        //                   ListSelectionModel.SINGLE_SELECTION);
        // Valet kan begränsas till celler i rader
        // i ett intervall:
        // valModell.setSelectionMode (
        //          ListSelectionModel.SINGLE_INTERVAL_SELECTION);
        // Val av godtyckliga celler kan tillåtas:
        // valModell.setSelectionMode (
        //        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        // det här är förvalt val

        // reagera på användarens val
        valModell.addListSelectionListener (
                                new ListSelectionListener ()
            {
                public void valueChanged (ListSelectionEvent e)
                {
                    // valda rader och kolumner
                    int[]  valdaRader = tabell.getSelectedRows ();
                    int[]    valdaKolumner =
                                    tabell.getSelectedColumns ();

                    // bestäm valda symboler
                    String  valdaSymboler = "VALDA SYMBOLER:\n\n";
                    for (int i = 0; i < valdaRader.length; i++)
                    {
                        for (int j = 0; j < valdaKolumner.length;
                                                        j++)
                        {
                            Object  enSymbol = tabell.getValueAt (
                                 valdaRader[i], valdaKolumner[j]);
                            valdaSymboler += enSymbol + "    ";
                        }
                        valdaSymboler += "\n";
                    }

                    // visa valda symboler
                    display.setText (valdaSymboler);
                }
            } );

        // placera komponenterna i den här behållaren
        this.setTopComponent (new JScrollPane (tabell));
        this.setBottomComponent (new JScrollPane (display));
    }
}

// ett program som visar hur olika val görs via en tabell
class ValViaEnTabell
{
```

```
    public static void main (String[] args)
    {
        // en behållare med en tabell och en textarea
        FSplitPane    behallare = new FSplitPane ();

        // ett fönster som innehåller behållaren
        JFrame    frame = new JFrame (" Val via en tabell");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setBounds (400, 280, 120, 120);
        frame.add (behallare);
        frame.setVisible (true);
    }
}
```

Programmets GUI

Ett fönster som innehåller en tabell och en textarea visas. Användaren kan markera (välja) ett antal celler i tabellen. I textarean visas innehållet i de markerade cellerna.