Föreläsning 12

Rörliga figurer
Klassen Timer
Undantag
Något om applets

Klassen `javax.swing.Timer`

I `Swing` finns en klass `Timer` som man kan använda för att upprepa en vis kodsekvens med jämna tidsmellanrum.

Ett objekt av klassen `Timer` exekveras som en egen tråd. Ett objekt av klassen `Timer` kan med jämna tidsmellanrum generera händelser av typen `ActionEvent`. Hur ofta dessa händelser genereras och vem som lyssnar efter dessa händelser anges som parameter till konstruktorn:

```
Timer t = new Timer(upprepningstiden, lyssnare);
```

Det går också att lägga till flera lyssnare till samma `Timer`-objekt med hjälp av metoden `addActionListener`.

```
t.addActionListner(lyssnare2);
```

Alla registrerade lyssnare på ett `Timer`-objekt får de `ActionEvent`-händelser som `Timer`-objektet genererar.

Klassen javax.swing.Timer

För att ett Timer-objekt skall börja generera händelser måste man anropa metoden start:

```
t.start();
```

Det går att stoppa ett Timer-objekt med metoden stop:

```
t.stop();
```

och det går att återstarta ett Timer-objekt med metoden restart:

```
t.restart();
```

Man kan se efter om ett Timer-objekt har startats med metoden isRunning:

```
t.isRunning();
```

Om inget annat anges genereras den första händelsen efter det antal millisekunder som angav i konstruktorn av Timer-objektet. Vill man ha ett annat tidsintervall till den första händelsen används metoden setInitialDelay:

```
t.setInitialDelay();
```

Det går att ändra tidsintervallet för genereringen av händelser med metoden setDealy:

```
t.setDelay(100);
```

Det finns ytterligare ett antal metoder i klassen Timer.

Klassen javax.swing.Timer är lämplig att använda för att skapa animerade grafiska objekt.

Problemexempel

Ett program som visar en rektangel som flyttar sig från höger till vänster över ritytan.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MovingRect extends JPanel implements ActionListener {
   private Timer t = new Timer(50, this);
   private int xPos = 0;
   public MovingRect() {
      setBackground(Color.WHITE);
      t.restart();
   }//konstruktor

   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      pen.setColor(Color.BLUE);
      int w = 30;
      int h = 20;
      int x = getWidth() - xPos;
      int y = (getHeight() -h) / 2;
      pen.fillRect(x,y,w,h);
      xPos = (xPos + 1) % (getWidth() + w);
   }//paintComponent
```

Problemexempel

```
   public void actionPerformed(ActionEvent e) {
      repaint();
   }//actionPerformed
   public static void main(String[] arg) {
      JFrame w = new JFrame();
      MovingRect m = new MovingRect();
      w.getContentPane().add(m);
      w.setSize(400,100);
      w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      w.setVisible(true);
   }//main
}//MovingRect
```


Problemexempel

En klass för att visa en rullande text som rör sig från vänster till höger i en etikett.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MovingText extends JLabel implements ActionListener {
   private javax.swing.Timer t = new javax.swing.Timer(50, this);
   private String str;
   private int xPos = 0;
   public MovingText(String str) {
      this.str = str;
      setBackground(Color.YELLOW);
      setForeground(Color.BLUE);
      setOpaque(true);
      t.restart();
   }//konstruktor
```

Problemexempel

```
   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      FontMetrics fm = pen.getFontMetrics();
      int s = fm.stringWidth(str);
      if (xPos < getWidth() + s)
         xPos = xPos + 1;
      else
         xPos = 0;
      int yPos = getHeight()/2 + fm.getHeight()/2;
      pen.drawString(str, getWidth()-xPos, yPos);
   }//paintComponent

   public void actionPerformed(ActionEvent e) {
      repaint();
   }//actionPerformed

   public void changeText(String str) {
      this.str = str;
   }//changeText
}//MovingText
```


Problemexempel

En klass som innehåller ett huvudprogram som skapar ett fönster med två objekt av klassen `MovingText`:

```
import java.awt.*;
import javax.swing.*;
public class TestMovingText {
   public static void main(String[] arg) {
      JFrame w = new JFrame();
      String str1 = "Denna text rör sig rullande från höger till vänster.";
      String str2 = "Viktigt meddelande! Dags att anmäla sig till tentan";
      MovingText message1 = new MovingText(str1);
      MovingText message2 = new MovingText(str2);
      message2.setForeground(Color.RED);
      w.getContentPane().setLayout(new GridLayout(2,1));
      w.getContentPane().add(message1);
      w.getContentPane().add(message2);
      w.setSize(400,100);
      w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      w.setVisible(true);
   }
}//TestMovingText
```

Problemexempel

Modifiering av klassen SnowFall från föregående föreläsning på så sätt att den ritar om sig själv var 5:e sekund.

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;

public class SnowFall extends JPanel implements ActionListener {
   private static Random slump = new Random();
   private javax.swing.Timer t = new javax.swing.Timer(5000, this);

   public SnowFall() {
      setBackground(new Color(123,123, 200));
      t.restart();
   }//konstruktor

   public void actionPerformed(ActionEvent e) {
      repaint();
   }//actionPerformed

   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      drawSnowFall(pen);
   }//paintComponent
```


Problemexempel

```
 public void drawSnowFall(Graphics pen) {
       pen.setColor(Color.white);
       int antal = slump.nextInt(40) + 20;
       for(int i = 1; i <= antal; i = i + 1) {
          int radie = slump.nextInt(15) + 10;
          int x = slump.nextInt(getWidth() - 2*radie);
          int y = slump.nextInt(getHeight() - 2*radie);
         drawSnowFlake(pen, x, y, radie);
       }
   }//drawSnowFall

   public void drawSnowFlake(Graphics pen, int x, int y, int radie) {
       int x0 = x + radie;
       int y0 = y + radie;
       for (int i = 0; i < 360; i = i + 20) {
          int x1 = x0+ (int) (radie * Math.cos(Math.toRadians(i)));
          int y1 = y0 + (int) (radie * Math.sin(Math.toRadians(i)));
          pen.drawLine(x0, y0, x1, y1);
       }
   }//drawSnowFlake
}//SnowFall
```

Problemexempel

En klass som innehåller ett huvudprogram som skapar ett fönster med ett objekt av klassen SnowFa11:

```
 import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestSnowFall {
   public static void main(String[] args) {
      JFrame window = new JFrame();
      SnowFall art = new SnowFall();
      window.setSize(300, 300);
      window.getContentPane().add(art);
      window.setLocation(50,50);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setVisible(true);
   }//main
}//TestSnowFall
```


Undantag – exceptionella händelser

Ett undantag (*exception*) är en händelse under exekveringen som signalerar att ett fel har uppstått.

Undantag är vanligtvis svåra att gardera sig emot och ligger ofta utanför programmets direkta ansvarsområde .

Om undantaget inte tas om hand avbryts programmet och ett felmeddelande skrivs ut.

Java har inbyggda mekanismer för att handha undantag.

Vi skall här titta på de mest grundläggande språkkonstruktionerna för undantagshantering.

Orsaker till undantag

Några orsaker till undantag:

- Programmeringsfel (refererar till ett objekt som inte finns, adresserar utanför giltiga index i ett fält, …).
- Användaren av koden har inte läst dokumentationen (anropar metoder på fel sätt).
- Resursfel (nätverket gick ner, hårddisken är full, minnet är slut, databasen har kraschat, DVD:n var inte insatt, ...).

Hantera undantag

För att handha undantag tillhandahåller Java:

- konstruktioner för att "kasta" undantag (konstruktionen **throw**)
- konstruktioner för att "specificera" att en metod kastar eller vidarebefordrar udantag (konstruktionen **throws**)
- konstruktioner för att fånga undantag (konstruktionerna **try**, **catch** och **finally**).

Felhanteringen blir därmed en explicit del av programmet, d.v.s. felhanteringen blir synlig för programmeraren och kontrolleras av kompilatorn.

Hantering av undantag

När det gäller hantering av undantag kan man ha olika ambitionsnivåer:

- Ta hand om händelsen och försöka vidta någon lämplig åtgärd i programenheten där felet inträffar så att exekveringen kan fortsätta.
- Fånga upp händelsen, identifiera den och skicka den vidare till anropande programenhet.
- Ignorera händelsen, vilket innebär att programmet avbryts om händelsen inträffar.

Undantag – exempel

Ett programexempel:

```
import javax.swing.*;
public class Squar {
   public static void main (String[] arg) {
      String indata = JOptionPane.showInputDialog("Ange ett heltal:");
      int tal = Integer.parseInt(indata);
       int res = tal * tal;
      JOptionPane.showMessageDialog(null, "Kavadraten av talet är " + res);
   } //main
} // Squar
```

Vad händer som användaren t.ex. anger "12.3" eller "ett" som indata?

Programmet avbryts och en felutskrift erhålls:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "12.3"at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:458)
at java.lang.Integer.parseInt(Integer.java:499)
at Squar.main(Squar.java:5)
```

Undantag – olika typer

Undantag kan inträffar, som i exemplet ovan, om en användare ger felaktig indata eller om programmeraren tänkt fel och förbisett olika situationer som kan inträffa.

Några vanliga undantag och felen som orsakar dessa:

ArrayIndexOutOfBoundsException
Försök att i ett fält indexera ett element som inte finns

NullPointerException
Försök att anropa ett objekt vars referens har värdet **null**

NumberFormatException
Försök att konvertera en stäng till ett tal, där strängen innehåller otillåtna tecken.

IllegalArgumentExecption
Försök att anropa en metod med ett otillåtet argument.

Kasta exceptions - Exempel

Låt oss säga att vi vill skriva en klass Account för bankkonton. Vi vill bland annat ha en metod withdraw som tar ut pengar ur kontot. Men vad ska vi göra om pengarna inte räcker till?

```
public class Account {
   private int balance;
   //flera instansvariabler och metoder som inte visas här
   public void withdraw(int amount) {
      if (amount < balance) {
         balance = balance - amount;
      }
      else {
         // Vad ska vi göra här?
      }
   }//withdraw
}//Account
```

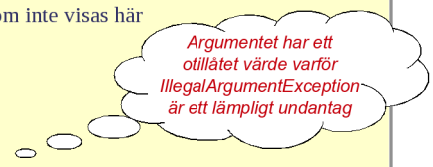
Kasta exceptions - Exempel

Ett sätt är att kasta ett exception. Detta görs med en **throw**-sats enligt följande kodmönster:

```
throw new ExceptionType();
```
där ExceptionType är en existerande undantagsklass.

Vårt programexempel blir:

```
public class Account {
   private int balance;
   //flera instansvariabler och metoder som inte visas här
   public void withdraw(int amount) {
      if (amount < balance) {
         balance = balance - amount;
      }
      else {
        throw new IllegalArgumentException("Sorry, you are short of money!");
      }
   }//withdraw
}//Account
```



Att fånga undantag

För att fånga undantag tillhandahåller Java **try-catch-finally**-satsen.

Kodmönster:

```
try {
    <Kod som kan kasta (generera) ett undantag>
} catch (ExceptionType e) {
   <Kod som vidtar lämpliga åtgärder om undantaget ExceptionType inträffar>
} finally {
   <Kod som utförs oberoende av om undantaget inträffar eller inte>
}
```

Om koden i **try**-blocket inte kastas något undantag av ExceptionType ignoreras **catch**-blocket. Om ett undantag av typen ExceptionType inträffar avbryts exekveringen i **try**-blocket och fortsätter i **catch**-blocket.

finally-blocket exekveras oberoende av om ett undantag har inträffat eller inte.

finally-blocket kan utelämnas och vi diskuterar inte detta ytterligare på kursen.

Att fånga undantag - exempel

I vårt tidigare programexempel är det möjligt att åtgärda det uppkomna undantaget genom att låta användaren göra en ny inmatning:

```
import javax.swing.*;
public class FaultTolerantSquar {
   public static void main (String[] arg) {
      while (true) {
         String indata = JOptionPane.showInputDialog("Ange ett heltal:");
         try {
            int tal = Integer.parseInt(indata);
            int res = tal * tal;
            JOptionPane.showMessageDialog(null, "Kavadraten av talet är " + res);
            break;
         } catch (NumberFormatException e) {
             JOptionPane.showMessageDialog(null, "Otillåten indata. Försök igen!");
         }
      }
   } //main
} // FaultTolerantSquar
```


Att skapa egna undantagstyper

Det finns fördefinierade typer av undantag:

- ArrayIndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- IllegalArgumentExecption

. . .

Man kan också skapa egna typer av undantag genom att skapa subklasser till klassen RuntimeException (eller till klassen Exception).

```
public class MyException extends RuntimeException {
   public MyException() {
      super();
   }
   public MyException(String str) {
      super(str);
   }
}//MyException
```

Att skapa egna exceptionella händelser

```
public class MyException extends RuntimeException {
   public MyException() {
      super();
   }
   public MyException(String str) {
      super(str);
   }
}//MyException
```

Konstruktorn MyException(String str) används för att beskriva det uppkomna felet. Beskrivningen kan sedan läsas när felet fångas m.h.a. metoden getMessage(), som ärvs från superklassen. Om felet inte fångas i programmet kommer den inlagda texten att skrivas ut när programmet avbryts.

Även metoden printStackTrace(). Denna metod skriver ut var felet inträffade och "spåret" av metoder som sänt felet vidare. Metoden printStackTrace() anropas automatiskt när en exceptionell händelse avbryter ett program.

Checked and Unchecked Exceptions

I Java skiljer man på två typer av undantag:

- unchecked exceptions, som är subklasser till RuntimeException
- checked exceptions, som är subklasser till Exception

De exception vi sett hittills har varit unchecked exceptions.

Unchecked Exceptions

Unchecked exceptions kan förekomma i princip var som helst och beror ofta på programmeringsfel. Vanligtvis låter man där bli att fånga denna typ av undantag, eftersom det är koden som skall rättas till. Undantaget fångas endast om det orsakas av interaktionen med en yttre användare.

Exempel:

ArithmeticException — När resultatet av en aritmetisk operation inte är väldefinierad, t.ex. division med noll.

NullPointerException — När man försöker använda en referensvariabel som inte har blivit tilldelad ett objekt

```
Scanner sc;
sc.nextInt();
```

IllegalArgumentException — Används när en metod får ett argument som inte kan behandlas.

Checked Exceptions

Checked exceptions används för att signalera fel som ofta *inte är programmeringsfel*. Dessa fel beror ofta på saker som är utanför programmerarens kontroll. Därför är det viktigt att fånga denna typ av undantag återställa programmet ett giltig tillstånd eller avbryta programmet på ett kontrollerat sätt.

Exempel:

FileNotFoundException — när man försöker läsa en fil som inte existerar.

LineUnavailableException — en ljudenhet man försöker använda är upptagen av ett annat program.

Checked Exceptions

Om man anropar en metod som kan kasta ett checked exception måste man antingen fånga undantaget eller deklarera att man kastar det vidare.

Om man vill kasta ett exception vidare i en metod deklarerar man detta genom att lägga till **throws** och namnet på typen för det undantag man vill kasta vidare. Detta skrivs efter parametrarna till en metod.

```
public static void main(String[] args) throws FileNotFoundException {
    ....
}
```

När man anropar en metod som kan kasta ett checked exception måste man alltså *aktivt ta ställning till om undantaget skall hanteras eller kastas vidare*.

Applets

När Sun microsystems började utveckla Java var tanken att skapa ett språk som var specialanpassat för inbyggda system, framförallt i konsumentelektronik (kameror, mikrovågsugnar, vidoapparater mm). En viktig tanke var att språket skulle vara plattformsoberoend.

Detta floppade dock totalt!

Men under arbetets gång hade Internet och framförallt WWW-utvecklingen tagit ordentlig fart, och tanken med ett plattformsoberoend språk fick plötsligt ett annat och mycket större användningsområde.

Sun lade om utvecklingsstrategin för Java och lanserade Java som programspråket för Internet.

En speciell typ av Java-program som kallas för applets köras från en wbbläsare (Internet Explorer, Netscape osv).

Java-program som inte är applets, kallas för en applikation eller ett fristående program.

Applets

En applet är ett grafiskt program och i paketet Swing finns klassen `JApplet` för att skapa applets.

Klassen `JApplet` är en underklass till `JPanel` och har i stort samma egenskaper som denna, vilket betyder att en applet kan innehålla grafiska komponenter.

När en webbläsare läser en webbsida som innehåller en applet kopieras de körbara filerna (dvs Javabytekoden) till den egna maskinen och körs där (här har vi en orsak till varför det är viktigt att Java är plattformsoberoende).

Av säkerhetsskäl tillåts man dock inte i en applet att göra allt som är möjligt att göra i Java-applikation.

Applets

Websidor skrivs i ett särskilt språk som kallas HTML och en referens till en applet görs med en särskild sk HTML- tag.

Om källkoden för det program man vill lägga in på en websida har namnet MinApplet.java se det i HTML-koden ut på följande sätt:

```
<HTML>
<HEAD>
<TITLE> En Applet</TITLE>
</HEAD>
<BODY>
. . .
<APPLET
CODE="MinApplet.class" WIDTH=150 HEIGHT=150>
</APPLET>
. . .
</BODY>
</HTML>
```

Filen som innehåller HTML-filen skall ha ett namn som har postfixet .html, t ex MinApplet.html.

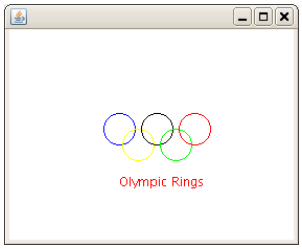
Hur skapar man en applet

Vid en jämförelse mellan en fristående applikation och en applet är det vissa skillnades som kan noters:

- en applet ärver alltid sina egenskaper från standardklassen `JApplet` som finns i paketet `javax.swing.JApplet`.
- i en applet skapar man inget fönster varför man inte heller sätter någon storlek på det eller gör det synligt (fönstret finns redan och storleken fås från web-sidan)
- en applet har ingen `main`-metod istället har en applet en `init`-metod
- `init`-metoden i en applet ersätter konstruktorn i en applikation (nästan sant)

Exempel

På tidigare föreläsning konstruerade vi en Java applikation som ritar ut de olympiska ringarna enligt figuren nedan:



Vi skall nu konstruera en applet som gör samma sak:

Implementationen för applikationen:

```
import java.awt.*;
import javax.swing.*;
public class OlympicRings extends JPanel {
   public OlympicRings() {
      setBackground(Color.white);
   }//konstruktor
 public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      pen.setColor(Color.blue);
      pen.drawOval(90,80,30,30);
      pen.setColor(Color.yellow);
      pen.drawOval(108,95,30,30);
      pen.setColor(Color.black);
      pen.drawOval(126,80,30,30);
      pen.setColor(Color.green);
      pen.drawOval(144,95,30,30);
      pen.setColor(Color.red);
      pen.drawOval(162,80,30,30);
      pen.drawString("Olympic Rings", 105, 150);
   }//paintComponent
}//OlympicRings
```

```
public class Main {
   public static void main(String[] args) {
      JFrame window = new JFrame();
      OlympicRings rings = new OlympicRings();
      window.setSize(282, 230);
      window.getContentPane().add(rings);
      window.setLocation(50,50);
      window.setVisible(true);
   }//main
//Main
```


Hur man gör om applikation till en applet

Utgår vi från applikationen är det enklet att göra om denna till en applet. Detta gör vi genom följande 4 steg:

1. ärv från klassen `JApplet` istället för från klassen `JPanel`
2. ta bort main-metoden
3. ändra namnet på metoden `paintComponent` till `paint` och ta bort satsen `super.paintComponent()` (det finns ingen sådan metod).
4. ta bort konstruktorn och lägg koden som fanns där istället i metoden `init`

Och vi har vår applet!!

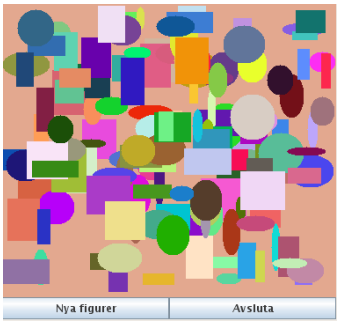
```
import javax.swing.*;
import java.awt.*;
public class OlympicApplet extends JApplet {
   public void init() {
      setBackground(Color.WHITE);
   }//init
   public void paint(Graphics pen) {
      pen.setColor(Color.BLUE);
      pen.drawOval(90,80,30,30);
      pen.setColor(Color.YELLOW);
      pen.drawOval(108,95,30,30);
      pen.setColor(Color.BLACK);
      pen.drawOval(126,80,30,30);
      pen.setColor(Color.GREEN);
      pen.drawOval(144,95,30,30);
      pen.setColor(Color.RED);
      pen.drawOval(162,80,30,30);
      pen.drawString("Olympic Rings", 105, 150);
    }//paint
}//OlympicApplet
```


Websidan får följande utseende

```
<HTML>
   <HEAD>
      <TITLE> Olympiad </TITLE>
   </HEAD>
   <BODY>
      . . .
      <APPLET
         CODE="OlympicApplet.class" WIDTH=300 HEIGHT=200>
      </APPLET>
      . . .
    </BODY>
</HTML>
```

Exempel

I förra föreläsningen skrev en applikation som använde klassen `RandomArt` för att skapa en fönster med nedanstående utseende. När man trycker på *Nya figurer* skall en ny uppsättning slumpmässiga figurer ritas, som antingen enbart består av rektanglar eller är en blandning av rektanglar och ovaler. Detta innebär att slumpen skall avgöra om metoden `randomRectangels` eller `randomFigures` anropas. När man trycker på *Avsluta* skall programmet avslutas.



Implementationen för applikationen:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestSlumpFig extends JFrame implements ActionListener {
   private JButton ny = new JButton("Nya figurer");
   private JButton sluta = new JButton("Avsluta");
   private JPanel knappar = new JPanel();
   private RandomArt figurer = new RandomArt();
   public TestSlumpFig() {
      knappar.setLayout(new GridLayout(1,2));
      knappar.add(ny);
      ny.addActionListener(this);
      sluta.addActionListener(this);
      knappar.add(sluta);
      setLayout(new BorderLayout());
      add("Center", figurer);
      add("South", knappar);
      setSize(300,300);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
   }//konstruktor
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == ny)
         figurer.repaint();
      else if (e.getSource() == sluta)
         System.exit(0);
   }
   public static void main(String[] args) {
      TestSlumpFig tsf = new TestSlumpFig();
   }//main
}//TestSlumpFig
```

En applet fås med samma recept som tidigare!

1. ärv från klassen `JApplet` istället för från klassen `JPanel`
2. ta bort main-metoden
3. ändra namnet på metoden `paintComponent` till `paint` och ta bort satsen `super.paintComponent()`.
4. ta bort konstruktorn och lägg koden som fanns där istället i metoden `init`

Här har vi en sak att notera. Avslutaknappen är meningslös då det inte tillåts att avsluta en applet med `System.exit(0)`. Appleten avslutas när man lämnar webbläsaren (och stoppas tillfälligt när man lämnar den aktuella sidan).


```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class AppletSlumpFig extends JApplet implements ActionListener {
   private JButton ny = new JButton("Nya figurer");
   private JButton sluta = new JButton("Avsluta");
   private JPanel knappar = new JPanel();
   private RandomArt figurer = new RandomArt();
   public void init() {
      knappar.setLayout(new GridLayout(1,2));
      knappar.add(ny);
      ny.addActionListener(this);
      sluta.addActionListener(this);
      knappar.add(sluta);
      getContentPane().setLayout(new BorderLayout());
      getContentPane().add("Center", figurer);
      getContentPane().add("South", knappar);
      setSize(300,300);
   }//init
      public void actionPerformed(ActionEvent e) {
         if (e.getSource() == ny)
            figurer.repaint();
         else if (e.getSource() == sluta)
            ; //gör inget. En applet kan inte avslutas med en knapp!!
      }//actionPerformed
}//AppletSlumpFig
```