Föreläsning 11

Att rita egna bilder.

Grafik

Varje gång ett fönster behöver ritas om, pga av att det flyttas eller varit övertäckt, anropas automatiskt en metod som har namnet `paintComponent`.

I ett program kan man också begära omritning av fönstret genom ett anrop av metoden `repaint`, som i sin tur anropar `paintComponent`.

Varje grafisk komponent har sin egen fördefinierade `paintComponent` metod.

Om komponenten är en behållare görs vid omritning anrop till *alla komponenter som behållaren innehåller* för att de också skall rita om sig.

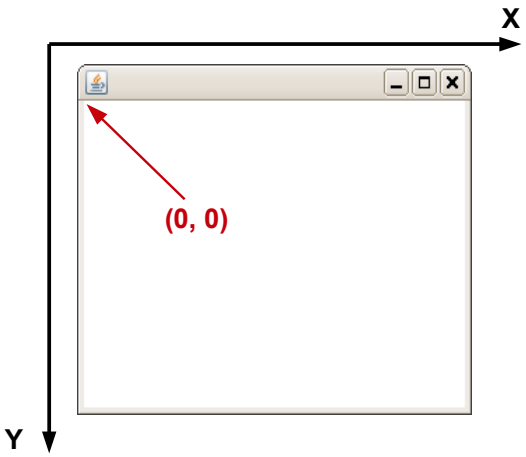
Om man vill ändra utseendet på en komponent skall man överskugga metoden `paintComponent`, eftersom det är denna metod som definierar utseendet på komponenten.

För att rita använder man ett objekt av klassen `Graphics`, som kan ses som en (avancerad) penna.

```
public void paintComponent(Graphics pen)
```

Fönstrets koordinatsystem

Koordinatsystemet som används i Java har en horisontell x-axel och en vertikal y-axel, med origo (0,0) i övre vänstra hörnet av fönstret.



Enheten som används är pixels.

Klassen `Graphics`

När man skall rita egna figurer används ett objekt av klassen `Graphics`. Ett sådant objekt kan ses som en avancerad en penna och det finns en rad olika metoder i klassen för att kunna ge pennan egenskaper och kunna utföra saker med pennan:

- skriva text
- rita linjer
- rita cirkelbågar
- rita rektanglar
- rita ovaler
- rita polygon
- sätta färger
- sätta fonter (skrivstil)
- m.m

Metoder i klassen `Graphics`

void drawString(String txt, **int** x, **int** y)

Ritar ut texten txt med början i punkt (x,y). y-koordinaten anger textens baslinje.

void drawLine(**int** x1, **int** y1, **int** x2,**int** y2)

Ritar en linje från punkten (x1,y1) till (x2,y2)

void drawRect(**int** x, **int** y, **int** b, **int** h)

Ritar en ofylld rektangel med övre vänstra hörnet i punkten (x,y), bredden b och höjden h.

void fillRect(**int** x, **int** y, **int** b, **int** h)

Ritar en fylld rektangel med övre vänstra hörnet i punkten (x,y), bredden b och höjden h.

void drawRoundRect(**int** x, **int** y, **int** b, **int** h, **int** m, **int** n)

Ritar en ofylld rundad rektangel med övre vänstra hörnet i punkten (x,y), bredden b och höjden h. Rundningen är m/2 bred och n/2 hög.

Metoder i klassen `Graphics`

void fillRoundRect(**int** x, **int** y, **int** b, **int** h, **int** m, **int** n)

Ritar en fylld rundad rektangel med övre vänstra hörnet i punkten (x,y), bredden b och höjden h. Rundningen är m/2 bred och n/2 hög.

void drawOval(**int** x, **int** y, **int** b, **int** h)

Ritar en ofylld ellips som är inskriven i en rektangel med övre vänstra hörn i (x,y), bredden b och höjden h.

void fillOval(**int** x, **int** y, **int** b, **int** h)

Ritar en fylld ellips som är inskriven i en rektangel med övre vänstra hörn i (x,y), bredden b och höjden h.

void drawArc(**int** x, **int** y, **int** b, **int** h, **int** s, **int** l)

Ritar ett segment av en ellips inskriven i en rektangel med övre vänstra hörn i (x,y), bredden b och höjden h. s är bågens startpunkt (i grader) och *l* är bågens längd (i grader). Startpunkten 0 grader motsvarar klockan 3. Ett positivt värde på *l* anger en riktning moturs och ett negativt värde *l* på anger en riktning medurs.

Metoder i klassen `Graphics`

void fillArc(**int** x, **int** y, **int** b, **int** h,**int** s, **int** l)

Ritar en fylld båge av en ellips inskriven i en rektangel med övre vänstra hörn i (x,y), bredden b och höjden h. s är bågens startpunkt (i grader) och *l* är bågens längd (i grader). Startpunkten 0 grader motsvarar klockan 3. Ett positivt värde på *l* anger en riktning moturs och ett negativt värde *l* på anger en riktning medurs.

void drawPolygon(Polygon p)

Ritar ut den ofyllda månghörningen p.

void fillPolygon(Polygon p)

Ritar ut den fyllda månghörningen p.

void getColor()

Returnerar den aktuella förgrundsfärgen.

void setColor(Color color)

Sätter förgrundsfärgen till color.

. . . .

Klassen `DebugGraphics`

I klassen `javax.swing.DebugGraphics`, som är en subklass till `Graphics` finns en metod som kan vara värt att nämna:

void fill3DRect(**int** x, **int** y, **int** b, **int** h, **boolean** raised)

Ritar en fylld rektangel med ett tredimensionellt utseende. Rektangeln har övre vänstra hörnet i punkten (x,y), bredden b och höjden h. Om parametern raised har värdet **true** ges illusionen att rektangeln är upphöjd, annars ges illusionen att rektangeln är nedsänkt.

Klassen Color

En färg representeras med hjälp av det så kallade RGB-formatet.

RGB-formatet definierar en färg som bestående av tre värden; mängden rött, mängden grönt och mängden blått. Mängden av en färgs representeras av ett heltal i intervallet [0, 255].

Med hjälp av konstruktorn i klassen Color kan man skapa egna färger:

```
Color lightBlue = new Color(175, 175, 255);
```

Ett objekt av klassen Color har dessutom ett fjärde värde. Detta värde kallas för alpha-kanalen och markerar hur ogenomskinlig en färg är. Genomskinligheten anges, liksom färgkomponenterna, som ett heltal mellan 0 och 255. Genomskinliga färger skapas med en särskild konstruktor:

```
Color newC = new Color(175, 175, 255, 175);
```

Värdet 0 representerar helt genomskinlig och värdet 255 representerar helt ogenomskinlig. Anges inget värde få värdet 255 som default.

Klassen Color

Färger representeras i Java med klassen `java.awt.Color`, i vilken det finns ett antal färger fördefinierade:

Color.BLACK	0, 0, 0
Color.BLUE	0, 0, 255
Color.CYAN	0, 255, 255
Color.GRAY	128, 128, 128
Color.DARKGRAY	64, 64, 64
Color.LIGHTGRAY	192, 192, 192
Color.GREEN	0, 255, 0
Color.MAGENTA	255, 0, 255
Color.ORANGE	255, 200, 0
Color.PINK	255, 175, 175
Color.RED	255, 0, 0
Color.WHITE	255, 255, 255
Color.YELLOW	255, 255, 0

Typsnitt

Typsnitt (alltså utseendet på bokstäverna) skapar man i Java med klassen `Font`. När man skapar ett `Font`-objekt anger man tre argument:

```
Font f = new Font(namn, stil, storlek);
```

Namnet på ett typsnitt är något av

"Serif", "SansSerif", "Monospaced", "Dialog", "DialogInput", "Symbol".

Stilen är någon av konstanterna

Font.PLAIN, Font.ITALIC, Font.BOLD, Font.BOLD+Font.ITALIC

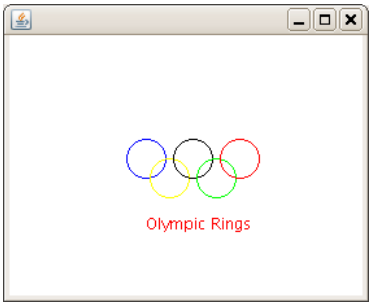
Storleken anges ett heltal; standardstorleken är 12

När man har skapat ett typsnitt kan man t ex använda det i `setFont`-metoderna som finns till de grafiska komponenterna:

```
JButton b = new JButton("Press here");
b.setFont(new Font("SansSerif", Font.BOLD, 24));
```

Exempel

Skriv ett program som ritar ut de olympiska ringarna enligt figuren nedan:



Diametern på ringarna är 30 pixels, avståndet mellan ringarna i horisontell ledd är 6 pixels. De två nedersta ringarna ligger 15 pixels nedanför de tre övre ringarna.

Implementation: Bestämda positioner i ritytan

```
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
```

Implementation: Ett huvudprogram som skapar ett fönster med ett objekt av klassen `OlympicRings`

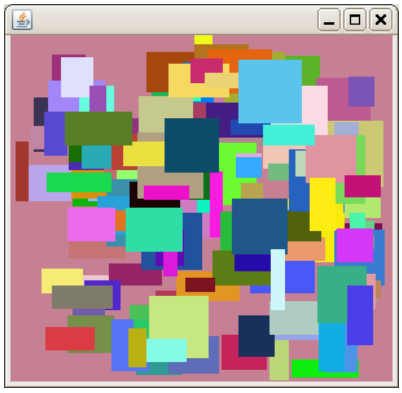

```
import java.awt.*;
import javax.swing.*;
public class MainOlympic {
   public static void main(String[] args) {
      JFrame window = new JFrame();
      OlympicRings rings = new OlympicRings();
      window.setSize(282, 230);
      window.add(rings);
      window.setLocation(50,50);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setVisible(true);
   }//main
}//MainOlympic
```

Implementation: Flytande positioner i ritytan

```
import java.awt.*;
import javax.swing.*;
public class OlympicRings2 extends JPanel {
   public OlympicRings2() {
      setBackground(Color.WHITE);
   }//konstruktor
   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      int mx = getWidth()/2;
      int my = getHeight()/2;
      pen.setColor(Color.BLUE);
      pen.drawOval(mx-51, my-15, 30, 30);
      pen.setColor(Color.YELLOW);
      pen.drawOval(mx-33, my, 30, 30);
      pen.setColor(Color.BLACK);
      pen.drawOval(mx-15, my-15, 30, 30);
      pen.setColor(Color.GREEN);
      pen.drawOval(mx+3, my, 30, 30);
      pen.setColor(Color.RED);
      pen.drawOval(mx+21, my-15, 30, 30);
      pen.drawString("Olympic Rings", mx-35, my+55);
   }//paintComponent
}//OlympicRings2
```

Exempel

Skriv ett program som ritar ut 150 rektanglar enligt figurerna nedan.
Rektanglarnas storlek, placering och färg skall genereras slumpmässigt.



Implementation:

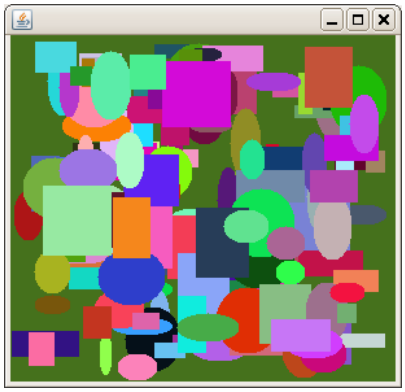
```
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class RandomArt extends JPanel {
   private static Random slump = new Random();
   public RandomArt() {}//konstruktor
   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      randomRectangels(pen);
   }//paintComponent
   private void randomRectangels(Graphics pen) {
      setCanvasColor(pen);
      for(int i = 1; i <= 150; i = i + 1) {
         int b = slump.nextInt(45) + 10;
         int h = slump.nextInt(45) + 10;
         int x = slump.nextInt(getWidth() - b);
         int y = slump.nextInt(getHeight() - h);
         pen.setColor(randomColor());
         pen.fillRect(x,y,b,h);
      }
   }//randomRectangels
   private void setCanvasColor(Graphics pen) {
      pen.setColor(randomColor());
      pen.fillRect(0, 0, getWidth(), getHeight());
   }//setCanvasColor
   private Color randomColor() {
      return new Color(slump.nextInt(256), slump.nextInt(256 ), slump.nextInt(256));
   }//randomColor
}// RandomArt
```

Huvudprogram som använder RandomArt:

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

Exempel

Utöka klassen `RandomArt` med en metod `randomFigures` för att rita ut 150 slumpmässiga rektanglar eller ovaler (enligt figuren nedan).



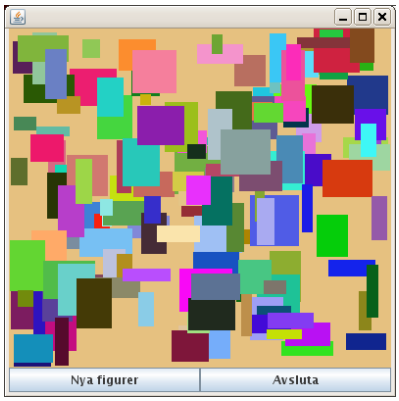
Implementation:

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class RandomArt extends JPanel {
   . . .
   private void randomFigures(Graphics pen) {
      setCanvasColor(pen);
      for(int i = 1; i <= 150; i = i + 1) {
         int b = slump.nextInt(45) + 10;
         int h = slump.nextInt(45) + 10;
         int x = slump.nextInt(getWidth() - b);
         int y = slump.nextInt(getHeight() - h);
         pen.setColor(randomColor());
         int fig = slump.nextInt(2);
         if (fig == 0)
            pen.fillRect(x,y,b,h);
         else
            pen.fillOval(x,y,b,h);
      }
   }//randomFigures
```

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

Exempel

Använd klassen `RandomArt` för att skriva en klass `ShowRandomArt` som visar 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.



Implementation

När man trycker på knappen *Nya figurer* skall en omritning av panelen som finns i fönstret ske. Denna panel är av klassen `RandomArt` och omritningen definieras av metoden `paintComponent` i klassen `RandomArt`, varför det är i denna metod det skall avgöras vilken av metoderna `randomRectangels` eller `randomFigures` som skall anropas.

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class RandomArt extends JPanel {
   . . .
   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      int artMode =  slump.nextInt(2);
      if (artMode == 0)
         randomRectangels(pen);
      else
         randomFigures(pen);
   }//paintComponent
   . . .
}// RandomArt
```

Implementation av klassen `ShowRandomArt`:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ShowRandomArt extends JFrame implements ActionListener {
   private JButton ny = new JButton("Nya figurer");
   private JButton sluta = new JButton("Avsluta");
   private RandomArt art = new RandomArt();

   public ShowRandomArt() {
      JPanel knappar = new JPanel();
      knappar.setLayout(new GridLayout(1, 2));
      knappar.add(ny);
      ny.addActionListener(this);
      sluta.addActionListener(this);
      knappar.add(sluta);
      setLayout(new BorderLayout());
      add("Center", art);
      add("South", knappar);
      setSize(300, 400);
      setResizable(false);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
   }//konstruktor

   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == ny)
         art.repaint();
      else if (e.getSource() == sluta)
         System.exit(0);
   }//actionPerformed

   public static void main(String[] args) {
      JFrame tsf = new ShowRandomArt();
   }//main
}//ShowRandomArt
```

Förhindrar att storleken på fönstret förändras

Klassen `Polygon`

I Java finns en klass `Polygon` som används för att avbilda månghörningar (klassen finns i paketet `java.awt`).

Bland annat finns följande bl.a. följande konstruktorer och instansmetoder:

Polygon()	skapar ett polygon utan hörn
addPoint(**int** x,**int** y)	lägger till hörnet (x, y)
translate(**int** dx,**int** dy)	flyttar polygonen `dx` i x-led och `dy` i y-led
contains(x,y)	ger **true** om punkten (x,y) ligger i polygonen, annars ges **false**
getBounds()	ger den minsta rektangel (av standardklassen `Rectangle`) som omsluter polygonen

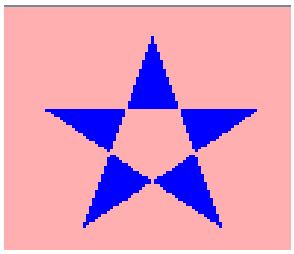
I klassen `Graphics` finns instansmetoderna

drawPolygon(Polygon p)
fillPolygon(Polygon p)

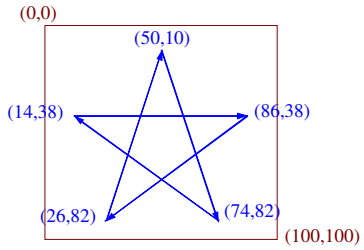
för att rita ut polygon.

Exempel

Skapa en klass `Star` som ritar ut den stjärna enligt bredvidstående figur:



Stjärnan utgör ett polygon enligt figuren bredvid:



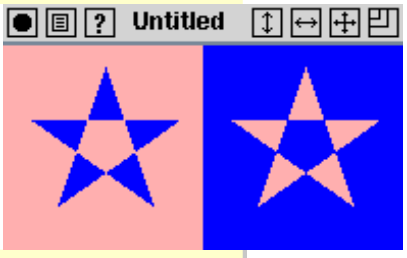
Det skall via konstruktorn vara möjligt att sätta stjärnas färg och bakgrundsfärgen.

Implementation:

```
import java.awt.*;
import javax.swing.*;
public class Star extends JPanel {
   private Color foreGroundColor;
   public Star(Color foreGroundColor, Color backGroundColor) {
      this.foreGroundColor = foreGroundColor;
      setBackground(backGroundColor);
   }//konstruktor
   public void paintComponent(Graphics pen) {
      super.paintComponent(pen);
      pen.setColor(foreGroundColor);
      Polygon star = new Polygon();
      star.addPoint(50, 10);
      star.addPoint(74, 82);
      star.addPoint(14, 38);
      star.addPoint(86, 38);
      star.addPoint(26, 82);
      star.addPoint(50, 10);
      pen.fillPolygon(star);
   }//paintComponent
}//Star
```

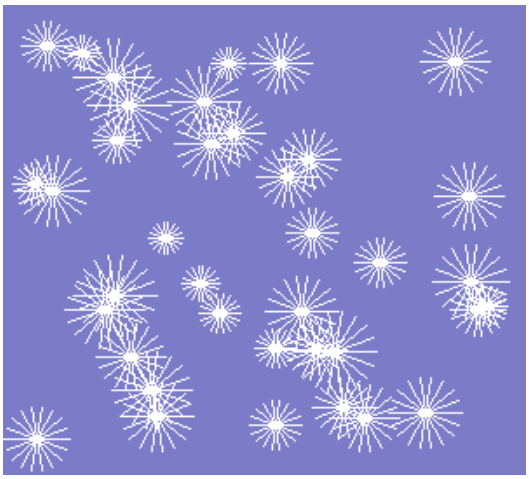
Implementation: Ett huvudprogram som skapar ett fönster med två objekt av klassen `Star`.

```
import java.awt.*;
import javax.swing.*;
public class ShowStars {
   public static void main(String[] args) {
      JFrame window = new JFrame();
      Star s1 = new Star(Color.BLUE, Color.PINK);
      Star s2 = new Star(Color.PINK, Color.BLUE);
      window.setLayout(new GridLayout(1,2));
      window.add(s1);
      window.add(s2);
      window.setLocation(50,50);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setSize(200, 130);
      window.setVisible(true);
   }//main
}//ShowStars
```



Exempel

Skriv ett program som ritar ut ett fönster som visar ett snöfall enligt figuren nedan. Snöflingornas antal,storlek och placering slumpas fram.



Implementation:

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class SnowFall extends JPanel {
   private static Random slump = new Random();

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

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

```
   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
```

Implementation: Ett huvudprogram som skapar ett fönster med ett objekt av klassen SnowFa11.

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