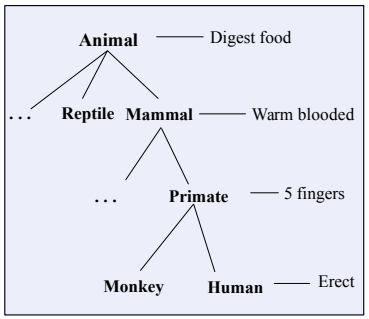
Föreläsning 9

Arv
Grafiska komponenter

Arv

Vi människor använder klassificering för att organisera vår tillvaro.

Klassificering innebär att sammanföra likartade objekt inom en domän i olika delgrupper (klasser).



Vid klassificering erhålls en hierarki,

Ett objekt som tillhör en viss nivå i hierarkin har alla egenskaper som objekten på högre nivå i hierarkin har (*dessa egenskaper har redan definierats och ärvs*).

Varje nivå i hierarkin tillför nya egenskaper hos objekten.

Ju längre ner i hierarkin, ju mer *specialiserade* objekt.

Ett objekt som tillhör en viss nivå i hierarkin tillhör också alla högre nivåer (*en människa är en primat, en primat är ett däggdjur, ett däggdjur är ett djur*).

Arv

I Java finns två konstruktioner för att bygga arvshierarkier:

- implementationsarv (**extends**)
- specifikationsarv (**implements**)

Vid implementationsarv ärvs kod från en superklass.

Vid specifikationsarv ärvs ingen kod utan enbart gränssnittet (dvs vilka publika metoder som klassen skall innehålla).

Implementationsarv

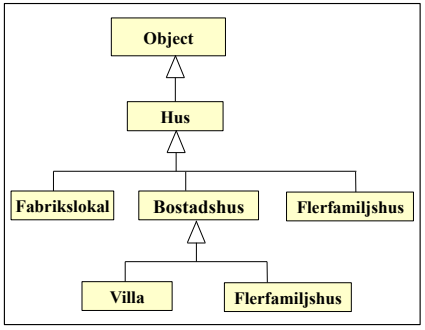
- Implementationsarv är en grundläggande objektorienterad teknik för att *organisera* och *återanvända* klasser.
- Med implementationsarv kan man definiera en klass utgående *från en redan existerande klass*.
- Den nya klassen återanvänder den gamla klassens definitioner, men *utökar* den med nya definitioner.
- Den klass som ärver kallas *subklass*.
- Den klass som en subklass ärver ifrån kallas för *superklass*.
- Subklassen har en *är-relation* till sin superklass, "en bil *är* ett fordon", "en politiker *är* en person", "en elefant *är* ett djur", "en villa *är* ett hus".
- En subklass kan vara en superklass till en annan klass, dvs det går att bygga en *arvshierarki*.
- När man skapar en arvshierarki skall gemensamma egenskaper och beteenden ligga så "långt upp" som möjligt i hierarkin.
- Alla klasser är subklasser till klassen Object.

Implementationsarv

- Alla klasser (utom Object) är i grunden utökningar av klassen Object.
- När en klass utökar en annan så ärver den alla dess fält och metoder.
- En subklass utökar superklassen
- Klasser är del i en klasshierarki

java.lang
Class Integer

java.lang.Object
  └ java.lang.Number
    └ java.lang.Integer



Klassen Rectangle

På en tidigare föreläsning implementerade vi en klass Rectangle för att avbilda rektanglar.

```
public class Rectangle {
  private double width;      //instansvariabel
  private double height;     //instansvariabel

  public Rectangle() {
    width = 0;
    height = 0;
  }//constructor

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }//constructor

  public double getWidth() {
    return width;
  }//getWidth

  public double getHeight() {
    return height;
  }//getHeight

  public void setWidth(double width) {
    this.width = width;
  } //setWidth

  public void setHeight(double height) {
    this.height = height;
  } //setHeight

  public double getArea() {
    return height * width;
  }//getArea

  public double getPerimeter() {
    return 2 * (height + width);
  } //getPerimeter

  public String toString() {
    return "Width = " + width + "\nHeight = " + height;
  }//toString
} //Rectangle
```

Arv

I klassen Rectangle beskrivs en rektangel med en höjd och en bredd.

I en tillämpning där vi hanterar rektanglar, behöver vi förutom höjden och bredden på rektanglarna även handha rektanglarnas färger.

Klassen Rectangle uppfyller således inte våra behov, utan vi behöver en ny klass ColoredRectangle, som avbildar en rektangel mha höjd, bredd och färg.

Klassen ColoredRectangle

Utveckla klassen ColoredRectangle från "scratch".

```
import java.awt.Color;
public class ColoredRectangle {
  private double width;
  private double height;
  private Color color;

  public ColoredRectangle(double width, double height, Color color) {
    this.width = width;
    this.height = height;
    this.color = color;
  }//constructor

  public ColoredRectangle() {
    width = 0;
    height = 0;
    color = Color.RED;
  }//constructor

  public double getWidth() {
    return width;
  }//getWidth

  public double getHeight() {
    return height;
  }//getHeight

  public Color getColor() {
    return color;
  } //getColor

  public void setWidth(double width) {
    this.width = width;
  } //setWidth

  public void setHeight(double height) {
    this.height = height;
  } //setHeight

  public void setColort(Color color) {
    this.color = color;
  } //setColor

  public double getArea() {
    return height * width;
  }//getArea

  public double getPerimeter() {
    return 2 * (height + width);
  } //getPerimeter

  public String toString() {
    return "Width = " + width + "\nHeight = " + height
           + "\nColor = " + color;
  }//toString
}//ColoredRectangle
```

Implementationsarv

I klassen ColoredRectangle finns mycket av den kod som också finns i klassen Rectangle:

- båda klasserna har instansvariablerna

  private double widh;

  private double height;

- båda klasserna har metoderna

  public double getWidth()
  public double getHeight()
  public void setWidth(**double** width)
  public void getHeight(**double** height)
  public double getArea()
  public double getPerimeter()
  public String toString()

Skulle vi då kunnat utnyttjat koden som finns i klassen Rectangle när vi konstruerade klassen ColoredRectangle utan att skriva om koden igen så som vi gjorde i ovanstående implementation av klassen ovan?

Svar: JA!! Vi skulle ha kunnat utnyttjat arvsmekanismen som finns i Java.

Klassen ColoredRectangle

```
import java.awt.Color;
public class ColoredRectangle extends Rectangle {
   private Color color;

   public ColoredRectangle(double width, double height, Color color) {
      super(width, height);
      this.color = color;
   }//constructor

   public ColoredRectangle() {
      super();
      color = Color.RED;
   }//constructor

   public Color getColor() {
      return color;
   } //getColor

   public void setColort(Color color) {
      this.color = color;
   } //setColor

   @Override
   public String toString() {
      return super.toString() +  "\nColor = " + color;
   }//toString
}//ColoredRectangle
```

Metoderna
  double getWidth()
  double getHeight()
  void setWidth(**double** width)
  void getHeight(**double** height)
  double getArea()
  double getPerimeter()
ärvs från klassen Rectangle.

Konstruktorer ärvs inte. Konstruktorerna använder sig av konstruktorerna i superklassen.

Metoden toString överskyggas och nyttjar metoden toString i superklassen.

Implementationsarv: super

En subklass kan referera till sin direkta superklass med **super**

- används i subklassens konstruktorer för att anropa superklassens konstruktorer

  super(width, height);

  Om subklassens konstruktor gör anrop till superklassens konstruktor skall detta anrop ligga som första sats i konstruktorn.

  Gör inte subklassens konstruktor något anrop till superklassens konstruktor sker automatiskt anrop till superklassens default-konstruktor (konstruktorn utan parametrar):

  - saknar superklassen helt konstruktorer propagerar anropet uppåt i klasshiearkin
  - saknar superklassen default-konstruktor men har någon annan konstruktor uppkommer ett kompileringsfel.

Implementationsarv

- För att ange att en subklass skall ärva från en annan klass används det reserverade ordet **extends**.

  public class ColoredRectangle **extends** Rectangle

- En subklass kan lägga till tillståndsvariabler och metoder:

  private Color color

- En subklass kan lägga till metoder:

  public Color getColor()
  public void setColor(Color color)

- En metod i subklassen kan omdefiniera, eller *överskugga*, en metod i superklassen:

```
public String toString() {
  return super.toString() +  "\nColor = " + color;
} //roll
```

- Superklassens parameterlösa konstruktor anropas automatiskt innan satserna i den egna konstruktorn utförs om inget explicit anrop till en konstruktor görs.

Klassen Die

Vi har tidigare använt en klass Die för att avbilda en 6-sidig tärning. Klassen har följande utseende:

```
import java.util.Random;
public class Die {
   private int dots;
   private static Random dieRandom  = new Random();

   public Die() {
      roll();
   }//constructor

   public Die(int dots) {
      if (dots < 1 || dots > 6)
         throw new IllegalArgumentException();
      else
         this.dots = dots;
   }//constructor

   public int getValue() {
      return dots;
   }//getDots

   public void roll() {
      dots = dieRandom.nextInt(6) + 1;
   }//roll
}//Die
```

Klassen NoRepeatDie

Utgå från klassen Die och använd implementationsarv för att skapa skapa en ny klass NoRepeatDie för att avbilda en tärning som aldrig ger samma värde för två på varandra följande kast.

```
public class NoRepeatDie extends Die {
   public NoRepeatDie() {
      super();
   }//constructor

   public NoRepeatDie(int dots) {
      super(dots);
   }//constructor

   @Override
   public void roll() {
      int oldValue = getValue();
      do {
         super.roll();
      } while (getValue() == oldValue);
   }//roll
}//NoRepeatDie
```

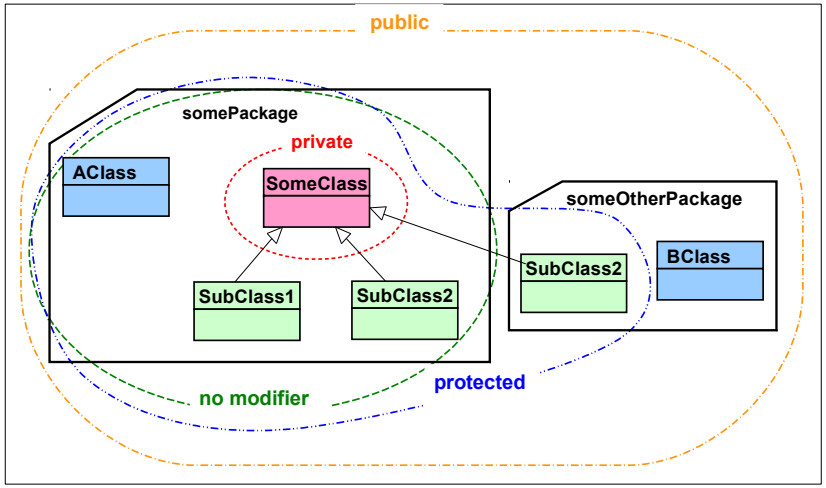
Synlighetsmodifierare

När man säger att en subklass ärver all data och alla metoder från sin superklass, betyder nödvändigtvis inte detta att subklassen direkt kan få access till all data och alla metoder i superklassen.

I Java finns fyra olika synlighetsmodifierare:

public	tillåter access för alla andra klasser.
protected	tillåter access för alla andra klasser i samma paket och för subklasser i andra paket.
utelämnad	tillåter access för alla andra klasser i samma paket.
private	tillåter access endast inom klassen själv.

Räckvidden hos synlighetsmodifierarna



Klassen CheaterDie

```
import java.util.Random;
public class Die {
   private int dots;
   private static Random dieRandom  = new Random();
   . . .
}//Die
```

Variablerna är private! Okända i subklasen

```
import java.util.Random;
public class CheaterDie extends Die {
   private static Random random  = new Random();
   public CheaterDie() {
      super();
   }//constructor

   public void roll() {
     if (random.nextInt(2) == 1)
       super.roll();
     else {
       do {
         super.roll();
       } while (getValue() != 6);
     }
   }//roll
}//CheaterDie
```

Implementation av klassen CheaterDie, som avbildar en 6-sidig tärning med egenskapen att sidan 6 kommer upp med sannolikheten 50 procent.

Klassen CheaterDie

```
import java.util.Random;
public class Die {
   protected int dots;
   protected static Random dieRandom  = new Random();
   . . .
}//Die
```

Variablerna är protected! Kända i subklasen

```
public class CheaterDie extends Die {
   public CheaterDie() {
      super();
   }//constructor

   public void roll() {
     if (randomDie.nextInt(2) == 1)
       dots = dieRandom.nextInt(6) + 1;
     else
        dots = 6;
   }//roll
}//CheaterDie
```

Implementation av klassen CheaterDie, som avbildar en 6-sidig tärning med egenskapen att sidan 6 kommer upp med sannolikheten 50 procent.

Arv

En referensvariabel som är av typen ***"referens till C" får referera till objekt som är av klassen C eller till objekt som är subklasser till C***.

Operatorn **instanceof** kan användas för att avgöra om ett objekt är av en viss klass.

```
public class TestDices {
   public static void main(String[] arg) {
      Die t1 = new Die();
      Die t2 = new NoRepeatDie();
      Die t3 = new CheaterDie();
      t1.roll();
      t2.roll();
      t3.roll();
      t1 = t2;
      if (t1 instanceof NoRepeatDie)
         System.out.println("Detta är en specialtärning");
      if (t3 instanceof CheaterDie)
         System.out.println("Detta är en fusktärning!!");
   }//main
}//TestDices
```

Arv: Sammanfattning

- En subklass kan lägga till tillståndsvariabler och metoder.
- Superklassens parameterlösa konstruktor anropas automatiskt innan satserna i den egna konstruktorn utförs, om inget explicit anrop till en konstruktor görs.
- Den egna klassen refereras med **this** och superklassen refereras med **super**.
- Alla klasser betraktas som subklasser till klassen Object.
- En metod i subklassen kan *omdefiniera* - eller *överskugga* - en metod i superklassen.
- Företeelsen med överskuggade metoder kallas för *polymorfism* (mångformighet).
- Vid överskuggade metoder är det objektets klass som avgör vilken metod som avses. Mekanismen som handhar att rätt metod väljs kallas för *dynamisk bindning*.

Arv: Sammanfattning

- Saknas en metod i en klass används i stället första motsvarande metod högre upp i klasshierarkin.
- Vid överskuggning av en metod får tillgängligheten (**public**, **protected**, **private**) hos den nya metoden inte vara lägre än tillgängligheten hos den överskuggade metoden.
- Vid överskuggning av en metod måste den nya metoden ha samma returtyp som originalet. (Signaturen hos metoderna är naturligtvis också lika, annars är det inte överskuggning utan överlagring.)
- En subklass når inte attribut i superklassen som är **private**.
- En referensvariabel som är av typen "referens till *C*" får referera till objekt som är av klassen *C* eller till objekt som är subklasser till *C*.
- Operatorn **instanceof** kan användas för att avgöra om ett objekt är av en viss klass.

Grafiska användargränssnitt (GUIs)

- Användningen av fönstersystem har gjort att tekniken för att göra in- och utmatning till program har förändrats.
- Istället för textbaserad inmatning via tangentbordet sker inmatningen med hjälp av grafiska användargränssnitt, dvs användning av dialogboxar, ifyllande av blanketter, klicka med musen osv.
- Istället för textbaserad utmatning sker utmatningen ofta i form av grafiska presentationer.

Grafiska användargränssnitt (GUIs)

Användning av grafiska användargränssnitt förändrar principerna för att skriva program.

- I traditionella textbaserade program gäller att
  - programmet styr vilka programsegment som skall exekveras och i vilken ordning detta skall ske
  - inläsningen av indata är programstyrd, dvs programmet avgör när indata skall läsas och vilken indatasekvens som skall läsas.
- GUI-program är "händelsestyrda".
- För händelsestyrda program gäller att
  - indata som ges till programmet (dvs en händelse som inträffar i det grafiska användargränssnittet) bestämmer vilka programsegment som exekveras.

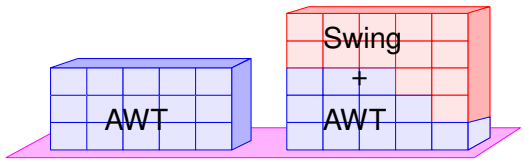
Händelsestyrt program

Grafiska användargränssnitt (GUIs)

Java har bra stöd för grafiska användargränssnitt tack vare standardbiblioteken AWT (*Abstract Windowing Toolkit*) och Swing.

I de första versionerna av Java fanns enbart AWT.

Swing är både ett komplement och en ersättning för AWT. Swing bygger på AWT och har definierat om eller byggt ut vissa klasser, medan andra klasser används direkt som de är i AWT.



AWT finns i paketet java.awt och Swing finns i paketet javax.swing.

Grafiska användargränssnitt (GUIs)

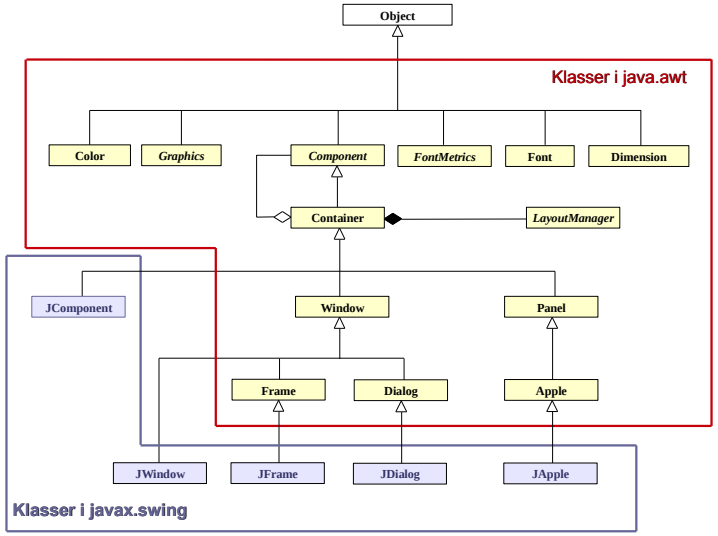
Swing är en verktygslåda som är till hjälp när man skriver program som har ett fönsterbaserat användargränssnitt. Swing består av många delar och en grov uppdelning kan göras på följande sätt:

- Grundläggande grafik: Linjer, cirklar, rektanglar etc.
- Grundläggande komponenter: Textfält, knappar, listor etc.
- Behållare: Komponenter som kan innehålla andra komponenter.
- Layouthantering: Bestämmer hur olika komponenter skall placeras ut i förhållande till varandra.
- Händelsehantering: Hur olika händelser skall hanteras.
- Avancerade komponenter: Fildialoger, tabeller, träd, editorer etc.
- Look-and-feel: Övergripande inställningar som gäller för alla komponenter.

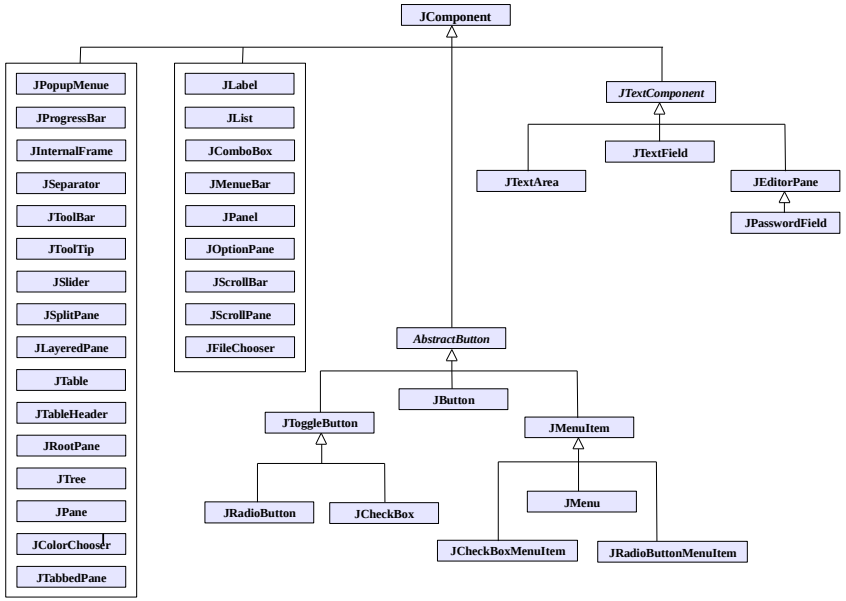
Grafiska användargränssnitt (GUIs)

- De flesta fönstermiljöprogram har ett användargränssnitt som är uppbyggt av komponenter som var och en för sig har en viss uppgift.
- En välskriven komponent har en väldefinierad och lättolkad uppgift, och är lätt att återanvända och kombinera med andra komponenter.
- Komponenter kan vara väldigt enkla, t.ex en knapp, eller mer komplicerade, som en fildialog.
- Det är viktigt att komponenter har ett konsekvent programmerings-gränssnitt, både för att förenkla programmeringen av de enskilda komponenterna, men också för att det skall vara enkelt att kombinera dem.

Komponentklasser i Swing



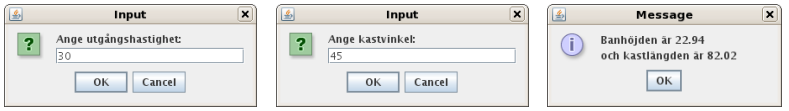
Subklasser till JComponent



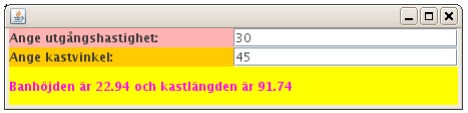
Grafiska användargränssnitt (GUIs)

I laboration 1 skrev ni ett program som läste in en utgångshastighet och en kastvinkel, och från dessa värden beräknade banhöjd och kastlängd.

I laborationen gjordes inläsningen i två separata dialogrutor och resultatet presenterades i en tredje dialogruta:



Ett mer överskådligt och lättanvänt (samt möjligen snyggare) användargränssnitt skulle kunna se ut enligt nedan:



Första delen i att bygga detta gränssnitt (se nästa sida vad vi får):

```
import javax.swing.*;
import java.awt.*;
public class Shoot1 extends JFrame {
   private JTextField hField = new JTextField(20);
   private JTextField vField = new JTextField(20);
   private JLabel resultLabel = new JLabel();
   public Shoot1() {
      JLabel hLabel = new JLabel("Ange utgångshastighet: ");
      hLabel.setBackground(Color.PINK);
      hLabel.setOpaque(true);
      JLabel vLabel = new JLabel("Ange kastvinkel: ");
      vLabel.setBackground(Color.ORANGE);
      vLabel.setOpaque(true);
      JPanel inputPanel = new JPanel();
      inputPanel.setLayout(new GridLayout(2,2));
      inputPanel.add(hLabel);
      inputPanel.add(hField);
      inputPanel.add(vLabel);
      inputPanel.add(vField);
```

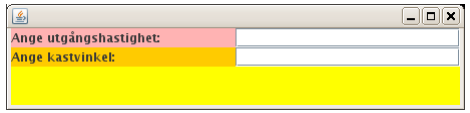
Observera att programmet inte är komplett, eftersom vi ännu inte har några lyssnare i programmet!

```
      resultLabel.setBackground(Color.YELLOW);
      resultLabel.setForeground(Color.MAGENTA);
      resultLabel.setOpaque(true);
      setLayout(new GridLayout(2,1));
      add(inputPanel);
      add(resultLabel);
      pack();
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
   }//constructor
} //Shoot1
```

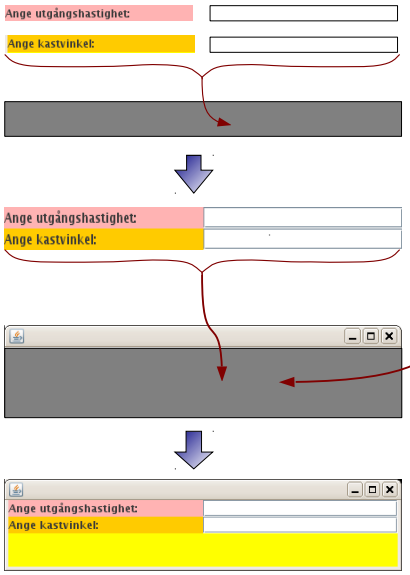
Om vi nu skriver ett huvudprogram som skapar ett objekt av klassen `Kast1` enligt:

```
import javax.swing.*;
public class Main {
   public static void main(String[] args) {
      JFrame w = new Shoot1();
   }//main
}//Main
```

får vi ett fönster med följande utseende:



Fönstret är uppbyggt på följande sätt:



Komponenterna hLabel, hField, vLabel och vField

Panelen inputPanel.

Komponenten resultatsLabel.

Själva fönstret, det ursprungliga JFrame-objektet som ärvs.

Slutligt utseende av det JFrame-objektet när alla komponenter är utplacerade.

Klassen javax.swing.JComponent

JComponent är basklassen för alla grafiska komponenter i Swing (med ett par undantag).

- JComponent är en abstrakt klass som innehåller metoder som är tillämpliga på alla grafiska komponenter.
- JComponent utökar java.awt.Container som är en AWT-klass för komponenter som kan innehålla andra komponenter. Det innebär att (i princip) alla Swing-komponenter kan innehålla andra komponenter.
- JComponent utökar java.awt.Container, vilket innebär att allting som går att göra med en AWT-komponent går också att göra med en Swing-komponent.

Klassen javax.swing.JComponent

Metoderna som kan användas på alla Swing-komponenter kan delas upp enligt vad de har att göra med:

- Komponentens utseende (färg, font, markör etc)
- Komponentens placering/storlek
- Ritning av komponenten
- Komponentens tillstånd (om den är visad/gömd, aktiverad/avaktiverad etc)
- Komponentens fokus (kan den fokuseras? vad är nästa fokuskomponent?)
- Om komponenten skall ha en ram eller inte.
- Om komponenten skall ha en beskrivande text som visas när muspekaren är över komponenten.

Klassen JLabel

Ett objekt av klassen JLabel visar en enkel text. Textens innehåll kan ändras av programmet, däremot kan den som använder programmet inte ändra texten.

I ett JLabel-objekt kan texten kombineras med en bild.

Det finns metoder för att t.ex. kunna ändra texten och bilden i ett JLabel-objekt, göra objektet synligt eller osynligt samt kunna placera texten/bilden på olika ställen på objektet. Vidare finns metoder för att sätta bakgrunds- och förgrundsfärg på objektet, förse objektet med olika slag av ramar samt ange utseendet på texten.

Vissa av dessa metoder finns i klassen JLabel, andra metoder ärvs från superklasserna JComponent och Container.

Exempel på metoder i klassen JLabel

```
JLabel label1 = new JLabel();
JLabel label2 = new JLabel("Some text");
JLabel label3 = new JLabel("A piture and some text",
                           new ImageIcon("work.gif"), JLabel.CENTER);
label1.setText("Detta är en text");
label1.setIcon(new ImageIcon("figur.gif"));
label1.setVisible(false);
label2.setBackground(Color.YELLOW);
label2.setForeground(Color.BLUE);
label2.setFont(new Font("SansSerif", Font.BOLD,14));
label2.setBorder(new LineBorder(Color.RED, 5));
label2.setOpaque(true);
label3.setBackground(Color.PINK);
label3.setForeground(Color.RED);
label3.setFont(new Font("Serif", Font.ITALIC, 20));
label3.setBorder(new TitledBorder("Rubrik på ramen"));
label3.setOpaque(true);
String str = label2.getText();
Icon bild = label2.getIcon();
```

För JLabel-objekt gäller som standard att de är genomskinliga. Därför syns t.ex. inte bakgrundsfärgen. För att bakgrundsfärgen skall synas måste vi anropa metoden setOpaque(**true**).

Klassen JButton

Klassen JButton beskriver en knapp som användaren kan klicka på.

Ett objekt av klassen JButton kan innehålla en text och/eller en bild.

Det finns metoder för att t.ex. kunna ändra knappens utseende när man trycker på den, göra knappen synlig eller osynlig, göra knappen åtkomlig eller icke åtkomlig samt för att placera texten/bilden på olika ställen på knappen.

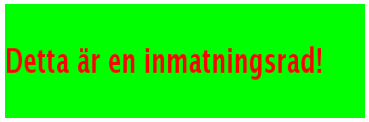
Vissa av dessa metoder finns i klassen JButton, andra metoder ärvs från superklasserna JComponent och Container.

Exempel på metoder för klassen JButton

```
JButton button1 = new JButton();
JButton button2 = new JButton("Some text");
JButton button3 = new JButton("A piture and some text",  new ImageIcon("work.gif"));
button1.setText("Detta är en text");
button1.setIcon(new ImageIcon("figur.gif"));
button1.addActionListener(this);
button1.setVisible(false);
button2.setBackground(Color.YELLOW);
button2.setForeground(Color.BLUE);
button2.setFont(new Font("SansSerif", Font.BOLD,14));
button2.setBorder(new LineBorder(Color.RED, 5));
button2.addActionListener(this);
button2.setEnabled(false);
button3.setBackground(Color.PINK);
button3.setForeground(Color.RED);
button3.setFont(new Font("Serif", Font.ITALIC, 20));
button3.setBorder(new TitledBorder("Rubrik på ramen"));
button3.addActionListener(this);
String str = button2.getText();
Icon bild = button2.getIcon();
```

Klassen JTextField och JTextArea

Klassen JTextField används för att skapa objekt för kunna göra enklare inläsning av data. I ett JTextField-objekt sker inmatningen från en inmatningsrad.



Klassen JTextArea används för att skapa objekt som används vid mer avancerad inmatning där indata behöver läsas från flera inmatningsrader.

Det finns metoder för att t.ex. läsa den text som finns i ett ett JTextField-objekt och ett JTextArea-objekt, göra objektet synligt eller osynligt och ändra storleken på objektet. Vidare finns metoder för att sätta bakgrunds- och förgrundsfärg på objektet, förse objektet med olika slag av ramar samt ange utseendet på texten.

Vissa av dessa metoder finns i klassen JTextField (resp JTextArea), andra metoder ärvs från superklasserna JComponent och Container.

Exempel på metoder för klassen JTextField

```
JTextField textF1 = new JTextField();
JTextField textF2 = new JTextField(20);
JTextField textF3 = new JTextField("Some text");
JTextField textF4 = new JTextField("Some text", 20);
textF1.setText("Detta är en text");
textF1.addActionListener(this);
textF1.setVisible(false);
textF2.setBackground(Color.YELLOW);
textF2.setForeground(Color.BLUE);
textF2.setFont(new Font("SansSerif", Font.BOLD,14));
textF2.setBorder(new LineBorder(Color.RED, 5));
textF2.addActionListener(this);
textF2.setEnabled(false);
textF3.setBackground(Color.PINK);
textF3.setForeground(Color.RED);
textF2.setFont(new Font("Serif", Font.ITALIC, 20));
textF3.setBorder(new TitledBorder("Rubrik på ramen"));
textF3.addActionListener(this);
String str1 = textF1.getText();
String str2 = textF1.getSelectedText();
```

Klassen JPanel

Det finns en särskilt viktig sorts komponenter, nämligen de som kan innehålla andra komponenter. Dessa komponenter används för att kunna gruppera komponenter till mer komplicerade strukturer. Komponenter som kan innehålla andra komponenter kallas ofta för behållare (*eng containers)* och är subklasser till klassen Container.

Den vanligaste komponenten för att gruppera andra komponenter i är JPanel. Den har alla de egenskaper som vanliga komponenter har, men dessutom kan man säga åt den att innehålla andra komponenter med metoden `add`:

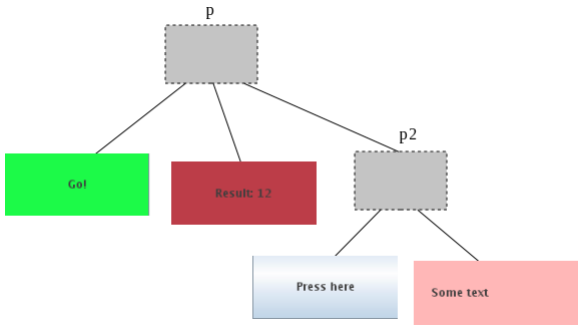
```
JPanel p = new JPanel();
p.add(annan komponent);
```

Klassen JPanel

En `JPanel` kan innehålla vilka andra komponenter som helst, även andra `JPanel`-komponenter. På så vis bildas det en trädstruktur med komponenter inuti andra komponenter osv.

Hur de olika komponenterna läggs ut bestäms av vilken `LayoutManager` som används.

```
JPanel p = new JPanel();
JPanel p2 = new JPanel();
JButton go = new JButton();
JButton press = new JButton();
JLabel result = new JLabel();
JLabel text = new Jlabel();
. . .
p.add(go);
p.add(result);
p.add(p2);
p2.add(press);
p2.add(text);
. . .
```



Exempel på metoder för klassen JPanel

```
JPanel panel1 = new JPanel();
JPanel panel2 = new JPanel(new FlowLayout(10));
panel1.setLayout(new GridLayout(1, 2, 10, 10));
panel1.setVisible(true);
panel1.setBackground(Color.YELLOW);
panel1.setBorder(new LineBorder(Color.RED, 5));
panel2.setBackground(Color.PINK);
panel2.setBorder(new TitledBorder("Rubrik på ramen"));
panel1.add(new JButton("Go!"));
panel1.add(new JLabel("Resultat:"));
panel1.add(panel2);
panel2.add(new JLabel("Some text"));
panel2.add(new JButton("Press Here!"));
panel1.setVisible(true);
panel2.setSize(100,200);
int w = panel2.getWidth();
int h = panel2.getHeight();
```

Klassen JFrame

JFrame är huvudfönstret som alla andra komponenter ligger i. Fönstret har flera olika delar, som t ex en menyrad, fönstrets kant, huvudytan osv. Huvudytan kommer man åt med metoden getContentPane som returnerar en Container.

```
import java.awt.*;
import javax.swing.*;
public class TestFrame extends JFrame {
   public TestFrame() {
      setLayout(new GridLayout(1, 2, 20,20));
      getContentPane().setBackground(Color.PINK);
      add(new JButton("Press here"));
      add(new JLabel("Some text"));
      setDefaultCloseOperation(EXIT_ON_CLOSE);
   }//konstruktor
   public static void main(String[] arg) {
      JFrame tf = new TestFrame();
      tf.setSize(350, 100);
      tf.setVisible(true);
   }//main
}//TestFrame
```

Utelämnas getContentPane blir inte fönstret rosa