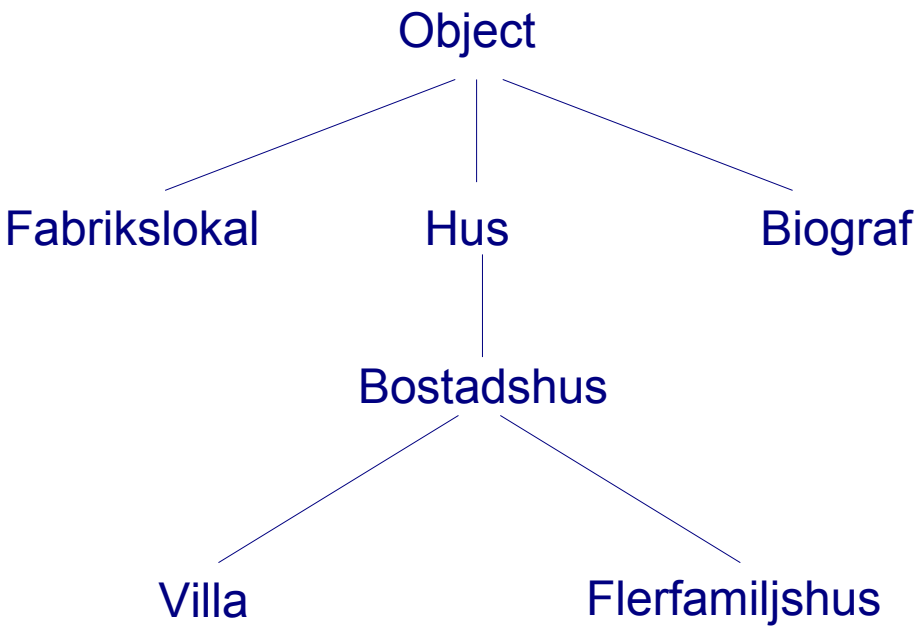
# Föreläsning 9

## Arv
## Grafiska komponenter

# Arv

- Arv är en grundläggande objektorienterad teknik för att organisera och återanvända klasser.
- Med arv 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.
- Arv kan t.ex användas för att modifiera klasser som redan finns (t.ex i Javas API).
- 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`.

# Arvshierarki



# Klassen `Dice`

Vi har i ett flertal exempel använt en klass `Dice` för att avbilda en 6-sidig tärning. Klassen har följande utseende:

```
import java.util.*;
public class Dice {
   private int dots;
   private static Random diceRandom  = new Random();
   public Dice() {
      roll();
   }// konstruktor
   public Dice(int dots) {
      if (dots > 6 || dots < 2)
         roll();
      else
         this.dots = dots;
   } //konstruktor
   public int getDots() {
      return dots;
   } //getDots
   public void roll() {
      dots = diceRandom.nextInt(6) + 1;
   } //roll
} //Dice
```

# Klassen `SpecialDice`

På en av era laborationsuppgifter hade ni i uppgift att skriva en klass `SpecialDice` för att avbilda en tärning med ett godtyckligt antal sidor. Denna klass fick ett utseende enligt nedan:

```
import java.util.*;
public class SpecialDice {
   private int dots;
   private int nrOfSides;
   private static Random diceRandom  = new Random();
   public SpecialDice() {
      nrOfSides = 6;
      roll();
   }//konstruktor
   public SpecialDice(int nrOfSides) {
      if (nrOfSides >= 2 )
         this.nrOfSides = nrOfSides;
      else
         this.nrOfSides = 6;
      roll();
   }//konstruktor
   public int getSides() {
      return nrOfSides;
   }//getSides
   public int getDots() {
      return dots;
   } //geDots
   public void roll() {
      dots = diceRandom.nextInt(nrOfSides) + 1;
   } //roll
} //SpecialDice
```

# Arv

I klassen `Special1Dice` finns mycket av den kod som också finns i klassen `Dice`:

- båda klasserna har en instansvariabel

```
private int dots;
```

som används för lagra vilken sida av tärningen som ligger upp.

- båda klasserna har en klassvariabel

```
private static Random diceRandom  = new Random();
```

som används för att slumpa fram vilken sida som kommer upp när tärningen kastas.

- båda klasserna har en metod

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

som returnerar den sida på tärningen som ligger upp.

Skulle vi då kunnat utnyttjat koden som finns i klassen `Dice` när vi konstruerade klassen `Special1Dice` utan att skriva om koden igen så som vi gjorde vår implementation av klassen ovan?

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

# Arv

För att utnyttja arvsmekanismen och göra klassen `SpecialDice` till en subklass av klassen `Dice` måste vi göra en liten men betydelsefull ändring i klassen `Dice`, nämligen att ändra synlighetsmodifieraren på instansvariabeln `dots` och klassvariabeln `diceRandom` från **private** till **protected**.


```
import java.util.*;
public class Dice {
   protected int dots;
   protected static Random diceRandom  = new Random();
   public Dice() {
      roll();
   }// konstruktor
   public Dice(int dots) {
      if (dots > 6 || dots < 2)
         roll();
      else
         this.dots = dots;
   } //konstruktor
   public int getDots() {
      return dots;
   } //getDots
   public void roll() {
      dots = diceRandom.nextInt(6) + 1;
   } //roll
} //Dice
```

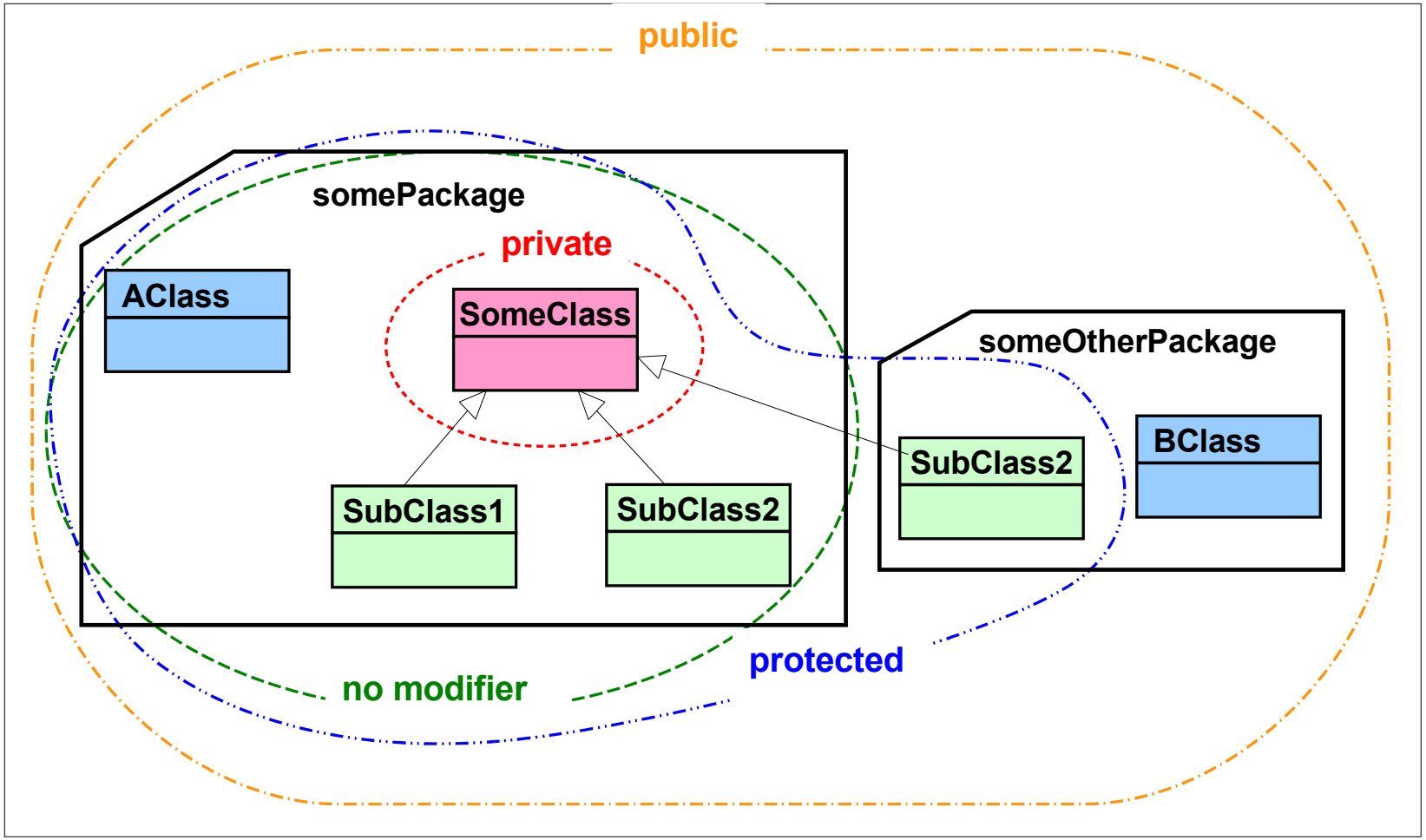
# Synlighetsmodifierare

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



# SpecialDice som subklass till Dice

Om vi gör klassen SpecialDice till en subklass av klassen Dice får SpecialDice följande utseende:

```
public class SpecialDice extends Dice {
   private int nrOfSides;
   public SpecialDice() {
      nrOfSides = 6;
      roll();
   }//konstruktor
   public SpecialDice(int nrOfSides) {
      if (nrOfSides >= 2)
         this.nrOfSides = nrOfSides;
      else
         nrOfSides = 6;
      roll();
   }// konstruktor
```

```
   public int getSides() {
      return nrOfSides;
   }//getSides
   public void roll() {
      dots = diceRandom.nextInt(nrOfSides) + 1;
   } //roll
} //SpecialDice
```

# Arv

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

```
public class SpecialDice extends Dice
```

- En subklass kan lägga till tillståndsvariabler och metoder:

```
private int nrOfSides
public int getSides()
```

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

```
public void roll() {
    dots = diceRandom.nextInt(nrOfSides) + 1;
} //roll
```

- Superklassens parameterlösa konstruktor anropas automatiskt innan satserna i den egna konstruktorn utförs.

# Arv

Låt oss utgå från klassen `Dice` och skapa ytterligare en subklass `CheaterDice`.

Ett objekt av klassen `CheaterDice` är en vanlig 6-sidig tärning, men har egenskapen att sidan 6 kommer upp med 50 procents sannolikhet.

Det vi behöver göra i klassen `CheaterDice` är således att överskygga metoden `roll()` i superklassen `Dice`, dvs införa en ny metod `roll()` i klassen `CheaterDice`, som då den anropas sätter instansvariabeln `dots` till 6 med 50 procents sannolikhet.

```
public class CheaterDice extends Dice {
   public void roll() {
      if (diceRandom.nextInt(2) == 1)
         dots = 6;
      else
         dots = diceRandom.nextInt(5) + 1;
   } //roll
} //CheaterDice
```

# 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) {
      Dice t1 = new Dice();
      Dice t2 = new SpecialDice(10);
      Dice t3 = new CheaterDice();
      t1.roll();
      t2.roll();
      t3.roll();
      t1 = t2;
      if (t1 instanceof SpecialDice)
         System.out.println("Detta är en specialtärning");
      if (t3 instanceof CheaterDice)
         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.
- 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
- Sekvensen av indata styr exekveringen av programmet

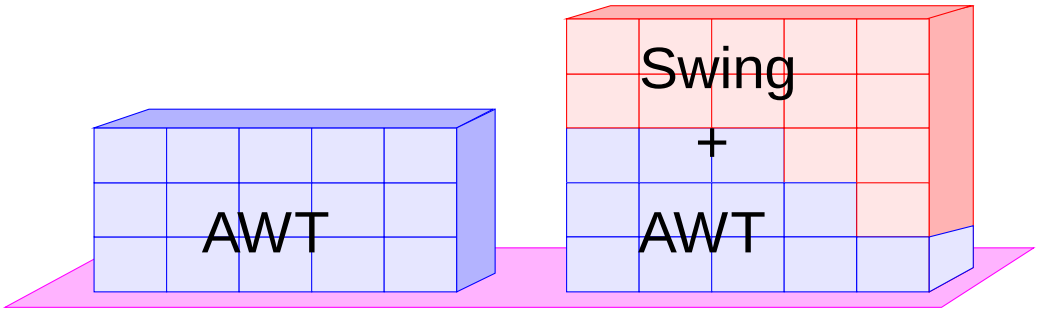
# Händelsestyrt program

# 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)

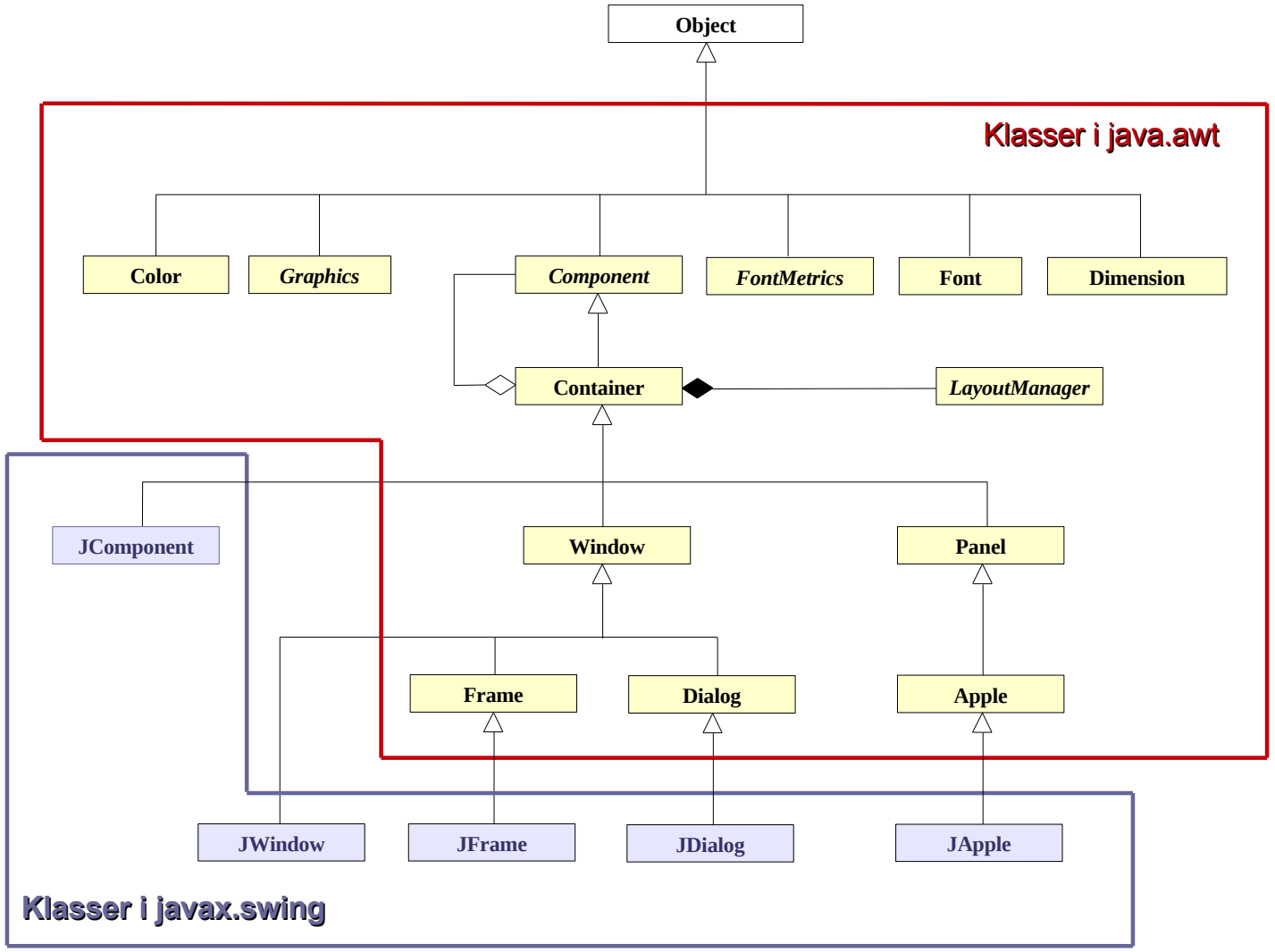
`Swing` är en verktygslåda som är till hjälp när man skriver program som har ett fönsterbaserat användargränssnitt. `Swing` består av många delar och en grov uppdelning kan göras på följande sätt:

- 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.

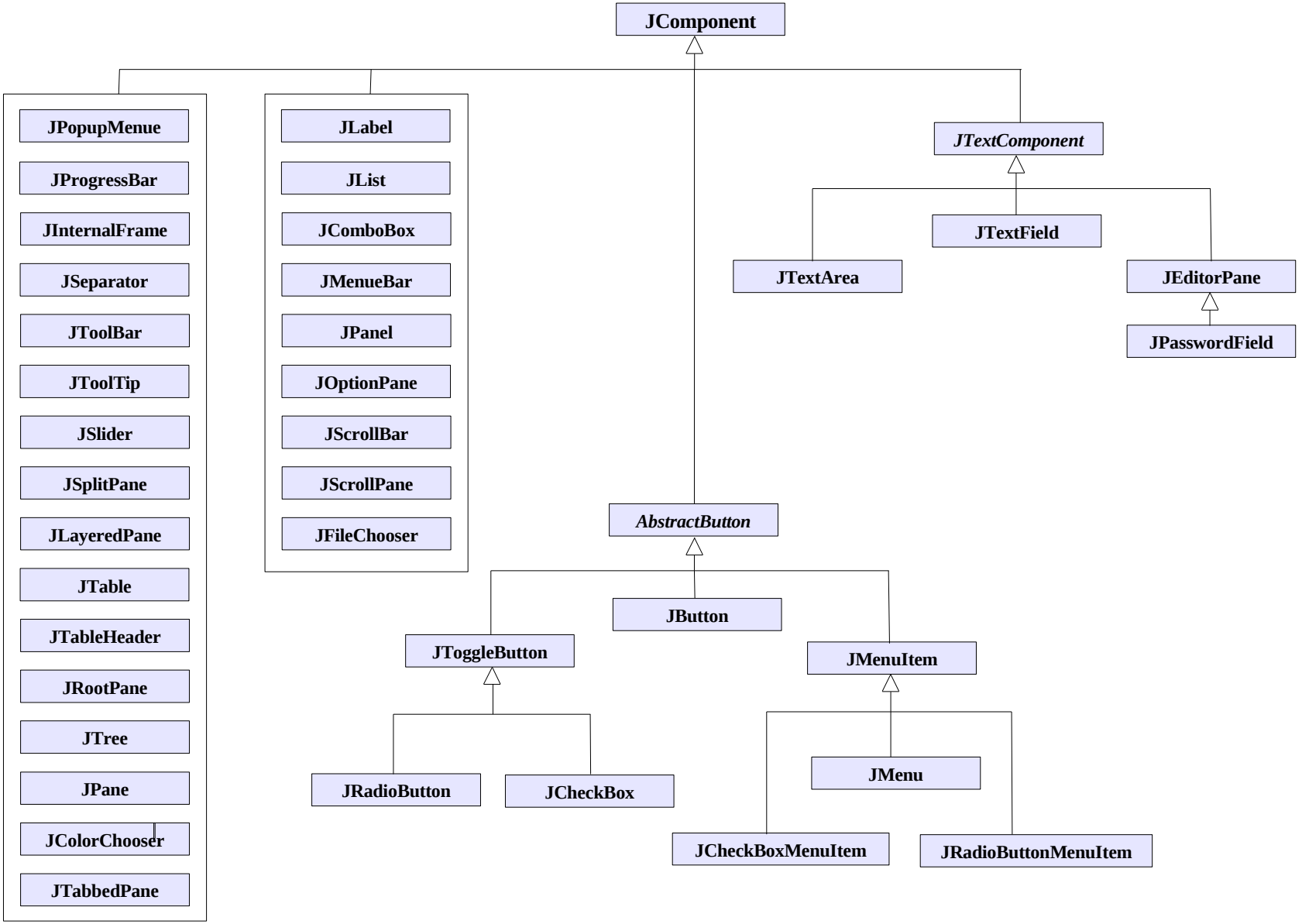
# Grafiska användargränssnitt (GUIs)

- 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



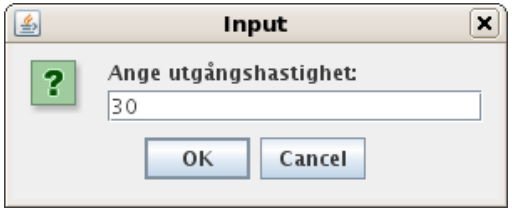
# Subklasser till `JComponent`

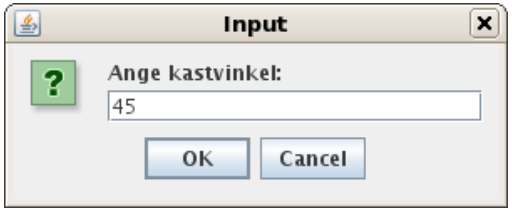


# Grafiska användargränssnitt (GUIs)

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

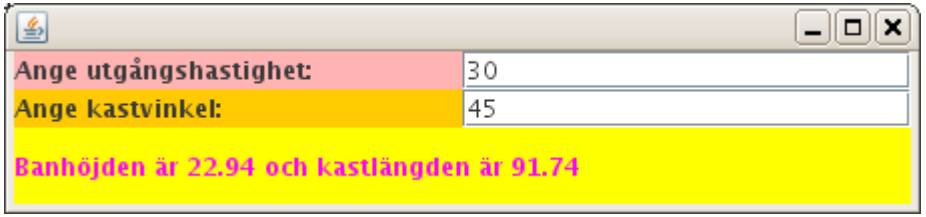
I laborationen gjordes inläsningen i två separata dialogrutor och resultatet presenterades i en tredje dialogruta:





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 Kast1 extends JFrame {
    private JTextField hField = new JTextField(20);
    private JTextField vField = new JTextField(20);
    private JLabel resultatLabel = new JLabel();
    public Kast1() {
        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 frågePanel = new JPanel();
        frågePanel.setLayout(new GridLayout(2,2));
        frågePanel.add(hLabel);
        frågePanel.add(hField);
        frågePanel.add(vLabel);
        frågePanel.add(vField);
```

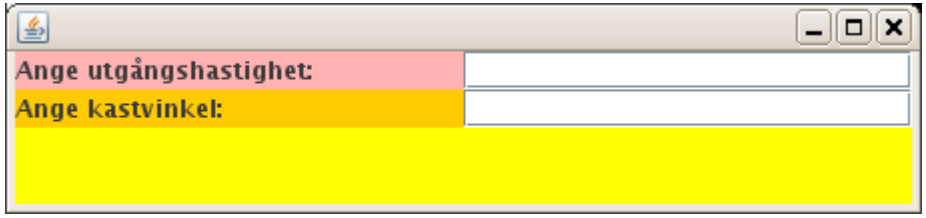
Observera att programmet inte är komplett, eftersom vi ännu inte har några lyssnare i programmet!

```
        resultatLabel.setBackground(Color.YELLOW);
        resultatLabel.setForeground(Color.MAGENTA);
        resultatLabel.setOpaque(true);
        setLayout(new GridLayout(2,1));
        add(frågePanel);
        add(resultatLabel);
        pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }//konstruktor
} //Kast1
```

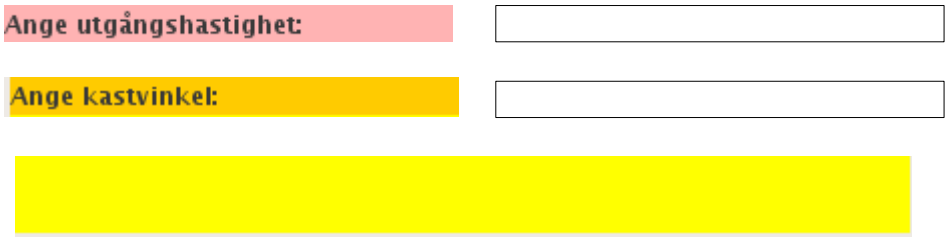
Om vi nu skriver ett huvudprogram som skapar ett objekt av klassen `Kast1` enligt:

```
import javax.swing.*;
public class Main {
    public static void main(String[] args) {
        JFrame w = new Kast1();
    }//main
}//Main
```

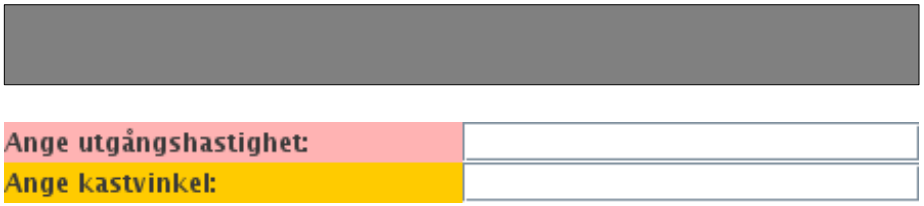
får vi ett fönster med följande utseende:



# Fönstret är uppbyggt på följande sätt:

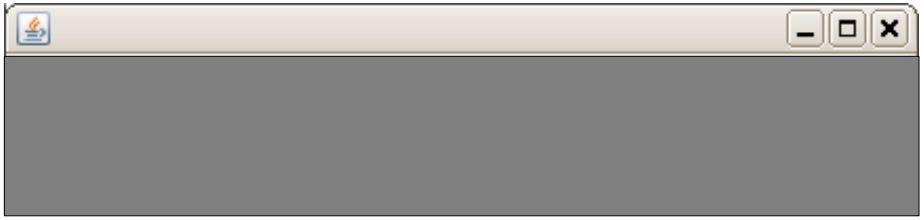



Komponenterna som visas i fönstret: `hLabel`, `hField`, `vLabel`, `vField` samt `resultatsLabel`.

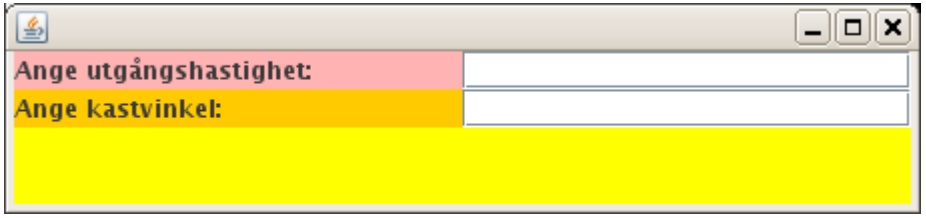



Panelen `frågePanel`.

På vilken komponenterna `hLabel`, `hField`, `vLabel` och `vField` placeras ut enligt i koden angivet mönster.



Själva fönstret, det ursprungliga `JFrame`-objektet som ärvs.




I vilket komponenterna `frågePanel` och `resultatsLabel` placeras ut enligt i koden angivet mönster.

# 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.

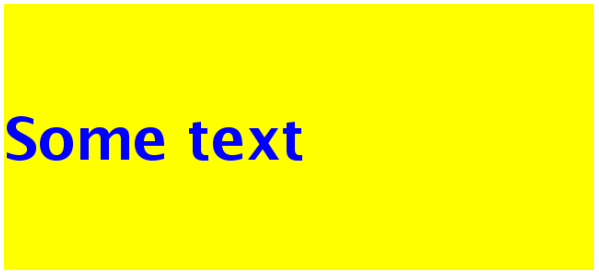
# Klassen `javax.swing.JComponent`

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

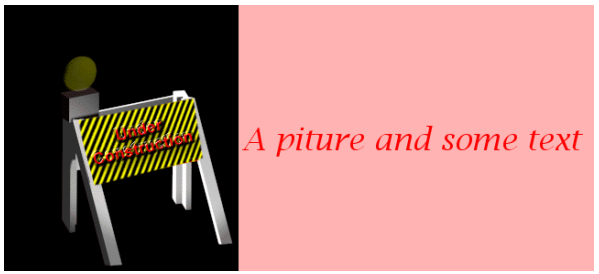
- Komponentens utseende (färg, font, markör etc)
- Komponentens placering/storlek
- Ritning av komponenten
- Komponentens tillstånd (om den är visad/gömd, aktiverad/avaktiverad etc)
- Komponentens fokus (kan den fokuseras? vad är nästa fokuskomponent?)
- Om komponenten skall ha en ram eller inte.
- Om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

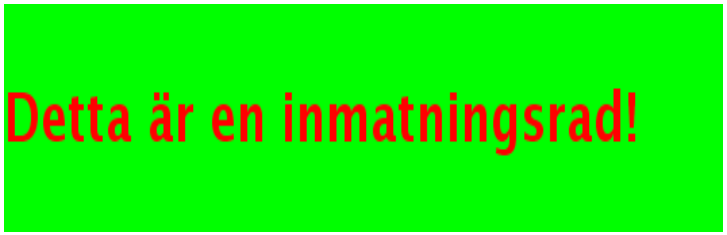

Vissa av dessa metoder finns i klassen JButton, andra metoder ärvs från superklasserna JComponent och Container.

# 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`:

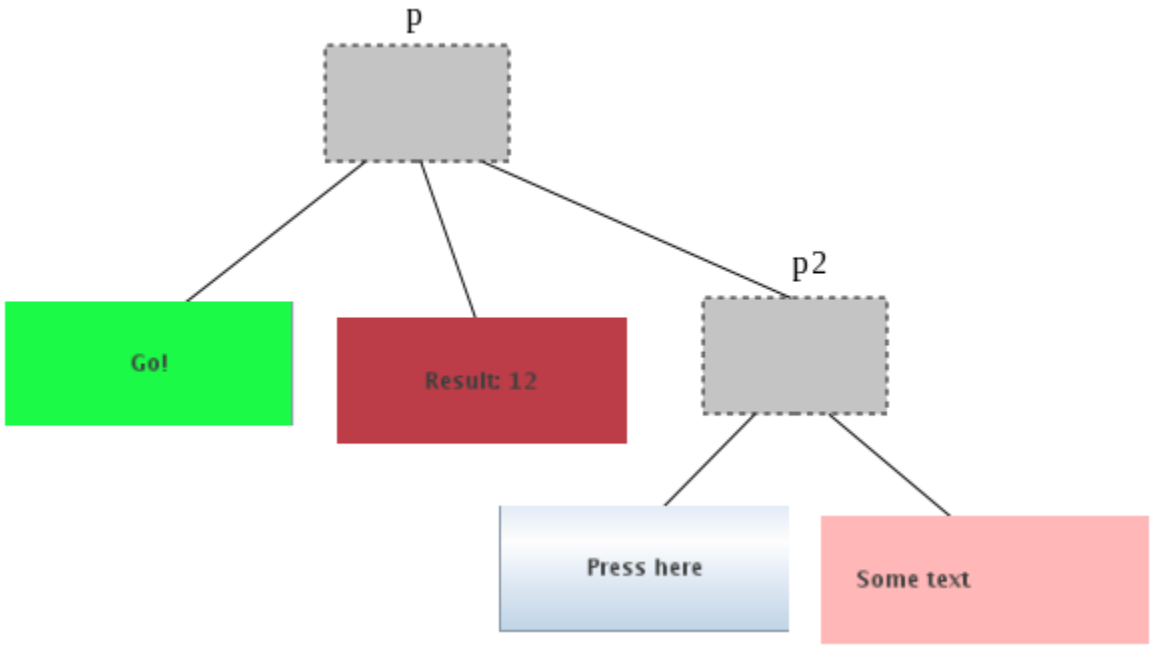
```
JPanel p = new JPanel();
p.add(annan komponent);
```

# 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. När man lägger komponenter i en JFrame vill man lägga dem i huvudytan, som man kommer 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));
      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
```