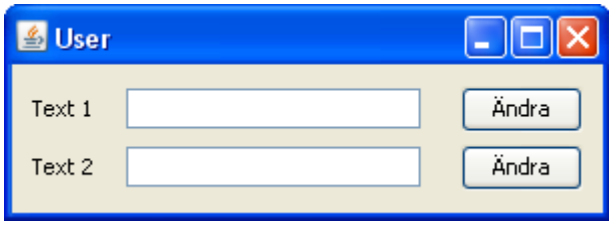
Föreläsning 16

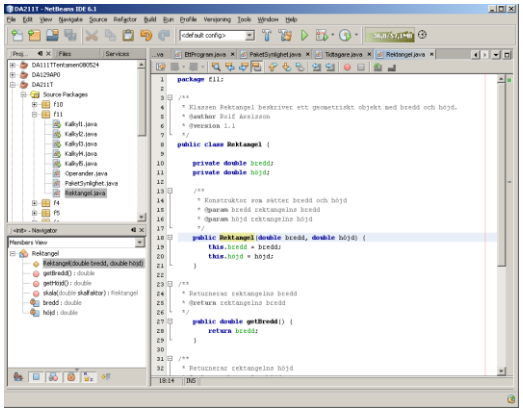
- GUI – Graphical User Interface
- Applikation - JFrame
- Ingen LayoutManager
- Skapa, designa och placera komponenter
- Händelsehantering – ActionListener
- Händelsehantering – ItemListener, ChangeListener

GUI – Graphical User Interface

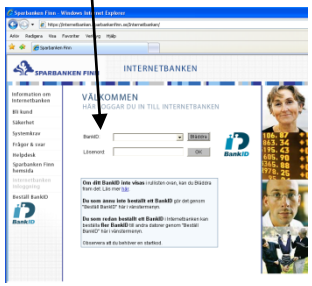
Ett javaprogram är antingen en fristående applikation eller en applet (körs normalt i en browser).

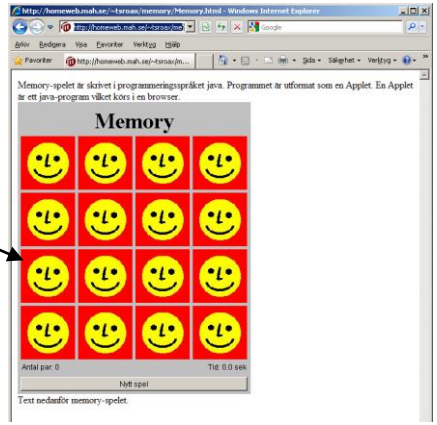
- Fönstret i en applikation bygger ofta på klassen JFrame.





- En applet bygger på klassen JApplet.





homeweb.mah.se/~tsroax/memory/Memory.html

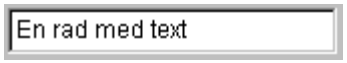
GUI – Graphical User Interface

Ett grafiskt program byggs upp av grafiska objekt (komponenter). Dessa objekt ärver klassen JComponent. Exempel på sådana objekt är:

JButton

JLabel

JTextField



JRadioButton, JCheckBox

JPanel

JSlider

JList

JComboBox

JTextArea

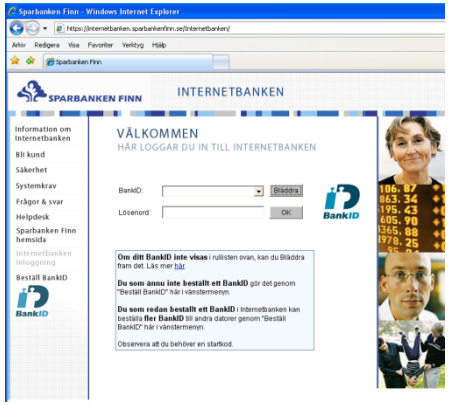
I paketet javax.swing finns mycket av det grafiska användargränssnittet samlat.

GUI – Container

Det går att placera grafiska komponenter i objekt som ärver Container. Exempel på sådana objekt är

- JFrame

- JApplet



- JPanel

Applikation - JFrame

I ett grafiskt program behöver man ett fönster.

```
JFrame frame = new JFrame( "Ett program :-)" );
```

Man kan göra vissa inställningar för fönstret, t.ex. fönstrets placering på skärmen, om fönstret ska vara stängbart, om man ska kunna ändra storleken på fönstret…

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);      // Stängbart
frame.setVisible(true);                                    // Synligt
```

Ett enkel program kan se ut så här:

```
import javax.swing.*;

public class Application1 {
   public void start() {
      JFrame frame = new JFrame( "Ett program :-)" );
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }

   public static void main(String[] args) {
      Application1 app = new Application1();
      app.start();
   }
}
```

Application0.java

Några metoder i JFrame

- **setBounds(x,y,bredd,höjd)**
  Fönstrets placering och storlek på bildskärmen.
  Kan ersättas med setLocation + setSize.
- **setLayout(LayoutManager)**
  Regler för hur komponenter ska placeras i fönstret
- **setLocation(x,y)**
  Fönstrets placering på bildskärmen.
- **setSize(bredd,höjd)**
  Fönstrets storlek på bildskärmen.
- **setTitle("Titel")**
  Fönstrets titel
- **setDefaultCloseOperatiron(JFrame.EXIT_ON_CLOSE)**
  Applikationen avslutas då fönstret stängs
- **setResizable(false)**
  Anger om fönstret kan minimeras/maximeras
- **setVisible(true / false)**
  Gör fönstret synligt / osynligt
- **pack()**
  anpassar fönstrets storlek till komponenterna

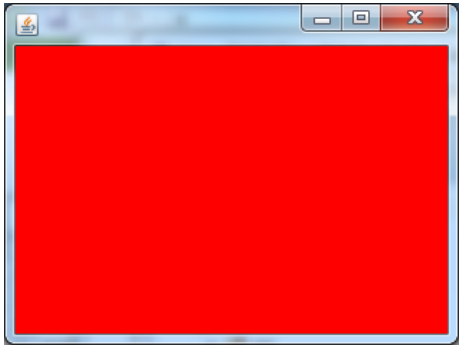
En applikation

```
import javax.swing.*;  // JFrame

public class Application {
   public void start() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      frame.add( new ApplicationPanel() );  // ApplicationPanel???
      frame.pack();
      frame.setVisible( true );
   }

   public static void main(String[] args) {
      Application app = new Application();
      app.start();
   }
}
```

```
import javax.swing.*;          // JPanel
import java.awt.*;             // Color, Dimension

public class ApplicationPanel extends JPanel { // Arv – mer om detta snart
   public ApplicationPanel() {
      setBackground( Color.RED );
      setPreferredSize( new Dimension(300,200) );
   }
}
```

En applikation

```
import javax.swing.*;  // JFrame, JPanel

public class Application {
   public void start() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      frame.add( new ApplicationPanel() );
      frame.pack();
      frame.setVisible( true );
   }

   public static void main(String[] args) {
      Application app = new Application();
      app.start();
   }
}
```



```
public class ApplicationPanel extends JPanel { // Arv – mer om detta snart
    public ApplicationPanel() {
        setBackground( Color.RED );
        setPreferredSize( new Dimension(300,200) );
    }
}
```

LayoutManager

När man ska placera ut komponenter i en container så kan man ha hjälp av s.k. layout-managers. En layout-manager placerar komponenterna efter bestämda regler. På kursen ska vi behandla följande layout-managers:

- BorderLayout – default i JFrame
- GridLayout
- FlowLayout – default i JPanel
- BoxLayout
- Null layout – ingen layout-manager

Med metoden **setLayout** anger man den layout-manager som ska användas.

```
import javax.swing.*;

public class ApplicationPanel extends JPanel {

   public ApplicationPanel() {
      setLayout( new GridLayout() ); // LayoutManager = GridLayout
      :
   }
}
```

FlowLayout

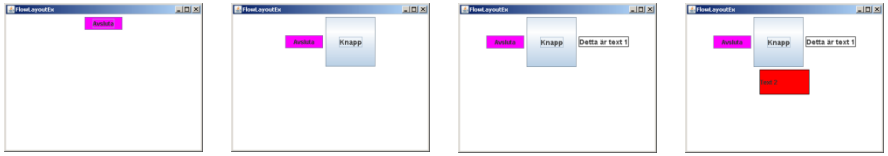
Komponenterna placeras från vänster till höger, uppifrån och ner. Komponenterna behåller den storlek de erhåller från metoden **setPreferredSize**.

JPanel använder FlowLayout om inget annat anges. Komponenter placeras på containern med add-metoden

container.add(komponent);

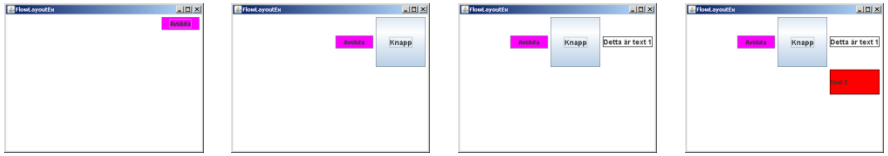
Konstruktorer:

public FlowLayout()
Komponenterna centreras



public FlowLayout(int align)
align = SwingConstants.LEFT / SwingConstants.CENTER / SwingConstants.RIGHT

SwingConstants.RIGHT



public FlowLayout(int align, int hgap, int vgap)
align = SwingConstants.LEFT / CENTER / RIGHT
hgap resp vgap anger avståndet mellan komponenterna

FlowLayoutApp.java

FlowLayoutPanel.java

ApplicationPanel – lägga till komponent

1. En komponentreferens ska deklareras och en komponent ska skapas. Detta gör man med fördel i början av klassen.

```
public class Application2Panel extends JPanel {
   private JLabel  lblText = new JLabel( "Jag visar en text" );
   private JButton btnExit = new JButton( "Avsluta" );
   private Font font = new Font( "SansSerif", Font.BOLD, 14 );
```

2 Med set-metoder kan man "designa" komponenten, dvs. ge den speciella egenskaper, t.ex.

```
public Application2Panel() {
   setPreferredSize( new Dimension( 200, 150 ) );
   setBackground( Color.BLACK );

   lblText.setForeground( Color.BLUE );  // Textens färg
   lblText.setFont( font );  // Typsnitt för texten

   btnEidt.setPreferredSize( new Dimension( 120, 40 ) );  // bredd=120, höjd=40
   btnExit.setForeground( new Color( 50, 184, 198 ) );  // Textens färg
   btnExit.setBackground( Color.RED );  // Knappens färg
}
```

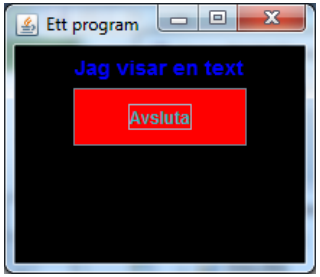
Detta gör man oftast i konstruktorn eller i metoder som anropas från konstruktorn.

Grafisk komponent – placera i Container

3. Man placerar en komponent med hjälp av add-metoden.

```
add( lblText );      // komponenten som lblText refererar till placeras i containern
add( btnExit );      // komponenten som btnExit refererar till placeras i containern
```

Denna kod placeras också som regel i konstruktorn.



Application2.java

Application2Panel.java

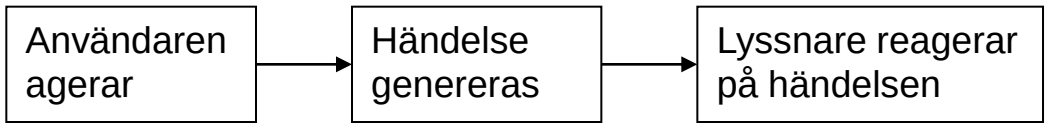
För att testa hur panelen ser ut kan man använda metoden *JOptionPane.showMessageDialog*

```
public static void main( String[] args ) {
   Application2Panel panel = new Application2Panel ();
   JOptionPane.showMessageDialog( null, panel );
}
```

Händelser - lyssnare

För att reagera på händelser använder man lyssnare. Här följer exempel på några lyssnare och händelser de kan reagera på:

Lyssnare	Ex på händelser
ActionListener	Klick på JButton Klick på JRadioButton Val i JComboBox
ItemListener	Markera/avmarkera JCheckBox
ChangeListener	Ändring i JSlider
MouseListener	Klick på komponent Musmarkör flyttad in på komponent Musmarkör flyttad från komponent Musknapp nedtryck på komponent Musknapp släppt på komponent



Händelser och lyssnare

Implementerar man en lyssnare så måste man skriva ett antal metoder:

- ActionListener
  actionPerformed
- ItemListener
  itemStateChanged
- ChangeListener
  stateChanged
- MouseListener
  mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased
- MouseMotionListener
  mouseDragged, mouseMoved

Metoderna man skriver ska exekveras snabbt. Det är en speciell tråd som hanterar händelser och uppdaterar fönstret i grafiska program. Om en händelse-metod tar lång tid för att exekveras så "låser man programmet". Användaren får sitta och vänta på att metoden ska bli färdig.

Händelser och lyssnare

När man använder sig av en lyssnare så anger man det i klassens huvud, efter ordet implements.
Om man i klassen Application2Panel vill använda sig av ActionListener så har man flera alternativ (programmet måste importera paketet java.awt.event).

1. Implementera lyssnaren direkt i klassen:

```
import java.awt.event.*;  // ActionListener, ActionEvent

public class Application2Panel extends JPanel implements ActionListener  {
   // instansvariabler, konstruktorer och metoder
}
```

2. Implementera lyssnaren i en inre klass:

```
import java.awt.event.*;  // ActionListener, ActionEvent

public class Application2Panel extends JPanel {
   // instansvariabler, konstruktorer och metoder
   private class AL implements ActionListener  {

   }
}
```

Händelser och lyssnare

Sedan måste man skriva metoden actionPerformed.

1. Implementera lyssnaren direkt i klassen

```
import java.awt.event.*;  // ActionListener, ActionEvent

public class Application2Panel extends JPanel  implements ActionListener  {
   // instansvariabler, konstruktorer och metoder
   public void actionPerformed( ActionEvent e) {
      System.exit( 0 );
   }
}
```

2. Implementera lyssnaren i inre klass

```
import java.awt.event.*;  // ActionListener, ActionEvent

public class Application2Panel extends JPanel {
   // instansvariabler, konstruktorer och metoder
   private class AL implements ActionListener {
      public void actionPerformed( ActionEvent e ) {
         System.exit( 0 );
      }
   }
}
```

Händelser och ActionListener

Det sista man ska göra för att lyssnaren ska fungera, det är att koppla lyssnaren till de komponenter som man vill reagera på. Det gör man med metoden

komponent.add…Listener(referens til lyssnare);

1. Om lyssnaren är implementerad direkt i klassen så

   ```
   btnExit.addActionListener( this );
   ```

   Med argumentet this talar man om att lyssnaren är implementerade i den aktuella klassen, dvs. i samma klass som komponentreferensen btnExit.

2. Om lyssnaren är implementerad i den inre klassen AL så:

   ```
   AL listener = new AL();
   btnExit.addActionListener( listener );
   ```

   eller

   ```
   btnExit.addActionListener( new AL() );
   ```

Kopplingen mellan lyssnare och komponent sker som regel i konstruktorn eller i metod som anropas från konstruktorn.

Application3Panel.java

Application4Panel.java

Klassen Color

Klassen Color används för att beskriva färger i Java. Systemet som används är RGB. Color-klassen tillhör paketet java.awt.

Det finns 13 fördefinierade färger:

Color.BLACK, Color.BLUE, Color.CYAN, Color.DARKGRAY, Color.GRAY, Color.GREEN, Color.LIGHTGRAY, Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE, Color.YELLOW

Om man önskar en annan färg deklarerar man ett färgobjekt och använder konstruktorn

Color(int r, int g, int b)

där r (röd), g (grön) och b (blå) ska vara värden i intervallet 0-255.

```
Color color = new Color(200,64,118);

Random rand = new Random();
Color rndColor = new Color( rand.nextInt(256), rand.nextInt(256), rand.nextInt(256) );
```

Klassen Font

Klassen Font används för att beskriva typsnitt i Java. Klassen Font finns i paketet java.awt.

Fördefinierade typsnitt är bl.a.:

`MonoSpaced`, SansSerif, Serif, Dialog och `DialogInput`

Varje gång man ska ändra Font så behöver man ett Font-objekt.

```
Font font = new Font(”Serif”, Font.PLAIN, 18);
:
label.setFont(font);
```

Ovanstående kod sätter aktuell Font till en liknande Times New Roman, normal stil och 18 punkter stor.

Man kan skapa Font-objektet när man gör anropet. Då gör man så här:

```
label.setFont( new Font(”Serif”, Font.PLAIN, 18) );
```

Först skapas Font-objektet. Referensen till Font-objektet är argument till metoden setFont.

MultiPanel.java