Objects First With Java
A Practical Introduction Using BlueJ

Att skapa grafiska användargränsnitt

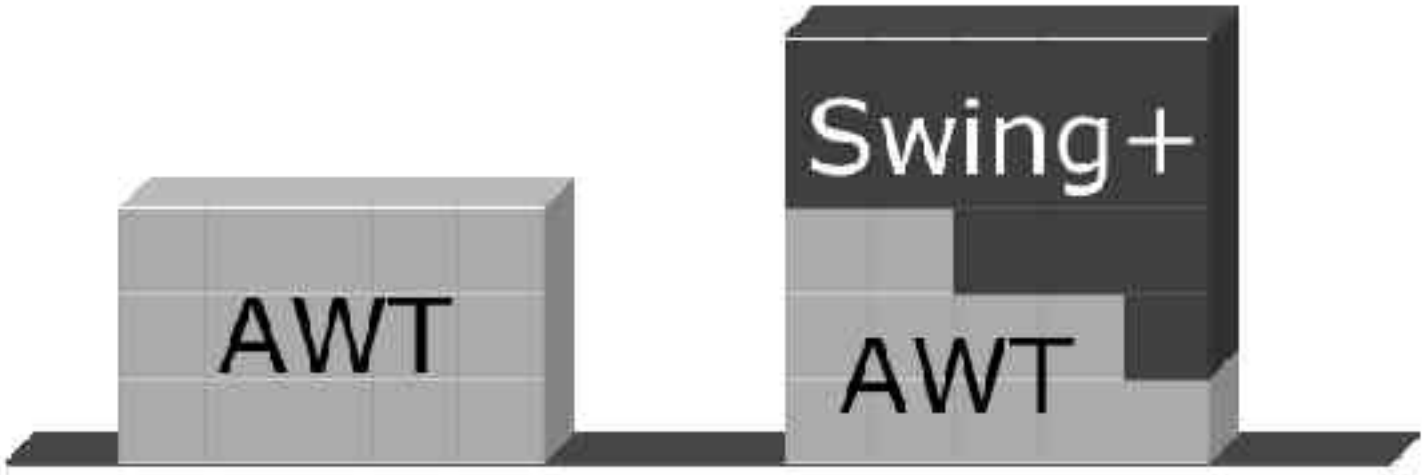
Översikt

- Konstruktion av grafiska användargränssnitt - GUIs
- Komponenter
- Layout
- Händelsehantering

GUI - principer

- Komponenter: Byggstenar för GUI.
  - Knappar, menyer, etc.
- Layout: Arrangerar komponenterna så att man får en användbar GUI.
  - Använder: *Layout managers*.
- Händelser: Objekt som skapas som en konsekvens av användaren ”gör något”.
  - En knapp klickas, ett menyval görs, etc.

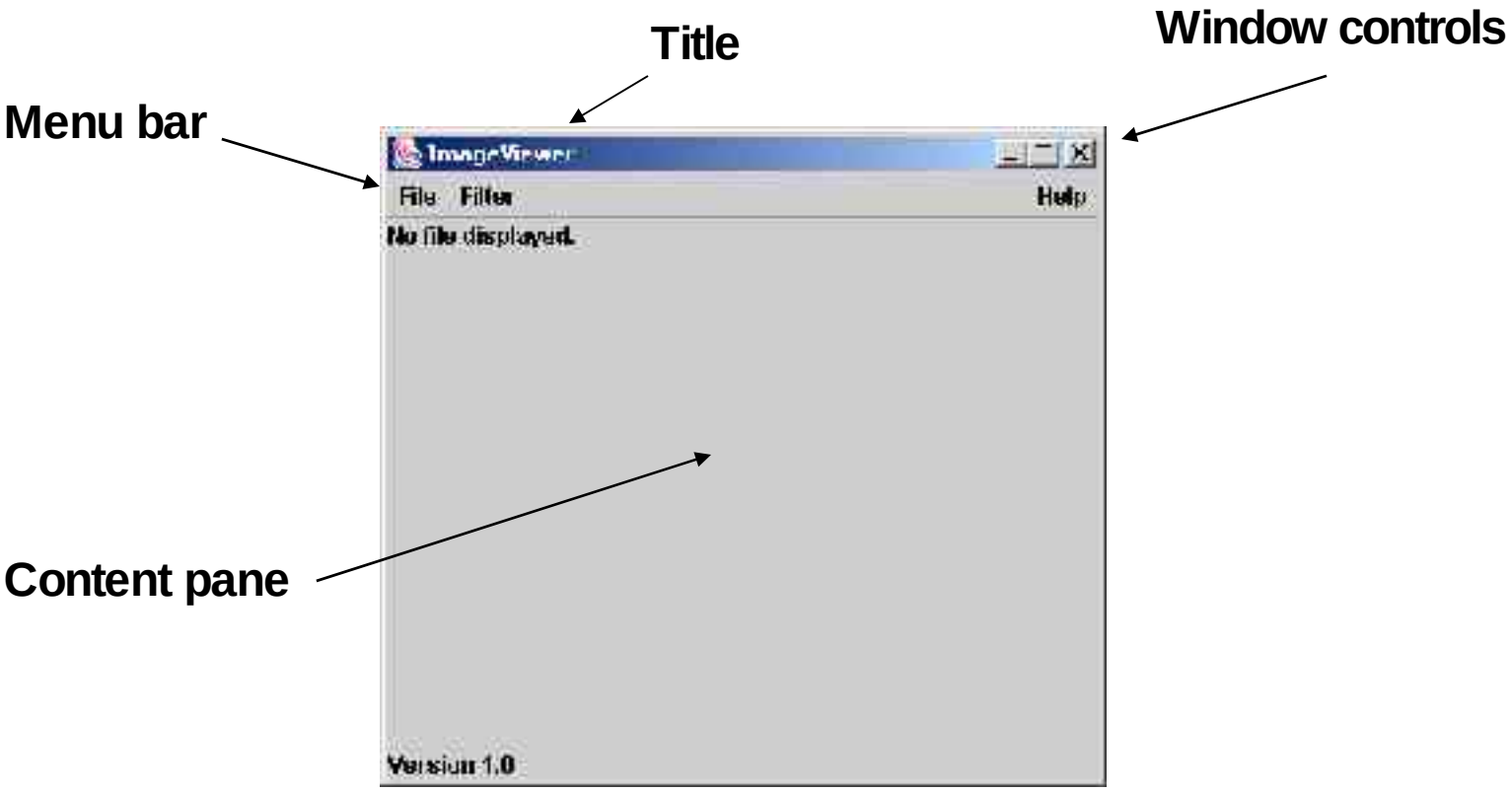
AWT och Swing



Hur man gör en ’frame’

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

// comment omitted.

public class ImageViewer
{
     private JFrame frame;

     /**
      * Create an ImageViewer show it on screen.
      */
     public ImageViewer()
     {
             makeFrame();
     }

     // rest of class omitted.

}
```

Beståndsdelarna hos en ’frame’



Content pane

```
/**
 * Create the Swing frame and its content.
 */
private void makeFrame()
{
    frame = new JFrame("ImageViewer");
    Container contentPane = frame.getContentPane();

    JLabel label = new JLabel("I am a label.");
    contentPane.add(label);

    frame.pack();
    frame.setVisible(true);
}
```

Hur menyer är organiserade

- `JMenuBar`
  - Visas under titelraden.
  - Innehåller menyerna.
- `JMenu`
  - i vårt exempel *File*. Innehåller menyalternativen.
- `JMenuItem`
  - i vårt exempel *Open.* De individuella menyalternativen.

```
private void makeMenuBar(JFrame frame)
{
    JMenuBar menubar = new JMenuBar();
    frame.setJMenuBar(menubar);

    // create the File menu
    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu);

    JMenuItem openItem = new JMenuItem("Open");
    fileMenu.add(openItem);

    JMenuItem quitItem = new JMenuItem("Quit");
    fileMenu.add(quitItem);
}
```

Händelsehantering

- Händelser uppstår när en användare interagrerar med en komponent.
- Komponenter är knutna till olika händelsetyper.
  - Frames är knutna till `WindowEvent`.
  - Menyer är knutna till `ActionEvent`.
- Objekt kan underrättas när en händelse uppstår.
  - Ett sådant objekt kallas ”lyssnare” (*listener)*.

Centraliserad händelsehantering

- Ett enda objekt hanterar alla händelser.
  - Implementerar gränssnittet `ActionListener`.
  - Definierar metoden `actionPerformed`.
- Registreras som lyssnare för alla komponenter.
  - `item.addActionListener(this)`
- Innebär att man i `actionPerformed` måste ta reda på vilket objekt som genererade händelsen.

```
public class ImageViewer implements ActionListener
{
    …
    public void actionPerformed(ActionEvent e)
    {
        String command = e.getActionCommand();
        if(command.equals("Open")) {
            …
        }
        else if (command.equals("Quit")) {
            …
        }
        …
    }
    …
    private void makeMenuBar(Jframe frame)
    {
        …
        openItem.addActionListener(this);
        …
    }
}
```

Centraliserad händelsehantering

- Det fungerar ...
- Det används, så man skall känna till det.
- Men …
  - Om man vill bygga ut så kan det bli jobbigt.
  - Att identifiera komponenter med ett namn blir lätt fel.
- Det finns bättre sätt.

Klasser får kapslas

```
public class Enclosing
{
    …
    private class Inner
    {
        …
    }
}
```

inre klass

Inre klasser

- Objekt av en inre klass har instansen av den omslutande klassen som omgivning.
- Objekt av den inre klassen kan referera privata fält och metoder hos det omslutande objektet.

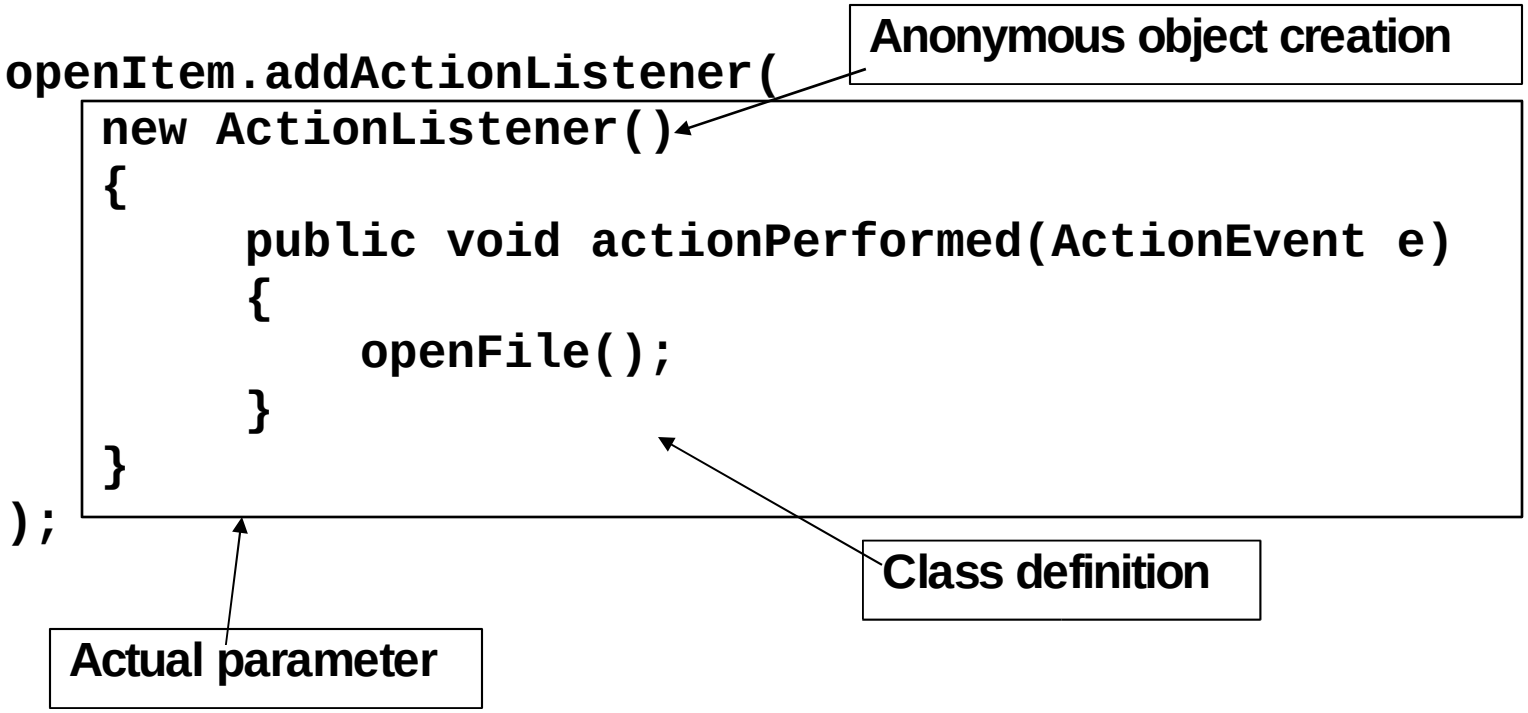
Anonyma inre klasser

- Fungerar enligt reglerna för inre klasser.
- Används för att generera ett enstaka objekt som inte behöver ha ett typnamn.
- En speciell syntax används.
- Objektet refereras via sin supertyp eftersom det inte har något (sub)typnamn.

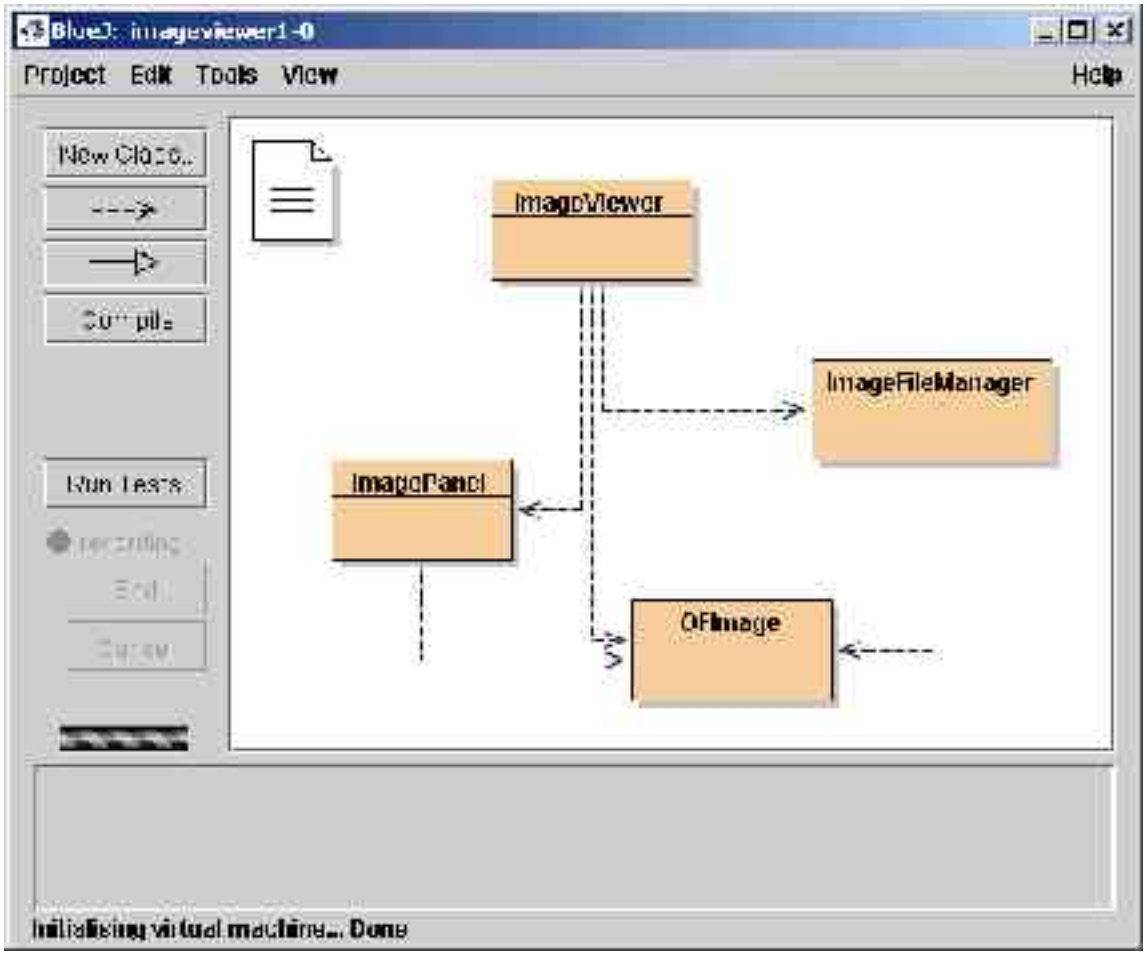
Anonym ’action listener’

```
JMenuItem openItem = new JMenuItem("Open");

openItem.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
         openFile();
     }
});
```

Anonyma klasselement



Projektet: imageviewer



Bildhantering

Vad som görs i de olika klasserna

- `ImageViewer`
  - Sätter upp strukturen för användargränssnittet.
- `ImageFileManager`
  - Innehåller statiska metoder för att ladda och spara bildfiler.
- `ImagePanel`
  - Visar bilden i användargränssnittet.
- `OFImage`
  - Subklass till den klass som innehåller bilden.

OFImage

- Vår subklass till `BufferedImage`.
- Representerar en 2D-array av pixlar.
- Viktiga metoder:
  - `getPixel`, `setPixel`
  - `getWidth`, `getHeight`
- Varje pixel har en färg.
  - Vi använder `java.awt.Color`.

Vi lägger till en ImagePanel

```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void makeFrame()
    {
        Container contentPane = frame.getContentPane();
        imagePanel = new ImagePanel();
        contentPane.add(imagePanel);
    }

    …
}
```

Vi laddar en bild

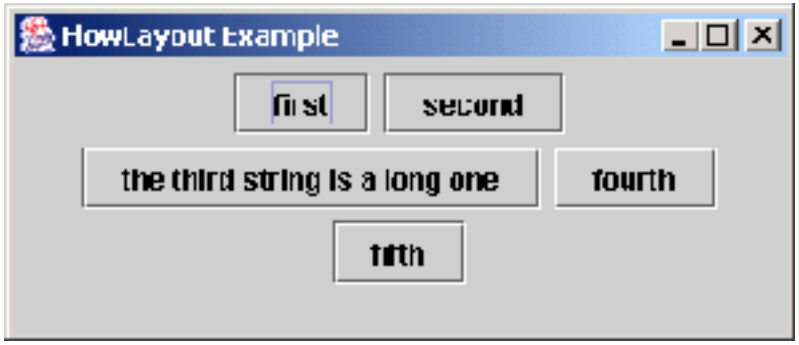
```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void openFile()
    {
        File selectedFile = …;
        OFImage image =
            ImageFileManager.loadImage(selectedFile);
        imagePanel.setImage(image);
        frame.pack();
    }

    …
}
```

Layout managers

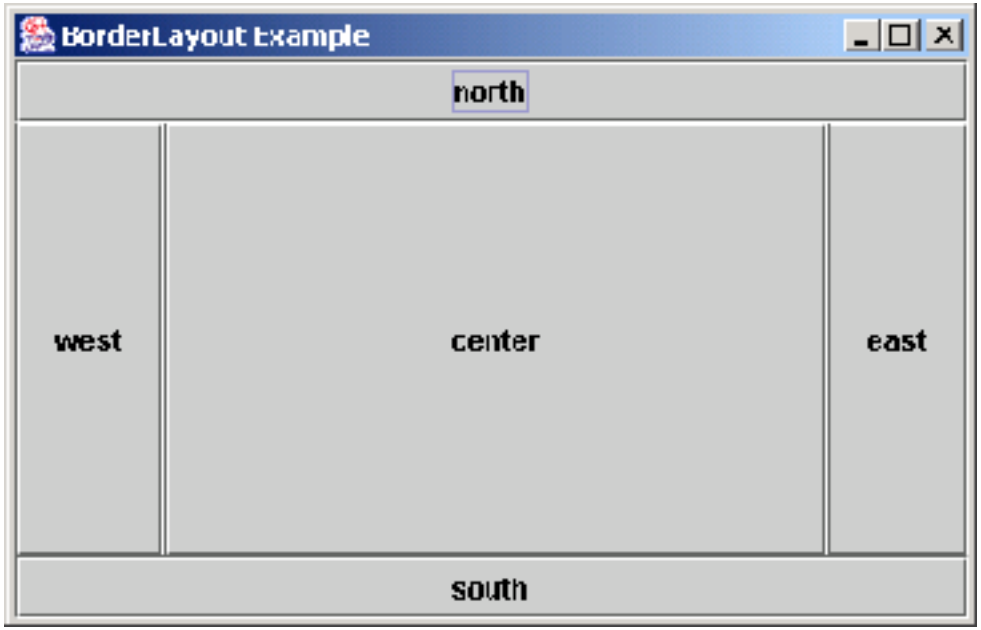
- Hanterar ett begränsat utrymme för konkurrerande komponenter.
  - `FlowLayout`, `BorderLayout`, `GridLayout`, `BoxLayout`, `GridBagLayout`.
- Hanterar ett antal `Container` objekt, som t.ex. *content pane*.
- Varje *layout manager* har en eget sätt att göra detta.

FlowLayout

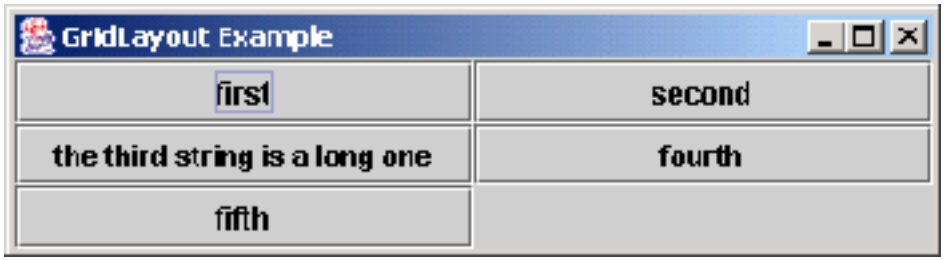

FlowLayout Example
first
second
the third string is a long one
fourth
fifth

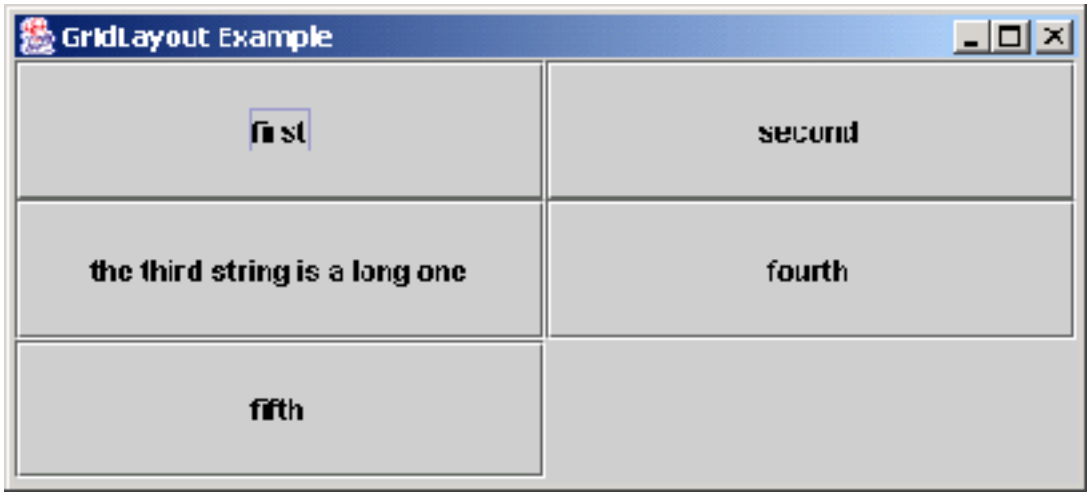

FlowLayout Example
first
second
the third string is a long one
fourth
fifth

BorderLayout

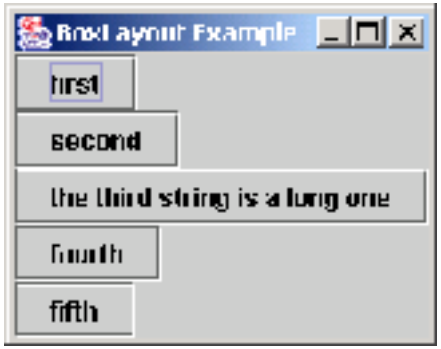


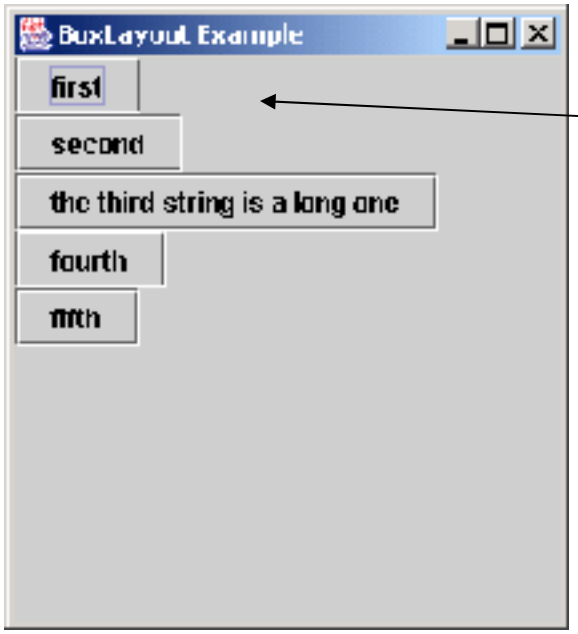
GridLayout


GridLayout Example
first
second
the third string is a long one
fourth
fifth


GridLayout Example
first
second
the third string is a long one
fourth
fifth

BoxLayout





Note: no component resizing.

Kapslade *containers*

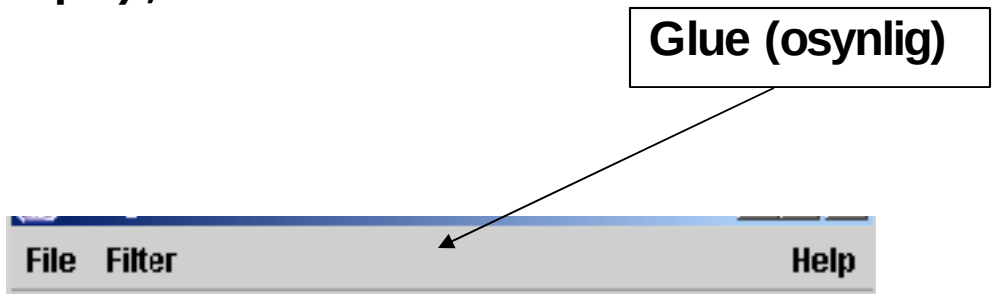
- Man kan göra mer sofistikerade *layouts* genom att kapsla *containers*.
  - Använd `JPanel` som en bas-*container*.
- Varje *container* skall ha en egen *layout manager*.
- Ofta bättre än att använda `GridBagLayout`.

Struts and Glue (Stöttor och lim)

- Använd osynliga komponenter för att åstadkomma ett utrymme.
- Finns i klassen `Box`.
- Strut: Ger en fix storlek.
  - **`Component createHorizontalStrut(int width)`**
  - **`Component createVerticalStrut(int height)`**
- Glue: Fyller ut tomt utrymme..
  - **`Component createHorizontalGlue()`**
  - **`Component createVerticalGlue()`**

Hur man delar på en menyrad

```
menu = new JMenu("File");
menubar.add(menu);

menu = new JMenu("Filter");
menubar.add(menu);

menubar.add(Box.createHorizontalGlue());

menu = new JMenu("Help");
menubar.add(menu);
```



Dialoger

- Modala dialoger blockerar all annan interaktion.
  - Tvingar fram ett svar från användaren.
- Icke-modala dialoger tillåter annan interaktion.
  - Ger användaren större frihet.
  - Kan innebära att programmet får in uppgifter som är inkonsistenta.

JOptionPane
Definierar ett antal standardiserade dialoger

- Meddelande
  - Meddelandetext plus en OK-knapp.
- Bekräftelse
  - Valmöjligheter: Yes, No, Cancel.
- Inmatning
  - Meddelandetext och ett fält för input.
- Kan varieras.

Ett meddelande

```
private void showAbout()
{
    JOptionPane.showMessageDialog(frame,
                "ImageViewer\n" + VERSION,
                "About ImageViewer",
                JOptionPane.INFORMATION_MESSAGE);
}
```

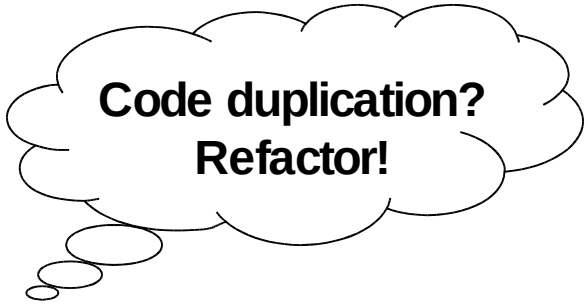
Bildfiltrering

- Funktioner som appliceras på hela bilden.

```
int height = getHeight();
int width = getWidth();
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        Color pixel = getPixel(x, y);
        alter the pixel's color value;
        setPixel(x, y, pixel);
    }
}
```

Fler filter

```
private void makeLighter()
{
    if(currentImage != null) {
        currentImage.lighter();
        frame.repaint();
        showStatus("Applied: lighter");
    }
    else {
        showStatus("No image loaded.");
    }
}
```

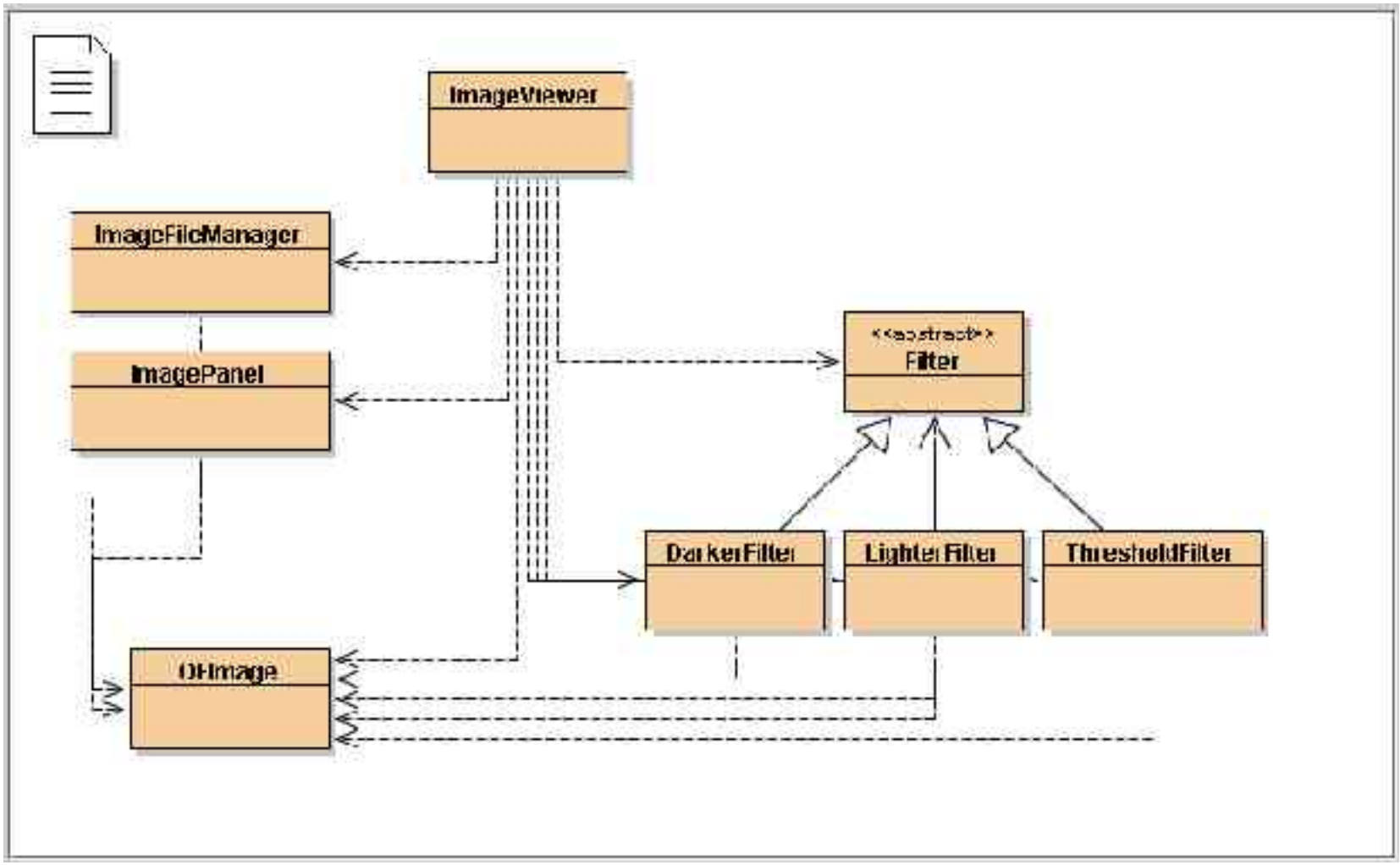



```
private void threshold()
{
    if(currentImage != null) {
        currentImage.threshold();
        frame.repaint();
        showStatus("Applied: threshold");
    }
    else {
        showStatus("No image loaded.");
    }
}
```

Ännu fler filter

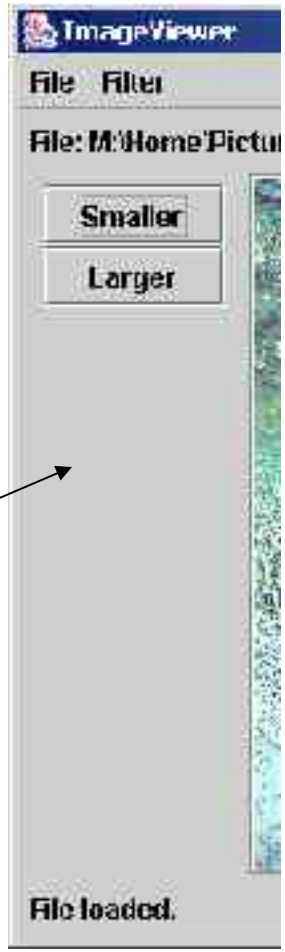
- Definiera en abstrakt superklass `Filter`.
- Skapa funktionsspecifika subklasser.
- Skapa en samling subklass-instanser i `ImageViewer`.
- Definiera en generisk metod `applyFilter`.
- Vi tittar på *imageviewer2-0*.

imageviewer2-0



Knappar och kapslade *layouts*

**En GridLayout inuti
en FlowLayout inuti
en BorderLayout.**



Komponentinramningar (*borders*)

- Används för att dekorera inramningen av en komponent.
- Definieras i `javax.swing.border`
  - `BevelBorder`, `CompoundBorder`, `EmptyBorder`, `EtchedBorder`, `TitledBorder`.

Extra utrymme i komponenter

```
JPanel contentPane = (JPanel)frame.getContentPane();
contentPane.setBorder(new EmptyBorder(6, 6, 6, 6));

// Specify the layout manager with nice spacing
contentPane.setLayout(new BorderLayout(6, 6));

imagePanel = new ImagePanel();
imagePanel.setBorder(new EtchedBorder());
contentPane.add(imagePanel, BorderLayout.CENTER);
```

Andra komponenter

- Slider
- Spinner
- Tabbed pane
- Scroll pane

Summering

- Det gäller som alltid att få en hög grad av samhörighet (cohesion) i de klasser man skriver.
  - Sträva efter att separera användargränssnittet från resten av applikationen: InputView, OutputView.
- Fördefinierade komponenter i AWT och Swing underlättar utformningen av stabila och funktionella användargränssnitt.
- Layout managers hanterar komponenternas inbördes lägen.
  - Kapsla containers för att få mer kontroll.

Summering

- Många komponenter kan förmedla användarinteraktion (musklick, musrörelser, tangenttryckning).
- Reaktiva komponenter skickar händelser till lyssnare som hanterar händelserna.
- Anonyma inre klasser används ofta för att implementera lyssnare.