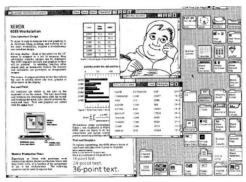


GUI-programmering

Cristian Bogdan
cristi@kth.se

DH2640 Grafik och Interaktionsprogrammering VT 2010

GUI till tusen

- Win32-API (C)
- MFC (C++)
- VCL/CLX (C++/Delphi)
- wxWidgets (C++)
- Qt (C++/Java)
- GTK+ (C)
- .NET Framework (VB/C++/C#)
- Java AWT/Swing/SWT (Java)
- osv…

GUI till tusen

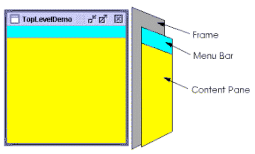
- Vi ska titta på Java Swing
- Mycket gemensamt med andra GUI-system, ibland under andra namn
- Bra uppsättning komponenter
- Platformsoberoende
- ”Bra” designat enligt designmönster
- Gratis utvecklingsmiljö (NetBeans) och mycket dokumentation från Sun

Java/Swing

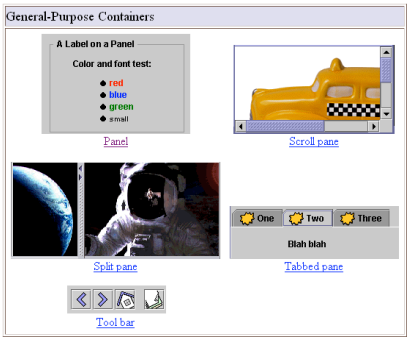
- Gränssnitt i Java (och de flesta andra GUI-system) är **hierarkiska**.
- Man börjar med en s.k. **Top Level Container**, och lägger till komponenter i den.
- Vissa komponenter är också containers och kan innehålla andra komponenter.
- Man specificerar dimensioner och position för varje komponent explicit eller via s.k. **Layout Managers**.

Top Level Containers

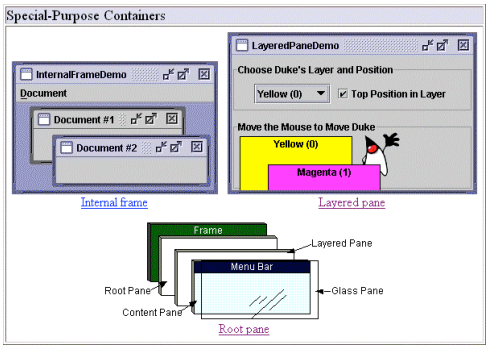
- Varje komponent kan bara befinna sig på en plats i hierarkin.
- Varje Top Level Component har en **Content Pane** som innehåller hierarkin.
- Man kan lägga till en **menu bar** (menyrad) ovanför sin content pane.



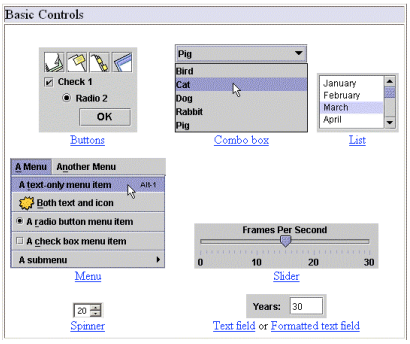
General Purpose Containers



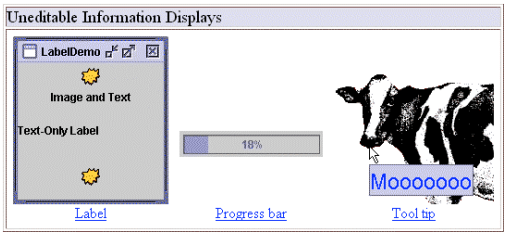
Special Purpose Containers



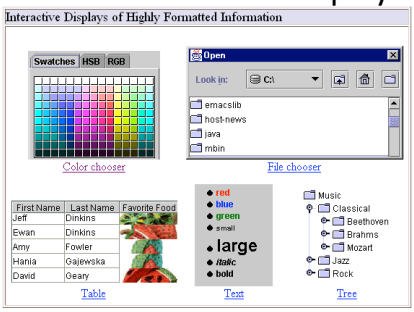
Basic Controls



Uneditable Info Displays



Formatted Info Displays



Typically hve an elaborate model which must be populated by the application

Containers, Controls och Displays

- **Top Level Containers**: Applet, Dialog, Frame.
- **General-Purpose Containers**: Panel, Scroll Pane, Split Pane, Tabbed Pane, Tool Bar.
- **Special-Purpose Containers**: Internal frame, Layered Pane, Root Pane.
- **Basic Controls**: Buttons, Lists, Menus, etc.
- **Uneditable Info Displays**: Label, Progress Bar, Tool Tip.
- **Interactive Displays of Highly Formatted Information**: File Chooser, Color Chooser, etc.

Kan vi göra något åt utseendet?

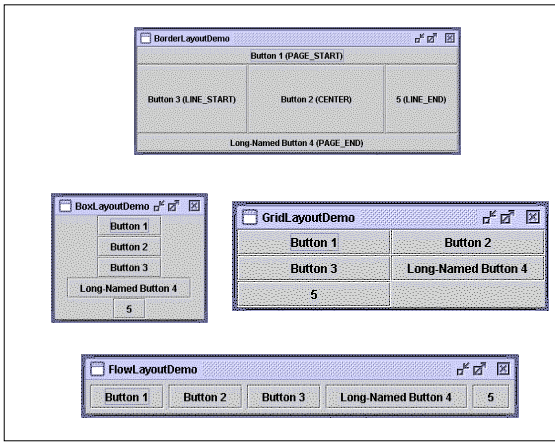
- Ja, det kan vi!
- Swing kan få ”native look & feel” om vi lägger till några rader kod…

```
// Get the native look and feel class name
String nativeLF = UIManager.getSystemLookAndFeelClassName();

// Install the look and feel
try {
        UIManager.setLookAndFeel(nativeLF);
}
catch(Exception e) {
}
```

Layout Managers

- Det är oftast en nackdel att specificera position och dimension för komponenter explicit.
- Om användaren t.ex. ändrar fönstrets storlek kommer inte storleken på komponenterna att hänga med.
- Layout managers hjälper en att räkna ut positioner och dimensioner dynamiskt!
- Man kan skapa egna layout managers.



Inmatningstyper

- Request
  - Lämna över kontrollen till användaren och vänta på att något sker.
  - Exempel: terminal i Unix.
- Sampling / polling
  - Läsa av värdet hos en enhet så ofta som möjligt.
  - Exempel: Spela in ljud från ljudkort

Sampling architectures

- the application needs to ask whether there are new events
- read/write data and continue execution
- leads often to active loops in the applications

Interrupt based architectures

- the application registers to be notified when a device state changes, and it provides an interrupt handler procedure
- the operating system calls the procedure when the device state changes
- leads to complicated asynchronous situations

Event-based architectures

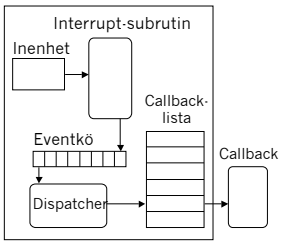
- the operating system takes care of the changes in the device states
- an event object (instance) is generated for every state change
- contains information about the device change (e.g. the left mouse button was pressed at time T at the 100, 200 coordinates)
- the applications register methods to be called when a certain event occurs (callback, see Inversion of Control)

Event-based architectures (cont)

- after its generation, the event is placed in an event queue
- in another thread, an event dispatcher takes the events from the queue and calls the registered callback routines

Inmatningstyper: Events

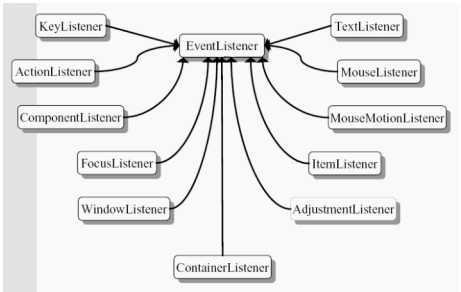
- Systemet placerar förändringar i en kö.
- Beta av kön vid lämpliga tillfällen.
- Skicka en s.k. **event-instans** med information om varje förändring till alla callbacks som är "intresserade".



Events i Java

- Man registrerar **Listeners** som var och en "lyssnar" på en viss typ av event
- Varje metod i en listener motsvarar något som hänt hos enheten i fråga

Events i Java



Event listener implementation

```
package tests;
import java.awt.*;
import java.awt.event.*;
class MyWindowListener implements WindowListener {
  public void windowOpened(WindowEvent e) {}
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
  public void windowClosed(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
}

public class MyFrame2 extends Frame {
  public static void main(String [] args) {
    Frame frame = new MyFrame2();
    frame.addWindowListener(new MyWindowListener());
    frame.setVisible(true);
  }
}
```

… only one class

```
package MyTests;
import java.awt.*;
import java.awt.event.*;
public class MyFrame1 extends Frame implements
WindowListener{
  public void windowOpened(WindowEvent e) {}
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
  public void windowClosed(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
  public static void main(String [] args) {
    MyFrame1 frame = new MyFrame1();
    frame.addWindowListener(frame);
    frame.setVisible(true);
  }
}
```

Adaptors

- Förenklar hanteringen av events
- Varje adaptor implementerar ett Listener-interface, men alla metoderna är tomma
- Om man subklassar en adaptor behöver man alltså bara skriva precis den kod som behövs

Adaptors

```
class MyWindowAdapter extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}

public class MyFrame3 extends Frame {
  public static void main(String [] args) {
    Frame frame = new MyFrame3();
    frame.addWindowListener(new MyWindowAdapter());
    frame.setVisible(true);
  }
}
```

...eller med anonym subklass

```
public class MyFrame4 extends Frame {
  public static void main(String [] args) {
    Frame frame = new MyFrame4();

    frame.addWindowListener(new WindowAdapter (){
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

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

Event listening alternatives

- Separate class listener
- Inner class listener
- Component (subclass of e.g. Frame, Button) class implements listener
- Anonymous inner class implements listener

- Adapter as separate class
- Adapter as inner class
- Anonymous inner class extends adapter

```
public class PlayerSteeringBehaviour
             implements Behaviour, KeyListener {

  protected boolean steering = false;
  protected float direction = 1.0f;

  public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == 37) {  // Cursor left
      steering = true;
      direction = -1.0f;
    }
    if (e.getKeyCode() == 39) {  // Cursor right
      steering = true;
      direction = 1.0f;
    }
  }

  public void keyReleased(KeyEvent e) {
    steering = false;
  }

  public void keyTyped(KeyEvent e) {}

...
```

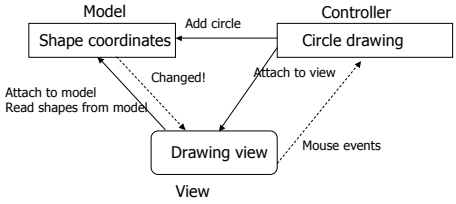
Avancerade gränssnitt

- Om man bygger mer avancerade program behövs ett bättre sätt att hantera användargränssnittet
- De flesta system bygger på ett **designmönster** som heter **Model-View-Controller** (eller bara **Model-View**)

Model View Controller

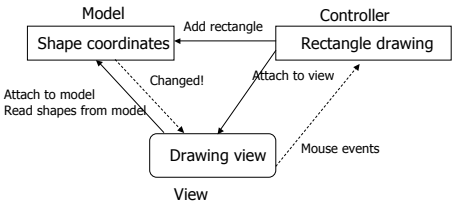
- En design pattern för användargränssnitt.
- Utvecklades under sent 70-tal för Smalltalk på Xerox-maskiner
- Example: a drawing editor with a toolbar
  - Selection, circle drawing, rectangle drawing
  - Each tool interprets a given sequence of mouse actions differently!
  - Same action, different semantics
- Shapes created can be viewed on screen
- Or sent as PostScript code to a printer

Model View Controller

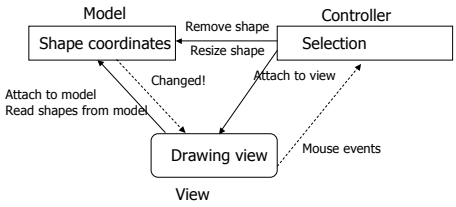


Om gränssnitten mellan de tre komponenterna inte ändras kan vilken som helst av dem bytas ut!

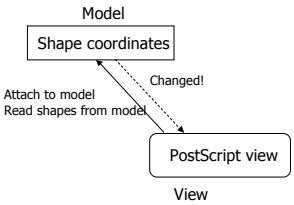
Model View Controller



Model View Controller



Model View Controller



No controller attached, you can't interact with a printout
But possible to keep an updated "print preview"
Both views can be active at the same time