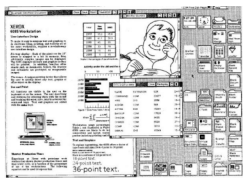
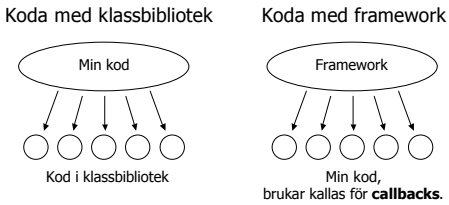


GUI-programmering

Cristian Bogdan
cristi@kth.se

Frameworks



Inversion of control (see Spring framework)
Hollywood principle (we'll call you)
Template method design pattern

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

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

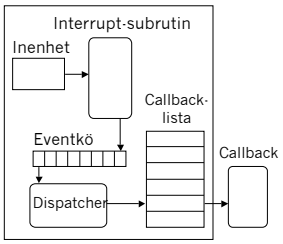
- the operating system takes care of the changes in the device states
- an event object (instance) is generated for every state change
- contains information about the device change (e.g. the left mouse button was pressed at time T at the 100, 200 coordinates)
- the applications register methods to be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

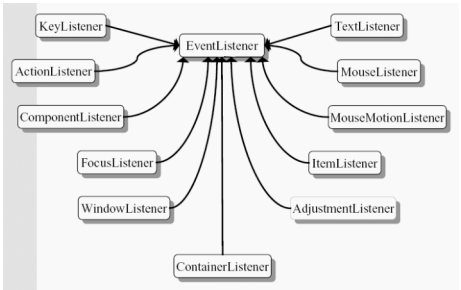
- Systemet placerar förändringar i en kö.
- Beta av kön vid lämpliga tillfällen.
- Skicka en s.k. **event-instans** med information om varje förändring till alla callbacks som är "intresserade".



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) {}

...
```

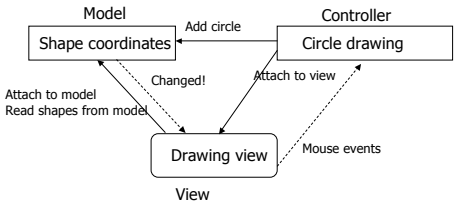
Avancerade gränssnitt

- 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

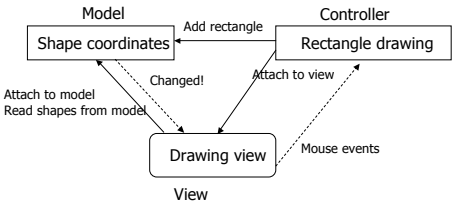
- En design pattern för användargränssnitt.
- Utvecklades under sent 70-tal för Smalltalk på Xerox-maskiner
- Example: a drawing editor with a toolbar
  - Selection, circle drawing, rectangle drawing
  - Each tool interprets a given sequence of mouse actions differently!
  - Same action, different semantics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

Model View Controller

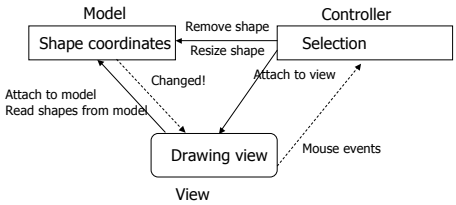


Om gränssnitten mellan de tre komponenterna inte ändras kan vilken som helst av dem bytas ut!

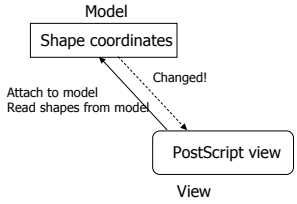
Model View Controller



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

Model-View

- Model: data and its changing rules ("application logic")
- A data model can have a number of views and can attach them dynamically
- Views are not necessarily graphical
- The model is unaware of the views' details, it just notifies them of changes, and allows the views to read the data

View-Controller

- A controller defines a pattern of interaction between a view and its model
- In the graphical case, the controller listens to events (or gets other forms of notification) from the view components and changes the model according to the event's semantics
- The view needs not know details about the controllers attached to it
- Alternative controllers of the same view define interaction strategies

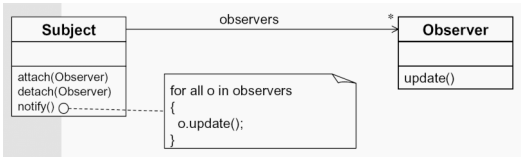
Model-View-Controller

- When the controller changes the model, the model notifies all its views to update
- Instead of changing the model, the controller can ignore or compensate for user interaction that is not consistent with the model assumptions (possibly warn user)
  - in an editor, you can't move past the end of the text
- S E P A R A T I O N
  - data description: MODEL
  - data illustration: VIEWs
  - interaction: CONTROLLERs

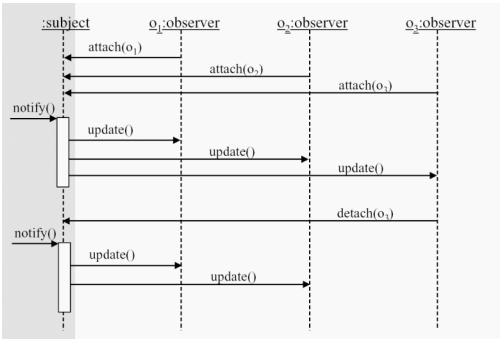
Observer

- För att implementera MVC används ofta designmönstret **Observer**
- Tillåter att ett objekt meddelas om det gjorts en förändring i ett annat objekt, utan att objekten görs starkt knutna till varandra
- Event listeners (delegation) are an alternative to Observer

Observer



Observer



Observer i Java

```
package java.util;

public interface Observer {
  void update(Observable o, Object arg);
}

public class Observable {
  private Observer[] arr;

  public void addObserver(Observer o) {
    arr.addElement(o);
  }

  public void notify(Object arg) {
    for (int i = 0; i < arr.size; i++) {
      arr[i].update(this, arg);
    }
  }
}
```

Det här är inte hela sanningen, men principen är densamma i den riktiga java-implementationen!

Issues with MVC

- There are different interpretations of the MVC architecture
- Strictly speaking, in Swing the controller is the event dispatcher
- Often a view will have only one controller, hard or not useful to separate
- But model and view separation is often easily possible and profitable
- MVC is implemented often in multiple layers
- Video prototype example, we identify the model and views

Video proto example

- Basic Model: tasks, task planning dates/lines, task rules:
  - Tasks can't collide on a line / date interval
  - Move tasks automatically to earliest possible dates
- Views:
  - task planning chart, controller: drag and drop
  - task table, controllers: editing, creation

Video proto example

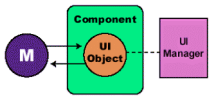
- Higher level model: current task
  - Does not affect the core application logic
  - But helps the two views coordinate
- Even higher level model: the table model
  - Cell data: from basic model
  - Cell selection: from current task
  - See Swing JTable

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**.

Separable Model Architecture

- Varje komponent hanterar view/control.
- Själva utritningen är delegerad till ett s.k. UI object (för att man ska kunna ändra look-and-feel, t.ex.).
- Varje komponent har en modell kopplad till sig.
  - Hierarchical MVC
  - Aka: Presentation-abstraction-control (PAC)



Separable Model Architecture

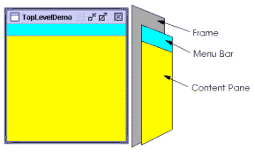
- De flesta komponenters modell har enbart att göra med själva gränssnittet (t.ex. om en checkbutton är nedtryckt eller ej).
- Vissa komponenters innehåll kan dock bara definieras av applikationen (t.ex. innehållet i en lista eller tabell).
- Några faller mitt emellan (t.ex. sliders).
- Man ersätter en modell genom att subklassa defaultmodellen och sedan anropa `setModel()` på komponenten.

Containers, Controls och Displays

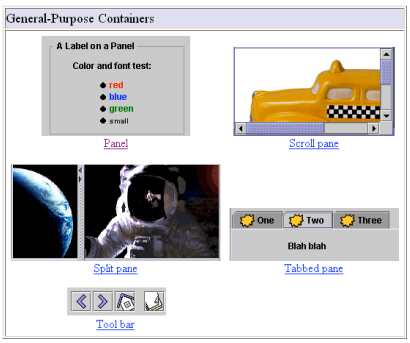
- **Top Level Containers**: Applet, Dialog, Frame.
- **General-Purpose Containers**: Panel, Scroll Pane, Split Pane, Tabbed Pane, Tool Bar.
- **Special-Purpose Containers**: Internal frame, Layered Pane, Root Pane.
- **Basic Controls**: Buttons, Lists, Menus, etc.
- **Uneditable Info Displays**: Label,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

Top Level Containers

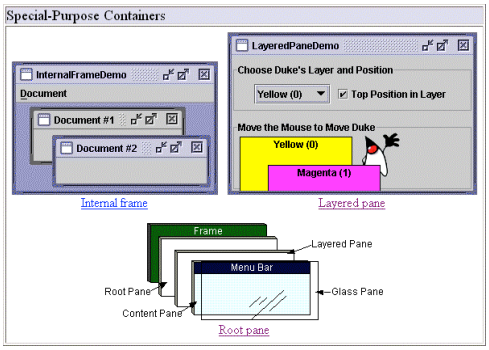
- Varje komponent kan bara befinna sig på en plats i hierarkin.
- Varje Top Level Component har en **Content Pane** som innehåller hierarkin.
- Man kan lägga till en **menu bar** (menyrad) ovanför sin content pane.



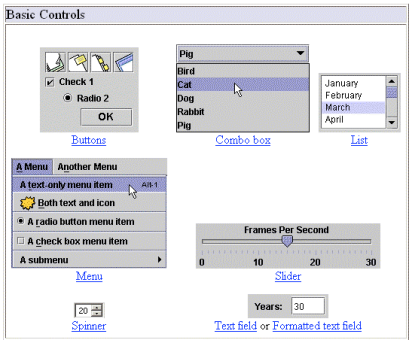
General Purpose Containers



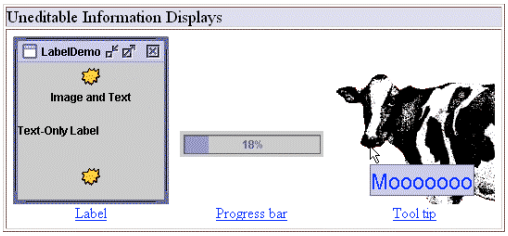
Special Purpose Containers



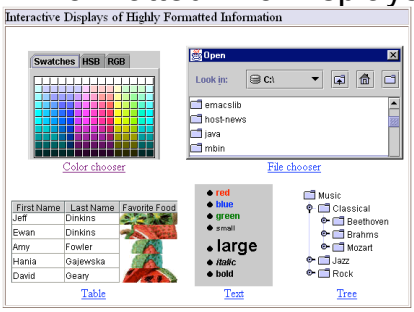
Basic Controls



Uneditable Info Displays



Formatted Info Displays



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

Kan vi göra något åt utseendet?

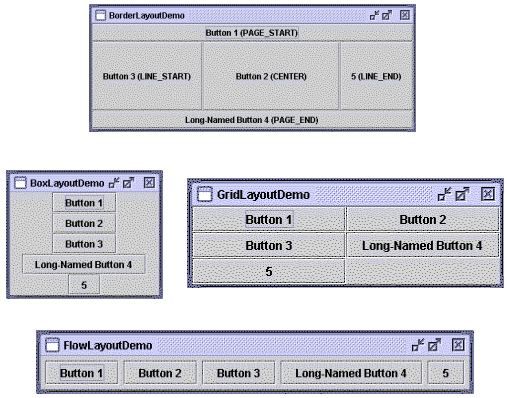
- Ja, det kan vi!
- Swing kan få ”native look & feel” om vi lägger till några rader kod…

```
// 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.



Your own components

- All Swing components are "lightweight" (drawn in Java), and you can make your own.
- http://java.sun.com/products/jfc/tsc/articles/painting
- Old approach: subclassing Canvas
  - Implement the `paint (Graphics g)` method
  - you get a rectangular shape covering the others
  - `new JComponent(){ public void paint(Graphics g){…} };` achieves the same, and more
  - paint() is called automatically by Swing when the container becomes visible
  - The Graphics object contains foreground and background color, as well as a clipping rectangle, in case not the whole component needs be drawn.

Your own components

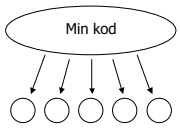
- for non-rectangular, or transparent components, you need to extend Component, Container or JComponent directly
  - implement the needed processXXX and addXXXListener
  - contains() for non-rectangular shapes
  - Implement isOpaque() or call setOpaque() to tell whether your component is transparent
  - as usual, paint() to draw your own component shape
  - If the component is opaque, paint() will need to cover all the area for which contains() is true

Drag and drop

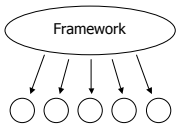
- Within your own Swing container, it is not difficult to show a drag and drop action
  - A transparent container on top, on which you draw the dragged object shape and move it with the mouse cursor
  - Still this is not recommended since Java 1.4 when dnd was reworked
- The standard swing drag and drop will work with
  - Standard Swing components (dragging from a table)
  - Objects dragged from or dropped to other applications
- http://java.sun.com/docs/books/tutorial/uiswing/dnd
- Drag and drop uses common concepts with copy/paste

Frameworks

Koda med klassbibliotek



Koda med framework



•What are the most important types of Swing callbacks?
- event listener methods like windowClosing()
- paint() method
•Other callbacks in Java? yes, finalize(), Observer's notify()
•Fundament: we implement the callbacks, but never call them. We wait for the framework to call them.

GUI-byggare

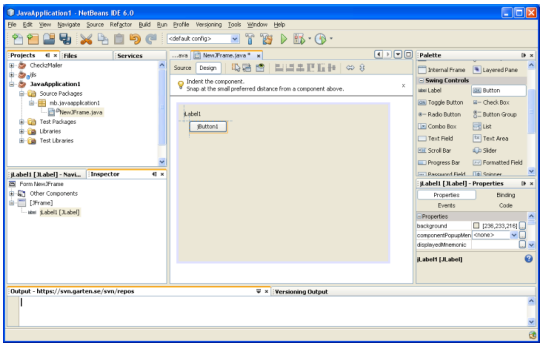
- Ofta integrerade med en **IDE**, Integrated Development Environment
- GUI:er består oftast av widgets, samt kod som skickar meddelanden mellan dessa
- GUI-byggare: "rita" gränssnittet, kod som skapar widgets och deras layout genereras automatiskt!
- Sedan lägger man till händelselyssnare etc. själv, ofta via någon form av **property inspector**.

Några GUI-byggare:

- **Visual Studio .NET Forms Designer**, www.microsoft.com
- **Borland C++ Builder**, www.borland.com
- **QT Designer**, www.trolltech.com
- **NetBeans**, www.netbeans.org
- Olika plug-ins för **Eclipse**, www.eclipse.org

- Men det finns många, många fler!

NetBeans 6.0



Lab2

- Implement the production planner
- Model: tasks, task planning dates/lines, task rules:
  - Tasks can't collide on a line / date interval
  - Move tasks automatically to earliest possible dates
- Views: task planning chart, task table

Designmönster (design patterns)

- Utgår från arkitekten Christopher Alexanders arbete på 70-talet
- Alexander försökte se mönster i hur man löst återkommande problem inom arkitektur och skrev en bok där han beskriver lösningarna
- Alexander, C. (1977). **A Pattern Language**. Oxford University Press
- Idén togs upp på allvar inom systemdesign i mitten av 90-talet
- Den mest kända boken är Gamma et al. (1995). **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley

Design patterns

- Namn och generell typ
- Intent (vad den gör)
- Also Known As
- Motivation
- Applicability
- Structure (ett UML-diagram)

Design patterns

- Participants (beskr. av klasser som ingår)
- Collaborations (hur klasserna arbetar tillsammans)
- Consequences (av att använda mönstret)
- Implementation
- Sample code
- Known uses
- Related patterns