Graphical User Interface (GUI), Part 1

- Applets
- The Model-View-Controller GUI Architecture
  - Separated Model Architecture
- Abstract Windowing Toolkit (AWT)
- Java Foundation Classes (JFC)

- Note this is a huge area many books are devoted solely to this topic.
- Here we will provide an overview and how to get started.
Applet

• An *applet* (application-let) is a Java program that runs in an internet browser.

• Characteristics of an Applet
  - Typically a smaller application.
  - Consists of a user interface component and various other components.
  - Program is downloaded.
    - Does not require any software to be installed on the client machine.
  - Has the methods `init`, `start`, `stop`, and `destroy`.
    - Called by the system not called by the programmer.
  - Show in an HTML page
    - Has a special `<APPLET>` tag for this.
  - Runs "inside the sandbox" => much more safe, no viruses.

• Applets are displayed through a browser or through the applet viewer (a JDK tool).
Applet, cont

```html
<applet code="MyClass.class"
       codebase="http://www.myHome.com"
       archive="MyJarFile.jar"
       height="100"
       width="200">
</applet> <!-- never omitted -->
```

- Deprecated in HTML 4.0 (and XHTML), widely supported.
- Replaced by the `<object>` tag.
- For details on applets see package `javax.swing.JApplet` and `java.applet.Applet`. 
Applet, cont

From `java.applet.Applet`.

- `init()` . Called by the browser or applet viewer to inform this applet that it has been loaded into the system.

- `start()` . Called by the browser or applet viewer to inform this applet that it should start its execution, e.g., when visible in browser.

- `stop()` . Called by the browser or applet viewer to inform this applet that it should stop its execution, e.g., when applet becomes invisible in browser.

- `destroy()` . Called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated.
Model-View-Controller Design

- Swing architecture is rooted in the *model-view-controller* (MVC) design (from the programming language SmallTalk).
- In the MVC architecture a visual application is broken up into three separate parts.
  - A *model* that represents the data for the application.
  - A *view* that is the visual representation of that data.
  - A *controller* that takes user input on the view and translates that to changes in the model.
Model-View-Controller, cont.

- **Philosophy**: Don't call use, we call you! (event driven).

- **Model**
  - The core logic of the program that processes data independent of the GUI.

- **View**
  - Presentation of the model.
  - There can be several views on the same model.
  - Output to screen.

- **Controller**
  - Input from devices such as keyboard and mouse.
  - Effect the model directly and the view indirectly.

- However, to strict so Java uses a modified MVC model.
Separated Model Architecture

- Based on the MVC Architecture.
  - Merge the view and controller parts into a single User-Interface (UI) part.
  - Very difficult to write a generic controller that does not know the specifics about a view.
  - Center an application around its data not its user interface.
// the model class
class Model {
    private int x;
    private int y;
    public Model () { x = 0; y = 0; }
    public int  getX() {return x;}
    public void setX(int x) {this.x = x;}
    public int  getY() {return y;}
    public void setY(int y) {this.y = y;}
    public int  calc() {return x*y;}
}
Separated Model Architecture, Example

// the view class
public class MVC1 extends JApplet {
    Model model = new Model();
    JLabel xl = new JLabel("x");
    JTextField x = new JTextField(10);
    JLabel yl = new JLabel("y");
    JTextField y = new JTextField(10);
    JLabel prodl = new JLabel("prod");
    JTextField prod = new JTextField(10);
    JButton calc = new JButton("Calculate");

    /* see next slide for ActionListener */
    AL al = new AL();
    public void init() {
        Container cp = getContentPane();
        cp.setLayout (new GridLayout(4,2)); // change layout
        cp.add(xl);    cp.add(x);
        cp.add(yl);    cp.add(y);
        cp.add(prodl); cp.add(prod);
        cp.add(calc);
        calc.addActionListener(al); // add action list
    }
}
// the controller class
class AL implements ActionListener {
    public void actionPerformed (ActionEvent e){
        int xValue = Integer.parseInt(x.getText());
        model.setX(xValue);
        int yValue = Integer.parseInt(y.getText());
        model.setY(yValue);
        String temp = Integer.toString(model.calc());
        prod.setText(temp);
    }
}

public static void main(String[] args) {
    MVC1 applet = new MVC1();
    JFrame frame = new JFrame("MVC");
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(applet);
    frame.setSize(150,150);
    applet.init();
    applet.start();
    frame.setVisible(true);
}
The HTML File

<html><head><title>Applet1</title></head><hr>

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
     width="100" height="50" align="baseline"
     codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-1_2_2-win.cab#Version=1,2,2,0">
     <PARAM NAME="code" VALUE="MVC1.class">
     <PARAM NAME="codebase" VALUE=".">
     <PARAM NAME="type" VALUE="application/x-java-applet;version=1.2.2">
     <COMMENT>
     <EMBED type="application/x-java-applet;version=1.2.2"
           width="200" height="200" align="baseline"
           code="Applet1.class" codebase=".">
           pluginspage="http://java.sun.com/products/plugin/1.2/plugin-install.html">
     </NOEMBED>
     </COMMENT>
     No Java 2 support for APPLET!!
     </NOEMBED>
</OBJECT>
<hr></body></html>
GUI Overview

• To create a Java GUI, you need to understand
  ▪ Containers
  ▪ Event
  ▪ Event Handlers
  ▪ Layout managers
  ▪ Components
  ▪ Special features
AWT and JFC/Swing

• Early Java development used graphic classes defined in the Abstract Windowing Toolkit (AWT).
  ▪ See the packages `java.awt`

• In Java 2, JFC/Swing classes were introduced.
  ▪ See the packages `javax.swing`

• Many AWT components have improved Swing counterparts.
  ▪ An example, the AWT `Button` class corresponds to a more versatile Swing class called `JButton`.

• Swing does not generally replace the AWT; still use for AWT events and the underlying AWT event processing model.

• Here we focus mostly on Swing.
Containers

• A container is a special component that can hold other components.

• The AWT `Applet` class, as well as the Swing `JApplet` class, are containers.

• Other containers include
  - Frames
    - A frame is a container that is free standing and can be positioned anywhere on the screen.
    - Frames give the ability to do graphics and GUIs through applications and applets.
  - Dialog boxes
  - Panels
  - Panes
  - Toolbars
Containers (Top Level and General)

- Applet
- Dialog
- Frame
- Panel
- Scroll Pane
- Split Pane
- Tabbed Pane
- Toolbar

[Source: java.sun.com]
Special Containers

Internal frame

Layered pane

Root pane

[Source: java.sun.com]
Events

- Every time the user types a character or pushes a mouse button, an event occurs.
- Any object can be notified of the event.
- All the objects have to do implement the appropriate interface and be registered as an event listener on the appropriate event source.

```java
Button

ActionEvent

ActionListener
// code
// do stuff
...
Events, cont.

- Several events implemented in `java.awt.AWTEvent` subclasses (`java.awt.Event` is deprecated).
  - Defines a lot of constants.

```java
public abstract class AWTEvent extends EventObject {
    public void setSource(Object newSource);
    public int getID();
    public String toString();
    public String paramString();
    protected void consume();
    protected boolean isConsumed();
}
```
Events Handlers

• In the declaration for the event handler class, one line of code specifies that the class either implements a listener interface (or extends a class that implements a listener interface).
  
  ```java
  public class MyClass implements ActionListener
  ```

• In the event handler class the method(s) in the listener interface must be implemented
  
  ```java
  public void actionPerformed(ActionEvent e) {
      // code that "reacts" to the event
  }
  ```

• Register an instance of the event handler class as a listener on one or more components.
  
  ```java
  myComponent.addActionListener(myClassInstance)
  ```
Events Handlers, cont.

```java
class AL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int xValue = Integer.parseInt(x.getText());
        model.setX(xValue);
        int yValue = Integer.parseInt(y.getText());
        model.setY(yValue);
        String temp = Integer.toString(model.calc());
        prod.setText(temp);
    }
}
```

- Often an event handler that has only a few lines of code is implemented using an *anonymous inner class*.
Events Handlers, cont.

- **SwingApplication** has two event handlers.
  - Window closing (window events).
    
    ```java
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    ```
  - Button clicks (action events).
    
    see previous slide.

- Types of events (listeners defined in `java.awt.event`)
  - Click button ⇒ `ActionListener`
  - Close frame ⇒ `WindowListener`
  - Press mouse button ⇒ `MouseListener`
  - Move mouse ⇒ `MouseMotionListener`
  - Component visible ⇒ `ComponentListener`
  - Component gets focus ⇒ `FocusListener`
WindowListener and MouseListener

```java
public interface WindowListener extends EventListerner {
    void windowActivated(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowClosing(WindowEvent e);
    void windowDeactivated(WindowEvent e);
    void windowDeiconified(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowOpened(WindowEvent e);
}
```

```java
public interface MouseListener extends EventListerner {
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```
A layout manager is an object that determines the manner in which components are displayed in a container.

There are several predefined layout managers defined in the Java standard class library:

- Flow Layout (in java.awt)
- Border Layout (in java.awt)
- Card Layout (in java.awt)
- Grid Layout (in java.awt)
- GridBag Layout (in java.awt)
- Box Layout (in javax.swing)
- Overlay Layout (in javax.swing)
Layout Managers, cont.

• Every container has a default layout manager, but we can also explicitly set the layout manager for a container.
• Each layout manager has its own particular rules governing how the components will be arranged.
• Some layout managers pay attention to a component's preferred size or alignment, and others do not.
• The layout managers attempt to adjust the layout as components are added and as containers are resized.
Flow Layout

• A flow layout puts as many components on a row as possible, then moves to the next row.

• Rows are created as needed to accommodate all of the components.

• Components are displayed in the order they are added to the container.

• The horizontal and vertical gaps between the components can be explicitly set.

• Default for JPanel.
Border Layout

- A border layout defines five areas into which components can be added.
- The default for most GUIs
Box Layout

- A box layout organizes components either horizontally (in one row) or vertically (in one column).
- Special rigid areas can be added to force a certain amount of spacing between components.
- By combining multiple containers using box layout, many different configurations can be created.
- Multiple containers with box layouts are often preferred to one container that uses the more complicated gridbag layout manager.
Other Layout Managers

**Card layout.** The area contains different components at different times.

**Gridbag layout.** The most sophisticated and flexible.

**Grid layout.** All equal size in a grid.
"Atomic" Components

• The root in the component hierarchy is `JComponent`.
• The `JComponent` provides the following functionality to its descendants, e.g., `JLabel`, `JRadioButton`, and `JTTextArea`.
  - Tool tips
  - Borders
  - Keyboard-generated actions
  - Application-wide pluggable look and feel
  - Various properties
  - Support for layout
  - Support for accessibility
  - Double buffering
Basic Components

Button

Combo Box

Menu

Slider

List

Text Field

[Source: java.sun.com]
Non-Editable Displays

- Label
- Progress bar
- Tool tip

[Source: java.sun.com]
Editable Displays

File Chooser

Color Chooser

Table

Text

Tree

[Source: java.sun.com]
Summary

• The Model-View-Controller GUI Architecture
  ▪ Separated Model Architecture
• Abstract Windowing Toolkit (AWT)
• Java Foundation Classes (JFC)