Lecture VII
SWING (II)

Reference: David Geary, Graphic Java, Vols.1 and 2.

All modern GUI systems are based on call-back functions which are called when events occur. Java is no different. But the original AWT architecture has many defects that were remedied by the Swing Architecture. But it does not mean that AWT is deprecated – it cannot be since Swing is built on top of AWT.

Swing will work on both Netscape Navigator (version 4.04 or later) as well as Microsoft Internet (version 4.0 or later). But it may involve some effort and bugs may still occur. For this reason, we generally prefer AWT for a multiplatform applet applications. In the appendix below, we give some instructions for viewing Swing applets on browsers.

A key idea here is the notion of **pluggable look and fell**. The intuitive idea is that when you are in an environment (e.g., Macs, or Windows, or Solaris), they provide a certain consistent visual presentation (i.e., look), and a certain behavior (i.e., feel). If you want Swing GUI's to be able to duplicate all such look-and-feels, you need a "pluggable" architecture. The problem with AWT 1.0 is that the peer-based architecture takes away much of this flexibility.

The second key concept in Swing is the **Model-View-Controller** (MVC) Design paradigm.
Model: This is the logical structure of the data and their associated values (state information). The model provides methods to access and modify these values.
View: This gives a visual presentation of the model. Each view has it own set state information. It is the "look" of look-and-feel. In Swing, you can identify this with the Components of Swing. What is important is that the view parameters are maintained somewhat independently of the model, and you can have multiple views.
Controller: This handle events and mediates between the Model and View. It is the "feel" of look-and-feel. In Swing, you can identify these with the listeners.

§1. The Old Event Model

The original AWT Event Model (version 1.02 and earlier) is said to be **inheritance-based**. The new model (version 1.1 and later) is said to be **delegation-based**. Since the new model is backwards compatible, it is worthwhile briefly summarizing the old model:

- If an event occurs in a component, a method of that component is called. E.g., the `setSize(w,h)` method of a component is called when the component is resized.
- Methods like `setSize(w,h)` are specific to an event type. There is one method called `HandleEvent(Event e)` that responds to a whole class of events (e.g. Mouse, Window, Keyboard, etc). By definition, events handled by `handleEvent(e)` are called **propagated events**. This method returns a boolean. If true is returned, it signifies that the handleEvent method has completely dealt with the event. If false, the event would be propagated to the component container's `handleEvent` method. Hence the name "propagated events".
- Direct implementation of handleEvent is tedious. There is a default implementation that checks the type of the event, and calls special methods (action(...), mouseUP(...), mouseDown(...), etc). The default implementation of the special methods are null, so you just have to override this default.

§2. Delegation Event Model

We now summarize the AWT 1.1 model for events. Instead of insisting

- The basic idea is that Components can fire events. Hence Components are known as **event sources**. (Java has 2 event source interfaces: Adjustable and ItemSelectable p.249, Geary).
- Events are responded to by **listeners**. The listeners must be registered with the event source (i.e., Component), typically by invoking the Component's method called `addXXXListener( XXXLister )`, where `XXX` is the name of the event. For instance, action events can be registered using `addActionListener( ActionListener )`. You can also `removeXXXListener(...)`
- When an event occurs in a Component, all the listeners for that event is invoked (in some non-deterministic order). The various listeners have suitable interfaces so that appropriate methods in the listeners are called. The listener methods are passed a reference to an Event object: this object can be queried for the event source, an ID, time, position, etc.
- Thus most events are "multicast". If you want to make a certain event "unicast", you can do so when you register the first listener for that event (p.248, Geary).

§3. Listener Interfaces

There are 12 listener interfaces, with the appropriate methods. To avoid clutter, we simply note the names of the methods:

Listener Type	**Methods**	**Event Classification**
ActionListener:	actionPerformed	Semantic Event
AdjustmentListener:	adjustmentValueChanged	Semantic Event
ComponentListener:	componentHidden, componentMoved, componentResized, componentShwon	Component Event
ContainerListener:	componentAdded, componentRemoved	Component Event
FocusListener:	focusGained, focusLost	Component Event
InputMethodListener:	caretPositionChanged, inputMethodTextChanged	Component Event
ItemListener:	itemStateChanged	Semantic Event
KeyListener:	keyTyped, keyPressed, keyReleased	Component Event
MouseListener:	mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased	Component Event
MouseMotionListener:	mouseDragged, mouseMoved	Component Event
TextListener:	textValueChanged	Semantic Event
WindowListener:	windowActivated, windowDeactivated, windowClosed, windowOpened, windowClosing, windowIconified, windowDeiconified	Component Event

As usual, to help user implement only selected methods easily, we have adapters for the above interfaces. Thus we have default implementations called `XXXAdapter` for `XXXListener` interface. E.g., `WindowAdapter` for E.g., `WindowListener`. (However, for the 4 interfaces with only one method each, there are no adapters.)

§4. Mouse Event Example

The following applet shows a Button, and whenever the mouse enters or exits the button area, an appropriate message is printed on the screen.

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

public class ButtonTest1
extends ApplicationFrame {

    Label lab = new Label("Initial");

    ButtonTest1(String s) {
        super(s);
        Button button = new Button("Press Me");
        button.addMouseListener(new ButtonMouseListener());

setBackground(Color.blue);

setLayout(new BorderLayout());
        add(button, "North");
add(lab, "South");
setVisible(true);
    }

    public static void main(String[] args) {
        ButtonTest1 myBut = new ButtonTest1("Press Me");
    }

   class ButtonMouseListener
    extends MouseAdapter {
       public void mouseEntered(MouseEvent event) {
           System.out.println("Mouse Entered Button");
   lab.setText("Mouse Entered Button");
       }
       public void mouseExited(MouseEvent event) {
           System.out.println("Mouse Exited Button");
   lab.setText("Mouse Exited Button");
       }
       public void mouseClicked(MouseEvent event) {
   dispose();
           System.exit(0);
       }
   }//ButtonMouseListener Class
}
```

CLICK HERE for source code.
CLICK HERE for an applet demo

§5. Border in AWT

Using Swing, you have a large selection of borders to choose from. In AWT, we can do simple borders (as you will need in HW3):

```
/*
```

```
 * @source: p.339, Geary vol.1
 */

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class ValidateApplet extends Applet {
private GrayPanel grayPanel = new GrayPanel();
public void init() {
add(grayPanel);
}
}
class GrayPanel extends Panel implements ActionListener {
private TextField  field    = new TextField("TextField");
private Button     lgButton = new Button   ("larger font");
private Button     smButton = new Button   ("smaller font");

public GrayPanel() {
lgButton.addActionListener(this);
smButton.addActionListener(this);

add(field);
add(lgButton);
add(smButton);
setBackground(Color.gray);
}
public void actionPerformed(ActionEvent event) {
Button button  = (Button)event.getSource();
Font   curFont = field.getFont();
int    newSize = curFont.getSize();

if(button == lgButton) newSize += 3;
if(button == smButton) newSize -= 3;

field.setFont(new Font(curFont.getFamily(),
                       curFont.getStyle(), newSize));
}
public void paint(Graphics g) {
Graphics2D g2 = (Graphics2D)g;
Stroke stroke = new BasicStroke( 8,
BasicStroke.CAP_BUTT,
BasicStroke.JOIN_ROUND
);
g2.setStroke(stroke);
g2.setColor(Color.green);
g2.drawRect(0,0,getSize().width-2,getSize().height-2);
}
}//GreyPanel
```

CLICK HERE for source code.
CLICK HERE for an applet demo

§6. Classification of Events

In the above table, the 3rd column classifies all events into either **Component Events** or **Semantic Events**.

Component events are subclassified into **input events** or not. Input events can be "consumed", and are not passed to the component's peers. Intuitively, the non-input component events are "output events".

Semantic events are not necessarily triggered by atomic actions, and depends on the class. E.g., lists fire action events when their items are double-clicked, while textfields fire action events when "enter" is typed.

§7. APPENDIX: Using Swing in Browsers

Swing is included in 1.2 release of JDK, but for 1.1 release, you need to download Swing from the URL http://java.sun.com/products/jfc/.

NETSCAPE NAVIGATOR: You need version 4.04 or later, and have 1.1 JDK patch installed. Go to http://developer.netscape.com/software/jdk/download.html. Once you have the Swing jar files, you must make it accessible to Netscape: either copy the file to a directory known to Netscape, or modify the system CLASSPATH variable. The following directory is known to Netscape navigator: it is usually `C:\ProgramFiles\Netscape\Communicator\Program\Java\Classes`. To set the CLASSPATH variable, see below.

INTERNET EXPLORER: You need Version 4.0 or later. You just need to tell IE where your Swing jar files are located. This you do by setting the system CLASSPATH variable.
Windows NT: go to the System icon in the control panel. Under System Properties, click the Environment tab. Add CLASSPATH variable to the lower list entitled "User Variables for Administrator". There are two text fields named "Variable:" and "Value:" for entering this information. The CLASSPATH variable should include the "classes.zip" files for JDK and "swingall.jar" file. The Variable should be CLASSPATH, and as an example of the Value, we have `D:\jdk\lib\classes.zip;D:\swing\swingall.jar`. (Note that the various paths are colon separated, and we have the classes in the D: drive in this example.) Reboot the system.

For some applications, you may further need to use a Java Plug-in from Sun.

END OF LECTURE