Swing

Advanced Java Programming

Points to be discussed

- **JFC**
- **MVC Architecture; difference between AWT and Swing**
- **Components from javax.swing package – Jcomponent,**

  JFrame, JWindow, JLabel, JButton, JTextComponent,

  JToggleButton, JradioButton, JCheckbox ;

- **Pluggable Look and Feel**

JFC (JAVA FOUNDATION CLASSES)

JFC stands for Java Foundation Classes. JFC is the set of GUI components that simplify desktop Applications. Many programmers think that JFC and Swing are one and the same thing, but that is not so. JFC contains Swing [A UI component package] and quite a number of other items:

- Cut and paste: Clipboard support.
- Accessibility features: Aimed at developing GUIs for users with disabilities.
- The Desktop Colors Features were first introduced in Java 1.1
- Java 2D: it has Improved colors, images, and text support.

What is swing ?

- Swing is a Set of API (API- Set of Classes and Interfaces)
- Swing is Provided to Design Graphical User Interfaces
- Swing is an Extension library to the AWT (Abstract Window Toolkit) 5:00 – 5:30 pm
- Includes New and improved Components that have been enhancing the looks and Functionality of GUIs'
- Swing can be used to build (Develop) The Standalone swing GUI Apps as Servlets and Applets
- It Employs model/view design architecture.
- Swing is more portable and more flexible than AWT, the Swing is built on top of the AWT.
- Swing is Entirely written in Java.
- Java Swing Components are Platform-independent, and The Swing Components are lightweight.
- Swing Supports a Pluggable look and feel and Swing provides more powerful components.
- such as tables, lists, Scroll Panes, Color Chooser, tabbed pane, etc.
- Swing Follows MVC

MVC Architecture

- Swing uses the *model-view-controller architecture* (MVC) as the fundamental design behind each of its components. Essentially, MVC breaks GUI components into three elements. shows how the model, view, and controller work together to create a scrollbar component.

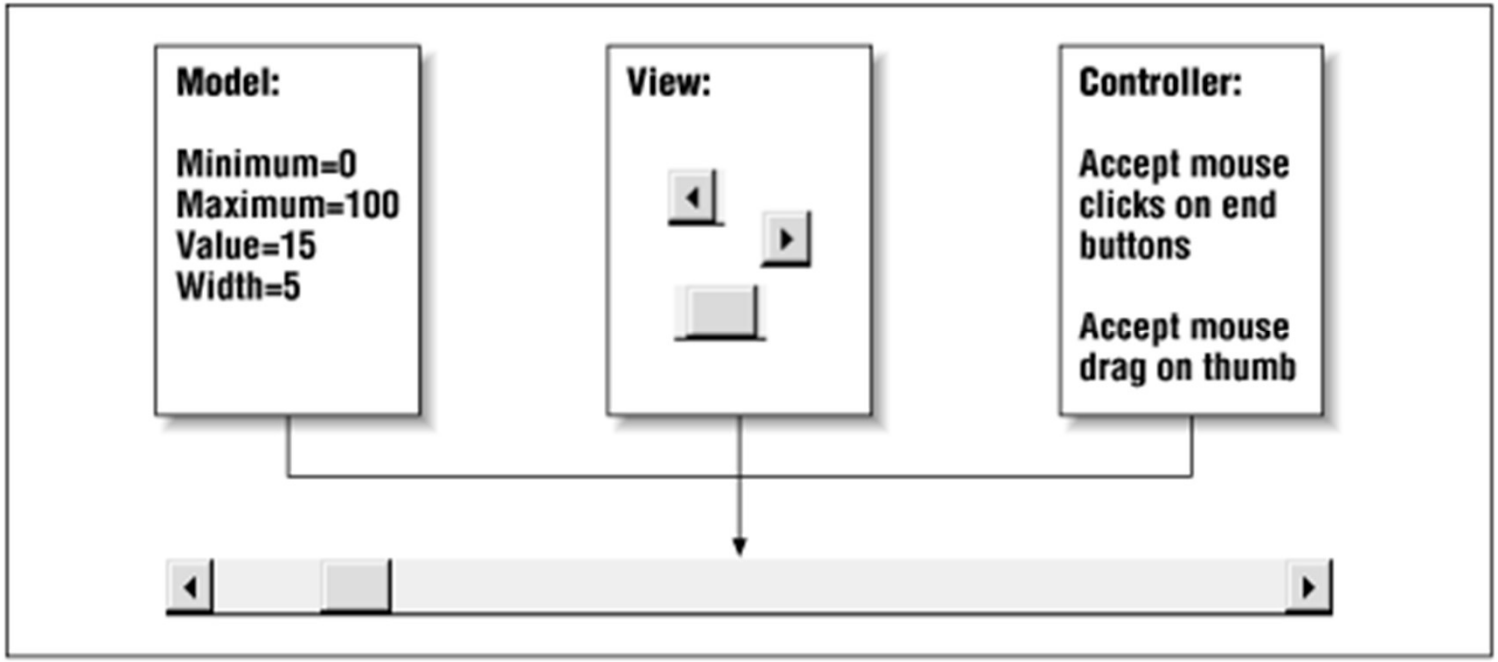


Image source

MVC Architecture

- **Model**

  The model encompasses the state data for each component. There are different models for different types of components. For example, the model of a scrollbar component might contain information about the current position of its adjustable “thumb,” its minimum and maximum values, and the thumb’s width (relative to the range of values). A menu, on the other hand, may simply contain a list of the menu items the user can select from. Note that this information remains the same no matter how the component is painted on the screen; model data always exists independent of the component’s visual representation.

MVC Architecture

View :

The view refers to how you see the component on the screen. For a good example of how views can differ, look at an application window on two different GUI platforms. Almost all window frames will have a titlebar spanning the top of the window. However, the titlebar may have a close box on the left side (like the older MacOS platform), or it may have the close box on the right side (as in the Windows 95 platform). These are examples of different types of views for the same window object.

MVC Architecture

Controller :

The controller is the portion of the user interface that dictates how the component interacts with events. Events come in many forms — a mouse click, gaining or losing focus, a keyboard event that triggers a specific menu command, or even a directive to repaint part of the screen. The controller decides how each component will react to the event—if it reacts at all.

AWT vs SWING

Java AWT	Java swing
Java AWT is an API to develop GUI applications in Java.	Swing is a part of Java Foundation Classes and is used to create various applications.
Components of AWT are heavy weighted.	The components of Java Swing are lightweight.
Components are platform dependent.	Components are platform independent.
Execution Time is more than Swing.	Execution Time is less than AWT.
AWT components require java.awt package.	Swing components requires javax.swing package.

SWING Components

Class	Description
Component	A Component is the Abstract base class for about the non-menu user-interface controls of Java SWING. Components are representing an object with a graphical representation.
Container	A Container is a component that can container Java SWING Components
JComponent	A JComponent is a base class for all swing UI Components In order to use a swing component that inherits from JComponent, the component must be in a containment hierarchy whose root is a top-level Java Swing container.
JLabel	A JLabel is an object component for placing text in a container.
JButton	This class creates a labeled button.

SWING Components

Class	Description
JColorChooser	A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color.
JCheckBox	A JCheckBox is a graphical (GUI) component that can be in either an on-(true) or off-(false) state.
JRadioButton	The JRadioButton class is a graphical (GUI) component that can be in either an on-(true) or off-(false) state. in the group
JList	A JList component represents the user with the scrolling list of text items
JComboBox	A JComboBox component is Presents the User with a show up Menu of choices.

SWING Components

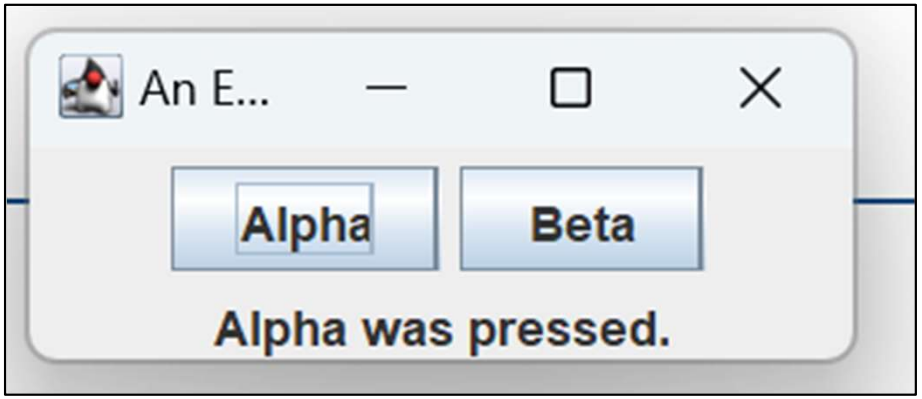
Class	Description
JTextField	A JTextField object is a text component that will allow for the editing of a single line of text.
JPasswordField	A JPasswordField object it is a text component specialized for password entry.
JTextArea	A JTextArea object is a text component that allows for the editing of multiple lines of text.
JToggleButton	When the user presses the toggle button, it toggles between being pressed or unpressed. JToggleButton is used to select a choice from a list of possible choices. Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

SWING Event Handling

EventDemo.java

```java
// Handle an event in a Swing program.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class EventDemo {
 JLabel jlab;
 EventDemo() {
 // Create a new JFrame container.
 JFrame jfrm = new JFrame("An Event Example");
 // Specify FlowLayout for the layout manager.
 jfrm.setLayout(new FlowLayout());
 // Give the frame an initial size.
 jfrm.setSize(220, 90);
 // Terminate the program when the user closes the application.
 jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 // Make two buttons.
 JButton jbtnAlpha = new JButton("Alpha");
 JButton jbtnBeta = new JButton("Beta");
 // Add action listener for Alpha.
 jbtnAlpha.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent ae) {
 jlab.setText("Alpha was pressed.");
 }
 });
 // Add action listener for Beta.
 jbtnBeta.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        jlab.setText("Beta was pressed.");
        }
        });
        // Add the buttons to the content pane.
        jfrm.add(jbtnAlpha);
        jfrm.add(jbtnBeta);
        // Create a text-based label.
        jlab = new JLabel("Press a button.");
        // Add the label to the content pane.
        jfrm.add(jlab);
        // Display the frame.
        jfrm.setVisible(true);
        }public static void main(String args[]) {
            // Create the frame on the event dispatching thread.
            SwingUtilities.invokeLater(new Runnable() {
            public void run() {
            new EventDemo();
            }
            });
            }
           }
```

Snipped



Pluggable look and feel

Swing is **GUI Widget Toolkit** for Java. It is an API for providing Graphical User Interface to Java Programs.

Unlike AWT, Swing components are written in Java and therefore are platform-independent.

Swing provides platform specific Look and Feel and also an option for pluggable Look and Feel, allowing application to have Look and Feel independent of underlying platform.

Pluggable look and feel

“Look” refers to the appearance of GUI widgets and “feel” refers to the way the widgets behave

. Sun’s JRE provides the following L&Fs:

1. **CrossPlatformLookAndFeel:** this is the “Java L&F” also known as “Metal” that looks the same on all platforms. It is part of the Java API (javax.swing.plaf.metal) and is the default.
2. **SystemLookAndFeel:** here, the application uses the L&F that is default to the system it is running on. The System L&F is determined at runtime, where the application asks the system to return the name of the appropriate L&F. For Linux and Solaris, the System L&Fs are “GTK+” if GTK+ 2.2 or later is installed, “Motif” otherwise. For Windows, the System L&F is “Windows”.
3. **Synth:** the basis for creating your own look and feel with an XML file.
4. **Multiplexing:** a way to have the UI methods delegate to a number of different look and feel implementations at the same time.

Pluggable look and feel

We can use UIManager to load the L&F class directly from classpath. For which the code goes like this:

```
UIManager.setLookAndFeel("fully qualified name of look and feel");
```

For example, following code changes application Look and Feel to Motif Look And Feel:

```
UIManager.setLookAndFeel ("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```

```
// Java sample code to get the list of
// installed Look and Feel themes, here is a sample code:
import javax.swing.UIManager;

public class MainClass {    public static void main(String[] a)
    {
        UIManager.LookAndFeelInfo[] looks =
UIManager.getInstalledLookAndFeels();
        for (UIManager.LookAndFeelInfo look : looks)
            System.out.println(look.getClassName());
        }
    }
}
```

```
C:\Users\Deepti\Desktop>java MainClass
javax.swing.plaf.metal.MetalLookAndFeel
javax.swing.plaf.nimbus.NimbusLookAndFeel
com.sun.java.swing.plaf.motif.MotifLookAndFeel
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel
```