To support the Model-Delegate architecture, most Swing components contain two objects. The first represents the model. The second represents the UI delegate. Models are defined by interfaces. For example, the model for a button is defined by the ButtonModel interface. UI delegates are classes that inherit ComponentUI.

## 1. Components and Containers

A Swing GUI consists of two key items: components and containers.

- A component is an independent visual control, such as a push button or slider.
- A container is a special type of component that is designed to hold other components.

In order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container. Because containers are components, a container can also hold other containers.

### 1.1 Components

In general, Swing components are derived from the JComponent class. All of Swing's components are represented by classes defined within the package javax.swing.

| | | | |
|---|---|---|---|
| JApplet (Deprecated) | JButton | JCheckBox | JCheckBoxMenuItem |
| JColorChooser | JComboBox | JComponent | JDesktopPane |
| JDialog | JEditorPane | JFileChooser | JFormattedTextField |
| JFrame | JInternalFrame | JLabel | JLayer |
| JLayeredPane | JList | JMenu | JMenuBar |
| JMenuItem | JOptionPane | JPanel | JPasswordField |
| JPopupMenu | JProgressBar | JRadioButton | JRadioButtonMenuItem |
| JRootPane | JScrollBar | JScrollPane | JSeparator |
| JSlider | JSpinner | JSplitPane | JTabbedPane |
| JTable | JTextArea | JTextField | JTextPane |
| JTogglebutton | JToolBar | JToolTip | JTree |
| JViewport | JWindow | | |

## 1.2 Containers

Swing defines two types of containers.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

The first are top-level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent. The top-level containers are heavyweight. This makes the top-level containers a special case in the Swing component library. A top-level container is not contained within any other container.

The second types of containers supported by Swing are lightweight containers. Lightweight containers do inherit JComponent. Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container. Thus, you can use lightweight containers such as JPanel to create subgroups of related controls that are contained within an outer container.

## 2. The Top-Level Container Panes

Each top-level container defines a set of panes. At the top of the hierarchy is an instance of JRootPane. JRootPane is a lightweight container whose purpose is to manage the other panes. It also helps manage the optional menu bar. The panes that comprise the root pane are called the glass pane, the content pane, and the layered pane.

The glass pane is the top-level pane. The glass pane enables you to manage mouse events that affect the entire container.

The layered pane is an instance of JLayeredPane. The layered pane allows components to be given a depth value. This value determines which component overlays another.

## 3. A Simple Swing Application

It uses two Swing components: JFrame and JLabel. JFrame is the top-level container that is commonly used for Swing applications. JLabel is the Swing component that creates a label, which is a component that displays information. The label is Swing's simplest component because it is passive. That is, a label does not respond to user input. It just displays output. The program uses a JFrame container to hold an instance of a JLabel. The label displays a short text message.

Swing programs are compiled and run in the same way as other Java applications. Thus, to compile this program, you can use this command line:

javac SwingDemo.java

To run the program, use this command line:

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

```
// A simple Swing application.

import javax.swing.*;

class SwingDemo {

  SwingDemo() {

    // Create a new JFrame container.
    JFrame jfrm = new JFrame("A Simple Swing Application");

    // Give the frame an initial size.
    jfrm.setSize(275, 100);

    // Terminate the program when the user closes the application.
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Create a text-based label.
    JLabel jlab = new JLabel(" Swing means powerful GUIs.");

    // 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 SwingDemo();
      }
    });
  }
}
```


java SwingDemo

When the program is run, it will produce a window similar to that shown in

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

**Some important points:-**

- Closing the window causes the entire application to terminate. The general form of setDefaultCloseOperation( ) is shown here:

void setDefaultCloseOperation(int what)

The value passed in what determines what happens when the window is closed. There are several other options in addition to JFrame.EXIT_ON_CLOSE.

They are shown here:

DISPOSE_ON_CLOSE

HIDE_ON_CLOSE

DO_NOTHING_ON_CLOSE

- Inside main( ), a SwingDemo object is created, which causes the window and the label to be displayed. Notice that the SwingDemo constructor is invoked using these lines of code:

```
SwingUtilities.invokeLater(new Runnable() {
  public void run() {
    new SwingDemo();
  }
});
```

This sequence causes a SwingDemo object to be created on the event dispatching thread rather than on the main thread of the application.

To avoid problems (including the potential for deadlock), all Swing GUI components must be created and updated from the event dispatching thread, not the main thread of the application. However, main( ) is executed on the main thread. Thus, main( ) cannot directly instantiate a SwingDemo object. Instead, it must create a Runnable object that executes on the event dispatching thread and have this object create the GUI.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

To enable the GUI code to be created on the event dispatching thread, you must use one of two methods that are defined by the SwingUtilities class. These methods are invokeLater( ) and invokeAndWait( ). They are shown here:

a) static void invokeLater(Runnable obj)
b) static void invokeAndWait(Runnable obj) throws InterruptedException, InvocationTargetException

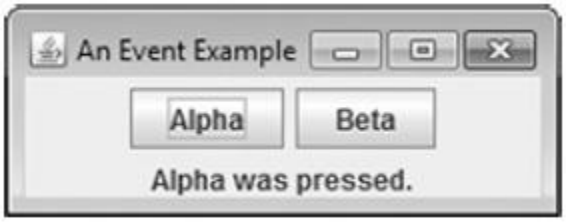
Here, obj is a Runnable object that will have its run( ) method called by the event dispatching thread. The difference between the two methods is that invokeLater( ) returns immediately, but invokeAndWait( ) waits until obj.run( ) returns.

# 4. Event Handling

The event handling mechanism used by Swing is the same as that used by the AWT. This approach is called the delegation event model.

Events specific to Swing are stored in javax.swing.event. Although events are handled in Swing in the same way as they are with the AWT, it is still useful to work through a simple example. The following program handles the event generated by a Swing push button.

```
// 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);
```




Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

```
    // 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();
      }
    });
  }
}
```

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)

## Explanation

First, notice that the program now imports both the java.awt and java.awt.event packages. The java.awt package is needed because it contains the FlowLayout class, which supports the standard flow layout manager used to lay out components in a frame. (See Chapter 26 for coverage of layout managers.) The java.awt.event package is needed because it defines the ActionListener interface and the ActionEvent class.

The EventDemo constructor begins by creating a JFrame called jfrm. It then sets the layout manager for the content pane of jfrm to FlowLayout. By default, the content pane uses BorderLayout as its layout manager.

In this example anonymous inner classes are used to provide the event handlers for the two buttons. Each time a button is pressed, the string displayed in jlab is changed to reflect which button was pressed.

Reference:- Herbert Schildt - Java_ The Complete Reference, Eleventh Edition 11(2019, McGraw-Hill Education)