Chapter 31

Introducing Swing

Limitations of AWT

- AWT defines basic set of controls, window and dialog boxes that provide limited GUI.
- Look and feel of AWT components is defined by platform not by JAVA.
  - Because of variation between OSs, a component might look or even act differently on different platforms.
- AWT components use native code, they are refered as *heavyweight.*
  - *Heavyweight* components are always opaque.

Swing

- Swing is built upon AWT (Abstract Window Toolkit) introduced in 1997.
- Swing eliminates a number of limitations inherent in AWT, Swing *does not* replace it.
- Swing also uses same event handling mechanism as the AWT.
- Two key features of Swing:
  - Lightweight components
  - A pluggable look and feel

Contd...

- Lightweight component means they are written entirely in JAVA and do not map directly to platform specific peers.
  - Lightweight components are more efficient and flexible.
  - As lightweight components do not translate into native peers, each component look and feel is consistent across all platform.

Contd...

- Swing supports a *pluggable look and feel* (PLAF). Because each swing component is rendered by java code rather than by native peers, the look and feel of a component is under control of swing.
  - Hence, it is possible to seperate the look and feel of a component from logic of the component.
  - This implies it is possible to change the way a component is rendered (plug-in a new look) without affecting any of its other aspects.
- Advantages of PLAF:
  - Consistent look and feel of components across all

Components and Containers

- A Swing GUI is consist of two key items:
  - Components
  - Containers
- Component is an independent visual control such as push button or slider.
- A container holds a group of components. So, Container is a special type of component that is designed to hold other components.
  - Since a container is also a component. So, A container can also hold other containers.
  - This enables Swing to define what is called a *containment hierarchy*

Components and Containers

- Swing components are derived from **JComponent** class. **JComponent** support pluggable look and feel. **JComponent** inherits AWT classes **Container** and **Component.**
- All of Swing's components are defined withing package **javax.swing**

Class names for Swing components

JApplet	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		

Container

- Swing defines two types of containers:
- Top-level containers:
  - **JFrame, JApplet, JWindow** and **JDialog**.
  - These containers are at top in containment heirarchy and they are heavyweight compare to other Swing's components.
  - Commonly used top level container is **Jframe.**
- Lightweight containers:
  - **JPanel**
  - These are inherited from **Jcomponent.**

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 user closes the application.
      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_C
```

Contd...

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

Contd...

- void setDefaultCloseOperation(int *what*):
  - The value in *what* determines what happens when the window is closed. There are several options for that:
    - DISPOSE_ON_CLOSE
    - HIDE_ON_CLOSE
    - DO_NOTHING_ON_CLOSE
    - EXIT_ON_CLOSE
  - Their name reflects their actions.

Contd...

- To add a component (ex. Label) to a container (ex. Frame), it needs to be added to frame's content pane.
    - General form to add a component is:
        - Component add(Component *comp*)
- By default the **JFrame** is invisible. So, **setVisible(true)** must be called to show that frame.
-

Contd...

```
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 main thread of the application. Because
    - In general, Swing programs are event-driven. For example, when a user interacts with a component, an event is generated. An event is passed to the application by calling an event handler defined by the application. However, the handler is executed on the event dispatching thread provided by Swing and not on the main thread of the application. Thus, although event handlers are defined by your program, they are called on a

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.

Contd...

```
// 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");
```

Contd...

```
// 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.");
}
});
```

Contd...

```
// 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
```