UNIT-3
Introducing Swing

4.1 The Origins and Design Philosophy of Swing

The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or peers. This means that the look and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as heavyweight.

The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. This potential variability threatened the overarching philosophy of Java: write once, run anywhere. Second, the look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed. Third, the use of heavyweight components caused some frustrating restrictions.

Two Key Swing Features: Swing was created to address the limitations present in the AWT. It does this through two key features: lightweight components and a pluggable look and feel. Together they provide an elegant, yet easy-to-use solution to the problems of the AWT. More than anything else, it is these two features that define the essence of Swing.

Swing Components Are Lightweight: With very few exceptions, Swing components are lightweight. This means that they are written entirely in Java and do not map directly to platform-specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. Furthermore, because lightweight components do not translate into native peers, the look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.

Swing Supports a Pluggable Look and Feel: 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 the control of Swing. This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what Swing does. Separating out the look and feel provides a significant advantage: it becomes possible to change the way that a component is rendered without affecting any of its other aspects. In other words, it is possible to ―plug in‖ a new look and feel for any given component without creating any side effects in the code that uses that component. Moreover, it becomes possible to define entire sets of look-and-feels that represent different GUI styles. To use a specific style, its look and feel is simply ―plugged in.‖ Once this is done, all components are automatically rendered using that style. Pluggable look-and-feels offer several important advantages. It is possible to define a look and feel that is consistent across all platforms. Conversely, it is possible to create a and feel that acts like a specific platform. For example, if you know that an application will be running only in a Windows environment, it is possible to specify the Windows look and feel. It is also possible to design a custom look and feel. Finally, the look and feel can be changed dynamically at run time.

Java SE 6 provides look-and-feels, such as metal and Motif, that are available to all Swing users. The metal look and feel is also called the Java look and feel. It is platform-independent and available in all Java execution environments. It is also the default look and feel. Windows environments also have access to the Windows and Windows Classic look and feel.

4.2 Components and Containers

A Swing GUI consists of two key items: components and containers. However, this distinction is mostly conceptual because all containers are also components. The difference between the two is found in their intended purpose: As the term is commonly used, a component is an independent visual control, such as a push button or slider. A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components. Furthermore, 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. This enables Swing to define what is called a containment hierarchy, at the top of which must be a top- level container.

Components: In general, Swing components are derived from the JComponent class. JComponent provides the functionality that is common to all components. For example, JComponent supports the pluggable look and feel. JComponent inherits the AWT classes Container and Component. Thus, a Swing component is built on and compatible with an AWT component. All of Swing‘s components are represented by classes defined within the package javax.swing. The following are the class names for Swing components.

JApplet, JButton, JCheckBox, JCheckBoxMenuItem, JColorChooser, JComboBox, JComponent, JDesktopPane, JDialog, JEditorPane, JFileChooser, JFormattedTextField, JFrame, JInternalFrame, JLabel, JLayeredPane, JList JMenu, JMenuBar and JMenuItem etc.

Notice that all component classes begin with the letter J. For example, the class for a label is JLabel; the class for a push button is JButton; and the class for a scroll bar is JScrollBar.

Containers

Swing defines two types of containers. The first are top-level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent. They do, however, inherit the AWT classes Component and Container. Unlike Swing‘s other components, which are lightweight, the top-level containers are heavyweight. This makes the top-level containers a special case in the Swing component library.

As the name implies, a top-level container must be at the top of a containment hierarchy. A top-level container is not contained within any other container. Furthermore, every containment hierarchy must begin with a top-level container. The one most commonly used for applications is JFrame. The one used for applets is JApplet. The second type of containers supported by Swing are lightweight containers. Lightweight containers do inherit JComponent. An example of a lightweight container is JPanel, which is a general-purpose container. 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.

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. It sits above and completely covers all other panes. By default, it is a transparent instance of JPanel. The glass pane enables you to manage mouse events that affect the entire container (rather than an individual control) or to paint over any other component, for example. In most cases, you won‘t need to use the glass pane directly, but it is there if you need it.

4.3 Layout Managers

layout manager automatically arranges controls within a window by using some type of algorithm. If you have programmed for other GUI environments, such as Windows, then you are accustomed to laying out your controls by hand. While it is possible to lay out Java controls by hand, too,you generally won‘t want to, for two main reasons. First, it is very tedious to manually layout a large number of components. Second, sometimes the width and height information is not yet available when you need to arrange some control, because the native toolkit components haven‘t been realized.

Each Container object has a layout manager associated with it. A layout manager is an instance of any class that implements the LayoutManager interface. The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The setLayout( ) method has the following general form:

void setLayout(LayoutManager layoutObj)

Here, layoutObj is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass null for layoutObj. If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component. Normally, you will want to use a layout manager.

4.4 Use Jbutton

The JButton class provides the functionality of a push button. JButton allows an icon, a string, or both to be associated with the push button. Three of its constructors are shown here:
JButton(Icon icon)
JButton(String str)
JButton(String str, Icon icon)
Here, str and icon are the string and icon used for the button. When the button is pressed, an ActionEvent is generated. Using the ActionEvent object passed to the actionPerformed( ) method of the registered ActionListener, you can obtain the action command string associated with the button. By default, this is the string displayed inside the button. However, you can set the action command by calling setActionCommand( ) on the button. You can obtain the action command by calling getActionCommand( ) on the event object. It is declared like this:
String getActionCommand( )
The action command identifies the button. Thus, when using two or more buttons within the same application, the action command gives you an easy way to determine which button was pressed.

4.5 Work with JTextField

TextField is the simplest Swing text component. It is also probably its most widely used text component. JTextField allows you to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to Swing text components. JTextField uses the Document interface for its model.

Three of JTextField‘s constructors are shown here:

JTextField(int cols)

JTextField(String str, int cols)

JTextField(String str)

Here, str is the string to be initially presented, and cols is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string. JTextField generates events in response to user interaction. For example, an ActionEvent is fired when the user presses ENTER. ACaretEvent is fired each time the caret (i.e., the cursor) changes position. (CaretEvent is packaged in javax.swing.event.) Other events are also possible. In many cases, your program will not need to handle these events. Instead, you will simply obtain the string currently in the text field when it is needed. To obtain the text currently in the text field, call getText( ).

4.6 Create a Checkbox

The JCheckBox class provides the functionality of a check box. Its immediate superclass is JToggleButton, which provides support for two-state buttons, as just described. JCheckBox defines several constructors. The one used here is JCheckBox(String str) It creates a check box that has the text specified by str as a label. Other constructors let you specify the initial selection state of the button and specify an icon. When the user selects or deselects a check box, an ItemEvent is generated. You can obtain a reference to the JCheckBox that generated the event by calling getItem( ) on the ItemEvent passed to the itemStateChanged( ) method defined by ItemListener. The easiest way to determine the selected state of a check box is to call isSelected( ) on the JCheckBox instance. In addition to supporting the normal check box operation, JCheckBox lets you specify the icons that indicate when a check box is selected, cleared, and rolled-over. We won‘t be using this capability here, but it is available for use in your own programs.

4.7 Work with Jlist

In Swing, the basic list class is called JList. It supports the selection of one or more items from a list. Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed. JList is so widely used in Java that it is highly unlikely that you have not seen one before. JList provides several constructors. The one used here is

JList(Object[ ] items)

This creates a JList that contains the items in the array specified by items. JList is based on two models. The first is ListModel. This interface defines how access to the list data is achieved. The second model is the ListSelectionModel interface, which defines methods that determine what list item or items are selected. Although a JList will work properly by itself, most of the time you will wrap a JList inside a JScrollPane. This way, long lists will automatically be scrollable, which simplifies GUI design. It also makes it easy to change the number of entries in a list without having to change the size of the JList component. AJList generates a ListSelectionEvent when the user makes or changes a selection. This event is also generated when the user deselects an item. It is handled by implementing

ListSelectionListener. This listener specifies only one method, called valueChanged( ), which is shown here:
void valueChanged(ListSelectionEvent le)
Here, le is a reference to the object that generated the event. Although ListSelectionEvent does provide some methods of its own, normally you will interrogate the JList object itself to determine what has occurred. Both ListSelectionEvent and ListSelectionListener are packaged in javax.swing.event.

By default, a JList allows the user to select multiple ranges of items within the list, but you can change this behavior by calling setSelectionMode( ), which is defined by JList. It is shown here:
void setSelectionMode(int mode)
Here, mode specifies the selection mode. It must be one of these values defined by ListSelectionModel:
SINGLE_SELECTION
SINGLE_INTERVAL_SELECTION
MULTIPLE_INTERVAL_SELECTION
The default, multiple-interval selection, lets the user select multiple ranges of items within a list.With single-interval selection, the user can select one range of items.With single selection, the user can select only a single item. Of course, a single item can be selected in the other two modes, too. It‘s just that they also allow a range to be selected.

You can obtain the index of the first item selected, which will also be the index of the only selected item when using single-selection mode, by calling getSelectedIndex( ), shown here: int getSelectedIndex( ) Indexing begins at zero. So, if the first item is selected, this method will return 0. If no item is selected, –1 is returned. Instead of obtaining the index of a selection, you can obtain the value associated with the selection by calling getSelectedValue(
):
Object getSelectedValue( )
It returns a reference to the first selected value. If no value has been selected, it returns null.

4.8 Use anonymous inner classes to handle events

Event handling is one important part of programming in swing. Because JLabel does not take input from the user, it does not generate events, so no event handling was needed. However, the other Swing components do respond to user input and the events generated by those interactions need to be handled. Events can also be generated in ways not directly related to user input. For example, an event is generated when a timer goes off. Whatever the case, event handling is a large part of any Swing-based application.

The event handling mechanism used by Swing is the same as that used by the AWT. This approach is called the delegation event model. In many cases, Swing uses the same events as does the AWT, and these events are packaged in java.awt.event. 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. Event handling program 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.

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 jfrmto FlowLayout. Recall that, by default, the content pane uses BorderLayout as its layout manager. However, for this example, FlowLayout is more convenient. Notice that FlowLayout is assigned using this statement
jfrm.setLayout(new FlowLayout());
As explained, in the past you had to explicitly call getContentPane( ) to set the layout manager for the content pane.

4.9 Create a Swing applet

The second type of program that commonly uses Swing is the applet. Swing-based applets are similar to AWT-based applets, but with an important difference: ASwing applet extends JApplet rather than Applet. JApplet is derived from Applet. Thus, JApplet includes all of the functionality found in Applet and adds support for Swing. JApplet is a top-level Swing container, which means that it is not derived from JComponent. Because JApplet is a top- level container, it includes the various panes described earlier. This means that all components are added to JApplet‘s content pane in the same way that components are added to JFrame‘s content pane.

Swing applets use the same four lifecycle methods as init( ), start( ), stop( ), and destroy( ). Of course, you need override only those methods that are needed by your applet. Painting is accomplished differently in Swing than it is in the AWT, and a Swing applet will not normally override the paint( ) method. All interaction with components in a Swing applet must take place on the event dispatching thread, as described in the previous section. This threading issue applies to all Swing programs.

Here is an example of a Swing applet. It provides the same functionality as the previous application, but does so in applet form.

```
// A simple Swing-based applet
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
This HTML can be used to launch the applet:
<object code="MySwingApplet" width=220 height=90>
</object>
*/
public class MySwingApplet extends JApplet {
JButton jbtnAlpha;
JButton jbtnBeta;
JLabel jlab;
// Initialize the applet.
public void init( ) {
    try {
            SwingUtilities.invokeAndWait(new Runnable ( ) {
                public void run() {
                    makeGUI( ); // initialize the GUI
                }
```

```
          });
      } catch(Exception exc) {
            System.out.println("Can't create because of "+ exc);
 }
          }
// This applet does not need to override start(), stop(),
// or destroy().
// Set up and initialize the GUI.
private void makeGUI( ) {
       // Set the applet to use flow layout.
       setLayout(new FlowLayout());
     // Make two buttons.
     jbtnAlpha = new JButton("Alpha");
     jbtnBeta = new JButton("Beta");
   // Add action listener for Alpha.
    jbtnAlpha.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent le) {
    jlab.setText("Alpha was pressed.");
  }
});
// Add action listener for Beta.
jbtnBeta.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent le) {
jlab.setText("Beta was pressed.");
}
});
// Add the buttons to the content pane.
add(jbtnAlpha);
add(jbtnBeta);
// Create a text-based label.
jlab = new JLabel("Press a button.");
// Add the label to the content pane.
add(jlab);
}
    }
```

There are two important things to notice about this applet. First, MySwingApplet extends JApplet. As explained, all Swing-based applets extend JApplet rather than Applet. Second, the init( ) method initializes the Swing components on the event dispatching thread by setting up a call to makeGUI( ). Notice that this is accomplished through the use of invokeAndWait( ) rather than invokeLater( ).

Applets must use invokeAndWait( ) because the init( ) method must not return until the entire initialization process has been completed. In essence, the start( ) method cannot be called until after initialization, which means that the GUI must be fully constructed. Inside makeGUI( ), the two buttons and label are created, and the action listeners are added to the buttons. Finally, the components are added to the content pane. Although this example is quite simple, this same general approach must be used when building any Swing GUI that will be used by an applet.

Assignment-3

1. Where are the following four methods commonly used?

1) public void add(Component c)
2) public void setSize(int width,int height)
3) public void setLayout(LayoutManager m)
4) public void setVisible(boolean)

A. Graphics class
B. Component class
C. Both A & B
D. None of the above
Answer: Option B

2. Implement the Listener interface and overrides its methods is required to perform in event handling.
A. True
B. False
Answer: Option A

3. Which is the container that doesn't contain title bar and MenuBars but it can have other components like button, textfield etc?
A. Window
B. Frame
C. Panel
D. Container
Answer: Option C

4. Which object can be constructed to show any number of choices in the visible window?
A. Labels
B. Choice
C. List
D. Checkbox
Answer: Option C
5. Object which can store group of other objects is called

A. Collection object
B. Java object
C. Package
D. Wrapper
Answer: Option A

6. JFrame myFrame = new JFrame (); Any command (such as the one listed above) which creates a new object of a specific class (in this case a new JFrame object called myFrame) is generally called a ...

A. Constructor
B. Layout manager
C. Parameter

D. GUI
Answer: Option A

7. What are the names of the list layout managers in Java?

A. Flow Layout Manager
B. Grid Layout Manager
C. Box Layout Manager
D. All of the above
Answer: Option D

8. What is the name of the Swing class that is used for frames?

A. Window
B. Frame
C. JFrame
D. SwingFrame
Answer: Option C

9. What is the name of the Swing class that is used for frames?

A. swing
B. Frame
C. applet
D. SwingFrame
Answer: Option A

10. Which method is used to set the text of a Label object?
A. setText( )
B. setLabel( )
C. setTextLabel( )
D. setLabelText( )
Answer: Option A

11. The layout of a container can be altered by using which of the following methods.
A. setLayout(aLayoutManager)
B. layout(aLayoutManager)
C. addLayout(aLayoutManager)
D. setLayoutManager(aLayoutManager)
Answer: Option A

12. Which of these methods can be used to know which key is pressed?
A. getActionEvent()
B. getActionKey()
C. getModifier()
D. getKey()
Answer: Option C

13. Which of these event is generated when a button is pressed?
A. WindowEvent

B. ActionEvent
C. WindowEvent
D. KeyEvent
Answer: Option B

14. Which of these packages contains all the classes and methods required for event handling in java?
A. Java.applet
B. Java.awt
C. Java.awt.event
D. Java.event
Answer: Option C

15. In Java, what do you call an area on the screen that has nice borders and various buttons along the top border?

A. A window
B. A screen
C. A box
D. A frame
Answer: Option D

16. Suppose you are developing a Java Swing application and want to toggle between various views of the design area. Which of the views given below are present for the users to toggle?

A. Design View
B. Requirements View
C. Source View
D. Management View
Answer: Option B

17. The size of a frame on the screen is measured in:

A. Inches
B. Nits
C. Dots
D. Pixels
Answer: Option D

18. The following specifies the advantages of

It is lightweight.
It supports pluggable look and feel.
It follows MVC (Model View Controller) architecture.

A. Swing
B. AWT
C. Both A & B
D. None of the above

Answer: Option A

19. Which class provides many methods for graphics programming?

A. java.awt
B. java.Graphics
C. java.awt.Graphics
D) None of the above

Answer: Option C

20. The ActionListener interface is used for handling action events,For example,it's used by a

A) JButton
B) JCheckbox
C) JMenuItem
D) All of these

Answer: Option D

21. Which class is used to create a pop-up list of items from which the user may choose?

A. List
B. Choice
C. Labels
D. Checkbox

Answer: Option B

22. Which class is used for this Processing Method processActionEvent( )?

A. Button,List,MenuItem
B. Button,Checkbox,Choice
C. Scrollbar,Component,Button
D. None of the above

Answer: Option A

23. The Swing Component classes that are used in Encapsulates a mutually exclusive set of buttons?

A. AbstractButton
B. ButtonGroup
C. JButton
D. ImageIcon

Answer: Option B

Long Answer Questions

1. Explain the advantages of Swing.
2. Write a short note on containers.
3. Briefly explain components.
4. Mention the purpose of different layout managers available in Swing.
5. Explain the use of JTextField and any methods associate with it.
6. Explain the use of JList and any methods associated with it.
7. Explain the purpose of JButton and explain any methods associated with it.
8. With an example explain how to create a swing applet.
9. Explain the use of JCheckBox and any methods associated with it.
10. Give one example for swing program