Chapter 1

1] What are the two types of applets? Explain.

It is important to state at the outset that there are two varieties of applets. The first are those based directly on the **Applet** class described in this chapter. These applets use the Abstract Window Toolkit (AWT) to provide the graphic user interface (or use no GUI at all). This style of applet has been available since Java was first created.
The second type of applets are those based on the Swing class **JApplet**. Swing applets use the Swing classes to provide the GUI. Swing offers a richer and often easier-to-use user interface than does the AWT. Thus, Swing-based applets are now the most popular. However, traditional AWT-based applets are still used, especially when only a very simple user interface is required. Thus, both AWT- and Swing-based applets are valid.
Because **JApplet** inherits **Applet**, all the features of **Applet** are also available in **JApplet**, and most of the information in this chapter applies to both types of applets. Therefore, even if you are interested in only Swing applets, the information in this chapter is still relevant and necessary. Understand, however, that when creating Swing-based applets, some additional constraints apply and these are described later in this topic, when Swing is covered.

2] What is Applet? Describe Applet Life Cycle.

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.
Life Cycle
Four methods in the Applet class give you the framework on which you build any serious applet:
init: This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
start: This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
stop: This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
paint: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

3] What is Applet. Explain with example.

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

import java.applet.*;

```
import java.awt.*;

public class HelloWorldApplet extends Applet
{
  public void paint (Graphics g)
  {
    g.drawString ("Hello World", 25, 50);
  }
}
```

4] What do you mean by Applet Skeleton? Elaborate it.
Most of the applets override a set of methods that controls its execution.

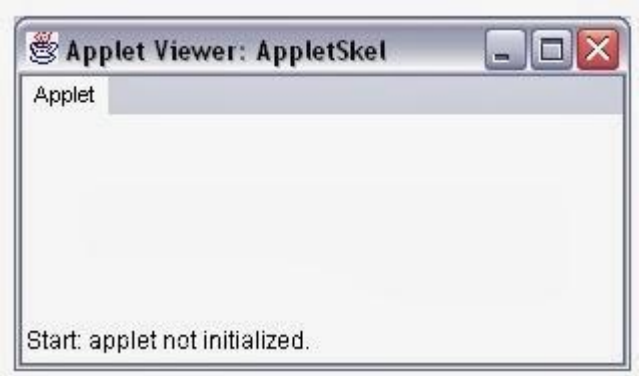
Four of these methods init( ), start( ), stop( ), and destroy( ) are defined by Applet.
paint( ), is defined by the AWT Component class.

These five methods can be assembled into the skeleton shown here:

```
// An Applet skeleton.

import java.awt.*;
import javax.swing.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/

public class AppletSkel extends JApplet {
 // Called first.
 public void init() {
   // initialization
 }

 /* Called second, after init().  Also called whenever
    the applet is restarted. */
 public void start() {
   // start or resume execution
 }

 // Called when the applet is stopped.
 public void stop() {
   // suspends execution
```

```
  }

  /* Called when applet is terminated.  This is the last
     method executed. */
  public void destroy() {
    // perform shutdown activities
  }

  // Called when an applet's window must be restored.
  public void paint(Graphics g) {
    // redisplay contents of window
  }
}
```

Although this skeleton does not do anything, it can be compiled and run. When run, it generates the following window when viewed with an applet viewer:



5] What do you mean by HTML Applet tag? Explain it with example.

The HTML <applet> tag specifies an applet. It is used for embedding a Java applet within an HTML document. It is not supported in HTML5.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML applet Tag</title>
</head>
<body>
<applet code="newClass.class" width="300" height="200">
</applet>
</body>
```

```
</html>
```

6] How can you pass parameter to an applet? Explain with suitable example.

```
import java.awt.*;
import java.applet.*;

publicclass passingPara extends Applet
{
      String str;
      int a,b,result;

     publicvoid init(){
           str=getParameter("a");
          a=Integer.parseInt(str);
          str=getParameter("b");
          b=Integer.parseInt(str);
          result=a+b;
          str=String.valueOf(result);
       }

   publicvoid paint(Graphics g){
        g.drawString(" Result of Addition is : "+str,0,15);
   }
}
```

Chapter 2

1] What are Two event handling mechanisms?

Before beginning our discussion of event handling, an important point must be made: The way in which events are handled changed significantly between the original version of Java (1.0) and modern versions of Java, beginning with version 1.1. The 1.0 method of event handling is still supported, but it is not recommended for new programs. Also, many of the methods that support the old 1.0 event model have been deprecated. The modern approach is the way that events should be handled by all new programs and thus is the method employed by programs in this topic.

2] What is Delegation event Model? Explain it.

The modern approach to handling events is based on the delegation event model, which defines standard and constituent mechanisms to generate and process events. Its concept is quite simple, A source generates an event and sends it to one or more listeners.

In delegation event model, listener must register with a source in order to receive the event notification.

Events

In delegation event model, the event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a GUI.

Event Sources

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Public void addTypeListener(TypeListener el)

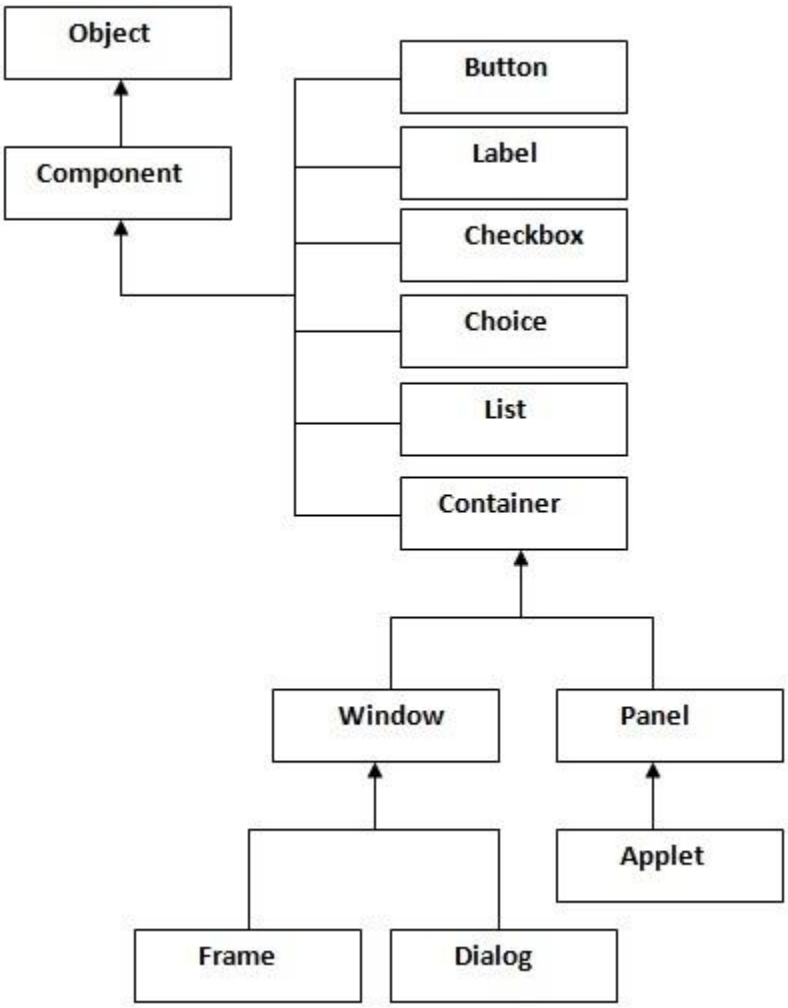
Event Listeners

A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notification. The methods that receive and process events are defined in a set of interfaces found in java.awt.event. For example, the MOouseMotionListener interface defines two methods to receive notifications when the mouse is dragges or moved.

Chapter 3

1] Describe AWT Hierarchy Model.

The hierarchy of Java AWT classes are given below.



2] What do you mean by AWT classes? Explain it.

The AWT classes are contained in the '**java.awt**' package. Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use.

Following are the list of some AWR classes.

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a checkbox control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.

Container	A subclass of **Component** that can hold other components.
Cursor	Encapsulates events.
Dialog	Creates a dialog window.
Dimension	Specifies the dimentions of an object. The width is stored in**width**, and the height is stored in **height**.
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This information helps you display text in a window.
Frame	Creates a standard window that has t title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GraphicsDevice	Describes a graphics device such as a screen printer.
GraphicsEnvironment	Describes the collection of a available **Font** and**GraphicsDevice** objects.
GridBagConstraints	Defines various constraints relating to the **GridBagLayout** class.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
GridBagLayout	The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by**GridBagConstraints**.
Image	Encapsulates graphical images.

Insets	Encapsulates th borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of **Container**.
Point	Encapsulates a Cartesian co-ordinate pair, stored in x and y.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT-based applications.(Added by Java 2, v1.3)
Scrollbar	Creates a scroll bar control.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.

TextArea	Creates a multiline edit control.
TextComponent	A superclass for **TextArea** and **TextField**.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

3**] What are window fundamental in AWT? Elaborate it.**

Window
The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel
The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame
The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Container
The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extend Container class are known as container such as Frame, Dialog and Panel.

Container
Container object is a component that can contain other components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list.

4] what are Advantages of GUI over CUI ?

- GUI provides graphical icons to interact while the CUI (Character User Interface) offers the simple text-based interfaces.
- GUI makes the application more entertaining and interesting on the other hand CUI does not.
- GUI offers click and execute environment while in CUI every time we have to enter the command for a task.

- New user can easily interact with graphical user interface by the visual indicators but it is difficult in Character user interface.
- GUI offers a lot of controls of file system and the operating system while in CUI you have to use commands which is difficult to remember.
- Windows concept in GUI allow the user to view, manipulate and control the multiple applications at once while in CUI user can control one task at a time.
- GUI provides multitasking environment so as the CUI also does but CUI does not provide same ease as the GUI do.
- Using GUI it is easier to control and navigate the operating system which becomes very slow in command user interface. GUI can be easily customized.

5] what are example of GUI based Application? Explain AWT example.

Following are some of the examples for GUI based applications.

- Automated Teller Machine (ATM)
- Airline Ticketing System
- Information Kiosks at railway stations
- Mobile Applications
- Navigation Systems

```
import java.awt.*;

class First extends Frame{

First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position

add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout now bydefault BorderLayout
setVisible(true);//now frame willbe visible, bydefault not visible

}
public static void main(String args[]){

First f=new First();

}
}
```

6] What are AWT control? Explain it.
Every user interface considers the following three main aspects:

- **UI elements** : Thes are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

7] How to create a frame window from within an applet?

Creating a new frame window from within an applet is actually quite easy.
The following steps may be used to do it,

- Create a subclass of Frame
- Override any of the standard window methods,such as init(),start(),stop(),and paint().
- Implement the windowClosing() method of the windowlistener interface,calling setVisible(false)when the window is closed
- Once you have defined a Frame subclass,you can create an object of that class.But it will note be initially visible
- When created,the window is given a default height and width
- You can set the size of the window explicitly by calling the setSize() method

The example program is shown below:

```
// Create a child frame window from within an applet.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="AppletFrame" width=400 height=60>
</applet>
*/
// Create a subclass of Frame.
class SampleFrame extends Frame {
SampleFrame(String title) {
super(title);
// create an object to handle window events
MyWindowAdapter adapter = new MyWindowAdapter(this);
// register it to receive those events
```

```
addWindowListener(adapter);
}
public void paint(Graphics g) {
g.drawString("This is in frame window", 10, 40);
}
}
class MyWindowAdapter extends WindowAdapter {
SampleFrame sampleFrame;
public MyWindowAdapter(SampleFrame sampleFrame) {
this.sampleFrame = sampleFrame;
}
public void windowClosing(WindowEvent we) {
sampleFrame.setVisible(false);
}
}
// Create frame window.
public class AppletFrame extends Applet {
Frame f;
public void init() {
f = new SampleFrame("A Frame Window");
f.setSize(150, 150);
f.setVisible(true);
}
public void start() {
f.setVisible(true);
}
public void stop() {
f.setVisible(false);
}
public void paint(Graphics g) {
g.drawString("This is in applet window", 15, 30);
}

}
```

8] How to set the dimension of windows in AWT?

There are several key methods you will use when working with **Frame** windows. They are examined here.

Setting the Window's Dimensions

The **setSize( )** method is used to set the dimensions of the window. Its signature is shown here:

```
void setSize(int newWidth, int newHeight)
void setSize(Dimension newSize)
```

The new size of the window is specified by *newWidth* and *newHeight*, or by the **width** and **height** fields of the **Dimension** object passed in *newSize*. The dimensions are specified in terms of pixels.

The **getSize( )** method is used to obtain the current size of a window. Its signature is shown here:

```
Dimension getSize( )
```

This method returns the current size of the window contained within the **width** and **height** fields of a **Dimension** object.

9] How to hide and close the window in AWT?

Hiding and Showing a Window

After a frame window has been created, it will not be visible until you call **setVisible( )**. Its signature is shown here:

```
void setVisible(boolean visibleFlag)
```

The component is visible if the argument to this method is **true**. Otherwise, it is hidden.

Setting a Window's Title

You can change the title in a frame window using **setTitle( )**, which has this general form:

```
void setTitle(String newTitle)
```

Here, *newTitle* is the new title for the window.

Closing a Frame Window

When using a frame window, your program must remove that window from the screen when it is closed, by calling **setVisible(false)**. To intercept a window-close event, you must implement the **windowClosing( )** method of the **WindowListener** interface. Inside **windowClosing( )**, you must remove the window from the screen. The example in the next section illustrates this technique.

Chapter-4

1] What are the different types of AWT support control fundamentals in java?

The AWT supports the following types of controls:-

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

2] How do you add and remove controls in AWT?

To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling add( ), which is defined by Container.

The add( ) method has several forms.

Component add(Component compObj)

Here, compObj is an instance of the control that you want to add. A reference to compObj is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call remove( ). This method is also defined by Container.

It has this general form:

void remove(Component obj)

Here, obj is a reference to the control you want to remove. You can remove all controls by calling removeAll( ).

3] How do you respond to control in AWT?

Except for labels, which are passive controls, all controls generate events when they are accessed by the user. For example, when the user clicks on a push button, an event is sent that identifies the push button. In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor. Once a listener has been installed, events are automatically sent to it.

4] Which Layout Managers Classes used controls while designed GUI using AWT? Explain in Brief.

Following is the list of commonly used controls while designed GUI using AWT.

Sr. No.	Layout Manager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
2	**CardLayout** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout. It layouts the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in form of a rectangular grid.
5	**GridBagLayout** This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

5] What is Label? How text can be aligned within the label? Explain with suitable example.

This class is a component which displays a single line of text. Labels are read-only. User cannot click on a label to edit the text it displays. Text can be aligned within the label

```
Label aLabel = new Label("Enter Your Name");
aLabel.setAlignmnet(Label.RIGHT);
aPanel.add(aLabel);
```

6]What is Button? Explain with suitable example.

This class represents a push-button which displays some specified text. When a button is pressed, it notifies its Listeners. To be a listener for a button, an object must implement the ActionListener Interface.

```
Panel aPanel = new Panel();
Button okButton = new Button("OK");
Button cancelButton = new Button("Cancel");
aPanel.add(okButton));
aPanel.add(cancelButton));
okButton.addActionListener(controller2);
cancelButton.addActionListener(controller1);
```

7] What is Checkboxes? Explain in brief.

This class represents a GUI checkbox with a textual label. The Checkbox maintains a Boolean state indicating whether it is checked or not. If a checkbox is added to a CheckBoxGroup, it will behave like a radio button.

```
Checkbox creamCheckbox =new CheckBox("Cream");
Checkbox sugarCheckbox =new CheckBox("Sugar");
If ( creamCheckbox.getState()){
Coffee.addCream();
}
```

8] What is Choice? Explain in brief.

This class represents a dropdown list of Strings. Similar to a list in terms of functionality, but displayed differently. Only one item from the list can be selected at one time and the currently selected element is displayed.

```
Choice aChoice = new Choice();
aChoice.add("Calgary");
aChoice.add("Edmonton");
aChoice.add("Alert Bay");
String selectedDestination= aChoice.getSelectedItem();
```

9] What is List? Explain in Brief.

This class is a Component which displays a list of Strings. The list is scrollable, if necessary. Sometimes called Listbox in other languages. Lists can be set up to allow single or multiple selections. The list will return an array indicating which Strings are selected

```
List aList = new List();
aList.add("Calgary");
aList.add("Edmonton");
aList.add("Regina");
aList.add("Vancouver");
aList.setMultipleMode(true);
```

10] What is Textarea? Explain in Brief.

This class displays multiple lines of optionally editable text.This class inherits several methods from TextComponent.TextArea also provides the methods: appendText(), insertText() and replaceText()

```
// 5 rows, 80 columns
TextArea fullAddressTextArea = new TextArea(5, 80);
String userFullAddress= fullAddressTextArea.getText();
```

11] What is Textfield? Explain in Brief.

This class displays a single line of optionally editable text.This class inherits several methods from TextComponent.This is one of the most commonly used Components in the AWT

```
TextField emailTextField = new TextField();
TextField passwordTextField = new TextField();
passwordTextField.setEchoChar("*");
String userEmail = emailTextField.getText();
String userpassword = passwordTextField.getText();
```

12] Differentiate between Swing and AWT.

- Swing is bigger, slower, and more complicated
  - But much faster than it used to be
- Swing is more flexible and better looking
- Swing and AWT are *incompatible*--you can use either, but you can't mix them
  - Actually, you can, but it's tricky and not worth doing
- Learning the AWT is a good start on learning Swing
- Many of the most common controls are just renamed

  AWT:Buttonb=newButton("ok");

  Swing: JButton b = new JButton("OK");

14] What is Dialog? Explain it.

A subclases of Window that can have a border and be modal(i.e prevent the user from doing anything else until he responds). It provides a 'dialog' between the user and your application. Thus its functionality is used for 'About dialog', messages, displaying a list of options, querying for input, etc.

There is still a bug in the JDK/AWT that prevents dialogs from being modal

```
// Note this class is created hidden
// You'll need to call show() to make it visible
class AboutDialog extends Dialog
{
        static int H_SIZE = 200;
        static int V_SIZE = 200;

        public AboutDialog(Frame parent)
        {
                // Calls the parent telling it this
                // dialog is modal(i.e true)
          super(parent, true);
                setBackground(Color.gray);
                setLayout(new BorderLayout());

                // Two buttons "Close" and "Help"
                Panel p = new Panel();
                p.add(new Button("Close"));
                p.add(new Button("Help"));
          add("South", p);
                resize(H_SIZE, V_SIZE);
        }

    public boolean action(Event evt, Object arg)
    {
                // If action label(i.e arg) equals
                // "Close" then dispose this dialog
          if(arg.equals("Close"))
          {
               dispose();
               return true;
          }
          return false;
    }

        public void paint(Graphics g)
        {
                g.setColor(Color.white);
          g.drawString("My Application", H_SIZE/4, V_SIZE/3);
          g.drawString("Version 1.0", H_SIZE/3, V_SIZE/3+20);
        }
}
```

Chapter 5

1] What is Swing? Describe with suitable example?

The original Java GUI subsystem was the Abstract Window Toolkit (AWT).
AWT translates it visual components into platform-specific equivalents (peers).
Under AWT, the look and feel of a component was defined by the platform.
AWT components are referred to as **heavyweight**.
Swing was introduced in 1997 to fix the problems with AWT.
Swing offers two key features:
Swing components are **lightweight** and don't rely on peers.
Swing supports a pluggable look and feel. The three PLAFs available to all users are Metal (default), Windows, and Motif.
Swing is built on AWT.

```
import javax.swing.*;
class SwingDemo {
SwingDemo() {
  // Create a new JFrame container.
    JFrame jfrm = new JFrame("A Simple Swing Program");

    // 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 powers the modern Java GUI.");

    // Add the label to the content pane.
    jfrm.getContentPane().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();
    }
   });

 }
}
```

2] What are two key swing features in java? Explain in brief.

As just explained, 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. Each is examined here.

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.

3] What is MVC?

If you've programmed with graphical user interface (GUI) libraries in the past 10 years or so, you have likely come across the model-view-controller (MVC) design. MVC was first introduced by Trygve Reenskaug, a Smalltalk developer at the Xerox Palo Alto Research Center in 1979, and helps to decouple

data access and business logic from the manner in which it is displayed to the user. More precisely, MVC can be broken down into three elements:

Model - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.

View - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a *push model*, in which the view registers itself with the model for change notifications, or a *pull model*, in which the view is responsible for calling the model when it needs to retrieve the most current data.

Controller - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POSTHTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user.

4] Describe Component and Container in swing.

In Java, a component is the basic user interface object and is found in all Java applications. Components include lists, buttons, panels, and windows.

To use components, you need to place them in a container.

A container is a component that holds and manages other components. Containers display components using a layout manager.

Swing components inherit from the javax.Swing.JComponent class, which is the root of the Swing component hierarchy.JComponent, in turn, inherits from the Container class in the Abstract Windowing Toolkit (AWT). So Swing is based on classes inherited from AWT.

Swing provides the following useful top-level containers, all of which inherit from JComponent:

JWindow

JWindow is a top-level window that doesn't have any trimmings and can be displayed anywhere on a desktop.JWindow is a heavyweight component. You usually use JWindow to create pop-up windows and "splash" screens.JWindow extends AWT's Window class.

JFrame

JFrame is a top-level window that can contain borders and menu bars. JFrame is a subclass of JWindow and is thus a heavyweight component. You place a JFrame on a JWindow. JFrame extends AWT's Frame class.

JDialog

JDialog is a lightweight component that you use to create dialog windows. You can place dialog windows on aJFrame or JApplet. JDialog extends AWT's Dialog class.

JApplet

JApplet is a container that provides the basis for applets that run within web browsers. JApplet is a lightweight component that can contain other graphical user interface (GUI) components. JApplet extends AWT's Applet class.