UNIT 5

GUI PROGRAMMING WITH SWING, EVENT HANDLING AND APPLETS

PART-A
SHORT QUESTIONS WITH SOLUTIONS

1. What is Swing in Java? How it differs from Applet.

Answer :

(Model Paper-I, Q1(i) | Nov./Dec.-18(R16), Q1(i))

Swing

Swing is a set of classes that provides more powerful and flexible functionalities when compared to AWT components. Besides components such as buttons, check boxes and labels. Swing adds other components such as tabedpanes, scrollpanes, trees and tables. And each component in Swing have more capabilities. For example, a button can have an image and a text string with it.

Differences Between Swing and Applet

	Swing		Applet
1.	It is considered as light weight component.	1.	It is considered as heavy weight component.
2.	It consists of a few thread rules.	2.	It does not consists any type of rules.
3.	It posses look and feel feature. This can be changed by UIManager.	3.	It does not posses such kind of feature or characteristic.
4.	It makes use of model view controller (MVC).	4.	It does not use model view control (MVC).
5.	It requires main method to execute the program.	5.	It requires HTML code to run the applet.
6.	It uses its own layout i.e., Box Layout.	6.	It uses AWT Layouts such as flow layout, Border layout, etc.

2. What are the limitations of AWT?

Answer :

(April/May-18(R16), Q1(i) | Nov./Dec.-16(R13), Q1(i))

AWT stands for Abstract Window Toolkit. It is a toolkit to develop GUI. There is a basic set of controls, dialog boxes and windows defined in AWT that support limited graphical interface which is usable. AWT has the limited nature because different visual components that are present in it are translated into the equivalents that are platform-specific such as 'peers'. Thus, the component's look and feel is defined by the platform. The AWT components are also called as heavy weight components as they make use of native code resources.

Problems faced because of the use of native peers are as follows,

Java's philosophy to 'Write once and Run anywhere' was threatened, because different operating systems varied from each other, the component's look and actions also varied on different platforms.

The component's look and feel was defined by the platform which cannot be changed easily.

There were disappointing restrictions when heavy weight components were used.

Q3. Write the differences between swing and AWT. Model Paper-II, Q1(j)

Answer :

	Swing		AWT
1.	Swing components are lightweight components, i.e., they are platform-independent.	1.	AWT components are heavy-weight components. This means that AWT components are platform-specific.
2.	The look and feel of each component is defined by swing, not by the platform. Hence, the working of swing components is consistent across any operating system or platform.	2.	The look and feel of each component is determined by the platform.
3.	Swing components are transparent.	3.	AWT components are always rectangular and opaque.

Q4. Discuss about the MVC connection. Nov./Dec.-17(R13), Q1(j)

Answer :

A visual component has the following characteristics,

1. The LOOK of the component when it is presented on the screen.
2. The RESPONSE from the component to the user.
3. The STATE INFORMATION related to the component.

Model View Controller or MVC architecture implicitly contains all the above characteristics. The design of MVC is based on the characteristics of a component.

- 'Model' term in the name of this architecture corresponds to the state information related to the component.

Example

When a checkbox is used, model holds a field that denotes whether or not the box is checked or unchecked.

- 'View' term in the name of this architecture corresponds to how the component will look when it is presented on the screen. It includes determination of even those characteristics that are affected by model's current state.
- 'Controller' term in the name of this architecture corresponds to how the component responds to the user.

Q5. Discuss about FlowLayout.

Answer : Nov./Dec.-17(R13), Q1(h)

FlowLayout Manager

FlowLayout Manager is the default layout manger. It is considered to be simplest of all layout managers. It lays out the components from upper-left corner from left to right and from top to bottom, by separating the components with a small space. When a line is filled it moves to the next row. The method it follows is similar to the way the words are wrapped in a page i.e., from left to right.

Constructors

It has three types of constructors,

1. FlowLayout( )
2. FlowLayout(int how)
3. FlowLayout(int how, int hor, int ver).

Q6. Define event.

Answer :

An event is an action performed by the user. In the delegation event model, an event is an object that shows a status i.e., state changes in a source. Events are generated when the user interacts with the elements in a graphical user interface. Some of the actions that generate events are pressing a button, entering the data via keyboard, selecting an item in a list, clicking the mouse etc. These are all the events that are generated by a user by interacting with a user interface.

Q7. What are sources of events?

Answer :

Nov./Dec.-17(R13), Q1(g)

The following are the sources of events,

(i) Button
(ii) Checkbox
(iii) Choice
(iv) List
(v) MenuItem
(vi) Scrollbar
(vii) Text components
(viii) Window.

Q8. What is event listener? Explain.

Answer :

Nov./Dec.-16(R13), Q1(g)

A listener is an object which is notified when an event occurs. It must not only register with one or more sources for receiving notifications but also must implement methods for receiving and processing those notifications. The methods for receiving and processing events are defined in a set of interfaces found in java.awt.event.

Q9. What is an adapter class? Explain with an example.

Answer :

(Model Paper-I, Q1(j) | Nov./Dec.-17(R16), Q1(j))

Adapter Class

Adapter class is time consuming to override all the methods of an interface to handle particular event. For example, to close a window the user needs to override all the abstract methods of the Window Listener interface. Eventhough we need to override only one method namely windowClosing( ) we are forced to override all the remaining methods of this interface.

Java provides a solution to this problem by providing adapter classes. By this adapter class we need not override all the methods of WindowListener interface, we can just override our interested method windowClosing( ) only by using an adapter class provided for WindowListener interface.

There is an adapter class for each interface having more than one abstract methods.

Example

WindowListener has adapter class called WindowAdapter.

Q10. Discuss in brief about applets.

Answer :

Model Paper-II, Q1(j)

An applet is a Java program that is embedded in HTML document and run with the help of Java enabled browsers such as, Internet Explorer. In other words, an applet is a Java program that runs in a browser. Unlike Java applications, applets don't have a main( ) method. All applets inherit the superclass 'Applet'. An Applet class contains several methods that helps to control the execution of an applet. All applets must import java. applet and java.awt packages.

Q11. Describe applet architecture.

Answer :

Nov./Dec.-17(R13), Q1(l)

Since, an applet is a window-based program, its architecture will be different from console-based programs architecture. Applets are event driven. An applet resembles a set of interrupt service routines. An applet waits for an event to occur. The AWT notifies the applet regarding an event by calling an event handler which is provided by the applet. After this, the applet should take a suitable action and then quickly return control to the AWT. In situations where, an applet requires to perform a repetitive task on its own, an additional thread of execution must be started.

The user initiates interaction with an applet not the other way around. In a windowed program, when the program needs input, it will prompt the user and then calls some input methods like readLine( ). But this is not the case in applets, instead the user can interact with the applet whenever he want and however he want. All these interactions will be forwarded to the applet as events to which the applet should respond. Applets can also contain different controls like push buttons, check boxes etc., an event is generated whenever user interacts with these controls.

Java programs can be divided into two categories. They are applications and applets. Applications contain main( ) method and runs with Java interpreter, whereas applets does not contain main( ) method and runs in a browser.

Q12. State the differences between applets and applications. Nov./Dec.-16(R13), Q1(j)

OR

How do applets differ from application program?

Answer : Nov./Dec.-18(R16), Q1(j)

	Application		Applet
1.	Applications are stand-alone programs.	1.	Applets are not stand-alone programs.
2.	The main( ) method exists.	2.	There is no main( ) method.
3.	They need a Java interpreter for execution.	3.	They need a browser like netscape for execution.
4.	They have no hard disk accessing restrictions.	4.	They can't access hard disk.
5.	They doesn't need any security.	5.	They need topmost security for hard disk files.

Q13. Why do applet classes need to be declared as public?

Answer : April/May-18(R16), Q1(j)

The applet will be run only on web browser. It is an fully functional application since it has the complete Java API at its convenience. The applet attributes will be sometimes accessed by the server when required. This is possible only if the applet is declared as public. If it is not declared as public no vendor can be able to access it.

Q14. What are the differences between JToggleButton and RadioButton?

Answer : Nov./Dec.-17(R16), Q1(j)

JToggleButton

JToggleButton looks similar to that of JButton but the functionality of it differs. It has two states namely push and release. When this button is pressed it remains as pressed rather than popping back. In order to release it in the user should press it again. It gets released on the second push. That is it toggles between states for each button press.

Toggle buttons are the objects of class JToggleButton. The classes such as JCheckBox and JRadioButton are derived from it. JToggleButton generates an action event when it is pressed. But it generates an item event when the concept of selection is used. When the toggle button is pressed selection is performed and when it is pressed again deselection is performed. In order to handle the item events, the interface called ItemListener must be implemented.

JRadioButton

Radio buttons are created using JRadioButton class, which are similar to check boxes that is they also have two states - selected or deselected. Radio buttons are group of buttons from which only one radio button can be selected at any time. Selecting other radio button makes the previous selected radio button to be deselected automatically. JRadioButton class is a subclass of JToggleButton class. The grouping of the radio button is made by adding the JRadioButton object to a ButtonGroup object that takes care of the state of the buttons in the group.

Q15. Write about checkbox.

Answer :

A JCheckBox is a swing GUI component that provides the functionality of a checkbox. Checkboxes have two states i.e., selected or not (true or false).

Checkboxes are created using JCheckBox class which is subclass of JToggleButton. Hierarchy of JCheckBox and JRadioButton is shown in the figure below. Here is an example of creating checkboxes and formatting the text as per the checkbox selected.

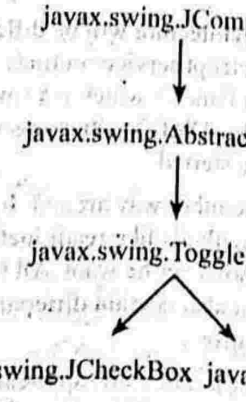


Figure: Hierarchy of JCheckBox and JRadioButton

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

5.1 GUI PROGRAMMING WITH SWING

5.1.1 Introduction, Limitations of AWT

6. What is swing? Discuss its features. List out the limitations of AWT.

Answer :

Swing

Swing is a set of classes that provides more powerful and flexible functionalities when compared to AWT components. Swing provides components such as buttons, check boxes and labels. Swing adds other components such as tabbedpanes, scrollpanes, trees and tables. And each component in Swing have more capabilities. For example, a button can have an image and a text string with it.

Features of Swing

The two basic features of swing are,

1. Light-weight components
2. Pluggable look and feel.

Swing's Light-weight Components

The components of swing are written in Java and are termed as light-weight components. These components cannot directly map to those peers that are platform-specific. The light-weight components are more flexible and efficient to use, as they are rendered using the graphics primitives, which will allow them to be transparent. Eventually, shapes other than rectangles can also be drawn using the light-weight components.

Swing finds out the look and feel of all the components as they cannot be translated into native peers. Thus, all the components will be become capable of working at all platforms and in a consistent manner.

Swing's Pluggable Look and Feel (PLAF)

Swing controls the look and feel of all its components, as Java code renders the swing component, but not the native peers. Thus, it is possible to keep the look and feel of a component separated from its logic. The separated look-and-feel of a component will provide the following advantages,

- A new look-and-feel can be plugged into a component without changing the code of that component.
- In addition, several looks-and-feels can be grouped together to correspond to different GUI styles. Any of these styles can be used by simply plugging into that style. When a specific style is plugged-in, that style will be used to render each of the components.
- A specific look-and-feel can also be defined that can be used in all types of platforms without any change.
- Alternatively, a look-and-feel can also be designed that corresponds to a specific platform.

Limitations of AWT

AWT stands for Abstract Window Toolkit. It is a toolkit to develop GUI. There is a basic set of controls, dialog boxes and windows defined in AWT that support limited graphical interface which is usable. AWT has the limited nature because different visual components that are present in it are translated into the equivalents that are platform-specific such as 'peers'. Thus, the component's look and feel is defined by the platform. The AWT components are also called as heavy weight components as they make use of native code resources.

Problems faced because of the use of native peers are as follows,

Java's philosophy to 'Write once and Run anywhere' was threatened, because different operating systems varied from each other, the component's look and actions also varied on different platforms.

The component's look and feel was defined by the platform which cannot be changed easily.

There were disappointing restrictions when heavy weight components were used.

Example

The shape and colour of a heavy weight component is always rectangular and opaque respectively.

Q17. With a neat sketch, explain the Swing Architecture.

Nov./Dec.-16(R13), Q11(a)

Answer :

Swing Architecture

Swing components are enhanced components compared to that of AWT. They are event driven and provide a good programming approach through OOP concepts. They are light weight components which makes use of model-view controller architecture. All the components of swing are contained in javax.swing package.

The hierarchy of swing components is shown in figure below,

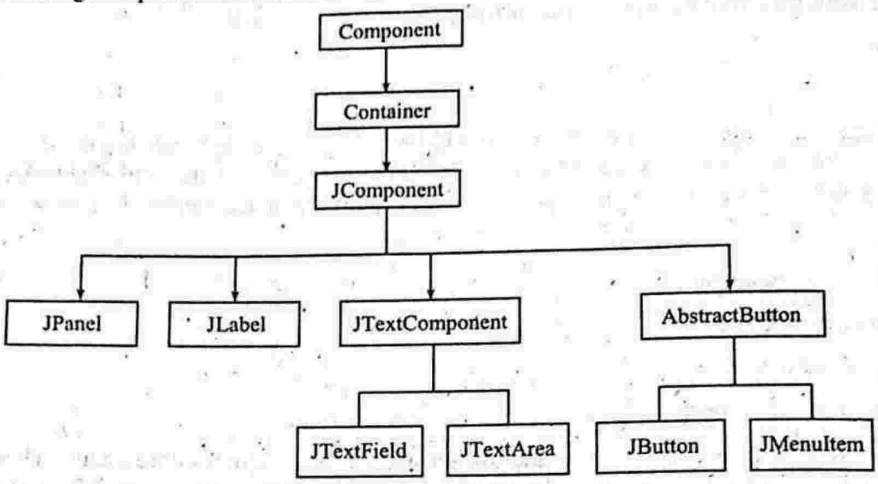


Figure: Swing Components Hierarchy

The hierarchy of swing components consists of following elements,

(i) **Component:** It is a visual control which is independent.

(ii) **Container:** It is a type of component that contains other components.

(iii) **JComponent:** It is a class that holds all the other components. It inherits the AWT classes container and components and provides pluggable look and feel.

(iv) **JPanel:** It is a light weight swing container. It is a component that can be added to contentPane of JFrame.

(v) **JLabel:** It is an easiest-to-use component that is used to create a component.

(vi) **JTextComponent:** It is the base class for JTextField and JTextArea.

(vii) **JTextField:** It allows the user to enter or edit single line of text.

(viii) **JTextArea:** It allows the user to enter or edit multiple line of text.

(ix) **AbstractButton:** It is the base class of JButton and JMenuItem.

(x) **JButton:** It is a push button associated with icon, a string or both.

(xi) **JMenuItem:** It represents an list of item in the form of menu.

5.1.2 MVC Architecture, Components, Containers

Q18. Discuss Model View Controller (MVC) architecture.

Answer :

Model Paper-I, Q10(a)

A visual component has the following characteristics,

1. The LOOK of the component when it is presented on the screen.
2. The RESPONSE from the component to the user.
3. The STATE INFORMATION related to the component.

Model View Controller or MVC architecture implicitly contains all the above characteristics. The design of MVC is based on the characteristics of a component.

- 'Model' term in the name of this architecture corresponds to the state information related to the component.

Example

When a checkbox is used, model holds a field that denotes whether or not the box is checked or unchecked.

'View' term in the name of this architecture corresponds to how the component will look when it is presented on the screen. It includes determination of even those characteristics that are affected by model's current state.

'Controller' term in the name of this architecture corresponds to how the component responds to the user.

When a component is distinguished into a model, view and a controller, a change in implementation of each will not affect the other two.

Swing makes use of a modified version of MVC. As its components are not benefited when the view and the controller have a high level of separation between them. The modified version of MVC has a logical entity called 'UI delegate' which is a combination of the view and controller. Thus the architecture used by the swing is called a 'Separable Model Architecture' or a 'Model-Delegate Architecture'.

As the model-delegate architecture is used in Swing, it separates the model from the look and feel of the component. Thus, change in the look and feel of the component will not affect the use of the component within a program and vice versa.

Swing components make use of two objects in order to support the model-delegate architecture. One to represent a model and the other to represent a UI delegate. Interfaces define the models. For instance, a button's model is defined using 'ButtonModel' interface. UI delegates are classes and they inherit 'Component UI'. For instance, the button's UI delegate is ButtonUI.

Q19. Describe about various components and JFC containers in swing.

OR

Discuss about components and containers. Nov./Dec.-17(R13), Q10(a)

OR

What are the various components of Swing? Explain.

(Refer Only Topic: Components of Swing)

Nov./Dec.-17(R16), Q11(b)

Answer :

Components of Swing

Swing is a set of classes that provides more powerful and flexible functionalities when compared to AWT components. Besides components such as buttons, check boxes and labels Swing adds other components such as tabedpanes, scrollpanes, trees and tables. And each component in Swing have more capabilities. For example, a button can have an image and a text string with it.

Unlike AWT components that are heavy weight, Swing components are light weight because they are written entirely in Java and therefore are platform independent and hence they have the same look and feel on different operating systems or for each platform.

The classes that are used to create the GUI components are from javax.swing package are as follows,

Class	Description
JApplet	This class is the Swing version of Applet.
JButton	This is the Swing version of Button class that generates an event when button is pressed.
JCheckBox	This class is the Swing Checkbox class.
JComboBox	It is the combination of a drop-down list and text field from which user can select an item by clicking in the text or by typing into the box.
JLabel	This is the Swing's Label class that provides an area for uneditable text.
JRadioButton	This is Swing's RadioButton class.
JScrollPane	It provides a scrollable window.
JTabbedPane	It provides a tabbed window.
JTextField	This is Swing's TextField class for user to input data from keyboard.
JTree	It provides a tree-based control.

JComponent class is the superclass to most of GUI components of Swing. It has the method that can be applied to any subclass of JComponent. Swing components that subclass JComponent have the following features,

1. They provide a pluggable look and feel when the program executes on different platforms.
2. Shortcut keys are provided to access GUI components directly through the use of keyboard.
3. If several GUI components initiates the same action then common event handling capabilities are provided.
4. Tool tips are provided for GUI components when the mouse cursor is positioned over components for a short period of time.
5. It provides support for assistive (related) technologies such as Braille screen for blind readers.
6. It also provides the support for user interface localization i.e., customizing the user interface for display in different languages and cultural conventions.

JFC Containers

For answer refer Unit-V, Q20.

Q20. What are various JFC containers? List them according to their functionality. Explain each of them with examples.

Answer :

There are two different types of containers that are defined in swing. They are,

1. Top-level or heavy-weight containers.
2. Light-weight containers.

1. Top-level or Heavy-weight Containers

The top level containers defined in swing are JApplet, JFrame, JDialog and JWindow.

These containers inherit 'Component' and 'Container' classes of AWT but they do not inherit 'Jcomponent' class.

The top level container is always at the top of a containment hierarchy. Thus any container cannot have a top-level container within it. A containment hierarchy always begins with a top-level container. The top-level container, JFrame is mostly used for an application and JApplet is used for applets.

(a) JFrame

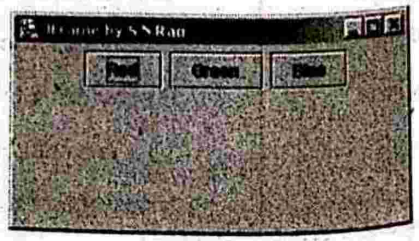
JFrame is a useful class for creating a window. JFrame is a window with a title bar and a border. JFrame class is a subclass of java.awt. Frame, this in turn is a subclass of java.awt. Window. JFrame is one of the important swing GUI components but not considered as a light weight GUI component. In contrast to many swing components, JFrame is not completely written in Java. When a window is displayed from a Java program, the window style is copied from the underlying operating system, i.e., JFrame developed on Unix is different from that of window platform. Whenever user want to draw a JFrame, the request moves to the underlying operating system for the style of the frame to be drawn that is why it is called heavy-weight component. JButton, JSlider etc., are examples for light weight swing components. When a window is no longer needed in an application, the window must be explicitly disposed. This can be achieved by calling windows dispose( ) method. User can display a message in the title bar of the frame. setSize( ) is used for dictating the size of the frame. setVisible (boolean val) decides whether the frame to appear on the monitor or not. If its value is true it appears else it won't appear.

Program on JFrame

File Name: JFDemo.java

```
import javax.swing.*;
import java.awt.*;
public class JFDemo extends JFrame
{
    JButton rb, gb, bb;
    public JFDemo( )
    {
        Container c = getContentPane( );
        c.setLayout(new FlowLayout( ));
        rb = new JButton("Red");
        gb = new JButton("Green");
        bb = new JButton("Blue");
        c.add(rb);
        c.add(gb);
        c.add(bb);
        setTitle("JFrame by S N Rao");
        setSize(300,350);
        setVisible(true);
    }
    public static void main(String args[ ])
    {
        new JFDemo( );
    }
}
```

Output



JApplet

JApplet class is the fundamental to Swing. JApplet extends Applet. Applets which use Swing should be subclasses of JApplet. JApplet is rich with functionality which is not found in Applet. JApplet supports different "passes", like content pane, glass pane and the root pane. When a component is added to an instance of JApplet, we must not invoke the add( ) method of the applet. Instead invoke add( ) for the content pane of the JApplet object. The content pane is obtained by using a method as follows:

```
Container getContentPane( )
```

A component can be added to a content pane by using add( ) method of the container as shown below.

```
void add(comp)
```

comp is nothing but the component which is to be added to the content pane.

Program on JApplet

File Name: JADemo.java

```
import javax.swing.*;
import java.awt.*;
public class JADemo extends JApplet
{
   JButton rb, gb;
   public void init()
   {
      Container c = getContentPane( );
         c.setLayout(new FlowLayout( ));
      rb = new JButton("Red");
      gb = new JButton("Green");
      c.add(rb);
      c.add(gb);
   }
}
```

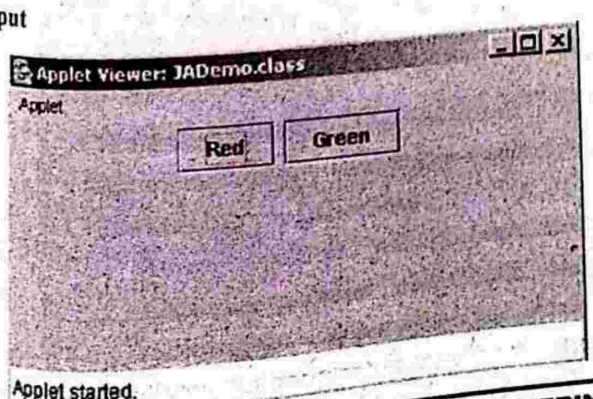
HTML file that executes above JApplet:

File Name: JA.html

```
<applet code = "JADemo.class" width="300"
                              height="350">

</applet>
```

Output



2. Light-weight Containers

JPanel

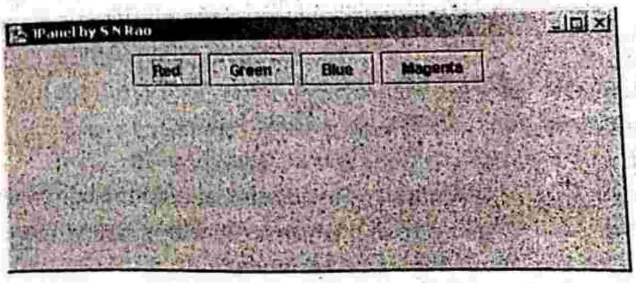
JPanel is one of the light-weight containers that is defined in swing. The light-weight containers inherit the 'JComponent' class and are generally used in organizing and managing groups of components that are related to each other. This is possible because a container can have a light-weight container within it.

Program on JPanel

File Name: JPDemo.java

```
import javax.swing.*;
import java.awt.*;
public class JPDemo extends JFrame
{
   JButton rb, gb, bb, mb;
   JPanel jp;
   public JPDemo( )
   {
      Container c = getContentPane( );
      c.setLayout(new FlowLayout( ));
      rb = new JButton("Red");
      gb = new JButton("Green");
      bb = new JButton("Blue");
      mb = new JButton("Magenta");
      jp = new JPanel( );
      jp.add(rb);
      jp.add(gb);
      jp.add(bb);
      jp.add(mb);
      c.add(jp, "North");
      setTitle("JPanel by S N Rao");
      setSize(300,350);
      setVisible(true);
   }
   public static void main(String args[ ])
   {
         new JPDemo();
   }
}
```

Output



Q21. Explain the functionality of JComponent with example. Differentiate JComponent and JPanel.

Answer :

JComponent

The JComponet class is the base for all swing components used in a window as part of the GUI. Since this class is derived from container, all of the swing components are also containers. The subclasses of JComponent define a range of standard components like menus, buttons, checkboxes etc. These classes are used for creating the GUI for our application or applet. All the swing component classes are defined in the javax.swing package and have class names which begin with J.

Class JComponent is the superclass to most Swing components. This class defines the set of methods that can be applied to any subclass of JComponent.

Swing components that subclass JComponent have many features including,

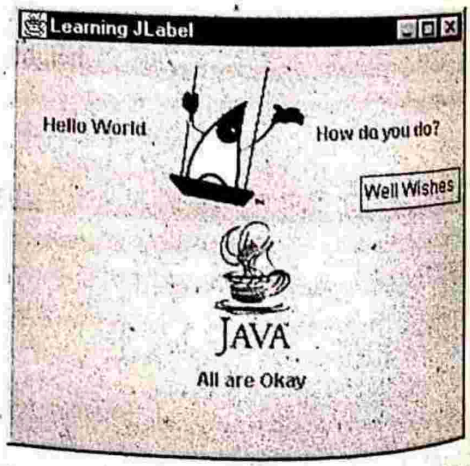
1. A pluggable look and feel that can be used to customize the look and feel when the program executes on different platforms.
2. Shortcut keys for direct access to GUI components through the use of keyboard.
3. Common event handling capabilities for cases where several GUI components initiate the same actions in a program.
4. Brief descriptions of a GUI component's purpose (called tool tips) that are displayed when the mouse cursor is positioned over the component for a short time.
5. Support for assistive(related) technologies such as Braille screen readers for blind people.
6. Support for user interface localization, customizing the user interface for display in different languages and cultural conventions.

Example

JLabel is used to display some message to the user. JLabel does not generate any events. The label can be aligned in different ways.like left, right, center etc.

```
import javax.swing.* ;
import java.awt.*; // for Container class
import java.awt.event.*; // for window closing
public class JLabelDemo extends JFrame
{
    JLabel lab1,lab2,lab3;
    public JLabelDemo( )
    {
    Container c = getContentPane( );
    c.setLayout(new FlowLayout( ));
    // layout managers discussed shortly
    // using JLabel constructor with a string argument
    lab1 = new JLabel("Hello World");
        // creating a JLabel object
    lab1.setToolTipText("Greetings");
    c.add(lab1); // to add the label to the container
    // using JLabel constructor with a string, icon
    //and alignment arguments
    ImageIcon ic1 = new ImageIcon("Duke.gif");
    lab2 = new JLabel("How do you do?", ic1,
                        SwingConstants.LEFT);
    lab2.setToolTipText("Well Wishes");
    c.add(lab2);
    // using JLabel constructor with no arguments.
    //Properties are set later
    ImageIcon ic2 = new ImageIcon("javalogo.gif");
    lab3 = new JLabel( );
    lab3.setText("All are Okay");
    lab3.setToolTipText("Quite and Peaceful");
    lab3.setIcon(ic2);
    lab3.setHorizontalTextPosition(SwingConstants.
                                        CENTER);
    lab3.setVerticalTextPosition(SwingConstants.
                                BOTTOM); // if kept in
    // comments, icon and label come in one line
    c.add(lab3);
    // to close the window, the shortest way
    add WindowListener( new WindowAdapter( )
    {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });
    setTitle("Learning JLabel");
                // to give title to the frame
    setSize(300,350); // to give size to the frame; 300
                    // pixels width and 350 height
    setVisible(true); // to make the frame visible
                    //and if false frame is not seen
    }
    public static void main(String args[ ])
    {
    new JLabelDemo( );   // just call the constructor
    }
}
```

Output



This program uses a setToolTipText( ) method of JComponent class to give description of JLabel component.

Component Vs JPanel

JComponent is a subclass of container class. It is the superclass of all the light weight swing components. Component is visible by default when first created. JPanel is a subclass of JComponent class. It is an area that can be used to group a set of components. JPanel does not have a border. JPanel can also be added to some other component like JFrame. Therefore JPanel can be used as both a component as well as a container. JPanel is visible by default when first created. The default layout for JPanel is Flowlayout.

22. How to move/drag a component placed in Swing Container? Explain.

Answer : Nov./Dec.-17(R16), Q10(b)

All the swing components in Java does not have built in drag support. JLabel is one such component and it needs to be coded to support drag functionality. The below program shows how icons can be moved/dragged. Two labels and one button are placed in the swing container content pane they are displayed as icons.

The drag support is not activated by default for the label. For this purpose a custom mouse adapter is registered for both labels. Every component has a TransferHandler class for the icon property. Both the drag sources and targets need the support of TransferHandler.

```
import javax.swing.*;
import javax.awt.Container;
import javax.awt.EventQueue;
import javax.awt.event.MouseAdapter;
import javax.awt.event.MouseEvent;
import javax.awt.event.MouseListener;
public class Move extends JFrame
{
public Move( )
{
initUI( );
}
private void initUI( )
{
ImageIcon i1 = new ImageIcon("c/src/img.png");
ImageIcon i2 = new ImageIcon("c/src/p.png");
ImageIcon i3 = new ImageIcon("c/src/smile.png");
JLabel l1 = new JLabel(i1, JLabel.CENTER);
JLabel l2 = new JLabel(i2, JLabel.CENTER);
JLabel l3 = new JLabel(i3, JLabel.CENTER);
MouseListener lt = New DragMouseAdapter( );
l1.addMouseListener(lt);
l2.addMouseListener(lt);
l3.addMouseListener(lt);
JButton b = new JButton(i2);
b.setFocusable(false);
l1.setTransferHandler(new TransferHandler("icon"));
l2.setTransferHandler(new TransferHandler("icon"));
l3.setTransferHandler(new TransferHandler("icon"));
b.setTransferHandler(new TransferHandler("icon"));
createLayout(l1, l2, l3, b);
setTitle("Icon Drag & Drop");
setDefaultCloseOperation(JFrame,EXIT_ON_CLOSE)
setLocationRelativeTo(null);
}
private class DragMouseAdapter extends MouseAdapter
{
public void mousePressed(MouseEvent me)
{
JComponent jc = (JComponent) e.getSource( );
TransferHandler handler = jc.getTransferHandler( );
handler.exportAsDrag(jc, me, TransferHandler.COPY);
}
}
private void createLayout(JComponent... arg)
{
Container pane = getContentPane( );
GroupLayout gl = new GroupLayout(pane);
pane.setLayout(gl);
gl.setAutoCreateContainerGaps(true);
gl.setAutoCreateGaps(true);
gl.setHorizontalGroup(gl.createParallelGroup
                (GroupLayout.Alignment.Center)
.addGroup(gl.createSequentialGroup( )
.addComponent(arg[0])
.addGap(30)
.addComponent(arg[1])
.addGap(30)
.addComponent(arg[2])
)
.addComponent(arg[3], GroupLayout.DEFAULT_SIZE,
GroupLayout.DEFAULT_SIZE, Integer.MAX_VALUE)
);
gl.setVerticalGroup(gl.createSequentialGroup( )
.addGroup(gl.createParallelGroup( )
.addComponent(arg[0])
.addComponent(arg[1])
.addComponent(arg[2]))
.addGap(30)
.addComponent(arg[3])
);
pack( );
}
public static void main(String[ ] args)
{
EventQueue.invokeLater(( ) ->
{
Move ex = new Move( );
ex.setVisible(true);
});
}
}
```

5.1.3 Understanding Layout Managers, FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout

Q23. What is an Layout manager? Explain different types of Layout managers.

April/May-18(R16), Q11(a)

OR

List and explain different types of layout managers with suitable examples.

Nov./Dec.-17(R16), Q10(a)

OR

Explain various layout managers in JAVA.

Nov./Dec.-18(R16), Q11(a)

OR

Explain about GridBagLayout.

(Refer Only Topic: GridBagLayout Manager)

Answer : Nov./Dec.-17(R13), Q8(b)

Layout Manager

Layout manager is responsible for arranging the components automatically within a window. It will be associated with every container object. A layout manager is an instance of a class that implements layout manager interface. The below figure shows the various AWT layout mangers.

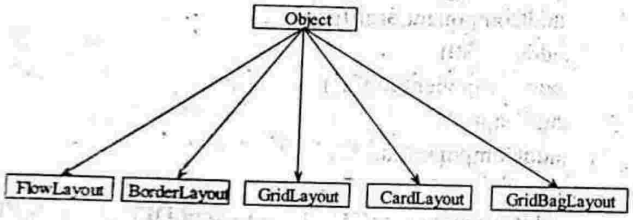


Figure: Hierarchy of Layout Managers

A layout manager can be set using setLayout( ) method. If it is not set, then default manager is used. The general form of setLayout( ) method is shown below.

void setLayout(LayoutManager obj)

Here, 'obj' indicates the object of the desired layout manager.

(a) BorderLayout Manager

BorderLayout lays out the components in four different regions i.e., North, South, East and West and one at the center. If a component is not added in any of the region, then the space of it will be occupied by the other regions. It can add up to 5 components, one for each region. The components of North and South region extend horizontally. The components of East and West region expand vertically and center region expand and take all the remaining space in the layout.

If North or South regions is not occupied then West, center and East regions expand vertically to occupy the space. If East or West region is not occupied then center region occupies the space horizontally. If Center region is left empty then no other GUI component occupies the space and the space is left empty. If all the regions are filled then the container space is full of GUI components.

This manager is useful to place the scrollbars at the borders. It is the default layout manager for Frame, Dialog and File Dialog.

Constructors

Constructor of the BorderLayout manager are as follows,

1. BorderLayout( ): It creates a default layout.
2. BorderLayout(int hor, int ver): It creates a border layout with a specified horizontal and vertical space between the components.

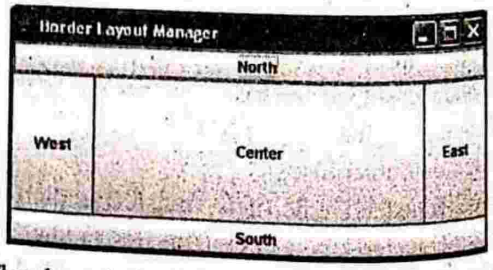
BorderLayout specifies following constants.

BorderLayout.CENTER
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.NORTH
BorderLayout.SOUTH

Example

```
import javax.swing.*;
import java.awt.*;
public class BorderDemo extends JFrame
{
    public BorderDemo( )
    {
        Container c = getContentpane( );
        BorderLayout bl = new BorderLayout( );
        c.setLayout(bl);
        JButton b1 = new JButton("North");
        JButton b2 = new JButton("South");
        JButton b3 = new JButton("East");
        JButton b4 = new JButton("West");
        JButton b5 = new JButton("Center");
        c.add(b1, bl.NORTH);
        c.add(b2, bl.SOUTH);
        c.add(b3, bl.EAST);
        c.add(b4, bl.WEST);
        c.add(b5, bl.CENTER);
        setTitle("Border Layout Manager");
        setSize(400, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.
                                 EXIT_ON_CLOSE);
    }
    public static void main(String args[ ])
    {
        new BorderDemo( );
    }
}
```

Output



The above program creates five buttons and places them into five different regions.

GridLayout Manager

GridLayout is used to lay the components in a two-dimensional grid in the form of rows and columns. Each grid cell contains one component. Components expand their size somewhat of BorderLayout components. And every component has same width and height.

Constructors

It defines the following constructors,

GridLayout( ): It creates a GridLayout with single column.

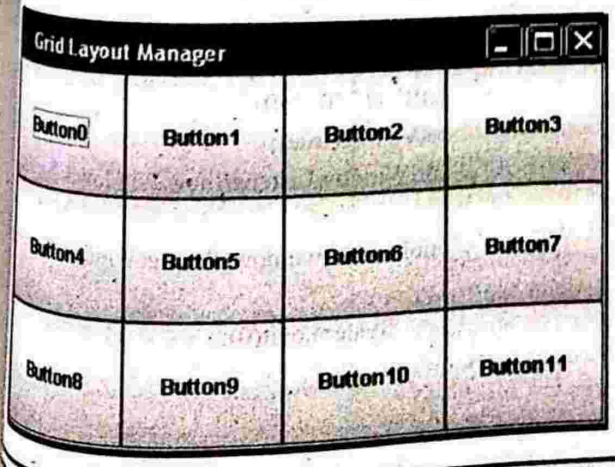
GridLayout(int rows, int columns): It creates a GridLayout with given rows and columns.

GridLayout(int rows, int columns, int hor, int ver): It creates a GridLayout with the given rows, columns and horizontal and vertical spaces.

Example

```
import java.awt.*;
import javax.swing.*;
public class GridsDemo extends JFrame
{
    public GridsDemo( )
    {
        Container c = getContentPane( );
        GridLayout g = new GridLayout(3, 4);
        c.setLayout(g);
        for(int i = 0; i<12; i++)
        c.add(new JButton("Button" +i));
        setTitle("Grid Layout Manager");
        setSize(200, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.
                                EXIT_ON_CLOSE);
    }
    public static void main(String args[ ])
    {
        new GridsDemo( );
    }
}
```

Output



(c) FlowLayout Manager

FlowLayout Manager is the default layout manger. It is considered to be simplest of all layout managers. It lays out the components from upper-left corner from left to right and from top to bottom, by separating the components with a small space. When a line is filled it moves to the next row. The method it follows is similar to the way the words are wrapped in a page i.e., from left to right.

Constructors

It has three types of constructors,

1. FlowLayout( ): It creates default layout i.e., components are placed in centre with 5 pixels space between each component.
2. FlowLayout(int how): It creates a layout with the specified alignment of the components. These alignments can be,
   FlowLayout.LEFT
   FlowLayout.CENTER
   FlowLayout.RIGHT
   FlowLayout.LEADING
   FlowLayout.TRAILING
3. FlowLayout(int how, int hor, int ver): It created a layout manger with the specified alignment as well as the horizontal and vertical spaces to be left between the components.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class FlowLayout extends JFrame implements
ActionListener
{
    FlowLayout f;
    Container c;
    JButton b1, b2, b3;
    public FlowLayout( )
    {
        f = new FlowLayout(FlowLayout.LEFT);
        c = getContentPane( );
        c.setLayout(f);
        b1 = new JButton("Left Alignment");
        b2 = new JButton("Center Alignment");
        b3 = new JButton("Right Alignment");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        c.add(b1);
        c.add(b2);
        c.add(b3);
        setTitle("Flow Layout Manager");
        setSize(500, 300);
        setVisible(true);
    }
```

```
    public void actionPerformed(ActionEvent e)
    {
        JButton btn = (JButton) e.getSource( );
        if (btn == b1)
            f.setAlignment(FlowLayout.LEFT);
        else if (btn == b2)
            f.setAlignment(FlowLayout.CENTER);
        else if (btn == b3)
            f.setAlignment(FlowLayout.RIGHT);
        f.layoutContainer(c);
    }
    public static void main(String args[ ])
    {
        new FlowLayout( );
    }
}
```

The above program initially sets the flow layout as a layout manager with the components laid as LEFT.

When the user presses a button such as "CENTER" button, then the flow layout is set as in CENTER.

(d) CardLayout Manager

CardLayout Manager is different from all other layout managers. It arranges the components in the form of deck of cards, where only the top most is visible. We can place any card from deck at the top using following methods.

```
void first(Container deck)
void last(Container deck)
void next(Container deck)
void previous(Container deck)
void show(Container deck, String cardName)
```

As their name implies, they are used to access the first, last, next or previous card. The show( ) method take out the exact card from the deck using its name.

Usually, each card is a container such as a panel and any card can have any layout manager.

Constructors

CardLayout manager has the following constructors,

1. CardLayout( ): It creates a default CardLayout.
2. CardLayout(int hor, int ver): It creates a CardLayout with specified horizontal and vertical space between the components.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class CardLayoutDemo extends JFrame
                        implements ActionListener
{
    JList hl, pl, bl, vl;
    JButton fb, lb, nb, pb;
    CardLayout cl;
    JPanel p1, p2;
    Container c;
    public CardLayoutDemo( )
    {
        c = getContentPane( );
        String hotels[ ] = {"Tajmahal", "Taj Banjara",
        "Central Court", "Viceroy"};
        String buildings[ ] = {"Chiran Palace", "Falaknuma
        Palace", "Charminar", "Golconda Fort"};
        String parks[ ] = {"Lumbini park", "Zoo park",
        "Indira park", "Sanjiva park"};
        String visits[ ] = {"Ramoji Filmcity",
        "Planetorium", "Hi-Tech city", "Shilparamam"};
        hl = new JList(hotels);
        pl = new JList(parks);
        bl = new JList(buildings);
        vl = new JList(visits);
        cl = new CardLayout;
        p1 = new JPanel( );
        p1.setLayout(cl);
        p1.add(hl, "h");
        p1.add(pl, "p");
        p1.add(bl, "b");
        p1.add(vl, "v");
        fb = new JButton("First Button");
        lb = new JButton("Last Button");
        nb = new JButton("Next Button");
        pb = new JButton("Previous Button");
        p2 = new JPanel( );
        p2.setLayout(new GridLayout(1, 4));
        p2.add(fb);
        p2.add(nb);
        p2.add(pb);
        p2.add(lb);
        fb.addActionListener(this);
        nb.addActionListener(this);
        pb.addActionListener(this);
        lb.addActionListener(this);
        c.add(p2, "South");
        c.add(p1, "Center");
        setTitle("Card Layout Manager");
        setSize(500, 250);
        setVisible(true);
        addWindowListener(new WindowAdapter()
        {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
        });
    }
```

```
public void actionPerformed(ActionEvent e)
{
String S = e.getActionCommand( );
if (S.equals("First Button"))
    cl.first(p1);
else if(S.equals("Next Button"))
    cl.next(p1);
else if (S.equals("Previous Button"))
    cl.previous(p1);
else if (S.equals("Last Button"))
    cl.last(p1);
}
public static void main(String args[ ])
{
    new CardLayoutDemo( );
}
}
```

GridBagLayout Manager

GridBagLayout is the most powerful and complex layout manager. GridBagLayout manager is similar to GridLayout which s the components in a two-dimensional array, but in this layout the grid (cell) size can vary either horizontally or vertically.

The procedure to use a GridBagLayout is to first create an object of GridBagLayout and set it as the current layout manager. en set the constraints for each component. Finally, add these components to the layout manager. It has the following form,

GridBagLayout( );

The constraints (properties) of each component are contained in an object of type GridBagConstraints class.

A component can have the following constraints associated with it, the height and width of the cell, its alignment, the acement of a component and its anchor point within the cell.

Constraints of a component are set using a setConstraints( ) method of GridBagLayout class.

It has the following form,

void setConstraints(Component comp, GridBagConstraints cons).

Where, 'comp' is a component for which constraints are specified by 'cons' object of GridBagConstraints class.

The GridBagConstrains class has different fields that can be used to set the size, placement and spacing of a component. ey are as follows,

Field/Variable	Description
gridx, gridy	These variables specify the position for the component to be placed. The top-left position is specified as gridx = 0 and gridy = 0. If no value is specified to these variables then these are considered as GridBagConstraint. RELATIVE, which places the component next to the existing one.
gridwidth gridheight	The default size of a cell is 1. These variables are used to increase the size of the cell. If a value is assigned as GridBagConstraint, REMAINDER to these variables, then it means that it is the last component to be added in its row.
fill	It specifies the resizing of a component, if component is smaller than its cell. It has the following values. GridBagConstraints.NONE (default) GridBagConstraints.HORIZONTAL GridBagConstraints.VERTICAL GridBagConstraints.BOTH
anchor	It specifies the placement of a component within a cell. It has following values. GridBagConstraints.NORTH GridBagConstraints.NORTHEAST GridBagConstraints.SOUTH GridBagConstraints.SOUTHEAST GridBagConstraints.EAST GridBagConstraints.SOUTHWEST GridBagConstraints.WEST GridBagConstraints.NORTHWEST GridBagConstraints.CENTER (default)

ipadx, ipady	ipadx pixels are added to left and right of the minimum size of the component and ipady pixels are added to the top and bottom of the component.
insets	This variable specifies the empty space between the border and components, Default value of insets are all zero.
weightx, weighty	These values are used to distribute the space when a container expands. This value ranges between 0.0 and 1.0. If the values are 0 then components are placed at the center of the container. Its behaviour is dependent on the 'fill' variable.

The program below shows the use of GridBagLayout manager.

Example

```
import java.awt.*;
import javax.swing.*;
public class GridBagDemo extends JFrame
{
    public GridBagDemo( )
    {
        Container c = getContentPane( );
        GridBagLayout gbl = new GridBagLayout( );
        GridBagConstraints gbc = new GridBagConstraints( );
        c.setLayout(gbl);
        Button btn1 = new Button("First Button");
        gbc.weightx = 1.0;
        gbc.weighty = 1.0;
        gbc.fill = GridBagConstraints.BOTH;
        gbl.setConstraints(btn1, gbc);
        Button btn2 = new Button("Second Button");
        gbc.weightx = 1.0;
        gbc.weighty = 1.0;
        gbc.fill = GridBagConstraints.BOTH;
        gbl.setConstraints(btn2, gbc);
        c.add(btn2);
        setTitle("GridBag Layout Manager")
        setSize(200, 200);
        setVisible(true);
    }
    public static void main(String args[ ])
    {
        new GridBagDemo( );
    }
}
```

The above program creates the two buttons and set the properties weightx, weighty, fill using the GridBagConstraints object. And these constraints are set using setConstraints( ) method.

5.2 EVENT HANDLING

5.2.1 The Delegation Event Model

Q24. What is delegation event model? Explain it. What are its benefits?

Answer :

Delegation Event Model

As the name indicates, the responsibility of event handling is assigned (delegated) to some special interfaces called listeners. If a listener fails to handle the event generated by a component, the event dies then and there only and does not travel through the hierarchy of containers as in the earlier model.

Delegation event model defines a standard mechanism to generate and process events. In this model, a component generates an event and sends to a listener (with which it registered earlier). The listener, which is actually waiting for an event, immediately receives it and processes it. This is a good design pattern where event handling mechanism code is entirely separated from the user interface components that generate the events. The advantage is without affecting the event handling code, we can change the code of GUI (screens) with which user interacts.

In the delegation event model, GUI component that needs event handling should register (link) with an appropriate listener to handle its events. That is, the component delegates the responsibility of event handling to a listener. When user will link or register the listener with the component, the listener is notified which takes immediate processing of the event.

The classes involved in this model are as follows,

(a) Event sources
(b) Event classes
(c) EventListener interfaces
(d) Adapter classes.

Benefits

1. It gives more performance than earlier model.
2. It is easy to operate.
3. We can build more robust and flexible programs.
4. It is known as "observer pattern" in design patterns.

5.2.2 Events, Event Sources, Event Listeners, Event Classes

Q25. Write short notes on,

(a) Events

(b) Event sources.

Answer :

(a) Events

An event is an action performed by the user. In the delegation event model, an event is an object that shows a status i.e., state changes in a source. Events are generated when the user interacts with the elements in a graphical user interface. Some of the actions that generate events are pressing a button, entering the data via keyboard, selecting an item in a list, clicking the mouse etc. These are all the events that are generated by a user by interacting with a user interface.

Events may also be generated without interacting with user interface. For example, an event may be generated when the timer expires, a counter reaches a value, a software or hardware failure occurs, or an operation is completed.

(b) Event Sources

In the delegation event model, a source is an object that makes the events to be generated. These events are generated when the internal state of that object changes from one state to another. The event sources may generate more than one event.

A source must be registered with the event listeners so that, listeners could know about the event. For that purpose each event has its own way of registration method.

Sources of events are given below,

(i) **Button:** It generates action events when the user presses a button.

(ii) **Checkbox:** It generates item events when the checkbox is selected or deselected.

(iii) **Choice:** It also generates item events when the user makes a new choice.

(iv) **List:** It generates action events when an item is double-clicked. It also generates item events when item is selected or deselected.

(v) **Menu Item:** Like list it generates both action and item events. Action event is generated when a menu item is selected and an item event is generated when a checkable menu item is selected/deselected.

(vi) **Scroll Bar:** When the scroll bar is manipulated, it generates adjustment events.

(vii) **Text Components:** Text events are generated when the user enters a character.

(viii) **Window:** It generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

These are all the user interface components that generates the events. In addition to these GUI components, events can also be generated by other components such as an applet.

Q26. Define Event. Give examples of events. Define event handler. How it handles events?

Answer :

Event

Whenever a user interacts with a GUI component, it generates an event. For example, if an user clicks over a Button, it generates an event known as ActionEvent. Following table gives components, the event they generate and the event handler that handles the events. The event handler are interfaces known as listeners and are defined in java.awt.event package.

S.No.	GUI Component	Event	Event Handler
1.	Button, TextField, Menu	ActionEvent	ActionListener
2.	Checkbox, Choice, List	ItemEvent	ItemListener
3.	TextArea	TextEvent	TextListener
4.	Scrollbar	AdjustmentEvent	AdjustmentListener
5.	Window	WindowEvent	WindowListener
6.	Mouse	MouseEvent	MouseListener, MouseMotionListener
7.	Keys on Keyboard	KeyEvent	KeyListener

In the process of event handling, the component is registered with the listener implemented by the class.

```
Button rb = new Button("Red");
rb.addActionListener(this);
```

this refers to the object of ActionListener. When the Button rb is clicked, the ActionListener takes cares of, it and calls its overridden method "actionPerformed(ActionEvent)". It is an abstract method in ActionListener interface which can be overridden in our class.

Event Handling

Event is an action performed by the user. These actions can be pressing a key, moving mouse, clicking mouse, pushing buttons etc. When such an action is performed then an event is generated, such events should be handled. The process of handling events is called event handling. There is an approach to event handling called delegation event model.

The modern approach for handling events is based on the delegation event model, that defines standard and consistent mechanisms for generating and processing events. Here a 'source' generates an event and sends it to one or more 'listeners'. The 'listener' waits until it receives an event. After receiving the event, the listener processes it and returns. In this model, listeners should register with a source for receiving an event notification. The advantage of this notifications are sent only to those listeners which are interested in receiving them.

Q27. Write short notes on Event Listeners.

Answer :

Event Listeners

A listener is an object which is notified when an event occurs. It must not only register with one or more sources for receiving notifications but also must implement methods for receiving and processing those notifications. The methods for receiving and processing events are defined in a set of interfaces found in java.awt.event.

EventListener Interfaces

Interface is a special case of an abstract class in which all methods are abstract. An interface specifies only what to do but not how to do. The inner details of methods are not given in an interface. Interfaces are used by implementing in different classes.

EventListener Class

EventListener is a class which is intended to listen to the occurrence of specific event. Whenever a source generates an event, the event source has to invoke an appropriate method suitable to respond to the event generated. This method is defined by the listener. The EventListener class implements EventListener interface methods, to respond in an appropriate way for the event generated.

Some of the EventListener interfaces are as follows,

1. **ActionListener Interface**

This interface defines a method which is called when an action event occurs.

void actionperformed(ActionEvent ae)

2. **AdjustmentListener Interface**

This interface defines a method, which is called when an adjustment of any field occurs,

void adjustmentvaluechanged(AdjustmentEvent ae)

ComponentListener Interface

This interface defines methods which are called when any component has undergone change in its size or if any component is moved from one place to another place. The prototypes of the methods are,

void componentresized(ComponentEvent ce)
void componentmoved(ComponentEvent ce)
void componentshown(ComponentEvent ce)
void componentHidden(ComponentEvent ce)

ContainerListener Interface

This interface defines methods which are called or activated when any component is added to the container or removed from the container. The methods are shown below.

void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce),

FocusListener Interface

Gaining focus of keyboard on a component means that the keyboard is currently working on that component. The FocusListener interface defines methods which are called when any component gains focus of keyboard or loses focus of keyboard.

The methods are,

void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)

ItemListener Interface

It defines a method, void itemStateChanged (ItemEvent ie) which is called when the state of an item gets changed.

KeyListener Interface

This interface defines three methods.

- void keyPressed(KeyEvent ke)
  This method is invoked or called when a key from the keyboard is pressed.
- void keyReleased(KeyEvent ke)
  This method is called when a key is released after pressing.
- void keyTyped(KeyEvent ke)

When a character is typed three events occur i.e., key is pressed, key is typed and then key is released. This method is called when a character of the key pressed is entered.

MouseListener Interface

This deals with the events that occur in mouse.

- When a mouse button is pressed and immediately released then the method.
  void mouseClicked(MouseEvent me) is invoked.
- When the mouse pointer enters any component of the container then the method,
  void mouseEntered(MouseEvent me) is called.
- When the mouse pointer exits out of any component then,
  void mouseExited(MouseEvent me) is called.
- When the mouse button is pressed,
  void mousePressed(MouseEvent me) is called.
- When a mouse button is released after pressing then,
  void mouseReleased(MouseEvent me) is called.

9. MouseMotionListener Interface

This interface detals with two methods.

- When a mouse is dragged then the method,
  void mouseDragged(MouseEvent me) is called multiple times.
- When a mouse is moved then the method,
  void mouseMoved(MouseEvent me) is called multiple times.

10. MouseWheelListener Interface

When the mouse wheel is moved a method, void mouseWheelMoved(MouseEvent me) is called. This method is defined by the MouseWheelListener Interface.

11. TextListener Interface

When a text area or text field is modified or changed a method, void textChanged(TextEvent te) is called. This method is defined by TextListener Interface.

12. WindowFocusListener Interface

This interface deals with two methods.

- When a window gains input focus then a method defined by WindowFocusListener Interface is called.
  The method is void windowGainedFocus (WindowEvent we)
- When a window looses its focus then,
  void windowLostFocus(WindowEvent we) is called.

13. WindowListener Interface

This interface defines several methods that deals with window events.

- When a window is activated,
  void windowActivated(WindowEvent we) is called.
- When a window is deactivated,
  void windowDeactivated(WindowEvent we) is called.
- When a window is opened,
  void windowOpened(WindowEvent we) is called.
- When a window is closed,
  void windowClosed(WindowEvent we) is called.
- When a window is about to be closed,
  void windowClosing(WindowEvent we) is called.
- When a window is changed into an icon,
  void windowIconified(WindowEvent we) is called.
- When a window is changed from icon to window,
  void windowDeiconified(WindowEvent we) is called.

Q28. What is an event driven programming and how is it structured? Nov./Dec.-16(R13), Q8(a)

Answer :

Event Driven Programming

The Java event model programs provide support for user control through mouse or keyboard called interactive programs. The actions that are performed by the user like mouse click or mouse movement are called events. The programs which are responsible to respond to such events are said to be event driven.

The events determine the flow of program execution in event driven programming paradigm when the event driven programs are written. The events to which the user must respond are specified by designating an object as a listener for that particular event. Therefore when the event occurs a message will be sent to the listener there by triggering a response.

For remaining answer refer Unit-V, Q27.

Q29. What are the methods supported by keyListener interface and mouseListener interface? Explain each of them with examples.

OR

How events are categorized in Java? Explain. Nov./Dec.-16(R13), Q9(b)

OR

Discuss about handling keyboard events.

(Refer Only Topics: Key Events, KeyListener)

Answer : Nov./Dec.-17(R13), Q8(a)

Java supports two types of events,

(a) Semantic Events: These events are generated by software components like Frame, Button, TextField etc.

(b) Low-level Events: These events are generated by hardware components like Keyboard and Mouse.

Mouse Events

Mouse events are handled by MouseListener(when mouse is stable) and MouseMotionListener(when mouse is in motion).

MouseListener

When a mouse is clicked, the mouse generates MouseEvent and is handled by MouseListener. The MouseListener includes five abstract methods. They are as follows,

(i) public abstract void mousePressed(MouseEvent e);

(ii) public abstract void mouseReleased(MouseEvent e);

(iii) public abstract void mouseClicked(MouseEvent e);

(iv) public abstract void mouseEntered(MouseEvent e);

(v) public abstract void mouseExited(MouseEvent e);.

The following program explains the usage of above methods.

File Name: MouseInfo.java

```
import java.awt.*;
import java.awt.event.*;
public class MouseInfo extends Frame implements MouseListener.
{
    setTitle("Mouse events by SNRao");
    setSize(200, 200);
    setVisible(true);
    addMouseListener(this);
    //Overriding each abstract method of the MouseListener
    public void mouseEntered(MouseEvent me)
    {
        setBackground(Color.cyan);
    }
        //Whenever mouse enters the frame, the background turns to cyan.
    public void mouseExited(MouseEvent me)
```

```
    {
        setBackground(Color.orange);
    }
        //Whenever mouse comes out, the background turns to orange.
    public void mousePressed(MouseEvent me)
    {
        setBackground(Color.red);
    }
        //Whenever mouse is pressed, the background turns to red.
    public void mouseReleased(MouseEvent me)
    {
        setBackground(Color.green);
    }
            //Green color is not noticeable as mouse releasing action is momentary.
    public void mouseClicked(MouseEvent me)
    {
        setBackground(Color.blue);
    }
            //Whenever mouse is clicked, the background turns to blue.
    public static void main(String args[ ])
    {
        new MouseInfo( );
    }
    }
```

A mouse click is a combination of mouse pressed and mouse released done quickly.

When mouse button is released, the background changes to green and the color change may not be detected by us as it is very momentary. MouseReleased is accompanied by mouseClicked by default.

Key Events

Key events are generated by keyboard keys and handled by KeyListener.

KeyListener

When a key is typed, the key generates KeyEvent and is handled by KeyListener. The KeyListener includes three abstract methods. They are as follows,

(i) public abstract void keyPressed(KeyEvent e);
(ii) public abstract void keyReleased(KeyEvent e);
(iii) public abstract void keyTyped(KeyEvent e);

The following program explains the usage of above methods.

File Name: KeyInfo.java

```
import java.awt.*;
import java.awt.event.*;
public class KeyInfo extends Frame implements KeyListener
{
public KeyInfo( )
{
    addKeyListener(this);
    setTitle("Using Key Events");
    setSize(200, 200);
    setVisible(true);
        //Start overriding each of the three abstract methods of the key listener
public void keyPressed(KeyEvent ke)
```

```
{
    setBackground(Color.cyan);
}
public void keyReleased(KeyEvent ke)
{
    setBackground(Color.yellow);
}
public void keyTyped(KeyEvent ke)
{
    setBackground(Color.gray);
}
public static void main(String args[ ])
{
    new KeyInfo( );
}
}
```

Q30. Explain different event classes supported by Java.

Answer :

The classes which represent events are at the core of Java's event handling mechanism. Event classes provide a consistent, easy-to-use means of encapsulating events. EventObject is at the root of the Java event class hierarchy, which is the superclass for all events. Its constructor is shown below.

EventObject(Object src)

Here,

src = Object generates this event.

The two methods which EventObject contains are getSource( ) and toString( ). The getSource( ) method returns source of the event whereas, toString( ) method returns the string equivalent of the event. The AWTEvent class is a subclass of EventObject. EventObject is a superclass of all events whereas AWTEvent is a superclass of all AWT events which are handled by the delegation event model. The java.awt.event package defines different types of events which are generated by various G.U.I elements.

1. ActionEvent Class

When a button is pressed, a menu item is selected or a list item is double clicked an ActionEvent is generated. This class defines four integer constants which can be used to identify any modifiers associated with an action event. They are,

(i) ALT_MASK
(ii) CTRL_MASK
(iii) META_MASK and
(iv) SHIFT_MASK.

In addition to these four, there is an integer constant that can be used for identifying action events i.e.,

ACTION_PERFORMED

The following are two constructors of ActionEvent.

ActionEvent(Object src, int type, String cmd)

ActionEvent(Object src, int type, String cmd, int modifiers)

2. ItemEvent Class

When a check box or a list box is clicked or when a checkable menu item is selected or deselected, an ItemEvent is generated. There are two types of item events, that are identified by the following integer constants.

DESELECTED - The user deselected an item.

SELECTED - The user selected an item.

ItemEvent defines one more integer constants which signifies a change of state i.e.,

ITEM_STATE_CHANGED

ItemEvent has the following constructors ItemEvent(ItemSelectable src, int type, Object entry, int state).

The getItem( ) method is used for obtaining a reference to the item which generated an event and getItemSelectable( ) is used for obtaining a reference to the ItemSelectable object which generated an event. The getStateChange( ) is used for returning the state change for the event. The following are their signatures.

Object getItem( )

ItemSelectable getItemSelectable( )

int getStateChange( )

3. InputEvent Class

The abstract InputEvent class is a subclass of ComponentEvent and a superclass of component input events. KeyEvent and MouseEvent are its subclasses. This class defines eight integer constants which can be used for obtaining information about any modifiers associated with this event. They are,

ALT_MASK
CTRL_MASK
BUTTON_MASK
BUTTON2_MASK
BUTTON3_MASK
ALT_GRAPH_MASK
META_MASK
SHIFT_MASK.

This class has following methods.

boolean isShiftDown( )
boolean isMetaDown( )
boolean isAltDown( )
boolean isAltGraphDown( )
boolean isControlDown( )

FocusEvent Class

When a component gains or loses input focus, a ...Event is generated. FOCUS_GAINED and FOCUS_ ...are the integer constants which identify these events. This ...is a subclass of ComponentEvent and has the following ...tructors.

FocusEvent(Component src, int type)

FocusEvent(Component src, int type, boolean temporaryFlag)

The isTemporary( ) method indicates if the focus change ...porary. If the change is temporary it returns true else ...e false.

ContainerEvent Class

When a component is added to or removed from a ...iner, a ContainerEvent is generated. COMPONENT ...DED and COMPONENT_REMOVED are the two types ...tainer events. It is a subclass of ComponentEvent and has ...llowing constructor.

ContainerEvent(Component src, int type, Component ...p)

The getContainer( ) and getChild( ) are its methods.

ComponentEvent Class

When the size, position or visibility of a component is ...nged, a ComponentEvent is generated. There are four types ...mponent events. This class defines integer constants which ...e used to identify them.

COMPONENT_HIDDEN

COMPONENT_MOVED

COMPONENT_RESIZED

COMPONENT_SHOWN

This class has the following constructor.

ComponentEvent(Component src, int type)

This class is a superclass of ContainerEvent, FocusEvent, ...owEvent, MouseEvent and KeyEvent classes. The ...owing method returns the component which generated the ...t.

Component getComponent( )

AdjustmentEvent Class

This event is generated by a scroll bar. There are five ...s of adjustment events. This class defines integer constants ...dentifying them.

BLOCK_DECREMENT

BLOCK_INCREMENT

UNIT_DECREMENT

TRACK

This class has the following constructor.

AdjustmentEvent(Adjustable src, int id, int type, int data)

This class has the following methods.

Adjustable getAdjustable( )

int getAdjustmentType( )

int getValue( )

8. WindowEvent Class

There are seven types of window events. This class defines integer constants which can be used to identify them. The following are those constants.

WINDOW_OPENED

WINDOW_ICONIFIED

WINDOW_DEICONIFIED

WINDOW_ACTIVATED

WINDOW_DEACTIVATED

WINDOW_CLOSED

WINDOW_CLOSING

This class is a subclass of ComponentEvent and has the following constructor.

WindowEvent(Window src, int type)

The following method is used to return the window object which generated the event.

Window getWindow( ).

9. TextEvent Class

These are generated by text fields and text areas when characters are entered by a user or program.TEXTVALUE_CHANGE is the integer constant defined by TextEvent. The constructor for this class is shown below.

TextEvent(Object src, int type)

10. MouseEvent Class

There are seven types of mouse events. This class defines the following integer constants which can be used to identify them.

MOUSE_PRESSED

MOUSE_RELEASED

MOUSE_CLICKED

MOUSE_DRAGGED

MOUSE_ENTERED

MOUSE_EXITED

MOUSE_MOVED

The getX( ) and getY( ) methods are used to return the X and Y coordinates of the mouse when the event occurred. getPoint( ) method is used to obtain the coordinates of the mouse. TranslatePoint( ) method is used for changing the location of the event. getClickCount( ) method obtains the number of mouse clicks for this event. The isPopupTrigger( ) method tests if this event causes a popup menu to appear on this platform.

11. KeyEvent Class

When keyboard input occurs, a KeyEvent is generated. There are three types of KeyEvent. The following integer constants are used to identify them.

KEY_TYPED

KEY_PRESSED

KEY_RELEASED

The first event occurs when a character is generated whereas, the next two events are generated when the key is pressed or released.

This class is a subclass of InputEvent and has the following constructors.

KeyEvent(Component src, int type, long when, int modifiers, int code)

KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)

There exist some other integer constants which are defined by KeyEvent. For example, VK_0 through VK_9 and VK_A through VK_Z define the ASCII equivalents of numbers and letters respectively.

This class defines many methods, but the most commonly used ones are getKeychar( ) and getKeycode( ).

Q31. Write short notes on color and font classes.

Answer :

Font Class

AWT provides flexibility by abstracting font manipulation operations and allowing dynamic font selection. The Font class encapsulates the fonts. Font class defines several methods and variables. Some of the important methods are listed below.

1. String getFamily( )
2. String getFontName( )
3. String getName( )
4. int getSize( )
5. int getStyle( )
6. boolean isBold( )
7. String toString( ).

Color Class

The first method returns the name of the font family to which the invoking font belongs. The second method returns the face name of the invoking font whereas the third method returns the logical name of the invoking font. The fourth method returns the size of invoking font and fifth method returns the style values of the invoking font. The sixth method returns true if the font includes the BOLD style value else returns false. The last method returns the string equivalent of the invoking font.

AWT color system allows us to specify the desired color. Color class encapsulates the colors. 'Color' defines various constants for specifying a number of colors. At the same time user can also create our own colors using any of the Color constructors shown below.

Color(int rgbvalue)

Color(int red, int green, int blue)

Color(float red, float green, float blue)

The Color class defines various methods which help us to manipulate colors, getRed( ), getGreen( ) and getBlue( ) methods are used for obtaining the red, green and blue components of a color respectively, getRGB( ) method is used to obtain a packed, RGB representation of a color.get.Color( ) and setColor( ) methods are used for obtaining the current color and changing the color respectively.

Q32. Write a program to create a frame window that responds to mouse clicks and key strokes.

OR

Write a program to create a frame window that responds to mouse clicks.

Nov./Dec.-18(R16), Q11(b)

OR

Write a program to create a frame window that responds to key strokes.

April/May-18(R16), Q11(b)

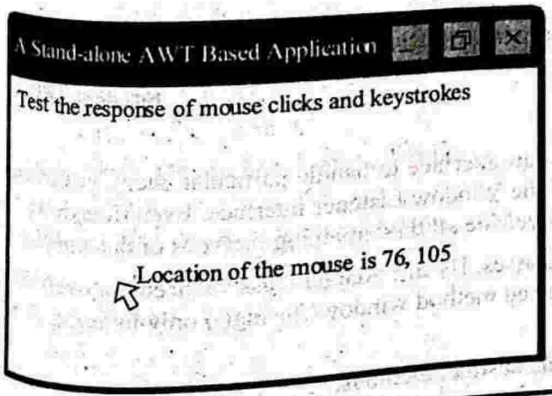
Answer :

Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//To create a frame window
public class ApInWindow extends Frame
{
   String keystrokemsg = "Test the response of mouse
                         clicks and key strokes";
   String mousemessage = " ";
   int mX = 80, mY = 80;
   public ApInWindow( )
   {
      addKeyListener(new MyKeyAdapter(this));
      addMouseListener(new MyMouseAdapter(this));
      addWindowListener(new MyWindowAdapter( ));
   }
   public void draw(Graphics gr)
   {
      gr.drawString(Keystrokemsg, 20, 60);
      gr.drawString(mousemessage, mX, mY);
   }
   //To create a window
   public static void main(String args[ ])
   {
      ApInWindow aw = new ApInWindow( );
      aw.setSize(new Dimension(400, 300));
      aw.setTitle("A stand-alone AWT based Application");
      aw.setVisible(true);
   }
}
class MyKeyAdapter extends KeyAdapter
{
   ApInWindow awin;
   public MyKeyAdapter(ApInWindow awin)
```

```
    {
        this.awin = awin;
    }
    public void TypedKey(KeyEvent k)
    {
        awin.keystrokemsg + = k.getKeyChar( );
        awin.repaint( );
    }
}
class MyMouseAdapter extends MouseAdapter
{
    ApInWindow awin;
    public MyMouseAdapter(ApInWindow awin)
    {
        this.awin = awin;
    }
    public void mouseClicked(MouseEvent m)
    {
        awin.mX = m.getX( );
        awin.mY = m.getY( );
        awin.mousemessage = "Location of the Mouse is" +
        awin.mX + "," + awin.mY;
        awin.repaint( );
    }
}
class MyWindowAdapter extends WindowAdapter
{
    public void closeWindow(WindowEvent w)
    {
        System.exit(0);
    }
}
```

Output



5.2.3 Handling Mouse and Keyboard Events

Q33. Write Java program to handle mouse and keyboard events.

Answer : *Model Paper-II, Q10(a)*

```
import java.applet.*;
import java.awt.*;
import.java.awt.event.*;
    /*<applet code="HandleMouse" width=350
                                    height=100>
    </applet>*/
public class KeyBoardMouse extends Applet
            implements MouseListener, KeyListener
{
    String msg= " ";
    int mouseX=0;
    int mouseY=0;
    public void init( )
    {
        addMouseListener(this);
        addKeyListener(this);
        requestFocus( );
    }
    public void mouseClicked(MouseEvent e)
    {
        mouseX=0;
        mouseY=10;
        msg="Mouse clicked!!"
        repaint( );
    }
    public void mousePressed(MouseEvent e)
    {
        mouseX=e.getX( );
        mouseY=e.getY( );
        msg="Mouse pressed!!";
        repaint( );
    }
    public void mouseReleased(MouseEvent e)
    {
        mouseX=e.getX( );
        mouseY=e.getY( );
        msg="Mouse released!!";
        repaint( );
    }
    public void mouseEntered(MouseEvent e)
    {
        mouseX=0;
        mouseY=10;
        msg="Mouse entered!!";
        repaint( );
    }
```

```
    public void mouseExited(MouseEvent e)
    {
        mouseX=0;
        mouseY=10;
        msg="mouse exited!!";
        repaint( );
    }
    public void keyPressed(KeyEvent e)
    {
        showStatus("Key pressed!!");
    }
    public void keyReleased(keyEvent e)
    {
        showStatus("Key pressed!!");
    }
    public void keyTyped(keyEvent e)
    {
        msg=msg + e.getKeyChar( );
        repaint( );
    }
    public void paint(Graphics g)
    {
        g.drawString(msg, mouseX, mouseY);
    }
}
```

5.2.4 Adapter Classes

Q34. What is an adapter class? Describe about various adapter classes in detail.

OR

What is an adapter class and how you can use it to make programming the handling of events easier?

Model Paper-I, Q10(b)

OR

What is an adapter class and how can adapter classes are effective?

Answer :

Adapter Class

Nov./Dec.-16(R13), Q8(b)

Adapter class is time consuming to override all the methods of an interface to handle particular event. For example, to close a window the user needs to override all the abstract methods of the Window Listener interface. Even though we need to override only one method namely windowClosing( ) we are forced to override all the remaining methods of this interface.

Java provides a solution to this problem by providing adapter classes. By this adapter class we need not override all the methods of WindowListener interface, we can just override our interested method windowClosing( ) only by using an adapter class provided for WindowListener interface.

There is an adapter class for each interface having more than one abstract methods.

Example

WindowListener has adapter class called WindowAdapter.

All adapter class provides an empty implementation for all the abstract methods of an EventListener interface. There are seven EventListener interfaces that have adapter classes in java.awt.event package. They are,

	EventListener Interface	Adapter Class	Add Method
1.	WindowListener	WindowAdapter	addWindowListener( )
2.	MouseListener	MouseAdapter	addMouseListener( )
3.	MouseMotionListener	MouseMotionAdapter	addMouseMotionListener( )
4.	ComponentListener	ComponentAdapter	addComponentListener( )
5.	ContainerListener	ContainerAdapter	addContainerListener( )
6.	FocusListener	FocusAdapter	addFocusListener( )
7.	KeyListener	KeyAdapter	addKeyListener( )

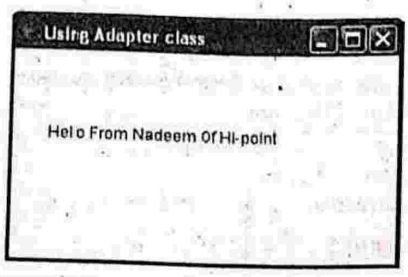
The remaining four event listener interfaces namely ActionListener, ItemListener, AdjustmentListener and TextListener do not have adapter class as they have only one abstract method to override.

Example

The program below illustrate the use of WindowAdapter class to close the window.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Example extends JFrame
{
    public Example( )
    {
        setTitle("Using Adapter class");
        setSize(300, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        HelpMe hm = new HelpMe( );
        addWindowListener(hm);
    }
    public void paint(Graphics g)
    {
        g.drawString("Hello From Nadeem Of Hi-point", 30, 100);
    }
    public static void main(String args[ ])
    {
        new Example( );
    }
}
import java.awt.event.*;
class HelpMe extends WindowAdapter
{
public void WindowClosing(WindowEvent e)
{
    System.exit(0);
}
}
```

Output



5.2.5 Inner Classes, Anonymous Inner Classes

Q35. What are inner classes? How are they useful in simplifying the code while using event adapter classes?

Answer :

Inner Classes

We can define a class within another class and such classes are known as nested classes. The nested class scope is bounded by its enclosing class scope. If suppose class Apple is defined within class Fruit, then Apple is known to fruit, but not outside of class Fruit. A nested class can access to the members (including private members) of the class in which it is nested, but the enclosing class cannot access the members of the nested class.

There exists two types of nested classes i.e., static and non-static. A static nested class has the 'static' modifier applied. Since it is static, it should access the members of its enclosing class through an object. It cannot refer to members of its enclosing class directly. Due to this problem, we seldom use static nested classes.

Inner class is the most important type of nested class. It is a non-static nested class. It can access all the variables and methods of its outer class and can refer to them directly just like other non-static members of the outer class do. An inner class is fully within the scope of its enclosing class.

To show that inner classes can simplify the code while using event adapter classes, let an applet be constructed that displays "MouseClicked" string within the status bar of the browser or applet viewer when the mouse is clicked.

Constructing this Applet Without Using Inner Class

The "MouseClickedDemo" and "MyMouseAdapter" are two top-level classes that are used in this program. The "MouseClickedDemo" class extends "Applet" and the "MyMouseAdapter" class extends "MouseAdapter". The "MouseClickedDemo" class has an init( ) method that instantiates MyMouseAdapter class. In addition to that, the init( ) method passes 'this' object as an argument to addMouseListener( ) method.

The MyMouseAdapter constructor obtains the applet reference as an argument, which is stored in an instance variable to be used later by MouseClicked( ) method. A click of a mouse will invoke showStatus( ) method of the applet using the reference of the applet that is stored already.

```
import java.applet.*;
import java.awt.event.*;
/* <applet code = "MouseClickedDemo"
                        width = 300 height = 150>

</applet>*/
public class MouseClickedDemo extends Applet
{
    public void init( )
    {
        addMouseListener(new My
                        MouseAdapter(this));

    }
}
class MyMouseAdapter extends MouseAdapter
{
    MouseClickedDemo mcd;
    public MyMouseAdapter(MouseClickedDemo mcd)
    {
        this.mcd = mcd;
    }
    public void MouseClicked(MouseEvent mev)
    {
        mcd.showStatus("Mouse Clicked");
    }
}
```

Improving Above Program Using Inner Classes

The 'InnerClassDemo' is the only top-level class and has an inner class i.e., "MyMouseAdapter" that is defined within the InnerClassDemo's scope. The 'InnerClassDemo' extends 'Applet' and 'MyMouseAdapters' extends 'MouseAdapter'.

The 'MyMouseAdapter' can access all the variables and methods that are present within the scope of the 'InnerClassDemo' class. Thus, the showStatus( ) method can be directly called from MouseClicked( ) method. Therefore, it is not required to use the reference of the applet to call showStatus( ) method and thus, there is no need to pass a reference of the invoking object to MyMouseAdapter( ).

```
import java.applet.*;
import java.awt.event.*;
/*  <applet code = "InnerClassDemo" width = 300
                                    height = 150>
</applet>*/
public class InnerClassDemo extends Applet
{
    public void init( )
    {
        addMouseListener(new MyMouseAdapter( ));
```

```
class MyMouseAdapter extends MouseAdapter
{
    public void MouseClicked(MouseEvent mev)
    {
        showStatus("Mouse Clicked");
    }
}
}
```

Anonymous Inner Classes

When a name is not assigned to an inner class, it is said to be an Anonymous Inner Class. Use of anonymous inner classes will make the event handlers to be written more easily. Consider an applet that makes use of anonymous inner class and has the same goal to display "MouseClicked" on mouse click.

The code is as follows,

```
import java.applet.*;
import java.awt.event.*;
/*<applet code = "Anonymous InnerClassDemo"
        width = 300 height = 150> </applet>*/
public class AnonymousInnerClassDemo extends Applet
{
    public void init( )
    {
        addMouseListener(new MouseAdapter( )
        {
            public void MouseClicked(MouseEvent mev)
            {
                showStatus("Mouse Clicked");
            }
        });
    }
}
```

In the above program, there is only one top-level class and that is 'AnonymousInnerClassDemo'. This class has init( ) method that calls addMouseListener( ) method. The argument passed to this method is an expression that defines an anonymous inner class and also instantiates it.

The syntax that denotes an anonymous inner class to the compiler is,

new MouseAdapter( ) {...Code that defines the anonymous inner class}

The anonymous inner class will extend MouseAdapter class. When the expression is executed, it will instantiate the anonymous inner class. All the variables and methods of the anonymous inner class. All the variables and methods of the AnonymousInnerClassDemo are accessible by the anonymous inner class as it is within its scope. Thus, the showStatus( ) method can be called directly.

5.3 A SIMPLE SWING APPLICATION

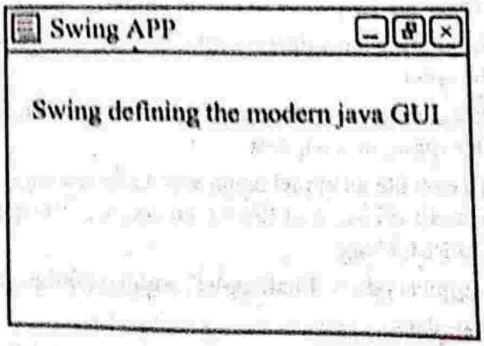
Q36. Write a simple swing application in java.

Answer :

Program

```
import java.swing.*;
class swingAPP
{
    Demo( )
    {
        JFrame jf = new JFrame("simple Application");
        jf.setSize(150, 50);
        jf.setDefaultCloseOperation (JFrame.
                                EXIT_NO_CLOSE);
        JLable jl = new JLable("Swing defining
                            the modern java GUI");
        jf.add(jl);
        jf.setVisible(true);
    }
    public static void main(String[ ] args)
    {
        swingUtilities.invokeLater/new Runnable( )
        {
            public void run( )
            {
                new Demo( );
            }
        }
    }
}
```

Output



5.3.1 Applets – Applets and HTML, Security Issues, Applets and Applications, Passing Parameters to Applets

Q37. Explain the concepts of applets.

OR

Write about applet basics and state how it runs in a window. Explain how an applet itself updates its window during an information change.

Answer : Nov./Dec.-16(R13), Q10

Applets

An applet is a Java program that is embedded in HTML document and run with the help of Java enabled browsers such as, Internet Explorer. In other words, an applet is a Java program that runs in a browser. Unlike Java applications, applets don't have a main( ) method. All applets inherit the superclass 'Applet'. An Applet class contains several methods that helps to control the execution of an applet. All applets must import java. applet and java.awt packages.

Unlike Java applications an applet does not start at main( ) method. Applets are event driven i.e., an applet waits until an event occurs. An applet is notified by AWT by calling event handler provided by the applet. Once this is done, an applet takes appropriate action and returns to the AWT. Below is an example to illustrate an applet.

Example

```
import java.awt.*;
import java.applet.*;
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
    g.drawString("Hello", 10, 20);
    }
}
```

In the above program, first two lines import the packages. The next line extends an 'Applet' class which is very important in all applets. The paint( ) method is used to draw strings in an applet. It has a method called drawString( ) that draws string "Hello" at position (10, 20) i.e., x and y pixels. After writing an applet, an applet is compiled in the same way as Java application but running of an applet is different.

There are two ways to run an applet,

1. Executing an applet within a Java-compatible web browser.
2. Executing an applet using 'applet viewer'. This executes the applet in a window.

To execute an applet using web browser, we must write a small HTML text file which contains the appropriate 'APPLET' tag.

```
<applet code = "FirstApplet" width = 300 height = 100>
</applet>
```

After creating this file, we can execute our browser and then load this file, which makes "FirstApplet' to execute.

In order to execute 'FirstApplet' with an applet viewer, we may also execute the HTML file. If suppose the HTML file is App.html, then the following command will run 'FirstApplet'.

```
C:\> appletviewer App.html.
```

There is one more method which simply includes a comment at the head of Java source code file that contains the 'APPLET' tag. Thus, our code is documented with a prototype of the necessary HTML statements and we can test our compiled applet by starting the applet viewer with the Java file.

Example

The following example shows this method,

```
import java.applet.*;
import java.awt.*;
/* <applet code = "FirstApplet" width = 150 height = 50>
</applet>*/
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
    g.drawString("Hello", 20, 20);
    }
}
```

Note

Run this program with the following command,

```
C:\>appletviewer FirstApplet.java.
```

Applet Udating a Window when there is a Change in Information

An applet will write to the window when the AWT calls update( ) or paint() method. Apart from this applet can even update the window by itself when the information changes. This is possible through repaint( ) method.

An applet can be repainted by invoking the method repaint( ). Generally, an applet can write in its window when the paint( ) method gets invoked at runtime. If the applet scroll its screen up or down, then the new information in that window must be updated. This can be performed by invoking repaint( ) method at runtime. The repaint( ) method is defined in AWT's component class and is derived from applet class.

The applet class can used to invoke paint( ) method during the execution of applet. Therefore, the other part of the window will get its information by storing the required output in a string variable and then invokes the repaint( ) method. Then, the paint( ) method is invoked automatically and displays the output information which is stored in a string variable. This can be done by using the method drawString( ) which is defined in paint( ) method.

The repaint( ) method can be declared in four forms. They are as follows,

(a) void repaint( )
(b) void repaint(int left, int top, int width, int height)
(c) void repaint(long maxi.Delay)
(d) void repaint(long maxDelay, int x, int y, int width, int height).

(a) **void repaint( )**

When the repaint( ) method is invoked with no arguments, then the entire applet window can be repainted.

(b) **void repaint(int left, int top, int width, int height)**

When the repaint( ) method is specified with the arguments such as left top, width and height, the applet window can be repainted in specified region. These arguments contain only integers and they can be declared in pixel. If the specified regions in window are repainted instead of the whole applet window, the time consumption can also be reduced.

(c) **void repaint(long maxDelay)**

When the repaint( ) method uses the maxDelay argument than the maximum number of milliseconds can be elapsed before invoking the update( ) method.

While invoking repaint( ) method, it may be invoked earlier due to some reasons. This may effect the updation of applet window. Therefore, the maxDelay( ) method can be used to delay the invoking of repaint( ) method upto specified milliseconds.

(d) **void repaint(long maxDelay, int x, int y, int width, int height)**

The other form of invoking repaint( ) method is by using the arguments such as maxDelay (which can delay the updation upto specify milliseconds), x, y (which can be used to assign the regions such as left and top) and width, height (which specify the width and height of output region).

Along these forms, repaint( ) method can be invoked mostly with no arguments. An example program to demonstrate repaint( ) method is as follows,

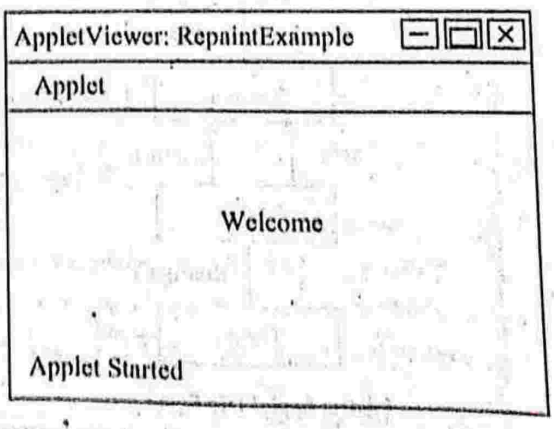
Program

```
import java.awt.*;
import java.applet.*;
public class RepaintExample extends Applet
                            implements Runnable
{
    String message = "Welcome";
    Thread th;
    boolean flag;
    public void init( )
    {
        th=null;
    }
    public void start( )
    {
        th=new Thread(this);
        flag=true;
        th.start( );
    }
    public void run( )
    {
        for(; ;)
        {
            try
            {
                repaint( );
                Thread.sleep(250);
                if(! flag)
                    break;
            }
            catch(InterruptedException ex)
            {
            }
        }
    }
    public void stop( )
    {
        flag=false;
        th=null;
    }
    public void paint(Graphics gh)
    {
    char c;
    c=message.charAt(0);
    message=message.substring(1, message.length( ));
    message += c;
    gh.drawString(message, 30, 30);
    }
}
```

HTML Code

```
<applet code= "RepaintExample" width=300
                                height=100>
</applet>
```

Sample Output



Q38. What is an applet? Explain the life cycle of Applet with a neat sketch.

Answer : April/May-18(R16), Q10(a)

Applet

For answer refer Unit-V, Q37, Topic: Applets.

Life Cycle of Applet

The life cycle of an applet consist of various phases like,

1. Applet creation
2. Applet initialization
3. Applet start
4. Applet stop
5. Applet destruction.

1. Applet Creation

Applets are created either in runtime environment by web browser or by the associated applet viewer.

2. Applet Initialization

An applet is initialized using init( ) method which will be processed after necessary parameters are passed to it.

3. Applet Start

An applet is started using start( ) method. It is automatically called after invoking init( ) method. This method is called whenever applet want to resume after interruption. Moreover, an applet is drawn using paint( ) method.

4. Applet Stop

An applet is stopped using stop( ) method. This method is called when any interruption occurs or when the controls moves to some other application from the applet.

5. Applet Destruction

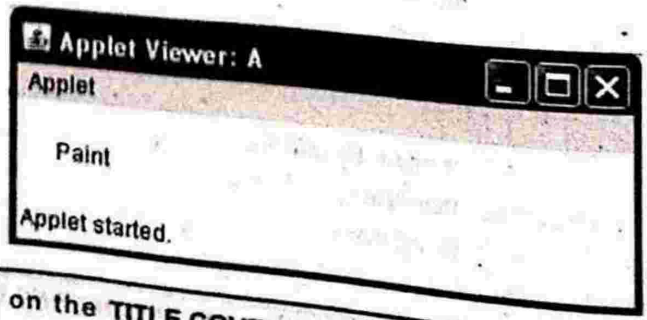
An applet is destroyed using destroy( ) method. It will be automatically called whenever a browser is called.

Figure: Applet Life Cycle

Example

```
import java.applet.Applet;
import java.awt.*;
/*
<applet code = "A" width = 300 height = 50>
</applet>
*/
public class A extends Applet
{
String s = " ";
String s1;
public void init( )
{
s = "Initialization";
display( );
}
public void start( )
{
s = "Start";
display( );
}
public void stop( )
{
s = "Stop";
display( );
}
public void display( )
{
System.out.println(s);
s1 += s;
repaint( );
}
public void paint(Graphics g)
{
g.drawString("Paint", 20 ,30);
}
}
```

Output



Q39. What are applet security issues?

Answer :

Because java applets are run on a web user's system, there are serious restrictions on the execution of an applet. An applet can be downloaded from internet and can be executed on our system. The applet may have some virus or malicious code that may corrupt our system files or may collect our personal information available on our system. For this reason, we require security for our system resources when an applet executes on our system.

Java designers restrict the applet's functionality. As a general rule, an applet cannot do any of the following,

- It is not capable of performing read or write operations on the user file system.
- It does not allow to communicate with an internet site other than the current site that served the web page which is included in the applet.
- It cannot run any program on the user's system.
- It does not load programs stored on the user's system

The stored programs includes executable programs and shared libraries.

Q40. What is Java Applet? How do applets differ from application programs? Explain with an example.

Answer :

Java Applets

Applets are special types of java programs which can run directly from a java compatible web browser. They are suitable for deploying web projects. These are small applications which are accessed on an internet server, transported over the internet, automatically installed and run as part of a web document.

Applet Architecture

Since, an applet is a window-based program, its architecture will be different from console-based programs architecture. Applets are event driven. An applet resembles a set of interrupt service routines. An applet waits for an event to occur. The AWT notifies the applet regarding an event by calling an event handler which is provided by the applet. After this, the applet should take a suitable action and then quickly return control to the AWT. In situations where, an applet requires to perform a repetitive task on its own, an additional thread of execution must be started

The user initiates interaction with an applet not the other way around. In a windowed program, when the program needs input, it will prompt the user and then calls some input methods like readLine( ). But this is not the case in applets, instead the user can interact with the applet whenever he want and however he want. All these interactions will be forwarded to the applet as events to which the applet should respond. Applets can also contain different controls like push buttons, check boxes etc., an event is generated whenever user interacts with these controls.

Java programs can be divided into two categories. They are applications and applets. Applications contain main( ) method and runs with Java interpreter, whereas applets does not contain main( ) method and runs in a browser.

Differences between an Application and an Applet

The following are the important differences between an application and an applet,

	Application		Applet
1.	Applications are stand-alone programs.	1.	Applets are not stand-alone programs.
2.	The main( ) method exists.	2.	There is no main( ) method.
3.	They need a Java interpreter for execution.	3.	They need a browser like netscape for execution.
4.	They have no hard disk accessing restrictions.	4.	They can't access hard disk.
5.	They doesn't need any security.	5.	They need topmost security for hard disk files.
6.	They can share any software which is available.	6.	They can't share any software which is available.
7.	Using plug-ins latest additions of software can be embededd.	7.	The plug-ins cannot be used by directly including browser plug-ins that are incorporated on the user's system.

Examples

(i) An example for java application program.

```
import java.io.*;
class Example
{
    public static void main(String args[ ])
    {
        System.out.println("Life is a climb");
    }
}
```

(ii) An example for java applet program.

```
import java.awt.*;
import java.applet.*;
public class Example extends Applet
{
    public void paint(Graphics gr)
    {
    gr.drawString("Life is a climb", 40, 20);
    }
}
```

Q41. Discuss about different applet display methods in brief.

Answer : Nov./Dec.-17(R16), Q11(a)

The applets in java are displayed in the windows. They make use of AWT to Perform the input and output functions. The method drawstring( ) is used to output a string to an applet. It is the member of Graphics class. It is called from update( ) or paint( ).

Syntax

```
void drawString(String msg, int x, int y)
```

Here, msg indicates the string to be displayed at x and y. This method will not detect newline characters. The x and y specify the location of text to be displayed.

The background color of the applets windows can be set by using setBackground( ).

The foreground color of the applets window can be set by using setForeground( ).

```
void setBackground(Color.newc)
void setForeground(Color.newc)
```

Here, newc indicates new color. The color class has values such as Color.blue, Color.orange, Color.yellow, Color. white, Color.gray, Color.pink etc.

Example

```
setBackground(Color.pink)
setForeground(Color.magenta)
```

The color set once can be changed when desired. The current settings of background and foreground can be obtained using the getBackground( ) and getForeground( ).

```
Color getBackground( )
Color getForeground( )
```

Example

```
import java.awt.*;
import java.applet.*;
/*<applet code = "AppletEx" width = 500 height = 500>
                                                </applet>*/

public class AppletEx extends Applet
{
public void paint(Graphics gr)
{
setBackground(Color.PINK)
setForeground(Color.MAGENTA)
gr.drawString("colors in Applet", 150, 200);
}
}
```

Q42. Explain briefly passing parameters to applets.

Answer :

In HTML, an "applet" tag is used to pass parameters to an applet. The getParameter( ) is used to retrieve these parameters. The getParameter( ) method returns a value of the given parameters in the form of string object. If integer, double or boolean values are passed as an argument then they need to be converted from their string representation into their formats. Passing of parameters to an applet is shown below,

Example

```
import java.awt.*;
import java.applet.*;
/*
<applet  code = "Demo" width = 400  height =100>
<param  name = fontName value = SanSerif>
<param  name = fontsize value = 20>
<param  name = leading value = 1>
<param  name = accountEnabled  value = true>
</ applet>
*/
public class Demo extends Applet
{
    String fontName;
    int fontSize;
    float leading;
    boolean active;
    public void start( )
    {
        String str;
        fontName = getParameter("font Name");
        if (fontName = = null)
        fontName = " Not Found";
        str = getParameter("font Size");
        try
        {
```

```
        if(str !=null)
            fontSize = Integer.parseInt(str);
        else
            fontSize = 0;
    }
    catch(NumberFormatException e)
    {
        fontSize = –1;
    }
    str = getParameter( "Leading");
    try
    {
    if(str! =null)
    leading = Float.ValueOf(str). floatValue( );
    else
    leading = 0;
    }
    catch(NumberFormatException e)
    {
    leading = –1;
    }
    str = getParameter("account Enabled");
if(str!=null)
active = Boolean.ValueOf(str). booleanValue( );
}
public void paint(Graphics g)
{
g.drawString("Font name:" + fontName, 10,20);
g.drawString("Font size:" + fontSize, 10,30);
g.drawString("Leading :" + leading, 10,40);
g.drawString("Account active:" + active, 10,50);
}
}
```

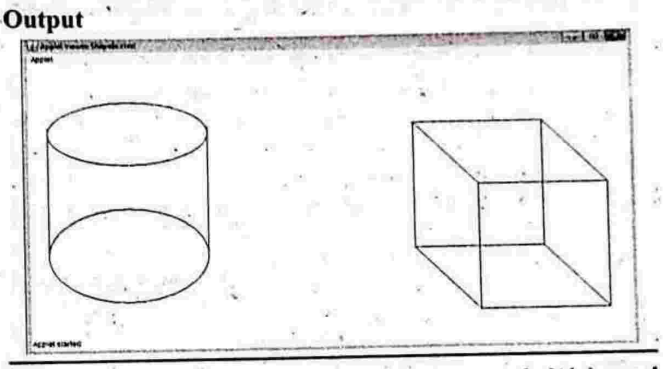
Q43. Write the applets to draw the Cube and Cylinder shapes.

Answer : April/May-18(R16), Q10(b)

```
Program
import java.awt.*;
import java.applet.*;
/*<applet code= "ShapeEx.class" height=900 width=1000>
                                    </applet>*/
public class ShapeEx extends Applet

    public void paint(Graphics g)
    {
        //Drawing Cylinder
        g.drawOval(30,60,250,100);
        g.drawLine(30,100,30,300);
        g.drawLine(280,100,280,300);
        g.drawOval(30,230,250,150);
        //Drawing a Cube
        g.drawRect(600,100,200,200);
        g.drawRect(700,200,200,200);
        g.drawLine(600,100,700,200);
        g.drawLine(800,100,900,200);
        g.drawLine(600,300,700,400);
        g.drawLine(800,300,900,400);
    }
}
```

Output



Q44. What is the difference between init( ) and start( ) methods in an Applet? When will each be executed?

Answer : Nov./Dec.-18(R16), Q10(a)

The difference between init( ) and start( ) method is that, init( ) method is used to initialize all the applet variables and also performs different start-up activities. Whereas, the start( ) method is used to start and resume the applet. If the user minimized and returned again to previous web page which consists of an applet, then the execution process of this applet can be resumed from start( ) method.

The init( ) method will be executed initially and it can be called only once in a program. Whereas, the start( ) method will be executed multiple times during the execution of a applet program.

Q45. Write the applets to draw the Cube and Circle shapes.

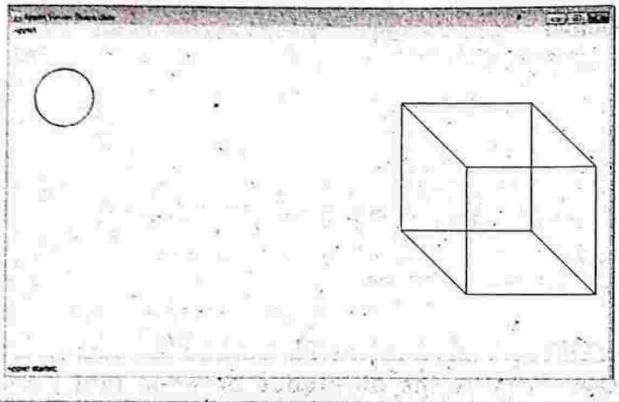
Answer : Nov./Dec.-18(R16), Q10(b)

Program

```
import java.awt.*;
import java.applet.*;
/*<applet code="Shape.class" height=900 width=1000>
                                    </applet>*/
public class Shape extends Applet
{
```

```
    public void paint(Graphics g)
    {
        // Drawing a Circle
        g.drawRoundRect(40, 50, 90, 90, 100, 100);
        //Drawing a Cube
        g.drawRect(600,100,200,200);
        g.drawRect(700,200,200,200);
        g.drawLine(600,100,700,200);
        g.drawLine(800,100,900,200);
        g.drawLine(600,300,700,400);
        g.drawLine(800,300,900,400);
    }
}
```

Output



5.3.2 Creating a Swing Applet, Painting in Swing, A Paint Example

Q46. Explain the process of creating a swing applet.

Answer :

The swing-based applet is similar to that of AWT-based applet. JApplet is a swing based applet that is derived from AWT's applet. Therefore, it has all the functionality that an applet has. It makes use of methods such as init( ), start( ), stop( ) and destroy( ) which the applet also uses. The paint( ) method need not be overridden in swing. The interaction with the swing applet components is done at event dispatching thread. Consider the below code of swing applet.

```
import Javax.swing.*;
import Java.awt.*;
import Java.awt.event.*;
/*
<applet code = "swingApplet" width = 200, height = 50>
</applet>
*/
public class swingApplet extends JApplet
{
    JButton jbal;
    JButton jbbt;
    JLabel jl;
    public void init( )
    {
    try
    {
    swingUtilites.invokeAndwait(new Runnable( )
    {
    public void run( )
    {
    makeGUI( );
    }
    });
    } catch (exception ex)
    {
    System.out.println("Unable to create
                                due to"+ex);
    }
    }
    private void makeGUI( )
    {
    setLayout (new FlowLayout( ));
    jbal = new JButton ("Alpha");
    jbbt = new JButton ("Beta");
    jbal.addActionListener (new ActionList
                                trner( )
    {
        public void actionPerformed (ActionEvent le)
    {
        jl.setText("Alpha is pressed");
    }
    });
    jbbt.addActionListener(new ActionListener( )
    {
        public void actionPerformed(ActionEvent ae)
        {
            jl.setText("Beta is pressed");
        }
    });
    add(jbal);
    add(jbbt);
    jl = new JLebel("press any button");
    add(jl);
    }
}
}
```

In the above program, the JApplet extends Applet. The init( ) method will initialize the swing components on the event dispatching thread by calling makeGUI( ). This is done through innokeandwait( ). In makeGUI( ) the buttons and labels must be created and the buttons are added with action listeners. At the end the components are added to the content pane.

47. Explain how painting is done in swing.

Answer :

Painting in swing depends upon the AWT based mechanism. In AWT the paint( ) method of component class used to draw the output on the surface of component. This paint( ) is called by the runtime system when the component needs to be rendered. It can be overridden by other application whenever the output needs to be written on component surface.

In swing Jcomponent inherits components so, all the lightweight components can inherit the paint( ) method. This method cannot be overridden in swing because it makes use of more sophisticated approach to paint by using the methods such as paintcomponent( ), paintBorder( ) and paintChildren( ). These methods will paint the specified portion of the component thereby dividing the process into three parts. The AWT paint( ) method in lightweight component will execute the call made to these methods in specified order. A subclass of the component is created and then paintComponent( ) is overridden inorder to paint the surface of component. A call to super. PaintComponent( ) is done to override paintComponent( ). Then output that is to be displayed is written. protected void paintComponent(Graphics gr) Here, gr is the graphic context that holds the output that is to be displayed. The repaint( ) is used to make the component to be painted under program control.

While the painting process is done the drawing in component surface must be restricted to the area within the border. If the output exceeds the border, swing will clip the exceeded output. It is possible even to paint within the border but will get overridden when the border is drawn. Inorder to avoid this the paintable area of the component must be computed.

Area = Component size – bordersize

Even the border width must be obtained to adjust the drawing accordingly. Border can be obtained by using getInsets( ).

Insets getInsets( )

Inset values are obtained using the fields top, bottom, left and right. They are used to compute the drawing area within the weight and height of component width and height can be obtained using getWidth( ) and getHeight( ). Subtract these values inorder to compute the usable width and weight. of component.

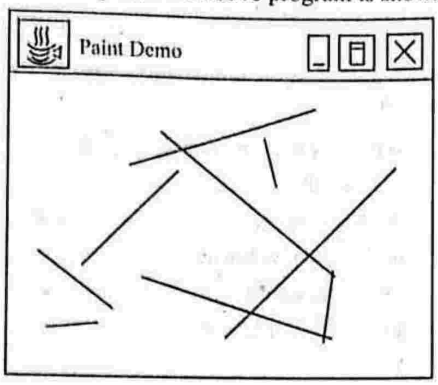
48. Write an example for paint in swing.

Answer :

A simple example would be to paint lines to a panel. The example will create a class called paintPanel that will extend JPanel.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
class painPanel extends JPanel
{
Insets in;
Random r;
PaintPanel( )
{
setBorder (BorderFactory.createLineBorder (color. Red, 5));
      r = new Random( );
}
protected void paintComponent (Graphics ga)
{
      super.paintComponent(gr);
      int  x1, y1, x2, y2;
      int  h = getHeight( );
      int  w = getWidth( );
      in = getInsets( );
      for(int i = 0; i<20; i++)
      {
            x1 = r.nextInt(w – in.left);
            y1 = r.nextInt(h – in.bottom);
            x2 = r.nextInt(w – ins.left);
            y2 = r.nextInt(h – ins.bottom);
            gr.drawLine(x1, y1, x2, y2);
            }
      }
}
class Demo
{
      JLabel jl;
      paintPanel p;
      Demo( )
      {
            JFrame jf = new JFrame("Demo :");
            jf.setSize(250, 200);
            jf.setDefaultCloseOperation(JFrame.
                                    EXIT_ON_CLOSE);
            p.new paintPanel( );
            jf.add(p);
            jf.setVisible(true);
            public static void main (String args[ ])
            {
            SwingUtilities.invokeLater(new Runnable)
            {
                public void run( )
                {
                new Demo( );
                }
            });
      }
}
}
```

In the above program, the object of the class paintPanel is used to display the lines whose endpaints are randomly generated. The output of the above program is shown below,



5.3.3 Exploring Swing Controls – JLabel and ImageIcon, JTextField

Q49. Explain icons and labels of swing with an example.

Answer : Model Paper-II, Q10(b)

Icons

In swing, icons are encapsulated by the ImageIcon class, which paints an icon from an image. Two of its constructors are shown below,

ImageIcon(String filename)

ImageIcon(URL url)

The first one uses the image in the file names "filename", whereas the second one use the image in the resource identified by "url".

The ImageIcon class implements the Icon interface which declares the methods as shown below.

1. **int getIconHeight( )**

   It returns the height of the icon in pixels.

2. **int getIconWidth**

   It returns the width of the icon in pixels.

3. **void paintIcon(Component comp, Graphics g, int x, int y)**

   It paints the icon at position x, y on the graphics context g. Additional information about the paint operation can be provided in "comp".

Labels

Swing labels are instances of the JLabel class, that extends JComponent. It can display text or/and an icon.

Below are some of its constructors.

JLabel(Icon i)

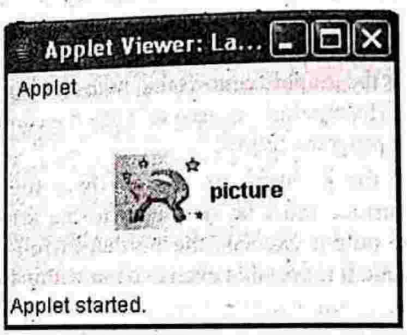
JLabel(String s)

JLabel(String s, Icon i, int align)

The following example creates and displays a label containing both an icon and a string.

```
import java.awt.*;
import javax.swing.*;
/* <applet code="LabelIcon" width = 600, height = 400>
</applet> */
public class LabelIcon extends JApplet
{
public void init( )
{
ImageIcon i = new ImageIcon("astrology.gif");
JLabel j = new JLabel("picture", i, JLabel.CENTER);
add(j);
}
}
```

Output



Q50. Explain various text components in JFC with examples.

OR

Explain about JTextField.

(Refer Only Topic: TextField)

Answer : Nov./Dec.-17(R13), Q10(b)

The different kinds of text components that are defined in JFC are as follows,

1. TextFields
2. PasswordFields
3. TextAreas.

1. TextFields

TextFields in swings provides the functionality common to swing text components. JTextFields are single-line area which are used to enter the text by the user from the keyboard. These text fields either can be used to enter the text or to display a single line information.

When the user types the data into a JTextField and presses an enter key, then an event is generated. If an event listener is present then the event is processed and the data in the JTextField can be used in the program. Class JTextField extends class JText component from package javax.swing.text, which provides many functionalities common to swing text components.

A TextField can be added to a window by adding it to a container or a panel as shown below,

```
JPanel = new JPanel( );
JTextField tf = new JTextField("Input By Default is", 40);
Panel.add(tf);
```

Where, the first parameter in JTextField is a string that is displayed inside the textfield and the second parameter is the width of the textfield.

JTextField is a text component that is mostly used for creating a single line of text in field. It makes use of Document interface for its model. When the user interacts with it, events are generated in response.

Example

```
import java.awt.FlowLayout;
import javax.swing.JApplet;
import javax.swing.JTextField;
public class TextField extends JApplet
{
    public void init( )
    {
        this.getContentPane( ).setLayout
                                (newFlowLayout( ));
        JTextField f = new JTextField( );
        add(f);
    }
}
```

2. **PasswordFields**

A special form of Textfield is a PasswordField. In the passwordfield, a password can be entered which cannot be read by the bystanders when the user is typing it. Because, the characters entered by the user are not displayed as it is, instead they are replaced with an echo character such as asterisk (i.e., *). This kind of Textfield is implemented using JPasswordField class of swing.

To create a new passwordfield, the following method is used.

JPasswordField(String text, int columns)

Example

```
JPasswordField pwdfield = JPasswordField("Enter password", 30);
```

The echo character that is to be used in the password field can be set using the setEchoChar( ) method.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code="DemoJPasswordField" width=250, height=100
</applet>*/
public class DemoJPasswordField extends JApplet
implements ActionListener
{
  JPasswordField jpf;
  public void init( )
  {
  try
  {
    SwingUtilities.invokeAndWait(new Runnable( )
    {
      public void run( )
      {
        makeGUI( );
      }
    });
  }
  catch(Exception e)
  {
    System.out.println("Not possible to create since"+e);
  }
  }
  private void makeGUI( )
  {
    setLayout(new FlowLayout( ));
    jpf = new JPasswordField( );
    jpf.addActionListener(this);
  }
  public void action Performed(ActionEvent ac)
  {
  showStatus(new String(passwordField.getPassword( )));
  }
}
```

3. **TextAreas**

TextAreas are used to obtain the user input that has multiple lines. The textareas can be created with the help of JTextArea component as shown below.

```
JTextArea textArea = new JTextArea(6,50);
                    //Creates a text area that has 6 lines of 50
```

columns each.

A user is not restricted to use the number of rows and columns specified while creating the TextArea. The 'rows' and 'columns' parameters specify only the preferred size of the TextArea. A user can input data that exceeds the size of the TextArea. When this happens the text scrolls up inorder to give place for additional text. Each line in the textarea will end with a '\n'.

The specified rows and columns for a TextArea can be changed using the setRows() and setColumns() methods, respectively.

In addition to that, the clipping of lines can be avoided, by setting the setLineWrap() method to 'true', as shown below.

```
textArea.setLineWrap(true);
```

The wrapping up of text does not effect the document, it will first provide a visual effect.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code = "DemoJTextArea" width = 250 height = 100 </applet>*/
public class DemoJTextArea extends JApplet implements ActionListener
{
    JTextArea jta;
    public void init( )
    {
    try
    {
        SwingUtilities.invokeAndWait(new Runnable( )
        {
            public void run( )
            {
                makeGUI( );
            }
        });
    }
    catch(Exception e)
    {
        System.out.println("Not possible to create since"+e);
    }
}
private void makeGUI( )
{
    setLayout(new FlowLayout u);
    jta = new JTextArea(6, 50);
    JScrollPane sp = new JScrollPane(jta);
    add(scrollPane, BorderLayout.CENTER);
    jta.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
    showStatus("\n");
}
}
```

5.3.4 The Swing Buttons – JButton, JToggleButton, JCheckBox, JRadioButton

Q51. Discuss JButton class.

Answer :

JButton Class Model Paper-I, Q11(a)

The functionality of this class is to create a push-button. It inherits 'Abstract Button' class. It has three different constructors that have their arguments as an icon or a string or both.

JButton(String s)

JButton(Icon ic)

JButton(String s, Icon ic)

An ActionEvent will be generated, each time when a button is pressed. The button's 'action command' string will be obtained when the object of ActionEvent class is passed as an argument to the method called actionperformed( ) which is defined in 'ActionListener' class. The 'action command' string will be displayed inside the button as a default. To set or get an action command setActionCommand( ) or getActionCommand( ) methods respectively, can be called. Their declaration can be done as,

String getActioncommand( ).

For each button, there will be a different action command thus, when multiple buttons are used within an application, the button that is pressed can be found out using its action command.

Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*<applet code = "ToolBarApplet.class" width = "300" height = "300">
</applet>*/
public class ToolBarApplet extends JApplet
{
    public void init( )
    {
        EventQueue.invokeLater(new Runnable( )
        {
            public void run( )
            {
                ToolBarFrame frame = new ToolBarFrame( );
                frame.setTitle("ToolBarFrame");
                frame.setSize("DEFAULT_WIDTH, DEFAULT_HEIGHT");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}
class ToolBarFrame extends JFrame
{
    public ToolBarFrame( )
    {
    //Action objects of buttons in ToolBar1
        Action newAction = new ButtonAction("New", new ImageIcon("new.gif"));
        Action openAction = new ButtonAction("Open", new ImageIcon("open.gif"));
        Action closeAction = new AbstractAction("Close", new ImageIcon("close.gif"))
        {
        public void actionPerformed(ActionEvent ae)
        {
            System.exit(0);
        }
    }
    CloseAction.putValue(Action.SHORT_DESCRIPTION, "Close");
    Action saveAction = new ButtonAction("Save", new ImageIcon("save.gif"));
    //3action objects of ToolBar2
        Action cutAction = new ButtonAction("Cut", new ImageIcon("cut.gif"));
        Action copyAction = new ButtonAction("Copy", new ImageIcon("copy.gif"));
        Action pasteAction = new ButtonAction("Paste", new ImageIcon("paste.gif"));
    //Button of ToolBar1
```

```
            JButton newButton = new JButton(newAction);
            JButton openButton = new JButton(openAction);
            JButton saveButton = new JButton(saveAction);
            JButton closeButton = new JButton(closeAction);
        //Populating ToolBar1
            JToolBar jtb1 = new JToolBar( );
            jtb1.add(newButton);
            jtb1.add(openButton);
            jtb1.add(saveButton);
            jtb1.addSeparator( );
            jtb1.add(closeButton);
            add(jtb1,BorderLayout.NORTH);
        //Populating ToolBar2
            JToolBar jtb2 = new JToolBar();
            jtb2.add(cutAction);
            jtb2.add(copyAction);
            add(jtb2, BorderLayout.SOUTH);
        }
    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 300;
    class ButtonAction extends AbstractAction
    {
        public ButtonAction(String name, Icon icon)
        {
            putValue(Action.NAME, name)
            putValue(Action.SMALL_ICON, icon)
        }
        public void actionPerformed(ActionEvent ae)
        {
            System.out.println(getValue(Action.NAME) +"Button is pressed");
        }
    }
}
```

Q52. Discuss about JToggleButton.

Answer :

JToggleButton **Model Paper-II, Q11(a)**

JToggleButton looks similar to that of JButton but the functionality of it differs. It has two states namely push and release. When this button is pressed it remains as pressed rather than popping back. In order to release it in the user should press it again. It gets released on the second push. That is it toggles between states for each button press.

Toggle buttons are the objects of class JToggleButton. The classes such as JCheckBox and JRadioButton are derived from it. JToggleButton generates an action event when it is pressed. But it generates an item event when the concept of selection is used. When the toggle button is pressed selection is performed and when it is pressed again de-selection is performed. In order to handle the item events, the interface called ItemListener must be implemented. Methods of this interface are as follows,

(i) itemStateChanged( )

(ii) getItem( )

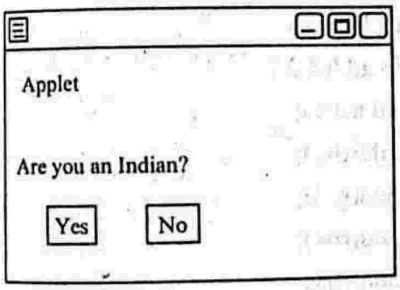
(iii) isSelected( )

The constructor of JToggleButton is as follows,

JToggle(Button(string str))

Here, str indicates the string to be contained in JToggleButton.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ToggleButton extends JApplet
{
public button( )
{
    Container c = getContentPane( );
    System.out.println("Are you an Indian?");
    ButtonGroup bg = new ButtonGroup( );
    JToggleButton[ ] tb = new JToggleButton[ ]
    {
        new JToggleButton("Yes");
        new JToggleButton("No");
    };
    c.setLayout(new FlowLayout( ));
    for(int i = 0; i < tb.length; ++i)
    {
        tg.add(tb[i]);
        c.add(tb[i]);
    }
}
```



3. Explain the following in swing controls,

(I) JCheckBox

(II) JRadioButton.

OR

Describe about applying check boxes.

(Refer Only Topic: JCheckBox)

Answer : **Nov./Dec.-17(R13), Q9(a)**

JCheckBox

A JCheckBox is a swing GUI component that provides functionality of a checkbox. Checkboxes have two states selected or not (true or false).

Checkboxes are created using JCheckBox class which subclass of JToggleButton. Hierarchy of JCheckBox and RadioButton is shown in the figure below. Here is an example creating checkboxes and formatting the text as per the ckbox selected.

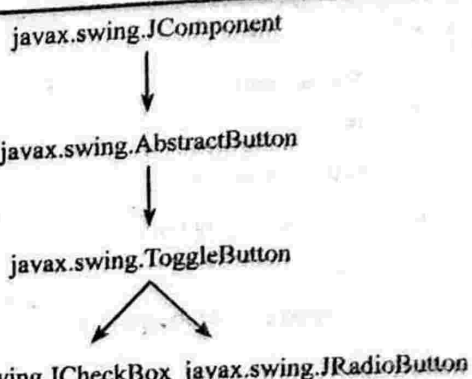


Figure: Hierarchy of JCheckBox and JRadioButton

Example

```
import java.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JCBDemo extends JFrame implements
ItemListener
{
    JTextField text;
    JCheckBox cb1, cb2, cb3;
    public JCBDemo( )
    {
        Container c = getContentPane( );
        c.setLayout(new FlowLayout( ));
        text = new JTextField("See Effect of Checkboxes
        Here", 30);
        c.add(text);
        cb1 = new JCheckBox("SanSerif");
        c.add(cb1);
        cb2 = new JCheckBox("Bold");
        c.add(cb2);
        cb3 = new JCheckBox("Italic");
        c.add(cb3);
        cb1.addItemListener(this);
        cb2.addItemListener(this);
        cb3.addItemListener(this);
        add WindowListener(new WindowAdapter( )
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            });
        setTitle("Using JCheckBox class");
        setSize(400, 200);
    }
    public void itemStateChanged(ItemEvent e)
```

```
    {
        String fname = "  ";
        int b = 0, i = 0;
        if(cb1.isSelected( ))
            fname = "SanSerif";
        else
            fname = "Monospace";
        if(cb2.isSelected( ))
            b = Font.Bold;
        else
            b = Font.PLAIN;
        if(cb3.isSelected( ))
            i = Font.ITALIC;
        else
            i = Font.PLAIN;
        Font f = new font (fname, b+i, 15);
        text.setFont(f);
    }
    public static void main(String args[ ])
    {
        new JCBDemo( );
    }
}
}
```

The above program creates three checkboxes as SanSerif, Bold and Italic by creating objects of JCheckBox class. If any checkbox is selected then its state is checked using is Selected( ) method that returns either true or false and then text is displayed as per the checkbox selected.

(ii) JRadioButton

Radio buttons are created using JRadioButton class, which are similar to check boxes that is they also have two states - selected or deselected. Radio buttons are group of buttons from which only one radio button can be selected at any time. Selecting other radio button makes the previous selected radio button to be deselected automatically. JRadioButton class is a subclass of JToggleButton class. The grouping of the radio button is made by adding the JRadioButton object to a ButtonGroup object that takes care of the state of the buttons in the group.

The following program creates a group of radio buttons and displays an appropriate message when a radio button is selected.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JRBDemo extends JFrame implements
ItemListener
{
    JRadioButton rb1, rb2, rb3, rb4, rb5;
    public JRBDemo( )
    {
        Container c = getContentPane( );
        c.setLayout(new FlowLayout( ));
        ButtonGroup grp = new ButtonGroup( );
        rb1 = new JRadioButton("S.S.C");
        rb2 = new JRadioButton("Intermediate");
        rb3 = new JRadioButton("Graduate");
        rb4 = new JRadioButton("Post Graduate");
        rb5 = new JRadioButton("Others");
        grp.add(rb1);
        grp.add(rb2);
        grp.add(rb3);
        grp.add(rb4);
        grp.add(rb5);
        c.add(rb1);
        c.add(rb2);
        c.add(rb3);
        c.add(rb4);
        c.add(rb5);
        rb1.addItemListener(this);
        rb2.addItemListener(this);
        rb3.addItemListener(this);
        rb4.addItemListener(this);
        rb5.addItemListener(this);
        add WindowListener(new WindowAdapter( )
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        }
        setTitle("creating RadioButtons");
        setSize(400, 200);
        setVisible(true);
    }
```

```
void itemStateChanged(ItemEvent e)

tring str = "  ";
if(rb1.isSelected( ))
   str = "S.S.C";
else if(rb2.isSelected( ))
   str = "Intermediate";
else if(rb3.isSelected( ))
   str = "Graduate";
else if(rb4.isSelected( ))
   str = "Post Graduate";
else if (rb5.isSelected( ))
   str = "Others";
OptionPane.showMessageDialog(null, "your
ualification is" + str);

ublic static void main(String args[ ])

  new JRBDemo( );
}
```

The above program creates five radio buttons and adds the ButtonGroup object "grp". The last statement prints appropriate message when a radio button is selected.

With a program, explain how to create popup menu's in Swing.

Nov./Dec.-16(R13), Q11(b)

wer :

```
mport javax.swing.*;
mport java.awt.event.*;
mport java.awt.*;
ublic class PopupMenu extends JFrame
{
opupMenu pm;
ublic PopupMenu( )
{
Container c = getContentPane( );
m = new JPopupMenu( );
                    //add menu items to popup
m.add(new JMenuItem("undo"));
m.add(newJMenuItem("redo"));
m.addSeparator( );
m.add(new JMenuItem("cut"));
m.add(newJMenuItem("copy"));
m.add(new JMenuItem("paste"));
addMouseListener(newMouseAdapter( )
{
ublic void mouseReleased(MouseEvent e)
{
   show(e);
   }
   });
   setTitle("PopupMenu");
   setSize(300, 300);
   setVisible(true);
   }
   void show(MouseEvent e)
   {
   if(e.isPopupTrigger( ))
   popup.show(e.getComponent( ), e.getX( ), e.getY( ));
   }
   public static void main(String args[ ])
   {
   new PopupMenu( );
   }
   }
```

5.3.5 JTabbedPane, JScrollPane, JList, JComboBox

Q55. Explain the tabbed panes swing control.

Answer : **Model Paper-I, Q11(b)**

Tabbed Pane

A tabbed pane is a component which appears as a group of folders in a file cabinet. Every folder has a title. The contents become visible when a folder is selected. Only one folder can be selected at a time. Tabbed panes are encapsulated by the JTappedPane class, which extends JComponent.

Example

```
import javax.swing.*;
/*<applet code = "JTabbedPaneExam" width = 500
height = 100>
</applet>*/
public class JTabbedPaneExam extends JApplet
{
    public void init( )
    {
        JTabbedPane j = new JTabbedPane( );
        j.addTab("Family", new FamilyPanel( ));
        j.addTab("Friends", new FriendPanel( ));
        j.addTab("Relatives",new RelativesPanel( ));
        getContentPane( ).add(j);
    }
}
class FamilyPanel extends JPanel
{
    public FamilyPanel( )
    {
```

```
            JButton b1 = new JButton("Ram Reddy");
            add(b1);
            JButton b2 = new JButton("Krishna Reddy");
            add(b2);
            JButton b3 = new JButton("Srinivas Reddy");
        }   add(b3);
}
class FriendsPanel extends JPanel
{
    public FriendsPanel( )
    {
        JCheckBox cb1 = new JCheckBox("Srinu");
        add(cb1);
        JCheckBox cb2 = new JCheckBox("Venkat");
        add(cb2);
        JCheckBox cb3 = new JCheckBox("Prasad");
        add(cb3);
    }
}
class RelativesPanel extends JPanel
{
    public RelativesPanel( )
    {
        JComboBox C = new JComboBox( );
        cj.addItem("Arvind");
        cj.addItem("Anu");
        cj.addItem("Krishna");
        add(c);
    }
}
```

Q56. Explain the following,

(a) JScrollPane

(b) Combo Boxes.

Answer :

(a) JScrollPane Nov./Dec.-17(R13), Q11

A scrollpane is a component which provides a rectangular area in which the component can be viewed. Scrollpanes are implemented in swing by the JScrollPane class, which extends JComponent. Horizontal and/or vertical scroll bars can also be provided if required.

Example

```
import java.awt.*;
import javax.swing.*;
/*<applet code = "ScrollPaneDemo" width = 300 height = 400>
</applet>*/
public class ScrollPaneDemo extends JApplet
{
    public void init( )
    {
```

```
Container contPane = getContentPane( );
contentPane.setLayout(new BorderLayout ( ));
JPanel jp = new JPanel( );
jp.setLayout(new GridLayout (20, 20));
int s = 0;
for(int i = 0; i < 20; i++)
{
    for(int j= 0; j < 20; j++)
    {
        jp.add(new JButton("Button" +s);
        ++s;
    }
}
int l = ScrollPaneConstants.VERTICAL_ SCROLLBAR_AS_NEEDED;
int m = ScrollPaneConstants.HORI-ZONTAL_ SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(jp, l, m);
contentPane.add(jsp, BorderLayout.CENTER);
}
```

Combo **Boxes**

For answer refer Unit-V, Q58.

Discuss about JList.

swer :

By default, JList its creates a list without any scrollbars. But it is possible to create a scrolling list. With scrollbars the list scrolled horizontally, vertically or both as necessary.

Scrollable list can be created in two ways, they are,

Pass the JList object to the JScrollPane constructor.

JScrollPane spane = new JScrollPane(JListobject);

Make the JList the viewport view of a JScrollPane in two steps.

JScrollPane in two steps.

JScrollPane spane = new JScrollPane( );

Spane.getViewPort( ).setView(JListobject);

mple

The following example creates a scrollbar list

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ScrollableList extends JFrame
{
    String colors[ ] = {"Red", "Black", "White", "Pink"};
    JList list;
    JScrollPane spane;
Container c = getContentPane( );
list = new JList(colors);                    //create a list
list.set SelectionModel(ListSelectionModel.SINGLE_SELECTION);
```

```
    spane = new JScrollPane(list);          //Scrollable list
    c.add(spane);
    addWindowListener(new WindowAdapter( )
    {
        public void WindowClosing(WindowEvent we)
        {
        System.exit(0);
        }
        });
        setTitle("CreateScrollableList");
        setSize(300, 350);
        setVisible(true);
        }
        public static void main(String args[ ])
        {
        new ScrollableList( );
        }
    }
```

Q58. Explain the swing control ComboBox.

Answer :

ComboBox

A combo box (or a drop-down list) provides a list of items from which user can select any one of the item. A combo box is created using JComboBox class, which inherits the JComponent class.

JComboBox is similar to choice component of AWT. An item is added to a combobox using an addItem( ) method. The program below illustrates the use of JComboBox class.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JComboBoxDemo extends JFrame
{
    JComboBox cb;
    JTextField text;
    public JComboBoxDemo( )
    {
        Container c = getContentPane( );
        c.setLayout(new Flow Layout( );
        String array[ ] = {"Idly", "Dosa", "Wada"};
        text = new JTextField("My Choice", 20);
        cb = new JComboBox(array);
        cb.addItem("Puri");
        cb.addItem("Puri");
        cb.addItem("Coffee");
        HelperListener hl = new HelperListener(text);
        cb.addItemListener(hl);
        c.add(cb);
        c.add(text);
        addWindowListener(new WindowAdapter( ))
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        setTitle("Creating Combo box");
        setSize(400, 200);
        setVisible(true);
    }
    public static void main(String args[ ])
    {
        new JComboBoxDemo( );
    }
}
class HelperListener implements ItemListener
{
    JTextField jtext;
    HelperListener(JTextField j)
    {
        jtext = j;
    }
    public void itemStateChanged(ItemEvent e)
    {
        String str = (String)e.getItem( );
        jtext.setText("You Selected" + str);
    }
}
```

In the above program, combo box is created using a JComboBox class. Items to the combo box can be added either by adding items individually using addItem( ) method by passing an array of string in the JComboBox constructor.

The above program displays all the items of a combo box. We can restrict the number of items to be displayed by adding the following statement.

```
cb.setMaximumRowCount(4);
```

It displays only 4 items and the user can scroll the combo box to select other items.

Combo box generates item events and these events are handled using a separate class called HelperListener.

Discuss about list, panels.

Nov./Dec.-17(R13), Q9(b)

For answer refer Unit-V, Q57.

For answer refer Unit-V, Q20, Topic: JPanel.

5.3.6 Swing Menus, Dialogs

Write in brief about JMenuBar.

Model Paper-II, Q11(b)

JMenuBar is derived from JComponent. It contains a menus. Every application has a menu bar which is empty the user need to add desired menus to it before it is used. only one constructor which is default. It defines several ds such as given as follows:

JMenu add(JMenu menu)

This method is used to add a JMenu to the menu bar. It returns a reference of the menu that is added. The menus are added in left to right order.

Component add(Component menu, int index)

This function is used to add a menu at a specific location. It is derived from the container. The index begins at 0, which will be the left most menu.

void remove (component menu)

This metod is used to remove the menu to which a reference is provided.

void remove(int index)

This method is used to remove the menu present at the specified index.

getMenuCount( )

This method is used to get the number of elements contained in the menu bar.

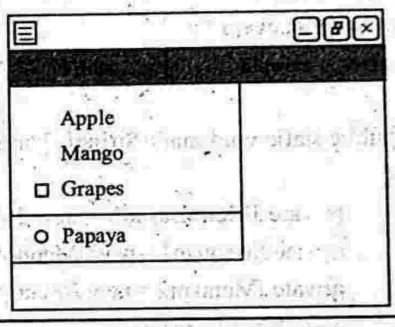
intsetJmenuBar(JmenuBar mbar)

This method is used to place the menubar at the specified position.

ple

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class Demo extends JFrame
{
    public Demo( )
    {
        JMenuBar mb = new JMenuBar( );
        setJMenuBar(mb);
        FinalContainer co = getContentPane( );
        JMenu fruits = new JMenu("fruits");
        fruits.add("Apple");
        JMenuItem i1 = new JMenuItem("Mango");
        JMenuItem i2 = new JMenuItem("Grapes");
        fruits.add(i1);
        fruits.add(i2);
        fruits.addSeparator( );
        JRadioButtonMenuItem i3 =
        new = JRadioButtonMenuItem("Papaya");
        theatres.add(i3);
        mb.add(fruits);
        JMenu flowers = new JMenu("Flowers");
        flowers.setMnemonic('F');
        final JMenuItem i4 = new JMenuItem("Rose");
        JMenuItem i5 = new JMenuItem("Marigold");
    }
    public static void main(String args[ ])
    {
        new Demo( );
    }
}
```

Output



Q61. Explain the following,

(i) JMenu

(ii) JMenuItem.

Answer :

(i) JMenu

JMenu indicates a menu in the menu bar. It contains a list of JMenuItems. A JMenu can be a part of (i.e., submenu) another JMenu. Hence, it is derived from JMenuItem. Constructor of JMenu is as follows,

JMenu(string name)

The above constructor creates a menu with the specified name. It will be empty initially. The user need to add menu items to it. Methods of JMenu are as follows,

1. **JMenuItem add(JMenuItem item)**

   This method is used to add the specified menu item to the menu. The items will be added at the end of the menu. It returns a reference to the added item.

2. **JMenuItem add(Component item, int index)**

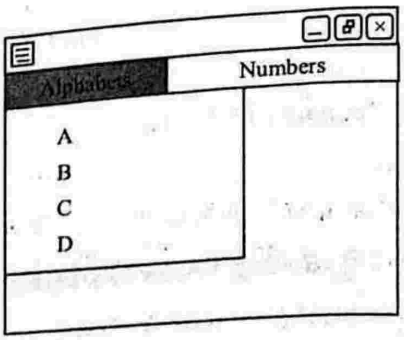
   This method is used to add the item at the specified index. It returns a reference to the added item.

3. **void addSeparator( )**
   This method is used to add a visual separator to the menu. The separator will be added at the end of the menu.
4. **void insertSeparator(int index)**
   This method is used to add a visual separator at the specified index in the menu.
5. **void remove(JMenuItem menu)**
   This method is used to remove the specified item from the menu. Reference to the item to be removed will be provided to it.
6. **void remove(int index)**
   This method is used to remove the item present at the specified index.
7. **int getMenuComponentCount( )**
   This method is used to get the number of items contained in the menu.
8. **Component[ ] getMenuComponents( )**
   This function is used to get the items contained in the menu.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class demo extends JFrame implements ActionListener
{
    public static void main(String[ ] args)
    {
        private JMenuBar mb = new JMenuBar( );
        private JMenu m1 = new JMenu("Alphabets");
        private JMenu m2 = new JMenu("Numbers");
        menuBar.add(m1);
        menuBar.add(m2);
        JMenuItem i1 = new JMenuItem("A");
        JMenuItem i2 = new JMenuItem("B");
        JMenuItem i3 = new JMenuItem("C");
        JMenuItem i4 = new JMenuItem("D");
        m1.add(i1);
        m1.add(i2);
        m1.add(i3);
        m1.add(i4);
        JMenuItem i5 = new JMenuItem("1");
        JMenuItem i6 = new JMenuItem("2");
        JMenuItem i7 = new JMenuItem("3");
        JMenuItem i8 = new JMenuItem("4");
        m2.add(i5);
        m2.add(i6);
        m2.add(i7);
        m2.add(i8);
    }
}
```

Output



(ii) JMenuItem

JMenuItem represents an element in the JMenu. It will be linked with a program action or it causes another menu to be displayed. It is considered as a special type of button that is derived from AbstractButton. An action event will be generated each time the button is pressed. The mostly used constructor defined by it is as follows,

JMenuItem(string name)

JMenuItem(icon image)

JMenuItem(string name, icon image)

JMenuItem(string name, int mn)

JMenuItem(Action action)

This constructor creates a menu item with the specified attributes. The methods of this class are as follows,

1. **void setEnabled(boolean b)**
   This method is used to enable or disable the menu item.
2. **protected void init(string text, icon icon)**
   This method is used to initialize the menu item with the given text and icon.
3. **void AddMenuKeyListener(MenyKeyListener m)**
   This method is used to add a MenuKeyListener to the menu item.
4. **protected string paramstring( )**
   This method is used to return the string representation of the menu item.
5. **void setModel(ButtonModel newModel)**
   This method is used to set the model represented by the button.
6. **void setArmed(boolean b)**
   This method is used to set the menu item as armed.

Q62. Discuss how JDialog is used.

Answer :

JDialog

JDialog is a type of swing class that is used to create a dialog. It does not inherit Jcomponent. It is a top-level container which is heavy in its weight than others. JOptionPane makes use of JDialog for creating dialogs. It has similar features to that of Jframe and it can be created and managed as a Jframe is done. It is inherited from AWT classes such as container, component, window and dialog.

The layout manager and the size of a JDialog can be set by the user. The windows of it can be hidden or shown using isible( ) method. It also allows to add a menu bar to it. JDialog can be either modal or modeless. A modal dialog pauses the plication if the dialog is closed. A modeless dialog activates the remaining parts of the application. Jdialog allows the user to ate any type of dialog. Steps for creating a JDialog are as follows,

Construct an object of type JDialog

Determine the properties for the dialog such as size, layout manager and default close policy.

Extend the content pane of the dialog by adding the components.

Make the dialog visible by making use of setVisible(true) method.

A dialog can be removed from the screen by making use of setVisible(false) or dispose( ) methods. These methods are oved from window. When a dialog is most frequently used by an application then the method setVisible(false) is used. This ded when called will delete the dialog from view. When a dialog does not have much chance to be displayed again then method dispose( ) is used. Thus method when called will cause the resources of a dialog to get released. One of the major structor defined this dialog is as follows,

JDialog(frame parent, string title, boolean isModal)

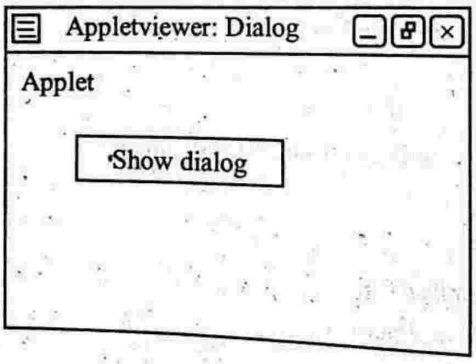
In the above statement, the parent indicates the owner of the dialog. The title indicates the title of the dialog. And the dal indicates the type of dialog whether modal or modeless. It will be passed. The values true if the dialog is modal otherwise value false is passed if the dialog is modeless.

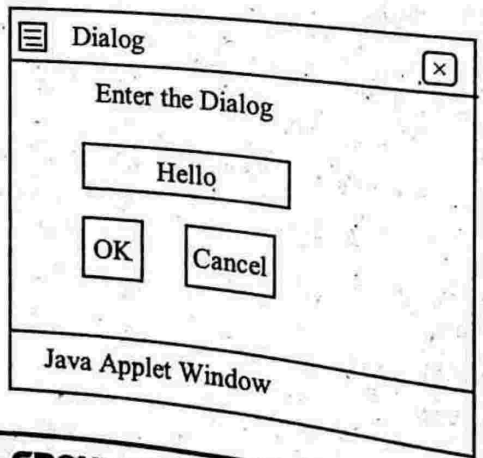
ample

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Dialog extends JApplet implements ActionListener
{
    JLabel jl = new JLabel("enter the dialog");
    JTextfield jt = new JTextfield(20);
    JButton jb = new JButton("ShowDialog");
    JButton jb1 = new JButton("OK");
    JButton jb2 = new JButton("Cancel");
    private JDialog jd = new JDialog((Frame) null, "Dialog", true);
    public void run( )
    {
        Container c = getContentPane( );
        Container c1 = jd.getContentPane( );
        c.setLayout(new flowLayout( ));
        c.add(jb);
        c1.setLayout(new flowLayout( ));
        c1.add(jl);
        c1.add(jt);
        c1.add(jb1);
        c1.add(jb2);
        jb.addActionListener(this);
        jb1.addActionListener(this);
        jb2.addActionListener(this);
    }
```

```
        public void actionPerformed(actionEvent ae)
        {
            if (ae.getSource( ) = = jb)
            {
                jd.setBounds(200, 200, 200, 160);
                jd.show( );
            }
            else if(ae.getSource( ) = = jb1)
            {
                showStatus (jt.getText( ));
                jd.dispose( );
            }
            else if (ae.getSource( ) = = jb2)
            {
                showStatus("you have pressed cancel");
                jd.dispose( );
            }
        }
    }
```

Output





Q63. Explain the process of creating modeless dialog.

Answer :

A modeless dialog can be created using JDialog. The JDialog by default creates a modeless dialog. It allows the remaining application to be in active state. This model is used when the remaining application does not depend on the user input that will be given in the dialog. The constructor for creating a modeless dialog is as follows.

JDialog(frame parent, string title)

In the above statement, parent indicate the owner of the dialog with specified title. A photo touch up is the best example of the modeless dialog. It allow several touchup filters to be selected. The user is allowed to change the filter interactively without any need to close and then reopen the dialog. Generally, the dialogs are modal, where the modeless dialogs are used they proved to be very much useful.

Generally, the modeless dialogs allow the user to switch from one window to another window in an application before it is closed.

Example

```
import java.awt.Dialog;
import java.awt.flowLayout;
import javax.swing.JDialog;
import javax.swing.JFrame;
public class ModelessDialog
{
    public static void main(String[ ] args)
    {
        final JFrame jf = new JFrame("parent frame");
        jf.setLayout(new flowLayout( ));
        jf.setDefaultCloseOperation(JFrame.EXIT_CLOSE);
        jf.setBounds(100, 100, 200, 200);
        jf.setVisible(true);
        JDialog jd = new JDialog(jf, "Modeless Dialog");
        jd.setBounds(100, 100, 200, 200);
        jd.setVisible(True);
        JDialog jd1 = new JDialog(jf, "Document_ModalDialog,"; Dialog.ModalityType.DOCUMENT_MODAL);
        JD2.SETBOUNDS(200, 200, 200, 100);
        JDIALOG JD2 = NEW JDIALOG(JD2, "MODELESSDIALOG2");
        JD2.SETBOUNDS(300, 300, 200, 100);
        JD1.SETVISIBLE(TRUE);
        JD2.SETVISIBLE(TRUE);
    }
}
```