Module 5 Questions Solution

- What are applets? Explain the different stages in the life cycle of an applet.
- What is an applet? With a skeletal code explain the methods that constitute the life cycle of an applet.
- What are applets? Demonstrate how to pass parameters for name, font size and type conversion in applet.

1. What are applets? OR Define Applet.

A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the internet and run. An applet is typically embedded inside a web page and runs in the context of a browser. An applet must be a subclass of the java.applet.Applet class. The Applet class provides the standard interface between the applet and the browser environment. An applet doesn't have a main method. Instead, there are several special methods that serve specific purposes. appletviewer is a program that can run applets.

2. What are the two types of applet?

There are two types of Applets.

1. The first are those based directly on the Applet class. These applets use the Abstract Window Toolkit (AWT) to provide the graphic user interface (or use no GUI at all). This style of applet has been available since Java was first created.
2. The second type of applets are those based on the Swing class JApplet. Swing applets use the Swing classes to provide the GUI. Swing offers a richer and often easier-to-use user interface than does the AWT. Thus, Swing-based applets are now the most popular.

3. Explain Applet Architecture.

An applet is a window-based program. Its architecture is different from the console-based programs. The key concepts of applet architecture are:

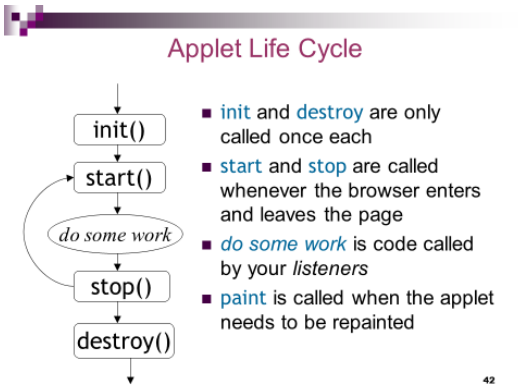
- First, applets are event driven. An applet waits until an event occurs. The run-time system notifies the applet about an event by calling an event handler that has been provided by the applet. Once this happens, the applet must take appropriate action and then quickly return. The applet must perform specific actions in response to events and then return control to the run-time system. In those situations in which your applet needs to perform a repetitive task on its own, you must start an additional thread of execution.
- Second, the user initiates interaction with an applet, not the other way around. In a non-windowed program, when the program needs input, it will prompt the user and then call some input method. In an applet, the user interacts with the applet as and when required. For example, when the user clicks the mouse inside the applet's window, a mouse-clicked event is generated. Applets can contain various controls, such as push buttons and check boxes. When the user interacts with one of these controls, an event is generated.

4. Explain the different stages in the life cycle of an applet. OR
List and discuss Applet initialization and termination methods.

Life Cycle of an Applet consist of following stages: It can *initialize* itself. It can *start* running. It can *stop* running. It can perform a *final cleanup*, in preparation for being unloaded.

- public void init(): This method is intended for whatever initialization is needed for an applet.
- public void start(): This method is automatically called after init method. It is also called whenever user returns to the page containing the applet after visiting other pages.
- public void paint(Graphics g): This method is called by the browser after init() and start(). Re-invoked whenever the browser redraws the screen. (Typically when part of the screen is obscured and then re-exposed). This method is where user level drawings are placed.
- public void stop(): This method is automatically called whenever the user moves away from the page containing applets. This method can be used to stop an animation.
- public void destroy(): This method is only called when the browser shuts down normally.

Following diagram depicts the various stages of Applet life cycle.



5. Explain the skeleton of an applet. Or
With a skeletal code explain the methods that constitute the life cycle of an applet. OR
With a skeletal code explain the methods that constitute the life cycle of an applet.

```
// An Applet skeleton.
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/
public class AppletSkel extends Applet {
       // Called first.
       public void init() { // initialization }
       /* Called second, after init.  Also called whenever the applet is restarted. */
        public void start() {  // start or resume execution }
       // Called when the applet is stopped.
        public void stop() { // suspends execution }
       /* Called when applet is terminated.  This is the last method executed. */
       public void destroy() { // perform shutdown activities }
       // Called when an applet's window must be restored.
       public void paint(Graphics g) { // redisplay contents of window }
}
```

See Question 2 for the Explanation of above methods.

6. Demonstrate how to pass parameters for name, font size and type conversion in applet. OR
Explain the applet architecture and demonstrate how to pass parameters for font size, font name, and type conversion in applets.
(For Applet architecture refer question 3 above)

An APPLET tag in HTML allows you to pass parameters to your applet. To retrieve a parameter, **getParameter( )** method is used. It returns the value of the specified parameter in the form of a **String** object. Thus, for numeric and **boolean** values, you will need to convert their string representations into their internal formats. Here is an example that demonstrates passing parameters:

```
// Use Parameters
import java.awt.*;
import java.applet.*;
/*
<applet code="ParamDemo" width=300 height=80>
<param name=fontName value=Courier>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
</applet>
*/
public class ParamDemo extends Applet{
       String fontName;
       int fontSize;
       float leading;
       boolean active;
       // Initialize the string to be displayed.
       public void start() {
```

```
                String param;
                fontName = getParameter("fontName");
                if(fontName == null)
                        fontName = "Not Found";
                param = getParameter("fontSize");
                try {
                        if(param != null) // if not found
                                fontSize = Integer.parseInt(param);
                        else
                                fontSize = 0;
                } catch(NumberFormatException e) {
                        fontSize = -1;
                }
                param = getParameter("leading");
                try {
                        if(param != null) // if not found
                                leading = Float.valueOf(param).floatValue();
                        else
                                leading = 0;
                } catch(NumberFormatException e) {
                        leading = -1;
                }
                param = getParameter("accountEnabled");
                if(param != null)
                        active = Boolean.valueOf(param).booleanValue();
        }
        // Display parameters.
        public void paint(Graphics g) {
                g.drawString("Font name: " + fontName, 0, 10);
                g.drawString("Font size: " + fontSize, 0, 26);
                g.drawString("Leading: " + leading, 0, 42);
                g.drawString("Account Active: " + active, 0, 58);
        }
}
```

7. What are the two types of applet? Explain the skeleton of an applet. Enlist applet tags.

Refer Question 2 for 2 types of applets.

Refer Question 5 for skeleton of an applet.

Explain Applet Tags OR
Enlist HTML Applet Tags

APPLET tag is used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page. The syntax for APPLET tag is shown here.

```
< APPLET
        [CODEBASE = codebaseURL]
        CODE = appletFile
        [ALT = alternateText]
        [NAME = appletInstanceName]
        WIDTH = pixels HEIGHT = pixels
        [ALIGN = alignment]
        [VSPACE = pixels] [HSPACE = pixels]
        [< PARAM NAME = AttributeName VALUE = AttributeValue>]
        [< PARAM NAME = AttributeName2 VALUE = AttributeValue>]
        . . .
</APPLET>
```

Each part is described below:

- **CODEBASE:** It is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file specified by the CODE tag.
- **CODE:** CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file.
- **ALT:** The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets.
- **NAME:** NAME is an optional attribute used to specify a name for the applet instance.
- **WIDTH and HEIGHT:** WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.
- **ALIGN:** ALIGN is an optional attribute that specifies the alignment of the applet. Possible values are: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, etc.

- **VSPACE and HSPACE:** These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet.
- **PARAM NAME and VALUE:** The PARAM tag allows you to specify applet-specific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

8. Give the different forms of repaint method.

Different forms of repaint() method:
Whenever an applet needs to update the information displayed in its window, it calls **repaint()**. The **repaint()** method is defined by the AWT calls **update( )** method, which calls **paint()**.

The **repaint( )** method has four forms:
1. void repaint( ): This causes the entire window to be repainted.
2. void repaint(int *left*, int *top*, int *width*, int *height*): it specifies a region that will be repainted.
3. void repaint(long *maxDelay*): Here, *maxDelay* specifies the maximum number of milliseconds that can elapse before **update()** is called.
4. void repaint(long *maxDelay*, int *x*, int *y*, int *width*, int *height*): It specifies the region to be repainted and the maximum delay to update.

9. Write a. Java Applet that sets the background color to cyan and foreground color to red and outputs a string message "A simple Applet".

```
// An applet that sets the foreground and background colors and outputs a string.
import java.awt.*;
import java.applet.*;
/*
<applet code="A Simple Applet" width=300 height=50>
</applet>
*/
public class Sample extends Applet{
       String msg;
       // set the foreground and background colors.
       public void init() {
              setBackground(Color.cyan);
              setForeground(Color.red);
              msg = "A Simple Applet";
       }

       // Display msg in applet window.
       public void paint(Graphics g) {
              g.drawString(msg, 10, 30);
       }
}
```

10. Write an applet program to display the message "VTU BELGAUM". Set the background color to cyan and foreground color to red.

Same as Question 9 solution, except one change i.e. modify the msg in method init():
`msg = "VTU BELGAUM";`

11. Write a JAVA applet that continuously plays an audio clip named "anthem.wav" loaded from applet's parent directory. Provide the necessary HTML file to run this applet.

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

public class PlayAudio extends Applet{
  // set the foreground and background colors.
  public void init() {
       AudioClip aClip;
       aClip = this.getAudioClip(this.getCodeBase(), " anthem.wav");
       aClip.play();     //Play it once.

  }
```

```
  // Display code and document bases.
  public void paint(Graphics g) {
    showStatus("Anthem Demo");
  }
}
```

HTML file to run the applet

```
<html>
<body>
<applet code="PlayAudio" width=300 height=50>
</applet>
</body>
</html>
```

12. Write a program using an applet which will print "key pressed" on the status window when you press the key, "key released" on the status window when you release the key and when you type the characters it should print "Hello" at co-ordinates (50, 50) on Applet.

```
// Demonstrate the key event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="SimpleKey" width=300 height=100>
  </applet>
*/

public class SimpleKey extends Applet
  implements KeyListener {

  String msg = "";
  int X = 50, Y = 50; // output coordinates

  public void init() {
    addKeyListener(this);
  }

  public void keyPressed(KeyEvent ke) {
    showStatus("Key Pressed");
  }

  public void keyReleased(KeyEvent ke) {
    showStatus("Key Released");
  }

  public void keyTyped(KeyEvent ke) {
    msg += "Hello";
    repaint();
  }

  // Display keystrokes.
  public void paint(Graphics g) {
    g.drawString(msg, X, Y);
  }
}
```

13. Develop an applet to create a label, a text field and 4 check boxes with the caption "red", "green", "blue" and "yellow".

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JCheckBoxDemo extends JApplet
       JLabel jlab;
       public void init() {
              try {
                     SwingUtilities.invokeAndWait( new Runnable() {
                            public void run() {
                                   makeGUI();
                            }
```

```
                } );
            } catch (Exception exc) {
            System.out.println("Can't create because of " + exc);
            }
    }

    private void makeGUI() {
            setLayout(new FlowLayout());

            // Create the label and add it to the content pane.
            JLabel jlab = new JLabel("Select Colour");
            add(jlab);

            // Create the text filed and add it to the content pane.
            JTextField jtf = new JTextField(15);
            add(jtf);


            // Create and Add 4 check boxes to the content pane.
            JCheckBox cb = new JCheckBox("Red");
            add(cb);

            cb = new JCheckBox("Green");
            add(cb);

            cb = new JCheckBox("Blue");
            add(cb);

            cb = new JCheckBox("Yellow");
            add(cb);

     }
}
```

14. What are swings? Provide any two typical applications of swings. OR
What is swing? List the main swing features. OR
What is swing? Explain important features of swing. OR
How AWT is different from Swings? What are the two key features of it? Explain. OR
What are the deficiencies of AWT that are overcome by swings? Explain the key features of swings. OR
Differentiate between AWT and swings.

What is Swing?

Java Swing is a Set of classes that provides powerful and flexible GUI components. Swing is built on the AWT. It eliminates a number of limitations and deficiencies inherent in AWT. Swings are:

- Lightweight- Not built on native window-system windows.
- Provide much bigger set of built-in controls. Trees, image buttons, tabbed panes, sliders, toolbars, etc.
- More customizable. Can change border, text alignment, or add image to almost any control. Can customize how minor features are drawn.
- Provide "Pluggable" look and feel. Can change look and feel at runtime, or design own look and feel.

What are the deficiencies of AWT that are overcome by swings?

Swing is developed to overcome the deficiencies present in Java's original GUI system AWT. AWT translates various components into their corresponding platform specific equivalents. Since AWT components use native code resources they are referred to as heavyweights and possess following deficiencies:

- Because of variations between operating systems a component might look, or even act differently on different platforms.
- Look and feel of each component is fixed (defined by the underlying platform) and cannot be changed easily.
- Use of heavy weight components caused some frustrating restrictions. Heavyweight component is always rectangular and opaque

Swing is built on the AWT. It eliminates a number of limitations inherent in AWT.

How AWT is different from Swings? Or Differentiate between AWT and swings.

AWT	Swing
AWT components are Heavyweight component.	Swings are called light weight component because swing components sits on the top of AWT components and do the work.
AWT components require java.awt package	Swing components require javax.swing package
AWT components are platform dependent	Swing components are made in purely java and they are platform independent
AWT is a thin layer of code on top of the OS	Swing is much larger. Swing also has very much richer functionality

Important/Key features of Swing:

Following are the two key features of Swing:

1. Swing components are lightweight
   - They are written entirely in Java and they do not map directly to platform specific peers
   - Light weight components are more efficient and more flexible
   - Look and feel is determined by swing and not by the underlying OS
2. Swing supports a pluggable look and feel
   - It is possible to separate the look and feel of component from the logic of the component
   - It is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component.
   - The look and feel can be changed dynamically at run time

Two typical applications of swings

1. Graphical User Interface designing for JAVA desk top applications
2. Graphical User Interface designing for JAVA Applets

15. Explain various components of the swing package. OR
Explain the components and containers in the swings. OR
List its components and containers. OR
What is a swing? Explain the components and containers in the swings. OR
Discuss about swing features. List its components and containers. OR
(For Swing Definition and Features refer question 14 above)

Swing Components: A Swing component is an independent visual control, such as push button, lable or list. In general Swing components are derived from JComponent class.

- JComponent supports the pluggable look and feel.
- Swing components are represented by classes defined within the package javax.swing
- Examples of Swing component classes
  - JApplet, JButton, JCheckBox, JColorChooser, JComboBox,
  - JComponent, JDialog, JFrame, JLable, JList, JMenu, JPanel, JRadioButton,
  - JScrollBar, JTable, JTree, JWindow etc

Swing Containers: A Swing container is a special kind of component that holds other components. In order for a component to be displayed it must be held within a container. Since containers are components, a container can also hold other containers. Swing defines two types of containers

1. Top level containers
   - Example: JFrame, JApplet, JWindow, JDialog
   - They do not inherit from JComponent, instead they inherit from AWTs Component and Container classes
   - They are heavyweight containers
   - They appear at the top of the containment hierarchy
   - Every containment hierarchy must begin with a top level component
     - The most commonly used for the applications is JFrame
     - The one used for applets is JApplet

2. Light Weight containers
    - Inherit from JComponent
    - Example JPanel
    - Often used to organize and manage group of related components
    - Light weight containers can be contained within another container

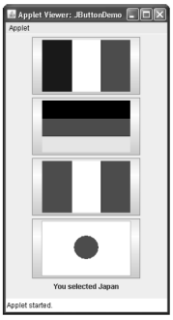
16. Describe the different types of swing buttons. OR
Name and explain different types of swing buttons. Give their syntax.

Swing defines four types of buttons. All are subclasses of the **AbstractButton** class.

1. JButton
2. JToggleButton
3. JCheckBox
4. JRadioButton

JButton:

- The JButton class provides the functionality of a push button.
- JButton allows an icon, a string, or both to be associated with the push button. Three of its constructors are:
  JButton(Icon *icon*)
  JButton(String *str*)
  JButton(String *str*, Icon *icon*)
- When the button is pressed, an ActionEvent is generated. This ActionEvent object passed to the actionPerformed( ) method of the registered ActionListener.



JToggleButton:

- A Toggle button has two states i) Pushed ii) Released
- When you press the toggle button it stays pressed, when you press a second time it releases
- By default the button is in the off position
- JToggleButton is super class of JCheckBox and JRadioButton classes
- Following constructor creates a toggle button that contains the text passed in str.
  JToggleButton(String str)
- JToggleButton generates an action event each time it is pressed. Unlike JButton, however, JToggleButton also generates an item event.

JCheckBox:

- The JCheckBox class provides the functionality of a check box. Its immediate superclass is JToggleButton.
- JCheckBox defines several constructors. One is
  JCheckBox(String *str*)
  It creates a check box that has the text specified by *str* as a label.
- Other constructors let you specify the initial selection state of the button and specify an icon.
- When the user selects or deselects a check box, an ItemEvent is generated. ItemEvent is passed to the itemStateChanged() method defined by ItemListener.

JRadioButton:

- Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time.
- **JRadioButton** class extends **JToggleButton**.
- **JRadioButton** provides several constructors. The one is shown here:
  JRadioButton(String *str*)
- A button group is created by the **ButtonGroup** class. Elements are then added to the button group using method:
  void add(AbstractButton *ab*)
- A**JRadioButton** generates action events, item events, and change events each time the button selection changes.
- In following example three radio buttons are created and then added to a button group. Pressing a radio button generates an action event, which is handled by **actionPerformed()**.

17. Create a swing application having two buttons named alpha and beta. When either of buttons pressed, it should display "alpha pressed" and "beta pressed" respectively.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class EventDemo {
       JLabel jlab;
       EventDemo() {
              // Create a new JFrame container.
              JFrame jfrm = new JFrame("An Event Example");

              // Specify FlowLayout for the layout manager.
              jfrm.setLayout(new FlowLayout());

              // Give the frame an initial size.
              jfrm.setSize(220, 90);

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

              // Make two buttons.
              JButton jbtnAlpha = new JButton("Alpha");
              JButton jbtnBeta = new JButton("Beta");
              // Add action listener for Alpha and Beta
               jbtnAlpha.addActionListener(new ActionListener() {
                     public void actionPerformed(ActionEvent ae) {
                            jlab.setText("Alpha was pressed.");
              }
              });
              jbtnBeta.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent ae) {
                            jlab.setText("Beta was pressed.");
                     }
              });
              // Add the buttons to the content pane.
              jfrm.add(jbtnAlpha);
              jfrm.add(jbtnBeta);

              // Create a text-based label.
              jlab = new JLabel("Press a button.");
              // Add the label to the content pane.
```

```
		jfrm.add(jlab);

		// Display the frame.
		jfrm.setVisible(true);
	}
	public static void main(String args[]) {
		// Create the frame on the event dispatching thread.
		SwingUtilities.invokeLater(new Runnable() {
		public void run() { new EventDemo(); }
		});
	}
}
```

18. Create swing applet that has two buttons named alpha and beta. When either of the buttons pressed, it should display "alpha vas pressed" and "beta was pressed", respectively.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MySwingApplet extends JApplet {
	JButton jbtnAlpha;
	JButton jbtnBeta;
	JLabel jlab;

	// Initialize the applet.
	public void init() {
		try {
		SwingUtilities.invokeAndWait( new Runnable () {
				public void run() {
					makeGUI(); // initialize the GUI
				}
		});
		} catch(Exception exc) {
			System.out.println("Can't create because of "+ exc);
		}
	}
	private void makeGUI() {
		// Set the applet to use flow layout.
		setLayout(new FlowLayout());
		// Make two buttons.
		jbtnAlpha = new JButton("Alpha");
		jbtnBeta = new JButton("Beta");

		// Add action listener for Alpha and Beta
		 jbtnAlpha.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent le) {
			jlab.setText("Alpha was pressed.");
			}
		});
		// Add action listener for Beta.
		 jbtnBeta.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent le) {
			jlab.setText("Beta was pressed.");
			}
		});
		// Add the buttons to the content pane.
		add(jbtnAlpha);
		add(jbtnBeta);

		// Create a text-based label.
		jlab = new JLabel("Press a button.");
		add(jlab);
	}
}
```

19. Write a swing applet program to demonstrate with two JButtons named India and Srilanka. When either buttons pressed, it should display respective label with its icon. Refer the image icons "India.gif" and "Srilanka.gif". Set the initial label is "press the button".

```
// A simple Swing-based applet

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/*
This HTML can be used to launch the applet:

<object code="IndiaSrilankaApplet" width=220 height=90>
</object>
*/

public class IndiaSrilankaApplet extends JApplet {
  JButton jbtnIndia;
  JButton jbtnSrilanka;

  JLabel jlab;

  // Initialize the applet.
  public void init() {
    try {
      SwingUtilities.invokeAndWait(new Runnable () {
        public void run() {
          makeGUI(); // initialize the GUI
        }
      });
    } catch(Exception exc) {
      System.out.println("Can't create because of "+ exc);
    }
  }

  // This applet does not need to override start(), stop(),
  // or destroy().

  // Setup and initialize the GUI.
  private void makeGUI() {
    // Set the applet to use flow layout.
    setLayout(new FlowLayout());

    // Make two buttons.
    jbtnIndia = new JButton("India");
    jbtnSrilanka = new JButton("Srilanka");

    // Add action listener for Alpha.
    jbtnIndia.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent le) {
        jlab.setText("India");
        jlab.setIcon(new ImageIcon("India.gif"));
      }
    });

    // Add action listener for Beta.
    jbtnSrilanka.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent le) {
        jlab.setText("Srilanka");
        jlab.setIcon(new ImageIcon("Srilanka.gif"));
        jlab.setIcon(new ImageIcon("italy.gif"));
      }
    });

    // Add the buttons to the content pane.
    add(jbtnIndia);
    add(jbtnSrilanka);

    // Create a text-based label.
    jlab = new JLabel("Press a button.");
```

```
    // Add the label to the content pane.
    add(jlab);
  }
}
```

20. List the different types of swing buttons. Write a program to create four types of buttons on JApplet. Use suitable events to show actions on the buttons and use JLabel to display the action invoked. OR
List four types of buttons in swings with their use. Write a program to create four different types of buttons on JApplet. Use suitable events to show actions on the buttons and use JLabel to display the action invoked.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ButtonDemo extends JApplet  implements ActionListener {
       JLabel jlab;
       public void init() {
              try {
                     SwingUtilities.invokeAndWait( new Runnable() {
                            public void run() {
                                   makeGUI();
                            }
                     } );
              } catch (Exception exc) {
              System.out.println("Can't create because of " + exc);
              }
}
private void makeGUI() {
              // Change to flow layout.
              setLayout(new FlowLayout());

              // Add buttons to content pane.
              JButton jb = new JButton("Button");
              jb.addActionListener(this);
              add(jb);

              // Create and add a toggle button.
              JToggleButton cb =  new JToggleButton("ToggleButton");
              cb.addActionListener(this);
              add(cb);

              // Create and add a check box.
              JCheckBox cb =  new JCheckBox ("CheckBox ");
              cb.addActionListener(this);
              add(cb);

              // Create and add a radio button.
              JRadioButton rb =  new JRadioButton ("JRadioButton ");
              rb.addActionListener(this);
              add(rb);

              // Create and add the label to content pane.
              jlab = new JLabel("Press/Select a Button");
              add(jlab);
       }

       // Handle button events.
       public void actionPerformed(ActionEvent ae) {
                     jlab.setText("You selected " + ae.getActionCommand());
       }
}
```

21. Explain the following, with an example for each: i) JTextFieId class ii) JButton class iii) JComboBox class.

JTextField:
The class **JTextField** is a component which allows the editing of a single line of text.

- It is the simplest and most widely used Swing text component.
- Constructors
  - JTextField(int *cols*) : Constructs a new empty TextField with the specified number of columns.

- JTextField(String *str*, int *cols*) : Constructs a new TextField initialized with the specified text and columns.
- JTextField(String *str*) : Constructs a new TextField initialized with the specified str

- To obtain the text currently in the text field, call getText( )
- JTextField generates events in response to user interaction. For example, an ActionEvent is fired when the user presses ENTER.

JTextField example creates a **JTextField** and adds it to the content pane. When the user presses ENTER, an action event is generated. This is handled by displaying the text in the status window.

```
// Demonstrate JTextField.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
</applet>
*/
public class JTextFieldDemo extends JApplet {
	JTextField jtf;
	public void init() {
		try {
			SwingUtilities.invokeAndWait( new Runnable() {
				public void run() {makeGUI();}
			}
			);
		} catch (Exception exc) {
			System.out.println("Can't create because of " + exc);
		}
	}
	private void makeGUI() {
		// Change to flow layout.
		setLayout(new FlowLayout());
		// Add text field to content pane.
		jtf = new JTextField(15);
		add(jtf);
		jtf.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent ae) {
				// Show text when user presses ENTER.
				showStatus(jtf.getText());
			}
		});
	}
}
```

22. Explain the JScrollpane and JComboBox with a program.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JComboBoxDemo extends JApplet {
	JLabel jlab;
	ImageIcon france, germany, italy, japan;
	JComboBox jcb;
	 String flags[] = { "India", "Bhutan", "Nepal", "Thailand", "Malaysia", "Indonesia",
	"Philippines", "Myanmar", "Kuwait", "UAE", "USA", "UK", "Brazil", "Peru", "Egypt", "Germany",
	"France", "Germany", "Italy", "Japan" };

	public void init() {
		try {
		SwingUtilities.invokeAndWait( new Runnable() {
				public void run() {
					makeGUI();
				}
			} );
		} catch (Exception exc) {
		System.out.println("Can't create because of " + exc);
	}
}
private void makeGUI() {
		// Change to flow layout.
		setLayout(new FlowLayout());
```

```
            // Instantiate a combo box
             jcb = new JComboBox(flags);

             // Handle selections.
            jcb.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                        String s = (String) jcb.getSelectedItem();
                        jlab.setIcon(new ImageIcon(s + ".gif"));
            }
            });

             // Add the combobox to a scroll pane.
            JScrollPane jsp = new JScrollPane(jcb);

            // Add the scroll pane to the content pane.
            add(jsp);

            // Create a label and add it to the content pane.
            jlab = new JLabel(new ImageIcon("India.gif"));
            add(jlab);
        }
}
```

23. Explain with syntax the following: i) JLabel ii) JTextField iii) JButton iv) JCheckBox.

i) **JLable**: **JLabel** is Swing's easiest-to-use component. It creates a label. **JLabel** can be used to display text and/or an icon. It is a passive component in that it does not respond to user input. **JLabel** defines several constructors. Here are three of them:

```
JLabel(Icon icon)
JLabel(String str)
JLabel(String str, Icon icon, int align)
```

Here, *str* and *icon* are the text and icon used for the label. The *align* argument specifies the horizontal alignment of the text and/or icon within the dimensions of the label.

ii) **JTextField: JTextField** is the simplest Swing text component. **JTextField** allows you to edit one line of text. Three of **JTextField**'s constructors are shown here:

```
JTextField(int cols)
JTextField(String str, int cols)
JTextField(String str)
```

Here, *str* is the string to be initially presented, and *cols* is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string. **JTextField** generates events in response to user interaction. For example, an **ActionEvent** is fired when the user presses ENTER. To obtain the text currently in the text field, call **getText()**.

iii) **JButton**: The **JButton** class provides the functionality of a push button. **JButton** allows an icon, a string, or both to be associated with the push button. Three of its constructors are shown here:

```
JButton(Icon icon)
JButton(String str)
JButton(String str, Icon icon)
```

Here, *str* and *icon* are the string and icon used for the button. When the button is pressed, an **ActionEvent** is generated. Using the **ActionEvent** object passed to the **actionPerformed( )** method of the registered **ActionListener**, you can obtain the *action command* string associated with the button. You can obtain the action command by calling **getActionCommand( )** on the event object.

iv) **JCheckBox**: **Check Boxes** The **JCheckBox** class provides the functionality of a check box. Its immediate superclass is **JToggleButton**, which provides support for two-state buttons, as just described. **JCheckBox** defines several constructors. For example JCheckBox(String *str*)

It creates a check box that has the text specified by *str* as a label. Other constructors let you specify the initial selection state of the button and specify an icon. When the user selects or deselects a check box, an **ItemEvent** is generated. You can obtain a reference to the **JCheckBox** that generated the event by calling **getItem()** on the **ItemEvent** passed to the **itemStateChanged()** method defined by **ItemListener**. The easiest way to determine the selected state of a check box is to call **isSelected( )** on the **JCheckBox** instance.

24. Explain the JScrollPane with an example.

```
import java.awt.*;
import javax.swing.*;
public class JScrollPaneDemo extends JApplet {
	public void init() {
		try {
		SwingUtilities.invokeAndWait( new Runnable() {
				public void run() {
					makeGUI();
				}
			} );
		} catch (Exception exc) {
		System.out.println("Can't create because of " + exc);
		}
}
	private void makeGUI() {
	`	// Add 400 buttons to a panel.
			JPanel jp = new JPanel();
			jp.setLayout(new GridLayout(20, 20));
			int b = 0;
			for(int i = 0; i < 20; i++) {
		for(int j = 0; j < 20; j++) {
				jp.add(new JButton("Button " + b));
				++b;
		}
			}
			// Create the scroll pane.
			JScrollPane jsp = new JScrollPane(jp);

			// Add the scroll pane to the content pane. Because the default border layout
			// is  used, the scroll pane will be added to the center.
		add(jsp, BorderLayout.CENTER);
	}
}
```

25. Explain the JComboBox with an example.

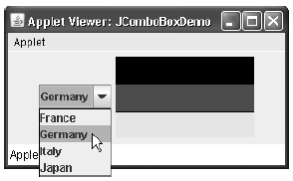
JComboBox:

- Swing provides a *combo box* (a combination of a text field and a drop-down list) through the **JComboBox** class.
- A combo box normally displays one entry, but it will also display a drop-down list that allows a user to select a different entry.
- The following **JComboBox** constructor initializes the combo box using *items* array
  JComboBox(Object[ ] *items*)
- **JComboBox** generates an action event when the user selects an item from the list. **JComboBox** also generates an item event when the state of selection changes, which occurs when an item is selected or deselected.
- One way to obtain the item selected in the list is to call **getSelectedItem( )** on the combobox.

The following example demonstrates the combo box. The combo box contains entries for “France,” “Germany,” “Italy,” and “Japan.” When a country is selected, an icon-based label is updated to display the flag for that country.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JComboBoxDemo extends JApplet {
	JLabel jlab;
	ImageIcon france, germany, italy, japan;
	JComboBox jcb;
	String flags[] = { "France", "Germany", "Italy", "Japan" };

	public void init() {
		try {
		SwingUtilities.invokeAndWait( new Runnable() {
				public void run() {
					makeGUI();
				}
```

```
                    } );
                } catch (Exception exc) {
                System.out.println("Can't create because of " + exc);
        }
}
private void makeGUI() {
                // Change to flow layout.
                setLayout(new FlowLayout());

                // Instantiate a combo box and add it to the content pane.
                 jcb = new JComboBox(flags);
                add(jcb);

                 // Handle selections.
                jcb.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                                String s = (String) jcb.getSelectedItem();
                                jlab.setIcon(new ImageIcon(s + ".gif"));
                }
                });

                // Create a label and add it to the content pane.
                jlab = new JLabel(new ImageIcon("france.gif"));
                add(jlab);
        }
}
```



26. Write the steps to create JTable. Write a program to create a table with column headings “fname, lname, age” and insert at least 5 records in the table and display. OR
Write the steps to create JTabIe. Write a program to create a table with the column headings "Fname, Lname, Age' and insert at least five records in the table and display. OR

Write the steps to create JTable.

JTable is a component that displays rows and columns of data. At the top of each column is a heading. In addition to describing the data in a column, the heading also provides the mechanism by which the user can change the size of a column or change the location of a column within the table.

The steps required to set up a simple **JTable** that can be used to display data are described below:

1. Create an instance of **JTable**.
   A commonly used constructor for JTable: **JTable(Object *data*[ ][ ], Object *colHeads*[ ])**
   - *data* is a two-dimensional array of the information to be presented, and
   - *colHeads* is a one-dimensional array with the column headings.

2. Create a **JScrollPane** object, specifying the table as the object to scroll.
3. Add the table to the scroll pane.
4. Add the scroll pane to the content pane.

Program to create a table with the column headings "Fname, Lname, Age' and insert at least five records in the table and display.

```
import java.awt.*;
import javax.swing.*;
public class JTableDemo extends JApplet {
        public void init() {
                try {
```

```
            SwingUtilities.invokeAndWait( new Runnable() {
                        public void run() {
                              makeGUI();
                        }
                  } );
            } catch (Exception exc) {
            System.out.println("Can't create because of " + exc);
            }
      }
      private void makeGUI() {
            // Initialize column headings.
            String[] colHeads = { "Fname", "Lname", "Age" };

            // Initialize data.
            Object[][] data = {
            { "Zahid", "Ansari", "50" },
            { "Abdullah", "Ansari", "19" },
            { "Musab", "Ansari", "17" },
            { "Muaaz", "Ansari", "16" },
            { "Abdurrahman", "Ansari", "10" }
            };
            // Create the table.
            JTable table = new JTable(data, colHeads);

            // Add the table to a scroll pane.
            JScrollPane jsp = new JScrollPane(table);

            // Add the scroll pane to the content pane.
            add(jsp);
      }
}
```

27. Write a program to create a table with the column headings Name, USN, Age, Address and insert at least five records in the table and display.

(Refer Question 26 above and change the colHeads to Name, USN, Age, Address and add 5 data items)

```
String[] colHeads = { "Name", "USN", "Age", "Address" };
Object[][] data = {
            { "Ahmed", "123456", "20", "Deralakatte" },
            .
            .
            .
            .

            };
```

28. Explain the different types of panes of swing containers.