MODULE 5

APPLETS

Applets in Java are small programs that run within a web browser. They are embedded in HTML pages using the <applet> or <object> tag and are executed by a Java Virtual Machine (JVM) within the browser.

Applets are client-side programs that run on the user's computer rather than the web server.

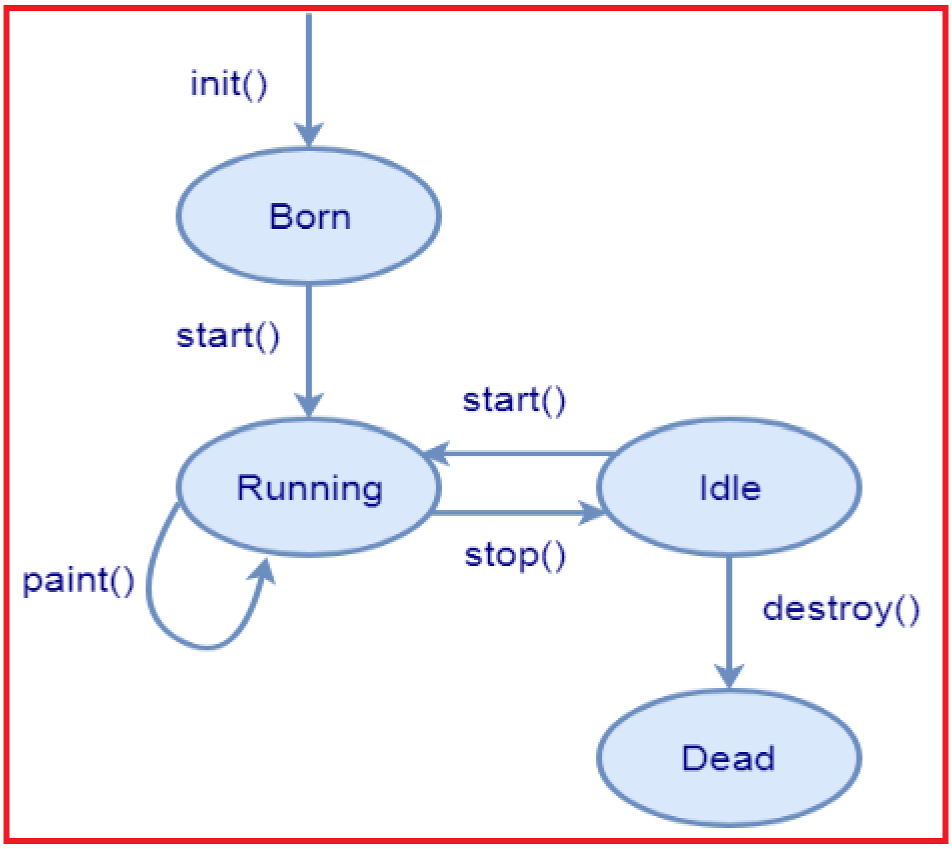
Applets typically create interactive web content, such as games, animations, and data visualizations. However, due to security concerns and the rise of other web technologies, applets are now rarely used.

Key Points:

- Applet Basics: Every applet is a child/subclass of the java.applet.Applet class.
- Not Standalone: Applets don't run on their own like regular Java programs. They need a web browser or a special tool called the applet viewer (which comes with Java).
- No main() Method: Applets don't start with main() method.
- Display Output: Applets don't use System.out.prinln() for displaying the output, instead they use graphics methods like drawString() from the AWT (Abstract Window ToolKit).

Java Applet Life Cycle

The below diagram demonstrates the life cycle of Java Applet:



- When an applet begins, the following methods are called, in this sequence:

  init( )

  start( )

  paint( )
- When an applet is terminated, the following sequence of method calls takes place:

  stop( )

  destroy( )

Let’s look more closely at these methods.

1. init( ): The init( ) method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

2. start( ): The start( ) method is called after init( ). It is also called to restart an applet after it has been stopped.

Note: init( ) is called once i.e. when the first time an applet is loaded whereas start( ) is called each time an applet’s HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( )

3. paint( ): The paint( ) method is called each time an AWT-based applet’s output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.

paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called.

The **paint( )** method has one parameter of type **Graphics.** This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

paint() is the only method among all the methods mention above (which is parameterized).

Example:

```
public void paint(Graphics g)
{
// Drawing a string on the applet window
// g is an object reference of class Graphic.
g.drawString(“Hello, Applet!”, 50, 50);
}
```

4. stop( ): The stop( ) method is called when a web browser leaves the HTML document containing the applet, when it goes to another page.

For example: When stop( ) is called, the applet is probably running. You should use stop( ) to suspend threads that don't need to run when the applet is not visible. You can restart them when start( ) is called if the user returns to the page.

5. destroy( ): The destroy( ) method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The stop( ) method is always called before destroy( ).

Key Packages for Java Applets

- java.applet.Applet: Base class for applets.
- java.awt.Graphics: Used for drawing on the applet screen.
- java.awt: Provides GUI components and event-handling mechanisms.

Creating Hello World Applet

Let's begin with the HelloWorld applet :

```
import java.applet.Applet;
import java.awt.Graphics;
// HelloWorld class extends Applet
public class HelloWorld extends Applet {
    // Overriding paint() method
  @Override public void paint(Graphics g)
  {
    g.drawString("Hello World", 20, 20);
  }
}
```

https://dotnettutorials.net/lesson/applet-in-java/

Graphics in Applet:

All graphics are drawn relative to a window. This can be the main window of an applet, a child window of an applet, or a stand-alone application window. The origin of each window is at the top-left Coordinates 0,0. Coordinates are specified in pixels. All output to a window takes place through a graphics context. A graphics context is encapsulated by the Graphics class and is obtained in two ways:

- It is passed to an applet when one of its various methods, such as paint() or update(), is called.
- It is returned by the getGraphics( ) method of Component.

The Graphics class defines several drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When a graphics object is drawn that exceeds the dimensions of the window, the output is automatically clipped.

Methods of Graphics class

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draw a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill the rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height)**: is used to draw an oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill an oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw a line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used to draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

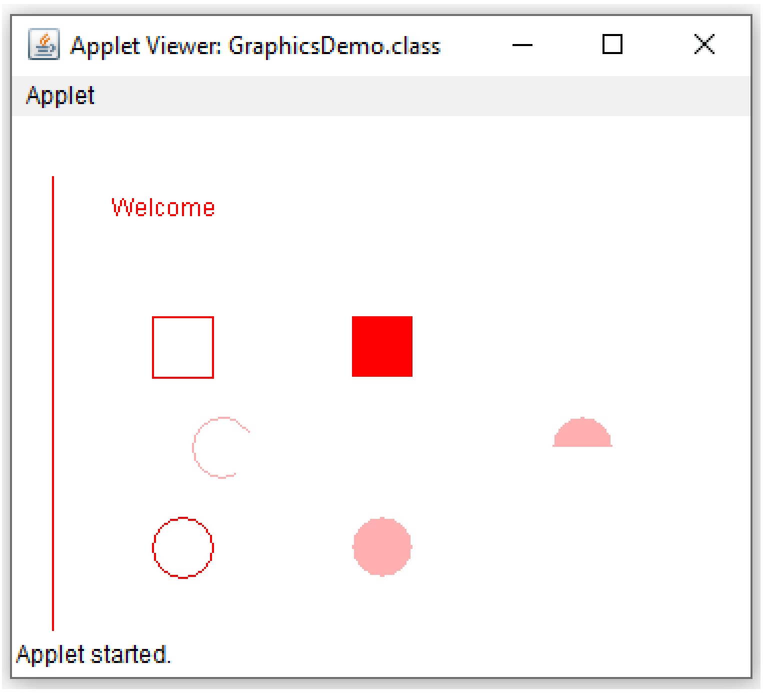
Examples to Understand Graphics in Applet:

```
import java.applet.Applet;
import java.awt.*;
/*
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
*/
public class GraphicsDemo extends Applet
{
   public void paint (Graphics g)
   {
      g.setColor (Color.red);
```

```
        g.drawString ("Welcome", 50, 50);
        g.drawLine (20, 30, 20, 300);
        g.drawRect (70, 100, 30, 30);
        g.fillRect (170, 100, 30, 30);
        g.drawOval (70, 200, 30, 30);

        g.setColor (Color.pink);
        g.fillOval (170, 200, 30, 30);
        g.drawArc (90, 150, 30, 30, 30, 270);
        g.fillArc (270, 150, 30, 30, 0, 180);
    }
}
```

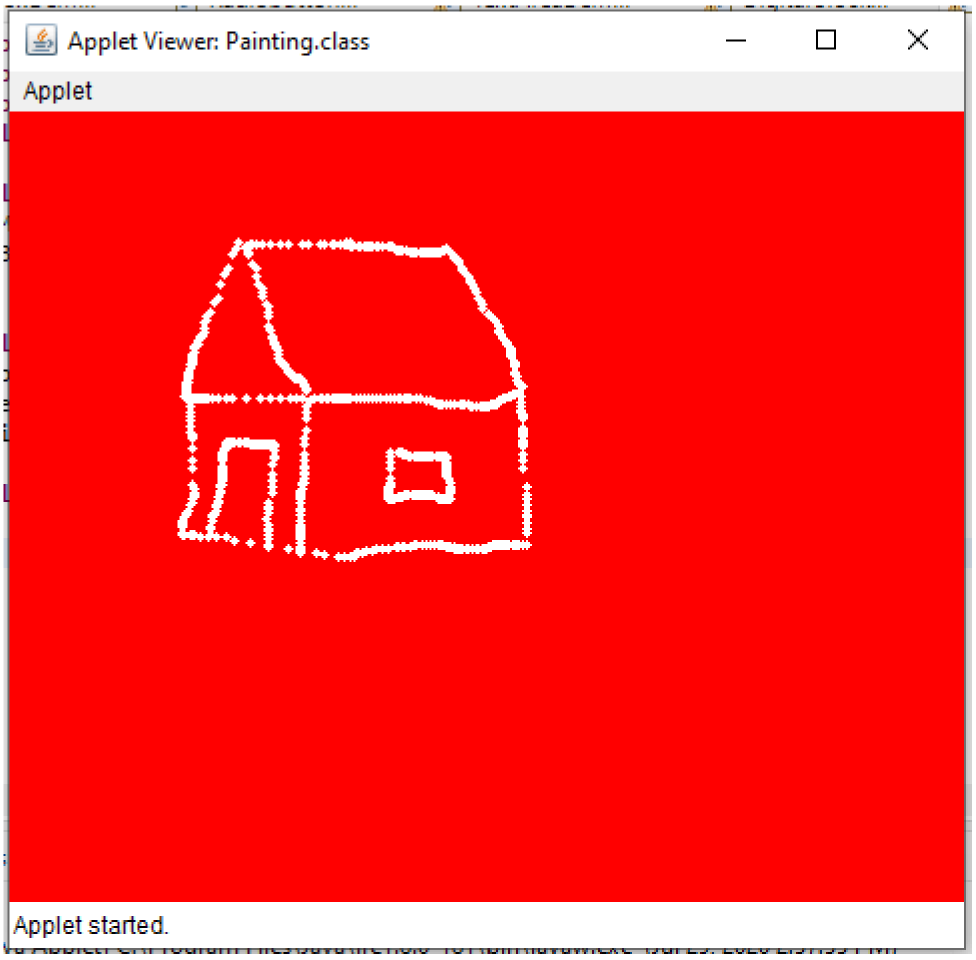
OUTPUT:



Example: Painting in Applet

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Painting extends Applet implements MouseMotionListener
{
    public void init ()
    {
        addMouseMotionListener (this);
        setBackground (Color.red);
    }
    public void mouseDragged (MouseEvent me)
    {
        Graphics g = getGraphics ();
        g.setColor (Color.white);
        g.fillOval (me.getX (), me.getY (), 5, 5);
    }
    public void mouseMoved (MouseEvent me)
    {
    }
}
```

OUTPUT:



Applet started.

Example to Display Digital Clock in an Applet

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;
/*
<applet code="DigitalClock" width="300" height="300">
```

```
</applet>
 */
public class DigitalClock extends Applet implements Runnable
{
    Thread t = null;
    int hours = 0, minutes = 0, seconds = 0;
    String timeString = "";

    public void init ()
    {
        setBackground (Color.green);
    }

    public void start ()
    {
        t = new Thread (this);
        t.start ();
    }
        public void run ()
    {
        try
        {
            while (true)
            {
                Calendar cal = Calendar.getInstance ();
                hours = cal.get (Calendar.HOUR_OF_DAY);
                if (hours > 12)
                  hours -= 12;
                minutes = cal.get (Calendar.MINUTE);
                seconds = cal.get (Calendar.SECOND);
                SimpleDateFormat formatter = new SimpleDateFormat ("hh:mm:ss");
                Date date = cal.getTime ();
                timeString = formatter.format (date);
                repaint ();
                t.sleep (1000); // interval given in milliseconds
            }
        }
        catch (Exception e)
        {
        }
    }
    public void paint (Graphics g)
    {
        g.setColor (Color.blue);
        g.drawString (timeString, 50, 50);
    }
}
```

HTML applet Tag

The **applet** tag in HTML was used to *embed Java applets into any HTML document*. The **<applet>** tag was deprecated in HTML 4.01, and its support has been completely discontinued starting from HTML 5. Alternatives available in HTML 5 are the **<embed>** and the **<object>** tags. Some browsers still support the <applet> tag with the help of some additional plug-ins/installations to work.

EX:

```
<!DOCTYPE html>
<html>
<applet code="HelloWorld">
  <param name="message" value="HelloWorld">
</applet>
</html>
```

Syntax:

<param name=parameter_name value=parameter_value>

The name assigned to the name attribute of the param tag is used by the applet code as a variable to access the parameter value specified in the **value** attribute. In this way, the applet is able to interact with the HTML page where it is embedded, and can work on values provided to it by the page during run-time.

Attribute Values	**Description**
align	Specifies the alignment of an applet.
alt	Specifies an alternate text for an applet.
archive	Specifies the location of an archive file.

Attribute Values	Description
border	Specifies the border around the applet panel.
codebase	Specifies a relative base URL for applets specified in the code attribute.
height	Specifies the height of an applet.
hspace	Defines the horizontal spacing around an applet.
mayscript	Indicates whether the Java applet is allowed to access the scripting objects of the web page.
name	Defines the name for an applet (to use in scripts).
vspace	Defines the vertical spacing around an applet.
width	Specifies the width of an applet.

Adding controls to a Java applets

Adding controls to a Java applet involves using the Abstract Window Toolkit (AWT) or Swing libraries. Here's a guide on how to add common controls like buttons, labels, and text fields to an applet.

- **Import necessary classes**: Start by importing the required classes from the java.awt or javax.swing packages.

  ```
   import java.applet.Applet;
  import java.awt.*;
   import java.awt.event.*; // Required for handling events
   import javax.swing.*; // For Swing components
  ```

- **Initialize controls**: Inside the init() method of your applet, create instances of the controls you want to add.

```
Button myButton = new Button("Click Me");
Label myLabel = new Label("Enter Text:");
TextField myTextField = new TextField(20); // 20 columns wide
```

- **Add controls to the applet**: Use the add() method to add the controls to the applet's layout.

```
add(myButton);
 add(myLabel);
 add(myTextField);
```

- **Handle events (if needed):** If you want the controls to perform actions, you'll need to add event listeners.

```
    myButton.addActionListener(new ActionListener
    public void actionPerformed(ActionEvent e)
{
 // Action to perform when button is clicked
myLabel.setText("Button Clicked!");
      }
  });
```

Example:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class ControlApplet extends Applet implements ActionListener {
    private Label myLabel;
    private TextField myTextField;
    public void init() {
        Button myButton = new Button("Click Me");
        myLabel = new Label("Enter Text:");
        myTextField = new TextField(20);
        myButton.addActionListener(this);
        add(myButton);
        add(myLabel);
        add(myTextField);
    }
    public void actionPerformed(ActionEvent e) {
         myLabel.setText("Button Clicked! Text: " + myTextField.getText());
    }
    public void paint(Graphics g) {
        // Optional: Add custom drawing
    }
}
```

Swings:

Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Java Swing offers much-improved functionality over AWT, new components, expanded component features, and excellent event handling with drag-and-drop support.

Its Classes became very popular with programmers creating GUI's for commercial applications.

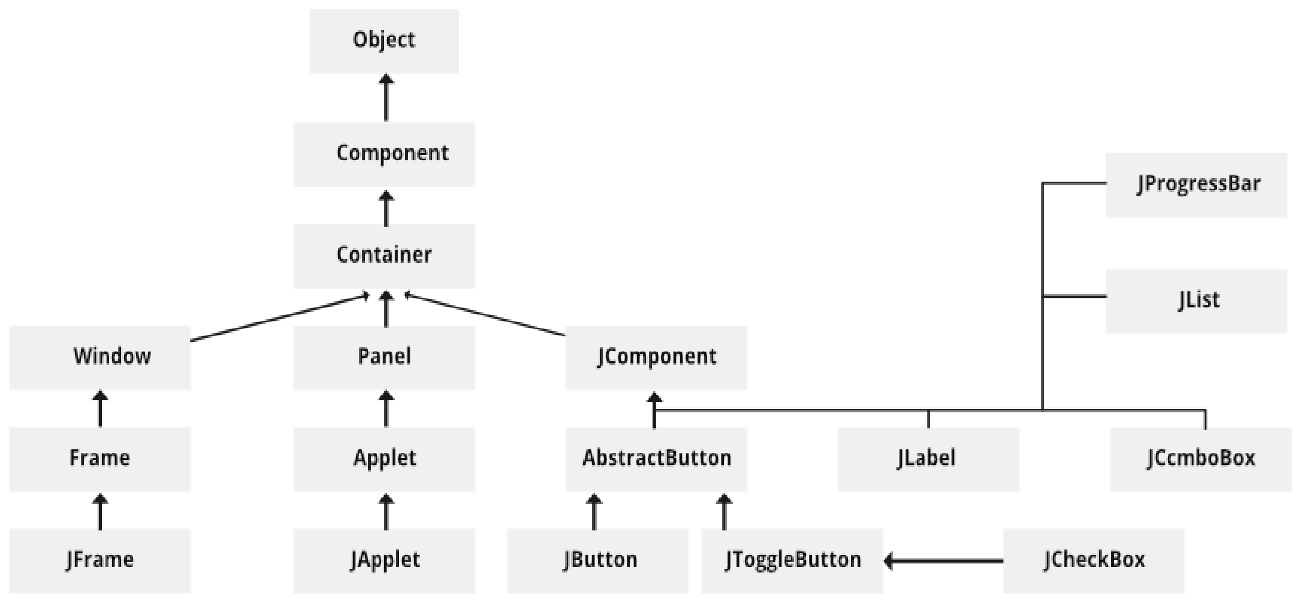
- Swing is a Set of API (API- Set of Classes and Interfaces)
- Swing is Provided to Design Graphical User Interfaces
- Swing is an Extension library to the AWT (Abstract Window Toolkit)
- Includes New and improved Components that have been enhancing the looks and Functionality of GUIs'
- Swing can be used to build (Develop) The Standalone swing GUI Apps as Servlets and Applets
- It Employs model/view design architecture.
- Swing is more portable and more flexible than AWT, the Swing is built on top of the AWT.
- Swing is Entirely written in Java.
- Java Swing Components are Platform-independent, and The Swing Components are lightweight.
- Swing Supports a Pluggable look and feel and Swing provides more powerful components.
- such as tables, lists, Scrollpanes, Colourchooser, tabbed pane, etc.
- Further Swing Follows MVC.

Difference between Java Swing and Java AWT

Java AWT	**Java Swing**
Java AWT is an API to develop GUI applications in Java.	Swing is a part of Java Foundation Classes and is used to create various applications.
Components of AWT are heavy weighted.	The components of Java Swing are lightweight.
Components are platform dependent.	Components are platform independent.
Execution Time is more than Swing.	Execution Time is less than AWT.

Java AWT	**Java Swing**
AWT components require java.awt package.	Swing components requires javax.swing package.

Swing Classes Hierarchy



In general, a visual component is a composite of **three distinct aspects:**

1. The way that the component looks when rendered on the screen.
2. The way such that the component reacts to the user.
3. The state information associated with the component.

- Over the years, one component architecture has proven itself to be exceptionally effective: - **Model-View-Controller** or **MVC** for short.
- In MVC terminology, the **model** corresponds to the state information associated with the Component.
- The **view** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
- The **controller** determines how the component reacts to the user.

Swing Components in Java

Swing components are the basic building blocks of an application. We know that Swing is a GUI widget toolkit for Java. Every application has some basic interactive interface for the user. For example, a button, check-box, radio-button, text-field, etc. These together form the components in Swing.

1) ImageIcon

The ImageIcon component creates an icon sized-image from an image residing at the source URL.

Example:

```
ImageIcon homeIcon = new ImageIcon("src/images/home.jpg");
```

This returns an icon of a home button. The string parameter is the path at which the source image is present.

2. JButton

The JButton class is used to create a push-button on the UI. The button can contain some display text or image. It generates an event when clicked and double-clicked. A JButton can be implemented in the application by calling one of its constructors.

Example:

JButton okBtn = new JButton("Ok");

This constructor returns a button with text Ok on it.

JButton homeBtn = new JButton(homeIcon);

It returns a button with a homeIcon on it.

JButton btn2 = new JButton(homeIcon, "Home");

It returns a button with the home icon and text Home.

3. JLabel

JLabel class is used to render a read-only text label or images on the UI. It does not generate any event.

Example:

```
JLabel textLbl = new JLabel("This is a text label.");
```

This constructor returns a label with text.

```
JLabel imgLabel = new JLabel(homeIcon);
```

It returns a label with a home icon.

5. JTextArea

The **JTextArea** class renders a multi-line text box. Similar to the JTextField, a user can input non-formatted text in the field. The constructor for JTextArea also expects two integer parameters which define the height and width of the text-area in columns. It does not restrict the number of characters that the user can input in the text-area.

Example:

```
JTextArea txtArea = new JTextArea("This text is default text for text area.", 5, 20);
```

The above code renders a multi-line text-area of height 5 rows and width 20 columns, with default text initialized in the text-area.

6. JPassword Field

JPasswordField is a subclass of JTextField class. It renders a text-box that masks the **user input** text with bullet points. This is used for inserting passwords into the application.

Example:

```
JPasswordField pwdField = new JPasswordField(15);
```

```
var pwdValue = pwdField.getPassword();
```

It returns a password field of 15 column width. The getPassword method gets the value entered by the user.

7. JCheckBox

JCheckBox renders a check-box with a label. The check-box has two states – on/off. When selected, the state is on and a small tick is displayed in the box.

Example:

```
CheckBox chkBox = new JCheckBox("Show Help", true);
```

It returns a checkbox with the label Show Help. Notice the second parameter in the constructor. It is a boolean value that indicates the default state of the check-box. True means the check-box is defaulted to on state.

8. JRadioButton

JRadioButton is used to render a group of **radio** buttons in the UI. A user can select one choice from the group.

Example:

```
ButtonGroup radioGroup = new ButtonGroup();
JRadioButton rb1 = new JRadioButton("Easy", true);
JRadioButton rb2 = new JRadioButton("Medium");
JRadioButton rb3 = new JRadioButton("Hard");
 radioGroup.add(rb1);
radioGroup.add(rb2);
radioGroup.add(rb3);
```

The above code creates a button group and three radio button elements. All three elements are then added to the group. This ensures that only one option out of the available options in the group can be selected at a time. The default selected option is set to Easy.

9. JList

JList component renders a scrollable list of elements. A user can select a value or multiple values from the list. This select behavior is defined in the code by the developer.

```
DefaultListItem cityList = new DefaultListItem();
 cityList.addElement("Mumbai"):
cityList.addElement("London"):
cityList.addElement("New York"):
cityList.addElement("Sydney"):
 cityList.addElement("Tokyo"):
JList cities = new JList(cityList);
cities.setSelectionModel(ListSelectionModel.SINGLE_SELECTION);
```

The above code renders a list of cities with 5 items in the list. The selection restriction is set to SINGLE_SELECTION. If multiple selections is to be allowed, set the behavior to MULTIPLE_INTERVAL_SELECTION.

10. JComboBox

JComboBox class is used to render a dropdown of the list of options.

Example:

```
String[] cityStrings = { "Mumbai", "London", "New York", "Sydney", "Tokyo" };
JComboBox cities = new JComboBox(cityList);
cities.setSelectedIndex(3);
```

The default selected option can be specified through the setSelectedIndex method. The above code sets Sydney as the default selected option.

11. JFileChooser

JFileChooser class renders a file selection utility. This component lets a user select a file from the local system.

Example:

```
JFileChooser fileChooser = new JFileChooser();
JButton fileDialogBtn = new JButton("Select File");
fileDialogBtn.AddEventListner(new ActionListner(){
fileChooser.showOpenDialog();
})
var selectedFile = fileChooser.getSelectedFile();
```

The above code creates a file chooser dialog and attaches it to the button. The button click would open the file chooser dialog. The selected file is returned through the getSelectedFile method.

12. JTabbedPane

JTabbedPane is another very useful component that lets the user switch between tabs in an application. This is a highly useful utility as it lets the user browse more content without navigating to different pages.

Example:

```
JTabbedPane tabbedPane = new JTabbedPane();

tabbedPane.addTab("Tab 1", new JPanel());

tabbedPane.addTab("Tab 2", new JPanel());
```

The above code creates a two tabbed panel with headings Tab 1 and Tab 2.

13. JSlider

JSlider component displays a slider which the user can drag to change its value. The constructor takes three arguments – minimum value, maximum value, and initial value.

Example:

```
JSlider volumeSlider = new JSlider(0, 100, 50);

var volumeLevel = volumeSlider.getValue();
```

The above code creates a slider from 0 to 100 with an initial value set to 50. The value selected by the user is returned by the getValue method.

Example of Java Swing Programs

Example 1: Develop a program using label (swing) to display the message "ACU WEB Site Click":

```
// Java program using label (swing)
// to display the message "ACUWEB Site Click"
import java.io.*;
import javax.swing.*;
// Main class
class ACU{
    // Main driver method
    public static void main(String[] args)
    {
        // Creating instance of JFrame
        JFrame frame = new JFrame();
        // Creating instance of JButton
        JButton button = new JButton(" ACU WebSite Click");
        // x axis, y axis, width, height
        button.setBounds(150, 200, 220, 50);
        // adding button in JFrame
        frame.add(button);
        // 400 width and 500 height
        frame.setSize(500, 600);
        // using no layout managers
        frame.setLayout(null);
        // making the frame visible
        frame.setVisible(true);
    }
}
```

Example 2: Write a program to create three buttons with caption OK, SUBMIT, CANCEL.

```
// Java program to create three buttons
// with caption OK, SUBMIT, CANCEL
import java.awt.*;
class button {
    button()
    {
        Frame f = new Frame();
        // Button 1 created
        // OK button
        Button b1 = new Button("OK");
        b1.setBounds(100, 50, 50, 50);
        f.add(b1);
        // Button 2 created
        // Submit button
        Button b2 = new Button("SUBMIT");
        b2.setBounds(100, 101, 50, 50);
        f.add(b2);
        // Button 3 created
        // Cancel button
        Button b3 = new Button("CANCEL");
        b3.setBounds(100, 150, 80, 50);
        f.add(b3);
        f.setSize(500, 500);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String a[]) { new button(); }
}
```

****