# CORE JAVA

By : Kalpesh Chauhan

## JAVA APPLET BASICS

## LET'S UNDERSTAND FIRST HOW MANY PACKAGE DOES GUI SUPPORT:

1. AWT(Abstract Window Toolkit)
2. Swing

## THROWBACK OF MAKING GUI APPLICATION:

Java was launched on 23-Jan-1996(JDK 1.0) and at that time it only supported CUI(Character User Interface) application. But in 1996 VB(Visual Basic) of Microsoft was preferred for GUI programming. So the Java developers in hurry(i.e within 7 days) have given the support for GUI from Operating System(OS). Now, the components like button,etc. were platform-dependent(i.e in each platform there will be different size, shape button). But they did the intersection of such components from all platforms and gave a small library which contains these intersections and it is available in AWT(Abstract Window Toolkit) technology but it doesn’t have advanced features like dialogue box, etc.

Now to run Applet, java needs a browser and at that time only “Internet Explorer” was there of Microsoft but Microsoft believes in monopoly. So “SUN Micro-System”(the company which developed Java) contracted with other company known as “Netscape”(which developed Java Script) and now the “Netscape” company is also known as “Mozilla Firefox” which we all know is a browser. Now, these two companies have developed a technology called “SWING” and the benefit is that the SWING components are produced by Java itself. Therefore now it is platform-independent as well as some additional features have also been added which were not in AWT technology. So we can say that SWING is much more advanced as compared to AWT technology.

## WHAT IS APPLET?

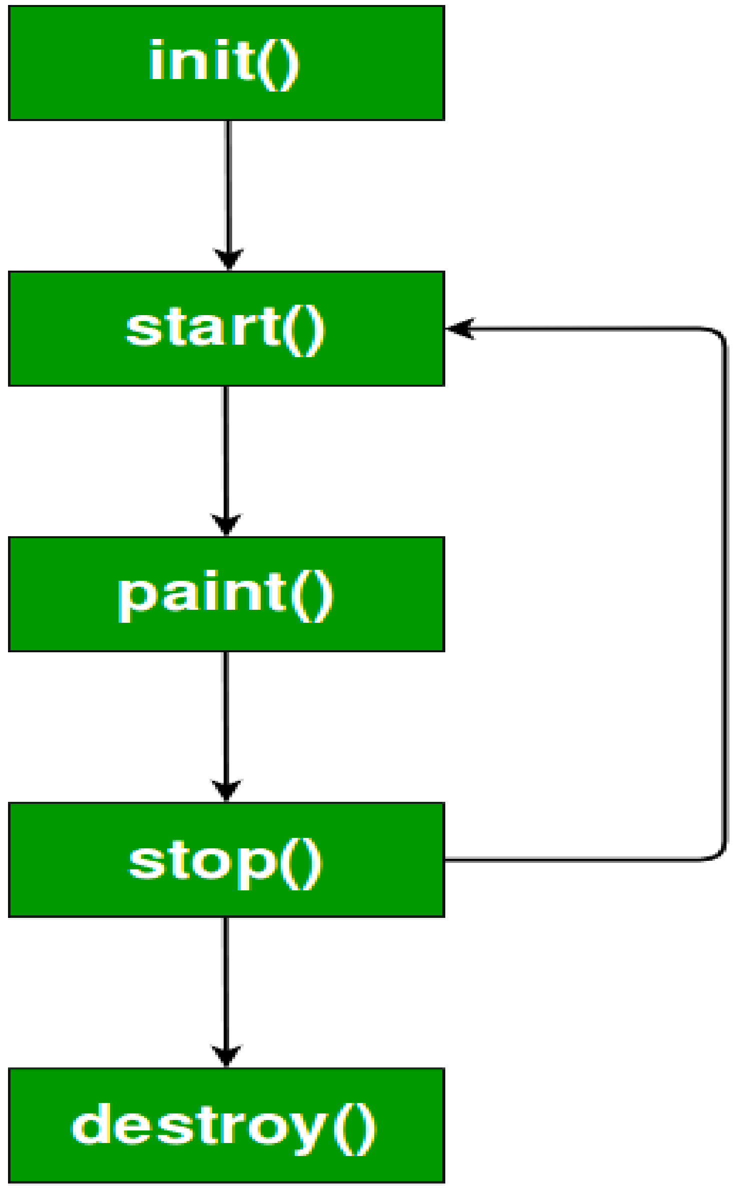
An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

Applets are used to make the web site more dynamic and entertaining.

## IMPORTANT POINTS :

1. All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

## LIFE CYCLE OF AN APPLET :



It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:

1. init( )
2. start( )
3. paint( )

When an applet is terminated, the following sequence of method calls takes place:

1. stop( )
2. 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 that **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( )**.

**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. Note: This is the only method among all the method mention above, which is parametrised. It's prototype is

public void paint(Graphics g)
where g is an object reference of class Graphic.

## NOW THE **QUESTION ARISES:**

**Question.** In the prototype of paint() method, we have created an object reference without creating its object. But how is it possible to create object reference without creating its object?

**Answer.** Whenever we pass object reference in arguments then the object will be provided by its caller itself. In this case the caller of paint() method is browser, so it will provide an object. The same thing happens when we create a very basic program in normal Java programs. For Example:

**<u>Public static void main(String args[])</u>**

Here we have created an object reference without creating its object but it still runs because it's caller,i.e JVM will provide it with an object.

**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.

**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( )**.

```
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);
    }

}
```

## RUNNING THE HELLOWORLD APPLET :

After you enter the source code for HelloWorld.java, compile in the same way that you have been compiling java programs(using *javac* command). However, running HelloWorld with the *java* command will generate an error because it is not an application.

here are **two** standard ways in which you can run an applet :

1. Executing the applet within a Java-compatible web browser.
2. Using an applet viewer, such as the standard tool, applet-viewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

## USING JAVA ENABLED WEB BROWSER

To execute an applet in a web browser we have to write a short HTML text file that contains a tag that loads the applet. We can use APPLET or OBJECT tag for this purpose. Using APPLET, here is the HTML file that executes HelloWorld :

**<applet code="HelloWorld" height="500" width="500"></applet>**

Chrome and Firefox no longer supports NPAPI (technology required for Java applets).

## USING APPLETVIEWER :

This is the easiest way to run an applet. To execute HelloWorld with an applet viewer, you may also execute the HTML file shown earlier. For example, if the preceding HTML file is saved with

## APPLETVIEWER WITH JAVA SOURCE FILE

If you include a comment at the head of your Java source code file that contains the APPLET tag then your code is documented with a prototype of the necessary HTML statements, and you can run your compiled applet merely by starting the applet viewer with your Java source code file. If you use this method, the HelloWorld source file looks like this :

```
// A Hello World Applet
// Save file as HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;
/*     <applet code=“HelloWorld” height=“500” width=“500”></applet> */

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

With this approach, first compile HelloWorld.java file and then simply run below command to run applet :

```
Appletviwer helloWorld
```

## AWT COMPONENTS

A component is an object with a graphical representation that can be displayed on the screen and that can interact with the user.

## LIST OF COMPONENTS

1. Button
2. Checkbox
3. Radiobutton
4. List
5. Choice
6. Scrollbar
7. Label
8. TextField
9. TextArea
10. image

## SWING

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls.

## SWING FEATURES

**Light Weight** − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.

**Rich Controls** − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.

**Highly Customizable** − Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.

**Pluggable look-and-feel** − SWING based GUI Application look and feel can be changed at run-time, based on available values.

## EVERY USER INTERFACE CONSIDERS THE FOLLOWING THREE MAIN ASPECTS

**UI Elements** – These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex, which we will cover in this tutorial.

**Layouts** – They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in the Layout chapter.

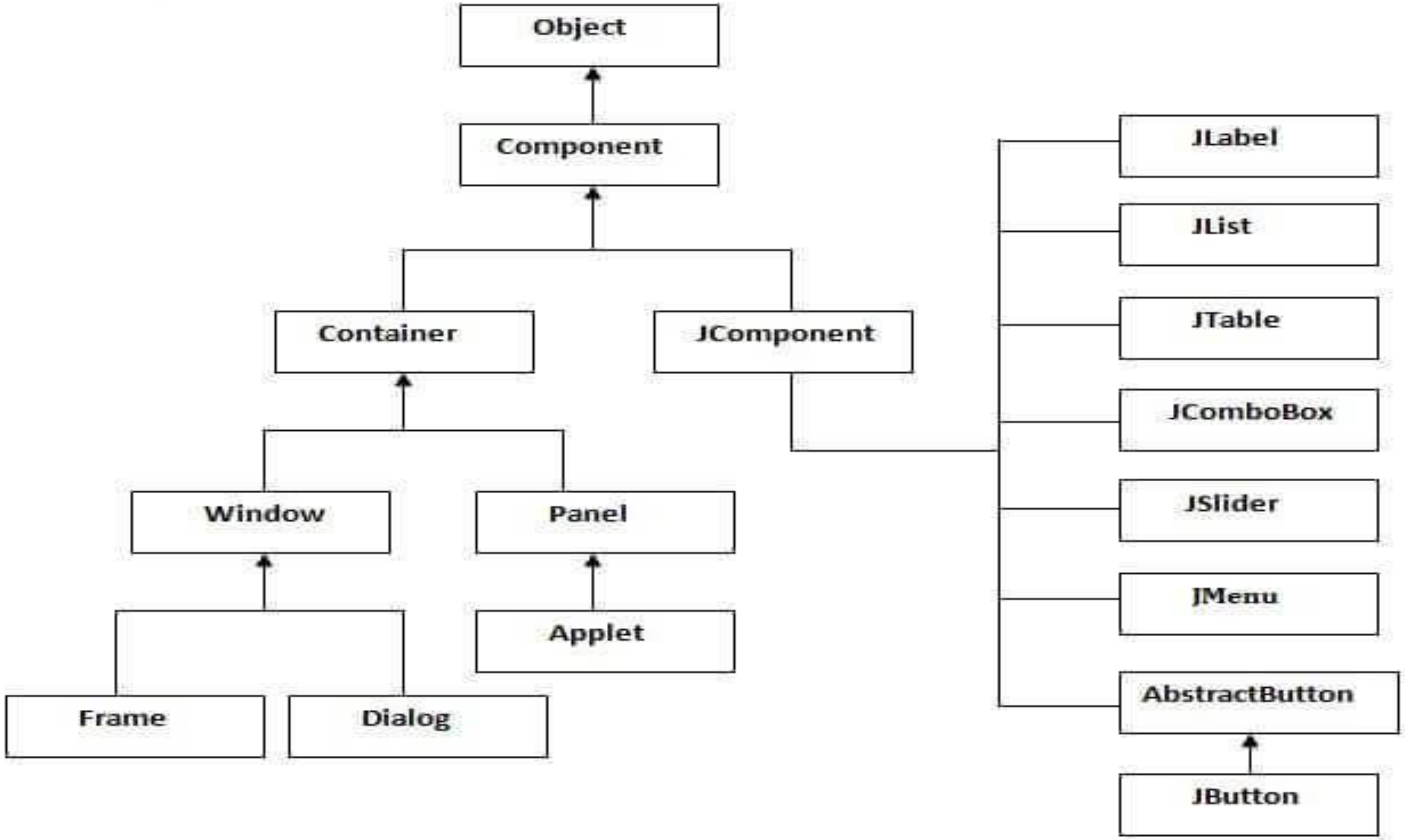
**Behavior** – These are the events which occur when the user interacts with UI elements.

## DIFFERENCE BETWEEN AWT AND SWING

There are many differences between java awt and swing that are given below.

| No. | Java AWT | Java Swing |
|---|---|---|
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| 4) | AWT provides **less components** than Swing. | Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing **follows MVC**. |

## HIERARCHY OF JAVA SWING CLASSES


Object
Component
Container
JComponent
Window
Panel
Frame
Dialog
Applet
JLabel
JList
JTable
JComboBox
JSlider
JMenu
AbstractButton
JButton

## CONTAINERS

Containers are an integral part of SWING GUI components. A container provides a space where a component can be located. A Container in AWT is a component itself and it provides the capability to add a component to itself. Following are certain noticable points to be considered.

1. Sub classes of Container are called as Container. For example, JPanel, JFrame and JWindow.
2. Container can add only a Component to itself.
3. A default layout is present in each container which can be overridden using **setLayout** method.

## COMMONLY USED METHODS OF COMPONENT CLASS

| Method | Description |
|---|---|
| public void add(Component c) | add a component on another component. |
| public void setSize(int width,int height) | sets size of the component. |
| public void setLayout(LayoutManager m) | sets the layout manager for the component. |
| public void setVisible(boolean b) | sets the visibility of the component. It is by default false. |

## JAVA SWING EXAMPLES

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

## EXAMPLE OF SWING BY ASSOCIATION INSIDE CONSTRUCTOR

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args) {
new Simple();
}
}
```

## SIMPLE EXAMPLE OF SWING BY INHERITANCE

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);
add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

## SWING COMPONENTS

## JAVA JBUTTON

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

## JAVA JLABEL

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

## JAVA JTEXTFIELD

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

## JAVA JTEXTAREA

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

## JAVA JPASSWORDFIELD

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class

## JAVA JCHECKBOX

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

## JAVA JRADIOBUTTON

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

## JAVA JCOMBOBOX

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

## JAVA JTABLE

The JTable class is used to display data in tabular form. It is composed of rows and columns.

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                      {"102","Jai","780000"},
                      {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

## JAVA JLIST

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

## JAVA JOPTIONPANE

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

## EXAMPLE 1

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

## EXAMPLE 2

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARNING_MESSAGE);
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

## EXAMPLE 3

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

## JAVA JSCROLLBAR

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

## JAVA JMENUBAR, JMENU AND JMENUITEM

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

## JAVA JPROGRESSBAR

The JProgressBar class is used to display the progress of the task. It inherits JComponent class.

# JAVA JCOLORCHOOSER

The JColorChooser class is used to create a color chooser dialog box so that user can select any color. It inherits JComponent class.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
JButton b;
Container c;
ColorChooserExample(){
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
}
public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}
public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

## JAVA JSLIDER

The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.

```
import javax.swing.*;
public class SliderExample1 extends JFrame{
public SliderExample1() {
JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
JPanel panel=new JPanel();
panel.add(slider);
add(panel);
}
public static void main(String s[]) {
SliderExample1 frame=new SliderExample1();
frame.pack();
frame.setVisible(true);
}
}
```

## EXAMPLE 2

```
import javax.swing.*;
public class SliderExample extends JFrame{
public SliderExample() {
JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
slider.setMinorTickSpacing(2);
slider.setMajorTickSpacing(10);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
JPanel panel=new JPanel();
panel.add(slider);
add(panel);
}
public static void main(String s[]) {
SliderExample frame=new SliderExample();
frame.pack();
frame.setVisible(true);
}
}
```

## JAVA JSPINNER

The object of JSpinner class is a single line input field that allows the user to select a number or an object value from an ordered sequence.

```
import javax.swing.*;
public class SpinnerExample {
    public static void main(String[] args) {
    JFrame f=new JFrame("Spinner Example");
    SpinnerModel value =
             new SpinnerNumberModel(5,1,10,1); // start, min, max, step
    JSpinner spinner = new JSpinner(value);
            spinner.setBounds(100,100,50,30);
            f.add(spinner);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
}
}
```

## JAVA JPANEL

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
     PanelExample()
        {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
        }
        public static void main(String args[])
        {
        new PanelExample();
        }
    }
```

## JAVA JTOGGLEBUTTON

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

## JAVA JFRAME

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

## HOW TO USE TOOLTIP IN JAVA SWING

You can create a tool tip for any JComponent with **setToolTipText()** method. This method is used to set up a tool tip for the component.

For example, to add tool tip to PasswordField, you need to add only one line of code:

```
import javax.swing.*;
public class ToolTipExample {
    public static void main(String[] args) {
     JFrame f=new JFrame("Password Field Example");
     //Creating PasswordField and label
     JPasswordField value = new JPasswordField();
     value.setBounds(100,100,100,30);
     value.setToolTipText("Enter your Password");
     JLabel l1=new JLabel("Password:");
l1.setBounds(20,100, 80,30);
     //Adding components to frame
     f.add(value);  f.add(l1);
     f.setSize(300,300);
     f.setLayout(null);
     f.setVisible(true);
}
}
```

## HOW TO CHANGE TITLEBAR ICON IN JAVA AWT AND SWING

The setIconImage() method of Frame class is used to change the icon of Frame or Window. It changes the icon which is displayed at the left side of Frame or Window.

The Toolkit class is used to get instance of Image class in AWT and Swing.

Toolkit class is the abstract super class of every implementation in the Abstract Window Toolkit(AWT). Subclasses of Toolkit are used to bind various components. It inherits Object class.

```
import java.awt.*;
class IconExample {
IconExample(){
Frame f=new Frame();
Image icon = Toolkit.getDefaultToolkit().getImage("D:\\icon.png");
f.setIconImage(icon);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);
}
public static void main(String args[]){
new IconExample();
}
}
```

## HOW TO MAKE AN EXECUTABLE JAR FILE IN JAVA

The **jar (Java Archive)** tool of JDK provides the facility to create the executable jar file. An executable jar file calls the main method of the class if you double click it.

To create the executable jar file, you need to create **.mf file,** also known as manifest file.

Creating manifest file

To create manifest file, you need to write Main-Class, then colon, then space, then classname then enter. For example:

```
Main-Class: First
```

## CREATING EXECUTABLE JAR FILE USING JAR TOOL

The jar tool provides many switches, some of them are as follows:

**-c** creates new archive file

**-v** generates verbose output. It displays the included or extracted resource on the standard output.

**-m** includes manifest information from the given mf file.

**-f** specifies the archive file name

**-x** extracts files from the archive file

Now, let's write the code to generated the executable jar using mf file.

You need to write **jar** then **swiches** then **mf_file** then **jar_file** then **.classfile** as given below:

jar -cvmf myfile.mf myjar.jar First.**class**

Now it will create the executable jar file. If you double click on it, it will call the main method of the First class.

## CONTINUE IN NEXT UNIT.....