OOP through JAVA (JNTUK-R19-2-1-ECE)
UNIT III: EVENT AND GUI PROGRAMMING

Syllabus:

Event handling in java, Event types, Mouse and key events, GUI Basics, Panels, Frames, Layout Managers: Flow Layout, Border Layout, Grid Layout, GUI components like Buttons, Check Boxes, Radio Buttons, Labels, Text Fields, Text Areas, Combo Boxes, Lists, Scroll Bars, Sliders, Windows, Menus, Dialog Box, Applet and its life cycle, Introduction to swing, Creating a swing applet, swing controls and components.

Introduction:

- Applets are basically small Java programs which can be easily transported over the network from one computer to other.
- This is the reason why applets are
  - used in Internet applications,
  - embedded in an html page
  - downloaded from the server and
  - run on the client
- Applets have the capability of
  - displaying graphics,
  - playing sound,
  - creating animation, and
  - performing other jobs
- Applets can be executed on the client using a Java-enabled browser or a utility known as appletviewer.
- In Java, applets can be dealt in two ways.
  - By using Applet class in Abstract Window Toolkit (AWT).
  - By using JApplet class in swings.

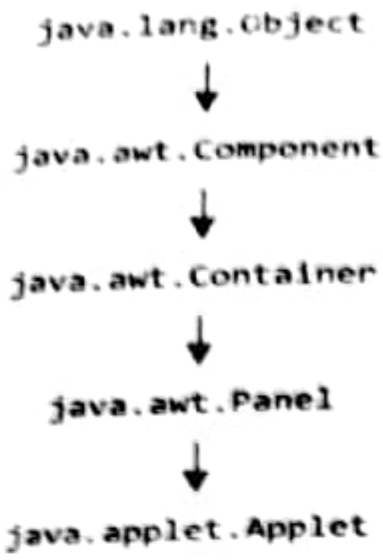
Applet Vs Application:

Applet	Application
The execution of the applet does not start from `main()` method. as it does not have one.	The execution of an application program starts from `main()`.
Applets cannot run on their own: They have to be embedded inside a web page to get executed.	These can run on their own. In order to get executed, they need not be embedded inside any web page.
Applets can only be executed inside a browser or appletviewer.	Applications are executed at command line.
Applets execute under strict security limitations that disallow certain operations (sandbox model security).	Applications have no inherent security restrictions.
Applets have their own life cycle `init()→start()→paint()→stop→destroy()`	Applications have their own life cycle. Their execution begins at `main()`.

1. Applet class:

- Applet is the superclass of all the applets from java.applet package.

- This class provides all the necessary methods for starting, stopping, and manipulating applets.
- It also has methods providing multimedia support to an applet.
- Applet class has a predefined hierarchy in Java, which shows the classes extended by Applet class.
- The Hierarchy of Applet Class is given as,



Applet Class Methods:

Method	Description
void init()	First method to be called when an applet begins execution.
boolean isActive()	Returns true if the applet is running, otherwise false.
URL getDocumentBase()	Gets the URL of the document in which this applet is embedded.
URL getCodeBase()	Returns the URL of the directory where the class file of the invoking applet exists.
String getParameter (String name)	Returns the value of the parameter associated with parameter's name. Null is returned if the parameter is not specified.
AppletContext getAppletContext()	Determines this applet's context, which allows the applet to query and affect the environment in which it runs.
void resize (int width,int height)	Resizes the applet according to the parameters, *width* and *height*.
void showStatus(String msg)	Displays the string, *msg*, in the status window of the browser or appletviewer (only if they support status window).
Image getImage(URL url)	Returns an object of image, which binds the image found at the URL, specified as the argument of the method.
Image getImage(URL url, String imgName)	Returns the image object which encapsulates the image found at the specified URL and having the name specified by imgName.
static final AudioClip newAudioClip(URL url)	Returns an AudioClip object that encapsulates the audio found at the URL specified as the argument.
void start()	Starts or resumes the execution of applet.
void stop()	Stop or suspends the applet.
void destroy()	Terminates the applet.
AccessibleContext getAccessibleContext()	Returns the accessibility context for the invoking object.
AudioClip getAudioClip(URL url)	Returns the AudioClip object, which encapsulates the audio clip found at the URL, specified as the argument to the method.
AudioClip getAudioClip(URL url,String clipName)	Returns the AudioClip object, which encapsulates the audio clip found at URL, specified as the argument to the method and having the name specified by clipName.
String getAppletInfo()	Returns the string describing the applet.
Locale getLocale()	Returns the Locale object that is used by various locale sensitive classes and methods.
String[][] getParameterInfo()	Returns a string table that describes the parameter recognized by the applet.

2. Applet Program Structure:

```
import java.awt.*;
import java.applet.*;
.....
.....
public class NewApplet extends Applet
  {
    .....
    .....
  public void paint(Graphics g)
    {
      .....
      .....
    }
  .....
  .....
 }
```

How to Run an Applet?

- There are two ways to run an applet.

Method1:

- Save the file as FirstApplet.java and compile it by using javac.
- Now, type HTML code given into a text editor and save the file as FirstApplet.html

```
<HTML>
<BODY>
<APPLET code = "FirstApplet.class" WIDTH = 200 HEIGHT = 150></APPLET>
</BODY>
</HTML>
```

- You can execute the HTML file by giving

```
appletviewer FirstApplet.html
```

Method2:

- Save the file as FirstApplet.java and compile it by using javac.
- In order to run the applet, you have to give the below HTML coding as a comment in FirstApplet.java.

```
/* <APPLET code = "FirstApplet.class" WIDTH = 200 HEIGHT = 150></APPLET> */
```

- Execute the applet as,

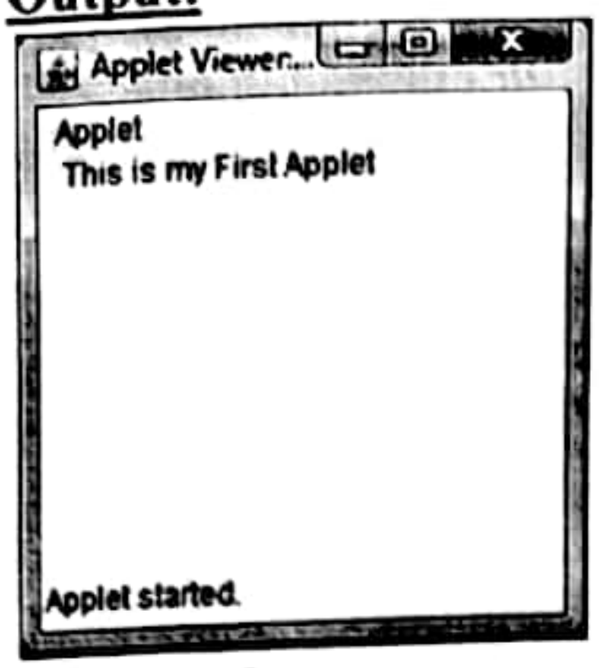
```
appletviewer FirstApplet.java
```

Example-1:

```
// Applet program to print a line of text
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet
 {
```

```
public void paint(Graphics g)
  {
   g.drawString("This is my First Applet", 10, 10);
  }
}
```

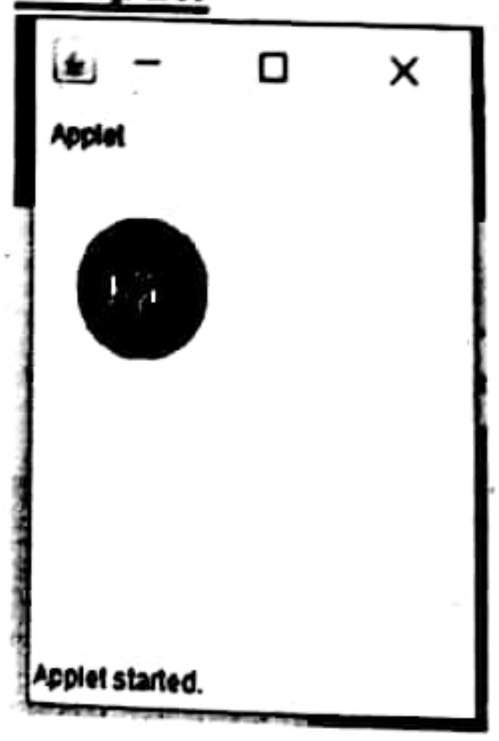
Output:



Example-2:

```
//Set the Color of the Applet and Draws a Fill Oval
/* <APPLET code = "FillOval.class" WIDTH = 200 HEIGHT = 200></APPLET> */
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
public class FillOval extends Applet
{
 public void paint(Graphics g)
   {
    g.setColor(Color.red);
    g.fillOval(20, 20, 60, 60);
   }
}
```

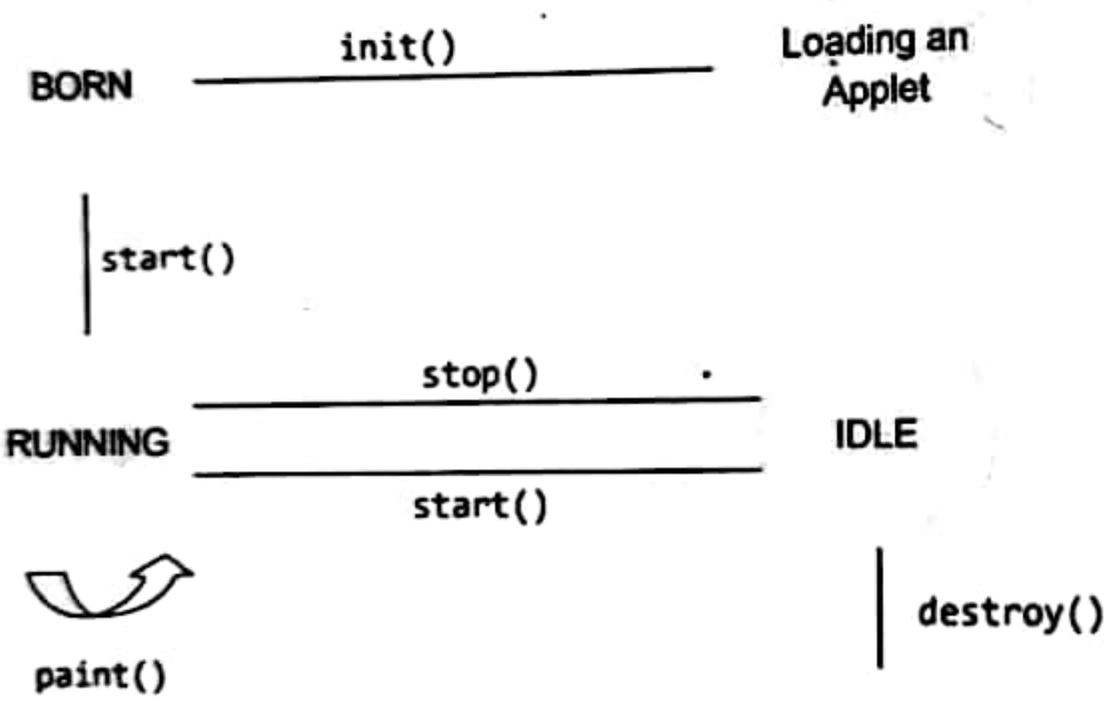
Output:



3. APPLET LIFE CYCLE:

- An applet may move from one state to another depending upon a set of default behaviors inherited in the form of methods from Applet class.
- These states can be summed up as,
  - Born
  - Running
  - Idle
  - Dead
- An applet may override some of the basic methods of class Applet.
  - init()
  - start()
  - stop()
  - destroy()
- These methods are responsible for the lifecycle of an applet.

- Applet's State Diagram or Life cycle of an Applet is depicted as



4. EVENT HANDLING:

- Event handling is a mechanism that is used to handle events generated by applets.
- An event could be the occurrence of any activity such as a mouse click or a key press
- In java, events are regarded as method calls with a certain task performed against the occurrence o each event

i) EVENT DELEGATION MODEL:

- It is an approach that has been followed since Java 1.0.
- In the event delegation model, a source generates events which are sent to one or more listeners.
- The listeners are responsible for receiving the event, which once received are processed or handle in the way required.
- Here the processing logic applied for handling an event means the logic that generates the event
- The model has three dimensions, namely

- events,
- event sources, and
- event listeners

- An event is an object that describes a state change in the source.
- An event source is an object which generates the event.
- Generation of event must cause a change in the state of the source.
- A source can generate more than one event.
- Event Listeners are the objects that get notified when an event occurs on an event source.

ii) SOURCES OF EVENTS:

- Sources of events can be either components of GUI or any other class derived from a component (such as an applet), which can generate event like events from keyboard and mouse.
- Components of GUI that Can Generate Events are depicted as,

Button	Choice	Menu item
Check box	List	Window
Scroll bar	Text components	

iii) EVENT LISTENERS:

- Event listeners are created by implementing one or more interfaces defined by the java.awt. event package.
- Whenever a source generates an event, it basically invokes the appropriate method defined in the listener interface.
- The method has an event object passed as an argument to it.
- List of Event Listeners is given as,

KeyListener	ItemListener	WindowListener
MouseListener	ActionListener	ComponentListener
MouseMotionListener	TextListener	ContainerListener
MouseWheelListener	FocusListener	AdjustmentListener

a) KeyListener Interface:

- This interface has three methods defined within it:
  - void keyPressed(KeyEvent e)
  - void keyReleased(KeyEvent e)
  - void keyTyped(KeyEvent e)
- keyPressed() is invoked when a key is pressed,
- keyReleased() is invoked when a key is released, and
- keyTyped() is invoked when a character is typed.

b) MouseListener Interface:

- This interface has five methods, having the signatures as follows:

- void mouseClicked(MouseEvent e)
- void mouseEntered(MouseEvent e)
- void mousePressed(MouseEvent e)
- void mouseReleased(MouseEvent e)
- void mouseExited(MouseEvent e)

- mouseClicked() is invoked when a mouse is clicked,
- mousePresssed() is invoked when a mouse is pressed but not released,
- mouseReleased() is invoked when a pressed mouse is released, and
- mouseExited() is invoked when the mouse leaves a component.

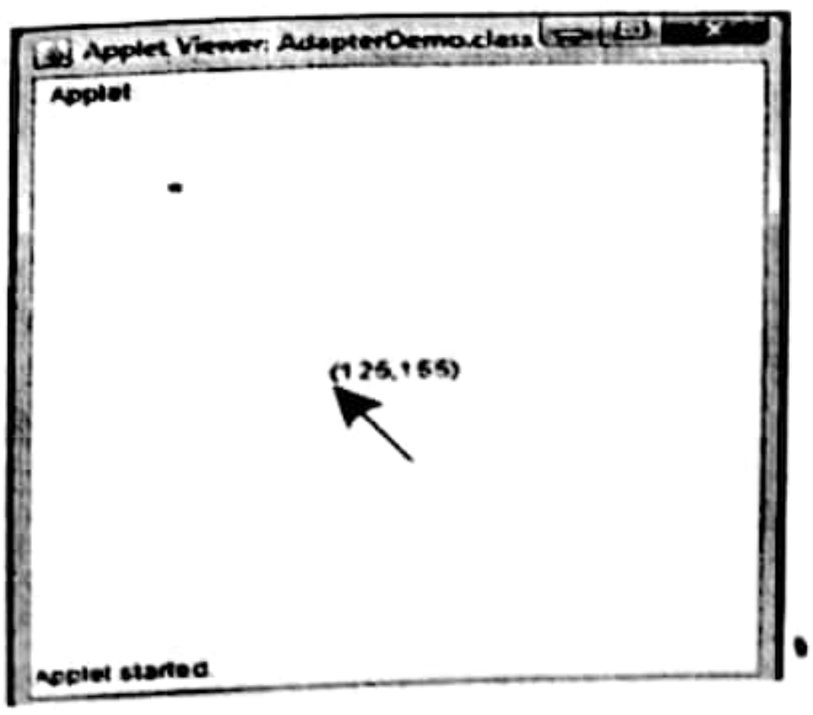
c) MouseMotionListener Interface:

- This interface has two methods having the signatures,
  - void mouseMoved(MouseEvent e)
  - void mouseDragged(MouseEvent e)
- mouseMoved() is invoked when the mouse is moved from one place to another and
- mouseDragged() is used when the mouse is dragged.

Example Program:

```
// Use of MouseMotionListener
/*<applet code = "MouseMotionEx.class" width = 300 height = 300></applet>*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class MouseMotionEx extends Applet implements MouseMotionListener
{
 int xcord;
 int ycord;
 public void init()
  {
   addMouseMotionListener(this);
  }
 public void paint(Graphics g)
  {
    g.drawString("("+xcord+","+ycord+")",xcord,ycord);
  }
 public void mouseMoved(MouseEvent me)
    {
     xcord = me.getX();
     ycord = me.getY();
     repaint();
    }
  public void mouseDragged(MouseEvent me) { }
}
```

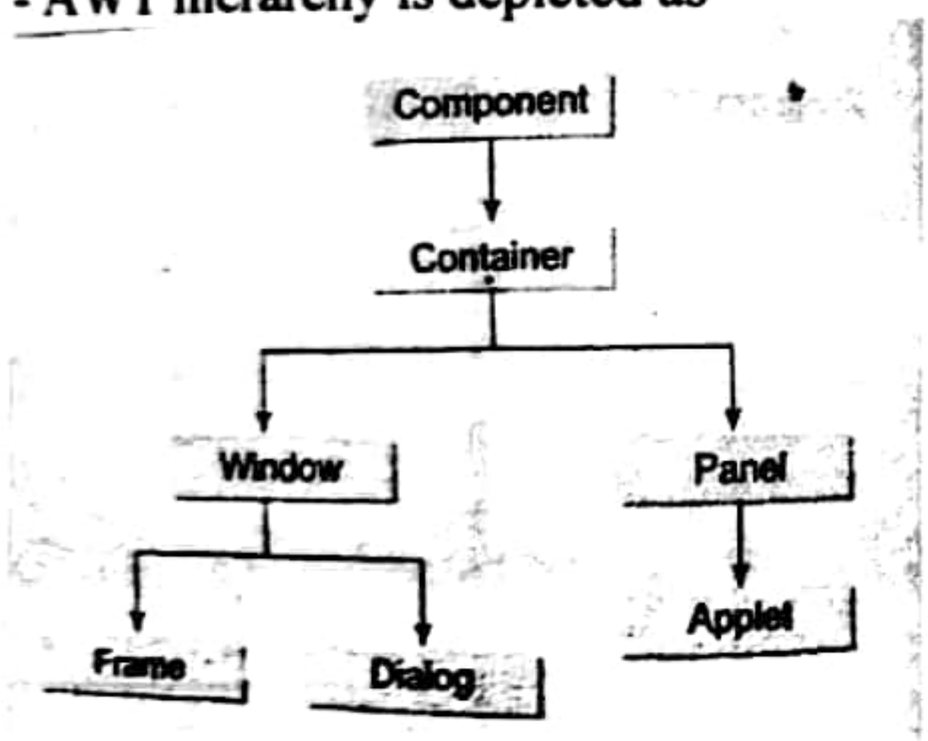
Output:



5. GUI BASICS: (AWT & SWINGS)

i) AWT:

- The AWT (Abstract Windowing Toolkit) package in java enables the programmers to create GUI-based applications.
- It contains a number of classes that help to implement common windows-based tasks, such as manipulating windows, adding scroll bars, buttons, list items, text boxes, etc.
- All the classes are contained in the java.awt package.
- AWT provides both standard and applet windows
- AWT hierarchy is depicted as



a) Component:

- Component class is the super class to all the other classes from which various GUI elements are realized.
- It is a primary responsible for effecting the display of a graphic object on the screen.
- It also handles the various keyboard and mouse events of the GUI application.

b) Container:

- The container object contains the other AWT components.
- It manages the layout and placement of the various awt components within the container.
- A container object can contain other containers objects as well; thus, allowing nesting of containers.

c) Window:

- The Window object realizes a top-level window but without any border or menu bar.
- It just specifies the layout of the window.

- A typical window that you would want to create in your application is not normally derived from the Window class but from its sub class that is Frame.

d) Panel:

- The superclass of applet, Panel represents Windows space on which the application's output is displayed.
- It is just like a normal window having no border, title bar, menu bar etc.
- A Panel can contain within itself other panels as well.

e) Frame:

- The Frame object realizes at top-level window complete with border and menu bar.
- It Supports common window related events such as open, close, activate, deactivate, etc.

ii) GUI COMPONENTS:

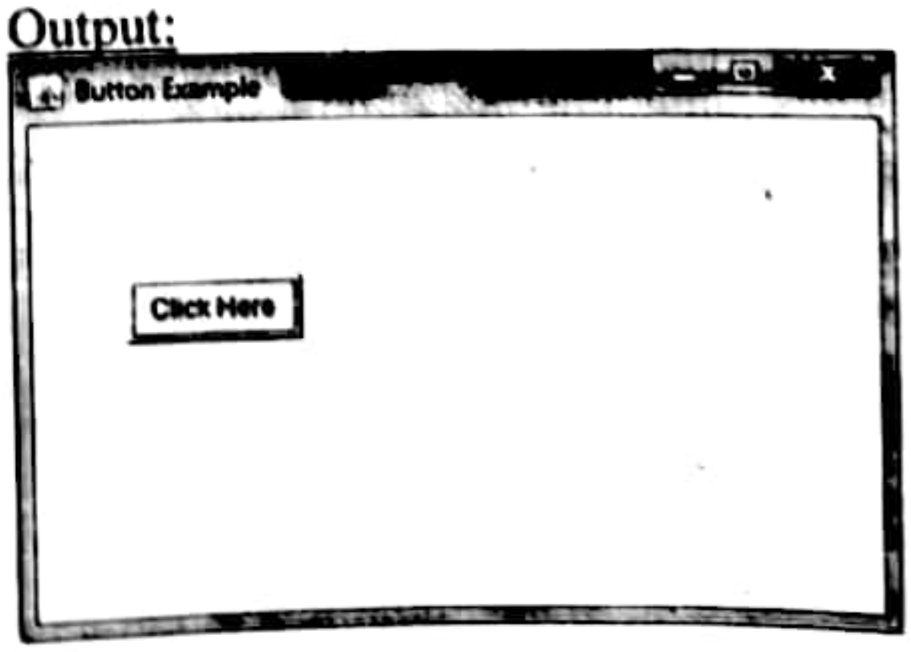
a)Java AWT Button:

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

Example:

```
import java.awt.*;
public class ButtonExample {
public static void main(String[] args) {
    Frame f=new Frame("Button Example");
    Button b=new Button("Click Here");
    b.setBounds(50,100,80,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



b) Java AWT Checkbox:

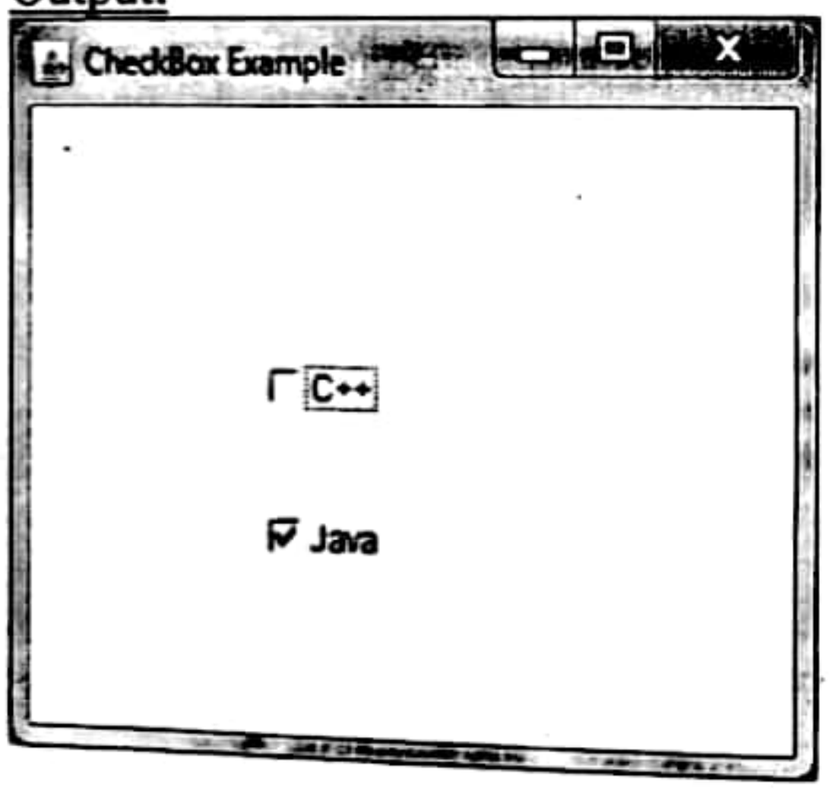
- The Checkbox class is used to create a checkbox.
- It is used to turn an option on (true) or off (false).
- Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Example:

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

```
public class CheckboxExample
{
    CheckboxExample(){
     Frame f= new Frame("Checkbox Example");
      Checkbox checkbox1 = new Checkbox("C++");
      checkbox1.setBounds(100,100, 50,50);
      Checkbox checkbox2 = new Checkbox("Java", true);
      checkbox2.setBounds(100,150, 50,50);
      f.add(checkbox1);
      f.add(checkbox2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
  public static void main(String args[])
  {
     new CheckboxExample();
  }
  }
```

Output:



c) Java AWT Radio Button/ CheckboxGroup:

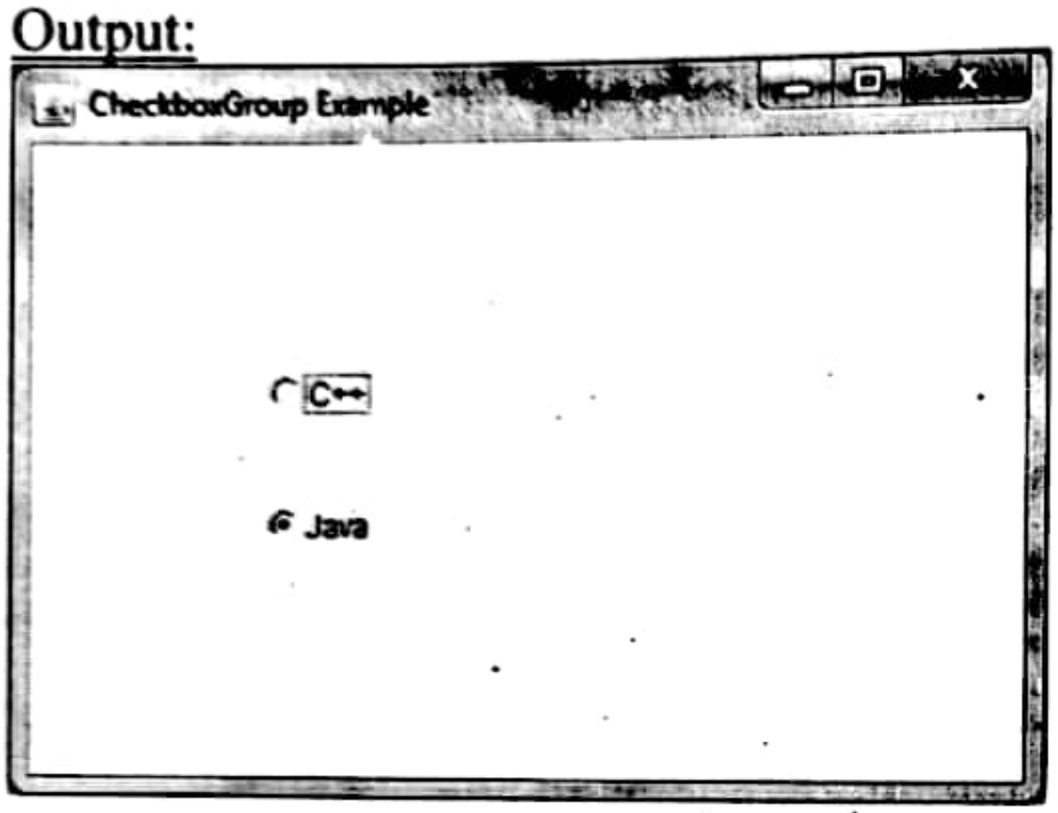
- The object of CheckboxGroup class is used to group together a set of Checkbox.
- At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state.
- It inherits the object class.

Example:

```
import java.awt.*;
public class CheckboxGroupExample
{
     CheckboxGroupExample(){
     Frame f= new Frame("CheckboxGroup Example");
      CheckboxGroup cbg = new CheckboxGroup();
      Checkbox checkBox1 = new Checkbox("C++", cbg, false);
      checkBox1.setBounds(100,100, 50,50);
```

```
        Checkbox checkBox2 = new Checkbox("Java", cbg, true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxGroupExample();
    }
    }
```

Output:



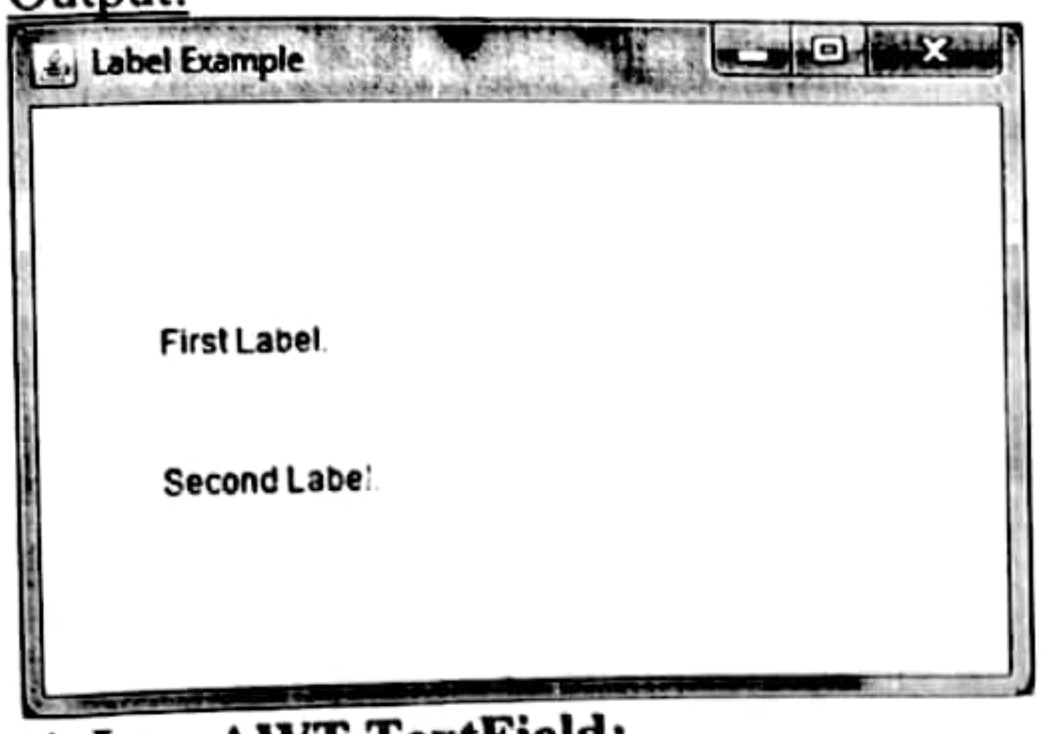
d) Java AWT Label:

- The object of Label 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.

Example:

```
    import java.awt.*;
    class LabelExample{
    public static void main(String args[]){
       Frame f= new Frame("Label Example");
       Label l1,l2;
       l1=new Label("First Label.");
       l1.setBounds(50,100, 100,30);
       l2=new Label("Second Label.");
       l2.setBounds(50,150, 100,30);
       f.add(l1); f.add(l2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
}
}
```

Output:



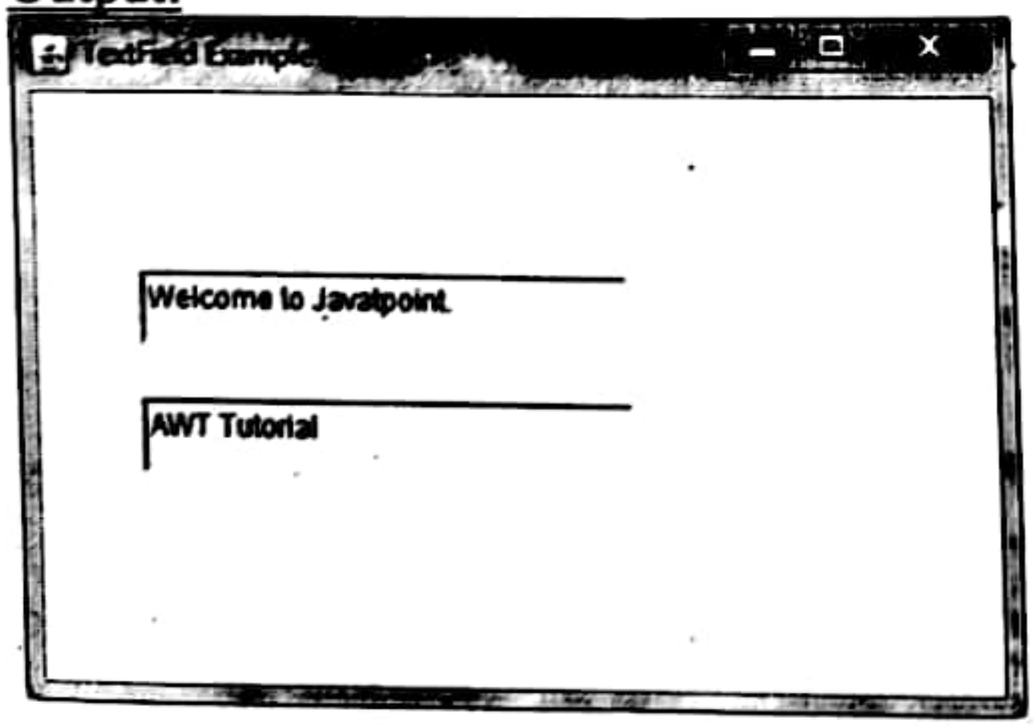
e) Java AWT TextField:

- The object of a TextField class is a text component that allows the editing of a single line text.
- It inherits TextComponent class

Example:

```
import java.awt.*;
class TextFieldExample{
public static void main(String args[]){
   Frame f= new Frame("TextField Example");
   TextField t1,t2;
   t1=new TextField("Welcome to Javatpoint.");
   t1.setBounds(50,100, 200,30);
   t2=new TextField("AWT Tutorial");
   t2.setBounds(50,150, 200,30);
   f.add(t1); f.add(t2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Output:



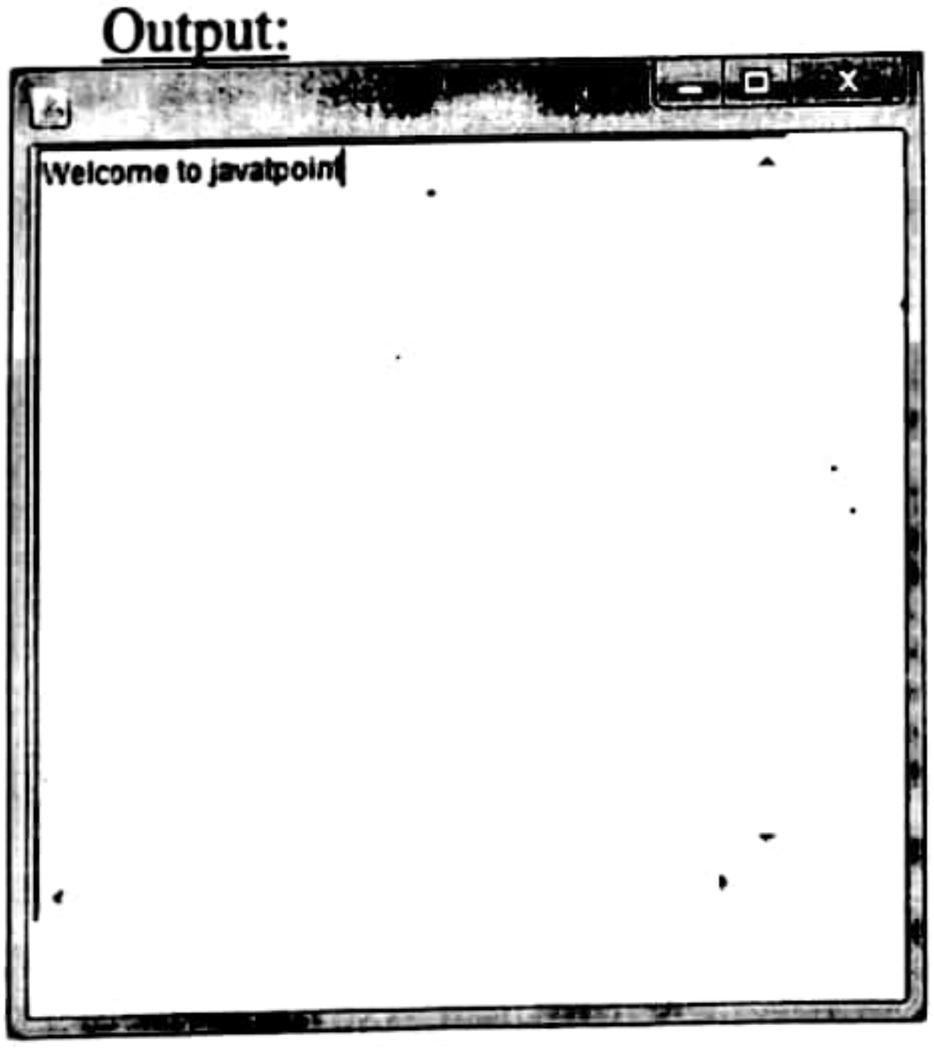
f) Java AWT TextArea:

- The object of a TextArea class is a multi line region that displays text.
- It allows the editing of multiple line text. It inherits TextComponent class.

Example:

```
import java.awt.*;
public class TextAreaExample
{
        TextAreaExample(){
        Frame f= new Frame();
                TextArea area=new TextArea("Welcome to javatpoint");
        area.setBounds(10,30, 300,300);
        f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        }
public static void main(String args[])
{
new TextAreaExample();
}
}
```

Output:



g) Java AWT Combo Boxes/ Choice:

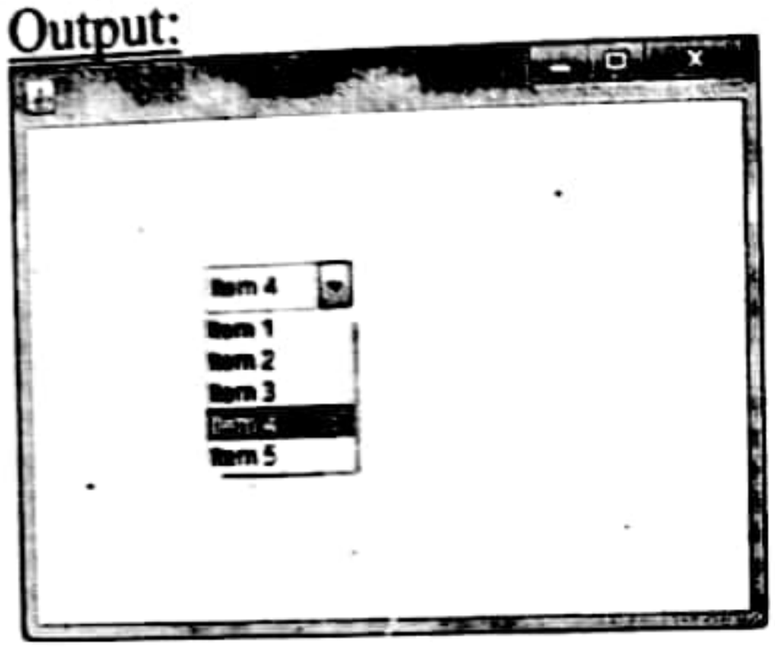
- The object of Choice class is used to show popup menu of choices.
- Choice selected by user is shown on the top of a menu.
- It inherits Component class.

Example:

```
import java.awt.*;
public class ChoiceExample
{
    ChoiceExample(){
    Frame f= new Frame();
    Choice c=new Choice();
```

```
c.setBounds(100,100, 75,75);
c.add("Item 1");
c.add("Item 2");
c.add("Item 3");
c.add("Item 4");
c.add("Item 5");
f.add(c);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
    }
public static void main(String args[])
{
  new ChoiceExample();
}
}
```

Output:



h) Java AWT List:

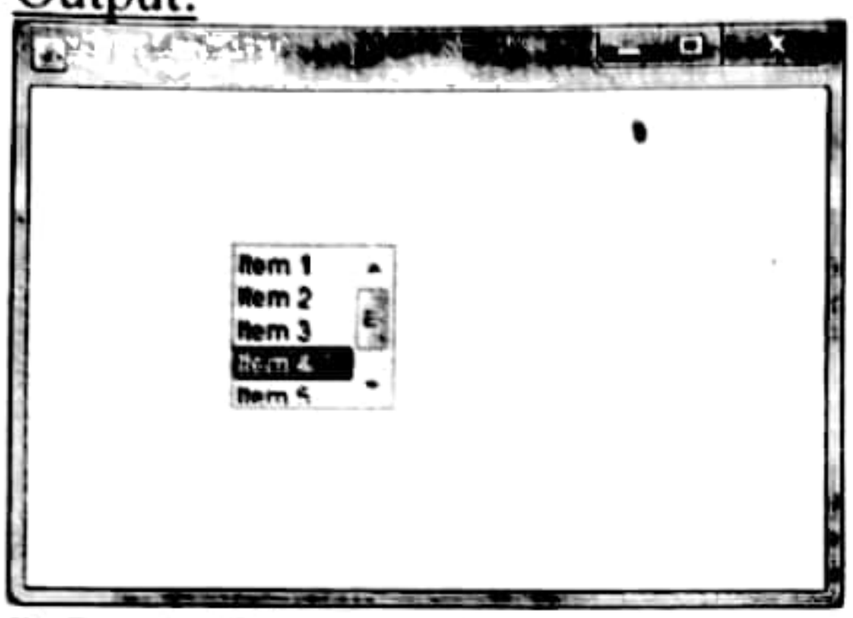
- The object of List class represents a list of text items.
- By the help of list, user can choose either one item or multiple items.
- It inherits Component class.

Example:

```
import java.awt.*;
public class ListExample
{
    ListExample(){
      Frame f= new Frame();
List l1=new List(5);
l1.setBounds(100,100, 75,75);
l1.add("Item 1");
l1.add("Item 2");
l1.add("Item 3");
l1.add("Item 4");
l1.add("Item 5");
f.add(l1);
f.setSize(400,400);
```

```
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
  new ListExample();
}
}
```

Output:



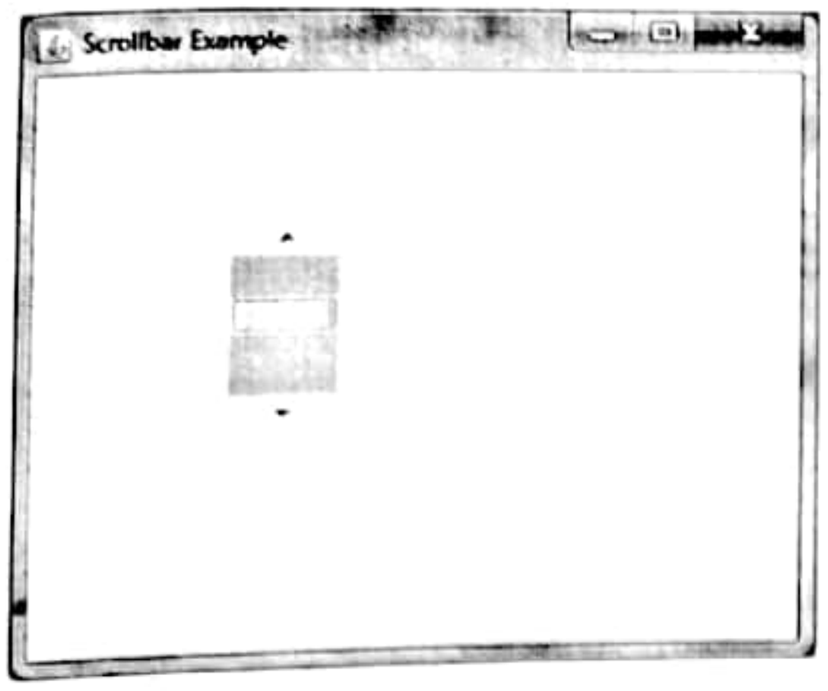
i) Java AWT Scrollbar:

- The object of Scrollbar class is used to add horizontal and vertical scrollbar.
- Scrollbar is a GUI component allows us to see invisible number of rows and columns.

Example:

```
import java.awt.*;
class ScrollbarExample{
ScrollbarExample(){
        Frame f= new Frame("Scrollbar Example");
        Scrollbar s=new Scrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[]){
    new ScrollbarExample();
}
}
```

Output:



i) Java AWT MenuItem and Menu:

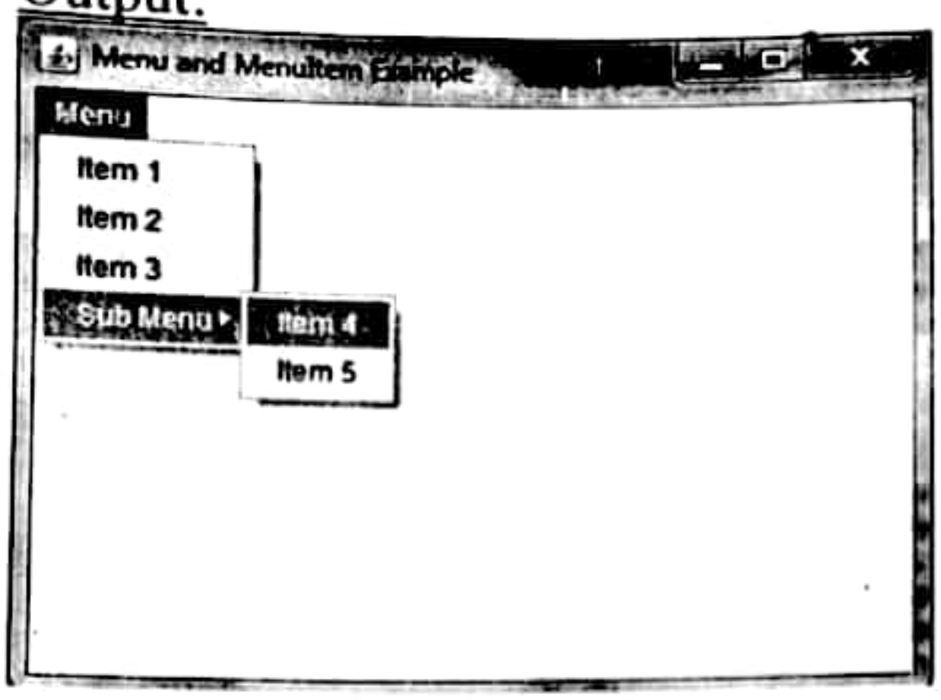
- The object of MenuItem class adds a simple labeled menu item on menu.
- The items used in a menu must belong to the MenuItem or any of its subclass.
- The object of Menu class is a pull down menu component which is displayed on the menu bar.
- It inherits the MenuItem class.

Example:

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
```

```
    }
    }
```

Output:



k) Java AWT PopupMenu:

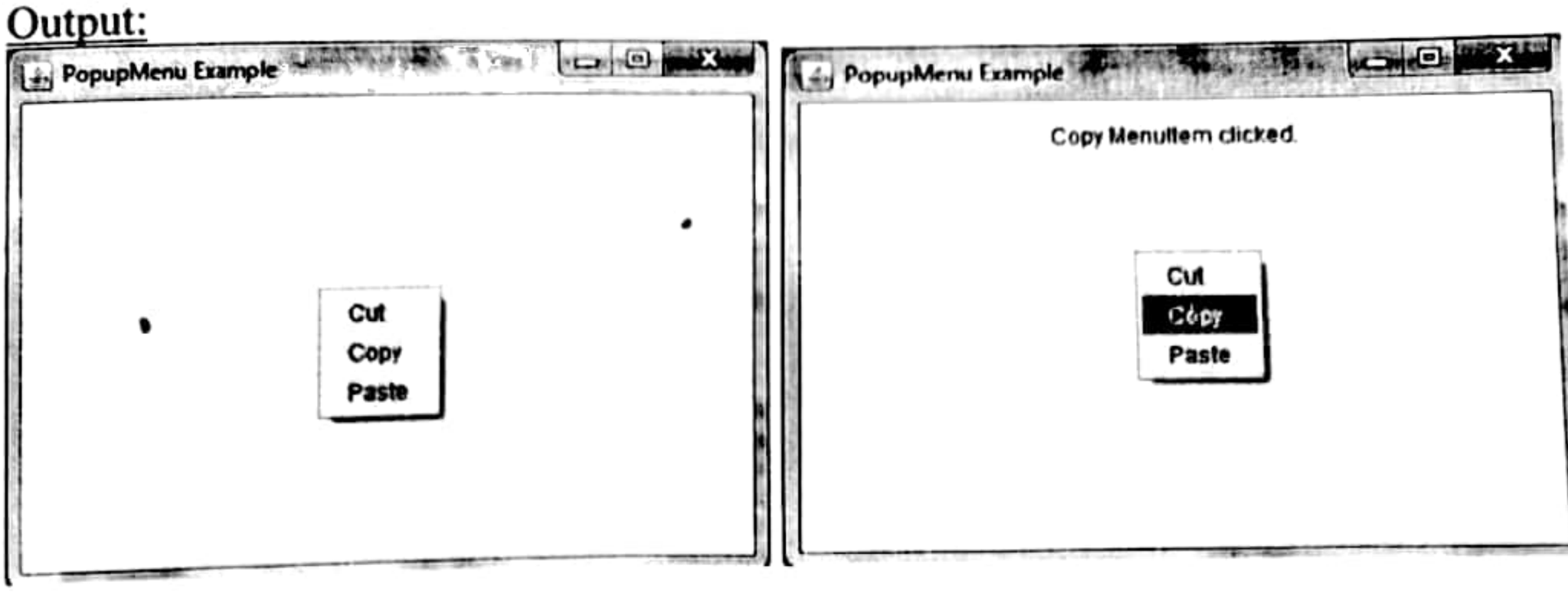
- PopupMenu can be dynamically popped up at specific position within a component.
- It inherits the Menu class.

Example:

```
import java.awt.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample(){
        final Frame f= new Frame("PopupMenu Example");
        final PopupMenu popupmenu = new PopupMenu("Edit");
        MenuItem cut = new MenuItem("Cut");
        cut.setActionCommand("Cut");
        MenuItem copy = new MenuItem("Copy");
        copy.setActionCommand("Copy");
        MenuItem paste = new MenuItem("Paste");
        paste.setActionCommand("Paste");
        popupmenu.add(cut);
        popupmenu.add(copy);
        popupmenu.add(paste);
        f.addMouseListener(new MouseAdapter() {
          public void mouseClicked(MouseEvent e) {
            popupmenu.show(f , e.getX(), e.getY());
          }
        });
        f.add(popupmenu);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
```

```
    {
        new PopupMenuExample();
    }
    }
```

Output:



I) Java AWT Dialog:

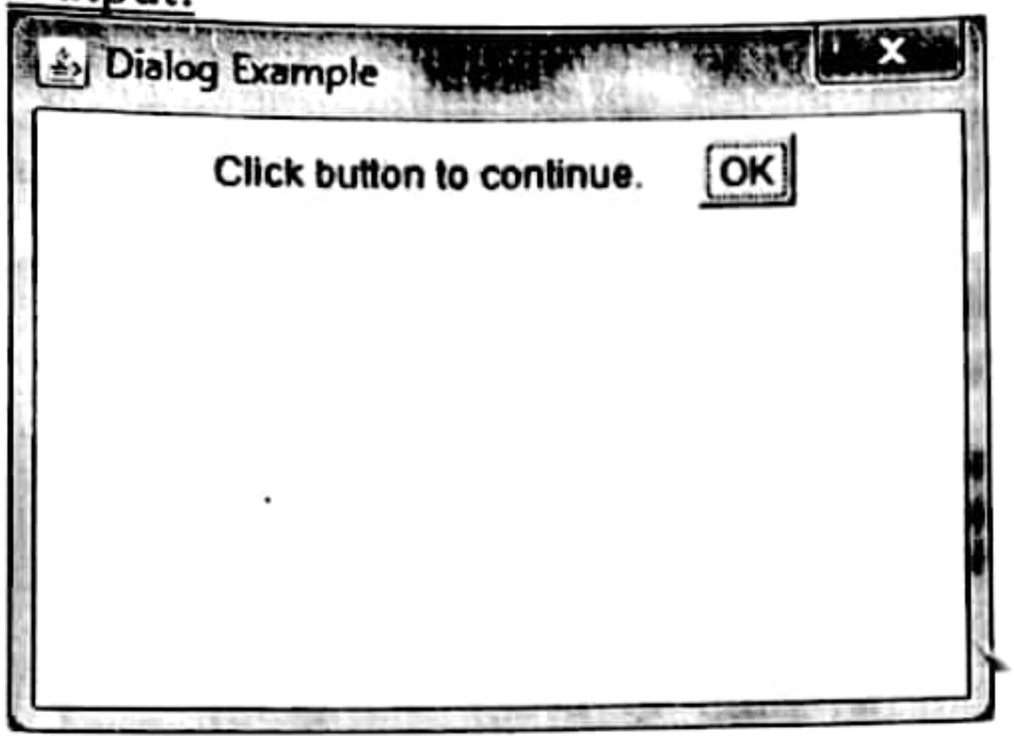
- The Dialog control represents a top level window with a border and a title used to take some form of input from the user.
- It inherits the Window class.
- Unlike Frame, it doesn't have maximize and minimize buttons.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
private static Dialog d;
DialogExample() {
   Frame f= new Frame();
   d = new Dialog(f , "Dialog Example", true);
   d.setLayout( new FlowLayout() );
   Button b = new Button ("OK");
   b.addActionListener ( new ActionListener()
   {
      public void actionPerformed( ActionEvent e )
      {
         DialogExample.d.setVisible(false);
      }
   });
   d.add( new Label ("Click button to continue."));
   d.add(b);
   d.setSize(300,300);
   d.setVisible(true);
}
public static void main(String args[])
{
```

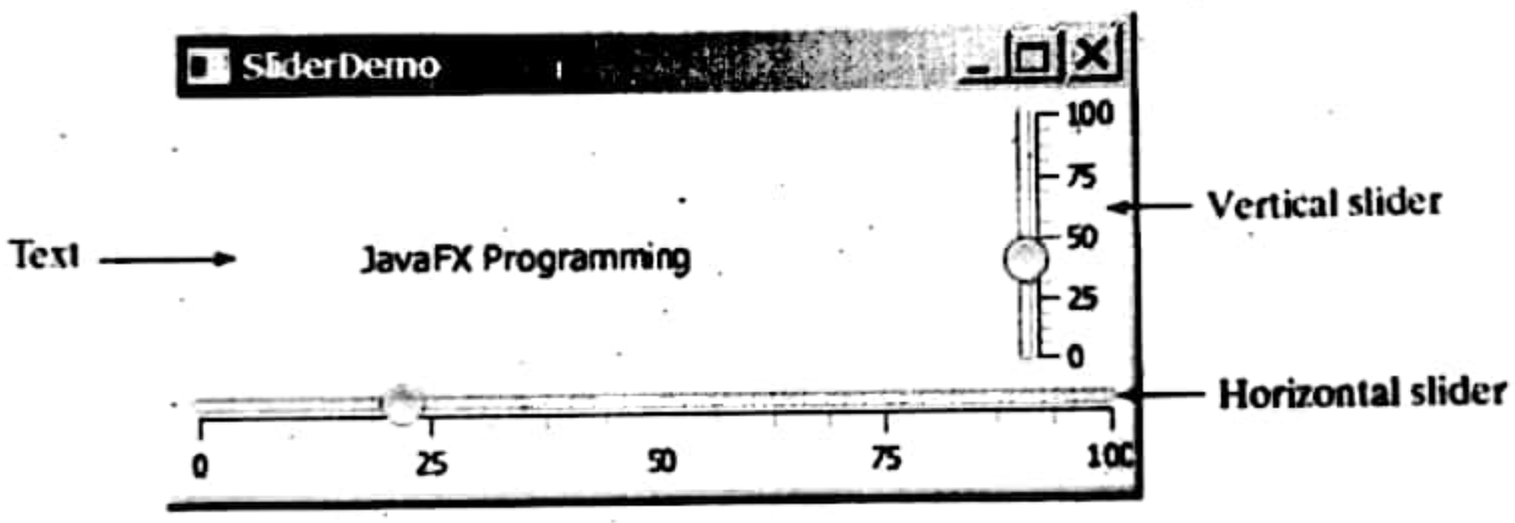
```
            new DialogExample();
        }
    }
```

Output:



JAVA Slider:

- **Slider** is similar to **ScrollBar**, but **Slider** has more properties and can appear in many forms
- The given diagram shows two sliders.



- **Slider** lets the user graphically select a value by sliding a knob within a bounded interval.
- The slider can show both major tick marks and minor tick marks between them.
- The number of pixels between the tick marks is specified by the **majorTickUnit** and **minorTickUnit** properties.
- Sliders can be displayed horizontally or vertically, with or without ticks, and with or with
- The sliders move the message on a pane horizontally and vertically

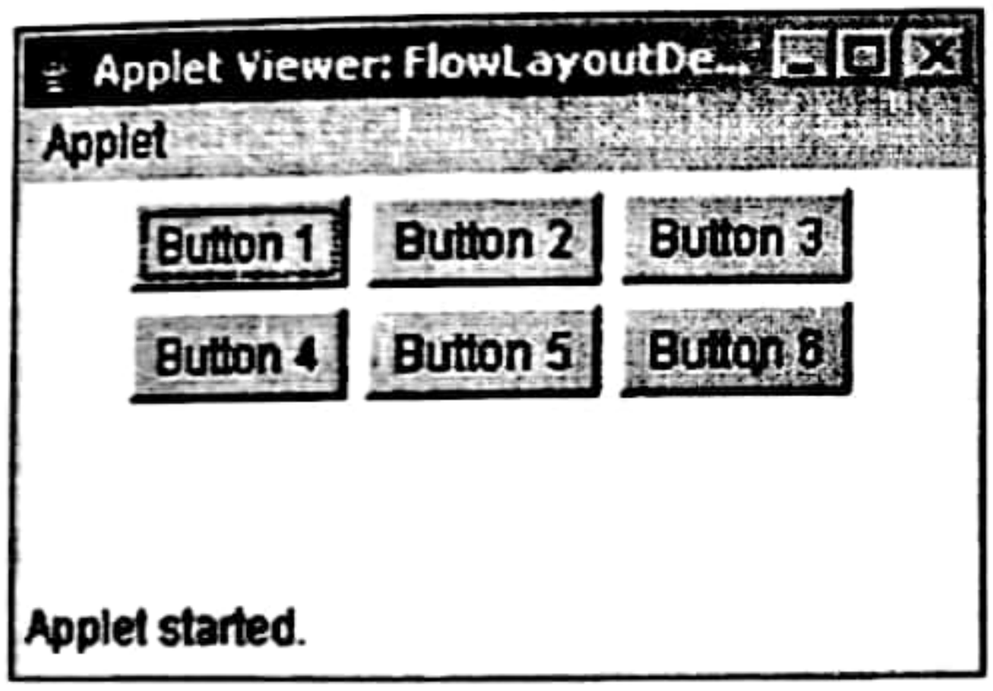
iii) LAYOUT MANAGERS IN AWT:

- Java has the mechanism to specify the type of layout schemes, where components can be to a Frame instance.
- This mechanism is specified by LayoutManager, which is used to specify how the compo be arranged inside a container, say a Frame.
- The different lay out managers are,
    a) FlowLayout
    b) BorderLayout
    c) CardLayout
    d) GridLayout
    e) GridBagLayout

a) FlowLayout:

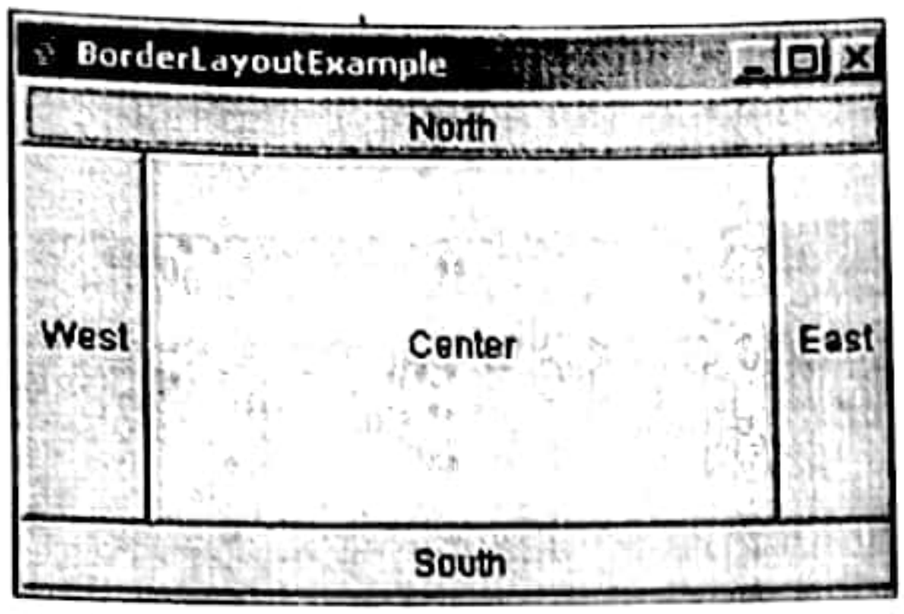
```
/*<applet code = FlowLayoutDemo.class width = 200 height = 200></applet>*/
import java.applet.Applet;
import java.awt.*;
public class FlowLayoutDemo extends Applet{
LayoutManager fl owLayout;
Button [] Buttons;
public FlowLayoutDemo() {
int i;
flowLayout = new FlowLayout ();
setLayout (fl owLayout);
Buttons = new Button [6];
for (i = 0; i < 6; i++) {
Buttons[i] = new Button ();
Buttons[i].setLabel ("Button " + (i + 1));
add (Buttons[i]);
}
}
}
```

Output:

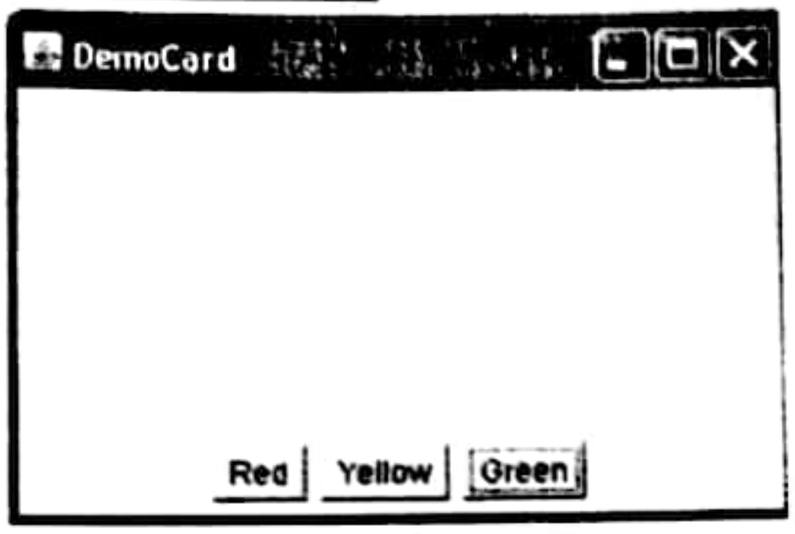


b) BorderLayout:

```
import java.awt.*;
public class BLayoutDemo extends Frame {
public BLayoutDemo(String title) {
super(title);
add(new Button("North"),BorderLayout.NORTH);
add(new Button("South"),BorderLayout.SOUTH);
add(new Button("East"),BorderLayout.EAST);
add(new Button("West"),BorderLayout.WEST);
add(new Button("Center"),BorderLayout.CENTER);
setSize(400, 270);
setVisible(true);
}
public static void main(String[] args) {
BLayoutDemo blaypout = new BLayoutDemo("Border Layout Example");
}}
```

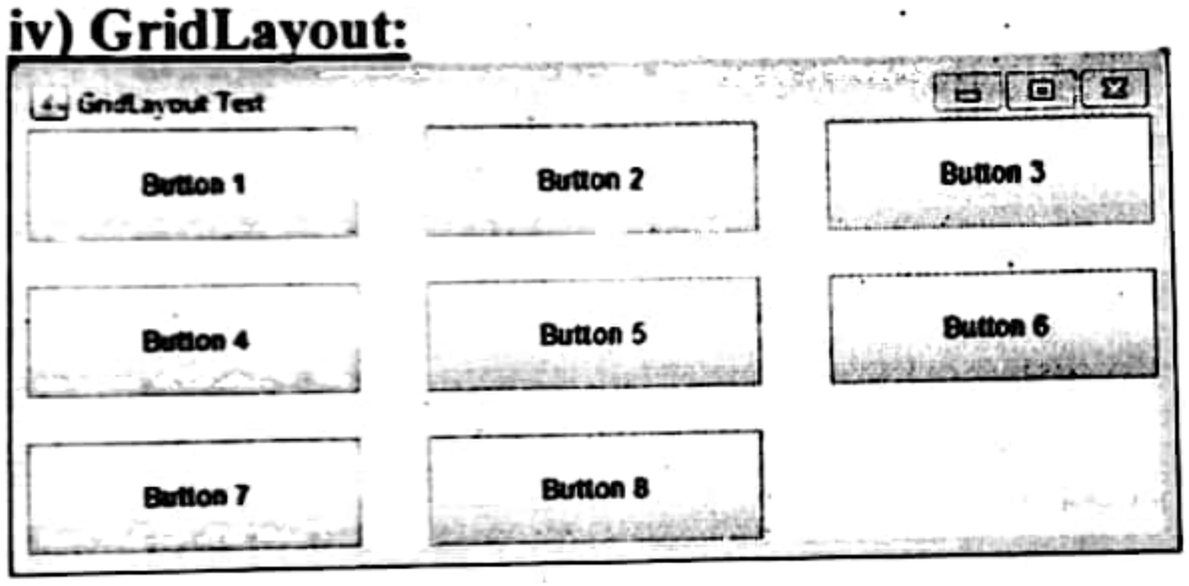


c) CardLayout:

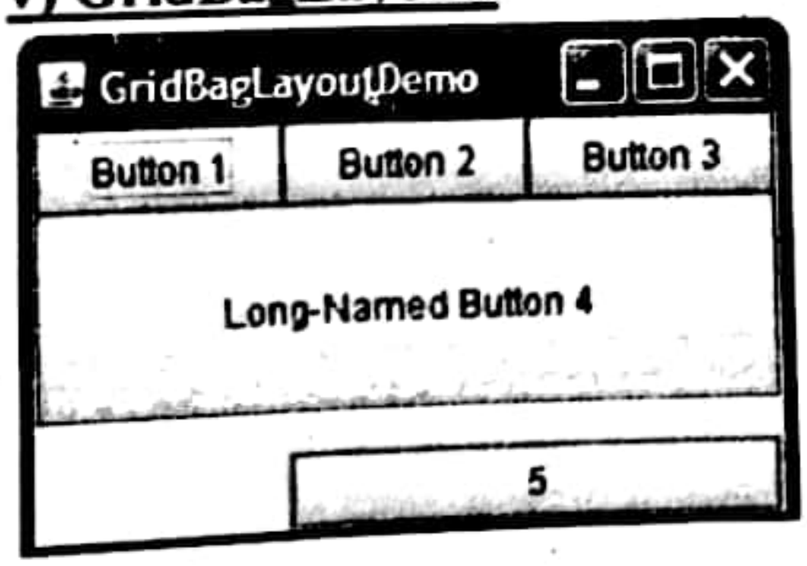


Actually

iv) GridLayout:



v) GridBagLayout:



iv) SWINGS:

- Similar to AWT, swing is also a GUI toolkit that facilitates the creation of highly interactive GUI applications.
- However, swing is more flexible and robust when it comes to implementing graphical componen
- One of the main differences between swing and awt is that swing will always generate similar ty of output irrespective of the underlying platform.

- AWT on the other hand it is more depending on the underlying operating system for generating the graphic components; thus output may vary from one platform to another.
- Swings can be regarded as more graphically rich than awt not only because they provide some and telling you graphical components like tabbed window and tree structure.
- But also due to the fact that they have enhance the some of the conventional AWT components like button with both image and text.
- Some of **Key swing classes** are,

a) JApplet:
- An extension of Applet class, it is the swing's version of applet.

b) JFrame:
- An extension of java.awt.Frame class, it is the swing's version of frame.

c) JButton:
- Helps to realize a push button in swing.

d) JTabbedPane:
- Helps to realize a tabbed pane in swing.

e) Jtree:
- Helps to realize a hierarchical (tree) structure in swing.

f) JComboBox:
- Helps to realize a combo box in swing.