Java Programming-IV

By:- Jagdish Chandra Pandey

H.O.D(I.T.), G.P. Dwarahat/Officiating Principal, G.P.Chaunaliya

Applet

- **Applet** is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

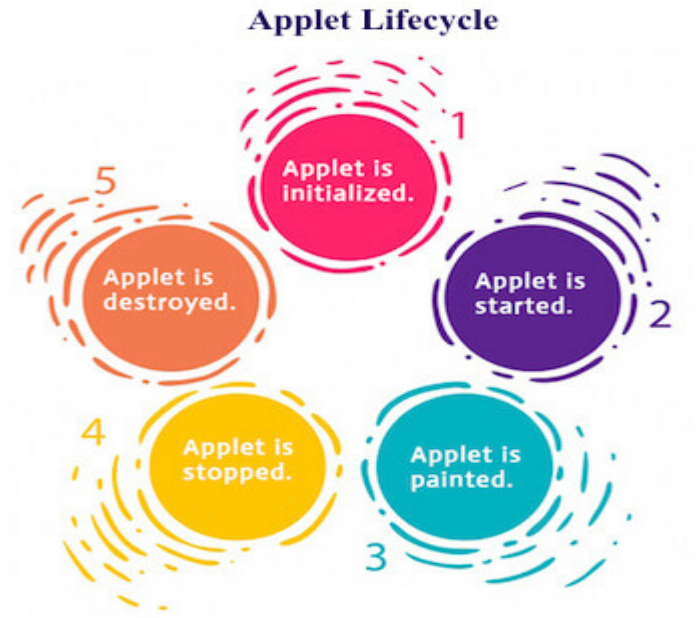
- The user launched the Java applet from a web page, and the applet was then executed within a Java virtual machine (JVM) in a process separate from the web browser itself. A Java applet could appear in a frame of the web page, a new application window , Sun's AppletViewer , or a stand-alone tool for testing applets.

- Java applets run at very fast speeds and until 2011, they were many times faster than JavaScript.

- Unlike JavaScript, Java applets had access to 3D hardware acceleration, making them well-suited for non-trivial, computation-intensive visualizations.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



Lifecycle methods for Applet:

java.applet.Applet class:-

- For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet:-

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class:-

- The Component class provides 1 life cycle method of applet:-

public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Simple example of Applet by appletviewer tool:

- To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome to applet",150,150);
}
  }
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

- To execute the applet by appletviewer tool, write in command prompt:-

```
c:\>javac First.java
c:\>appletviewer First.java
```

HTML <applet> Tag and Parameter tag

- The <applet> tag was used in HTML 4 to define an embedded applet (Plug-in).
- Not Supported in HTML5.

- **Syntax:-**

<applet *attribute1 attribute2....*>
<param *parameter1*>
<param *parameter2*> .... </applet>

- Plug-ins are a computer programs that extend the standard functionality of the browser.

- Plug-ins have been used for many different purposes:-

1. Run Java applets
2. Run ActiveX controls
3. Display Flash movies
4. Display maps
5. Scan for viruses
6. Verify a bank id

- Most browsers no longer support Java Applets and Plug-ins.
- ActiveX controls are no longer supported in any browsers.
- The support for Shockwave Flash has also been turned off in modern browsers.

Simple example of Applet by html file:

- To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome",150,150);
}

}
```

class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

HTML | applet align Attribute

- The **HTML applet align attribute** is used to specify the alignment of the **<applet>** element or the content present inside the applet element.

- **Syntax:-**

<applet align="left | right | center | justify";>

- **Attribute Values:**

1. **left:** It sets the content to the left-align.
2. **right:** It sets the content to the right-align.
3. **center:** I sets the content element to the center.
4. **justify:** It sets the content to the justify position.

Example of HTML | applet align Attribute

```
<!DOCTYPE html>
<html>
<head>
<title> HTML | applet align Attribute </title>
</head>
 <body>
<applet code="HelloWorld" alt="GeeksForGeeKs"
 align="right" width=200 height=60
 name="geeks">
</applet>
  <h1>HTML applet align attribute</h1>
<h2>Hello GeeksForGeeks</h2>
  <p>
 a computer science
 portal for Geeks
</p>
</body>
</html>
```

Output:-

AWT (Abstract Window Toolkit)

- **Java AWT** (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

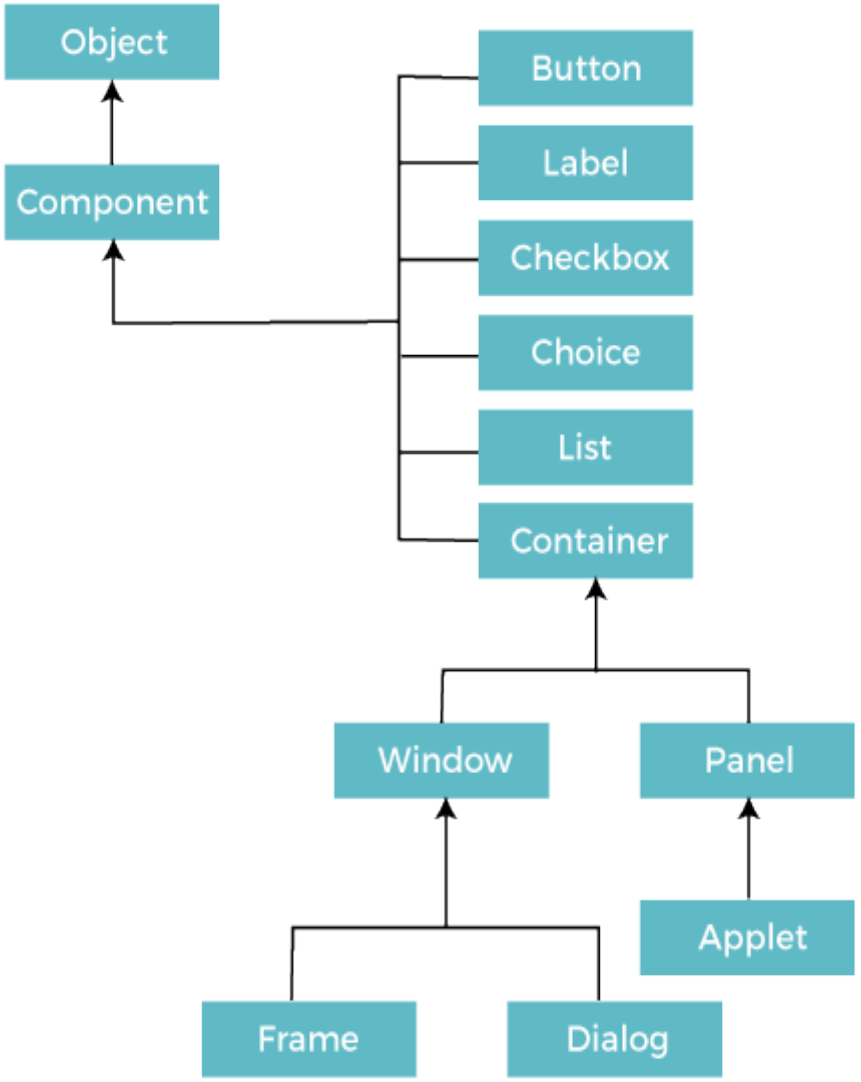
- The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Why AWT is platform independent?

- An AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

- In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy



Java AWT Hierarchy

- Components:-

  All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

- Container:-

1. The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.
2. It is basically a screen where the components are placed at their specific locations. Thus it contains and controls the layout of components.
3. There are four types of containers in Java AWT:-

a) Window
b) Panel
c) Frame
d) Dialog

- To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT:-

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

Types of Container

- Window:-

  The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

- Panel:-

  The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

- Frame:-

  The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Useful Methods of Component Class

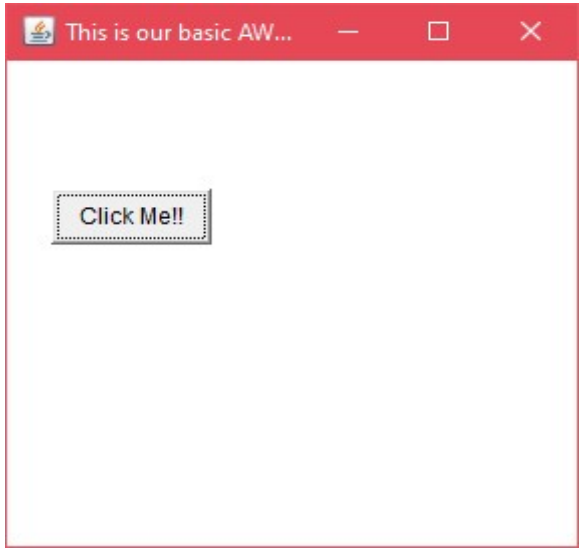
Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

AWT Example by Inheritance

```
import java.awt.*;
class First extends Frame //inheriting Frame class
{
First()
{
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button
      position

add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300
      height
setLayout(null);//no layout now by default
      BorderLayout
setVisible(true);//now frame willbe visible, by
      default not visible
}
public static void main(String args[])
{
First f=new First();
}
}
```

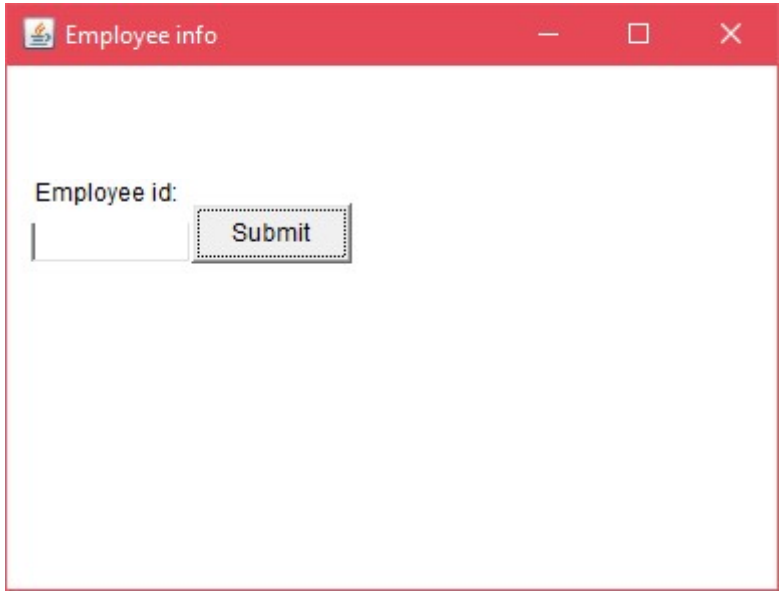
Output:-



AWT Example by Association

```
// importing Java AWT class
import java.awt.*;
// class AWTExample2 directly creates instance of Frame class
class AWTExample2
{
   // initializing using constructor
AWTExample2()
 {
 // creating a Frame
 Frame f = new Frame();
  // creating a Label
  Label l = new Label("Employee id:");
// creating a Button
  Button b = new Button("Submit");
// creating a TextField
 TextField t = new TextField();
 // setting position of above components in the frame
  l.setBounds(20, 80, 80, 30);
 t.setBounds(20, 100, 80, 30);
  b.setBounds(100, 100, 80, 30);
 // adding components into frame
  f.add(b);
  f.add(l);
 f.add(t);
 // frame size 300 width and 300 height
  f.setSize(400,300);
 // setting the title of frame
 f.setTitle("Employee info");
 // no layout
  f.setLayout(null);
// setting visibility of frame
 f.setVisible(true);
}
// main method
public static void main(String args[])
 {
// creating instance of Frame class
AWTExample2 awt_obj = new AWTExample2();
}
}
```

Output:-



Event and Listener (Java Event Handling)

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.

- The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Registration Methods

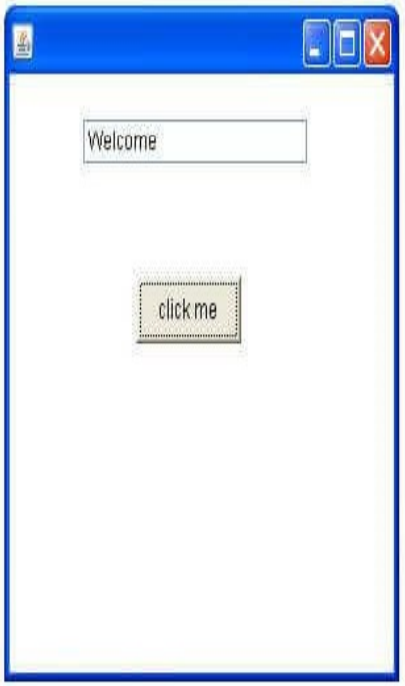
- For registering the component with the Listener, many classes provide the registration methods. For example:-

1. **Button**
   - public void addActionListener(ActionListener a){}
2. **MenuItem**
   - public void addActionListener(ActionListener a){}
3. **TextField**
   - public void addActionListener(ActionListener a){}
   - public void addTextListener(TextListener a){}
4. **TextArea**
   - public void addTextListener(TextListener a){}
5. **Checkbox**
   - public void addItemListener(ItemListener a){}
6. **Choice**
   - public void addItemListener(ItemListener a){}
7. **List**
   - public void addActionListener(ActionListener a){}
   - public void addItemListener(ItemListener a){}

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener
{
TextField tf;
AEvent()
{
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
b.addActionListener(this);//passing current instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

Output:-



Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame
{
TextField tf;
AEvent2()
{
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
```

```
import java.awt.event.*;
class Outer implements ActionListener
{
AEvent2 obj;
Outer(AEvent2 obj)
{
this.obj=obj;
}
public void actionPerformed(ActionEvent e)
{
obj.tf.setText("welcome");
}
}
```

Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame
{
TextField tf;
AEvent3()
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);
 b.addActionListener(new ActionListener()
{
public void actionPerformed()
{
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[])
{
new AEvent3();
}
}
```

Java AWT Button

- A button is basically a control component with a label that generates an event when pushed. 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.

- When we press a button and release it, AWT sends an instance of **ActionEvent** to that button by calling **processEvent** on the button. The **processEvent** method of the button receives the all the events, then it passes an action event by calling its own method **processActionEvent**. This method passes the action event on to action listeners that are interested in the action events generated by the button.

- To perform an action on a button being pressed and released, the **ActionListener** interface needs to be implemented. The registered new listener can receive events from the button by calling **addActionListener** method of the button.

- AWT Button Class Declaration:-

  public class Button **extends** Component **implements** Accessible

Button Class Constructors

Sr. no.	Constructor	Description
1.	Button( )	It constructs a new button with an empty string i.e. it has no label.
2.	Button (String text)	It constructs a new button with given string as its label.

Button Class Methods

Sr. no.	Method	Description
1.	void setText (String text)	It sets the string message on the button
2.	String getText()	It fetches the String message on the button.
3.	void setLabel (String label)	It sets the label of button with the specified string.
4.	String getLabel()	It fetches the label of the button.
5.	void addNotify()	It creates the peer of the button.
6.	AccessibleContext getAccessibleContext()	It fetched the accessible context associated with the button.
7.	void addActionListener(ActionListener l)	It adds the specified action listener to get the action events from the button.
8.	String getActionCommand()	It returns the command name of the action event fired by the button.
9.	ActionListener[ ] getActionListeners()	It returns an array of all the action listeners registered on the button.
10.	T[ ] getListeners(Class listenerType)	It returns an array of all the objects currently registered as FooListeners upon this Button.
11.	protected String paramString()	It returns the string which represents the state of button.
12.	protected void processActionEvent (ActionEvent e)	It process the action events on the button by dispatching them to a registered ActionListener object.
13.	protected void processEvent (AWTEvent e)	It process the events on the button
14.	void removeActionListener (ActionListener l)	It removes the specified action listener so that it no longer receives action events from the button.
15.	void setActionCommand(String command)	It sets the command name for the action event given by the button.

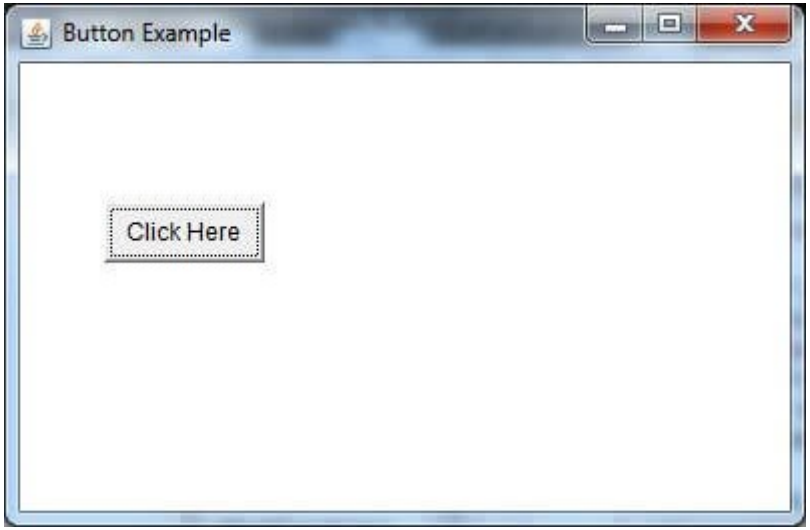
Java AWT Button Example

ButtonExample.java

```
import java.awt.*;
public class ButtonExample {
public static void main (String[] args) {
// create instance of frame with the label
    Frame f = new Frame("Button Example");

    // create instance of button with label
Button b = new Button("Click Here");
// set the position for the button in frame
    b.setBounds(50,100,80,30);
// add button to the frame
    f.add(b);
    // set size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:-



Java AWT Label

- The object of the 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 a programmer but a user cannot edit it directly.

- To create a label, we need to create the object of **Label** class.

public class Label **extends** Component **implements** Accessible

- The java.awt.Component class has following fields:-

1. **static int LEFT:** It specifies that the label should be left justified.

2. **static int RIGHT:** It specifies that the label should be right justified.

3. **static int CENTER:** It specifies that the label should be placed in center.

Label class Constructors

Sr. no.	Constructor	Description
1.	Label()	It constructs an empty label.
2.	Label(String text)	It constructs a label with the given string (left justified by default).
3.	Label(String text, int alignement)	It constructs a label with the specified string and the specified alignment.

Label Class Methods

Sr. no.	Method name	Description
1.	void setText(String text)	It sets the texts for label with the specified text.
2.	void setAlignment(int alignment)	It sets the alignment for label with the specified alignment.
3.	String getText()	It gets the text of the label
4.	int getAlignment()	It gets the current alignment of the label.
5.	void addNotify()	It creates the peer for the label.
6.	AccessibleContext getAccessibleContext()	It gets the Accessible Context associated with the label.
7.	protected String paramString()	It returns the string the state of the label.

Java AWT Label Example

```
import java.awt.*;
public class LabelExample
{
public static void main(String args[])
{
    // creating the object of Frame class and Label
       class
   Frame f = new Frame ("Label example");
   Label l1, l2;
    // initializing the labels
   l1 = new Label ("First Label.");
   l2 = new Label ("Second Label.");
    // set the location of label
   l1.setBounds(50, 100, 100, 30);
   l2.setBounds(50, 150, 100, 30);
// adding labels to the frame
   f.add(l1);
f.add(l2);
    // setting size, layout and visibility of frame
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Output:-

Java AWT TextField

- The object of a **TextField** class is a text component that allows a user to enter a single line text and edit it. It inherits **TextComponent** class, which further inherits **Component** class.

- When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to **TextField**. Then the **KeyEvent** is passed to the registered **KeyListener**. It can also be done using **ActionEvent**; if the ActionEvent is enabled on the text field, then the ActionEvent may be fired by pressing return key. The event is handled by the **ActionListener** interface.

- **public class** TextField **extends** TextComponent

TextField Class constructors

Sr. no.	Constructor	Description
1.	TextField()	It constructs a new text field component.
2.	TextField(String text)	It constructs a new text field initialized with the given string text to be displayed.
3.	TextField(int columns)	It constructs a new textfield (empty) with given number of columns.
4.	TextField(String text, int columns)	It constructs a new text field with the given text and given number of columns (width).

TextField Class Methods

Sr. no.	Method name	Description
1.	void addNotify()	It creates the peer of text field.
2.	boolean echoCharIsSet()	It tells whether text field has character set for echoing or not.
3.	void addActionListener(ActionListener l)	It adds the specified action listener to receive action events from the text field.
4.	ActionListener[] getActionListeners()	It returns array of all action listeners registered on text field.
5.	AccessibleContext getAccessibleContext()	It fetches the accessible context related to the text field.
6.	int getColumns()	It fetches the number of columns in text field.
7.	char getEchoChar()	It fetches the character that is used for echoing.
8.	Dimension getMinimumSize()	It fetches the minimum dimensions for the text field.
9.	Dimension getMinimumSize(int columns)	It fetches the minimum dimensions for the text field with specified number of columns.
10.	Dimension getPreferredSize()	It fetches the preferred size of the text field.

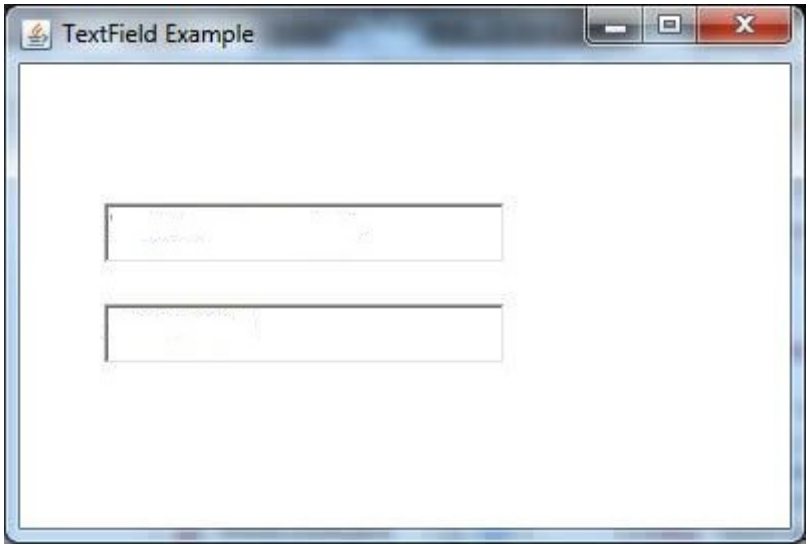
TextField Class Methods

Sr. no.	Method name	Description
11.	Dimension getPreferredSize(int columns)	It fetches the preferred size of the text field with specified number of columns.
12.	protected String paramString()	It returns a string representing state of the text field.
13.	protected void processActionEvent(ActionEvent e)	It processes action events occurring in the text field by dispatching them to a registered ActionListener object.
14.	protected void processEvent(AWTEvent e)	It processes the event on text field.
15.	void removeActionListener(ActionListener l)	It removes specified action listener so that it doesn't receive action events anymore.
16.	void setColumns(int columns)	It sets the number of columns in text field.
17.	void setEchoChar(char c)	It sets the echo character for text field.
18.	void setText(String t)	It sets the text presented by this text component to the specified text.

Java AWT TextField Example

```
// importing AWT class
import java.awt.*;
public class TextFieldExample1
 {
    // main method
    public static void main(String args[])
{
    // creating a frame
    Frame f = new Frame("TextField Example");
 // creating objects of textfield
    TextField t1, t2;
    // instantiating the textfield objects
    // setting the location of those objects in the fram
       e
    t1 = new TextField("Welcome to Javatpoint.");
    t1.setBounds(50, 100, 200, 30);
    t2 = new TextField("AWT Tutorial");
    t2.setBounds(50, 150, 200, 30);
    // adding the components to frame
    f.add(t1);
    f.add(t2);
    // setting size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:-



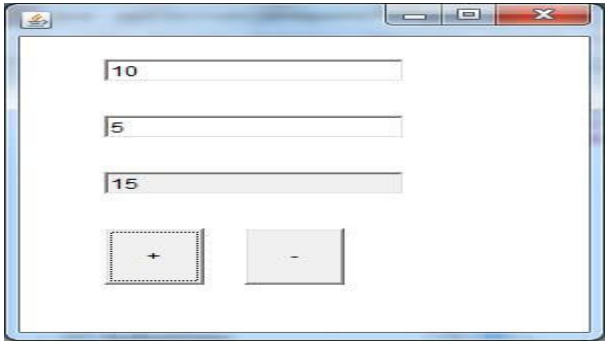
Java AWT TextField Example with ActionListener

```
// importing necessary libraries
import java.awt.*;
import java.awt.event.*;
// Our class extends Frame class and implements ActionListener interface
public class TextFieldExample2 extends Frame implements ActionListener
{
   // creating instances of TextField and Button class
   TextField tf1, tf2, tf3;
   Button b1, b2;
   // instantiating using constructor
   TextFieldExample2()
{
      // instantiating objects of text field and button
      // setting position of components in frame
      tf1 = new TextField();
      tf1.setBounds(50, 50, 150, 20);
      tf2 = new TextField();
      tf2.setBounds(50, 100, 150, 20);
      tf3 = new TextField();
      tf3.setBounds(50, 150, 150, 20);
      tf3.setEditable(false);
      b1 = new Button("+");
      b1.setBounds(50, 200, 50, 50);
      b2 = new Button("-");
      b2.setBounds(120,200,50,50);
      // adding action listener
      b1.addActionListener(this);
      b2.addActionListener(this);
      // adding components to frame
      add(tf1);
      add(tf2);
      add(tf3);
      add(b1);
      add(b2);
      // setting size, layout and visibility of frame
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
```

```
 // defining the actionPerformed method to generate an event on buttons
   public void actionPerformed(ActionEvent e)
{
      String s1 = tf1.getText();
 String s2 = tf2.getText();
      int a = Integer.parseInt(s1);
      int b = Integer.parseInt(s2);
      int c = 0;
      if (e.getSource() == b1)
{
         c = a + b;
      }
      else if (e.getSource() == b2)
{

         c = a - b;
      }
      String result = String.valueOf(c);
      tf3.setText(result);
   }
// main method
public static void main(String[] args)
{
   new TextFieldExample2();
}
}
```

- Output:-



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".

- AWT Checkbox Class Declaration:-

public class Checkbox **extends** Component **implements** ItemSelectable, Accessible

Checkbox Class Constructors

Sr. no.	Constructor	Description
1.	Checkbox()	It constructs a checkbox with no string as the label.
2.	Checkbox(String label)	It constructs a checkbox with the given label.
3.	Checkbox(String label, boolean state)	It constructs a checkbox with the given label and sets the given state.
4.	Checkbox(String label, boolean state, CheckboxGroup group)	It constructs a checkbox with the given label, set the given state in the specified checkbox group.
5.	Checkbox(String label, CheckboxGroup group, boolean state)	It constructs a checkbox with the given label, in the given checkbox group and set to the specified state.

Checkbox Class Methods

Sr. no.	Method name	Description
1.	void addItemListener(ItemListener IL)	It adds the given item listener to get the item events from the checkbox.
2.	AccessibleContext getAccessibleContext()	It fetches the accessible context of checkbox.
3.	void addNotify()	It creates the peer of checkbox.
4.	CheckboxGroup getCheckboxGroup()	It determines the group of checkbox.
5.	ItemListener[] getItemListeners()	It returns an array of the item listeners registered on checkbox.
6.	String getLabel()	It fetched the label of checkbox.
7.	T[] getListeners(Class listenerType)	It returns an array of all the objects registered as FooListeners.
8.	Object[] getSelectedObjects()	It returns an array (size 1) containing checkbox label and returns null if checkbox is not selected.

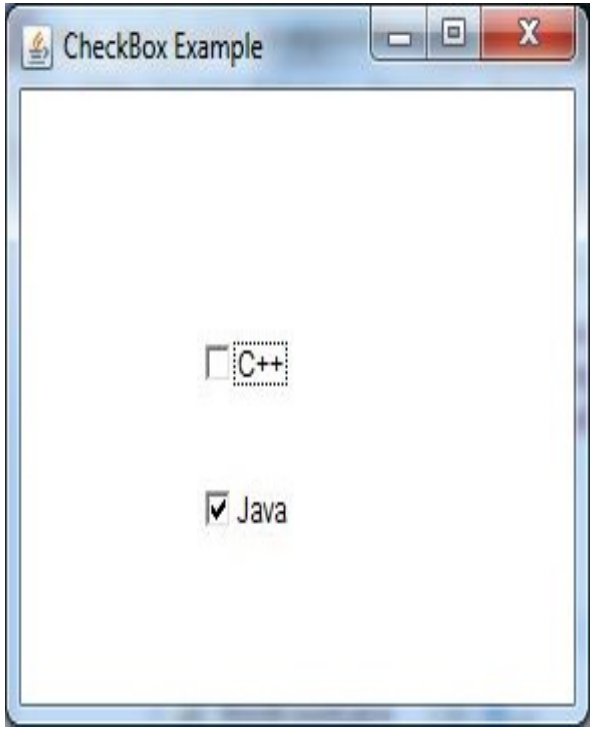
Checkbox Class Methods

Sr. no.	Method name	Description
9.	boolean getState()	It returns true if the checkbox is on, else returns off.
10.	protected String paramString()	It returns a string representing the state of checkbox.
11.	protected void processEvent(AWTEvent e)	It processes the event on checkbox.
12.	protected void processItemEvent(ItemEvent e)	It process the item events occurring in the checkbox by dispatching them to registered ItemListener object.
13.	void removeItemListener(ItemListener l)	It removes the specified item listener so that the item listener doesn't receive item events from the checkbox anymore.
14.	void setCheckboxGroup(CheckboxGroup g)	It sets the checkbox's group to the given checkbox.
15.	void setLabel(String label)	It sets the checkbox's label to the string argument.
16.	void setState(boolean state)	It sets the state of checkbox to the specified state.

Java AWT Checkbox Example

```
// importing AWT class
import java.awt.*;
public class CheckboxExample1
{
// constructor to initialize
    CheckboxExample1()
{
// creating the frame with the title
      Frame f = new Frame("Checkbox Example");
// creating the checkboxes
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100, 100,  50, 50);
       Checkbox checkbox2 = new Checkbox("Java", true);
// setting location of checkbox in frame
checkbox2.setBounds(100, 150,  50, 50);
// adding checkboxes to frame
       f.add(checkbox1);
       f.add(checkbox2);
// setting size, layout and visibility of frame
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
// main method
public static void main (String args[])
{
   new CheckboxExample1();
}
}
```

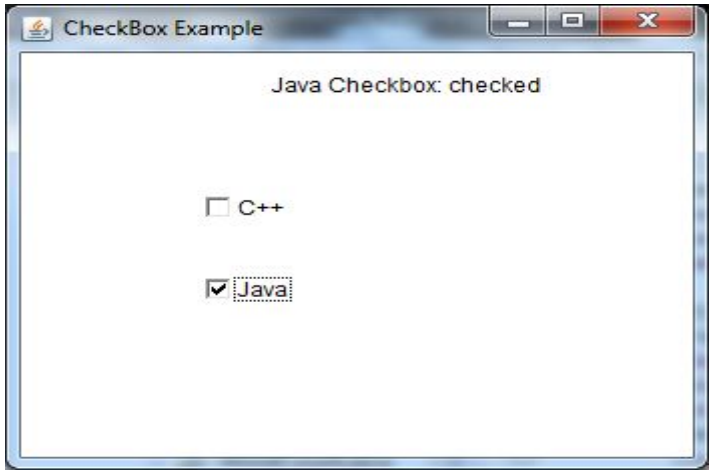
Output:-



Java AWT Checkbox Example with ItemListener

```
// importing necessary packages
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample2
{
// constructor to initialize
    CheckboxExample2()
{
// creating the frame
      Frame f = new Frame ("CheckBox Example");
// creating the label
      final Label label = new Label();
// setting the alignment, size of label
  label.setAlignment(Label.CENTER);
      label.setSize(400,100);
// creating the checkboxes
      Checkbox checkbox1 = new Checkbox("C++");
      checkbox1.setBounds(100, 100,  50, 50);
      Checkbox checkbox2 = new Checkbox("Java");
      checkbox2.setBounds(100, 150,  50, 50);
// adding the checkbox to frame
f.add(checkbox1);
f.add(checkbox2);
f.add(label);
// adding event to the checkboxes
      checkbox1.addItemListener(new ItemListener()
{
          public void itemStateChanged(ItemEvent e)
{
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
        });
checkbox2.addItemListener(new ItemListener()
{
          public void itemStateChanged(ItemEvent e)
 {
label.setText("Java Checkbox: " + (e.getStateChange()==1?"checked":"unchecked")
          );
            }
          });
// setting size, layout and visibility of frame
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
// main method
public static void main(String args[])
{
   new CheckboxExample2();
}
}
```

- Output:-



Java AWT 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.
- AWT Choice Class Declaration:-

public class Choice **extends** Component **implements** ItemSelectable, Accessible

- Choice Class constructor:-

Sr. no.	Constructor	Description
1.	Choice()	It constructs a new choice menu.

Choice Class Methods

Sr. no.	Method name	Description
1.	void add(String item)	It adds an item to the choice menu.
2.	void addItemListener(ItemListener l)	It adds the item listener that receives item events from the choice menu.
3.	void addNotify()	It creates the peer of choice.
4.	AccessibleContext getAccessibleContext()	It gets the accessbile context related to the choice.
5.	String getItem(int index)	It gets the item (string) at the given index position in the choice menu.
6.	int getItemCount()	It returns the number of items of the choice menu.
7.	ItemListener[] getItemListeners()	It returns an array of all item listeners registered on choice.
8.	T[] getListeners(Class listenerType)	Returns an array of all the objects currently registered as FooListeners upon this Choice.
9.	int getSelectedIndex()	Returns the index of the currently selected item.
10.	String getSelectedItem()	Gets a representation of the current choice as a string.

Choice Class Methods

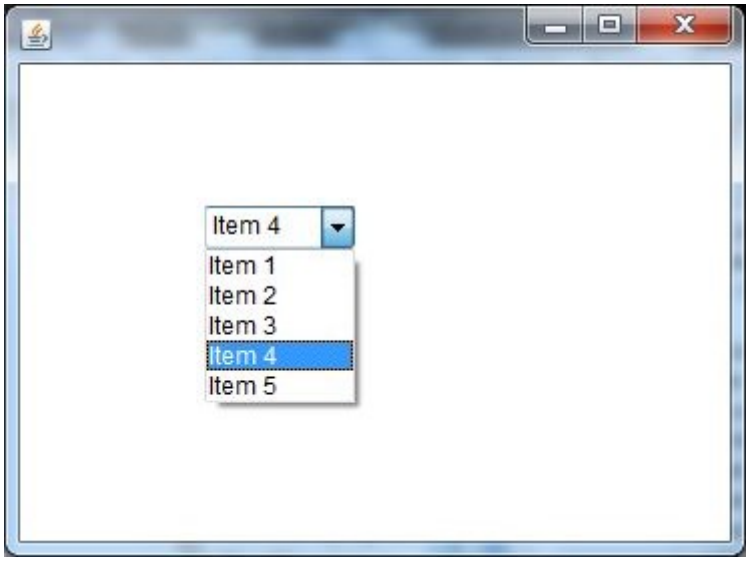
Sr. no.	Method name	Description
11.	Object[] getSelectedObjects()	Returns an array (length 1) containing the currently selected item.
12.	void insert(String item, int index)	Inserts the item into this choice at the specified position.
13.	protected String paramString()	Returns a string representing the state of this Choice menu.
14.	protected void processEvent(AWTEvent e)	It processes the event on the choice.
15.	protected void processItemEvent (ItemEvent e)	Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
16.	void remove(int position)	It removes an item from the choice menu at the given index position.
17.	void remove(String item)	It removes the first occurrence of the item from choice menu.
18.	void removeAll()	It removes all the items from the choice menu.
19.	void removeItemListener (ItemListener l)	It removes the mentioned item listener. Thus is doesn't receive item events from the choice menu anymore.
20.	void select(int pos)	It changes / sets the selected item in the choice menu to the item at given index position.
21.	void select(String str)	It changes / sets the selected item in the choice menu to the item whose string value is equal to string specified in the argument.

Java AWT Choice Example

```
// importing awt class
import java.awt.*;
public class ChoiceExample1
 {
 // class constructor
ChoiceExample1()
{
      // creating a frame
      Frame f = new Frame();
// creating a choice component
Choice c = new Choice();
// setting the bounds of choice menu
 c.setBounds(100, 100, 75, 75);
// adding items to the choice menu
      c.add("Item 1");
      c.add("Item 2");
      c.add("Item 3");
      c.add("Item 4");
      c.add("Item 5");
// adding choice menu to frame
      f.add(c);
// setting size, layout and visibility of frame
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);
    }

// main method
public static void main(String args[])
{
  new ChoiceExample1();
}
}
```

Output:-



Java AWT Choice Example with ActionListener

```
// importing necessary packages
import java.awt.*;
import java.awt.event.*;
public class ChoiceExample2
{

    // class constructor
     ChoiceExample2()
{

      // creating a frame
      Frame f = new Frame();
// creating a final object of Label class
      final Label label = new Label();
       // setting alignment and size of label component
      label.setAlignment(Label.CENTER);
      label.setSize(400, 100);
// creating a button
      Button b = new Button("Show");
// setting the bounds of button
      b.setBounds(200, 100, 50, 20);
// creating final object of Choice class
      final Choice c = new Choice();
// setting bounds of choice menu
      c.setBounds(100, 100, 75, 75);
// adding 5 items to choice menu
      c.add("C");
      c.add("C++");
      c.add("Java");
      c.add("PHP");
      c.add("Android");
// adding above components into the frame
      f.add(c);
      f.add(label);
      f.add(b);
// setting size, layout and visibility of frame
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);
// adding event to the button
 // which displays the selected item from the list when button is clicked
  b.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
 {
  String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());
label.setText(data);
}
});
}
// main method
public static void main(String args[])
{
  new ChoiceExample2();
}
}
```

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.

- It can be added to top-level container like Frame or a component like Panel. The Scrollbar class extends the **Component** class.

- AWT Scrollbar Class Declaration:-

public class Scrollbar **extends** Component **implements** Adjustable, Accessible

Scrollbar Class Constructors

Sr. no.	Constructor	Description
1	Scrollbar()	Constructs a new vertical scroll bar.
2	Scrollbar(int orientation)	Constructs a new scroll bar with the specified orientation.
3	Scrollbar(int orientation, int value, int visible, int minimum, int maximum)	Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Where the parameters:-

1. **orientation**: specifiey whether the scrollbar will be horizontal or vertical.
2. **Value**: specify the starting position of the knob of Scrollbar on its track.
3. **Minimum**: specify the minimum width of track on which scrollbar is moving.
4. **Maximum**: specify the maximum width of track on which scrollbar is moving.

Scrollbar Class Methods

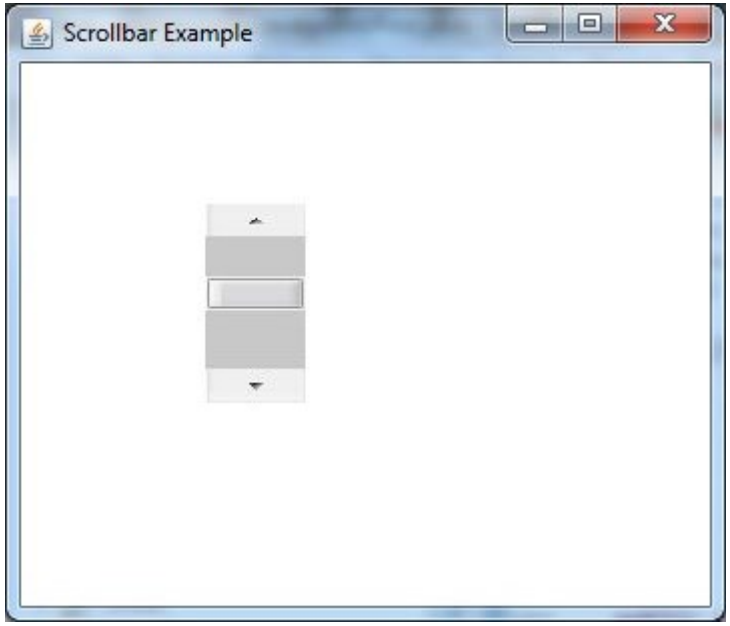
Sr. no.	Method name	Description
1.	void addAdjustmentListener (AdjustmentListener l)	It adds the given adjustment listener to receive instances of AdjustmentEvent from the scroll bar.
2.	void addNotify()	It creates the peer of scroll bar.
3.	int getBlockIncrement()	It gets the block increment of the scroll bar.
4.	int getMaximum()	It gets the maximum value of the scroll bar.
5.	int getMinimum()	It gets the minimum value of the scroll bar.
6.	int getOrientation()	It returns the orientation of scroll bar.
7.	int getUnitIncrement()	It fetches the unit increment of the scroll bar.
8.	int getValue()	It fetches the current value of scroll bar.
9.	int getVisibleAmount()	It fetches the visible amount of scroll bar.
10.	boolean getValueIsAdjusting()	It returns true if the value is in process of changing where action results are being taken by the user.

Java AWT Scrollbar Example

```
// importing awt package
import java.awt.*;
public class ScrollbarExample1
{
// class constructor
ScrollbarExample1()
{
      // creating a frame
          Frame f = new Frame("Scrollbar Example");
      // creating a scroll bar
          Scrollbar s = new Scrollbar();

      // setting the position of scroll bar
          s.setBounds (100, 100, 50, 100);
      // adding scroll bar to the frame
          f.add(s);
      // setting size, layout and visibility of frame
          f.setSize(400, 400);
          f.setLayout(null);
          f.setVisible(true);
}
// main method
public static void main(String args[]) {
      new ScrollbarExample1();
}
}
```

Output:-

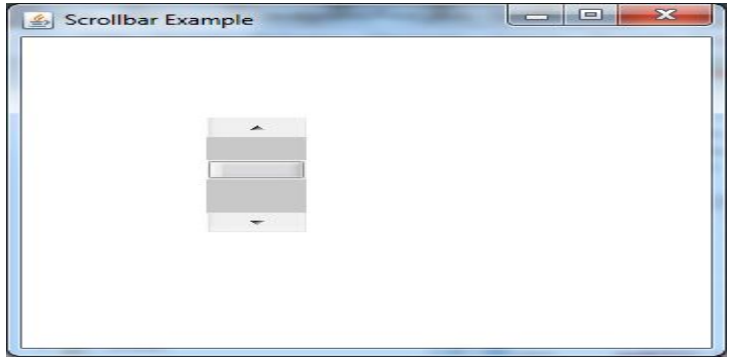


Java AWT Scrollbar Example with AdjustmentListener

```
// importing necessary packages
import java.awt.*;
import java.awt.event.*;
public class ScrollbarExample2
 {
     // class constructor
    ScrollbarExample2()
 {
     // creating a Frame with a title
         Frame f = new Frame("Scrollbar Example");

         // creating a final object of Label
         final Label label = new Label();
         // setting alignment and size of label object
         label.setAlignment(Label.CENTER);
         label.setSize(400, 100);
         // creating a final object of Scrollbar class
         final Scrollbar s = new Scrollbar();
// setting the position of scroll bar
         s.setBounds(100, 100, 50, 100);
         // adding Scrollbar and Label to the Frame
         f.add(s);
     f.add(label);
// setting the size, layout, and visibility of frame
         f.setSize(400, 400);
         f.setLayout(null);
         f.setVisible(true);
 // adding AdjustmentListener to the scrollbar object
  s.addAdjustmentListener(new AdjustmentListener()
{
public void adjustmentValueChanged(AdjustmentEvent e)
 {
    label.setText("Vertical Scrollbar value is:"+ s.getValue());
            }
         });
       }
// main method
public static void main(String args[])
{
new ScrollbarExample2();
}
}
```

Output:-



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.

- AWT MenuItem class declaration:-

public class MenuItem **extends** MenuComponent **implements** Accessible

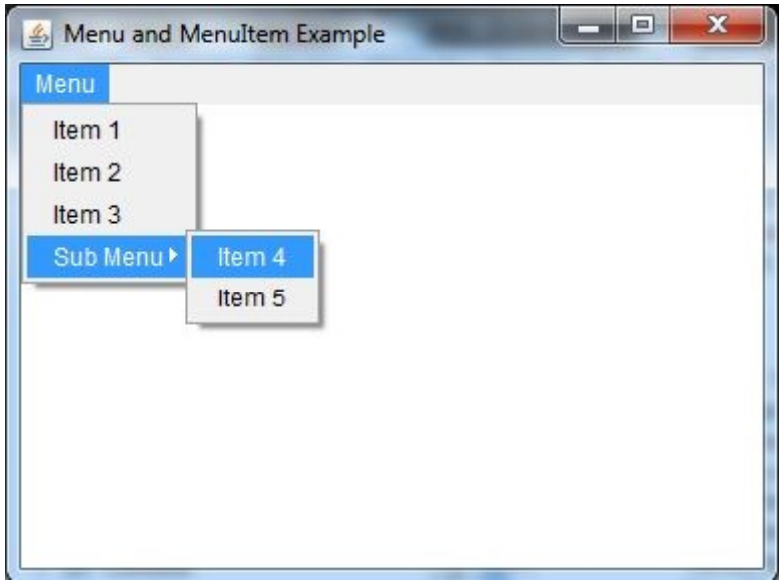
- AWT Menu class :-

declaration**public class** Menu **extends** MenuItem **implements** MenuContainer, Accessible

Java AWT MenuItem and Menu 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:-



Java AWT Toolkit

- Toolkit class is the abstract superclass of every implementation in the Abstract Window Toolkit. Subclasses of Toolkit are used to bind various components. It inherits Object class.

- AWT Toolkit class declaration:-

public abstract class Toolkit **extends** Object

Java AWT Toolkit Example

```
import java.awt.*;
public class ToolkitExample
{
  public static void main(String[] args)
{
    Toolkit t = Toolkit.getDefaultToolkit();
 System.out.println("Screen resolution = " + t.getScreenResolution());
      Dimension d = t.getScreenSize();
 System.out.println("Screen width = " + d.width);
System.out.println("Screen height = " + d.height);
  }
}
```

Output:-

```
Screen resolution = 96
 Screen width = 1366
Screen height = 768
```

Java LayoutManagers

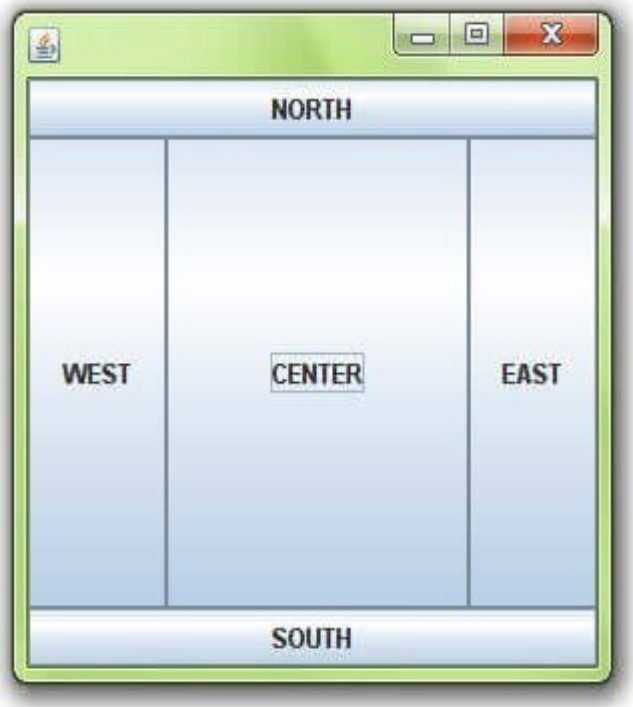
- The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:-

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Example of BorderLayout class: Using BorderLayout() constructor

```
import java.awt.*;
import javax.swing.*;
public class Border
 {
JFrame f;
Border()
{
      f=new JFrame();
      JButton b1=new JButton("NORTH");
      JButton b2=new JButton("SOUTH");
      JButton b3=new JButton("EAST");
      JButton b4=new JButton("WEST");
      JButton b5=new JButton("CENTER");
      f.add(b1,BorderLayout.NORTH);
      f.add(b2,BorderLayout.SOUTH);
      f.add(b3,BorderLayout.EAST);
      f.add(b4,BorderLayout.WEST);
      f.add(b5,BorderLayout.CENTER);
      f.setSize(300,300);
      f.setVisible(true);
}
public static void main(String[] args)
{
      new Border();
}
}
```

Output:-




"THANK
YOU"