Event Handling

An event is one of the most important concepts in Java.

Event is nothing but Object, it performs a special task.
The change in the state of an object or behavior by performing actions is referred to as an Event in Java.
Example of events: Clicking a button,
Clicking a mouse and Dragging a mouse

A mechanism for controlling the events and deciding what should happen after an event occur is referred to as **event handling**. Java follows the Delegation Event Model for handling the events.

Delegation Event Model

It consists of two components:
Events Sources
Events Listeners

Event Sources

A source is an object that causes and generates events and sends to the Listener. The sources are allowed to generate several different types of events.
A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

public void addTypeListener (TypeListener e1)

From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener. For example, for a keyboard event listener, the method will be called as addKeyListener().

Event Listeners

An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events. Second, it must implement the methods to receive and process the received notifications.

Types of Events and Listeners

S.No.	Event Class	Listener Interface	Methods	Descriptions
1.	ActionEvent	ActionListener	actionPerformed()	ActionEvent will be occurred when we click a Button.
2.	AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()	Adjustment events occur by an adjustable object like a scrollbar.
3.	ComponentEvent	ComponentListener	componentResized(), componentMoved(), componentShown() and componentHidden()	An event occurs when a component moved, changed its visibility or the size changed.
4.	ItemEvent	ItemListener	itemStateChanged()	Item event occurs when an item is selected.
5.	KeyEvent	KeyListener	keyPressed(), keyReleased(), and keyTyped().	A key event occurs when the user presses a key on the keyboard.

6.	MouseEvent	MouseListener and MouseMotionListener	mouseClicked(), mousePressed(), mouseEntered(), mouseExited() and mouseReleased() are the mouseListener methods. mouseDregged() and mouseMoved() are the MouseMotionListener() methods.	A mouse event occurs when the user interacts with the mouse.
7.	WindowEvent	WindowListener	windowActivated(), windowDeactivated(),	Window events occur when a

			windowOpened(), windowClosed(), windowClosing(), windowIconfied() and windowDeiconified().	window's status is changed.

Handling Mouse Events

To handle mouse events, we must implement the MouseListener and the MouseMotionListener interfaces and provide the implementations of the below events.

MOUSE_CLICKED The user clicked the mouse.
MOUSE_DRAGGED The user dragged the mouse.
MOUSE_ENTERED The mouse entered a component.
MOUSE_EXITED The mouse exited from a component.
MOUSE_MOVED The mouse moved.
MOUSE_PRESSED The mouse was pressed.
MOUSE_RELEASED The mouse was released.
Program for handling mouse events :

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

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

/*<applet code="MouseDemo" width=900 height=900>
</applet>*/
public class MouseDemo extends Applet implements MouseListener, MouseMotionListener {
 int mx = 0;
 int my = 0;
 String msg = "";
 public void init() {
  addMouseListener(this);
  addMouseMotionListener(this);
 }

 public void mouseClicked(MouseEvent me) {
  mx = 20;
  my = 40;
  msg = "Mouse Clicked";
  repaint();
 }

 public void mousePressed(MouseEvent me) {
  mx = 30;
  my = 60;
  msg = "Mouse Pressed";
  repaint();
 }

 public void mouseReleased(MouseEvent me) {
  mx = 30;
  my = 60;
  msg = "Mouse Released";
  repaint();
 }
```

```
 public void mouseEntered(MouseEvent me) {
  mx = 40;
  my = 80;
  msg = "Mouse Entered";
  repaint();
 }

 public void mouseExited(MouseEvent me) {
  mx = 40;
  my = 80;
  msg = "Mouse Exited";
  repaint();
 }

 public void mouseDragged(MouseEvent me) {
  mx = me.getX();
  my = me.getY();
  showStatus("Currently mouse dragged " + mx + " " + my);
repaint();
}

public void mouseMoved(MouseEvent me) {
mx = me.getX();
my = me.getY();
showStatus("Currently mouse is moving at " + mx + " " + my);
repaint();
}
public void paint(Graphics g) {
```

```
g.drawString(msg, 60, 40);
}
}
```

Handling Keyboard Events

To handle keyboard events we must implements KeyListener interface When a key is pressed, a KEY_PRESSED event is generated and the keyPressed( ) handler is executed. When the key is released, a KEY_RELEASED event is generated and the keyReleased( ) handler is executed. If a character is generated by the keystroke, then a KEY_TYPED event is sent and the keyTyped( ) handler is invoked.

Program for handling KeyEvents

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

/*<applet code="KeyDemo" width=300 height=300>
</applet>*/
public class KeyDemo extends Applet implements KeyListener {
 int mx = 40;
 int my = 60;
 String msg = "";

 public void init() {
  addKeyListener(this);
 }
 public void keyPressed(KeyEvent me) {
```

```
 msg = "key Pressed";
 repaint();
}

public void keyReleased(KeyEvent me) {
 msg = "key Released";
 repaint();
}

public void keyTyped(KeyEvent me) {
 msg = "key Typed";
 repaint();
}


public void paint(Graphics g) {
g.drawString(msg, mx,my);
}
}
```

Adapter Classes

Java provides a special feature, called an adapter class, that can simplify the creation of event handlers in certain situations.
An adapter class provides an empty implementation of all methods in an event listener interface.
Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

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

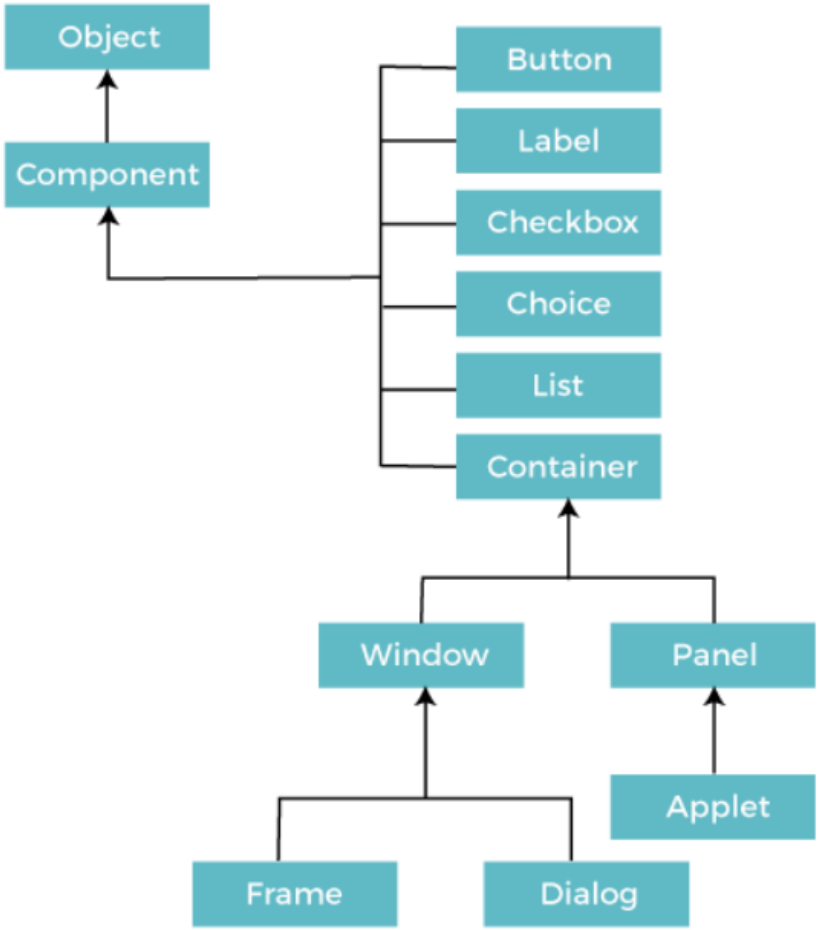
```
import java.awt.event.*;
import java.applet.*;
/*<applet code="AdapterDemo" width=300 height=100>
</applet>*/

public class AdapterDemo extends Applet {
public void init() {
addKeyListener(new MyKeyAdapter(this));
}
}
class MyKeyAdapter extends KeyAdapter {
AdapterDemo a1;
public MyKeyAdapter(AdapterDemo a) {
a1=a;
}

public void keyReleaed(KeyEvent me) {
a1.showStatus("Key Released");
}
}
```

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 AWT class hierarchy

The hierarchy of Java AWT classes are given below.



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.

Container

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**.

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.

user interface components

Label component

- It is a component in AWT.
- 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.

Label can be created using **Label** class, existed in java.awt package
Constructors

1. Label() It creates an empty label.

2. Label(String text) It creates a label with the given string

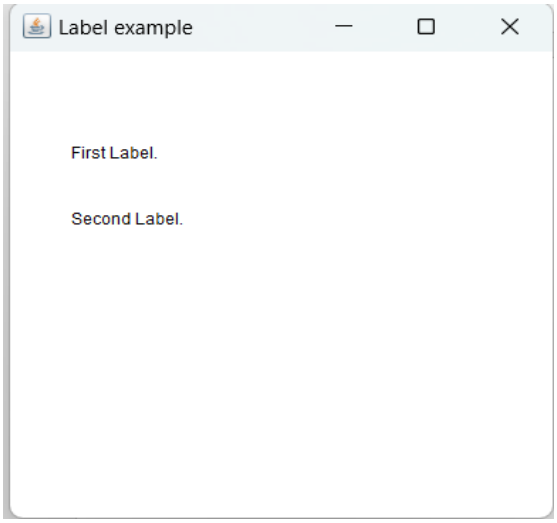
3. Label(String text, int alignement) It creates a label with the specified string and the specified alignment (LEFT,RIGHT and CENTER)

Example:

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

```
    f.setVisible(true);
}
}
```

Output:



Button component

- It is a component in AWT.
- When we press a button ActionEvent will be generated.
- Button can be created using Button class, existed in java.awt package.

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.

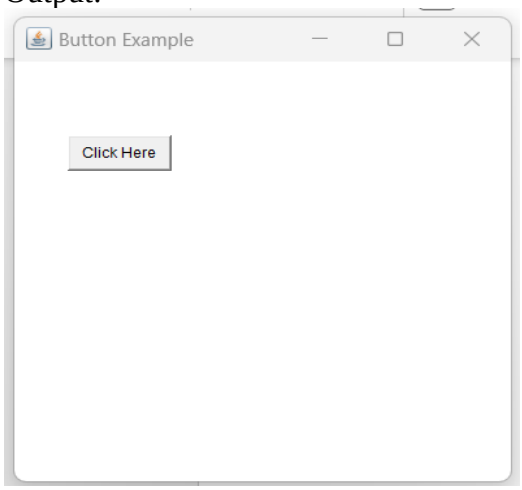
Constructors

1. B**utton ():** 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.

Ex:

```
import java.awt.*;
public class LabelDemo{
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:



TextField component

- **It is a component in AWT.**
- **TextField can be created using TextField class, existed in java.awt package.**
- **It allows a user to enter a single line text and edit it.**

Constructors

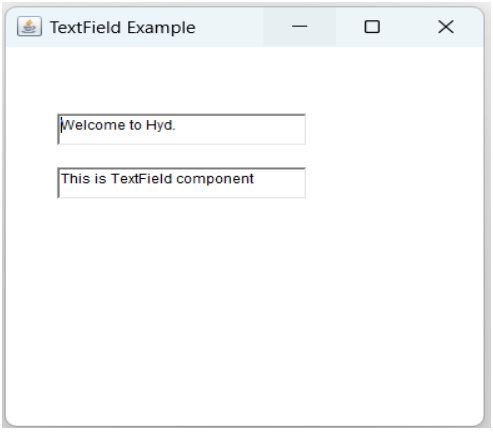
1.TextField(): It created a new text field component.

2.TextField(String text): It creates a new text field initialized with the given string text to be displayed.

Ex:

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

Output:



TextArea Component

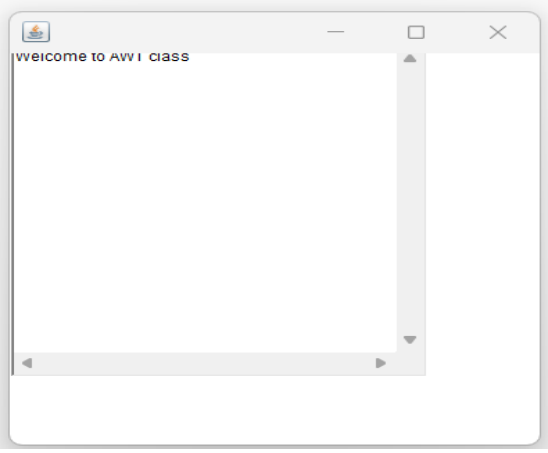
- It is a component in AWT.
- TextArea can be created using TextArea class, existed in java.awt package.
- It allows to enter multiple lines of text in the TextArea.

When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.

Constructors

1.TextArea(): It constructs a new and empty text area with no text in it.
2.TextArea (int row, int column): It constructs a new text area with specified number of rows and columns and empty string as text.

```
public class TextAreaDemo
{
    TextAreaDemo()
  {
       Frame f = new Frame();
       TextArea area = new TextArea("Welcome to Java AWT class");
       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 TextAreaDemo();
}
}
```



Checkbox Component

- It is a component in AWT
- 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".

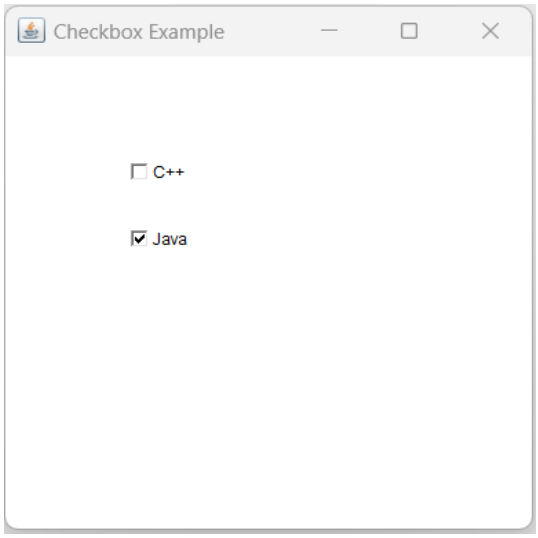
Constructors

- Checkbox() : It constructs a checkbox with no string as the label.
- Checkbox(String label): It constructs a checkbox with the given label.
- Checkbox(String label, boolean state) :
  It constructs a checkbox with the given label and sets the given state.

Ex:

```
import java.awt.*;
public class CheckboxDemo {
        public static void main(String args[]) {
                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);
        }
}
```

Output:



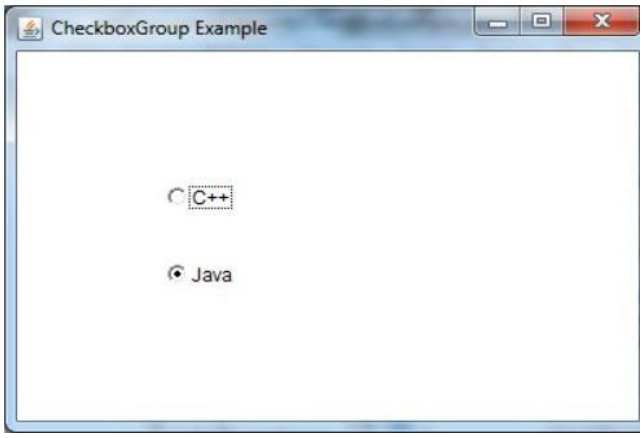
CheckboxGroup component

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.

```
import java.awt. *;
public class CheckboxGroupExample
{
public static void main(String args[])
{
  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);
}
}
```

Output:



Choice Component

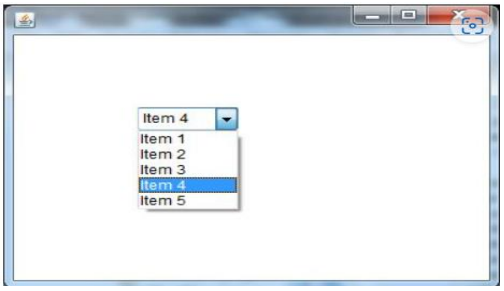
The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Constructor

Choice (): It constructs a new choice menu.

ex:

```
import java.awt.*;
public class ChoiceDemo {
public static void main(String args[])
{
    Frame f = new Frame(" Choice example");
    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);
}
}
```



List Component

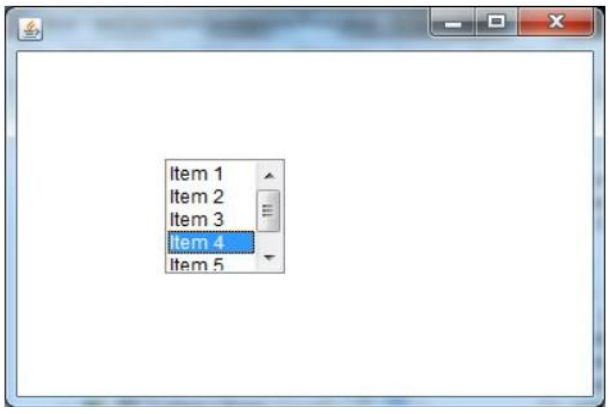
The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

Constructors

List() : It constructs a new scrolling list.
List(int row_num): It constructs a new scrolling list initialized with the given number of rows visible.
Ex:

```
import java.awt.*;
public class ListDemo
{
    ListExample1() {
       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 ListDemo();
}
}
```

Output:



Canvas Component

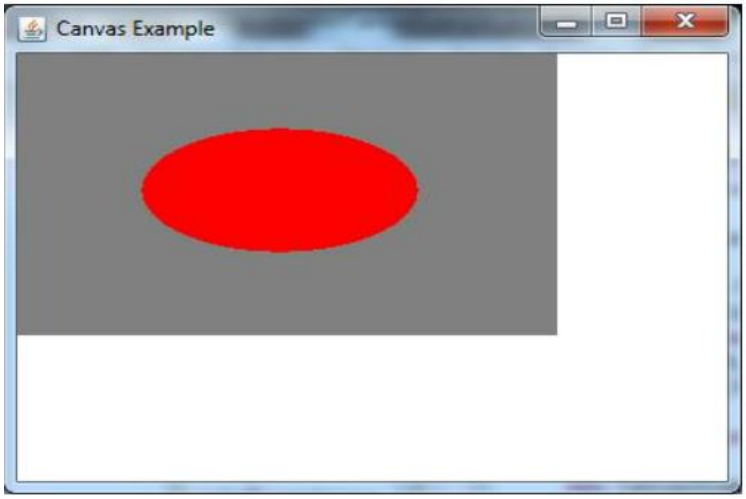
The Canvas class controls and represents a blank rectangular area where the application can draw or trap input events from the user.

It inherits the Component class.

Ex:

```
import java.awt. *;
public class CanvasExample
{
  public static void main(String args[])
  {
    Frame f = new Frame("Canvas Example");
    f.add(new MyCanvas());
    f.setLayout(null);
    f.setSize(400, 400);
    f.setVisible(true);
  }
}
    class MyCanvas extends Canvas
{
        public MyCanvas()
     {
        setBackground (Color.GRAY);
        setSize(300, 200);
     }
  public void paint(Graphics g)
  {
    g.setColor(Color.red);
    g.fillOval(75, 75, 150, 75);
  }
}
```

Output:



Scrollbar Component

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

It can be added to top-level container like Frame.

The Scrollbar class extends the Component class.

1.Scrollbar() Constructs a new vertical scroll bar.

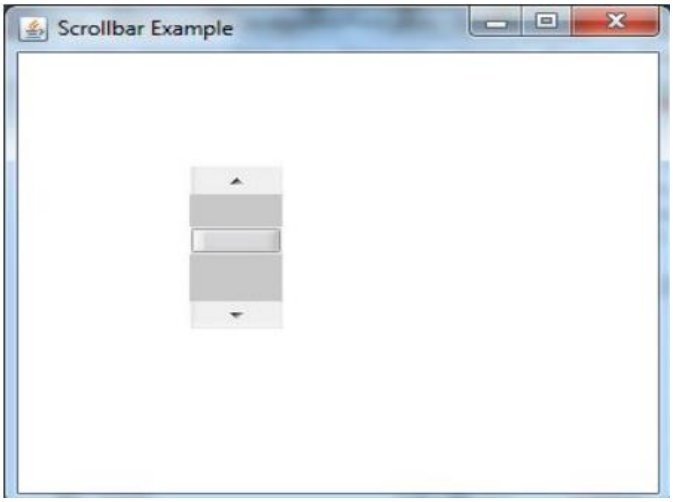
2.Scrollbar(int orientation)

Constructs a new scroll bar with the specified orientation.

Ex:

```
import java.awt. *;
public class ScrollbarExample1 {
 public static void main(String args[]) {
  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);
}
```

Output:



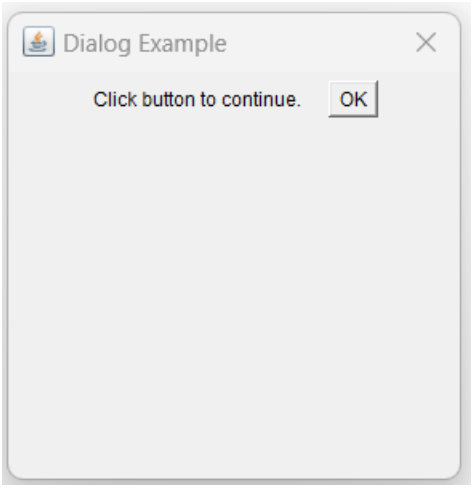
Dialog

Java AWT provides Dialog class that represents a dialog window, which is a pop-up window that typically requests information from or gives information to the user.

```
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");
        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:



ScrollPane

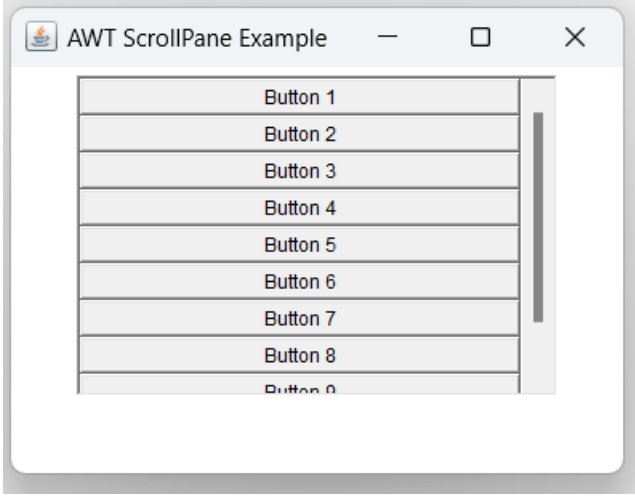
Java AWT ScrollPane is a container that provides a scrollable view of its contents.
It allows to place a large component inside a smaller container, and provides scroll bars to navigate through the component if it doesn't fit within the container.

Ex:

```
import java.awt.*;
import java.awt.event.*;
        public class ScrollPaneExample {
            public static void main(String[] args) {
                Frame frame = new Frame("AWT ScrollPane Example");
                frame.setSize(400, 300);
```

```
        frame.setLayout(new FlowLayout());
        frame.setVisible(true);
        ScrollPane scrollPane = new ScrollPane();
        scrollPane.setSize(300, 200);
        // Creating a large Panel to add inside the ScrollPane
        Panel panel = new Panel();
        panel.setLayout(new GridLayout(10, 1)); // Creating a 10 row grid layout
        for (int i = 1; i <= 10; i++) {
            panel.add(new Button("Button " + i)); // Adding buttons to the panel
        }
        scrollPane.add(panel);
        frame.add(scrollPane);          // Adding the ScrollPane to the Frame

    }
}
```

Output:



MenuBar

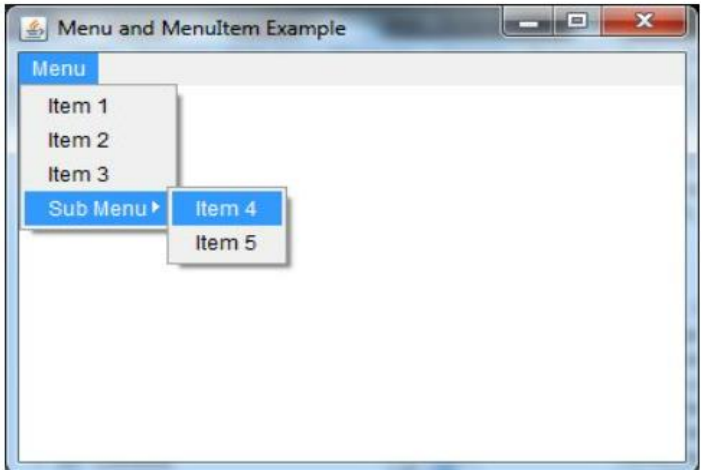
Java AWT provides the `MenuBar`, `Menu`, and `MenuItem` classes to create and manage menus in applications. A menu bar contains menus, which in turn contain menu items.

Ex:

```
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:



Understanding Layout Managers

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. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout

1. BorderLayout

The **BorderLayout** is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

```
import java.awt.*;
public class Border
{
Border()
{
   Frame f = new Frame();
   Button b1 = new Button("NORTH");
   Button b2 = new Button("SOUTH");
   Button b3 = new Button("EAST");
   Button b4 = new Button("WEST");
   Button b5 = new Button("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 Window:

2. GridLayout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

```
import java.awt. *;
public class MyGridLayout{
MyGridLayout(){
    Frame f=new Frame();
    Button    b1=new    Button("1");
    Button    b2=new    Button("2");
    Button    b3=new    Button("3");
    Button    b4=new    Button("4");
```

```
    Button    b5=new    Button("5");
    Button    b6=new    Button("6");
    Button    b7=new    Button("7");
    Button    b8=new    Button("8");
    Button    b9=new    Button("9");
    f.add(b1); f.add(b2); f.add(b3);
    f.add(b4); f.add(b5); f.add(b6);
    f.add(b7); f.add(b8); f.add(b9);
    f.setLayout(new GridLayout(3,3));
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) { new
    MyGridLayout();
}
}
```

Output:

3. FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Constructors of FlowLayout class

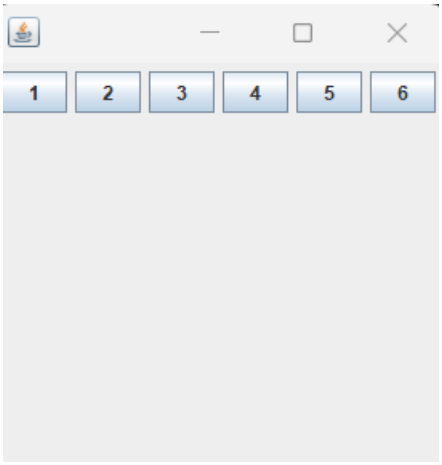
FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

```
import java.awt.*;
public class FlowLayoutExample
{
FlowLayoutExample()
{
Frame f = new Frame();
Button btn1 = new Button("1");
Button btn2 = new Button("2");
```

```
Button btn3 = new Button("3");
Button btn4 = new Button("4");
Button btn5 = new Button("5");
Button btn6 = new Button("6");
f.add(btn1); f.add(btn2); f.add(btn3);
f.add(btn4); f.add(btn5); f.add(btn6);
f.setLayout(new FlowLayout());
f.setSize(300, 300);
f.setVisible(true);
}
public static void main(String argvs[])
{
new FlowLayoutExample();
}
}
```

Output:



4. CardLayout

The Java CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout Class

CardLayout(): creates a card layout with zero horizontal and vertical gap.

CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

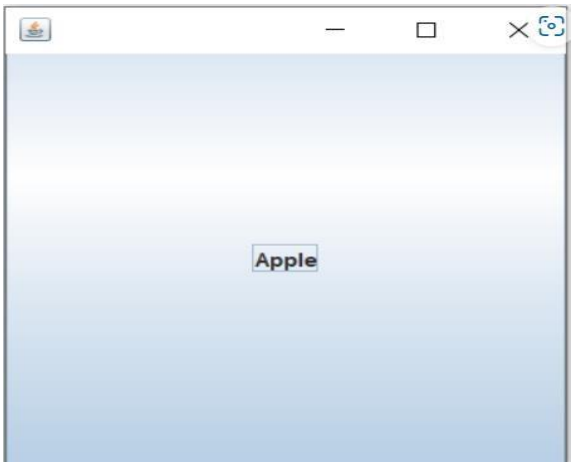
Commonly Used Methods of CardLayout Class

public void next(Container parent): is used to flip to the next card of the given container.
public void previous(Container parent): is used to flip to the previous card of the given container.
public void first(Container parent): is used to flip to the first card of the given container.
public void last(Container parent): is used to flip to the last card of the given container.

```
import java.awt.*;
import java.awt.event.*;
public class CardLayoutExample extends Frame implements ActionListener
{
CardLayout crd;
Button btn1, btn2, btn3;
Container cPane;
CardLayoutExample()
{
cPane = getContentPane();
crd = new CardLayout();
cPane.setLayout(crd);
btn1 = new Button("Apple");
btn2 = new Button("Orange");
btn3 = new Button("Banana");
btn1.addActionListener(this);
btn2.addActionListener(this);
btn3.addActionListener(this);
cPane.add("a", btn1);
cPane.add("b", btn2);
cPane.add("c", btn3);
}
public void actionPerformed(ActionEvent e)
{
crd.next(cPane);
}
```

```
public static void main(String argvs[])
{
CardLayoutExample crdl = new CardLayoutExample();
crdl.setSize(300, 300);
crdl.setVisible(true);
}
}
```

Output:



When the button named apple is clicked, we get another button.