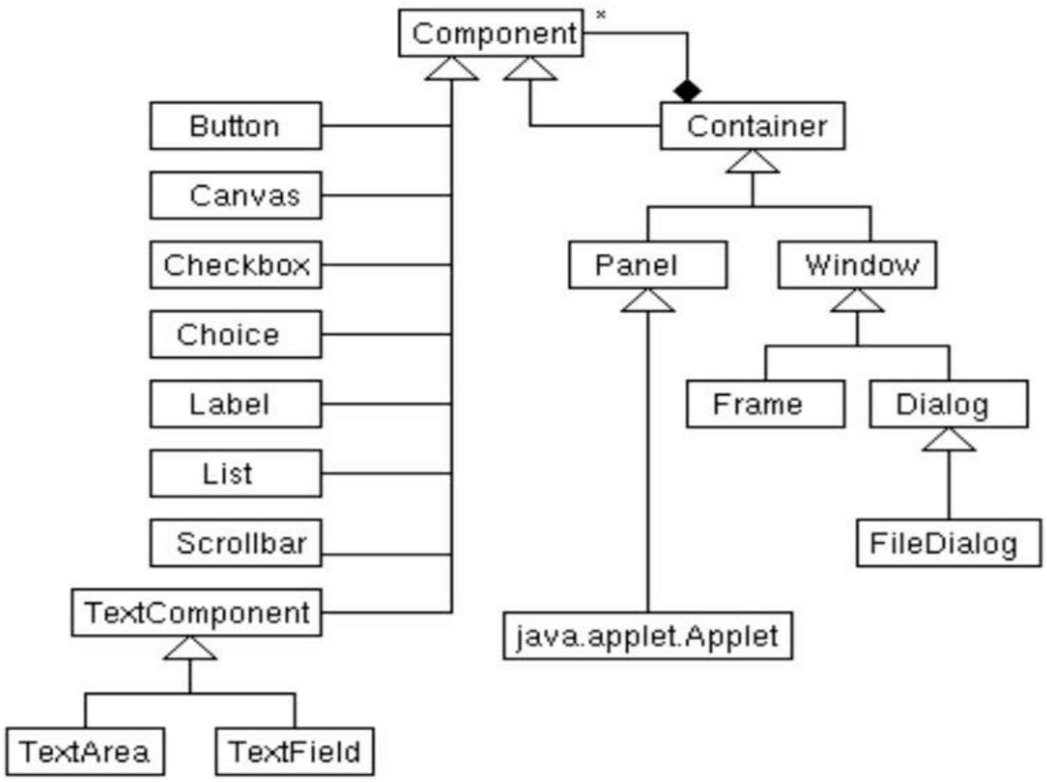
OOPS Through Java
Chapter – 4

What is AWT in java, Explain?

AWT stands for Abstract Window Toolkit. It is a platform-dependent API (Application Programming Interface) to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by Sun Microsystems In 1995. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods, which are used for creating and managing GUI. The java.awt package provides classes such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

AWT class Hierarchy



Features of AWT in Java

- AWT is a set of native user interface components
- It is based upon a robust event-handling model
- It provides Graphics and imaging tools, such as shape, color, and font classes
- AWT also avails layout managers which helps in increasing the flexibility of the window layouts
- Data transfer classes are also a part of AWT that helps in cut-and-paste through the native platform clipboard
- Supports a wide range of libraries that are necessary for creating graphics for gaming products, banking services, educational purposes, etc.

Characteristics

- It is a set of native user interface components.
- It is very robust in nature.
- It includes various editing tools like graphics tool and imaging tools.

- It uses native window-system controls.
- It provides functionality to include shapes, colors and font classes.

Advantages

- It takes very less memory for development of GUI and executing programs.
- It is highly stable as it rarely crashes.
- It is dependent on operating system so performance is high.
- It is easy to use for beginners due to its easy interface.

Disadvantages

- The buttons of AWT does not support pictures.
- It is heavyweight in nature.
- Some important components like Tree, Tables, TabbedPane, etc. are not present.
- Extensibility is not possible as it is platform dependent.

Useful Methods of Component class

public void add(Component c)	: It inserts sub component on a main component.
public void setSize(int width,int height)	: It sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	: It defines the layout manager for the component.
public void setVisible(boolean status)	: It changes the visibility of the component, by default false.

Basic Terminology in AWT

1. **Component** : - Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.
2. **Container** : - Container object is a component that can contain other components.Components added to a container are tracked in a list. The order of the list will define the components' front-toback stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list.
3. **Panel** : - Panel provides space in which an application can attach any other components, including other panels.
4. **Window** : - Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data. Window provide us with multitasking environment. A window must have either a frame, dialog, or another window defined as its owner when it's constructed.
5. **Frame** : - A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates window. It and has a title bar, menu bar, borders, and resizing corners.
6. **Canvas: -** Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use from that blank area of Canvas component.

AWT Controls

Every user interface considers the following three main aspects.

- **UI elements:** - These are the core visual elements the user eventually sees and interacts with AWT. AWT provides a huge list of widely used and common elements varying from basic to complex.
- **Layouts: -** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.

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

Events and Listeners in AWT

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. java.awt package provides many event classes and Listener interfaces for event handling.

TYPES OF EVENTS

1. Foreground Events - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
2. Background Events: - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling in Java?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. The Delegation Event Model has the following key participants. They are …
Source: - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
Listener: - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event.
Once the event is received, the listener processes the event and then returns. The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to that listener that wants to receive them.
Steps involved in event handling: -

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- The method is now getting executed and returns.

AWT Event Classes:

1. **AWT Event: -** It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.
2. **ActionEvent: -** The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3. **InputEvent: -** The InputEvent class is root event class for all component-level input events.
4. **KeyEvent: -** On entering the character the Key event is generated.

5. **MouseEvent:** - This event indicates a mouse action occurred in a component.
6. **TextEvent: -** The object of this class represents the text events. Reference:- www.javatpoint.com and www.tutorialspoint.com
7. **WindowEvent: -** The object of this class represents the change in state of a window.
8. **AdjustmentEvent: -** The object of this class represents the adjustment event emitted by Adjustable objects.
9. **ComponentEvent: -** The object of this class represents the change in state of a window.
10. **ContainerEvent: -** The object of this class represents the change in state of a window.
11. **MouseMotionEvent: -** The object of this class represents the change in state of a window.
12. **PaintEvent: -** The object of this class represents the change in state of a window

Java Event classes and Listener interfaces

The following event classes and listener interfaces are available in java.awt package.

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

Explain AWT UI Elements in Java?

Following is the list of commonly used controls while designed GUI using AWT.

1 Label: - A Label object is a component for placing text in a container.
2 Button: - This class creates a labeled button.
3 CheckBox: - A check box is a graphical component that can be in either an on (true) or off (false) state.
4 CheckBoxGroup: - The CheckboxGroup class is used to group the set of checkbox.
5 List The List component presents the user with a scrolling list of text items.
6 TextField: - A TextField object is a text component that allows for the editing of a single line of text.
7 TextArea: - A TextArea object is a text component that allows for the editing of a multiple lines of text.
8 Choice: - A Choice control is used to show popup menu of choices. Selected choice is shown on the top of the menu.
9 Canvas: - A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10 Image: - An Image control is superclass for all image classes representing graphical images.
11 ScrollBar: - A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12 Dialog: - A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13 FileDialog: - A FileDialog control represents a dialog window from which the user can select a file

Registration Methods

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

Button
- ✓ public void addActionListener(ActionListener a){}

MenuItem
- ✓ public void addActionListener(ActionListener a){}

TextField
- ✓ public void addActionListener(ActionListener a){}
- ✓ public void addTextListener(TextListener a){}

TextArea
- ✓ public void addTextListener(TextListener a){}

Checkbox
- ✓ public void addItemListener(ItemListener a){}

Choice
- ✓ public void addItemListener(ItemListener a){}

List
- ✓ public void addActionListener(ActionListener a){}
- ✓ public void addItemListener(ItemListener a){}

Explain about Dialog and Canvas classes in AWT?

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

AWT Canvas Class: -
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.

Constructor in the Canvas class: -
Canvas class provides two constructors.
1. **Canvas()**: Creates a new blank canvas.
2. **Canvas(GraphicsConfiguration c)**: Creates a new canvas with a specified graphics configuration.

Methods of Canvas Class: -
1. **void addNotify() :** It creates the canvas's peer.
2. **void createBufferStrategy (int numBuffers)**
It creates a new multi buffering strategies on the particular component.
3. **void createBufferStrategy (int numBuffers, BufferCapabilities caps)**
It creates a new multi buffering strategies on the particular component with the given buffer capabilities.
4. **AccessibleContext getAccessibleContext():** It gets the accessible context related to the Canvas.
5. **BufferStrategy getBufferStrategy():** It returns the buffer strategy used by the particular component.
6. **void paint(Graphics g) :** It paints the canvas with given Graphics object.
7. **void pdate(Graphics g):** It updates the canvas with given Graphics object.

Note1: - Canvas class has inherited these methods from java.lang.Component and java.lang.Object classes.
Note2: - When it comes to design, there are two terms that are often used interchangeably: Canva and Canvas. Although both of them refer to a blank space or surface where you can create something, there is a big difference between the two. Canva is a user-friendly online platform that anyone can use to create stunning designs, while Canvas is an advanced tool that is mostly used by professional graphic designers and artists.

Explain about MouseListener Interface and MouseMotionListener Interface in Java?

Java MouseListener Interface

The MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

<u>Methods of MouseListener interface: -</u>

The signature of 5 methods found in MouseListener interface are given below:

1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

Java MouseMotionListener Interface

The MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

<u>Methods of MouseMotionListener interface</u>

The signature of 2 methods found in MouseMotionListener interface are given below:

1. public abstract void mouseDragged(MouseEvent e);
2. public abstract void mouseMoved(MouseEvent e);

Explain about ItemListener and KeyListener Interfaces in Java?

ItemListener Interface: - The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method as follows.

itemStateChanged(): - This method is invoked automatically whenever you click or unclick on the registered checkbox component.

KeyListener Interface: - Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.
The signature of the three methods found in KeyListener interface are given below.

1. public abstract void keyPressed(KeyEvent e);
2. public abstract void keyReleased(KeyEvent e);
3. public abstract void keyTyped(KeyEvent e);

Explain about WindowListener Interface in Java?

WindowListener interface is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.
Methods of WindowListener interface: -

The signature of 7 methods found in WindowListener interface are given below:

1. public void windowActivated(WindowEvent e) {}
2. public void windowClosed(WindowEvent e) {}
3. public void windowClosing(WindowEvent e) { dispose(); }
4. public void windowDeactivated(WindowEvent e) {}
5. public void windowDeiconified(WindowEvent e) {}
6. public void windowIconified(WindowEvent e) {}
7. public void windowOpened(WindowEvent arg0) {}

How to close java.awt.Frame?

We can close the AWT Window or Frame by calling dispose() or System.exit() inside windowClosing() method. The windowClosing() method is found in WindowListener interface and WindowAdapter class.

The WindowAdapter class implements WindowListener interfaces. It provides the default implementation of all the seven methods of WindowListener interface. To override the windowClosing() method, you can either use WindowAdapter class or WindowListener interface.

If you implement the WindowListener interface, you will be forced to override all the 7 methods of WindowListener interface. So it is better to use WindowAdapter class.

Explain about Adapter Classes in Java?

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code. The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages. The Adapter classes with their corresponding listener interfaces are given below.

i) java.awt.event Adapter classes

Adapter class	**Listener interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

ii) java.awt.dnd Adapter classes

Adapter class	**Listener interface**
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

iii) javax.swing.event Adapter classes

Adapter class	**Listener interface**
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Explain about LayoutManagers in Java?

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.
The following are the classes that represent layout managers.

1. java.awt.BorderLayout
2. java.awt.GridLayout
3. java.awt.FlowLayout
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.

1) BorderLayout

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 of BorderLayout class: -

- **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.

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.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

3) FlowLayout

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.

Fields of FlowLayout class: -

public static final int LEFT
public static final int RIGHT
public static final int CENTER
public static final int LEADING
public static final int TRAILING

Constructors of FlowLayout class: -

1. **FlowLayout():**
   It creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):**
   creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):**
   creates a flow layout with the given alignment and the given horizontal and vertical gap.

4) CardLayout

The **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: -

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

void first(Container parent)	**:** is used to flip to the first card of the given container.
void last(Container parent)	**:** is used to flip to the last card of the given container.
void show(Container, String)	: is used to flip to the specified card with the given name.

5) GridBagLayout

GridBagLayout class is used to align components vertically, horizontally or along their baseline. The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of the constraints object, we arrange the component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine the component's size. GridBagLayout components are also arranged in the rectangular grid but can have many different sizes and can occupy multiple rows or columns.

Fields of GridBagLayout class: -

- columnWeights It is used to hold the overrides to the column weights.
- columnWidths It is used to hold the overrides to the column minimum width.
- rowHeights It is used to hold the overrides to the row minimum heights.
- rowWeights It is used to hold the overrides to the row weights.
- defaultConstraints It is used to hold a gridbag constraints instance containing the default values.

Methods of GridBagLayout class: -

- **arrangeGrid(Container parent)** It Lays out the grid.
- **addLayoutComponent(Component comp, Object constraints)**
  - It adds specified component to the layout, using the specified constraints object.
- **getLayoutWeights()** It determines the weights of the layout grid's columns and rows.
- **getLayoutDimensions()** It determines column widths and row heights for the layout grid.

6) BoxLayout

BoxLayout class is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

1. public static final int X_AXIS
2. public static final int Y_AXIS
3. public static final int LINE_AXIS
4. public static final int PAGE_AXIS

Constructor of BoxLayout class: -

BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

7) GroupLayout

GroupLayout class groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class. Group is an abstract class, and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup. SequentialGroup positions its child sequentially one after another whereas ParallelGroup aligns its child on top of each other. The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

8) ScrollPaneLayout

The layout manager is used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.

Methods of ScrollPaneLayout: -

ScrollPaneLayout(): This parameterless constructor is used to create a new ScrollPanelLayout.

addLayoutComponent(String s, Component c)	It adds the specified component to the layout.
getColumnHeader()	It returns the JViewport object that is the column header.
getCorner(String key)	It returns the Component at the specified corner.
getHorizontalScrollBar()	It returns the JScrollBar object that handles horizontal scrolling.
getHorizontalScrollBarPolicy()	It returns the horizontal scrollbar-display policy.
getRowHeader()	It returns the JViewport object that is the row header.
getVerticalScrollBar()	It returns the JScrollBar object that handles vertical scrolling.
getVerticalScrollBarPolicy()	It returns the vertical scrollbar-display policy.
getViewport()	It returns the JViewport object that displays the scrollable contents.

9) SpringLayout

The SpringLayout class arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object. Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them. Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

PROGRAMS USING AWT

1. /* Write java program to demonstrate button event */

```
import java.awt.*;
import java.awt.event.*;
public class ButtonExample1
{
  ButtonExample1()
  {
     Frame f=new Frame("ButtonDemo");
     TextField tf=new TextField();
     Button b1=new Button("Click Me");
     Button b2=new Button("Clear Me");
     tf.setBounds(50,50, 150,20);
     b1.setBounds(50,100,60,30);
     b2.setBounds(50,150,60,30);
     f.add(b1);
     f.add(b2);
     f.add(tf);
     f.setSize(400,400);
     f.setLayout(null);
     f.setVisible(true);

     b1.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent e){
     tf.setText("Welcome to Java");}});

     b2.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent e){
     tf.setText("");}});
```

```
        f.addWindowListener(new WindowAdapter()
        { public void windowClosing(WindowEvent e) { f.dispose(); } });
  }
  public static void main(String[] args)
  {
        new ButtonExample1();
  }
}
```

2. /* Write java program to demonstrate swapping using button event */

```
import java.awt.*;
import java.awt.event.*;
public class ButtonExample2
{
  ButtonExample2()
  {
        Frame f=new Frame("Swap Window");
        Label l1 = new Label("First Name");
        Label l2 = new Label("Last Name");
        TextField tf1=new TextField();
        TextField tf2=new TextField();
        Button b1=new Button("Swap");
        Button b2=new Button("Clear");
        l1.setBounds(50,50, 100,20);
        l2.setBounds(50,100, 100,20);
        tf1.setBounds(150,50, 150,20);
        tf2.setBounds(150,100, 150,20);
        b1.setBounds(50,150,60,30);
        b2.setBounds(50,200,60,30);
        f.add(l1);
        f.add(tf1);
        f.add(l2);
        f.add(tf2);
        f.add(b1);
        f.add(b2);
        f.setSize(400,300);
        f.setBackground(Color.pink);
        f.setLayout(null);
        f.setVisible(true);

        b1.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        String t= tf1.getText();
        tf1.setText(tf2.getText());
        tf2.setText(t);}});

        b2.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        tf1.setText(""); tf2.setText("");}});
        f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e) { f.dispose(); } });
    }
    public static void main(String[] args)
    {
        new ButtonExample2();
    }
}
```

3. /* Java program to check given number is EVEN/ODD using button event */

```
import java.awt.*;
import java.awt.event.*;
public class ButtonExample3
{
    ButtonExample3()
    {
        Frame f=new Frame("Even/Odd Window");
        Label l1 = new Label("Enter a number");
        Label l2 = new Label("Output");
        TextField tf1=new TextField();
        tf1.setBounds(200,50, 150,20);
        l1.setBounds(50,50, 150,20);
        l2.setBounds(50,100, 150,20);
        Button b1=new Button("Check");
        Button b2=new Button("Clear");
        b1.setBounds(50,150,60,30);
        b2.setBounds(50,200,60,30);
        f.add(l1);
        f.add(tf1);
        f.add(l2);
        f.add(b1);
        f.add(b2);
        f.setSize(400,300);
        f.setBackground(Color.pink);
        f.setLayout(null);
        f.setVisible(true);

        b1.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        int n = Integer.parseInt(tf1.getText());
        if(n%2==0)
            l2.setText("It is EVEN number");
        else
            l2.setText("It is Odd number"); } } );

        b2.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        tf1.setText("");  l2.setText("Output"); } } );
        f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e) { f.dispose(); } } );
    }
    public static void main(String[] args)
    {
        new ButtonExample3();
    }
}
```

4. /* Write java program to demonstrate Checkbox event */

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample extends Frame implements ItemListener
{
    Checkbox cb1,cb2;
    Label l1,l2;
    CheckboxExample ()
    {
        Frame f= new Frame("CheckBox Example");
        l1= new Label("C++ Checkbox");
        l2= new Label("Java Checkbox");
        cb1 = new Checkbox("C++");
        cb2 = new Checkbox("Java");
        l1.setSize(300,100);
        l2.setSize(300,100);
        cb1.setBounds(100,200, 50,50);
        cb2.setBounds(200,200, 50,50);
        l1.setBounds(100,50, 300,100);
        l2.setBounds(100,70, 300,200);
        cb1.addItemListener(this);
        cb2.addItemListener(this);
        f.add(cb1);
        f.add(cb2);
        f.add(l1);
        f.add(l2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e) { f.dispose(); } } );
    }

    public void itemStateChanged(ItemEvent e)
    {
        if(e.getSource()==cb1)
            l1.setText(e.getStateChange()==1 ? "C++: Checked" : "C++: Unchecked");
        if(e.getSource()==cb2)
            l2.setText(e.getStateChange()==1 ? "Java : Checked" : "Java : Unchecked");
    }
```

```
   public static void main(String args[])
   {
     new CheckboxExample();
   }
}
```

5. /* Write java program to demonstrate CheckboxGroup event */

[OR]

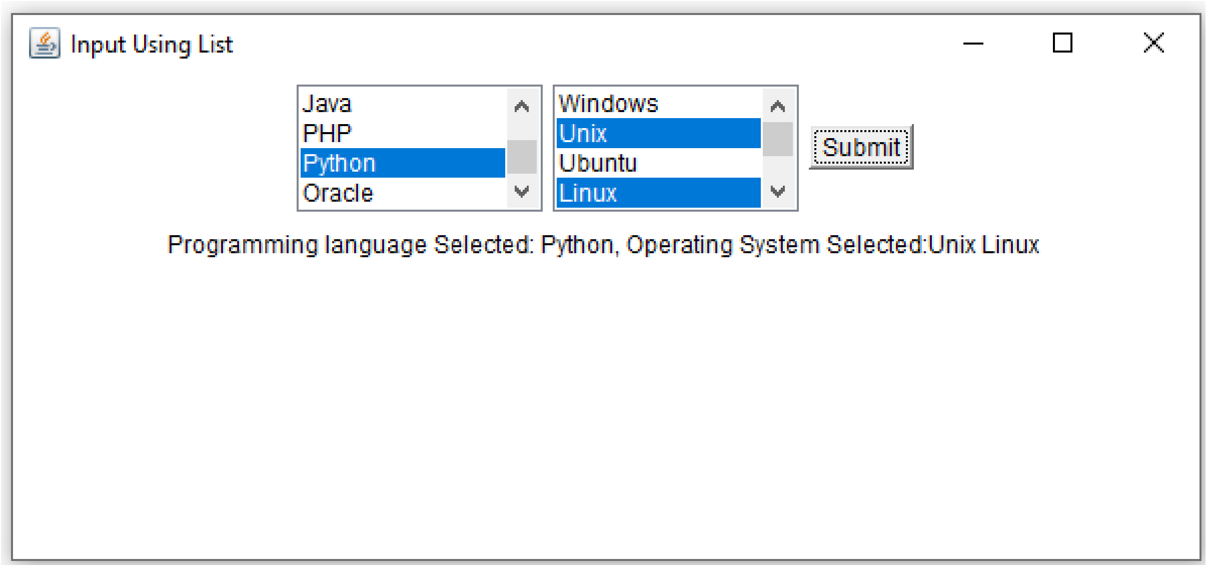
/* Write java program to read input using radio button (option button) */

```
import java.awt.*;
import java.awt.event.*;
public class RadioButtonExample
{
   RadioButtonExample()
   {
      Frame f= new Frame("CheckboxGroup Example");
      Label label = new Label();

      CheckboxGroup cbg = new CheckboxGroup();

      Checkbox cb1 = new Checkbox("C++", cbg, false);
      Checkbox cb2 = new Checkbox("Java", cbg, false);
      label.setSize(400,100);
      cb1.setBounds(100,100, 50,50);
      cb2.setBounds(100,150, 50,50);
      label.setBounds(100,200, 400,100);
      f.add(cb1);
      f.add(cb2);
      f.add(label);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);

      cb1.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
      label.setText("Output : C++ Selected");  }  } );

      cb2.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
      label.setText("Output: Java Selected"); }  } );

      f.addWindowListener(new WindowAdapter()
      { public void windowClosing(WindowEvent e) { f.dispose(); } } );
   }
   public static void main(String args[])
   {
      new RadioButtonExample();
   }
}
```

6. /* Write java program to read input using awt list box */

```
import java.awt.*;
import java.awt.event.*;
public class ListExample
{
  ListExample()
  {
        Frame f = new Frame("Input Using List");
        Label label = new Label("Output");
        Button b = new Button("Submit");

        label.setAlignment(Label.CENTER);
        label.setSize(500, 100);

        List l1 = new List(4, false);
        l1.add("C");
        l1.add("C++");
        l1.add("Java");
        l1.add("PHP");
        l1.add("Python");
        l1.add("Oracle");

        List l2=new List(4, true);
        l2.add("Windows");
        l2.add("Unix");
        l2.add("Ubuntu");
        l2.add("Linux");
        l2.add("iOS");
        l2.add("MacOS");
        l2.add("Android");

        f.add(l1);
        f.add(l2);
        f.add(b);
        f.add(label);
        f.setSize(450,450);
        f.setLayout(new FlowLayout());
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter()
        { public void windowClosing(WindowEvent e) { f.dispose(); } } );

         b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
         String data = "Programming language Selected:"+l1.getItem(l1.getSelectedIndex());
         data = data + ", Operating System Selected:";
         for(String os : l2.getSelectedItems())
         {
            data = data + os + " ";
        }
        label.setText(data); } } );
  }
  public static void main(String args[])
  {
    new ListExample();
  }
}
```

Output: -



```
Programming language Selected: Python, Operating System Selected:Unix Linux
```

7. /* Write java program to display an image on a frame */

```
import java.awt.*;
import java.awt.event.*;
public class AwtImage extends Frame
{
  Image img;
  public static void main(String[] args)
  {
     new AwtImage();
  }

  public AwtImage()
  {
    super("Image Frame");
    MediaTracker mt = new MediaTracker(this);
    img = Toolkit.getDefaultToolkit().getImage("icon_confused.gif");
    mt.addImage(img,0);
    setSize(500,500);
    setVisible(true);
    addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent we){ dispose(); } });
  }
  public void update(Graphics g)
  {
    paint(g);
  }

  public void paint(Graphics g)
  {
    if(img != null)
      g.drawImage(img, 100, 100, this);
    else
      g.clearRect(0, 0, getSize().width, getSize().height);
  }
}
```

8. /* Java program to draw something on a frame using AWT mouse events */

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener
{
 MouseMotionListenerExample()
 {
    addMouseMotionListener(this);
    addWindowListener(new WindowAdapter()
    {  public void windowClosing(WindowEvent e) { dispose(); } });
    setSize(300,300);
    setLayout(null);
    setVisible(true);
 }
 public void mouseDragged(MouseEvent e)
 {
   Graphics g=getGraphics();
   g.setColor(Color.BLUE);
   g.fillOval(e.getX(),e.getY(),10,10);
 }
 public void mouseMoved(MouseEvent e) {}
 public static void main(String[] args)
 {
   new MouseMotionListenerExample();
 }
}
```

9. /* Write java program to demonstrate Regular Menu event */

```
import java.awt.*;
import java.awt.event.*;
class MenuExample
{
   MenuExample()
   {
      Frame f= new Frame("Menu and MenuItem Example");
```

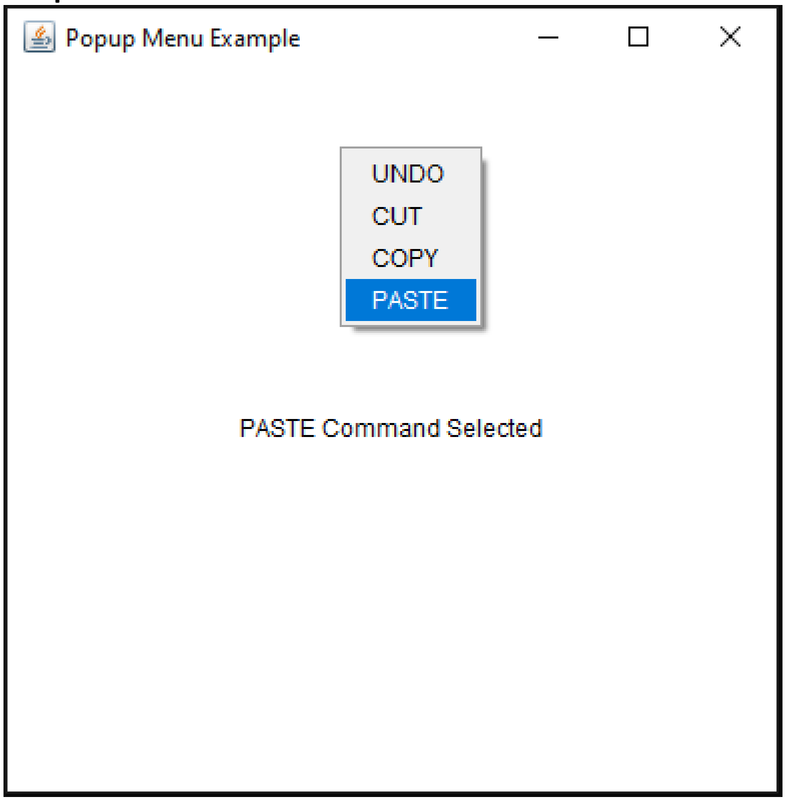
```
        Label  statusLabel = new Label();
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("PRINT");
        MenuItem i1=new MenuItem("OPEN");
        MenuItem i2=new MenuItem("SAVE");
        MenuItem i3=new MenuItem("CLOSE");
        MenuItem i4=new MenuItem("Current Page");
        MenuItem i5=new MenuItem("All Pages");
        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.add(statusLabel);
        statusLabel.setSize(350,350);
        statusLabel.setAlignment(Label.CENTER);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent windowEvent){ System.exit(0); } } );

        menu.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
        statusLabel.setText(ae.getActionCommand() + " Command Selected"); } });

        submenu.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
        statusLabel.setText(ae.getActionCommand() + " Command Selected"); } });
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}
```

10. /* Write java program to demonstrate Popuo menu event */

```
import java.awt.*;
import java.awt.event.*;
public class PopupMenuExample
{
   Frame f = new Frame("Popup Menu Example");
   Label  statusLabel = new Label();
   Panel  controlPanel = new Panel();
```

```
void showPopup()
{
   PopupMenu pm= new PopupMenu("Edit");
   MenuItem undo = new MenuItem("UNDO");
   undo.setActionCommand("UNDO");

   MenuItem cut = new MenuItem("CUT");
   cut.setActionCommand("CUT");

   MenuItem copy = new MenuItem("COPY");
   copy.setActionCommand("COPY");

   MenuItem paste = new MenuItem("PASTE");
   paste.setActionCommand("PASTE");

   MenuItemListener m = new MenuItemListener();

   undo.addActionListener(m);
   cut.addActionListener(m);
   copy.addActionListener(m);
   paste.addActionListener(m);

   pm.add(undo);
   pm.add(cut);
   pm.add(copy);
   pm.add(paste);

   controlPanel.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)  {
   pm.show(f, e.getX(), e.getY()); } });

   controlPanel.add(pm);
   f.add(pm);
   f.setVisible(true);
}

PopupMenuExample()
{
    statusLabel.setSize(350,100);
    statusLabel.setAlignment(Label.CENTER);
    controlPanel.setLayout(new FlowLayout());

    f.setSize(400,400);
    f.setLayout(new GridLayout(3, 1));

    f.add(controlPanel);
    f.add(statusLabel);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent windowEvent){ System.exit(0); } } );
  }

  public static void main(String[] args)
  {
    PopupMenuExample  x = new PopupMenuExample();
    x.showPopup();
  }

  class MenuItemListener implements ActionListener
  {
    public void actionPerformed(ActionEvent e)
    {
      statusLabel.setText(e.getActionCommand() + " Command Selected");
    }
  }
}
```

Output: -

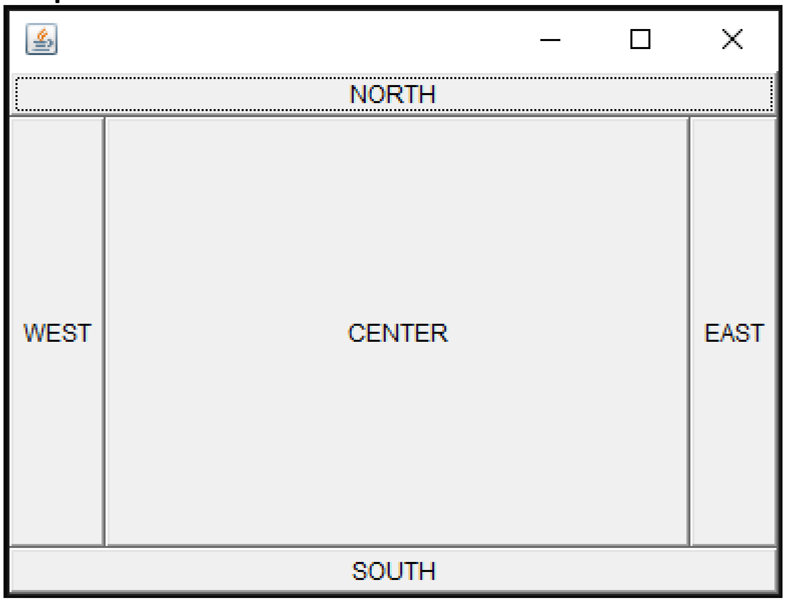


PASTE Command Selected

11. /* Write java program to demonstrate Frame BorderLayout */

```
import java.awt.*;
import java.awt.event.*;
public class BorderLayoutExample
{
  Frame  f = new Frame();
  BorderLayoutExample()
  {
    Button b1 = new Button("NORTH");       // the button will be labeled as NORTH
    Button b2 = new Button("SOUTH");       // the button will be labeled as SOUTH
    Button b3 = new Button("EAST");        // the button will be labeled as EAST
    Button b4 = new Button("WEST");        // the button will be labeled as WEST
    Button b5 = new Button("CENTER");      // the button will be labeled as CENTER

    f.add(b1, BorderLayout.NORTH);         // b1 will be placed in the North Direction
    f.add(b2, BorderLayout.SOUTH);         // b2 will be placed in the South Direction
    f.add(b3, BorderLayout.EAST);          // b2 will be placed in the East Direction
    f.add(b4, BorderLayout.WEST);          // b2 will be placed in the West Direction
    f.add(b5, BorderLayout.CENTER);        // b2 will be placed in the Center

    f.addWindowListener(new WindowAdapter()
    { public void windowClosing(WindowEvent e) { f.dispose(); } });

    f.setSize(400, 300);
    f.setVisible(true);
  }
  public static void main(String[] args)
  {
   new BorderLayoutExample();
  }
}
```

Output: -


NORTH
WEST
CENTER
EAST
SOUTH