Unit – IV:
AWT (ABSTRACT WINDOW TOOL KIT)

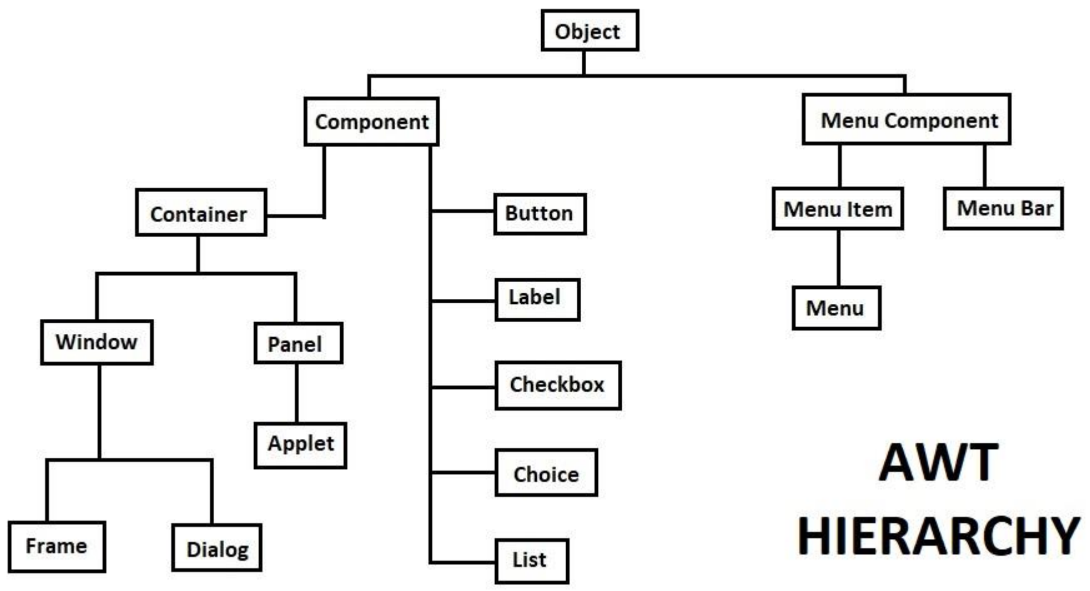
AWT: Class hierarchy, Component, Container, Panel, Window, Frame, Graphics.

AWT Controls:
Labels, Button, Scrollbar, Text Components, Checkbox, Checkbox Group, Choice, List, Panes –Scroll Pane, Dialog and Menu Bar.

Event Handling:
Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes.

Abstract Window Toolkit

AWT (Abstract Window Toolkit) in Java is an API used for developing Graphical User Interfaces (GUIs) and window-based applications. It is part of the Java Foundation Classes (JFC) and provides a set of classes for creating platform-independent graphical applications. AWT includes components like buttons, text boxes, and menus, and serves as a bridge between Java applications and the native operating system's windowing system



GUI Programing with Java

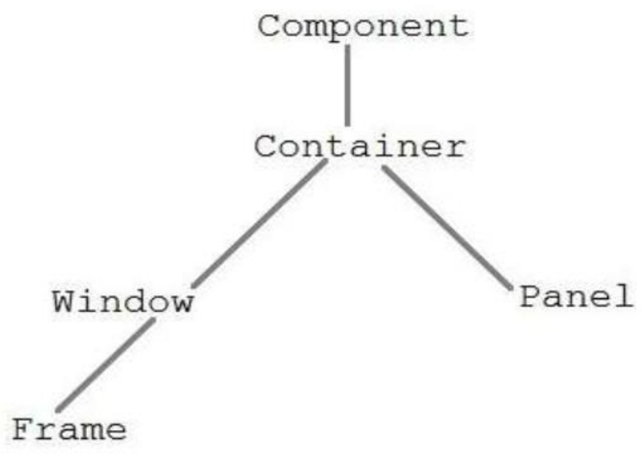
Java Provides 2 Frameworks for building GUI-based applications. Those are

- AWT-(Abstract Window Toolkit)
- Swing

AWT-(Abstract Window Toolkit):

- **AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- The **java.awt** package provides classes for AWT API such as TextField, Label, TextArea, Checkbox, Choice, List etc.
- **AWT** components are platform-dependent i.e. components are displayed according to the view of operating system.

AWT Hierarchy



- **Component:**

Component is an abstract class that contains various classes such as Button, Label,Checkbox,TextField, Menu and etc.

- **Container:**

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The Container class extends Frame and Panel.

- **Window:**

  The window is the container that have no borders and menu bars. You must use frame for creating a window.

- **Frame:**

  The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

- **Panel:**

  The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Commonly used Methods of Component class

Method	Description
add(Component c)	inserts a component on this component.
setSize(int width,int height)	sets the size (width and height) of the component.
setLayout(LayoutManager m)	defines the layout manager for the component.
setVisible(boolean status)	changes the visibility of the component, by default false.

→To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)

  Ex:

```
class Example extends Frame
{
  ……..
}
```

- By creating the object of Frame class (association)

  Ex:

```
class Example
{
Frame obj=new Frame();
  ……..
}
```

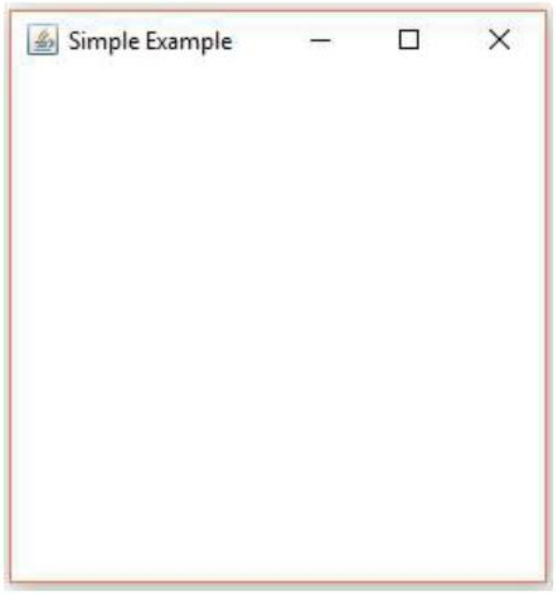
A Simple AWT Example:

SimpleExample.java

```
import java.awt.*;
public class AwtExample
{
public static void main(String[] args)
{
   Frame f=new Frame();
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   f.setTitle("Simple Example");
}
}
```

Output:

```
Javac AwtExample.java
Java AwtExample
```



AWT Components or Elements:

- **Button:**

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

    Syntax:

    ```
    Button b=new Button("Text");
    (Or)
    Button b1,b2;
    b1=new Button("Text");
    b.setBounds(50,100,80,30);
    ```

setBounds(int x,int y,int width,int height)

This method is used to declare location, width & height of all components of AWT.

Example: setBounds(50,100,80,30);

X Y

width Height

- **Label:**

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 an application but a user cannot edit it directly.

Syntax:

```
Label l1=new Label(“Text”);
(or)
Label l1,l2;
l1=new Label(“Text”);
```

- **TextField:**

The TextField class is a text component that allows the editing of a single line text.

Syntax:

```
TextField t1=new TextField(“Text”);
(or)
TextField t1,t2;
t1=new TextField(“Text”);
```

- **TextArea :**

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

Syntax:

```
 TextArea t1=new TextArea(“Text”);
(or)
 TextArea t1,t2;
t1=new TextArea(“Text”);
```

- **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".

Syntax:

```
 Checkbox c1=new Checkbox(“Text”);
(or)
 Checkbox c1,c2;
c1=new Checkbox(“Text”);
```

- **Choice :**

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

  Syntax:

  ```
  Choice c=new Choice();
  c.add("Item 1");
  c.add("Item 2");
  c.add("Item 3");
  ```

- **List :**

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

  Syntax:

  ```
  List ls=new List(Size);
  ls.add("Item 1");
  ls.add("Item 2");
  ls.add("Item 3");
  ```

- AWT does allow the user to close window directly…

- We need code to close window

  ```
  //Code for Close window
  addWindowListener(new WindowAdapter(){
  public void windowClosing(WindowEvent we)
  {
   System.exit(0);
  }
  });
  ```

Note: We need to import one package **"java.awt.event.*".**

Example Programs for AWT Components:

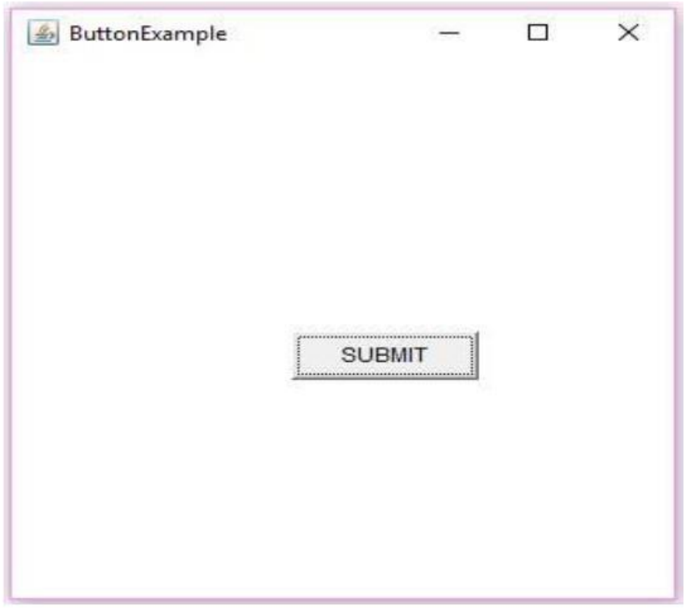
Example: An example for **Button** Component in AWT.

ButtonExample.java

```
import java.awt.*;
import java.awt.event.*;
class ButtonExample
{
public static void main(String[] args)
{
// creation of Frame
   Frame f=new Frame();
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   f.setTitle("ButtonExample");
//creation of Button
   Button b=new Button("SUBMIT");
   b.setBounds(150,200,95,30);
   f.add(b);
//Code for Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
javac ButtonExample.java
java ButtonExample
```



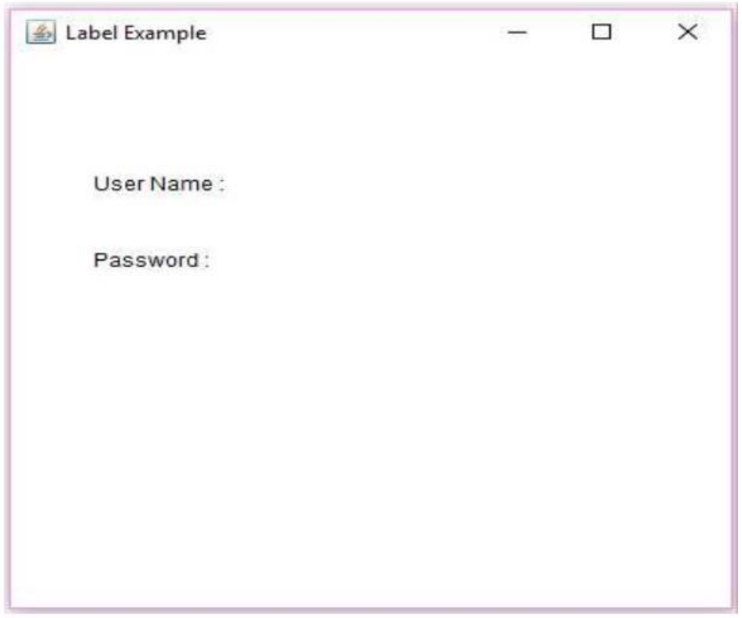
Example: An example for **Label** Component in AWT.

LabelExample.java

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

//Code for Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac LabelExample.java
Java LabelExample
```



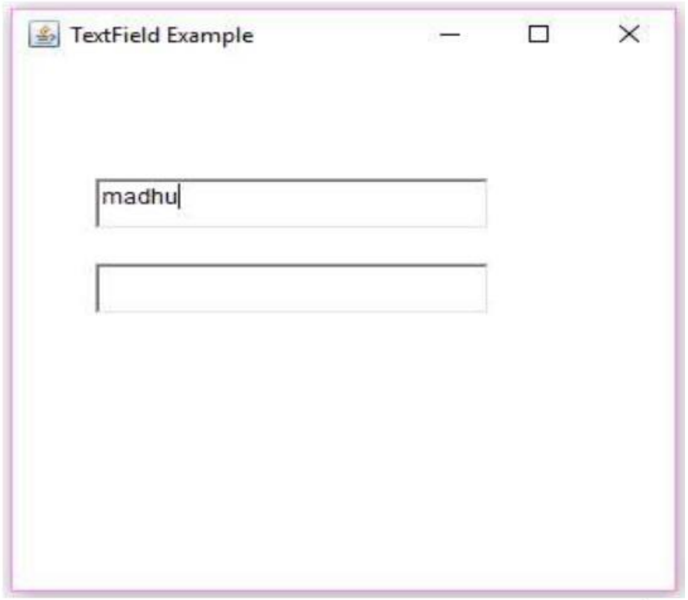
Example: An example for **TextField** Component in AWT.

TextFieldExample.java

```
import java.awt.*;
import java.awt.event.*;
class TextFieldExample
{
public static void main(String args[]){
  Frame f= new Frame();
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
  f.setTitle("TextField Example");
  TextField t1,t2;
  t1=new TextField("");
  t1.setBounds(50,100, 200,30);
  t2=new TextField("");
  t2.setBounds(50,150, 200,30);
  f.add(t1);
  f.add(t2);
//Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac TextFiledExample.java
Java TextFiledExample
```



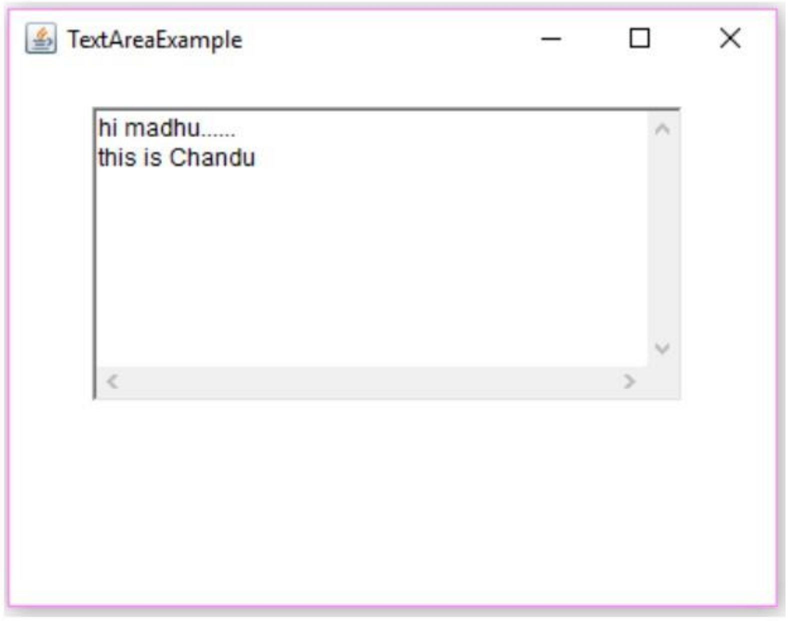
Example: An example for **TextArea** Component in AWT.

TextAreaExample.java

```
import java.awt.*;
import java.awt.event.*;
public class TextAreaExample
{
public static void main(String args[])
{
      Frame f= new Frame();
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      f.setTitle("TextAreaExample");
      TextArea area=new TextArea();
      area.setBounds(50,50, 300,150);
      f.add(area);
//Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac TextFieldExample.java
Java TextFieldExample
```



```
hi madhu......
this is Chandu
```

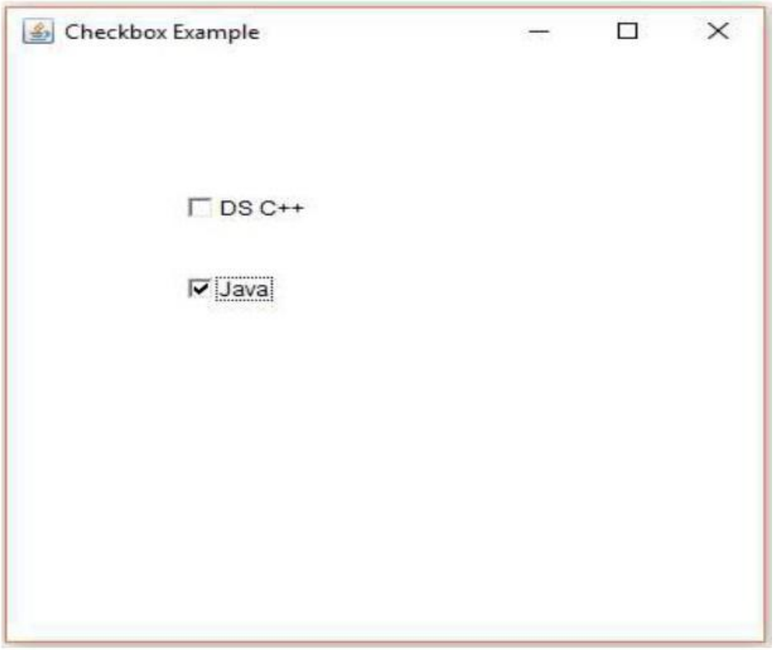
<u>Example:</u> An example for **Checkbox** Component in AWT.

<u>CheckboxExample.java</u>

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample
{
public static void main(String args[])
{
         Frame f= new Frame("Checkbox Example");
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
         f.setTitle("Checkbox Example");
      Checkbox chb1 = new Checkbox("DS C++");
      chb1.setBounds(100,100, 100,50);
      Checkbox chb2 = new Checkbox("Java", true);
      chb2.setBounds(100,150, 50,50);
      f.add(chb1);
      f.add(chb2);
//Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac CheckboxExample.java
Java CheckboxExample
```



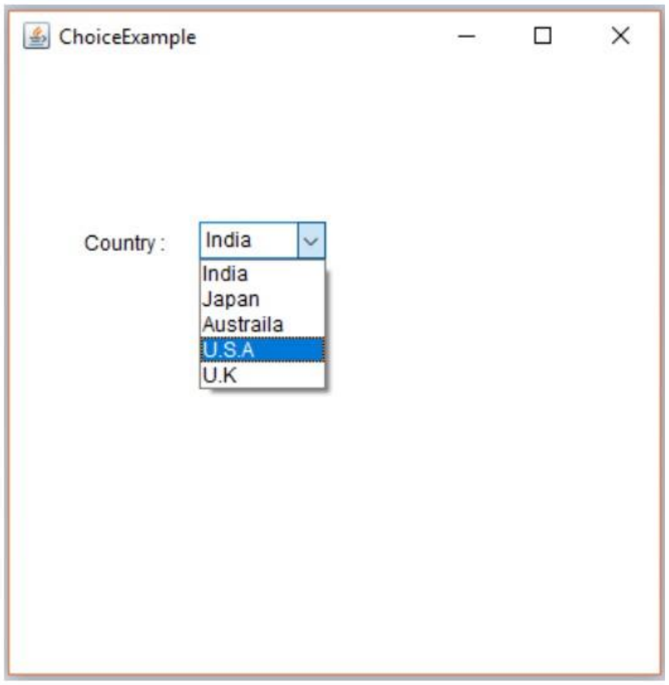
<u>Example:</u> An example for **Choice** Component in AWT.

<u>ChoiceExample.java</u>

```
import java.awt.*;
import java.awt.event.*;
public class ChoiceExample
{
public static void main(String args[])
{
      Frame f= new Frame();
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      f.setTitle("ChoiceExample");
      Label l=new Label("Country :");
      l.setBounds(50,100,75,75);
      Choice c=new Choice();
      c.setBounds(120,125,75,75);
      c.add("India");
      c.add("Japan");
      c.add("Austraila");
      c.add("U.S.A");
      c.add("U.K");
      f.add(c);
      f.add(l);
//code for close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac ChoiceExample.java
Java ChoiceExample
```



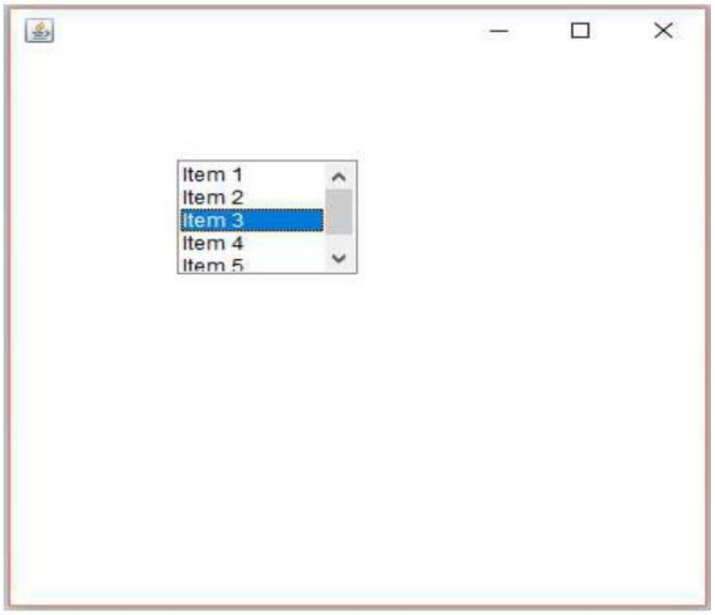
<u>Example:</u> An example for **List** Component in AWT.

<u>ListExample.java</u>

```
import java.awt.*;
import java.awt.event.*;
public class ListExample
{
  public static void main(String args[])
{
      Frame f= new Frame();
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      List l1=new List(5);
      l1.setBounds(100,100, 100,75);
      l1.add("Item 1");
      l1.add("Item 2");
      l1.add("Item 3");
      l1.add("Item 4");
      l1.add("Item 5");
      f.add(l1);
//Code for Close window
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
}
}
```

Output:

```
Javac ListExample.java
Java ListExample
```



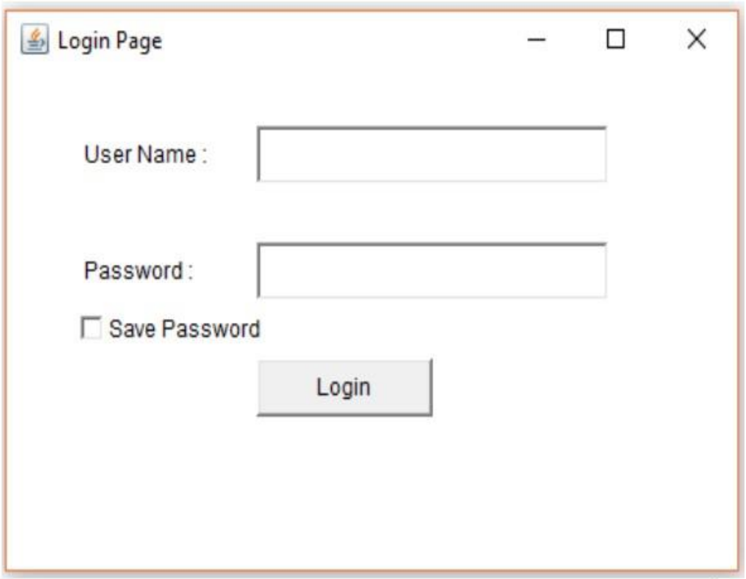
Example: An example program for **Login page** in AWT.

LoginExample.java

```
import java.awt.*;
import java.awt.event.*;
class LoginExample
{
public static void main(String args[])
{
  Frame f= new Frame ("Login Page");
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
  Label l1,l2;
  TextField t1,t2;
  Checkbox cb;
  Button b;
  l1=new Label("User Name :");
  l1.setBounds(50,60,100,30);
  t1=new TextField("");
  t1.setBounds(150,60, 200,30);
  l2=new Label("Password :");
  l2.setBounds(50,120,100,30);
  t2=new TextField("");
  t2.setBounds(150,120, 200,30);
  cb=new Checkbox("Save Password");
  cb.setBounds(50,150,200,30);
  b=new Button("Login");
  b.setBounds(150,180,100,30);
  f.add(l1);   f.add(l2);
  f.add(t1);   f.add(t2);
  f.add(cb);
  f.add(b);
}
}
```

Output:

```
Javac LoginExample.java
Java LoginExample
```



Example: An example program for **Registration page** in AWT.

RegistrationExample.java

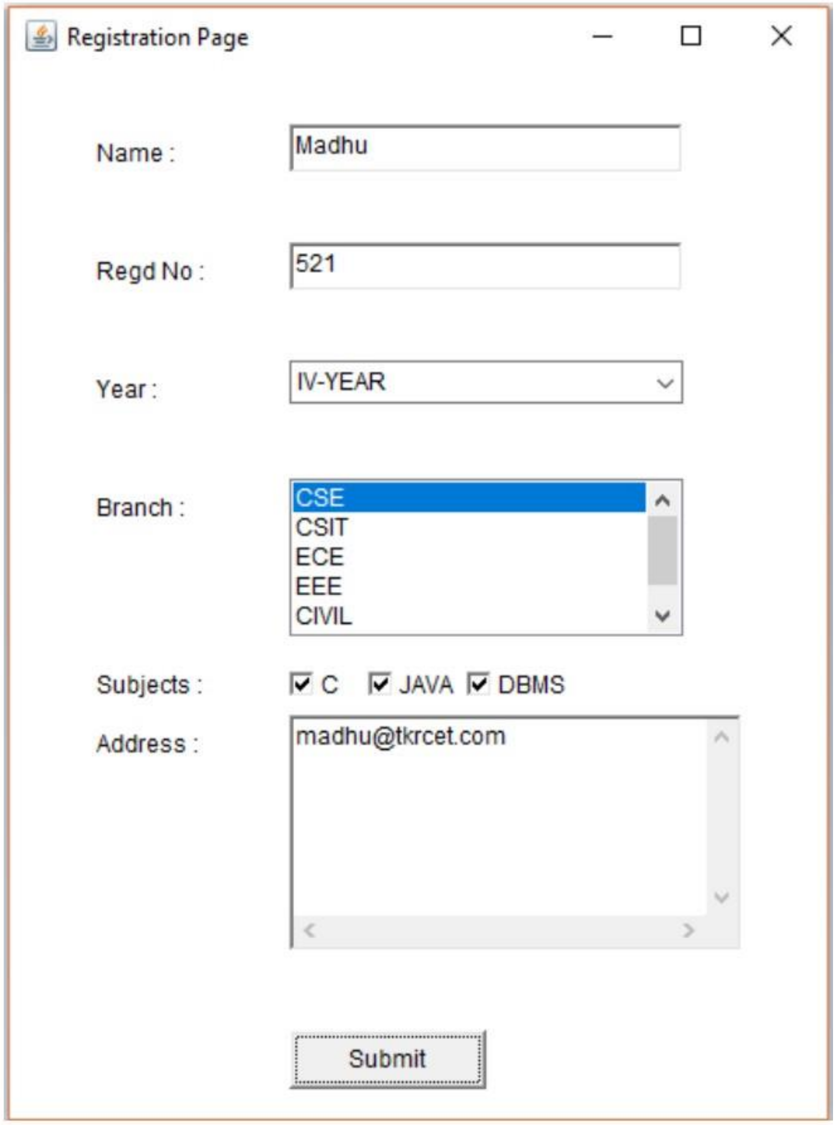
```
import java.awt.*;
import java.awt.event.*;
class RegistrationExample
{
public static void main(String args[])
{
  Frame f= new Frame();
  f.setSize(600,800);
  f.setLayout(null);
  f.setVisible(true);
  f.setTitle("Registration Page");
  Label l1,l2,l3,l4,l5,l6;
  TextField t1,t2;
  Choice ch;
  List ls;
  Checkbox cb1,cb2,cb3;
  TextArea ta;
  Button b;
  l1=new Label("Name :");
  l1.setBounds(50,60,100,30);
  t1=new TextField("");
  t1.setBounds(150,60, 200,25);
  l2=new Label("Regd No :");
  l2.setBounds(50,120,100,30);
  t2=new TextField("");
  t2.setBounds(150,120, 200,25);
  l3=new Label("Year :");
  l3.setBounds(50,180,100,30);
  ch=new Choice();
  ch.setBounds(150,180,200,30);
  ch.add("I-YEAR");
  ch.add("II-YEAR");
  ch.add("III-YEAR");
  ch.add("IV-YEAR");
  l4=new Label("Branch :");
```

```
    l4.setBounds(50,240,100,30);
    ls=new List(4);
    ls.setBounds(150,240,200,80);
    ls.add("CSE");
    ls.add("CSIT");
    ls.add("ECE");
    ls.add("EEE");
    ls.add("CIVIL");
    ls.add("MECH");
    l5=new Label("Subjects :");
    l5.setBounds(50,330,100,30);
    cb1=new Checkbox("C");
    cb1.setBounds(150,330,30,30);
    cb2=new Checkbox("JAVA");
    cb2.setBounds(190,330,50,30);
    cb3=new Checkbox("DBMS");
    cb3.setBounds(240,330,100,30);
    l6=new Label("Address :");
    l6.setBounds(50,360,100,30);
    ta=new TextArea();
    ta.setBounds(150,360,230,120);
    b=new Button("Submit");
    b.setBounds(150,520,100,30);
    f.add(l1);f.add(l2);f.add(l3);
    f.add(l4);f.add(l5);f.add(l6);
    f.add(t1);f.add(t2);
    f.add(ch);
    f.add(ls);
    f.add(cb1);f.add(cb2);f.add(cb3);
    f.add(ta);
    f.add(b);
}
}
```

Output:

Javac RegistrationExample.java

Java RegistrationExample


Registration Page

Name : Madhu

Regd No : 521

Year : IV-YEAR

Branch : [✓ CSE] CSIT ECE EEE CIVIL

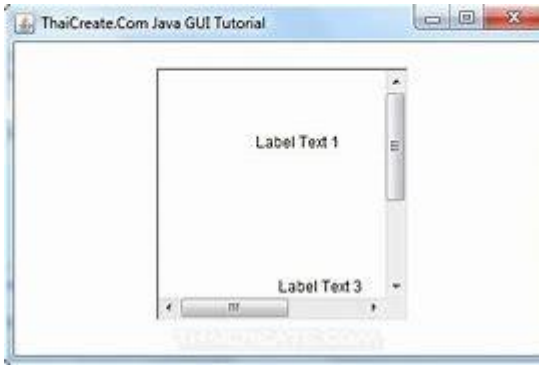
Subjects : [✓ C] [✓ JAVA] [✓ DBMS]

Address : madhu@tkrcet.com

SCROLL PANE IN AWT

A ScrollPane is a container that provides a scrollable view of its child components.

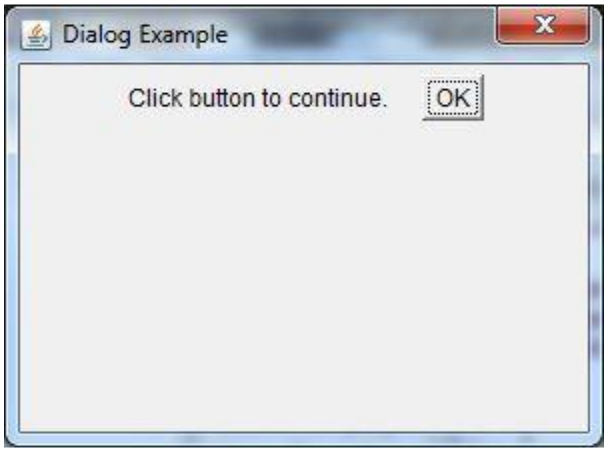
```
import java.awt.*;

public class ScrollPaneExample
 {
   public static void main(String[] args)
{
      // Create a Frame
      Frame frame = new Frame("AWT ScrollPane Example");

      // Create a ScrollPane
      ScrollPane scrollPane = new ScrollPane();

      // Create a Panel to add components
      Panel panel = new Panel();
      panel.setLayout(new GridLayout(5, 1)); // Grid layout with 5 rows and 1
column

      // Add some labels to the panel
      for (int i = 1; i <= 5; i++)
{
         panel.add(new Label("Label " + i));
      }

      // Add the panel to the ScrollPane
      scrollPane.add(panel);

      // Add the ScrollPane to the Frame
      frame.add(scrollPane);

      // Set Frame properties
      frame.setSize(300, 200);
      frame.setVisible(true);

      // Add a window listener to close the frame
      frame.addWindowListener(new java.awt.event.WindowAdapter()
{
         public void windowClosing(java.awt.event.WindowEvent windowEvent)
{
            System.exit(0);
         }
      });
   }
}
```



DIALOG IN AWT

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

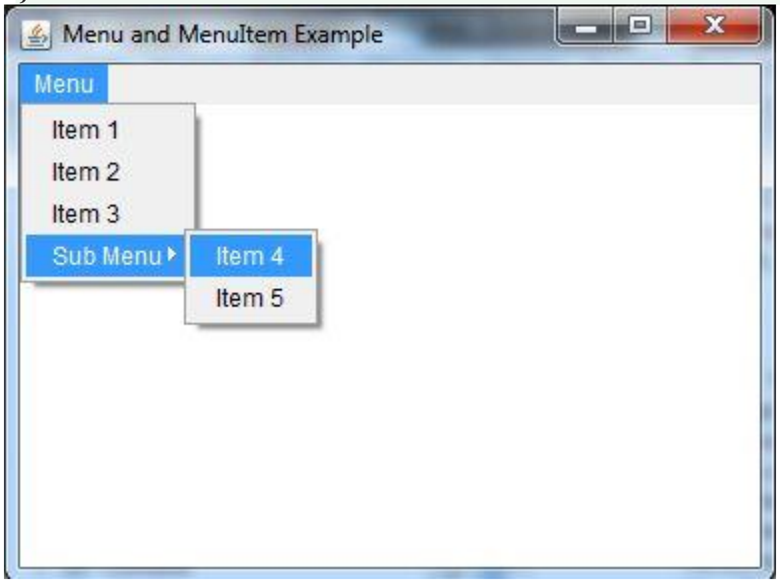
```
import java.awt.*;
import java.awt.event.*;
public class DialogExample
 {
private static Dialog d;
DialogExample()
{
Frame f= new Frame();
d = new Dialog(f , "Dialog Example", true);
d.setLayout( new FlowLayout() );
Button b = new Button ("OK");
b.addActionListener ( new ActionListener()
{
public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



MENUBAR IN AWT

Write a java program to create Menubar and add menu items to the menubar

```
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();
}
}
```

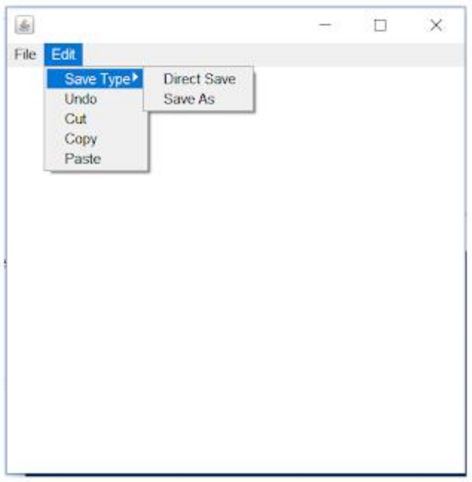


WRITE A PROGRAM USING AWT TO CREATE A MENU BAR WHERE MENUBAR CONTAINS MENU ITEMS SUCH AS FILE, EDIT, VIEW AND CREATE A SUBMENU UNDER THE FILE MENU: NEW AND OPEN

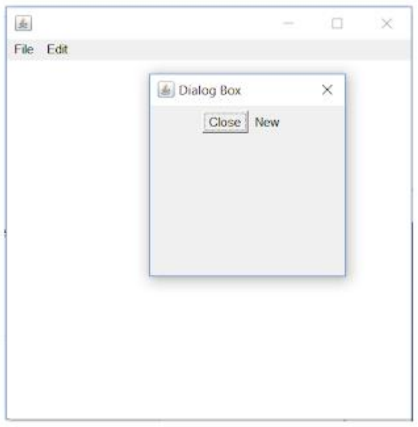
```
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.ActionEvent;

public    class    MemuDialog    extends    Frame    implements    ActionListener
,ItemListener
{
   Dialog  dialog;
   Label l;

   MemuDialog()
   {
      MenuBar mBar = new MenuBar();
      setMenuBar(mBar);

      Menu file = new Menu("File");
      MenuItem new_file = new MenuItem("New");
      MenuItem open_file = new MenuItem("Open");
      MenuItem save_file = new MenuItem("Save");
      new_file.addActionListener(this);
      open_file.addActionListener(this);
      save_file.addActionListener(this);

      file.add(new_file);
      file.add(open_file);
      file.add(save_file);
      mBar.add(file);

      Menu edit = new Menu("Edit");
      MenuItem undo_edit = new MenuItem("Undo");
      CheckboxMenuItem cut_edit = new CheckboxMenuItem("Cut");
      CheckboxMenuItem copy_edit = new CheckboxMenuItem("Copy");
      CheckboxMenuItem edit_edit = new CheckboxMenuItem("Paste");
      undo_edit.addActionListener(this);
      cut_edit.addItemListener(this);
      copy_edit.addItemListener(this);
      edit_edit.addItemListener(this);

      Menu sub = new Menu("Save Type");
      MenuItem sub1_sum = new MenuItem("Direct Save");
      MenuItem sub2_sum = new MenuItem("Save As");
      sub.add(sub1_sum);
```

```
    sub.add(sub2_sum);
    edit.add(sub);
    edit.add(undo_edit);
    edit.add(cut_edit);
    edit.add(copy_edit);
    edit.add(edit_edit);
    mBar.add(edit);

    dialog = new Dialog(this,false);
    dialog.setSize(200,200);
    dialog.setTitle("Dialog Box");

    Button b = new Button("Close");
    b.addActionListener(this);

    dialog.add(b);
    dialog.setLayout(new FlowLayout());
    l = new Label();
    dialog.add(l);
}


public void actionPerformed(ActionEvent ie)
{
    String selected_item = ie.getActionCommand();

    switch(selected_item)
    {
        case "New":  l.setText("New");
                 break;
        case "Open":  l.setText("Open");
                 break;
        case "Save":  l.setText("Save");
                 break;
        case "Undo":  l.setText("Undo");
                 break;
        case "Cut":  l.setText("Cut");
                 break;
        case "Copy":  l.setText("Copy");
                 break;
        case "Paste":  l.setText("Paste");
                 break;
        default: l.setText("Invalid Input");
    }
    dialog.setVisible(true);
    if(selected_item.equals("Close"))
    {
        dialog.dispose();
    }
}
```

```
 public void itemStateChanged(ItemEvent ie)
 {
     this.repaint();
 }

 public static void main(String[] args)
 {
     MemuDialog md = new MemuDialog();
     md.setVisible(true);
     md.setSize(400,400);
 }
}
```



Coding Atharva



Coding Atharva

EVENT HANDLING IN AWT

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

Components in Event Handling

The following are the three main components of event handling in Java:

- Events
- Event Sources
- Event Listeners/Handlers

Events

The **events** are defined as an object that describes a change in the state of a source object. Java defines a number of such Event Classes inside **java.awt.event package**. Some of the events are as follows:

- ActionEvent: Occurs when a button is clicked or an action is performed.
- MouseEvent: Triggered by mouse actions (click, drag, move).
- KeyEvent: Generated when a key is pressed, released, or typed.
- **FocusEvent:** Occurs when a component gains or loses focus.
- **ItemEvent:** Fired when an item is selected/deselected (e.g., checkbox).

Event Sources

A source is an object that generates an event. An event generation occurs when an internal state of that object changes in some way. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Some of the event sources are as follows:

- **Button:** Generates an ActionEvent when clicked.
- **CheckBox:** Generates an ItemEvent on selection change.
- **List:** Fire ActionEvent or ItemEvent on item selection.
- **Choice:** Triggers ItemEvent when an option is chosen.
- **Window:** Shows WindowEvent on close/minimize/maximize.

Event Listeners

A listener is an object that is notified when an event occurs. A Listener has two major requirements: it should be registered to one or more source objects to receive event notifications, and it must implement methods to receive and process those notifications. Java has defined a set of interfaces for receiving and processing the events under the java.awt.event package.

Some of the listeners are as follows:

ActionListener: Handles button clicks like **actionPerformed()**.

MouseListener: Handles mouse events like **mouseClicked()** and **mousePressed()**.

ItemListener: Manages ItemEvent like **checkbox** and **radio button** toggles.

KeyListener: Handles keyboard input like **keyPressed()** and **keyReleased()**.

WindowListener: Handles window operations like **windowOpened()** and **windowClosed()**.

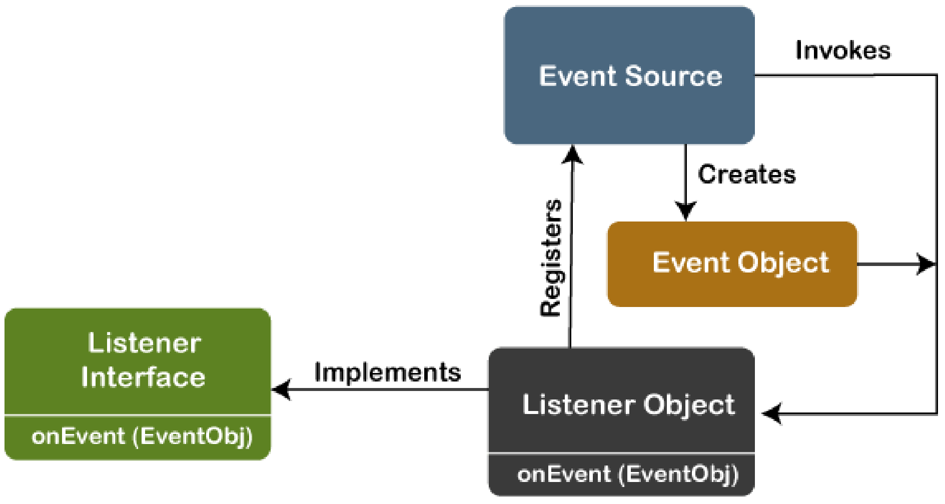
Explain Delegation Event Model in Java

The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

Event Processing in Java

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events
The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc. We can define events for any of the applied actions.

Event Sources
A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. 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()**. For the mouse event listener, the method will be called as **addMouseMotionListener()**. When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object. This process is known as event multicasting. In few cases, the event notification will only be sent to listeners that register to receive them.

Some listeners allow only one listener to register. Below is an example:

```
public void addTypeListener(TypeListener e2) throws java.util.TooManyListenersException
```

From the above syntax, the Type is the name of the event, and e2 is the event listener's reference. When the specified event occurs, it will be notified to the registered listener. This process is known as **unicasting** events.

A source should contain a method that unregisters a specific type of event from the listener if not needed. Below is an example of the method that will remove the event from the listener.

```
public void removeTypeListener(TypeListener e2?)
```

From the above syntax, the Type is an event name, and e2 is the reference of the listener. For example, to remove the keyboard listener, the **removeKeyListener()** method will be called.

The source provides the methods to add or remove listeners that generate the events. For example, the Component class contains the methods to operate on the different types of events, such as adding or removing them from the listener.

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.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

For example, the **MouseMotionListener** interface provides two methods when the mouse is dragged and moved. Any object can receive and process these events if it implements the MouseMotionListener interface.

The Delegation Model

The Delegation Model is available in Java since Java 1.1. it provides a new delegation-based event model using AWT to resolve the event problems. It provides a convenient mechanism to support complex Java programs.

Design Goals

The design goals of the event delegation model are as following:

- It is easy to learn and implement
- It supports a clean separation between application and GUI code.
- It provides robust event handling program code which is less error-prone (strong compile-time checking)

- It is Flexible, can enable different types of application models for event flow and propagation.
- It enables run-time discovery of both the component-generated events as well as observable events.
- It provides support for the backward binary compatibility with the previous model.

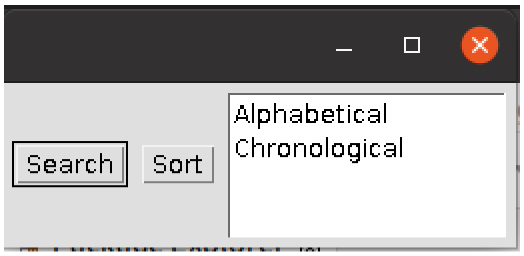
Java Program to Implement the Event Deligation Model
The below is a Java program to handle events implementing the event deligation model:

TestApp.java:

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

public class TestApp
{
public void search()
{
//  For  searching
System.out.println("Searching...");
}
public void sort() {
//  for  sorting
System.out.println("Sorting....");
}

static public void main(String args[])
 {
TestApp app = new TestApp();
GUI gui = new GUI(app);
}
}

class Command implements ActionListener
 {
static final int SEARCH = 0;
static final int SORT = 1;
int id;
TestApp app;

public Command(int id, TestApp app)
 {
this.id = id;
this.app = app;
}
public void actionPerformed(ActionEvent e)
 {
switch(id)
{
```

```
case SEARCH:
app.search();
break;
case SORT:
app.sort();
break;
}
}
}

class GUI
{

public GUI(TestApp app)
 {
Frame f = new Frame();
f.setLayout(new FlowLayout());

Command searchCmd = new Command(Command.SEARCH, app);
Command sortCmd = new Command(Command.SORT, app);

Button b;
f.add(b = new Button("Search"));
b.addActionListener(searchCmd);
f.add(b = new Button("Sort"));
b.addActionListener(sortCmd);

List l;
f.add(l = new List());
l.add("Alphabetical");
l.add("Chronological");
l.addActionListener(sortCmd);
f.pack();

f.show();
}
}
```

Output:



A simple Java AWT program that demonstrates handling mouse events such as mouse clicks, movements, and drags

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

public class MouseEventDemo extends Frame implements MouseListener,
MouseMotionListener
{

   String message = "";

   public MouseEventDemo()
{
      // Set up the frame

      setTitle("Mouse Event Demo");
      setSize(400, 300);
      setLayout(null);
      setVisible(true);

      // Add mouse listeners
      addMouseListener(this);
      addMouseMotionListener(this);

      // Close the frame on clicking the close button
      addWindowListener(new WindowAdapter()
{
         public void windowClosing(WindowEvent e)
 {
            System.exit(0);
         }
      });
   }

   // MouseListener methods
   public void mouseClicked(MouseEvent e)
 {
      message = "Mouse clicked at (" + e.getX() + ", " + e.getY() + ")";
      repaint();
   }

   public void mousePressed(MouseEvent e)
 {
      message = "Mouse pressed at (" + e.getX() + ", " + e.getY() + ")";
      repaint();
   }

   public void mouseReleased(MouseEvent e)
{
        message = "Mouse released at (" + e.getX() + ", " + e.getY() + ")";
        repaint();
    }

    public void mouseEntered(MouseEvent e)
  {
        message = "Mouse entered the frame";
        repaint();
    }

    public void mouseExited(MouseEvent e)
  {
        message = "Mouse exited the frame";
        repaint();
    }

    // MouseMotionListener methods
    public void mouseDragged(MouseEvent e) {
        message = "Mouse dragged to (" + e.getX() + ", " + e.getY() + ")";
        repaint();
    }

    public void mouseMoved(MouseEvent e) {
        message = "Mouse moved to (" + e.getX() + ", " + e.getY() + ")";
        repaint();
    }

    // Paint method to display messages
    public void paint(Graphics g)
 {
        g.drawString(message, 50, 100);
    }

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

How to Run:

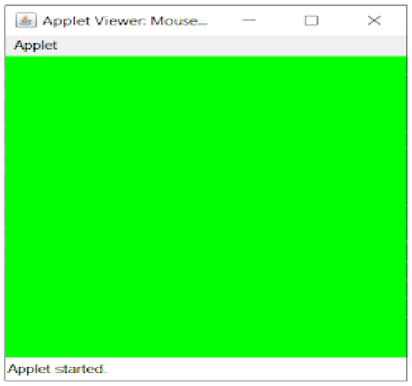
1. Save the code in a file named `MouseEventDemo.java`.
2. Compile it using `javac MouseEventDemo.java`.
3. Run it using `java MouseEventDemo`.

This program will display messages based on mouse interactions, making it a great starting point for learning AWT mouse event handling!

Write a program to change the background color of Applet when user performs event using Mouse.

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

public class MouseColor extends Applet implements MouseMotionListener
{
    public void init()
    {
        addMouseMotionListener(this);
    }

    public void mouseDragged(MouseEvent me)
    {
        setBackground(Color.red);
        repaint();
    }

    public void mouseMoved(MouseEvent me)
    {
        setBackground(Color.green);
        repaint();
    }

}
/*
    <applet code="MouseColor" width=300 height=300>
    </applet>
  */
```

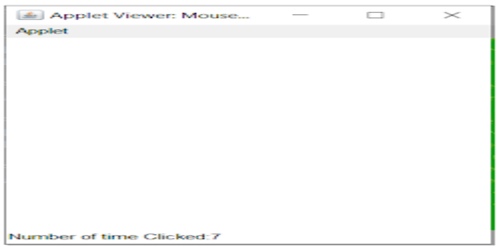
Output:



2) Write a program to count the number of clicks performed by the user in a applet?

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

public class MouseCount extends Applet implements MouseListener
{
    int count = 0;
    public void init()
    {
        addMouseListener(this);
    }

    public void mouseClicked(MouseEvent me)
    {
        count++;
        showStatus("Number of time Clicked:"+count);
        repaint();
    }

    public void mouseEntered(MouseEvent me)
    {
    }
    public void mouseExited(MouseEvent me)
    {
    }
    public void mousePressed(MouseEvent me)
    {
    }
    public void mouseReleased(MouseEvent me)
    {
    }
}
/*
    <applet code="MouseCount" width=300 height=300>
    </applet>
*/
```

Output:



```
Number of time Clicked:7
```

KEY EVENT IN AWT

A simple Java program using AWT to demonstrate handling key events. This program listens for key presses and displays the key pressed in a window.

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

public class KeyEventDemo extends Frame implements KeyListener {

   private String message = "";

   public KeyEventDemo()
 {
      // Set up the frame
      setTitle("Key Event Demo");
      setSize(400, 200);
      setLayout(null);
      setVisible(true);

      // Add KeyListener to the frame
      addKeyListener(this);

      // Close the frame on clicking the close button
      addWindowListener(new WindowAdapter()
{
         public void windowClosing(WindowEvent e)
{
            System.exit(0);
         }
      });
   }

   @Override
   public void keyPressed(KeyEvent e)
 {
      message = "Key Pressed: " + e.getKeyChar();
      repaint();
   }

   @Override
   public void keyReleased(KeyEvent e)
{
      message = "Key Released: " + e.getKeyChar();
      repaint();
   }

   @Override
   public void keyTyped(KeyEvent e) {
      message = "Key Typed: " + e.getKeyChar();
      repaint();
```

```
    }

    @Override
    public void paint(Graphics g)
   {
        g.drawString(message, 50, 100);
    }

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

How to Run:

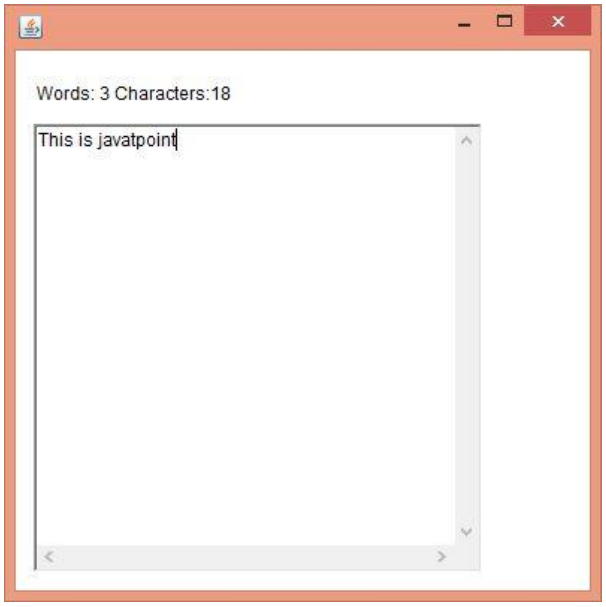
Save the code in a file named KeyEventDemo.java.
Compile it using javac KeyEventDemo.java.
Run it using java KeyEventDemo.
Press keys in the window to see the events displayed.

Write a java program using key event to Count Words & Characters

In the following example, we are printing the count of words and characters of the string. Here, the string is fetched from the TextArea and uses the KeyReleased() method of KeyListener interface.

KeyListenerExample2.java

```
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;
// class which inherits Frame class and implements KeyListener interface
public class KeyListenerExample2 extends Frame implements KeyListener
{
// object of Label and TextArea
Label l;
TextArea area;
// class constructor
KeyListenerExample2()
{
// creating the label
l = new Label();
// setting the location of label
l.setBounds (20, 50, 200, 20);
// creating the text area
area = new TextArea();
// setting location of text area
area.setBounds (20, 80, 300, 300);
// adding KeyListener to the text area
area.addKeyListener(this);
// adding label and text area to frame
add(l);
add(area);
// setting size, layout and visibility of frame
setSize (400, 400);
setLayout (null);
setVisible (true);
}
// even if we do not define the interface methods, we need to override them
public void keyPressed(KeyEvent e)
 {
}
// overriding the keyReleased() method of KeyListener interface
public void keyReleased (KeyEvent e)
 {
// defining a string which is fetched by the getText() method of TextArea class
String text = area.getText();
// splitting the string in words
String words[] = text.split ("\\s");
// printing the number of words and characters of the string
l.setText ("Words: " + words.length + " Characters:" + text.length());
}
public void keyTyped(KeyEvent e)
{
}
// main method
public static void main(String[] args)
{
new KeyListenerExample2();
}
}
```
Output:



```
Words: 3 Characters:18

This is javatpoint
```

ADAPTER CLASSES

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*.

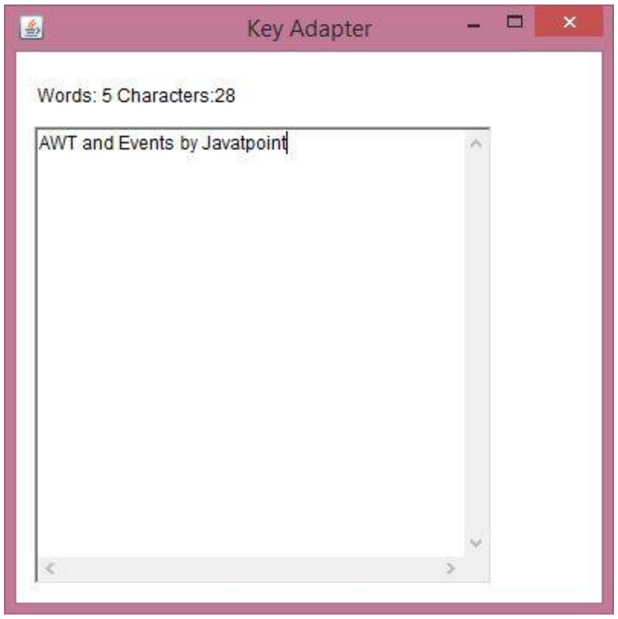
Java.awt.event Adapter classes

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

KeyAdapterExample.java

```
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;
// class which inherits the KeyAdapter class
public class KeyAdapterExample extends KeyAdapter
{
// creating objects of Label, TextArea and Frame
    Label l;
    TextArea area;
    Frame f;
// class constructor
    KeyAdapterExample()
{
// creating the Frame with title
        f = new Frame ("Key Adapter");
// creating the Label
        l = new Label();
// setting the location of label
        l.setBounds (20, 50, 200, 20);
// creating the text area
        area = new TextArea();
// setting the location of text area
        area.setBounds (20, 80, 300, 300);
// adding KeyListener to text area
        area.addKeyListener(this);
 // adding the label and text area to frame
        f.add(l);
f.add(area);
// setting the size, layout and visibility of frame
        f.setSize (400, 400);
        f.setLayout (null);
        f.setVisible (true);
    }
// overriding the keyReleased() method
    public void keyReleased (KeyEvent e)
 {
// creating the String object to get the text fromTextArea
        String text = area.getText();
// splitting the String into words
        String words[] = text.split ("\\s");
// setting the label text to print the number of words and characters of given
 string
        l.setText ("Words: " + words.length + " Characters:" + text.length());
    }
  // main method
    public static void main(String[] args)
{
        new KeyAdapterExample();
    }
}
```

Output:



```
Words: 5 Characters:28
AWT and Events by Javatpoint
```