THIRD YEAR DIPLOMA

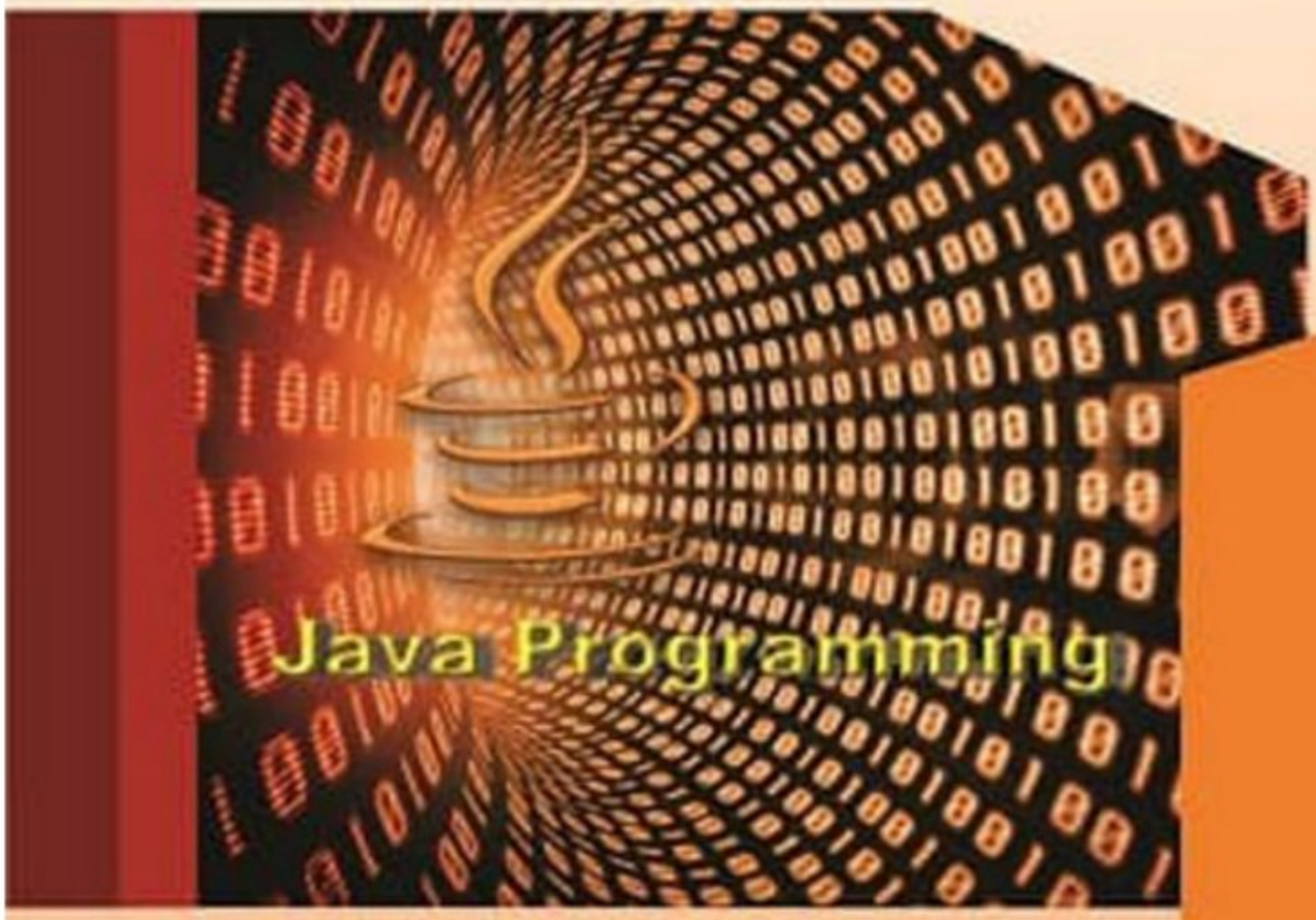
ENGINEERING AND TECHNOLOGY

COMPUTER ENGINEERING AND
INFORMATION TECHNOLOGY GROUP

SEMESTER-V

MSBTE's I SCHEME

ADVANCED JAVA PROGRAMMING



Dr. MEENAKSHI A. THALOR
MAHESH GURUNANI
PRASHANT SOMWANSHI
RAJESH YEMUL

NIRALI PRAKASHAN

A Text Book Of

ADVANCED JAVA PROGRAMMING

(22517)

Semester - V

THIRD YEAR DIPLOMA IN COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY GROUP

As Per MSBTE's 'I' Scheme Syllabus

Dr. Meenakshi A. Thalor
Ph.D. (Computer Engineering), M.E. (I.T.), B. Tech. (I.T.)
Head of Dept. & Associate Professor, I.T.
AISSMS Institute of Information Technology
Pune

Mahesh Gurunani
M.E. (I.T.), SJCP, ISO Auditor
Assistant Professor, Dept. of Post Graduation (C.S.)
Thakur College of Science & Commerce
Kandivali (E), Mumbai

Prashant Somwanshi
M.E. (Computer Engineering)
Lecturer, Dept. of Computer Engineering
JSPM & TSSM Group of Institutes,
Bhivarabai Sawant College of Engineering & Research Polytechnic, Narhe, Pune

Rajesh Yemul
B.E. (Comp.), MISTE
Lecturer, in Computer Technology Department,
STES's Sou. Venutai Chavan Polytechnic,
Vadgaon (Bk.) Pune

Price ₹ 160.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4540

ADVANCED JAVA PROGRAMMING **ISBN 978-93-89108-11-8**

First Edition : June 2019

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar
Off J.M. Road, PUNE – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041
Mobile No. 9404233041/9850046517

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
(For orders within Pune) Tel : (020) 2445 2044, Fax : (020) 2445 1538; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan : S. No. 28/27, Dhayari, Near Asian College Pune 411041
(For orders outside Pune) Tel : (020) 24690204 Fax : (020) 24690316; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Preface ...

We take an opportunity to present this book entitled as "**Advanced Java Programming**" to the students of Fifth Semester (Third Year Diploma in Computer Engineering and Information Technology Group). The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the Revised Syllabus (I-Scheme) prepared by Maharashtra State Board of Technical Education (MSBTE).

This book contains six constructive chapters.

- First chapter gives an introduction to Abstract Windowing Toolkit (AWT). This chapter also includes AWT controls and layout managers.
- Second chapter focuses on Swing. Swing components and advanced Swing components are described in this chapter.
- Third chapter covers event handling.
- Fourth chapter gives basic concepts networking.
- Fifth chapter gives an idea of interacting with database in Advanced Java.
- Servlets is covered in Sixth chapter.

A special words of thank to Shri. Dineshbhai Furia, Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Amar Salunkhe and Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takale, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

We welcome constructive suggestions from our colleagues and students.

Authors

■■■

M

Syllabus ...

1. Abstract Windowing Toolkit (AWT)

1.1 Component, Container, Window, Frame, Panel.
1.2 Creating Windowed Programs and Applets.
1.3 AWT Controls and Layout Managers: Use of AWT Controls: Labels, Buttons, Checkbox, Checkbox Group, Scroll Bars, Text Field, Text Area.
1.4 Use of Layout Managers: FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout, Menu Bars, Menus, Dialog Boxes, File Dialog.

2. Swings

2.1 Introduction to Swing: Swing Features, Difference between AWT and Swing.
2.2 Swing Components: JApplet, Icons and Labels, Text Fields, Combo Boxes.
2.3 Buttons: The JButton, Check Boxes, Radio Buttons.
2.4 Advanced Swing Components: Tabbed Panes, Scroll Panes, Trees, Tables, Progressbar, Tooltips.
2.5 MVC Architecture.

3. Event Handling

3.1 The Delegation Event Model: Event Sources, Event Listeners.
3.2 Event Classes: The Action Event Class, The Item Event Class, The Key Event Class, The Mouse Event Class, The Text Event Class, The Window Event Class.
3.3 Adapter Classes.
3.4 Inner Classes.
3.5 Event Listener Interfaces: ActionListener Interface, ItemListener Interface, KeyListener Interface, MouseListener Interface, MouseMotion Interface, TextListener Interface, WindowsListener Interface.

4. Networking Basics

4.1 Socket Overview: Client/Server, Reserved Sockets, Proxy Servers, Internet Addressing.
4.2 Java and the Net: The Networking Classes and Interfaces.
4.3 InetAddress: Factory Methods, Instance Methods.
4.4 TCP/IP Client Sockets: Whois
4.5 URL: Format, The URI Class.
4.6 URLConnection: TCP/IP Server Sockets.
4.7 Datagrams: Datagram Packet, Datagram Server and Client.

5. Interacting with Database

5.1 Introduction to JDBC, ODBC.
5.2 JDBC Architecture: Two Tier and Three Tier Models.
5.3 Types of JDBC Drivers.
5.4 Driver Interfaces and Driver Manager Class: Connection Interface Statement Interface, Prepared Statement Interface, ResultSet Interface.
5.5 The Essential JDBC Program.

6. Servlets

6.1 The Life Cycle of a Servlet.
6.2 Creating Simple Servlet: The Servlet API, javax.servlet Package, Servlet Interface, ServletConfig Interface, ServletContext Interface, ServletRequest Interface, ServletResponse Interface, GenericServlet Class.
6.3 The javax.servlet.http, Package: HttpServletRequest Interface, HttpServletResponse Interface, HttpSession Interface, Cookie Class, HttpServlet Class, HttpSessionEvent Class, HttpSessionBindingEvent Class.
6.4 Handling HTTP Requests and Responses, Handling HTTP GET Requests, Handling HTTP POST Requests.
6.5 Cookies and Session Tracking.

■■■

Contents ...

■■■

1...

Abstract Windowing Toolkit (AWT)

Chapter Outcomes...

- Develop Graphical User Interface (GUI) programs using AWT components for the given problem.
- Create Frame window with the specified AWT components.
- Arrange the GUI components using specified Layout Manager.
- Develop a program using Menu and Dialog boxes for the given problem.

Learning Objectives...

- To understand Basic Concepts of AWT
- To learn Component, Container, Window, Frame, Panel etc.
- To study AWT controls such as Labels, Buttons, Checkbox, Scroll Bars, Text Field, Text Area etc.
- To understand Layout Managers with its Types
- To learn Concepts like Menus and Dialog Boxes

1.0 INTRODUCTION

- Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in Java.
- Java AWT components are platform-dependent i.e., components are displayed according to the view of operating system. AWT is heavy weight as its components uses the resources of system.
- Java graphics APIs (AWT) provide a huge set of reusable GUI components, such as button, text field, label, panel and frame for building GUI applications.
- AWT consists of twelve (12) packages but only two packages java.awt and java.awt.event are commonly used.
  1. The **java.awt package** contains the core AWT graphics classes:
     - GUI Component classes such as , TextField, and Label.
     - GUI Container classes such as Frame, Panel, Dialog and ScrollPane.
     - Layout managers such as FlowLayout, BorderLayout and GridLayout.
     - Custom graphics classes such as Graphics, Color and Font.
  2. The **java.awt.event package** supports event handling:
     - Event classes such as ActionEvent, MouseEvent, KeyEvent and WindowEvent.
     - Event Listener Interfaces such as ActionListener, MouseListener, KeyListener and WindowListener.
     - Event Listener Adapter classes such as MouseAdapter, KeyAdapter, and WindowAdapter.

AWT Classes:

- The AWT classes are contained in the java.awt package. It is one of Java's largest packages.
- The commonly used of Java AWT classes are given below:

Sr. No.	Class	Description
1.	AWTEvent	Encapsulates AWT events.
2.	BorderLayout	The border layout manager. Border layouts use five components i.e. North, South, East, West and Centre.
3.	Button	Creates a push button control.
4.	CardLayout	The card layout manager. Card layouts emulate indexcards. Only the one on top is showing.
5.	Checkbox	Creates a check box control.
6.	CheckboxGroup	Creates a group of check box controls.
7.	Choice	Creates a pop-up list.
8.	Container	A subclass of Component that can hold other components.
9.	Dialog	Creates a dialog window.
10.	Event	Encapsulates events.
11.	FileDialog	Creates a window from which a file can be selected.
12.	FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
13.	Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
14.	TextArea	Creates a multiline edit control.
15.	TextField	Creates a single-line edit control.
16.	Window	Creates a window with no frame, no menu bar.

1.1 COMPONENT, CONTAINER, WINDOW, FRAME AND PANEL

- The Java AWT contains the fundamental classes used for constructing GUIs.
- Fig. 1.1 shows Java AWT hierarchy.
- Component, Container, Panel, Window, Frame are the fundamental elements in AWT.
- 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.
- Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data. A Frame is a top-level window with a title and a border.

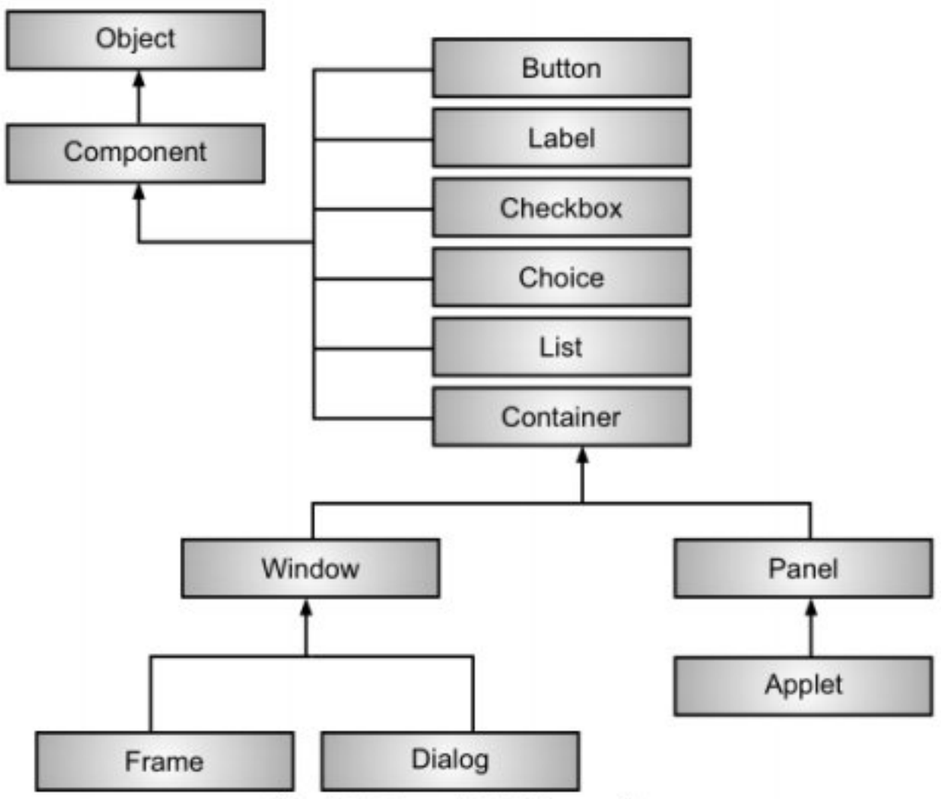


Fig. 1.1: Java AWT Hierarchy

- Container is a component that can contain other components. Panel provides space in which an application can attach any other components, including other panels.

- Commonly used elements in AWT Hierarchy class are explained as follows:

1. Object:

- The Object class is the top most class and parent of all the classes in java by default. Every class in Java is directly or indirectly derived from the Object class.
- The widely used **methods of Object class** are given below:

Sr. No.	Methods	Description
1.	`String toString()`	This method returns the string representation of this object.
2.	`void finalize()throws Throwable`	This method is invoked by the garbage collector before object is being garbage collected.

2. Component:

- At the top of the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component.
- All User Interface (UI) elements that are displayed on the screen and that interact with the user are subclasses of Component.
- A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.
- The widely used **methods of Component class** are given below:

Sr. No.	Methods	Description
1.	`void add(Component c)`	Add a component on another component.
2.	`void setSize(int width,int height)`	Sets size of the component.
3.	`void setLayout(LayoutManager m)`	Sets the layout manager for the component.
4.	`void setVisible(boolean b)`	Sets the visibility of the component. It is by default false.
5.	`void remove(Component c)`	Remove a component.
6.	`void setBounds(int x, int y, int width, int height)`	Used to set the location and size of a single component, and is only useful if null layout is used by the container that holds this component.

3. Container:

- The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it.
- Other Container objects can be stored inside of a Container (since, they are themselves instances of Component). This makes for a multileveled containment system.
- A container is responsible for laying out (positioning) any components that it contains. It does this through the use of various layout managers.
- This class inherits methods from the java.awt.Component and java.lang.Object.

Sr. No.	Methods	Description
1.	`void setFont(Font f)`	Sets the font of this container.
2.	`void setLayout(LayoutManager mgr)`	Sets the layout manager for this container.

4. Panel:

- The Panel class is a concrete subclass of Container. It doesn't add any new methods; it simply implements Container. A Panel may be thought of a concrete screen component.
- In essence, a Panel is a window that does not contain a title bar, menu bar, or border. This is the reason we don't see a title bar, menu bar, or border when an applet is run inside a browser. Other components can be added to a Panel object by its add( ) method (inherited from Container).

5. Window:

- The Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop.
- Generally, we won't create Window objects directly. Instead, we will use a subclass of Window called Frame.

6. Frame:

- Frame encapsulates what is commonly thought of as a 'window'. It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners.
- The widely used **methods of Frame class** are given below:

Sr. No.	Methods	Description
1.	`void setTitle(String title)`	Sets the title for this frame to the specified string.
2.	`void setBackground(Color bgColor)`	Sets the background color of this window.

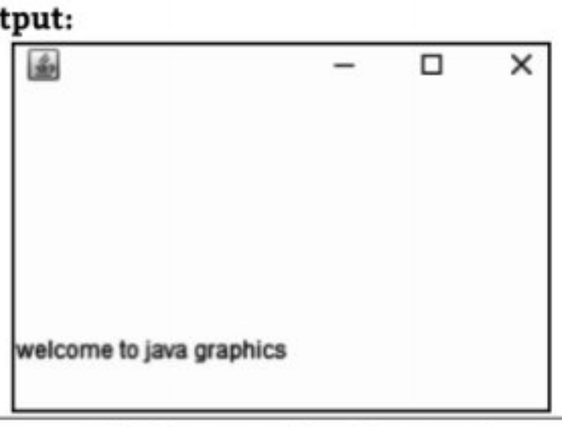
1.1.1 Frame Class

- We can create stand-alone AWT-based applications. To do this, simply we need to create an instance of the window or windows inside main( ). For example, the following program creates a simple frame window.
- A Frame provides the "main window" for the GUI application, which has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.
- Frame is a AWT class in java that creates a top-level window with a title and a border.
- Two of **Frame class's constructors** are:
  1. `Frame()`: The first form creates a standard window that does not contain a title.
  2. `Frame(String title)`: The second form creates a window with the title specified by title.
- This class inherits methods from the following classes as per AWT class hierarchy, (Fig. 1.1).
  - `java.awt.Window`
  - `java.awt.Container`
  - `java.awt.Component`
  - `java.lang.Object`
- There are two ways to create a Frame through program. They are:
  1. By Instantiating Frame class, and
  2. By extending Frame class.

Program 1.1: By instantiating Frame class.

```
import java.awt.*;
public class Testawt
{
   Testawt()
   {
      Frame fm=new Frame();
      Label lb = new Label("welcome to java graphics");
      lb.setBounds(10,10,100,50);
      fm.add(lb);
      fm.setSize(300, 300);
      fm.setVisible(true);
 }
 public static void main(String args[])
 {
      Testawt ta = new Testawt();
 }
}
```

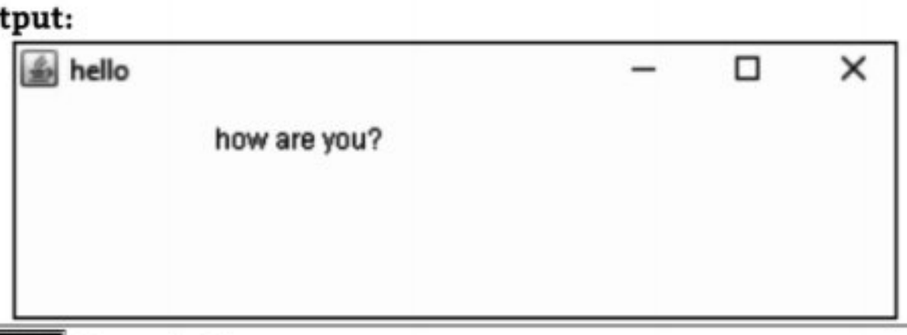
Output:



Program 1.2: By extending Frame class.

```
import java.awt.*;
public class SimpleFrame extends Frame
{
 SimpleFrame()
 {
   setLayout(null);
   setVisible(true);
   setSize(500,500);
   setTitle("hello");
 }
 public void paint(Graphics g)
 {
   g.drawString("how are you?",100,50);
 }
 public static void main(String args[])
 {
   SimpleFrame f=new SimpleFrame();
 }
}
```

Output:



1.1.2 Panel Class

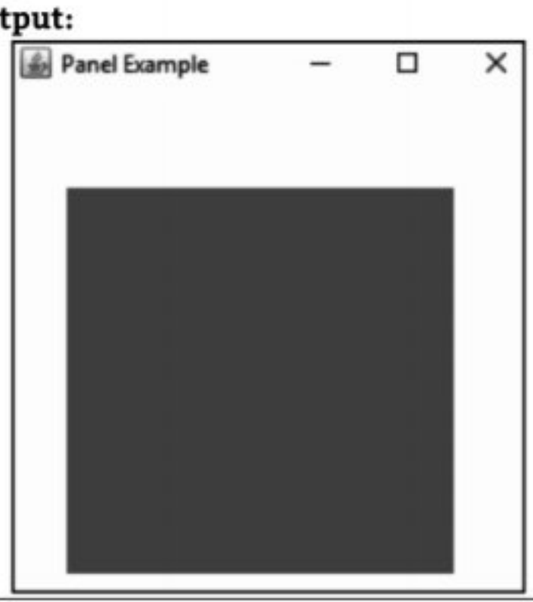
- The class Panel is the simplest container class.
- It provides space in which an application can attach any other component, including other panels. It uses FlowLayout as default layout manager.

Program 1.3: Program for panel.

```
import java.awt.*;
public class SimplePanel extends Frame
{
 SimplePanel()
 {
   Frame f= new Frame("Panel Example");
```

```
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        SimplePanel p=new SimplePanel();
    }
}
```

Output:



1.2 CREATING WINDOWED PROGRAMS AND APPLETS

- In previous section we have seen creation of frame and panel. The frame created by the programmer and displayed on the monitor comes with a title bar and three icons (buttons) on the title bar – Minimize, Maximize and Close. The first two works implicitly and the Close does not work and requires extra code to close the frame.
- So to solve this problem one solution is to create a Frame Window within an applet window. To do applet programming we need to extend Applet class.
- An applet is not a standalone application. It is a mini program invoked within a larger program, such as an applet viewer or a Java-enabled World Wide Web (WWW) browser. An applet depends on the viewer or browser to provide a context in which to run.
- Following are the methods of Applet class:
  1. **init():** In this method, the applet object is created by the browser. Because this method is called before all the other methods, programmer can utilize this method to instantiate objects, initialize variables, setting background and foreground colors in GUI etc.
  2. **start():** To make the applet active, the init() method calls start() method. In start() method, applet becomes active and thereby eligible for processor time.
  3. **paint():** This method takes a java.awt.Graphics object as parameter. This class includes many methods of drawing necessary to draw on the applet window. This is the place where the programmer can write his code of what he expects from applet like animation etc
  4. **stop():** In this method the applet becomes temporarily inactive. An applet can come any number of times into this method in its life cycle and can go back to the active state (paint() method) whenever would like. It is the best place to have cleanup code. It is equivalent to the blocked state of the thread.

5. **destroy():** This method is called just before an applet object is garbage collected. This is the end of the life cycle of applet. It is the best place to have cleanup code. It is equivalent to the dead state of the thread.

- The following program creates two classes named as SampleFrame and AppletFrame. SampleFrame class extends the Frame class and AppletFrame class extends the Applet class.
- A window of SampleFrame is instantiated within the init() method of AppletFrame. The start( ) and stop( ) method show and hide the child window, respectively. It also causes the child window to be shown when the browser returns to the applet.

Program 1.4: Program for Frame in Applet.

```
import java.applet.Applet;
import java.awt.Frame;
import java.awt.Graphics;
public class AppletFrame extends Applet
{
   SampleFrame f;
   public void init() {
      f = new SampleFrame("A Frame Window");
      f.setSize(250, 250);
      f.setVisible(true);
   }
   public void start() {
      f.setVisible(true);
   }
   public void stop() {
      f.setVisible(false);
   }
   public void paint(Graphics g) {
      g.drawString("This is in applet window", 10, 20);
   }
}
class SampleFrame extends Frame {
   SampleFrame(String title) {
      super(title);
   }
   public void paint(Graphics g) {
      g.drawString("This is in frame window", 10, 40);
   }
}
```

Output:

1.3 AWT CONTROLS AND LAYOUT MANAGERS

- Controls are components that allow a user to interact with his/her application in various ways, for example a commonly used control is the push button.
- A layout manager automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them.
- In addition to the controls, a frame window can also include a standard-style menu bar. Each entry in a menu bar activates a drop-down menu of options from which the user can choose. A menu bar is always positioned at the top of a window.
- In this section we will use the setBounds() method of component class to position the component on the Frame Before using that method need to set the Layout of the container to null.

1.3.1 AWT Controls

- Controls are the components that allow a user to interact with java application in various ways. Controls are the most common ways that users give instructions to Java programs.
- The AWT supports various types of controls such as Labels, Push Buttons, Check Boxes, Choice Lists, Lists, Scroll Bars, Text Area, Text Field and so on.

1.3.1.1 Labels

- The easiest control to use in Java is a label. A label displays a single line of read-only text. A label is an object of type Label, and it contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.

Constructors of Label Class:

Sr. No.	Constructors	Description
1.	`Label(String strLabel)`	Creates a blank/empty label.
2.	`Label(String strLabel)`	Creates a label that contains the string specified by str. This string is left-justified.
3.	`Label(String strLabel, int alignment)`	Construct a Label with the given text String, of the text alignment. Three alignments are Label. LEFT, Label.RIGHT and Label.CENTER.
4.	`Label(String strLabel)`	Construct a Label with the given text String in default of left-aligned.
5.	`Label()`	Construct an initially empty Label.

Methods of Label Class:

Sr. No.	Methods	Description
1.	`String getText()`	The getText() method can be used to read the Label's text.
2.	`void setText(String strLabel)`	The setText() method can be used to modify the Label's text.
3.	`int getAlignment()`	The getAlignment() method can be used to retrieve the alignment of the text.
4.	`void setAlignment(int alignment)`	The setAlignment() method can be used to modify the alignment of the text.

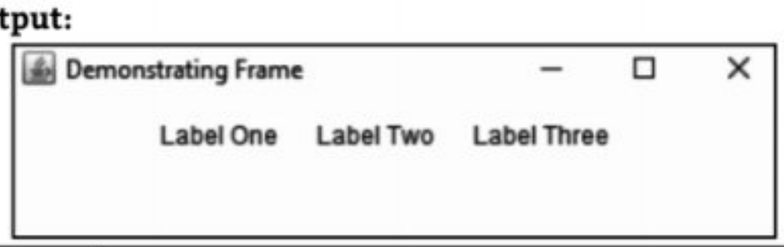
- Following program creates three labels and adds them to frame window.

Program 1.5: Program to demonstrate label.

```
import java.awt.*;
class MyFrame extends Frame
{
   MyFrame(String s)
   {
```

```
        super(s);
        setLayout(null);
        setVisible(true);
        setSize(500,300);
        Label one = new Label("Label One");
        Label two = new Label("Label Two");
        Label three = new Label("Label Three");
        //set the position of controls
        one.setBounds(50,50,100,100);
        two.setBounds(150,50,100,100);
        three.setBounds(250,50,100,100);
        add(one);
        add( two);
        add( three);
    }
    public static void main(String[] args)
    {
       MyFrame f= new MyFrame("Demonstrating Frame");
    }
    }
```

Output:



1.3.1.2 Buttons

- Button class is used to create a push button control, which can generate an ActionEvent when it is clicked or presses.
- A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button.

Constructors of Button Class:

Sr. No.	Constructors	Description
1.	`Button(String buttonLabel)`	Construct a Button with the given label.
2.	`Button()`	Construct a Button with empty label.

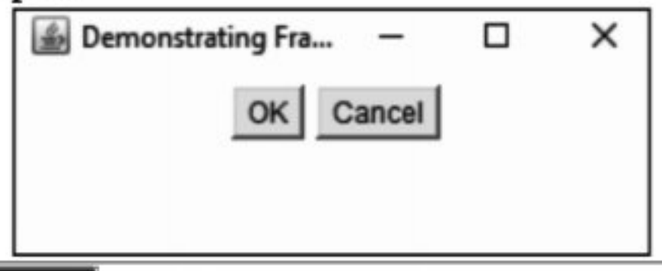
Methods of Button Class:

Sr. No.	Methods	Description
1.	`String getLabel()`	Get the label of this Button instance.
2.	`void setLabel(String buttonLabel)`	Set the label of this Button instance.
3.	`void setEnable(boolean enable)`	Enable or disable this Button. Disabled Button cannot be clicked.
4.	`void addActionListener(ActionListener l)`	Adds the specified action listener to receive action events from this button.
5.	`void removeActionListener(ActionListener l)`	Removes the specified action listener so that it no longer receives action events from this button.
6.	`String getActionCommand()`	Returns the command name of the action event fired by this button.
7.	`void setActionCommand(String command)`	Sets the command name for the action event fired by this button.

Program 1.6: Program to demonstrate buttons.

```
import java.awt.*;
class MyFrame extends Frame
{
    Button ok, cancel;
    MyFrame(String s) {
        super(s);
        setLayout(null);
        setVisible(true);
        setSize(500, 300);
        ok = new Button("OK");
        cancel = new Button("Cancel");
        ok.setBounds(50,50,50,50);
        cancel.setBounds(120,50,100,50);
        add(ok);
        add(cancel);
    }
    public static void main(String[] args) {
        MyFrame f = new MyFrame("Demonstrating Frame");
    }
}
```

Output:



1.3.1.3 CheckBox

- A check box is a control that is used to turn an option on (true) or off (false). It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents. We change the state of a check box by clicking on it.
- Check boxes can be used individually or as part of a group. Check boxes are objects of the Checkbox class.

Constructors of CheckBox Class:

Sr. No.	Constructors	Description
1.	`Checkbox()`	Creates a check box with an empty string for its label.
2.	`Checkbox(String label)`	Creates a check box with the specified label.
3.	`Checkbox(String label, boolean state)`	Creates a check box with the specified label and sets the specified state.
4.	`Checkbox(String label, boolean state, CheckboxGroup group)`	Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
5.	`Checkbox(String label, CheckboxGroup group, boolean state)`	Creates a check box with the specified label, in the specified check box group, and set to the specified state.

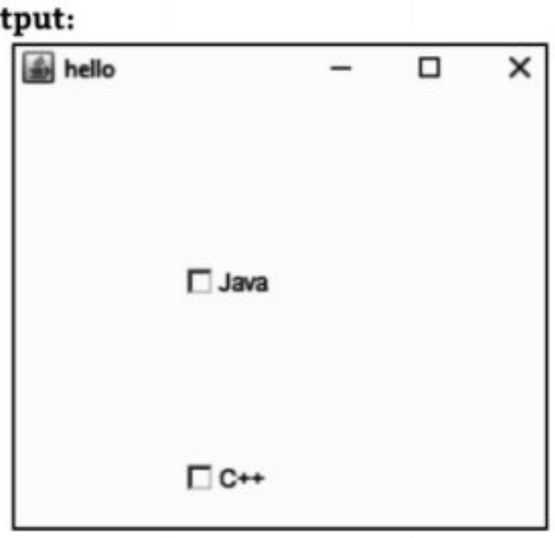
Methods of CheckBox Class:

Sr. No.	Methods	Description
1.	`String getLabel()`	Gets the label of this check box.
2.	`boolean getState()`	Determines whether this check box is in the on or off state.
3.	`void setLabel(String label)`	Sets this check box's label to be the string argument.
4.	`void setState(boolean state)`	Sets the state of this check box to the specified state.
5.	`void addItemListener(ItemListener l)`	Adds the specified item listener to receive item events from this check box.
6.	`void removeItemListener(ItemListener l)`	Removes the specified item listener so that the item listener no longer receives item events from this check box.

Program 1.7: Program to demonstrate checkboxes.

```
import java.awt.*;
public class MyFrame extends Frame
{
    Checkbox cb1,cb2;
    String msg="";
    MyFrame()
    {
        setLayout(null);
        setVisible(true) ;
        setSize(300,300);
        setTitle("hello");
        cb1=new Checkbox("Java",false);
        cb1.setBounds(100,100,50,50);
        add(cb1);
        cb2=new Checkbox("C++",false);
        cb2.setBounds(100,200,50,50);
        add(cb2);
    }
    public static void main(String args[])
    {
    MyFrame f=new MyFrame();
    }
}
```

Output:



1.3.1.4 CheckBoxGroup (Radio Button)

- The CheckBoxGroup class is used to group the set of check box.
- It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time.
- These check boxes are often called radio buttons, because they act like the station selector on a car radio where only one station can be selected at any onetime.
- For creating a set of mutually exclusive check boxes, we must first define the group to which they will belong and then specify that group when we construct the check boxes.

Constructors of CheckBoxGroup Class:

Sr. No.	Constructors	Description
1.	`CheckboxGroup()`	Creates a new instance of CheckboxGroup.

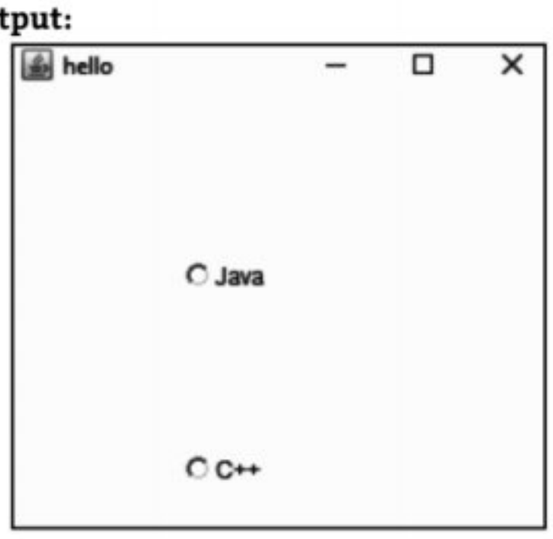
Methods of CheckBoxGroup Class:

Sr. No.	Methods	Description
1.	`Checkbox getSelectedCheckbox()`	Gets the Checkbox component group currently selected.
2.	`void setSelectedCheckbox(Checkbox)`	Determines which Checkbox component selected group.

Program 1.8: Program to demonstrate Checkbox group.

```
import java.awt.*;
class MyFrame extends Frame {
    Checkbox cb1, cb2;
    CheckboxGroup cbg;
    String msg = "";
    MyFrame(String s) {
        setLayout(null);
        setVisible(true);
        setSize(300, 300);
        setTitle("hello");
        cbg = new CheckboxGroup();
        cb1 = new Checkbox("Java", false, cbg);
        cb1.setBounds(100, 100, 50, 50);
        add(cb1);
        cb2 = new Checkbox("C++", false, cbg);
        cb2.setBounds(100, 200, 50, 50);
        add(cb2);
    }
    public static void main(String[] args) {
        MyFrame f = new MyFrame();
    }
}
```

Output:



1.3.1.5 Choice Controls

- The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu.
- When a Choice component is inactive, it takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
- Each item in the list is a string that appears as a left justified label in the order it is added to the Choice object.

Constructors of Choice Control Class:

Sr. No.	Constructors	Description
1.	`Choice()`	Creates an empty choice box.

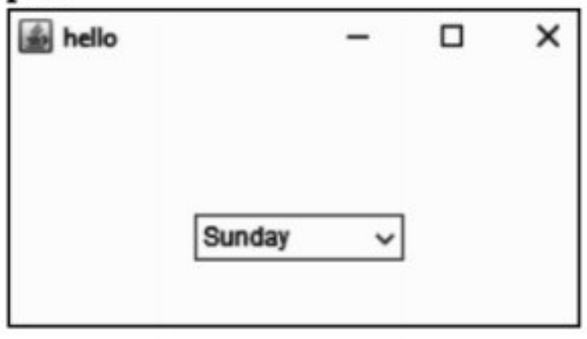
Methods of Choice Control Class:

Sr. No.	Methods	Description
1.	`void addItem(String item)`	Use addItem() to populate.
2.	`int countItems()`	Useful for interrogating the Choice contents.
3.	`String getItem(int index)`	Get an item by index from 0 to countItems() – 1.
4.	`int getSelectedIndex()`	Tells which item is selected by number.
5.	`String getSelectedItem()`	Returns the selected item's text.
6.	`void select(int pos)`	Make a particular cell the default.
7.	`void select(String str)`	Highlight the first cell with text equal to str.

Program 1.9: Program to demonstrate choice controls.

```
import java.awt.*;
class MyFrame extends Frame
{
    String msg = "";
    Choice c;
    MyFrame(String s)
    {
        setLayout(null);
        setVisible(true);
        setSize(300, 300);
        setTitle("hello");
        c = new Choice();
        c.add("Sunday");
        c.add("Monday");
        c.add("Tuesday");
        c.add("Wednesday");
        c.setBounds(100, 100, 100, 100);
        add(c);
    }
    public static void main(String[] args)
    {
        MyFrame f = new MyFrame();
    }
}
```

Output:



1.3.1.6 Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible Window. It can also be created to allow multiple selections.

Constructors of Lists Class:

Sr. No.	Constructors	Description
1.	`List()`	Creates a new scrolling list.
2.	`List(int rows)`	Creates a new scrolling list initialized with the specified number of visible lines.
3.	`List(int rows, boolean multipleMode)`	Creates a new scrolling list initialized to display the specified number of rows.

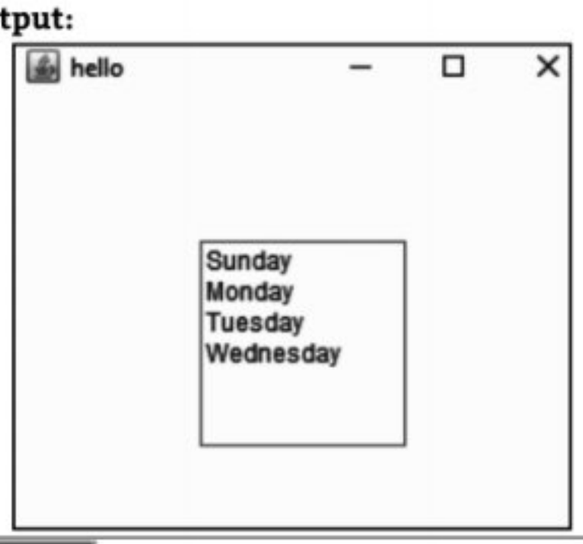
Methods of Lists Class:

Sr. No.	Methods	Description
1.	`void add(String item)`	Adds the specified item to the end of scrolling list.
2.	`void add(String item, int index)`	Adds the specified item to the scrolling list at the position indicated by the index.
3.	`void addActionListener(ActionListener l)`	Adds the specified action listener to receive action events from this list.
4.	`void addItemListener (ItemListener l)`	Adds the specified item listener to receive item events from this list.
5.	`int[] getSelectedIndexes()`	Gets the selected indexes on the list.
6.	`int getSelectedIndex()`	Gets the index of the selected item on the list.
7.	`String getSelectedItem()`	Gets the selected item on this scrolling list.
8.	`String[] getSelectedItems()`	Gets the selected items on this scrolling list.
9.	`void clear()`	Empty the list.

Program 1.10: Program to demonstrate lists.

```
import java.awt.*;
class MyFrame extends Frame {
   String msg = "";
   List l;
   MyFrame(String s) {
      setLayout(null);
      setVisible(true);
      setSize(300, 300);
      setTitle("hello");
      l = new List(5, true);
      l.add("Sunday");
      l.add("Monday");
      l.add("Tuesday");
      l.add("Wednesday");
      l.setBounds(100, 100, 100, 100);
      add(l);
   }
   public static void main(String[] args) {
   MyFrame f = new MyFrame("");
   }
}
```

Output:



1.3.1.7 Scroll Bars

- Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Scrollbars allows users to scroll the components. Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically.
- The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar.

Constructors of Scroll Bar Class:

Sr. No.	Constructors	Description
1.	`Scrollbar()`	Creates a new scrollbar.
2.	`Scrollbar(int style)`	Creates a scroll bar with specified orientation, that is, Scrollbar.HORIZONTAL or Scrollbar.VERTICAL.
3.	`Scrollbar(int style, int initialvalue, int thumbsize,int min, int max)`	Specifies the initial value and the maximal value along with the thumb size.

Methods of Scroll Bar Class:

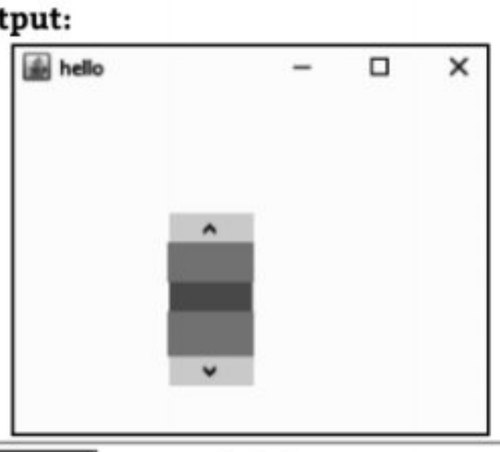
Sr. No.	Methods	Description
1.	`int getMaximum()`	Gets the largest value the scrollbar will report.
2.	`int getMinimum()`	Gets the smallest value the scrollbar will report.
3.	`void setValue(int value)`	Set the handle at the value.
4.	`void setValues(int value, int visible, int minimum, int maximum)`	Reset the scrollbar to reflect the bounds passed in.
5.	`int getValue()`	Gets the current value of the bar.

Program 1.11: Program to demonstrate scroll bar.

```
import java.awt.*;
class MyFrame extends Frame
{
    Scrollbar sb;
    MyFrame(String s)
    {
setLayout(null);
setVisible(true);
setSize(300, 300);
setTitle("hello");
```

```
            sb = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 100);
            sb.setBounds(100, 100, 50, 100);
            add(sb);
        }
        public static void main(String[] args) {
            MyFrame f = new MyFrame("");
        }
    }
```

Output:



1.3.1.8 Text Field

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- TextField is a subclass of TextComponent.

Constructors of TextField Class:

Sr. No.	Constructors	Description
1.	`TextField(String strInitialText, int columns)`	Construct a TextField instance with the given initial text string with the number of columns.
2.	`TextField(String strInitialText)`	Construct a TextField instance with the given initial text string.
3.	`TextField(int columns)`	Construct a TextField instance with the number of columns.

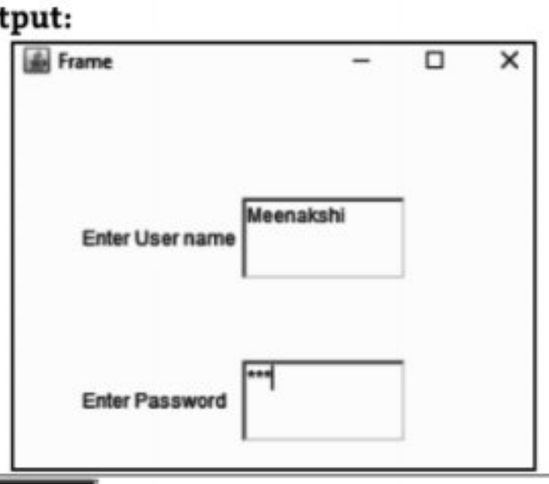
Methods of TextField Class:

Sr. No.	Methods	Description
1.	`String getText()`	Get the current text on this TextField instance.
2.	`void setText(String strText)`	Set the display text on this TextField instance. Take note that getText()/SetText() operates on String.
3.	`void setEditable(boolean editable)`	Set this TextField to editable (read/write) or non-editable (read-only).
4.	`addActionListener (ActionListener l)`	Adds the specified action listener to receive action events from this text field.
5.	`void removeActionListener (ActionListener l)`	Removes the specified action listener so that it no longer receives action events from this text field.

Program 1.12: Program to demonstrate textfield.

```
import java.awt.*;
class MyFrame extends Frame {
TextField t1, t2;
Label l1,l2;
MyFrame(String s) {
    super(s);
    setLayout(null);
    setSize(500, 500);
    setVisible(true);
    l1=new Label("Enter User name");
    l1.setBounds(50, 100, 100, 50);
    t1 = new TextField(12);
    t1.setBounds(150, 100, 100, 50);
    add(l1);
    add(t1);
    l2=new Label("Enter Password");
    l2.setBounds(50, 200, 100, 50);
    t2 = new TextField(8);
    t2.setBounds(150, 200, 100, 50);
    t2.setEchoChar('*');
    add(l2);
    add(t2);
}
public static void main(String[] args) {
    MyFrame f = new MyFrame("Frame");
}
}
```

Output:



1.3.1.9 Text Area

- Sometimes, a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multi-line editor called TextArea.
- The user can type here as much as he/she wants. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up and down and right and left.

Constructors of TextArea Class:

Sr. No.	Constructors	Description
1.	`TextArea()`	Constructs a new text area with the empty string as text.
2.	`TextArea(int rows, int columns)`	Constructs a new text area with the specified number of rows and columns and the empty string as text.
3.	`TextArea(String text)`	Constructs a new text area with the specified text.
4.	`TextArea(String text, int rows, int columns)`	Constructs a new text area with the specified text, and with the specified number of rows and columns.
5.	`TextArea(String text, int rows, int columns, int scrollbars)`	Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified below: (i) **SCROLLBARS_BOTH:** Create and display both vertical and horizontal scrollbars. (ii) **SCROLLBARS_HORIZONTAL_ONLY:** Create and display horizontal scrollbar only. (iii) **SCROLLBARS_NONE:** Do not create or display any scrollbars for the text area. (iv) **SCROLLBARS_VERTICAL_ONLY:** Create and display vertical scrollbar only.

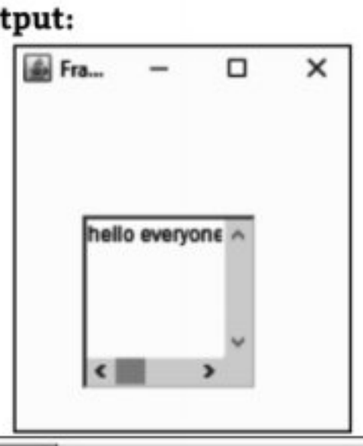
Methods of TextArea Class:

Sr. No.	Methods	Description
1.	`void setColumns(int columns)`	Sets the number of columns for this text area.
2.	`int getColumns()`	Returns the number of columns in this text area.
3.	`int getRows()`	Returns the number of rows in the text area.
4.	`void insert(String str, int pos)`	Inserts the specified text at the specified position in this text area.
5.	`void append(String str)`	Appends the given text to the text area's current text.
6.	`void replaceRange(String str, int start, int end)`	Replaces text between the indicated start and end positions with the specified replacement text.

Program 1.13: Program to demonstrate textarea.

```
import java.awt.*;
class MyFrame extends Frame {
   String msg = "hello everyone,This unit is Introduction to AWT and its components.";
   TextArea t1;
   MyFrame(String s) {
      super(s);
      setLayout(null);
      setSize(500, 500);
      setVisible(true);
      t1 = new TextArea(msg);
      t1.setBounds(50, 100, 100, 100);
      add(t1);
   }
   public static void main(String[] args) {
      MyFrame f = new MyFrame("Frame");
   }
}
```

Output:



1.4 USE OF LAYOUT MANAGERS

- Layout means the arrangement of components within the container. The task of layouting the controls are done automatically by the Layout Manager. All of the components that we have shown so far have been positioned by the default layout manager.
- The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used.
- Whenever, a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.
- The setLayout( ) method has the following general **form/syntax**:

```
void setLayout(LayoutManager layoutObj)
```

Here, layoutObj is a reference to the desired layout manager.

- Java has several predefined Layout Manager Classes. Layout manager classes are founded in the java.awt package same as the classes used for displaying graphical components.
- Several layout managers of which are described next sections. We can use the layout manager that best fits our application.
- The java.awt package provides five layout manager classes as given below:
  1. **FlowLayout:** Arranges components in variable-length rows.
  2. **BorderLayout:** Arranges components along the sides of the container and in the middle.
  3. **CardLayout:** Arrange components in "cards" Only one card is visible at a time.
  4. **GridBagLayout:** Aligns components horizontally and vertically; components can be of different sizes.
  5. **GridLayout:** Arranges components in fixed-length rows and columns.

1.4.1 FlowLayout

- FlowLayout is the default layout manager. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right.

Constructors of FlowLayout Class:

Sr. No.	Constructors	Description
1.	`FlowLayout()`	Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
2.	`FlowLayout(int align)`	Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap. We can use one of the followings alligment constants. FlowLayout.LEFT (or LEADING), FlowLayout.RIGHT (or TRAILING), or FlowLayout.CENTER
3.	`FlowLayout(int align, int hgap, int vgap)`	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

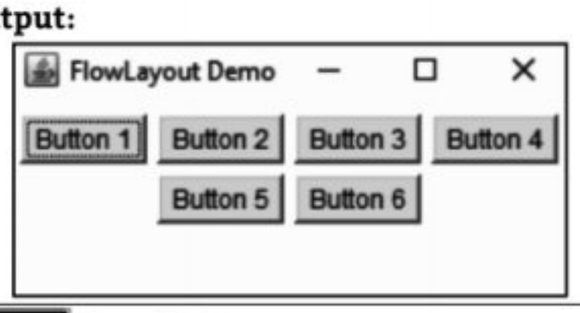
Methods of FlowLayout Class:

Sr. No.	Methods	Description
1.	`int getAlignment()`	Gets the alignment of this layout.
2.	`int getHgap()`	Gets the horizontal spacing.
3.	`int getVgap()`	Gets the vertical spacing.
4.	`int setAlignment(int)`	Specifies the alignment of this layout.
5.	`int setHgap(int)`	Specifies the horizontal spacing.
6.	`int setVgap(int)`	Specifies the vertical spacing.

Program 1.14: Program for FlowLayout.

```
import java.awt.*;
 public class FlowLayoutDemo extends Frame
 {
    private Button btn1, btn2, btn3, btn4, btn5, btn6;
    public FlowLayoutDemo ()
    {
        setLayout(new FlowLayout());
        btn1 = new Button("Button 1");
        add(btn1);
        btn2 = new Button("Button 2");
        add(btn2);
        btn3 = new Button("Button 3");
        add(btn3);
        btn4 = new Button("Button 4");
        add(btn4);
       btn5 = new Button("Button 5");
        add(btn5);
       btn6 = new Button("Button 6");
        add(btn6);
        setTitle("FlowLayout Demo");
        setSize(280, 150);
        setVisible(true);
     }
    public static void main(String[] args)
    {
        new FlowLayoutDemo();
    }
 }
```

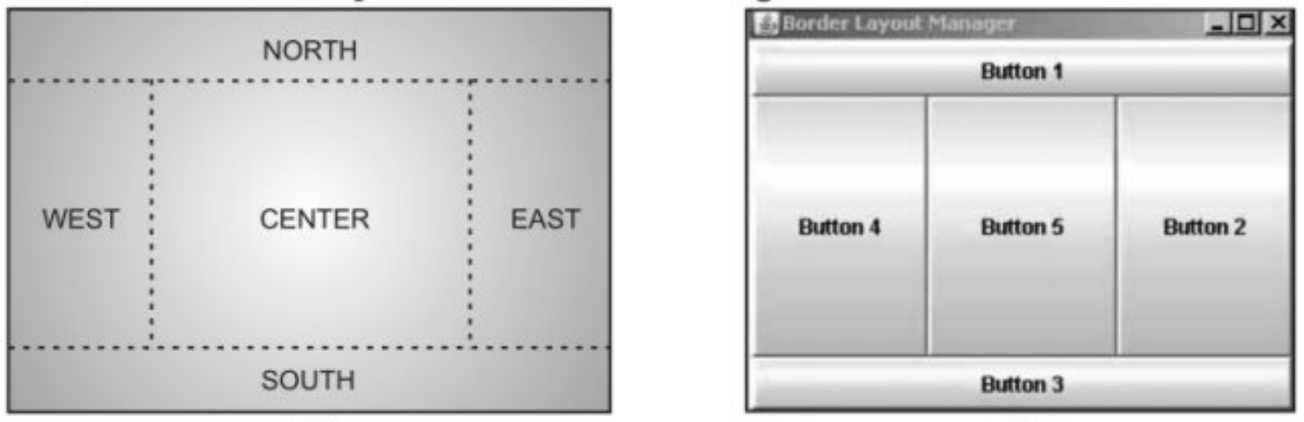
Output:



1.4.2 BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north (upper region), south (lower region), east (left region), and west (right region). The middle area is called the center (central region).

- Five zones, one for each component, as can be seen in Fig. 1.2.



(a) (b)

Fig. 1.2

Constructors of BorderLayout Class:

Sr. No.	Constructors	Description
1.	`BorderLayout()`	Creates a new layout manager without spacing between regions.
2.	`BorderLayout(int, int)`	Creates a new layout manager with the spacing horizontal and vertical specified. By default hgap=0, vgap=0.

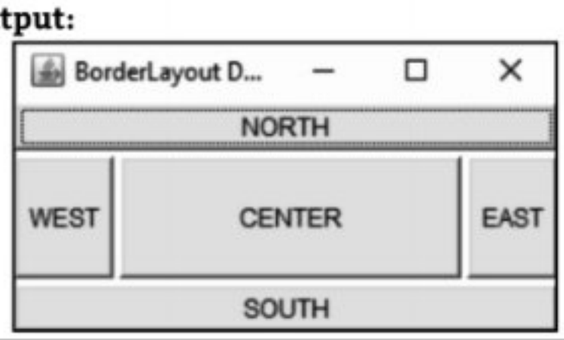
Methods of BorderLayout Class:

Sr. No.	Methods	Description
1.	`int getHgap()`	Gets the horizontal spacing.
2.	`int getVgap()`	Gets the vertical spacing.
3.	`void setHgap(int)`	Specifies the horizontal spacing.
4.	`void setVgap(int)`	Specifies the vertical spacing.

Program 1.15: Program to demonstrate BorderLayout.

```
import java.awt.*;
public class BorderLayoutDemo extends Frame
{
    private Button btnNorth, btnSouth, btnCenter, btnEast, btnWest;
    public BorderLayoutDemo()
    {
        setLayout(new BorderLayout(3, 3));
        btnNorth = new Button("NORTH");
        add(btnNorth, BorderLayout.NORTH);
        btnSouth = new Button("SOUTH");
        add(btnSouth, BorderLayout.SOUTH);
        btnCenter = new Button("CENTER");
        add(btnCenter, BorderLayout.CENTER);
        btnEast = new Button("EAST");
        add(btnEast, BorderLayout.EAST);
        btnWest = new Button("WEST");
        add(btnWest, BorderLayout.WEST);
        setTitle("BorderLayout Demo");
        setSize(280, 150);
        setVisible(true);
    }
    public static void main(String[] args) {
        new BorderLayoutDemo();
    }
}
```

Output:



1.4.3 GridLayout

- GridLayout lays out components in a two-dimensional grid.
- When we instantiate a GridLayout, we define the number of rows and columns as shown in Fig. 1.3.
- The Grid layout manager places items in rows (left to right) and columns (top to bottom).
- The number of rows and columns is defined when the Grid layout manager is created.

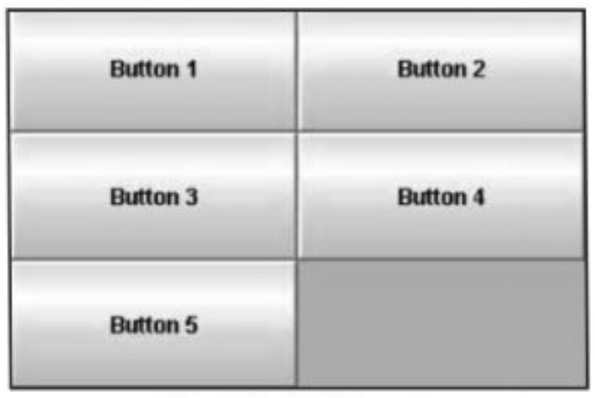


Fig. 1.3: GridLayout

Constructors of GridLayout Class:

Sr. No.	Constructors	Description
1.	`GridLayout()`	Creates a GridLayoutmanager with a defaultrow and a column.
2.	`GridLayout(int, int)`	Creates a GridLayoutmanager with the number of rows and columns specified.
3.	`GridLayout(int, int, int, int)`	Creates a GridLayout manager with the number of specified rows and columns and alignment, horizontal and vertical spacing data. By default: rows=1, cols=0, hgap=0, vgap=0.

Methods of GridLayout Class:

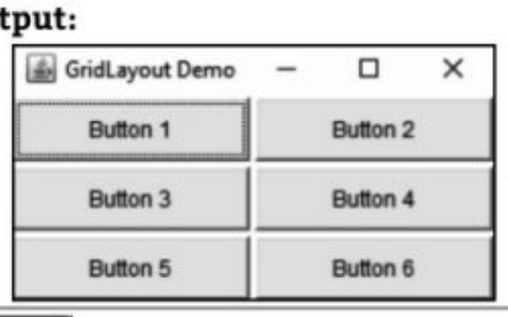
Sr. No.	Methods	Description
1.	`int getColumns()`	Gets the number of columns in the layout.
2.	`int getHgap()`	Gets the horizontal spacing.
3.	`int getRows()`	Gets the number of rows of the layout.
4.	`int getVgap()`	Gets the vertical spacing.
5.	`int SetColumns()`	Specifies the number of columns in the layout.
6.	`int setHgap(int)`	Specifies the horizontal spacing.
7.	`int SetRows()`	Specifies the number of rows of the layout.
8.	`int setVgap(int)`	Specifies the vertical spacing.

Program 1.16: Program to demonstrate GridLayout.

```
import java.awt.*;
public class GridLayoutDemo extends Frame {
   private Button btn1, btn2, btn3, btn4, btn5, btn6;
public GridLayoutDemo() {
   setLayout(new GridLayout(3, 2, 3, 3));
   btn1 = new Button("Button 1");
   add(btn1);
```

```
            btn2 = new Button("Button 2");
            add(btn2);
            btn3 = new Button("Button 3");
            add(btn3);
            btn4 = new Button("Button 4");
            add(btn4);
            btn5 = new Button("Button 5");
            add(btn5);
            btn6 = new Button("Button 6");
            add(btn6);
            setTitle("GridLayout Demo");
            setSize(280, 150);
            setVisible(true);
        }
        public static void main(String[] args) {
            new GridLayoutDemo();
        }
    }
```

Output:



1.4.4 CardLayout

- The CardLayout manager provides the means to manage multiple components, displaying one at a time. Components are displayed in the order in which they are added to the layout.
- The class CardLayout arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.

Constructors of CardLayout Class:

Sr. No.	Constructors	Description
1.	`CardLayout()`	Creates a new layout manager without spacing between regions.
2.	`CardLayout(int, int)`	Creates a new layout manager with the spacing horizontal and vertical specified.

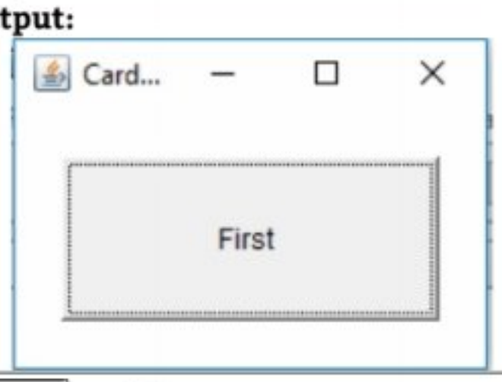
Methods of CardLayout Class:

Sr. No.	Methods	Description
1.	`void first(Container)`	Displays the first component added to the layout container specified.
2.	`int getHgap()`	Gets the horizontal spacing.
3.	`int getVgap()`	Gets the vertical spacing.
4.	`void last(Container)`	Returns the last component added to the layout container specified.
5.	`void next(Container)`	Returns the next component added to the layout container specified.
6.	`void previous(Container)`	Displays the previous component added to the layout container specified.
7.	`void setHgap(int)`	Specifies the horizontal spacing.
8.	`void setVgap(int)`	Specifies the vertical spacing.

Program 1.17: Program for CardLayout.

```
import java.awt.*;
class CardLayoutDemo extends Frame
{
   CardLayout card = new CardLayout(20,20);
   CardLayoutDemo()
   {
      setTitle("CardLayout in Java Example");
      setSize(220,150);
      setVisible(true);
      setLayout(card);
      Button Btnfirst = new Button("First ");
      Button BtnSecond = new Button ("Second");
      Button BtnThird = new Button("Third");
      add(Btnfirst,"Card1");
      add(BtnSecond,"Card2");
      add(BtnThird,"Card3");
   }
   public static void main(String args[])
   {
      CardLayoutDemo frame = new CardLayoutDemo();
   }
}
```

Output:



1.4.5 GridBagLayout

- The GridBagLayout layout manager is the most powerful and flexible of all the predefined layout managers but more complicated to use.
- Unlike GridLayout where the component are arranged in a rectangular grid and each component in the container is forced to be the same size, in GridBagLayout, components are also arranged in rectangular grid but can have different sizes and can occupy multiple rows or columns, (See Fig. 1.4).
- The class GridBagLayout arranges components in a horizontal and vertical manner.
- A GridBagLayout layout manager requires a lot of information to know where to put a component in a container. A helper class called GridBagConstraints provides all this information.
- It specifies constraints on how to position a component, how to distribute the component and how to resize and align them.

Fig. 1.4:GridBagLayout

- Each component in a GridBagLayout has its own set of constraints, so you have to associate an object of type GridBagConstraints with each component before adding component to the container.
- Instance variables to manipulate the GridBagLayout object Constraints are:

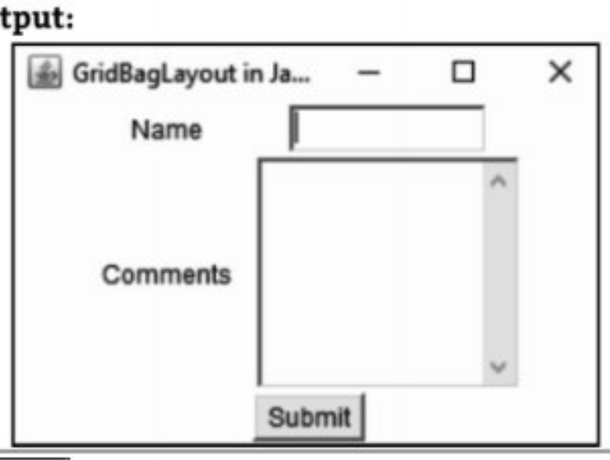
Sr. No.	Variable	Role
1.	gridx and gridy	These contain the coordinates of the origin of the grid. They allow at specific position of a component positioning. By default, they have GrigBagConstraint.RELATIVE value which indicates that a component can be stored to the right of previous.
2.	gridwidth, gridheight	Define how many cells will occupy component (height and width). By default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints.REMAINDER value specifies that the next component inserted will be the last of the line or the current column. the value GridBagConstraints.RELATIVE up the component after the last component of a row or column.
3.	fill	Defines how to resize the component which is smaller than the grid cell. GridBagConstraints.NONE retains the original size i.e., Default. GridBagConstraints.HORIZONTAL expanded horizontally. GridBagConstraints.VERTICAL expanded vertically. GridBagConstraints.BOTH expanded to the dimensions of the cell.
4.	ipadx, ipady	Used to define the horizontal and vertical expansion of components. not works if expansion is required by fill. The default value is (0, 0).
5.	anchor	When a component is smaller than the cell in which it is inserted, it can be positioned using this variable to define theside from which the control should be aligned within the cell.Possible variables NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST and EAST.
6.	weightx, weighty	Used to define the distribution of space in case of change of dimension.

Program 1.18: Program for GridBayLayout.

```
import java.awt.*;
class GridBagLayoutDemo extends Frame
{
    GridBagLayoutDemo()
    {
        Label lblName = new Label("Name");
        TextField txtName =new TextField(10);
        Label lblcomments = new Label("Comments");
        TextArea TAreaComments=new TextArea(6,15);
        Button btnSubmit = new Button("Submit");
        setLayout(new GridBagLayout());
        GridBagConstraints gc =new GridBagConstraints();
        add(lblName,gc,0,0,1,1,0,0);
        add(txtName,gc,1,0,1,1,0,20);
        add(lblcomments,gc,0,1,1,1,0,0);
        add(TAreaComments,gc,1,1,1,1,0,60);
        add(btnSubmit,gc,0,2,2,1,0,20);
    }
```

```
void add(Component comp,GridBagConstraints gc,int x,int y,int w,int h,int wx,int wy)
{
    gc.gridx = x;
    gc.gridy = y;
    gc.gridwidth = w;
    gc.gridheight= h;
    gc.weightx = wx;
    gc.weighty = wy;
    add(comp,gc);
    }
}
class GridBagLayoutJavaExample
{
    public static void main(String args[])
    {
        GridBagLayoutDemo frame = new GridBagLayoutDemo();
        frame.setTitle("GridBagLayout in Java Example");
        frame.setSize(300,200);
        frame.setVisible(true);
    }
}
```

Output:



1.4.6 Menu Bars and Menus

- A top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices. Each choice is associated with a drop down menu.
- Fig. 1.5 shows menu hierarchy.

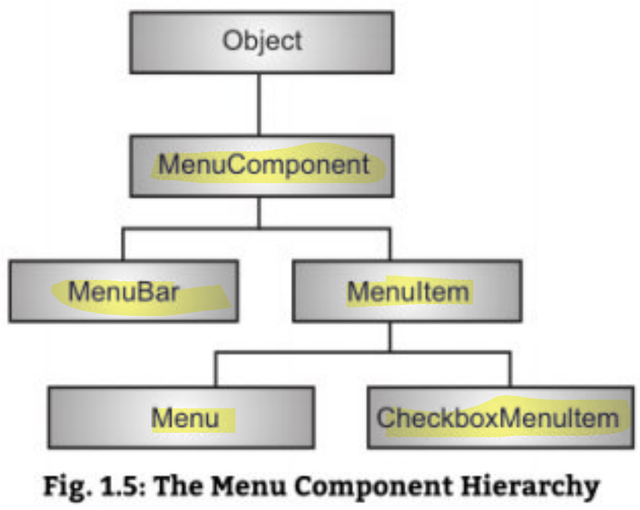


Fig. 1.5: The Menu Component Hierarchy

- Menu controls are listed in following table:

Sr. No.	Control and Description
1.	**MenuComponent:** It is the top level class for all menu related controls.
2.	**MenuBar:** The MenuBar object is associated with the top-level window.
3.	**MenuItem:** The items in the menu must belong to the MenuItem or any of its subclass.
4.	**Menu:** The Menu object is a pull-down menu component which is displayed from the menu bar.
5.	**CheckboxMenuItem:** CheckboxMenuItem is subclass of MenuItem.
6.	**PopupMenu:** PopupMenu can be dynamically popped up at a specified position within a component.

- Creation of menus involves many steps to be followed in an order. Following are the steps:
  Step 1 : Create menu bar.
  Step 2 : Add menu bar to the frame.
  Step 3 : Create menus.
  Step 4 : Add menus to menu bar.
  Step 5 : Create menu items.
  Step 6 : Add menu items to menus.
  Step 7 : Event handling.

1.4.6.1 Menu Bar

- In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.
- Since, Menu is a subclass of MenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items.
- These are menu options of type CheckboxMenuItem and will have a check mark next to them when they are selected.
- For creating a menu bar, we first create an instance of MenuBar. This class only defines the default constructor MenuBar() which Creates a new menu bar. Next, create instances of Menu that will define the selections displayed on the bar.

Constructors of MenuBar Class and Menu Class:

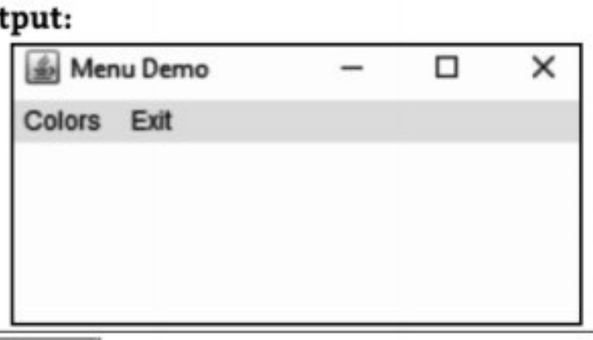
Sr. No.	Constructors	Description
1.	`MenuBar()`	Create menubar.
2.	`Menu()`	To create a default menu.
3.	`Menu(String name)`	To create a menu with name.
4.	`Menu(String name, Boolean tearoff)`	Removable is true means menu can be torn off.

Program 1.19: Program to demonstrate menu bar.

```
import java.awt.*;
public class MenuDemo extends Frame
{
 public MenuDemo()
 {
    MenuBar mBar = new MenuBar();
    setMenuBar(mBar); // add menu bar to frame
Menu files = new Menu("Colors");
Menu exit = new Menu("Exit");
    mBar.add(files);            // add menus to menu bar
    mBar.add(exit);
```

```
        setTitle("Menu Demo");
        setSize(400, 400);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuDemo();
    }
}
```

Output:



1.4.6.2 Menu

- The Menu class represents pull-down menu component which is deployed from a menu bar.
- Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.

Constructors of MenuItem Class:

Sr. No.	Constructors	Description
1.	`MenuItem()`	To create a default menu item.
2.	`MenuItem(String itemname)`	Create Menu item by using name of menu item.
3.	`MenuItem(String itemname, MenuShortcut shortcut)`	Create Menu item by using name of menu item and shortcut.

Methods of MenuItem Class:

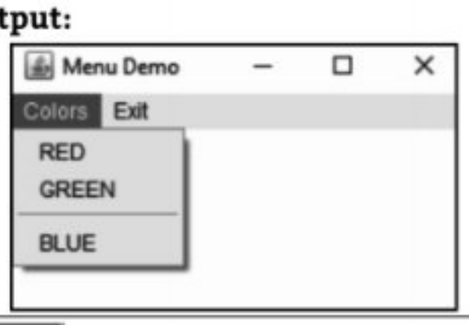
Sr. No.	Methods	Description
1.	`MenuItem add(MenuItem mi)`	Adds the specified menu item to this menu.
2.	`void add(String label)`	Adds an item with the specified label to this menu.
3.	`void addSeparator()`	Adds a separator line, or a hypen, to the menu at the current position.
4.	`MenuItem getItem(int index)`	Gets the item located at the specified index of this menu.
5.	`int getItemCount()`	Get the number of items in this menu.
6.	`void insert(MenuItem menuitem, int index)`	Inserts a menu item into this menu at the specified position.
7.	`void insert(String label, int index)`	Inserts a menu item with the specified label into this menu at the specified position.
8.	`void insertSeparator (int index)`	Inserts a separator at the specified position.
9.	`Boolean isTearOff()`	Indicates whether this menu is a tear-off menu.
10.	`void remove(int index)`	Removes the menu item at the specified index from this menu.
11.	`void remove (MenuComponent item)`	Removes the specified menu item from this menu.
12.	`void removeAll()`	Removes all items from this menu.

- You can create checkable menu item by using a class CheckboxMenuItem, which is a subclass of menu item. It has the following constructors:
  - `CheckboxMenuItem()`
  - `CheckboxMenuItem(String itemname)`
  - `CheckboxMenuItem(String itemname, Boolean on).`

Program 1.20: Program for menus.

```
import java.awt.*;
public class MenuDemo extends Frame
{
    public MenuDemo()
    {
        MenuBar mBar = new MenuBar();
        setMenuBar(mBar);                          // add menu bar to frame
        Menu files = new Menu("Colors");
        Menu exit = new Menu("Exit");
        mBar.add(files);                           // add menus to menu bar
        mBar.add(exit);
        MenuItem mi1 = new MenuItem("RED");
        files.add(mi1);                            //add menuitem in menu
        files.add(new MenuItem("GREEN"));
        files.addSeparator();
        files.add(new MenuItem("BLUE"));
        exit.add(new MenuItem("Close"));
        setTitle("Menu Demo");
        setSize(400, 400);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuDemo();
     }
}
```

Output:



1.4.7 Dialog Boxes

Child Window Of Top Level Window

- A dialog box is a GUI object in which you can place messages that you want to display on the screen.
- Often, we will want to use a dialog box to hold a set of related controls. Dialog boxes are primarily used to obtain user input.
- Dialog boxes are similar to frame windows, except that dialog boxes are always child windows of a top-level window. Also, dialog boxes don't have menu bars.
- Dialog boxes may be modal or modeless.
  1. When a modal dialog box is active, we cannot access other parts of our program until we have closed the dialog box.
  2. When a modeless dialog box is active, input focus can be directed to another window in our program. Thus, other parts of our program remain active and accessible.

Constructors of Dialog Box Class:

Sr. No.	Constructors	Description
1.	`Dialog(Frame parent)`	This constructor creates an instance of Dialog with no title and with parent as the Frame owning it. It is not modal and is initially resizable.
2.	`Dialog(Frame parent, boolean modal)`	This constructor creates an instance of Dialog with no title and with parent as the Frame owning it. If modal is true, the Dialog grabs all the user input of the program until it is closed. If modal is false, there is no special behavior associated with the Dialog.
3.	`Dialog(Frame parent, String title)`	Creates an instance of Dialog with parent as the Frame owning it and a window title of title. It is not modal and is initially resizable.
4.	`Dialog(Frame parent, String title, boolean modal)`	Creates an instance of Dialog with parent as the Frame owning it and a window title of title. If mode is true, the Dialog grabs all the user input of the program until it is closed. If modal is false, there is no special behavior associated with the Dialog.

Program 1.21: Program to demonstrate dialog box.

```
import java.awt.*;
public class DialogDemo extends Dialog
 {
    DialogDemo(Frame parent, String title)
    {
       super(parent, title, false);
       setLayout(new FlowLayout());
       setSize(300, 200);
       setBackground(Color.yellow);
       Button b = new Button("cancel");
       add(b);
    }
    public static void main(String args[])
    {
       Frame f1 = new Frame();
       DialogDemo dd = new DialogDemo(f1, "hello");
       dd.setVisible(true);
    }
 }
```

Output:

1.4.8 File Dialog

- Java provides a built-in dialog box that lets the user specify a file.
- To create a file dialog box, instantiate an object of type FileDialog. This causes a file dialog box to be displayed. Usually, this is the standard file dialog box provided by the operating system.

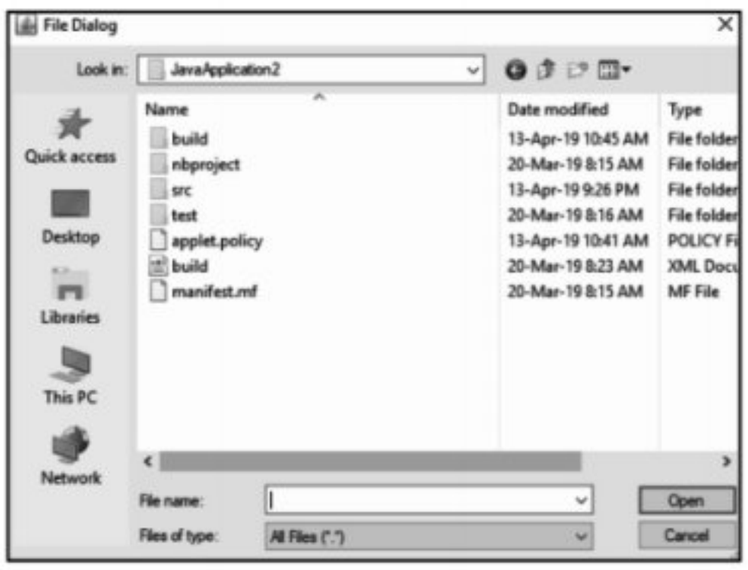
Constructors of FileDialog Class:

Sr. No.	Constructors	Description
1.	`FileDialog(Frame parent)`	Creates a file dialog for loading a file.
2.	`FileDialog(Frame parent, String boxName)`	Creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar.
3.	`FileDialog(Frame parent, String boxName, int how)`	Creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar. If how is FileDialog.LOAD, then the box is selecting a file for reading. If how is FileDialog.SAVE, the box is selecting a file for writing

Program 1.22: Program to demonstrate standard file dialog box.

```
import java.awt.*;
class SampleFrame extends Frame
{
   SampleFrame(String title)
   {
      super(title);
   }
}
class FileDialogDemo
{
   public static void main(String args[])
   {
      Frame f = new SampleFrame("File Dialog Demo");
      f.setVisible(true);
      f.setSize(100, 100);
      FileDialog fd = new FileDialog(f, "File Dialog");
      fd.setVisible(true);
   }
}
```

Output:



Practice Questions

1. What is AWT?
2. Explain AWT class hierarchy diagrammatically.
3. What are different AWT controls?
4. Construct a list to display the book list as shown below:
   (i) Java in 21 days
   (ii) C++ Made Easy
   (iii) C Programming
   (iv) System Analysis and Design.
5. What is Frame? Explain with example.
6. What is Panel? How to create it? Describe with example.
7. What is layout?
8. List out different layouts and explain any one in detail.
9. Explain check box control with suitable programming example.
10. Write a program to design a form using components textbox, text field, checkbox, buttons, list and handle various events related to each component.
11. Write a program to design a calculator using Java components and handle various events related to each component and apply proper layout to it.
12. Write a program to demonstrate use of GridLayout.
13. Write a program to demonstrate use of FlowLayout.
14. Write a program to demonstrate use of CardLayout.
15. Write a program to demonstrate use of BorderLayout.
16. Write a program to create a menu bar with various menu items and sub menu items. Also create a checkable menu item. On clicking a menu Item display a suitable Dialog box.
17. Write a program to increase the font size of a font displayed when the value of thumb in scrollbar increases at the same time it decreases the size of the font when the value of font decreases.
18. What is label? How to create it? Explain with example.
19. With the help of diagram explain scroll bar control.
20. What is menu and menu item?
21. What is dialog box?
22. How to create menus in AWT? Explain with example.
23. Describe file dialog with example.

■■■

2... Swing

Chapter Outcomes...

- Differentiate between AWT and Swing on the given aspect.
- Develop Graphical User Interface (GUI) programs using Swing components for the given problem.
- Use the given type of button in Java based GUI.
- Develop Graphical User Interface (GUI) programs using advanced Swing components for the given problem.

Learning Objectives...

- To learn Basic Concepts of Swing
- To understand Components of Swing (JApplet, Icons, Combo Boxes JButton, Check Boxes etc.)
- To learn Advanced Components of Swing (Tabbed Panes, Scroll Panes, Trees, Tables etc.)
- To study MVC Architecture

2.0 INTRODUCTION

- Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a Graphical User Interface (GUI) for Java programs.
- Swing is the User Interface (UI) toolkit of Java Foundation Classes JFC). When Sun developed the original version of Java, it introduced the Abstract Windowing Toolkit (AWT), which drew user interfaces based on an abstract layer that set on top of the native windowing toolkit. This caused many problems, as the abstraction tended to blur when faced with the peculiarities of many windowing platforms.
- To resolve these issues Swing were introduced as a more sophisticated toolkit with much better cross-platform support. Swing are based on AWT, so the core AWT is still part of Java.
- Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract WindowingToolkit (AWT).
- Swing has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and so on.
- Swing API is a set of extensible GUI components to ease the developer's life to create Java based Front End/GUI Applications.
- It is built on top of AWT API and acts as a replacement of AWT API, since, it has almost every control corresponding to AWT controls.

2.1 INTRODUCTION TO SWING

- Swing is a Java foundation classes library and it is an extension to Abstract Windowing Toolkit (AWT). Swing is actually part of a Java Foundation Classes (JFC).
- Swing is an extension to the AWT components which provides feature of pluggable look and feel for the components. It provides classes to design lightweight components.

- Swing is a set of classes, this provides more powerful and flexible components than are possible with the AWT. In addition to the familiar components, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables.
- Even familiar components such as buttons have more capabilities in Swing.
- For example, a button may have both an image and a text string associated with it.
- Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are plat-form-independent. The term lightweight is used to describe such elements.
- Fig. 2.1 shows the class hierarchy of the Swing GUI classes.
- Component represents an object with graphical representation. A Container is a component that can contain other SWING components.

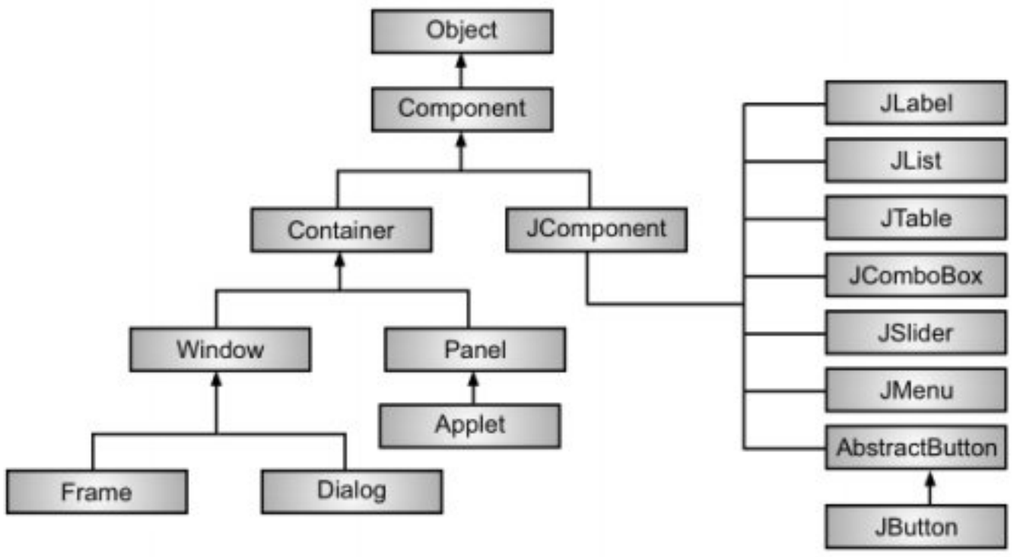


Fig. 2.1: AWT and Swing Classes

- A JComponent is a base class for all Swing UI components. In order to use a Swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.

2.1.1 Swing Features

- Various features of Swing are listed below:
  1. **Light Weight**: Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
  2. **Rich Controls**: Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, color picker, table controls and so on.
  3. **Borders:** We can draw borders in many different styles around components using the setborder( ) method.
  4. **Easy Mouseless Operation:** It is easy to connect keystrokes to components.
  5. **Tooltips:** We can use the setToolTipText method of JComponent to give components a tooltip, one of those small windows that appear when the mouse hovers over a component and gives explanatory text.
  6. **Easy Scrolling:** We can connect scrolling to various components-something that was impossible in AWT.
  7. **Pluggable Look and Feel:** We can set the appearance of applets and applications to one of three standard looks, i.e., Windows, Motif (Unix) or Metal (Standard Swing look).

2.1.2 MVC Architecture

- Swing components follow the Model-View-Controller (MVC) paradigm and thus can provide a much more flexible UI.
- Swing uses MVC architecture as the fundamental design behind each of its components.
- The MVC architectural pattern is used in software engineering to allow for the separation of three common features of GUI applications:
  - The data access (typically via a database).
  - The business logic (how the data will be used). User interaction (how the data and actions will be visually presented).

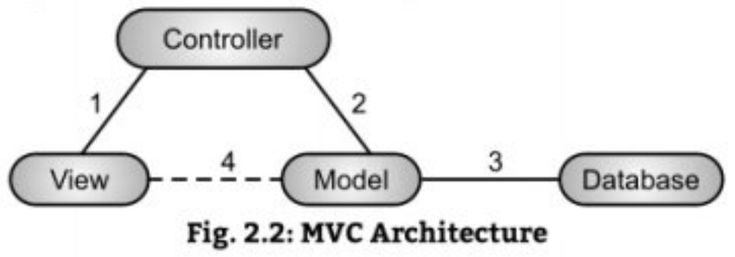


Fig. 2.2: MVC Architecture

- Towards this end one defines three components of such an architecture, the model, view and controller with the following interactions:
- The components are:
  1. **Model**: This provides the means by which data is retrieved and manipulated.
  2. **View**: This represents the visual interface components of the Graphical User Interface (GUI) application which interact with the user. The interaction is of an event-driven nature where actions are initiated via keyboard and mouse.
  3. **Controller**: This joins the Model with the View and is the heart of the control logic by associating user-generated events with data actions.
- The interactions correspond to the Fig. 2.2 as follows:
  1. All actions begin in the view through events generated by the user. The controller provides listeners for the events. The controller also has the ability to manipulate the state of the view objects in a way which offers a response to the events generated by the user.
  2. The controller interacts with the model by either requesting information from the data source based on user-generated events, or by modifying the data based on these events.
  3. The model provides the programming interface which the controller must use to access the database, i.e., the controller does not interact directly with the database. In particular the notion of "connection" is never seen in the controller. Furthermore, the model also provides means to avoid, direct SQL queries to the database.
  4. The view interacts with the model only to know about type information and other data abstractions held in the model. The relatively weak link indicates that the View/Model interaction should be "minimal" because the manipulation of data is to be done within the event handlers of the controller.

2.1.3 Difference between AWT and Swing

- There are many differences between Java AWT and Swing that are given below:

Sr. No.	Java AWT	Swing
1.	AWT stands for Abstract Windowing Toolkit.	Swing is also called as JFC's (Java Foundation classes).
2.	AWT components require java.awt package.	Swing components require javax.swing package.
3.	AWT components are platform-dependent.	Java Swing components are platform-independent.
4.	AWT components are heavyweight.	Swing components are lightweight.
5.	AWT does not support pluggable look and feel.	Swing supports pluggable look and feel.
6.	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scroll panes, color chooser, tabbed pane etc.
7.	AWT does not follow MVC (Model-View Controller).	Swing follows MVC. Where, Model (M) represents data, View (V) represents presentation and Controller (C) acts as an interface between model and view.

2.1.4 Working with Swing

- While designing any GUI, we need to have a main window on which we can place or position the different visual components.
- In Swing the main window, also called a top level container is the root of a hierarchy, which contains all the other Swing components that appear inside the window. All Swing applications have at least one top level container.

- Every top level container has one intermediate container called content pane. This content pane contains all the visible components in the GUI window.
- The content plane is the base pane upon which all other components or container objects are placed. One exception to this rule is, if there is a menu bar in the top level container it will be placed outside the content pane.
- Fig. 2.3 illustrates the relationship between the components discussed above.

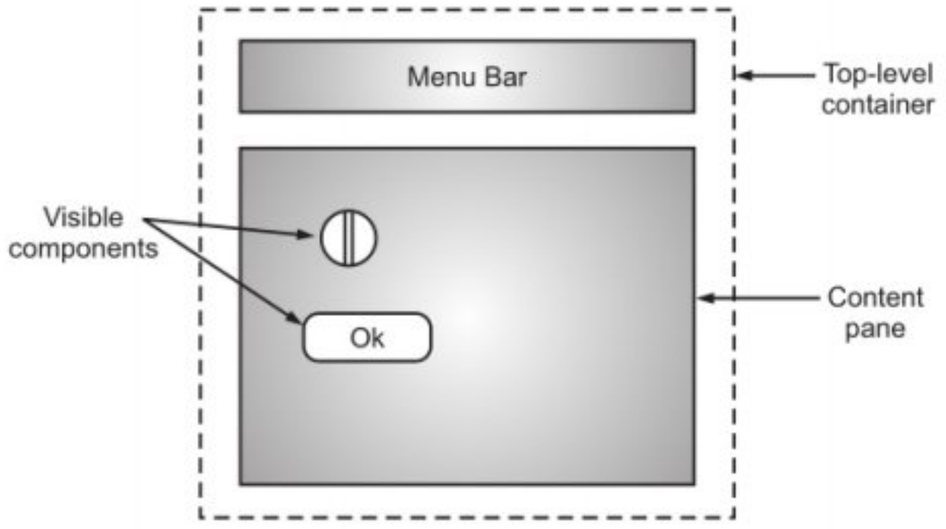


Fig. 2.3: Swing Application Window

- All Swing components names start with J. For instance the Swing button class is named as JButton commonly used top-level containers are:
- Just like AWT application, a Swing application requires a top-level container. There are three top-level containers in Swing:
  1. **JFrame:** Used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane).
  2. **JDialog:** Used for secondary pop-up window (with a title, a close button, and a content-pane).
  3. **JApplet:** Used for the applet's display-area (content-pane) inside a browser's window.
- Similarly to AWT, there are secondary containers (such as JPanel) which can be used to group and layout relevant components. Panels are an example of intermediate containers.

2.1.5 Advantages and Disadvantages of Swing

Advantages:

1. Swing provides paint debugging support for when you build your own component.
2. Swing components are lightweight than AWT.
3. Swing components follow the Model-View-Controller (MVC) paradigm and thus can provide a much more flexible UI.
4. Swing provides both additional components like JTable, JTree etc and added functionality to replacement of AWT components.
5. Swing provides built-in double buffering.

Disadvantages:

1. It can be slower than AWT (all components are drawn), when you are not careful about programming.
2. Swing components might not behave exactly like native components which look like native components.
3. It requires Java 2 or a separate JAR file.

2.2 SWING COMPONENTS

- Swing components are not implemented by platform specific code. Instead they are written entirely in Java and therefore, are platform independent. All Swing components name start with letter J.
- Swing component is called light weight, as it does not depend on any non-java system classes. Swing components have their own view supported by java's look and feel classes.
- Swing components have pluggable look and feel so that with a single set of components you can achieve the same look and feel on any operating system platform.

2.2.1 JFrame

- The frame is a top-level container or window, on which other Swing components are placed.
- A JFrame component is used to create windows in a Swing program. It extends java.awt.Frame class. Every JFrame object has a root pane.

Constructors of JFrame Class:

Sr. No.	Constructors	Description
1.	`JFrame()`	This constructor creates a new frame without title.
2.	`JFrame(String title)`	This constructor constructs the new frame with title.

Methods of JFrame Class:

Sr. No.	Methods	Description
1.	`Container getContentPane()`	This method returns, a Content Pane for the JFrame.
2.	`void setLayout(LayoutManager)`	This method sets the LayoutManager for the JFrame.
3.	`void setJMenuBar(JMenuBar)`	This method sets the JMenu Bar to the JFrame.
4.	`Void setIconImage(Image image)`	Prints icon image on JFrame.

- There are two way to create a JFrame Window:
  1. By instantiating JFrame class.
  2. By extending JFrame class.

Program 2.1: Creating JFrame window by instantiating JFrame class.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo {
    JFrame jf;
    JFrameDemo() {
        jf = new JFrame("MyWindow");
        JButton btn = new JButton("Say Hello");
        jf.add(btn);      //jf.getContentPane.add(btn) for  pre-JDK 1.5 versions
        jf.setLayout(new FlowLayout());
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(400, 400);
        jf.setVisible(true);
    }
    public static void main(String[] args) {
        new JFrameDemo();
    }
}
```

Output:

Program 2.2: Creating JFrame window by extending JFrame class.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame
{
    JFrame jf;
JFrameDemo() {
   setTitle("MyWindow");
   JLabel lb = new JLabel("Welcome");
   add(lb);
```

```
        setLayout(new FlowLayout());
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 400);
          setVisible(true);
        }
        public static void main(String[] args) {
            new JFrameDemo();
        }
    }
```

Output:

Note: JFrame.add() and JFrame.getContentPane().add() both do the same thing - JFrame.add() is overridden to call JFrame.getContentPane().add(). In pre-JDK 1.5 versions you always had to specify JFrame.getContentPane().add() explicitly and JFrame.add()threw a RuntimeException.

2.2.2 JApplet

- JApplet is a class that represents the Swing applet. It is a subclass of Applet class and must be extended by all the applets that use Swing. It provides all the functionalities of the AWT applet as well as support for menubars and layering of components.
- Whenever we require to add a component to it, the component is added to the content pane. JApplet supports various panes such as root pane glass pane and content pane.

1. **Root Pane:**

- Swing containers like JApplet, JFrame, JWindow and JDialog delegate their duties to the root pane represented by the class JRootPane.
- The root pane is made up of a content pane, layered pane, glass pane and an optional menu bar, other GUI components must be added to one of these roots pane members, rather than to the root pane itself.

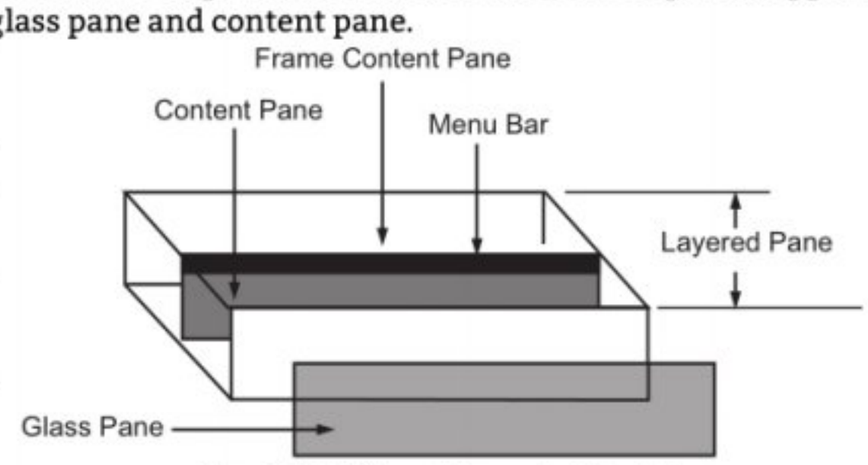


Fig. 2.4: Different Panes in JApplet

2. **Content Pane:**

- This pane is actually sits on one of the layers of a specially layered pane.
- The layered pane in Fig. 2.4 contains several layers meant for displaying different Swing components, depending on the GUI's requirements.
- Fig. 2.4 shows the position of the content pane in the specially layered pane. The menu bar and the content pane sit on the Frame-Content-Layer.

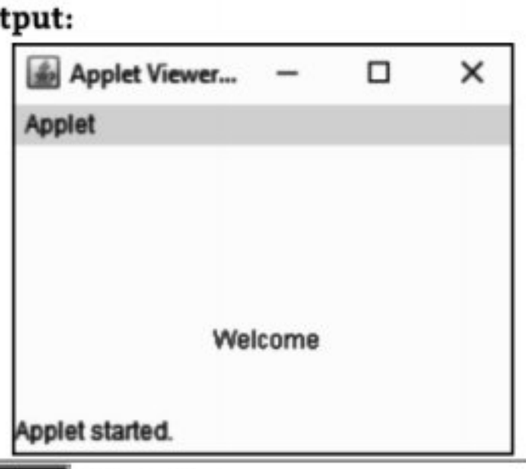
3. **Glass Pane:**

- It is a member of the multi-layered, root pane. It is used to draw over an area that already contains some components.
- The content pane makes Swing applets different from regular applets in the following ways:
  - Components are added to the content pane of Swing applets, not directly to the applets.
  - The layout manager is set for the content pane of a Swing Applet, not directly for the applet.
  - The default layout manager for the content pane of a Swing based applet is BorderLayout. This differs from the default Layout Manager for Applet which is FlowLayout.

Program 2.3: Program to demonstrate glass pane.

```
import javax.swing.*;
import java.awt.*;
/*
<applet code="JAppletDemo" width=250 height=150>
</applet>
 */
public class JAppletDemo extends JApplet {
    public void paint(Graphics g) {
        g.drawString("Welcome",100,100);
    }
}
```

Output:



2.2.3 JLabel

- One of the simplest Swing component is JLabel. A JLabel object is a component for placing text in a container.
- This class is used to create single line read only text which describes the other component. Labels can display text as well as Images.

Constructors of JLabel Class:

Sr. No.	Constructors	Description
1.	`JLabel()`	Create a empty label having no text and icon.
2.	`JLabel(String str)`	Create a label having some text.
3.	`JLabel(ImageIcon i)`	Creates a label having icon image.

Methods of JLabel Class:

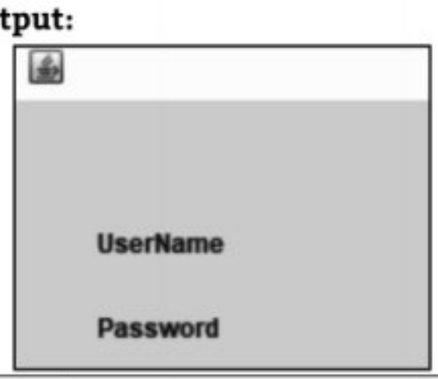
Sr. No.	Methods	Description
1.	`void setHorizontalAlignment(int a)`	This method sets the alignment of the text.
2.	`String getText()`	This method returns the text associated with the Label.
3.	`void setText(String s)`	This method is used to set the text to the Label.
4.	`void setIcon(Icon i)`	This method sets the Icon to the icon.

Program 2.4: Program to demonstrate JLabel.

```
import java.awt.Dimension;
import java.awt.Rectangle;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JFrameDemo extends JFrame
{
   private JLabel jLabel1 = new JLabel();
   private JLabel jLabel2 = new JLabel();
```

```
        public JFrameDemo()
        {
        setLayout( null );
        setSize( new Dimension(400, 300) );
        jLabel1.setText("UserName");
        jLabel1.setBounds(new Rectangle(40, 55, 80, 25));
        jLabel2.setText("Password");
        jLabel2.setBounds(new Rectangle(40, 95, 60, 25));
        add(jLabel2, null);
        add(jLabel1, null);
        }
        public static void main(String args[])
        {
        JFrameDemo frame=new JFrameDemo();
        frame.setVisible(true);
        }
    }
```

Output:



2.2.4 Icons

- Many Swing components, such as labels, buttons, and tabbed panes, can be decorated with an icon - a fixed-sized picture.
- An icon is an object that adheres to the Icon interface. Swing provides a ImageIcon class which implements the Icon interface which paints an icon from a GIF, JPEG, or PNG image.

Constructors of Icon Class:

Sr. No.	Constructors	Description
1.	`ImageIcon(String filename)`	This constructor creates object with image which is specified by filename.
2.	`ImageIcon(URL url)`	This constructor creates object with image in the resource identified by url.

Methods of Icon Class:

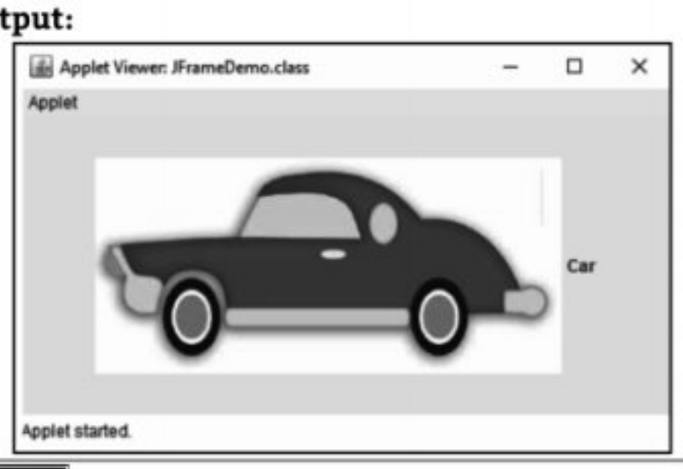
Sr. No.	Methods	Description
1.	`int getIconHeight( )`	Returns the height of the icon in pixels.
2.	`int getIconWidth( )`	Returns the width of the icon in pixels.

Program 2.5: Program of demonstrate icons.

```
import javax.swing.*;
/*
<applet code="JAppletDemo" width=250 height=150>
</applet>
 */
public class JAppletDemo extends JApplet {
```

```
        public void init() {
            ImageIcon icon = new
          ImageIcon("C:\\Users\\Meenakshi\\Documents\\NetBeansProjects\\JavaApplication2\\s
            rc\\car.jpg");
              JLabel jl = new JLabel("Car",icon,JLabel.CENTER);
              add(jl);
        }
    }
```

Output:



2.2.5 JTextField

- The JTextField component allows us to enter/edit a single line of text.
- Following are the important constructors of the subclasses of the JTextField class.
  1. `JTextField()`: Constructs a new TextField.
  2. `JTextField(int columns)`: Constructs a new empty TextField with the specified number of columns.
  3. `JTextField(String text)`: Constructs a new TextField initialized with the specified text.

Program 2.6: Program to demonstrate JTextField.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame {
    public JFrameDemo() {
        setLayout(new FlowLayout());
        JLabel j1 = new JLabel("TextField");
        add(j1);
        JTextField tf1 = new JTextField(40);
        add(tf1);
        setSize(300, 300);
        setVisible(true);
    }
    public static void main(String args[]) {
        new JFrameDemo();
    }
}
```

Output:

2.2.6 JTextArea

- The JTextArea is used to accept several lines of text from the user and it has capabilities not found in the AWT class.
- JTextArea class itself does not handle scrolling but implements the Scrollable interface which allows it to be placed inside a JScrollPane and thus implement scrolling.
- A JTextArea can be created using one of the following constructors:
  1. `JTextArea()`: Constructs a new TextArea.
  2. `JTextArea(int rows, int columns)`: Constructs a new empty TextArea with the specified number of rows and columns.
  3. `JTextArea(String text)`: Constructs a new TextArea with the specified text displayed.
  4. `JTextArea(String text, int rows, int columns)`: Constructs a new TextArea with the specified text and number of rows and columns.

Program 2.7: Program to demonstrate JTextArea.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame {
    public JFrameDemo() {
     setLayout(new FlowLayout());
     JLabel j11 = new JLabel ("TextArea");
     add(j11);
     JTextArea ta1 = new JTextArea(10, 20);
     add(ta1);
     setSize(600, 600);
      setVisible(true);
    }
    public static void main(String args[]) {
        new JFrameDemo();
    }
}
```

Output:

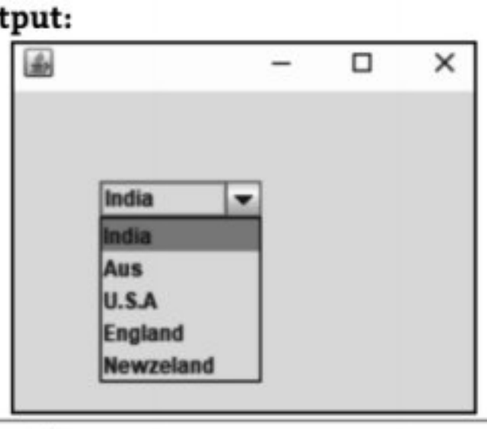
2.2.7 Combo Boxes

- Swing provides a combo box (a combination of a text field and a dropdown list) through the JComboBox class, which extends JComponent. A combobox normally displays one entry.
- However, it can also display a drop-down list that allows a user to select a different entry. We can also type our selection into the text field.
- The JComboBox component, similar to Choice component in AWT.
- JComboBox's constructors are shown below:
  1. `JComboBox()`: This constructor creates an empty JComboBox instance.
  2. `JComboBox(Object[] items)`: Creates a JComboBox that contains the elements of the specified array.

Program 2.8: Program to demonstrate JComboBox.

```
import javax.swing.*;
public class JFrameDemo extends JFrame
{
    public JFrameDemo()
   {
        String country[] = {"India", "Aus", "U.S.A", "England", "Newzeland"};
        JComboBox cb = new JComboBox(country);
        cb.setBounds(50, 50, 90, 20);
        add(cb);
        setLayout(null);
        setSize(400, 500);
        setVisible(true);
    }
    public static void main(String args[]) {
        new JFrameDemo();
    }
}
```

Output:



2.3 BUTTONS

- In this section we study buttons in Swing.

2.3.1 JButton

- This class is used to create a push buttons. This class is subclass of Abstract Button class.

Constructors of JButton Class:

Sr. No.	Constructors	Description
1.	`JButton()`	Create a empty button having no title and icon.
2.	`JButton(String str)`	Create a button having label.
3.	`JButton(Icon i)`	Creates a button having icon image.
4	`JButton(String str, Icon i)`	Creates a button having string and Icon image.

Methods of JButton Class:

Sr. No.	Methods	Description
1.	`void addActionListenerActionListener obj)`	This method is used to register the push button to throw an event.
2.	`String getText()`	This method returns the text associated with the button.
3.	`void setText(String s)`	This method is used to set new Label of the button

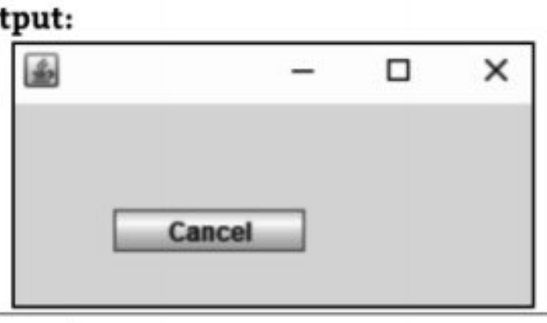
contd. ...

4.	`void setHorizontalTextPosition(int pos)`	Sets the horizontal text position relative to the graphics.
5.	`void setVerticalTextPosition(int pos)`	Sets the vertical text position relative to the graphics.
6.	`void setIcon( Icon i)`	This method sets the Icon to the button.
7.	`vois setMnemonic(char c)`	Sets the mnemonic character to the button.

Program 2.9: Program of demonstrate JButton.

```
import javax.swing.*;
public class JFrameDemo extends JFrame
 {
    public JFrameDemo()
  {
       setLayout(null);
       setSize(400, 500);
       setVisible(true);
       JButton jb = new JButton("Cancel");
       jb.setBounds(50, 50, 90, 20);
       add(jb);
    }
    public static void main(String args[]) {
       new JFrameDemo();
    }
}
```

Output:



2.3.2 JRadioButton

- This class is used to create radio button with a text or icon. Radio buttons are supported by the JRadioButton class, which is a concrete implementation of AbstractButton.
- Once, a radio button is created they must be kept in one group, which is created by using the ButtonGroup class Radio buttons must be configured into a group.
- In JRadioButton only one of the buttons in that group can be selected at any time.

Constructors of JRadioButton:

1. `JRadioButton()`: Create a empty radio button having no title and icon.
2. `JRadioButton(String l)`: Create a radio button having label.
3. `JRadioButton(Icon i)`: Creates a radio button having icon image.
4. `JRadioButton(String l, Icon i)`: Creates a radio button having string label and icon image.
5. `JRadioButton(String l, Icon i,boolean selected)`: Creates a radio button having string label and icon image and default selected if true is passed.
6. `JRadioButton(String l, boolean selected)`: Creates a radio button having string label and option for default selected policy.

Program 2.10: Program of demonstrate JRadioButton.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame
 {
    public JFrameDemo()
   {
        setLayout( new FlowLayout());
        setSize(400, 500);
        setVisible(true);
        JRadioButton jr1 = new JRadioButton("Java",true);
        add(jr1);
        JRadioButton jr2 = new JRadioButton("C++");
        add(jr2);
    }
    public static void main(String args[])
   {
        new JFrameDemo();
    }
}
```

Output:

2.3.3 JCheckBox

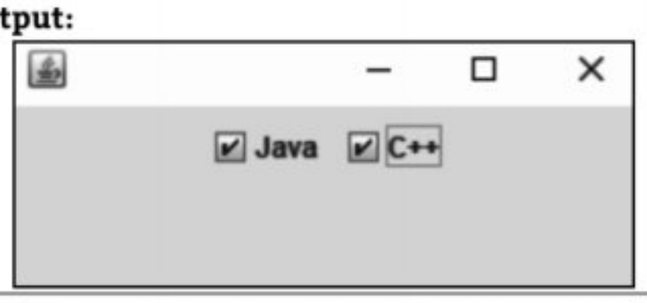
- A checkbox is a control that may be turned ON or OFF by the user to indicate some option.
- The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.
- The JCheckBox class is used to create CheckBox in Swing. CheckBox has the following constructors:
  1. `JCheckBox()`: Creates an initially unselected check box button with no text, no icon.
  2. `JCheckBox(Icon icon)`: Creates an initially unselected check box with an icon.
  3. `JCheckBox(Icon icon, boolean selected)`: Creates a check box with an icon and specifies whether or not it is initially selected.
  4. `JCheckBox(String text)`: Creates an initially unselected check box with text.
  5. `JCheckBox(String text, boolean selected)`: Creates a check box with text and specifies whether or not it is initially selected.
  6. `JCheckBox(String text, Icon icon)`: Creates an initially unselected check box with the specified text and icon.
  7. `JCheckBox(String text, Icon icon, boolean selected)`: Creates a check box with text and icon, and specifies whether or not it is initially selected.

Program 2.11: Program to demonstrate JCheckBox.

```
import javax.swing.*;
import java.awt.*;
public class JFrameDemo extends JFrame
{
    public JFrameDemo()
```

```
        {
            setLayout( new FlowLayout());
            setSize(400, 500);
            setVisible(true);
            JCheckBox jr1 = new JCheckBox("Java",true);
            add(jr1);
            JCheckBox jr2 = new  JCheckBox("C++");
            add(jr2);
        }
        public static void main(String args[])
        {
            new JFrameDemo();
        }
    }
```

Output:



2.4 ADVANCED SWING COMPONENTS

- Swing provides several advanced components such as tabbed panes, scroll panes, trees, tables, progress bars and so on. This sections deals with this advanced component in Swing.

2.4.1 Tabbed Panes (JTabbedPane)

- A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options. Tabbed panes are encapsulated by the JTabbedPane class, which extends JComponent.
- There are three **constructors of JTabbedPane**:
  1. `JTabbedPane()`: Creates an empty. TabbedPane with a default tab placement of JTabbedPane.TOP.
  2. `JTabbedPane(int tabPlacement)`: Creates an empty TabbedPane with the specified tab placement of any of the following:

     `JTabbedPane.TOP`

     `JTabbedPane.BOTTOM`

     `JTabbedPane.LEFT`

     `JTabbedPane.RIGHT`
  3. `JTabbedPane(int tabPlacement, int tabLayoutPolicy)`: Creates an empty TabbedPane with thespecified tab placement and tab layout policy.
- Tab placements are listed above. Tab layout policy may be either of the following:

  `JTabbedPane.WRAP_TAB_LAYOUT`

  `JTabbedPane.SCROLL_TAB_LAYOUT`
- Tabs are added via the following method:

  `void addTab(String str, Component comp)`
- Here, str is the title for the tab, and comp is the component that should be added to the tab. Typically, a JPanel or a subclass of it is added.

- The general procedure to use a tabbed pane in an applet is outlined here:
  Step 1 : Create a JTabbedPane object.
  Step 2 : Call addTab( ) to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)
  Step 3 : Repeat step 2 for each tab.
  Step 4 : Add the tabbed pane to applet/Frame.
- The following program illustrates how to create a tabbed pane. The first tab is titled Languages and contains four buttons. Each button displays the name of a language. The second tab is titled Colors and contains three checkboxes. Each check box displays the name of a color. The third tab is titled Flavors and contains one combo box. This enables the user to select one of three flavors.

Program 2.12: Program to demonstrate JTabbedPane.

```
import javax.swing.*;
/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
 */
public class JTabbedPaneDemo extends JApplet
{
    public void init()
    {
        JTabbedPane jtp = new JTabbedPane();
        jtp.addTab("Languages", new LangPanel());
        jtp.addTab("Colors", new ColorsPanel());
        jtp.addTab("Flavors", new FlavorsPanel());
        add(jtp);
    }
}
class LangPanel extends JPanel
{
    public LangPanel()
     {
        JButton b1 = new JButton("Marathi");
        add(b1);
        JButton b2 = new JButton("Hindi");
        add(b2);
        JButton b3 = new JButton("Bengali");
        add(b3);
        JButton b4 = new JButton("Tamil");
        add(b4);
    }
}
class ColorsPanel extends JPanel
{
    public ColorsPanel()
    {
        JCheckBox cb1 = new JCheckBox("Red");
        add(cb1);
```

```
        JCheckBox cb2 = new JCheckBox("Green");
        add(cb2);
        JCheckBox cb3 = new JCheckBox("Blue");
        add(cb3);
    }
}
class FlavorsPanel extends JPanel
{
    public FlavorsPanel()
    {
        JComboBox jcb = new JComboBox();
        jcb.addItem("Vanilla");
        jcb.addItem("Chocolate");
        jcb.addItem("Strawberry");
        add(jcb);
    }
}
```

Output:

2.4.2 JScrollPane

- A scroll pane is a component that presents a rectangular area in which a component may be viewed.
- Horizontal and/or vertical scroll bars may be provided if necessary. Scroll panes are implemented in Swing by the JScrollPane class.

Constructors of JScrollPane Class:

Sr. No.	Constructors	Description
1.	`JScrollPane(Component comp)`	This creates a scroll pane with the component specified with comp.
2.	`JScrollPane(int vsb, int hsb)`	This creates a scroll panel with the vsb and hsb are int constants that define when vertical and horizontal scroll bars for this scroll panel are shown.
3.	`JScrollPane(Component comp, int vsb, int hsb)`	This constructor scroll panel creates a combination of above both constructor.

Constants of JScrollPane Class:

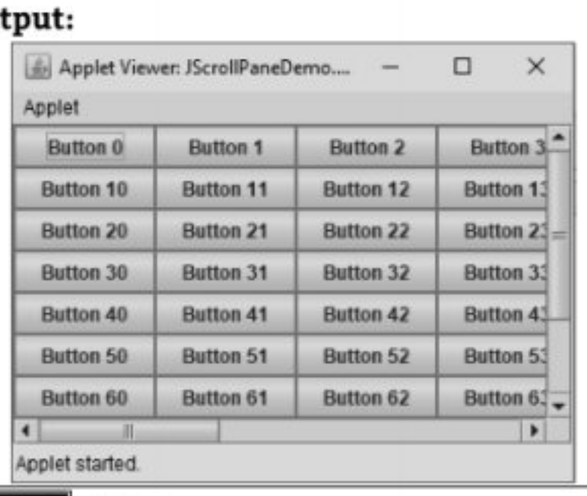
Sr. No.	Constants	Description
1.	`HORIZONTAL_SCROLLBAR_ALWAYS`	Always provide horizontal scroll bar.
2.	`HORIZONTAL_SCROLLBAR_AS_NEEDED`	Provide horizontal scroll bar, if needed.
3.	`VERTICAL_SCROLLBAR_ALWAYS`	Always provide vertical scroll bar.
4.	`VERTICAL_SCROLLBAR_AS_NEEDED`	Provide vertical scroll bar, if needed.

- Here, are the steps that you should follow to use a scroll pane in an applet:
  Step 1 : Create a JComponent object.
  Step 2 : Create a JScrollPane object, (the arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)
  Step 3 : Add the scroll pane to the applet.

Program 2.13: Program to demonstrate JScrollPane.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JScrollPaneDemo" width=300 height=250>
</applet>
 */
public class JScrollPaneDemo extends JApplet
{
    public void init()
    {
        setLayout(new BorderLayout());
        JPanel jp = new JPanel();
        jp.setLayout(new GridLayout(10, 10));
        int b = 0;
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                jp.add(new JButton("Button " + b));
                ++b;
            }
        }
        int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
        JScrollPane jsp = new JScrollPane(jp, v, h);
        add(jsp, BorderLayout.CENTER);
    }
}
```

Output:



2.4.3 JTree

- A tree is a component that presents a hierarchical view of data. A user has the ability to expand or collapse individual sub-trees in this display.
- Any computer user who had used Windows Explorer or File Manager will recall seeing a tree-like structure depicting files and folders, (See Fig. 2.5).

- This tree-like structure will display folders and subfolders like branches of a tree one below the other.

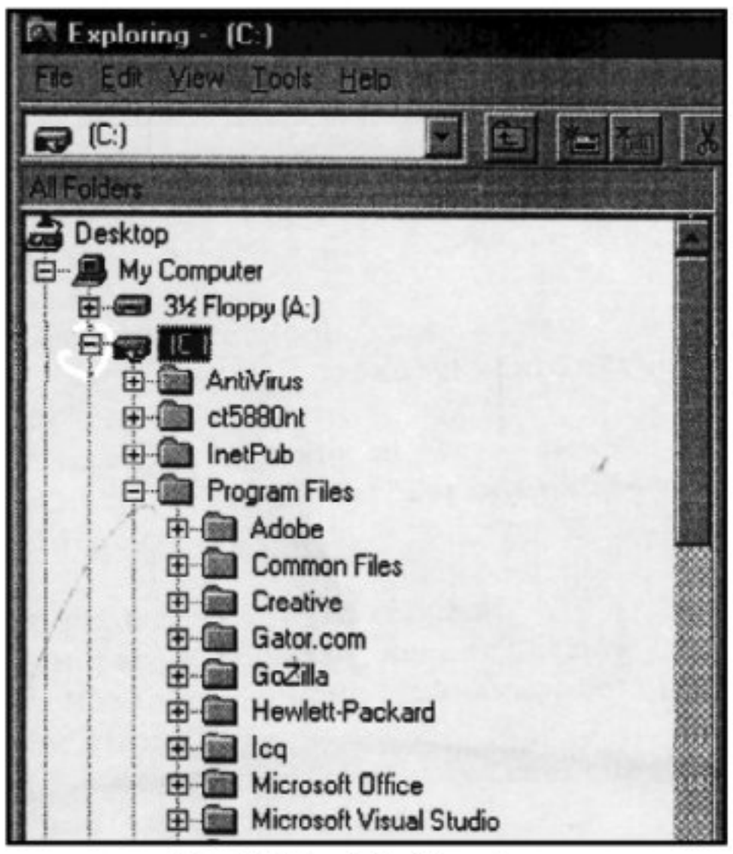


Fig. 2.5: Tree like Structure

- Similar, tree-like structures can be created in Java using JTree. Trees are implemented in Swing by the JTree class, which extends JComponent.

Constructors of JTree Class:

Sr. No.	Constructors	Description
1.	`JTree(Hashtable ht)`	To create a tree with elements of hash table as nodes.
2.	`JTree(Object ob[ ])`	To create a tree with objects as nodes.
3.	`JTree(TreeNode tn)`	To create a tree with tree node as root of tree.
4.	`JTree(Vector v)`	To create a tree with elements of vector as nodes.

Methods of JTree Class:

Sr. No.	Methods	Description
1.	`TreePath getPathForLocation(int x, int y)`	It is used to translate a mouse click on a point of tree to a tree path. Where, x and y are coordinates of mouse click.

DefaultMutableTreeNode Class:

- It is used to represent a node in a tree. Using DefaultMutableTree Node, you can create nodes for the root and for all of the data you want to represent in the tree.

Constructors of DefaultMutableTreeNode Class:

Sr. No.	Constructors	Description
1.	`DefaultMutableTreeNode()`	Creates a node with no associated user object.
2.	`DefaultMutableTreeNode(Object ob)`	Creates a node to which you can attach children.
3.	`DefaultMutableTreeNode(Object ob, boolean allowschildren)`	To specify that child nodes cannot be attached by supplying the third argument as false.

Methods of DefaultMutableTreeNode Class:

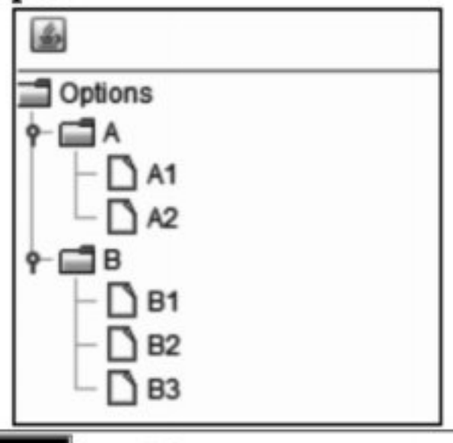
Sr. No.	Methods	Description
1.	`void add(MutableTreeNode child)`	It is used to create tree node hierarchy.

- Follow the following procedure for creating JTree:
  Step 1 : Create a JTree object.
  Step 2 : Create a JScrollPane object.
  Step 3 : Add tree to scroll pane.
  Step 4 : Add scroll pane to Frame.

Program 2.14: Program for JTree.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class JTreeDemo
{
    JTree tree;
    JFrame  f;
    JTreeDemo()
    {
        f=new JFrame();
        f.setLayout(new BorderLayout());
        // Create top node of tree
        DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");
        DefaultMutableTreeNode a = new DefaultMutableTreeNode("A");
        top.add(a);
        DefaultMutableTreeNode b = new DefaultMutableTreeNode("B");
        top.add(b);
        // Create subtree of "A"
        DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("A1");
        a.add(a1);
        DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("A2");
        a.add(a2);
        // Create subtree of "B"
        DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("B1");
        b.add(b1);
        DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("B2");
        b.add(b2);
        DefaultMutableTreeNode b3 = new DefaultMutableTreeNode("B3");
        b.add(b3);
        tree = new JTree(top);
        JScrollPane jsp = new JScrollPane(tree);
        f.add(jsp, BorderLayout.CENTER);
       f.setSize(500,500);
        f.setVisible(true);
}
 public static void main(String args[])
    {
          JTreeDemo j1=new JTreeDemo();
     }
}
```

Output:



2.4.4 JTable

- A table is a component that displays rows and columns of data. JTable class use to create tables for a GUI based application.
- We can drag the cursor on column boundaries to resize columns. We can also drag a column to a new position.
- Tables are implemented by the JTable class, which extends JComponent.
- JTable class defines following constructors:

```
JTable(Object data[ ][ ], Object colHeads[ ])
JTable(int numRows, int numColumns)
JTable(Vector rowData, Vector columnData)
```

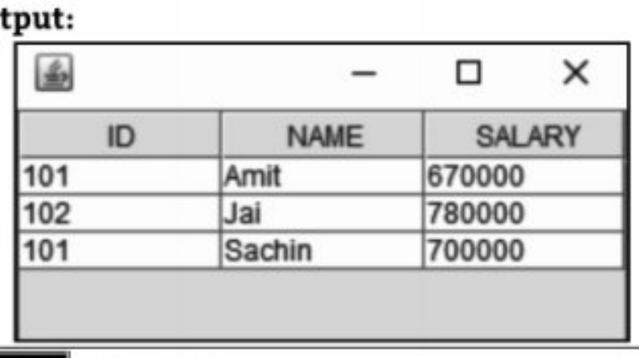
  Here, data is a two-dimensional array of the information to be presented and colHeads is a one-dimensional array with the column headings. The 'numRows' and 'numColumns' are values with which the table is to be created. The 'rowData' and 'columnData' are the vector values by which the table is constructed.
- Here, are the steps for using a table in an applet:
  Step 1 : Create a JTable object.
  Step 2 : Create a JScrollPane object (The arguments to the constructor specify the table and the policies for vertical and horizontal scroll bars).
  Step 3 : Add the table to the scroll pane.
  Step 4 : Add the scroll pane to the container (applet/frame).
- The following program illustrates how to create and use a table.
- A one-dimensional array of strings is created for the column headings. This table has three columns. A two-dimensional array of strings is created for the table cells. We can see that each element in the array is an array of three strings. These arrays are passed to the JTable constructor. The table is added to a scroll pane and then the scroll pane is added to the container.

Program 2.15: Program to demonstrate JTable.

```
import javax.swing.*;
public class JTableDemo
{
    JFrame f;
    JTableDemo()
    {
        f=new JFrame();
         String rows[][]={    {"101","Amit","670000"},
                              {"102","Jai","780000"},
                              {"101","Sachin","700000"}
                              };
String columns[]={"ID","NAME","SALARY"};
```

```
        JTable jt=new JTable(rows,columns);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
          f.add(sp);
          f.setSize(300,400);
          f.setVisible(true);
      }
      public static void main(String[] args)
      {
          new JTableDemo();
      }
  }
```

Output:



2.4.5 JToolTip

- A tooltip is a textual pop-up that momentarily appears when the mouse cursor rests inside the components painting region.
- We can add tooltip text to almost all the components of Java Swing by using the method setToolTipText(String s). This method sets the tooltip of the component to this specified strings.
- When the cursor enters the boundary of that component a popup appears and text is displayed.
- A JToolTip is a small pop-up window designed to contain informative text about a component when the mouse moves over it.
- In short, tooltip is a string has short description and they are used to explain the functionality of the component.

Program 2.16: Program for tooltip.

```
import javax.swing.*;
public class ToolTipExample {
    public static void main(String[] args) {
     JFrame f=new JFrame("Password Field Example");
     //Creating PasswordField and label
     JPasswordField value = new JPasswordField();
     value.setBounds(100,100,100,30);
     value.setToolTipText("Enter your Password");
     JLabel l1=new JLabel("Password:");
     l1.setBounds(20,100, 80,30);
     //Adding components to frame
     f.add(value);  f.add(l1);
     f.setSize(300,300);
     f.setLayout(null);
     f.setVisible(true);
}
}
```

Output:

2.4.6 JProgressBar

- The JProgressBar class is used to display the progress of the task. It inherits JComponent class.
- JProgressBar visually displays the progress of some specified task. JProgressBar shows the percentage of completion of specified task.
- The progress bar fills up as the task reaches it completion. In addition to show the percentage of completion of task, it can also display some text.

Constructors of JProgressBar Class:

Sr. No.	Constructors	Description
1.	`JProgressBar()`	Creates a horizontal progress bar but no string text.
2.	`JProgressBar(int min, int max)`	Creates a horizontal progress bar with the specified minimum and maximum value.
3.	`JProgressBar(int orient)`	Creates a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
4.	`JProgressBar(int orient, int min, int max)`	Creates a progress bar with the specified orientation, minimum and maximum value.

Methods of JProgressBar Class:

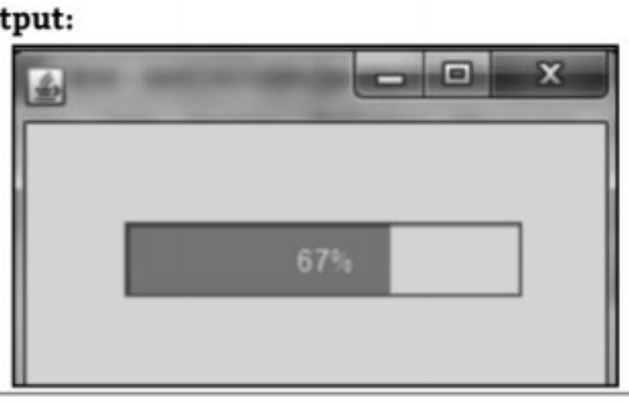
Sr. No.	Methods	Description
1.	`void setStringPainted(boolean b)`	It is used to determine whether string should be displayed.
2.	`void setString(String s)`	It is used to set value to the progress string.
3.	`void setOrientation(int orientation)`	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants. VERTICAL and SwingConstants.HORIZONTAL constants.
4.	`void setValue(int value)`	It is used to set the current value on the progress bar.

Program 2.17: Program for progress bar in Swing.

```
import javax.swing.*;
public class ProgressBarExample extends JFrame{
    JProgressBar jb;
```

```
int i=0,num=0;
ProgressBarExample(){
   jb=new JProgressBar(0,2000);
   jb.setBounds(40,40,160,30);
   jb.setValue(0);
   jb.setStringPainted(true);
   add(jb);
   setSize(250,150);
   setLayout(null);
}
public void iterate(){
while(i<=2000){
   jb.setValue(i);
   i=i+20;
try{Thread.sleep(150);}catch(Exception e){}
}
}
public static void main(String[] args) {
   ProgressBarExample m=new ProgressBarExample();
   m.setVisible(true);
   m.iterate();
   }
}
```

Output:



Practice Questions

1. What is Swing?
2. Enlist various features of Swing?
3. With the help of diagram describe following Swing components:
   (i) JComboBox
   (ii) JChechBox
   (iii) JProcessBar.
4. With the help of diagram describe MVC architecture.
5. What is meant by MVC?
6. Explain the text area Swing components with example.
7. Differentiate between Swing and AWT.

8. Which advanced controls used by Swing? Explain one of the min detail.
9. With the help of example describe the term combo box.
10. What is check box? How to create it? Explain with example.
11. Describe JApplet with example.
12. What is radio button? How to create it? Explain with example.
13. Explain MVC architecture diagrammatically.
14. Describe the following terms:
    (i) Scroll panes
    (ii) Tooltip.
    (iii) Tree.

■■■

3... Event Handling

Chapter Outcomes...

- Use delegation event model to develop event driven program for the given problem.
- Use relevant AWT/swing component(s) to handle the given event.
- Use Adapter classes in Java program to solve the given problem.
- Use inner classes in java program to solve the given problem.

Learning Objectives...

- To understand Concept of Event and Delegation Event Model
- To learn Event Handling in Java
- To study Adapter Classes and Inner Classes in Java
- To learn various Event Listener Interfaces in Java

3.0 INTRODUCTION

- In java, events represent all the interaction between an application and its user.
- When a user interacts with a program, the system creates an event that represents the action and delegates it to the event handling code within the program.
- This code determines how to handle the event so that the user gets an appropriate response.
- Change in the state of an object is known as event i.e. event describes the change in state of source.
- Events are generated as result of user interaction with the graphical user interface components.
- In java, events are represented by Objects.

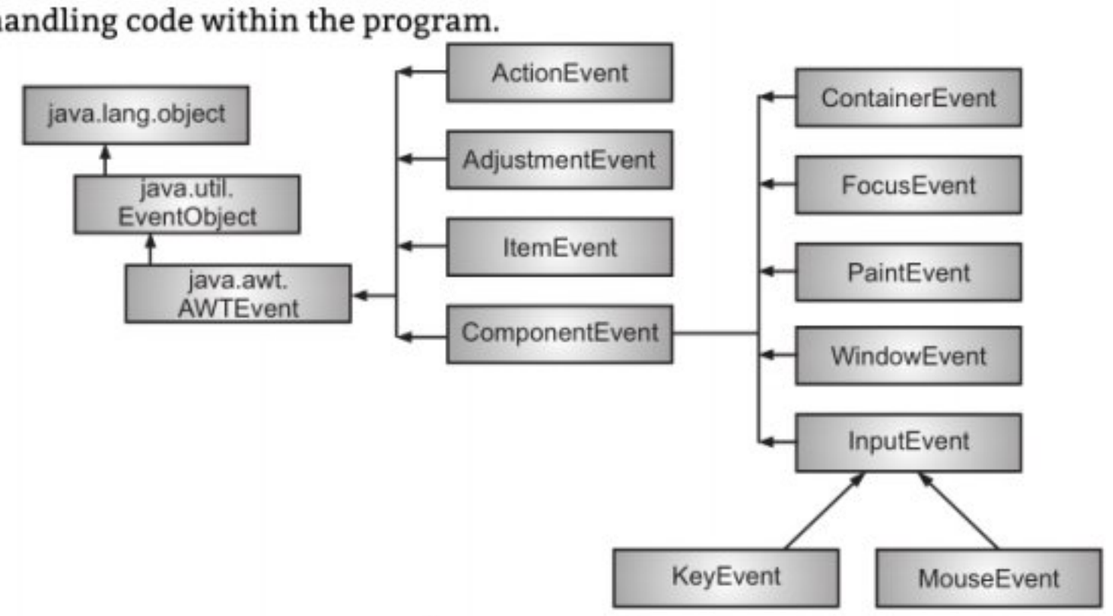


Fig. 3.1: Event Classes

- These objects tell us about event and its source. Examples are ActionEvent (Clicking a button), WindowEvent (Doing something with window e.g. closing, minimizing).
- Some event classes of java.awt.event are shown in Fig. 3.1.

Event Handling:

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have the code which is known as event handler that is executed when an event occurs. Events are generated as result of user interaction with the graphical user interface components.

- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.
- The events can be broadly classified into two categories:

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 (GUI).
   - 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.

- Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

3.1 DELEGATION EVENT MODEL

- There are several types of events. The most commonly handled events are those generated by the mouse, the keyboard, and various controls, such as a push button.
- Events are supported by the java.awt.event package.
- In Java both AWT and Swing components use Event Delegation Model (See Fig. 3.2). In this model, processing of an event is delegated to a particular object (handlers) in the program.
- Event model is quite simple, a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

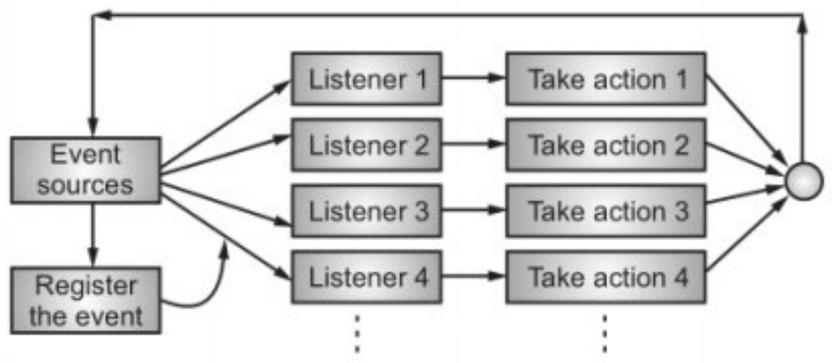


Fig. 3.2: Delegation Event Model

- A user interface element is able to "delegate" the processing of an event to a separate piece of code. In the delegation event model, listeners must register with a source in order to receive an event notification.
- Event delegation model consists of three parts namely, Event, Source and Listener.

3.1.1 Events

- In the delegation model, an event is an object that describes a state change in a source.
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface. For ex. pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

3.1.2 Source

- A source is an object that generates an event (Example, Button). Event sources are components, subclasses of java.awt.Component, capable to generate events.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event.
- Notifications are sent only to listeners that register to receive them.
- Each type of event has its own registration method. The general **form/syntax** of such a method is:

```
public void addTypeListener(TypeListener el)
```

- Here, type is the name of an event and el is reference to the event listeners. For e.g.: the method that registers a mouse motion is called addMouseMotionListener()
- When an event occurs, all registered listeners are notificated and receive a copy of event object and this is called 'multicasting' the event.

- Some sources may allow only one listener to register. It is called 'unicasting' the event.
  Syntax: `public void addTypeListener(TypeListener el) throws java.util.TooManyListenersException`
- A source also allows a listener to unregister an interest in a specific type of event.
  Syntax: `public void removeTypeListener(TypeListener el)`
- Following is list of some of the user interface components that can generate the events. In addition to these graphical user interface elements, other components, such as an applet, can generate events.
- For example, we receive key and mouse events from an applet, (we may also build our own components that generate events).

Description of Event Sources:

1. **Button:** Generates action events when the button is pressed.
2. **Checkbox:** Generates item events when the check box is selected or deselected.
3. **Choice:** Generates item events when the choice is changed.
4. **List:** Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
5. **Menu Item:** Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
6. **Scrollbar:** Generates adjustment events when the scroll bar is manipulated.
7. **Text:** Generates text events when the user enters a character.
8. **Window:** Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

3.1.3 Event Listener

- The events generated by the GUI components are handled by a special group of interfaces known as "listeners".
- A listener is an object that is notified when an event occurs.
- It has following two major requirements:
  1. It must have been registered with one or more sources to receive notifications about specific types of events.
  2. It must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces found in java.awt.event.
- For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface.
- Fig. 3.3 shows Event Listeners.

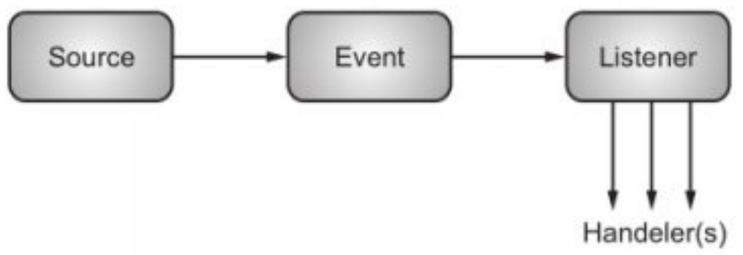


Fig. 3.3: Event Listeners

3.2 EVENT CLASSES

- Event class provide a consistent, easy-to-use means of encapsulating events.
- Almost every event source generates an event and is named by some Java class. For example, the event generated by button is known as ActionEvent and that of Checkbox is known as ItemEvent.
- The class AWT Event, defined within the java.awt package, is a subclass of EventObject. It is the super class, (either directly or indirectly) of all AWT-based events used by the delegation event model.
- The package java.awt.event defines several types of events that are generated by various user interface elements.
- Event class description in Fig. 3.4 are listed below:
  1. **ActionEvent:** Generated when a button is pressed, a list item is double clicked, or a menu item is selected.

2. **AdjustmentEvent:** Generated when a scroll bar is manipulated.
3. **ComponentEvent:** Generated when a component is hidden, moved, resized, or becomes visible.
4. **ContainerEvent:** Generated when a component is added to or removed from a container.
5. **FocusEvent:** Generated when a component gains or losses keyboard focus.
6. **InputEvent:** Abstract super class for all component input event classes.
7. **ItemEvent:** Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
8. **KeyEvent:** Generated when input is received from the keyboard.
9. **MouseEvent:** Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
10. **MouseWheelEvent:** Generated when the mouse wheel is moved.
11. **TextEvent:** Generated when the value of a text area or text field is changed.
12. **WindowEvent:** Generated when a window is activeted, closed, deactivated, deiconified, iconified, opened or quit.

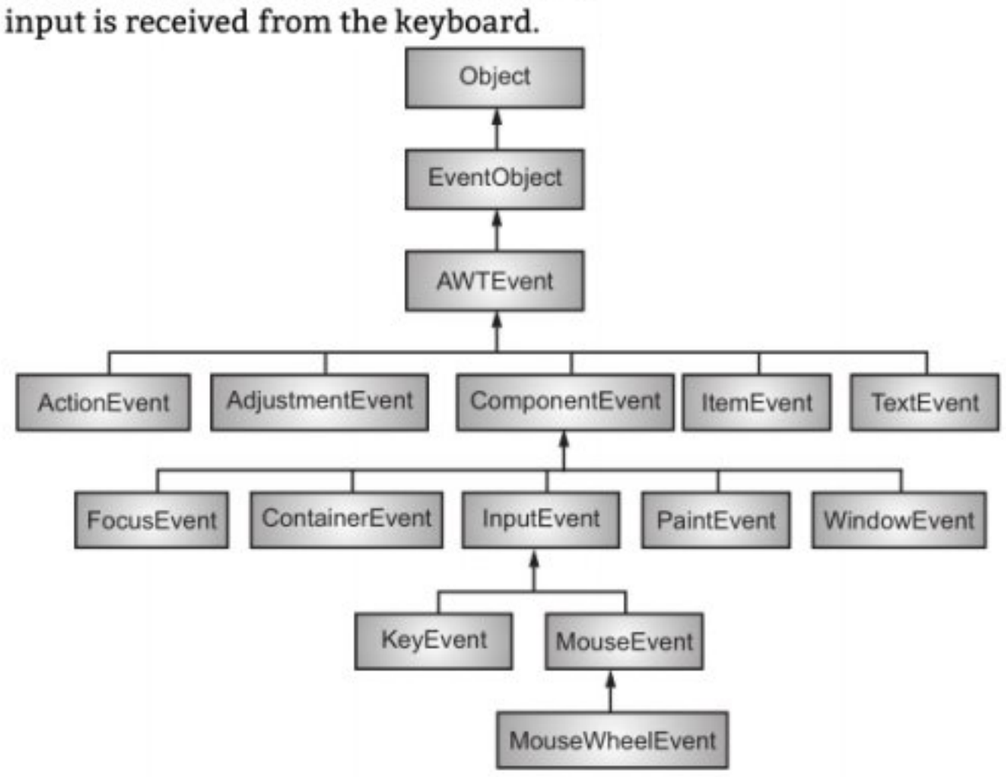


Fig. 3.4: Event Classes Hierarchy

3.2.1 ActionEvent Class

- The ActionEvent class is defined in java.awt.event package.
- An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
- The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event:
  `ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK.`
- In addition, there is an integer constant, ACTION_PERFORMED, which can be used to identify action events.

Constructors of ActionEvent Class:

Sr. No.	Constructors	Description
1.	`ActionEvent(java.lang.Object source, int id, java.lang.String command)`	Constructs an ActionEvent object.
2.	`ActionEvent(java.lang.Object source, int id, java.lang.String command, int modifiers)`	Constructs an ActionEvent object with modifier keys.

- We can obtain the command name for the invoking ActionEvent object by using the getActionCommand( ) method.

Syntax: `String getActionCommand( )`

For example, when a button is pressed, an action event is generated that has a command name equal to the label on that button.

3.2.2 ItemEvent Class

- An ItemEvent is generated when a check box or a list item is clicked.
- There are two types of item events, which are identified by the following integer constants:
  1. **DESELECTED:** The user deselected an item.
  2. **SELECTED:** The user selected an item.
- In addition, ItemEvent defines one integer constant, ITEM_STATE_CHANGED, that signifies a change of state.
- The getItem( ) method can be used to obtain a reference to the item that generated an event. Its **syntax** is shown below:
  ```
  Object getItem( )
  ```
- The getItemSelectable( ) method can be used to obtain a reference to the ItemSelectable object that generated an event. Its general **form/syntax** is shown below:
  ```
  ItemSelectable getItemSelectable( )
  ```
- Lists and choices are examples of user interface element that implement the ItemSelectable interface.
- The getStateChange( ) method returns the state change (i.e., SELECTED or DESELECTED) for the event. It is shown below:
  ```
  int getStateChange( )
  ```

3.2.3 KeyEvent Class

- On entering the character the Key event is generated.
- A KeyEvent is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: KEY_PRESSED, KEY_RELEASED and KEY_TYPED.
- The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated. Remember, not all key presses result in characters. For example, pressing the SHIFT key does not generate a character.
- There are many other integer constants that are defined by KeyEvent.
  For example,
  VK_0 through VK_9 and VK_A through VK_Z define the ASCII equivalents of the numbers and letters.
  Here, are some others:

VK_ENTER	VK_ESCAPE	VK_CANCEL	VK_UP
VK_DOWN	VK_LEFT	VK_RIGHT	VK_PAGE_DOWN
VK_PAGE_UP	VK_SHIFT	VK_ALT	VK_CONTROL

- The VK constants specify Virtual Key codes.
- KeyEvent have following **constructors:**
  1. `KeyEvent(Component src, int type, long when, int modifiers,int code)`
  2. `KeyEvent(Component src, int type, long when, int modifiers,int code, char ch)`
- Here, src is a reference to the component that generated the event. The type of the event is specified by type. The system time at which the key was pressed is passed in when. The modifiers argument indicates which modifiers were pressed when this key event occurred.

3.2.4 MouseEvent Class

- This event indicates a mouse action occurred in a component.
- There are eight types of mouse events. The MouseEvent class defines the following integer constants that can be used to identify them:
  1. **MOUSE_CLICKED:** The user clicked the mouse.
  2. **MOUSE_DRAGGED:** The user dragged the mouse.
  3. **MOUSE_ENTERED:** The mouse entered a component.
  4. **MOUSE_EXITED:** The mouse exited from a component.
  5. **MOUSE_MOVED:** The mouse moved.
  6. **MOUSE_PRESSED:** The mouse was pressed.
  7. **MOUSE_RELEASED:** The mouse was released.
  8. **MOUSE_WHEEL:** The mouse wheel was moved.

- The most commonly used methods in this class are getX( ) and getY( ). These returns the X and Y coordinate of the mouse when the event occurred.
- Their **form/syntax** are shown below:

```
int getX()
int getY()
```

- Alternatively, we can use the getPoint() method to obtain the coordinates of the mouse. It is shown below:

```
Point getPoint()
```

- It returns a Point object that contains the X, Y coordinates in its integer members x and y.

3.2.5 TextEvent Class

- Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.
- TextEvent defines the integer constant, TEXT_VALUE_CHANGED.
- The one **constructor** for this class is shown below:

```
TextEvent(Object src, int type)
```

Here, src is a reference to the object that generated this event. The type of the event is specified by type.

- The TextEvent object does not include the characters currently in the text component that generated the event. Instead, your program must use other methods associated with the text component to retrieve that information.

3.2.6 WindowEvent Class

- The object of this class represents the change in state of a window.
- This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.
- There are ten types of window events. The WindowEvent class defines integer constants that can be used to identify them. The constants and their meanings are shown here:
  1. **WINDOW_ACTIVATED:** The window was activated.
  2. **WINDOW_CLOSED:** The window has been closed.
  3. **WINDOW_CLOSING:** The user requested that the window be closed.
  4. **WINDOW_DEACTIVATED:** The window was deactivated.
  5. **WINDOW_DEICONIFIED:** The window was deiconified.
  6. **WINDOW_GAINED_FOCUS:** The window gained input focus.
  7. **WINDOW_ICONIFIED:** The window was iconified.
  8. **WINDOW_LOST_FOCUS:** The window lost input focus.
  9. **WINDOW_OPENED:** The window was opened.
  10. **WINDOW_STATE_CHANGED:** The state of the window changed.
- WindowEvent is a subclass of ComponentEvent. The most commonly used method in this class is getWindow( ). It returns the Window object that generated the event. Its general form/syntax is shown below:

```
Window getWindow( )
```

3.3 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 we want to receive and process only some of the events that are handled by a particular event listener interface.
- We can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which we are interested. For example, the MouseMotionAdapter class has two methods, mouseDragged( ) and mouseMoved( ).

- The signatures of these empty methods are exactly as defined in the MouseMotionListener interface. If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter and implement mouseDragged( ). The empty implementation of mouseMoved( ) would handle the mouse motion events for you.
- List below shows the commonly used adapter classes in java.awt.event and the interface that each class implements.

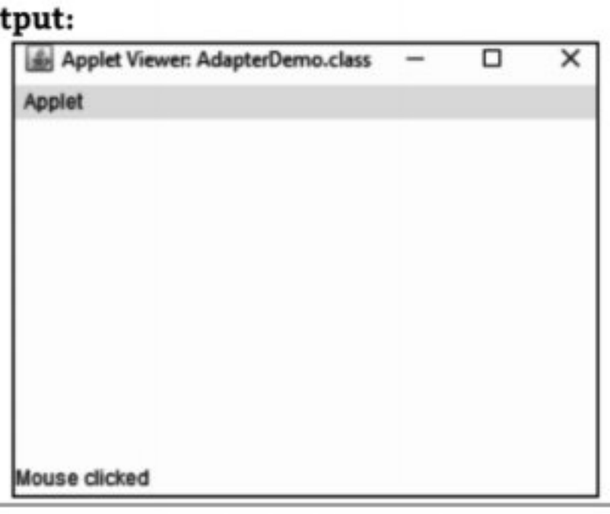
Adapter Class	Listener Interface
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

- The following program demonstrates an adapter. It displays a message in the status bar of an applet viewer or browser when the mouse is clicked or dragged. However, all other mouse events are silently ignored.
- The program has following three classes:
  1. **AdapterDemo** extends Applet. Its init( ) method creates an instance of MyMouseAdapter and registers that object to receive notifications of mouse events. It also creates an instance of MyMouseMotionAdapter and registers that object to receive notifications of mouse motion events. Both of the constructors take a reference to the applet as an argument.
  2. **MyMouseAdapter** class implements the mouseClicked( ) method. The other mouse events are silently ignored by code inherited from the MouseAdapter class.
  3. **MyMouseMotionAdapter** class implements the mouseDragged( ) method. The other mouse motion event is silently ignored by code inherited from the MouseMotionAdapter class.

Program 3.1: Program to demonstrate adapter classes.

```
import java.awt.event.*;
import java.applet.*;
/* <applet code="AdapterDemo" width=300 height=100></applet> */
public class AdapterDemo extends Applet {
    public void init() {
        addMouseListener(new MyMouseAdapter(this));
        addMouseMotionListener(new MyMouseMotionAdapter(this));
    }
}
class MyMouseAdapter extends MouseAdapter {
    AdapterDemo adapterDemo;
    public MyMouseAdapter(AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }
// Handle mouse clicked
    public void mouseClicked(MouseEvent me) {
        adapterDemo.showStatus("Mouse clicked");
    }
}
class MyMouseMotionAdapter extends MouseMotionAdapter {
    AdapterDemo adapterDemo;
    public MyMouseMotionAdapter(AdapterDemo adapterDemo) {
        this.adapterDemo = adapterDemo;
    }
// Handle mouse dragged
    public void mouseDragged(MouseEvent me) {
        adapterDemo.showStatus("Mouse dragged");
    }
}
```

Output:



- As we can see by looking at the program, not having to implement all of the methods defined by the MouseMotionListener and MouseListener interfaces which saves our considerable amount of effort and prevents our code from becoming cluttered with empty methods.

3.4 INNER CLASSES

- Inner classes are class within class.
- Inner class instance has special relationship with Outer class. This special relationship gives inner class access to member of outer class as if they are the part of outer class.
- There are four types of inner classes i.e. member, static member, local and anonymous.

1. Member Inner Class:

- A member class is defined at the top level of the class. It may have the same access modifiers as variables (public, protected, package, static, final), and is accessed in much the same way as variables of that class.

Program 3.2: Program to demonstrate member inner class.

```
public class OuterClass {
    int outerVariable = 100;
    class MemberClass {
        int innerVariable = 20;
        int getSum(int parameter) {
            return innerVariable + outerVariable + parameter;
        }
    }
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        MemberClass inner = outer.new MemberClass();
        System.out.println(inner.getSum(3));
        outer.run();
    }
    void run() {
        MemberClass localInner = new MemberClass();
        System.out.println(localInner.getSum(5));
    }
}
```

Output:

```
123
125
```

2. Static Member Inner Class:

- A static member class is defined like a member class, but with the keyword static. Despite its position inside another class, a static member class is actually an "outer" class--it has no special access to names in its containing class.
- To refer to the static inner class from a class outside the containing class, use the **syntax**, `OuterClassName.InnerClassName`. A static member class may contain static fields and methods.

Program 3.3: Program to demonstrate static member inner class.

```
public class OuterClass {
    int outerVariable = 100;
    static int staticOuterVariable = 200;
    static class StaticMemberClass {
        int innerVariable = 20;
        int getSum(int parameter) {
            // Cannot access outerVariable here
            return innerVariable + staticOuterVariable + parameter;
        }
    }
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        StaticMemberClass inner = new StaticMemberClass();
        System.out.println(inner.getSum(3));
        outer.run();
    }
    void run() {
        StaticMemberClass localInner = new StaticMemberClass();
        System.out.println(localInner.getSum(5));
    }
}
```

Output:

```
223
225
```

3. Local Inner Class:

- A local inner class is defined within a method, and the usual scope rules apply to it. It is only accessible within that method, therefore access restrictions (public, protected, package) do not apply.
- However, because objects (and their methods) created from this class may persist after the method returns, a local inner class may not refer to parameters or non-final local variables of the method.

Program 3.4: Program to demonstrate local inner class.

```
public class OuterClass {
    int outerVariable = 10000;
    static int staticOuterVariable = 2000;
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        System.out.println(outer.run());
    }
    Object run() {
int localVariable = 666;
final int finalLocalVariable = 300;
class LocalClass {
    int innerVariable = 40;
```

```
            int getSum(int parameter) {
                // Cannot access localVariable here
                return outerVariable + staticOuterVariable
                                     + finalLocalVariable + innerVariable + parameter;
                }
            }
            LocalClass local = new LocalClass();
            System.out.println(local.getSum(5));
            return local;
        }
    }
```

Output:

```
12345
OuterClass$1LocalClass@5e2de80c
```

4. Anonymous Inner Class:

- An amonymous inner class is one that is not assigned a name.
- An anonymous inner class is one that is declared and used to create one object (typically as a parameter to a method), all within a single statement.
- An anonymous inner class may extend a class as given below:

  `new SuperClass(parameters){ class body }`

  Here, SuperClass is not the name of the class being defined, but rather the name of the class being extended. The parameters are the parameters to the constructor for that superclass.
- An anonymous inner class may implement an interface:

  `new Interface(){ class body }`
- Because anonymous inner classes are almost always used as event listeners, the example below uses an anonymous inner class as a button listener.

Program 3.5: Program to demonstrate anonymous inner class.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
public class OuterClass extends JFrame
{
    public static void main(String[] args)
    {
        OuterClass outer = new OuterClass();
        JButton button = new JButton("Don't click me!");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                System.out.println("Oops!!!");
            }
        });
        outer.add(button);
        outer.pack();
        outer.setVisible(true);
    }
}
```

Output:

3.5 EVENT LISTENER INTERFACES

- The delegation event model has two parts i.e. sources (an object on which event occurs.) and listeners (responsible for generating response to an event).
- Listener is also known as event handler.
- The Event listener represent the interfaces responsible to handle events.
- Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.
- Following is the **declaration** for java.util.EventListener interface:

```
public interface EventListener
```

- When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

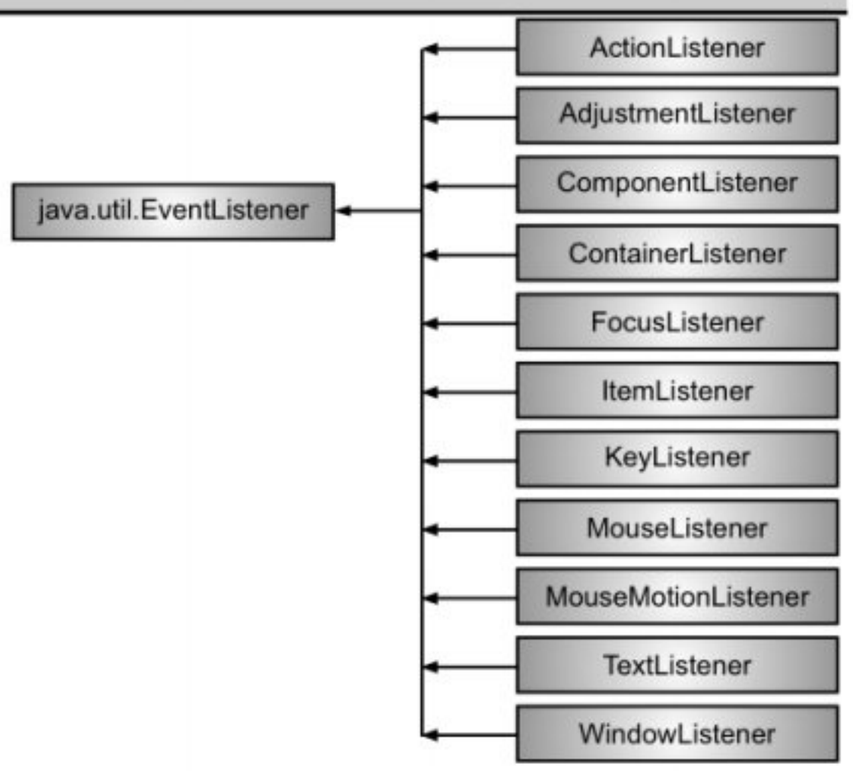


Fig. 3.5: Event Listener in Java

3.5.1 ActionListener Interface

- This interface is used for receiving the action events.
- This interface defines the actionPerformed( ) method that is invoked when an action event occurs.
- Its general **form/syntax** is shown below:

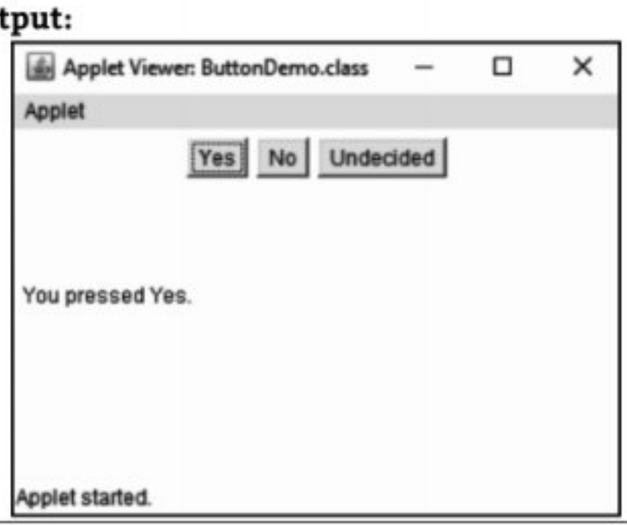
```
void actionPerformed(ActionEvent ae)
```

Program 3.6: Handling buttons events using ActionListener interface.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
 */
public class ButtonDemo extends Applet implements ActionListener {
    String msg = "";
    Button yes, no, maybe;
    public void init() {
        yes = new Button("Yes");
        no = new Button("No");
        maybe = new Button("Undecided");
        add(yes);
        add(no);
        add(maybe);
        yes.addActionListener(this);
        no.addActionListener(this);
        maybe.addActionListener(this);
}
public void actionPerformed(ActionEvent ae) {
    String str = ae.getActionCommand();
```

```
        if (str.equals("Yes")) {
            msg = "You pressed Yes.";
        } else if (str.equals("No")) {
            msg = "You pressed No.";
        } else {
            msg = "You pressed Undecided.";
        }
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString(msg, 6, 100);
    }
}
```

Output:



3.5.2 ItemListener Interface

- This interface is used for receiving the item events.
- This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes.
- Its general **form/syntax** is shown below:

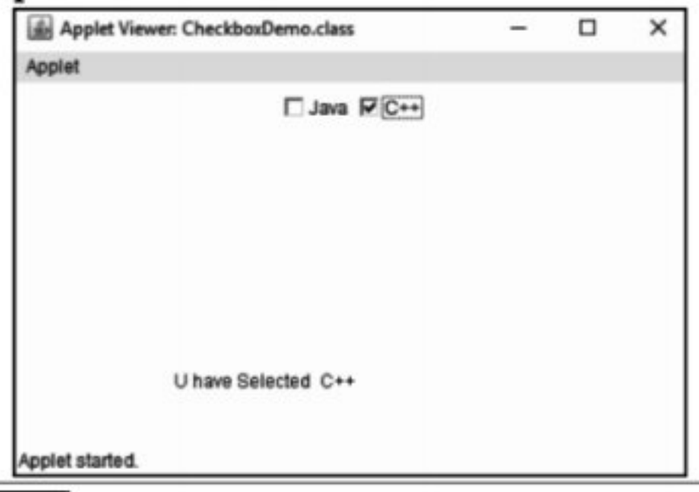
```
void itemStateChanged(ItemEvent ie)
```

Program 3.7: Handling checkboxes events using ItemListener interface.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CheckboxDemo" width=250 height=150></applet>
 */
public class CheckboxDemo extends Applet implements ItemListener {
   Checkbox cb1,cb2;
   String msg="";
      public void init()
       {
         cb1=new Checkbox("Java",false);
cb1.setBounds(100,200,50,50);
add(cb1);
cb2=new Checkbox("C++",false);
cb2.setBounds(100,300,50,50);
```

```
            add(cb2);
            cb1.addItemListener(this);
            cb2.addItemListener(this);
            }
            public void itemStateChanged(ItemEvent ae)
            {
                if(cb1.getState() && cb2.getState())
                            msg= cb1.getLabel() +"  "+ cb2.getLabel();
                else if(cb1.getState()==false && cb2.getState()==false)
                            msg= "";
                else if(cb2.getState())
                            msg= cb2.getLabel();
                else if(cb1.getState())
                            msg= cb1.getLabel() ;
                      repaint();
        }
            public void paint(Graphics g)
            {
                g.drawString(" U have Selected  "+ msg,100,200 );
            }
    }
```

Output:



3.5.3 KeyListener Interface

- The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addKeyListener() method.

Methods of KeyListener Interface:

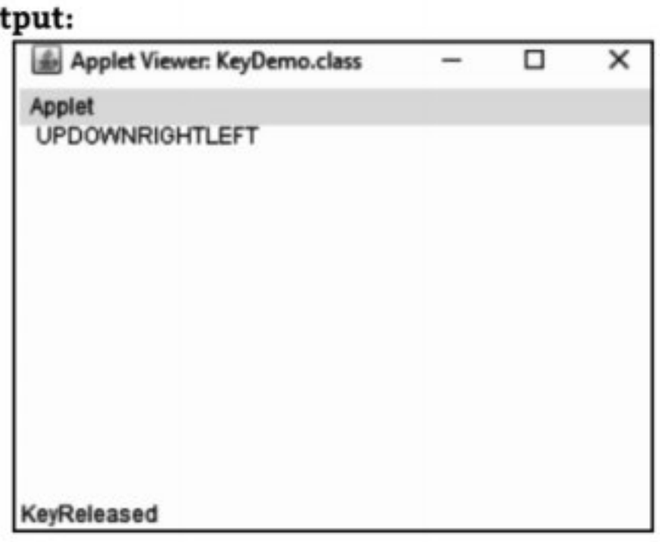
1. `void keyPressed(KeyEvent e)`: Invoked when a key has been pressed.
2. `void keyReleased(KeyEvent e)`: Invoked when a key has been released.
3. `void keyTyped(KeyEvent e)`: Invoked when a key has been typed.

Program 3.8: Handling key events.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*<applet code="KeyDemo" width=300 height=400></applet>*/
public class KeyDemo extends Applet implements KeyListener {
    String msg = "";
    public void init() {
        addKeyListener(this);
    }
```

```
    public void keyReleased(KeyEvent k) {
        showStatus("KeyReleased");
        repaint();
    }
    public void keyTyped(KeyEvent k) {
        msg += k.getKeyCode();
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString(msg, 10, 10);
    }
    public void keyPressed(KeyEvent k) {
        int key = k.getKeyCode();
        switch (key) {
            case KeyEvent.VK_F1:
                msg += "F1";
                break;
            case KeyEvent.VK_F2:
                msg += "F2";
                break;
            case KeyEvent.VK_F3:
                msg += "F3";
                break;
            case KeyEvent.VK_F4:
                msg += "F4";
                break;
            case KeyEvent.VK_RIGHT:
                msg += "RIGHT";
                break;
            case KeyEvent.VK_LEFT:
                msg += "LEFT";
                break;
            case KeyEvent.VK_UP:
                msg += "UP";
                break;
            case KeyEvent.VK_DOWN:
                msg += "DOWN";
                break;
        }
    }
}
```

Output:



3.5.4 MouseListener Interface

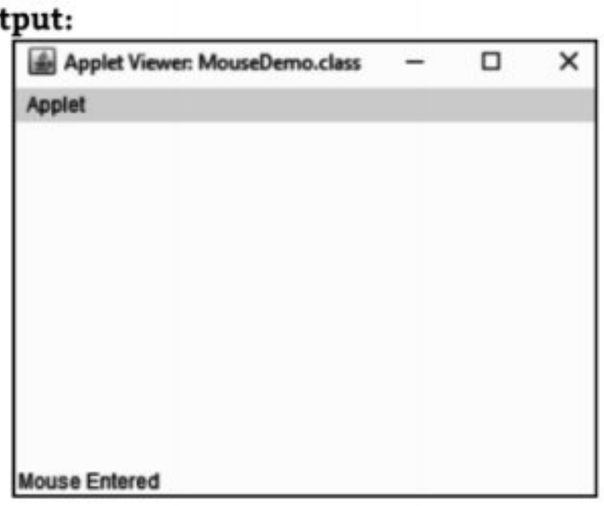
- This interface is used for receiving the key events.
- This interface defines five methods. If the mouse is pressed and released at the same point, mouseClicked( ) is invoked.
- When the mouse enters a component, the mouseEntered( ) method is called.
- When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.
- The general **forms** of these methods are shown below:

```
void mouseClicked(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)
```

Program 3.9: Handling mouse events.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*/*<applet code="MouseDemo" width=300 height=400></applet>*/
public class MouseDemo extends Applet implements MouseListener {
    public void init() {
        addMouseListener(this);
    }
    public void mouseClicked(MouseEvent me) {
        showStatus("Mouse Clicked");
        repaint();
    }
    public void mousePressed(MouseEvent me) {
        showStatus("Mouse Pressed");
        repaint();
    }
    public void mouseReleased(MouseEvent me) {
        showStatus("Mouse Released");
        repaint();
    }
    public void mouseEntered(MouseEvent me) {
        showStatus("Mouse Entered");
        repaint();
    }
    public void mouseExited(MouseEvent me) {
        showStatus("Mouse Exited");
        repaint();
    }
}
```

Output:



3.5.5 MouseMotionListener Interface

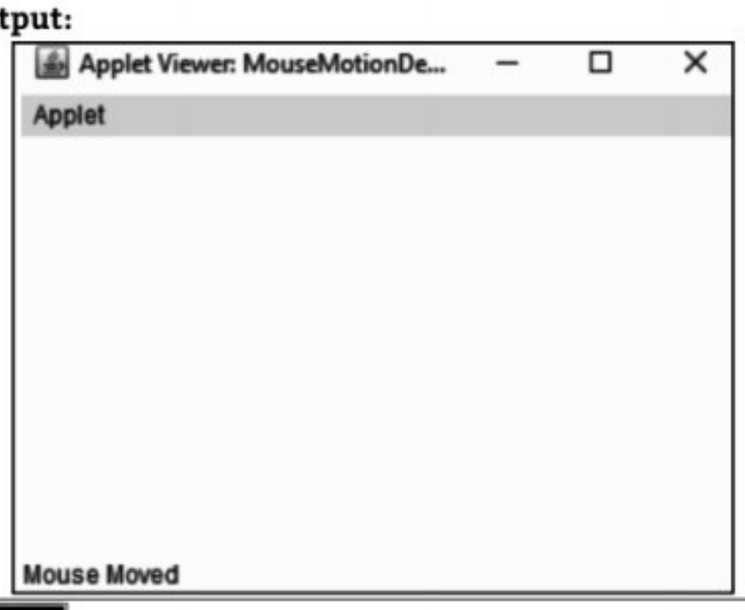
- This interface is used for receiving the mouse motion events.
- This interface defines following two methods:
  1. The `mouseDragged()` method is called multiple times as the mouse is dragged.
  2. The `mouseMoved()` method is called multiple times as the mouse is moved.
- Their general **forms** are shown here:

```
void mouseDragged(MouseEvent me)
void mouseMoved(MouseEvent me)
```

Program 3.10: Handling mouse motion events.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*/*<applet code="MouseMotionDemo" width=300 height=400></applet>*/
public class MouseMotionDemo extends Applet implements MouseMotionListener
{
    public void init()    {
        addMouseMotionListener(this);
    }
    public void mouseMoved(MouseEvent m)    {
        showStatus("Mouse Moved");
        repaint();
    }
    public void mouseDragged(MouseEvent m)    {
        showStatus("Mouse Dragged");
        repaint();
    }
}
```

Output:



3.5.6 TextListener Interface

- This interface is used for receiving the text events. This interface allow to react to changes the text contained in components of type TextField and TextArea.
- This interface defines only one method textValueChanged(TextEvent e) method. The general **form/syntax** is shown below:

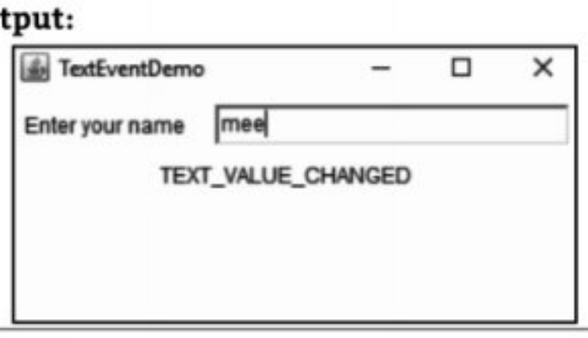
```
public void textValueChanged(TextEvent e)
```

- This method is called when a value in TextField or TextArea is entered or edited.
- The addTextListener () method enables a text component to generate user events.

Program 3.11: Program for TextListener interface.

```
import java.awt.*;
import java.awt.event.*;
public class TextEventDemo implements TextListener {
    Label label1, label2;
    TextField field1;
    Frame jf;
    String str;
    TextEventDemo() {
        jf = new Frame("TextEventDemo");
        label1 = new Label("Enter your name");
        label2 = new Label();
        field1 = new TextField(25);
        jf.setLayout(new FlowLayout());
        jf.add(label1);
        jf.add(field1);
        jf.add(label2);
        field1.addTextListener(this);
        jf.setSize(340, 200);
        jf.setVisible(true);
    }
    public void textValueChanged(TextEvent te) {
        label2.setText(te.paramString());
        jf.setVisible(true);
    }
    public static void main(String args[]) {
        new TextEventDemo();
    }
}
```

Output:



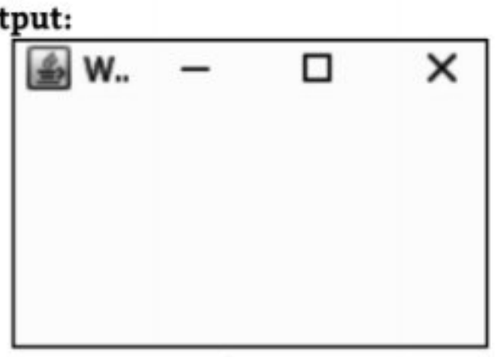
3.5.7 WindowListener Interface

- This interface is used for receiving the window events.
- This interface defines seven methods, these method perform operation on the window. The general **forms** of these methods are given below:
  1. `void windowActivated(WindowEvent we)`: This method is Invoked when a window is activated.
  2. `void windowClosed(WindowEvent we)`: This method is called when a window is closed.
  3. `void windowClosing(WindowEvent we)`: This method is called when a window is being closed.
  4. `void windowDeactivated(WindowEvent we)`: This method is invoked when a window is deactivated.
  5. `void windowDeiconified(WindowEvent we)`: This method is called when a window is deiconified.
  6. `void windowIconified(WindowEvent we)`: This method is called when a window is iconified.
  7. `void windowOpened(WindowEvent we)`: This method is called when a window is opened.

Program 3.12: Program for WindowListener interface.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class WindowDemo extends Frame implements WindowListener {
    WindowDemo() {
        setTitle("Window Listener");
        setBounds(100, 200, 200, 200);
        setVisible(true);
        addWindowListener(this);
    }
    public void windowClosing(WindowEvent e) {
        System.out.println("Window Closing");
        dispose();
        System.exit(0);
    }
    public void windowOpened(WindowEvent e) {
        System.out.println("Window Open");
    }
    public void windowClosed(WindowEvent e) {
        System.out.println("Window Closed");
    }
    public void windowActivated(WindowEvent e) {
        System.out.println("Window Activated");
    }
    public void windowDeactivated(WindowEvent e) {
        System.out.println("Window Deactivated");
    }
    public void windowIconified(WindowEvent e) {
        System.out.println("window Iconified");
    }
    public void windowDeiconified(WindowEvent e) {
        System.out.println("Window Deiconified");
    }
    public static void main(String[] args) {
        WindowDemo frame = new WindowDemo();
    }
}
```

Output:



Output: On Console

```
Window Activated
window Iconified
Window Deactivated
Window Deiconified
Window Activated
Window Closing
```

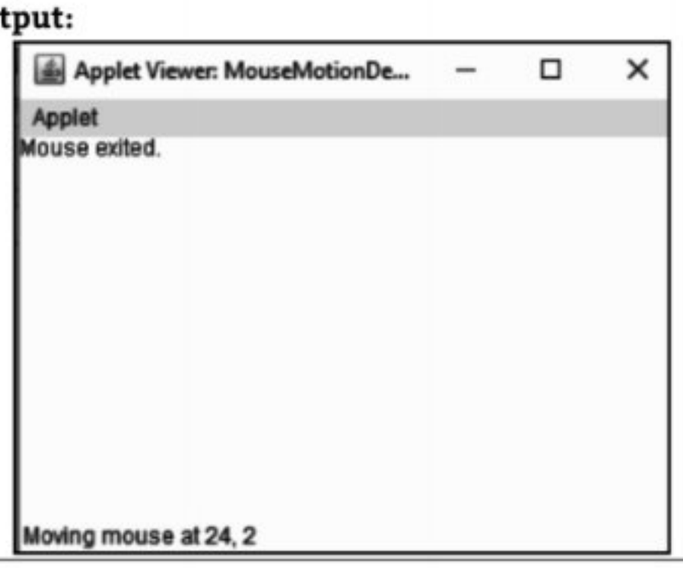
Additional Programs:

Program 1: Handling mouse events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code=" MouseMotionDemo" width=300 height=100>
</applet>*/
public class MouseMotionDemo extends Applet implements MouseListener,
MouseMotionListener {
    String msg = "";
    int mouseX = 0, mouseY = 0; // coordinates of mouse
    public void init() {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
    public void mouseClicked(MouseEvent me) {
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse clicked.";
        repaint();
    }
    public void mouseEntered(MouseEvent me) {
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse entered.";
        repaint();
}
public void mouseExited(MouseEvent me) {
    mouseX = 0;
    mouseY = 10;
    msg = "Mouse exited.";
    repaint();
}
public void mousePressed(MouseEvent me) {
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Down";
    repaint();
}
public void mouseReleased(MouseEvent me) {
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Up";
    repaint();
}
public void mouseDragged(MouseEvent me) {
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "*";
    showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
    repaint();
}
```

```
    public void mouseMoved(MouseEvent me) {
        showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
    }
    public void paint(Graphics g) {
        g.drawString(msg, mouseX, mouseY);
    }
}
```

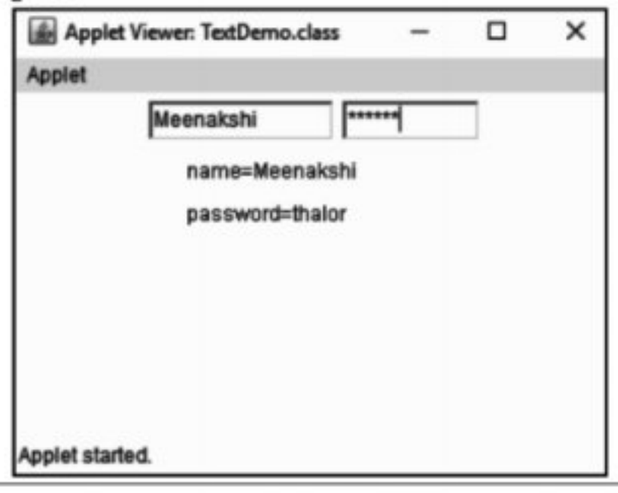
Output:



Program 2: Handling TextField events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="TextDemo" width=250 height=150></applet>
 */
public class TextDemo extends Applet implements ActionListener {
    String msg = "hello";
    TextField t1, t2;
    Label l;
    public void init() {
        t1 = new TextField(12);
        t1.setBounds(50, 100, 100, 100);
        add(t1);
        t2 = new TextField(8);
        t2.setBounds(300, 100, 100, 100);
        t2.setEchoChar('*');
        add(t2);
        t1.addActionListener(this);
        t2.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae) {
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString("name=" + t1.getText(), 100, 50);
        g.drawString("password=" + t2.getText(), 100, 75);
    }
}
```

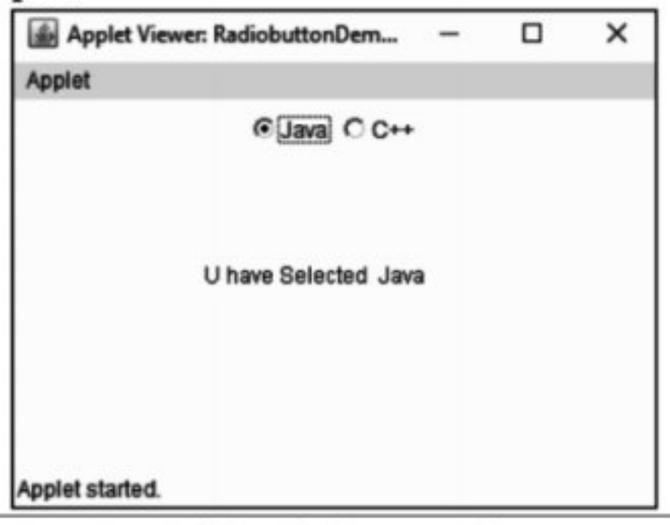
Output:



Program 3: Handling RadioButton events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="RadiobuttonDemo" width=250 height=150></applet>
 */
public class RadiobuttonDemo extends Applet implements ItemListener
{
    Checkbox cb1, cb2;
 CheckboxGroup cbg;
 String msg = "";
 public void init()
{
     cbg = new CheckboxGroup();
     cb1 = new Checkbox("Java", false, cbg);
     cb1.setBounds(100, 200, 50, 50);
     add(cb1);
     cb2 = new Checkbox("C++", false, cbg);
     cb2.setBounds(100, 300, 50, 50);
     add(cb2);
     cb1.addItemListener(this);
     cb2.addItemListener(this);
 }
 public void itemStateChanged(ItemEvent ae)
{
     msg = cbg.getSelectedCheckbox().getLabel();
     repaint();
 }
 public void paint(Graphics g)
    {
         g.drawString(" U have Selected  " + msg, 100, 100);
     }
}
```

Output:

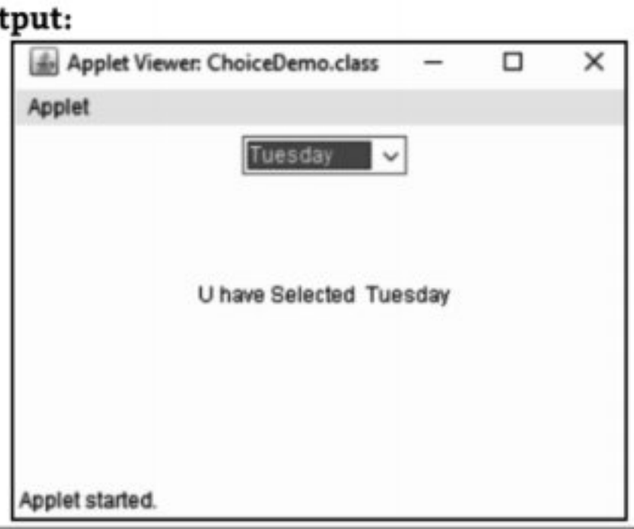


Program 4: Handling Choice control events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ChoiceDemo" width=250 height=150></applet>
 */
public class ChoiceDemo extends Applet implements ItemListener
{
    String msg = "";
 Choice c;
 public void init()
{
     c = new Choice();
     c.add("Sunday");
     c.add("Monday");
     c.add("Tuesday");
     c.add("Wednesday");
     c.setBounds(100, 100, 100, 100);
     add(c);
     c.addItemListener(this);
 }
 public void itemStateChanged(ItemEvent ae)
{
     msg = c.getSelectedItem();
     repaint();
 }
 public void paint(Graphics g)
{
        g.drawString(" U have Selected  " + msg, 100, 100);
    }

}
```

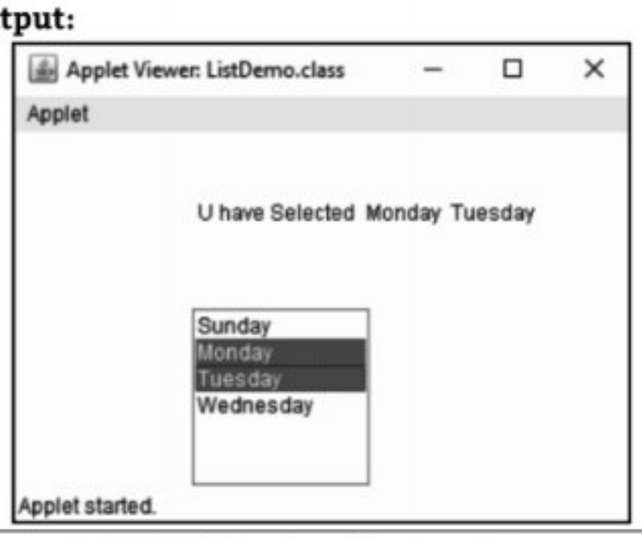
Output:



Program 5: Handling Lists events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ListDemo" width=250 height=150></applet>
 */
public class ListDemo extends Applet implements ItemListener
{
    String msg = "";
    List l;
    public void init()
{
     setLayout(null);
     l = new List(5, true);
     l.add("Sunday");
     l.add("Monday");
     l.add("Tuesday");
     l.add("Wednesday");
     l.setBounds(100, 100, 100, 100);
     add(l);
     l.addItemListener(this);
 }
 public void itemStateChanged(ItemEvent ae)
{
     int index[] = l.getSelectedIndexes();
     for (int i = 0; i < index.length; i++) {
         msg += l.getItem(index[i]) + "  ";
     }
     repaint();
 }
 public void paint(Graphics g)
{
         g.drawString(" U have Selected  " + msg, 100, 50);
         msg = "";
     }
}
```

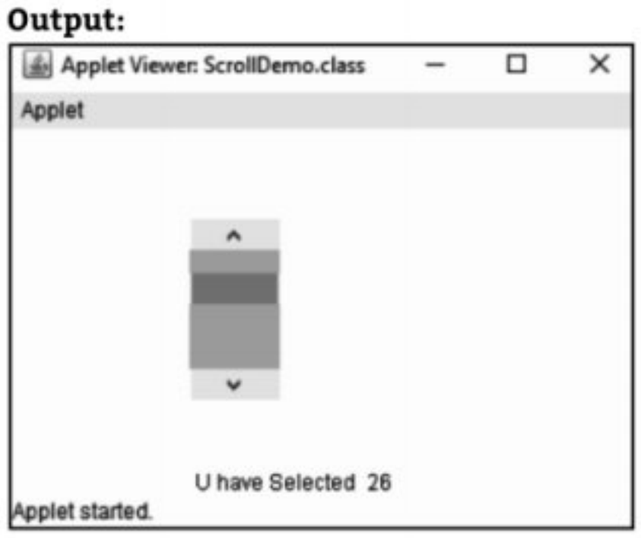
Output:



Program 6: Handling Scrollbar events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ScrollDemo" width=250 height=150></applet>
 */
public class ScrollDemo extends Applet implements AdjustmentListener
{
    String msg = "";
    Scrollbar s;
    public void init()
   {
        setLayout(null);
        s = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 100);
        s.setBounds(100, 50, 50, 100);
        add(s);
        s.addAdjustmentListener(this);
    }
    public void adjustmentValueChanged(AdjustmentEvent ae)
   {
        repaint();
    }
    public void paint(Graphics g)
  {
        msg = "" + s.getValue();
        g.drawString(" U have Selected  " + msg, 100, 200);
    }
}
```

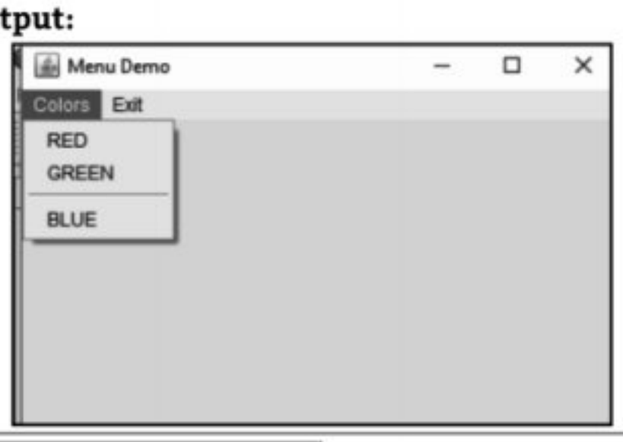
Output:



Program 7: Handling Menus events.

```
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame implements ActionListener
{
    public MenuDemo()
    {
            MenuBar mBar = new MenuBar();
            setMenuBar(mBar);                        // add menu bar to frame
            Menu files = new Menu("Colors");
            Menu exit = new Menu("Exit");
            mBar.add(files);   // add menus to menu bar
            mBar.add(exit);
            MenuItem mi1 = new MenuItem("RED");
            files.add(mi1);                          //add menuitem in menu
            files.add(new MenuItem("GREEN"));
            files.addSeparator();
            files.add(new MenuItem("BLUE"));
            exit.add(new MenuItem("Close"));
            files.addActionListener(this);
            exit.addActionListener(this);
            setTitle("Menu Demo");
            setSize(400, 400);
            setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
         String str = e.getActionCommand();
         if(str.equals("Close"))
         {
             System.exit(0);
      }
         else if(str.equals("RED"))
         {
             setBackground(Color.red);
         }
          else if(str.equals("GREEN"))
         {
             setBackground(Color.green);
         }
          else if(str.equals("BLUE"))
      {
             setBackground(Color.blue);
         }
    }
    public static void main(String args[])
      {
        new MenuDemo();
      }
    }
```

Output:



Practice Questions

1. What is event? How to handle it?
2. Explain what the meaning of event sources is.
3. List out the components who supports ActionEvent ActionListener events?
4. What are different events? Explain any one in detail.
5. Explain event delegation model with diagram.
6. What are the event classes in Java?
7. Explain the following event listener:
   (i) ActionListener
   (ii) FocusListener.
8. What is adapter classes? Explain in detail.
9. Describe inner class in detail.
10. Explain event listener in detail?
11. Describe following listener interfaces with example:
    (i) ItemListener
    (ii) KeyListener
    (iii) TextListener
    (iv) MouseMotionListener
    (v) WindowListener.

■■■

4...

Networking Basics

Chapter Outcomes...

- Use InetAddress class to know the IP address of the given host name.
- Use URLConnection classes to read and write data to the specified resource referred by the given URL.
- Develop program for client/server communication through TCP/IP server sockets for the given problem.
- Write program to illustrate the client/server communication using datagram protocol for the given problem.

Learning Objectives...

- To understand Basic Networking Concepts in Java
- To learn Socket, Client/Server Communication, Internet Addressing etc.
- To study Concept of URL, Datagrams etc.

4.0 INTRODUCTION

- Java is the programming language designed from the ground up with networking in mind. As the global Internet continues to grow, Java is uniquely suited to build the next generation of network applications.
- Java provides solutions to a number of problems - platform independence, security, and international character sets being the most important - that are crucial to Internet applications, yet difficult to address in other programming languages.
- Communication in the present day is vital part of our existence. In today's computing world, there are many expensive resources such as laser printers, fax machines and so on that needs to be shared.
- To facilitate this robust network have come into existence. Networks allow expensive resources to be shared. A network is a collection of computers and other devices that can send data to and receive data from one another.
- Java programming environment provides simple yet robust techniques that permit Java applications to communicate across networks and share valuable resources.
- Java communication is built on standard client/ server architecture as shown in Fig. 4.1.
- A server must have a port number on which some software is listener/talker. This software is consistently listening for client requests.

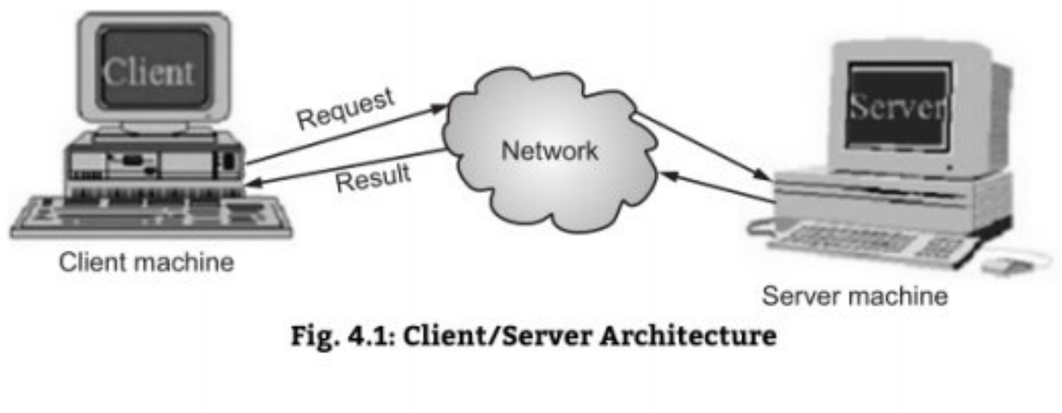


Fig. 4.1: Client/Server Architecture

- Both the client and server on the networks must be uniquely identified. This is where TCP/IP comes into the picture. Using TCP/IP each computer on the network whether a server or a client can be uniquely identified.
- Network feature made the Java programming language, more appropriate for writing networked programs. The Internet is all about connecting machines together.

4.1 SOCKET OVERVIEW

- Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
- When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

4.1.1 Concept of Socket

- Network programming revolves around the concept of sockets. A socket is one end-point of a two-way communication link between two programs running on the network.
- Java programs communicate through programming abstraction called a socket.
- A socket is a bi-directional communication channel between hosts i.e., a computer on a network often termed as host.
- In Java sockets are created with the help of certain classes and these class are found in the java.net package.
- There are separate classes in the java.net package for creating TCP sockets and UDP sockets.
- The client socket is created with the help of the Socket class and the server socket is created using the ServerSocket class.
- Fig. 4.2 shows concept of socket.

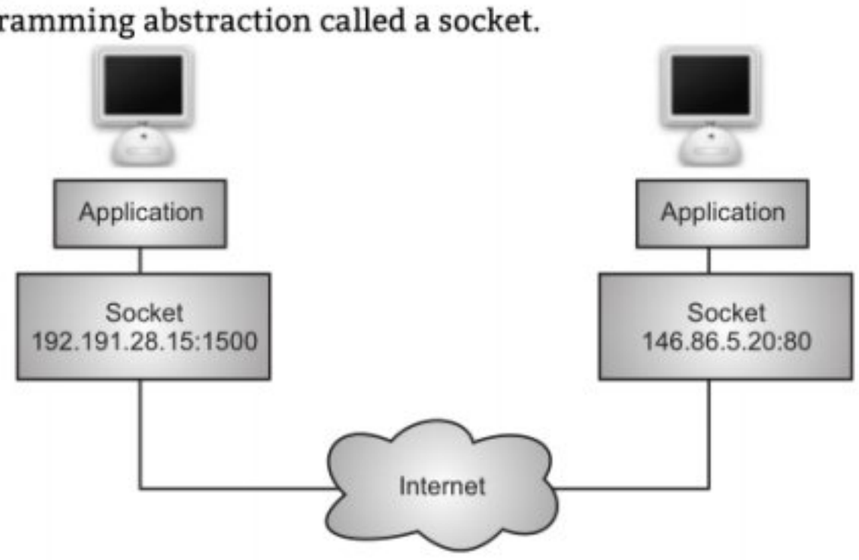


Fig. 4.2: Concept of Socket

4.1.2 Client/Server Networking

- In client/server networking the client and server computers are communicate with each other through well established protocols.
- In client/server networking the client makes as service request to server then server fulfill (response) to client request.
- A server is anything that has some resource that can be shared.
- There are computer servers, which provide computing power like print servers, which manage a collection of printers; disk servers, which provide networked disk space; and web servers, which store web pages.
- A client is simply any other entity that wants to gain access to a particular server. A server is allowed to accept multiple clients connected to the same port number, although each session is unique.
- To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

4.1.3 Reserved Ports / Sockets

- Once connected, a higher-level protocol ensues, which is dependent on which port we are using. A port is an endpoint to a logical connection.
- Port is like a medium through which an application establishes a connection with another application by binding a socket by a port number.

- A port number identifies a specific application running in the machine. A port is a 16-bit unsigned integer number in the range 1 -65535. It is a transport layer address used by TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) to handle communication between applications as shown in Fig. 4.3.

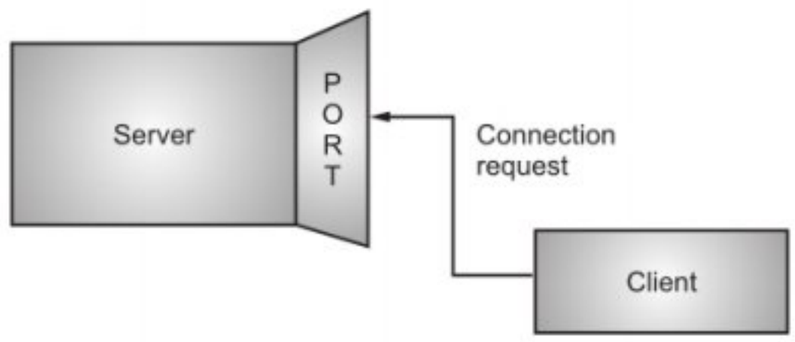


Fig. 4.3: Communication using Port

- The port numbers are divided into three ranges: the well-known ports, the registered ports, and the dynamic or private ports.
- TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to us if we have spent any time surfing the Internet.
- Port number 21 is for FTP, 23 is for Telnet, 25 is for email, 79 is for finger, 80 is for HTTP, 119 is for net-news and the list goes on. It is up to each protocol to determine how a client should interact with the port.
- For example, HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images.
- When a client requests a file from an HTTP server, an action known as a hit, it simply prints the name of the file in a special format to a predefined port and reads back the contents of the file.
- The server also responds with a status code number to tell the client whether the request can be fulfilled and why.

4.1.4 Communication Protocols

- Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as shown in Fig. 4.4.
- When you write Java programs that communicate over the network, you are programming at the application layer. Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the java.net package.

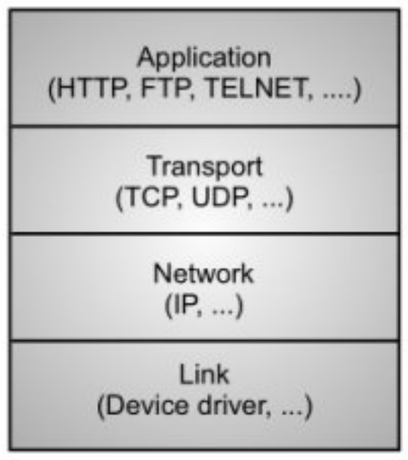


Fig. 4.4: TCP/IP Layers

1. Network Layer Protocol:

(i) Internet Protocol (IP):

- The Internet Protocol (IP) is a Layer 3 protocol, (network layer) that is used to transmit data packets over the Internet.
- Each computer (known as a host) on the Internet has at least one IP address that uniquely identifies it from all other computers on the Internet.
- IP acts as a bridge between networks of different types, forming a worldwide network of computers and smaller subnetworks.
- The Internet Protocol is a packet-switching network protocol. Information is exchanged between two hosts in the form of IP packets, also known as IP datagrams.
- Each datagram is treated as a discrete unit, unrelated to any other previously sent packet, there are no "connections" between machines at the network layer.
- During transmission each message is divided into a number of packets and each packet sent by a different route across the Internet. Packets can arrive in a different order than the order they were sent in. The Internet Protocol just delivers them. It's up to another protocol, the Transmission Control Protocol (TCP) to put them back in the right order.

2. Transport Layer Protocols:

(i) TCP:

- TCP stands for Transmission Control Protocol.
- TCP is a connection-based protocol that provides a reliable flow of data between two computers.

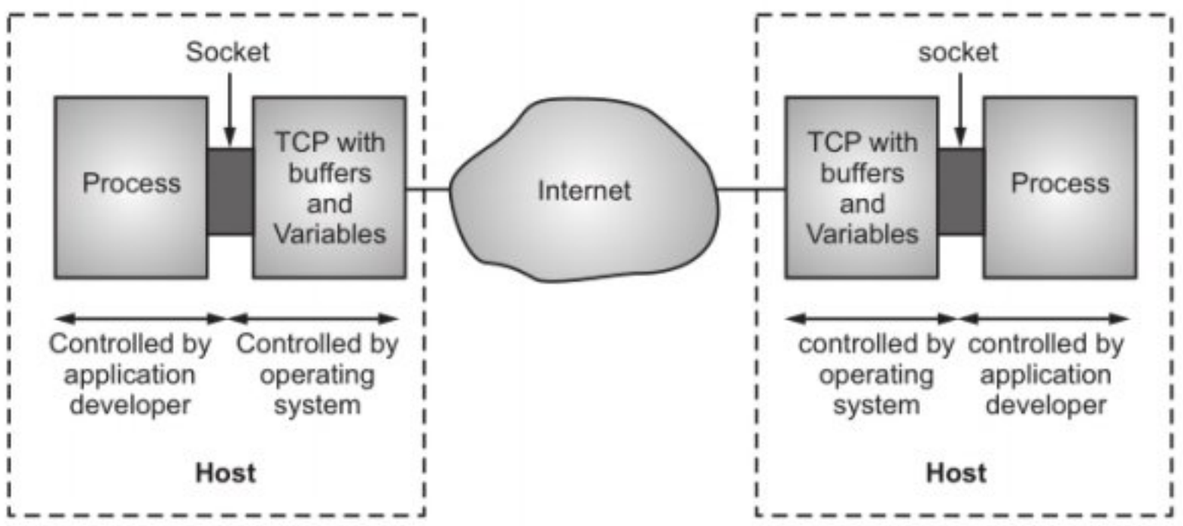


Fig. 4.5: Processes Communicating through TCP Sockets

- TCP allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- As shown in Fig. 4.5, the socket is the door between the application process and TCP. The application developer has control of everything on the application layer side of the socket.
- Now let us to a little closer look at the interaction of the client and server programs. The client has the job of initiating contact with the server.
- In order for the server to be able to react to the client's initial contact, the server has to be ready. This implies two things.
  (a) The server program can not be dormant; it must be running as a process before the client attempts to initiate contact.
  (b) The server program must have some sort of door (i.e., socket) that welcomes some initial contact from a client. (running on an arbitrary machine).
- TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel

(ii) User Datagram Protocol (UPD):

- UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.
- UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination. It is described in RFC 768.
- The UDP is an alternative protocol for sending data over IP that is very quick, but not reliable.
- When you send UDP data, you have no way of knowing whether it arrived, whether different pieces of data arrived in the order in which you sent them. However, the pieces that do arrive generally arrive quickly.
- In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in the Fig. 4.6.

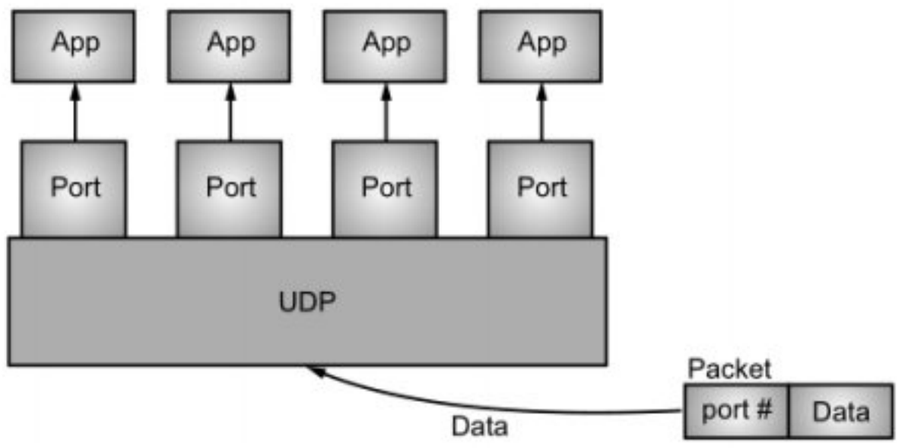


Fig. 4.6: Datagram Based Communication

Comparison between TCP and UDP:

Sr. No.	TCP	UDP
1.	Reliable	Unreliable.
2.	Connection-oriented	Connectionless.
3.	Segment retransmission and flow control through windowing	No windowing or retransmission.
4.	Segment sequencing	No sequencing.
5.	Acknowledge sequencing	No acknowledgment.

3. Application Layer Protocols:

(i) Domain Name System (DNS):

- DNS is the service used to convert human readable names of hosts to IP addresses.
- DNS name is nothing but resolving host names, such as www.yahoo.com, to their corresponding IP addresses.
- The DNS is basically a large database which resides on various computers and it contains the names and IP addresses of various hosts on the internet and various domains.
- While every device on the Internet has an IP address such as 137.170.4.124, humans generally find it easier to refer to machines by names, such as www.niralibooks.org. DNS provides a mechanism for mapping back and forth between IP addresses and host names.
- DNS is a system that is used to map the user friendly host names to their IP addresses and vice versa and used over the Internet.
- Basically, there are a number of DNS servers on the Internet, each listening through UDP software to a port. When a computer on the Internet needs DNS services-for example, to convert a host name such as www.niralibooks.org to a corresponding IP address-it uses the UDP software running on its system to send a UDP message to one of these DNS servers, requesting the IP address.
- The Domain Name System (DNS) works on Client Server model and uses UDP protocol for transport layer communication. DNS uses UDP port 53.

(ii) Simple Mail Transfer Protocol (SMTP):

- The Simple Mail Transfer Protocol (SMTP) controls the transfer of e-mail messages on the Internet. The SMTP defines the interaction between Internet hosts which participate in forwarding emails from a sender to its destination.
- SMTP is a text-based protocol. It works across TCP/IP from ports 25 or 587.
- It operates as the application layer, defining the transport as opposed to the mail content. Handling mails can again be thought of as a client-server system. The mail server uses SMTP to send and receive messages. The client uses SMTP to send mails.

(iii) File Transfer Protocol (FTP):

- FTP is a protocol (or set of rules) which enables files to be transferred between computers. The File Transfer Protocol (FTP) is used widely on the Internet for transferring files to and from a remote host. FTP is commonly used for uploading pages to a Website and for providing online file archives.
- FTP works on the client/server principle. A client program enables the user to interact with a server in order to access information and services on the server computer.
- Files that can be transferred are stored on computers called FTP servers. To access these files, an FTP client program is used. This is an interface that allows the user to locate the file(s) to be transferred and initiate the transfer process.
- FTP uses TCP port 20 for exchanging controlling information and the actual data is sent over TCP port 21.

(iv) Post Office Protocol (POP):

- The Post Office Protocol version 3 (POP 3) is a simple mail retrieval protocol used by User Agents (client email software) to retrieve mails from mail server.

- When a client needs to retrieve mails from server, it opens a connection with the server on TCP port 110. User can then access his mails and download them to the local computer. POP3 works in two modes.
- The most common mode the delete mode, is to delete the emails from remote server after they are downloaded to local machines. The second mode, the keep mode, does not delete the email from mail server and gives the user an option to access mails later on mail server.

(v) HyperText Transfer Protocol (HTTP):

- The primary TCP-based protocol used for communication between web servers and browsers is called the HTTP. HTTP is a key component in the definition of the World Wide Web (WWW) or Web.
- It is set of rules (protocol) that governs the transfer of hypertext between two or more computers. The World Wide Web encompasses the universe of information that is available via HTTP.
- HTTP is a protocol that utilizes TCP to transfer information between computers connected on the web this includes web servers and clients as well. This is the communication mechanism that browsers use to exchange data from clients and servers.
- The client makes an HTTP request to a web server using a web browser and the web server sends the requested information (website) to the client.

4.1.5 Proxy Servers

- A proxy server is software that runs on server that speaks the language of clients. This software hides the actual server from clients that communicate with it.
- A proxy server speaks the client side of a protocol to another server. This is often required when clients have certain restrictions on which servers they can connect to.
- Thus, a client would connect to a proxy server, which did not have such restrictions, and the proxy server would in turn communicate for the client.
- A proxy server has the additional ability to filter certain requests or cache the results of those requests for future use.
- A caching proxy HTTP server can help reduce the bandwidth demands on a local network's connection to the Internet.
- When a popular web site is being hit by hundreds of users, a proxy server can get the contents of the web server's popular pages once, saving expensive Internet work transfers while providing faster access to those pages to the clients.

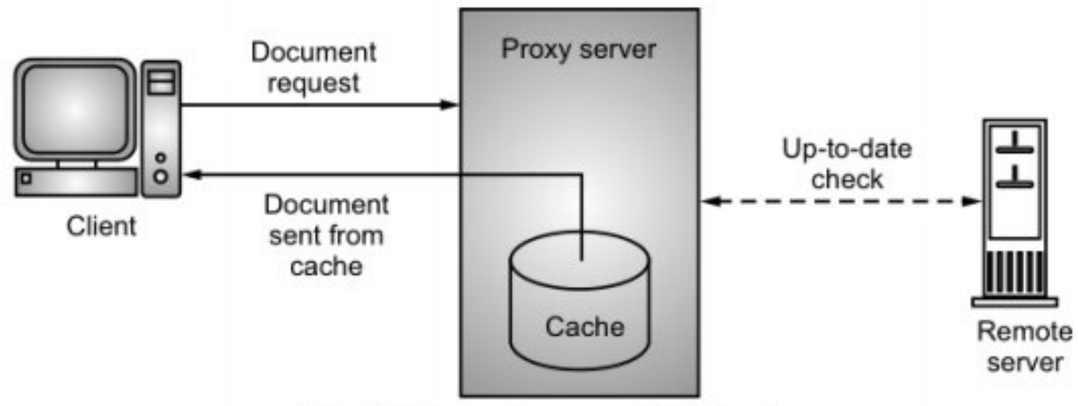


Fig. 4.7: Proxy Document Retrieval

- Caching reduces network traffic and offers faster response time for clients that are using the proxy server instead of going directly to remote servers.

4.1.6 Internet Addressing

- Internet is the world's largest computer network, the network of networks, scattered all over the world. An IP address is a unique identification number allotted to every computer on a network or Internet.
- IP address contains some bytes which identify the network and the actual computer inside the network.
- Every computer on a network is known by a unique number which is known as IP Address.
- Internet Corporation for assigned names and numbers is responsible for assigning IP addresses.
- An IP address is represented by 4 octetes in IPv4 for example, 172.17.167.4. The first octet referred here is the left most of all. The octets numbered as follows depicting dotted decimal notation of IP address as shown in Fig. 4.8.

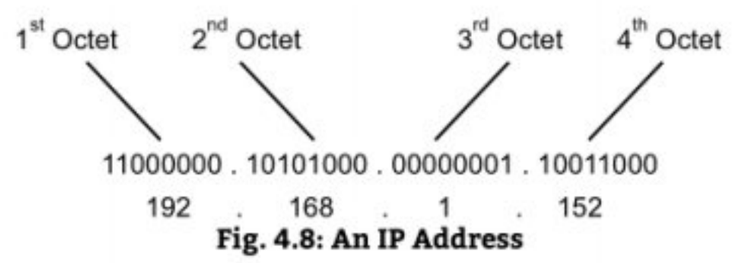


Fig. 4.8: An IP Address

- The number of networks and the number of hosts per class can be derived by following formula:

  Number of networks = $2^{\text{usable network\_bits}} - 2$

  Number of Hosts/Network = $2^{\text{host\_bits}} - 2$
- When calculating hosts' IP addresses, 2 IP addresses are decreased because they cannot be assigned to hosts, i.e. the first IP of a network is network number and the last IP is reserved for Broadcast IP.
- Table 4.1 shows the calculation of number of networks for each class.

1. **Class A Address:**

- The first bit of the first octet is always set to 0 (zero). Thus, the first octet ranges from 1 – 127, i.e.

  00000001 – 01111111

  1 – 127
- Class A addresses only include IP starting from 1.x.x.x to 126.x.x.x only. The IP range 127.x.x.x is reserved for loopback IP addresses.
- The default subnet mask for Class A IP address is 255.0.0.0.

2. **Class B Address:**

- An IP address which belongs to class B has the first two bits in the first octet set to 10, i.e.

  10000000 – 10111111

  128 – 191
- Class B IP Addresses range from 128.0.x.x to 191.255.x.x. The default subnet mask for Class B is 255.255.x.x.

3. **Class C Address:**

- The first octet of Class C IP address has its first 3 bits set to 110, i.e.

  11000000 – 11011111

  192 – 223
- Class C IP addresses range from 192.0.0.x to 223.255.255.x. The default subnet mask for Class C is 255.255.255.x.

4. **Class D Address:**

- Very first four bits of the first octet in Class D IP addresses are set to 1110, giving a range of:

  11100000 – 11101111

  224 – 239
- Class D has IP address rage from 224.0.0.0 to 239.255.255.255. Class D is reserved for Multicasting. In multicasting data is not destined for a particular host, that is why there is no need to extract host address from the IP address, and Class D does not have any subnet mask.

5. **Class E Address:**

- This IP Class is reserved for experimental purposes only for R&D or Study. IP addresses in this class ranges from 240.0.0.0 to 255.255.255.254. Like Class D, this class too is not equipped with any subnet mask.

Table 4.1: IP Class Summary

Class	1st Octet Decimal Range	1st Octet High Order Bits	Network/Host ID (N=Network, H=Host)	Default Subnet Mask	No. of Network ID Bits used to Identify Class	Usable No. of Network ID Bits	No. of Possible Network IDs	Hosts per Network (Usable Addresses)
A	1 – 126	0	N.H.H.H	255.0.0.0	1	8 – 1 = 7	$2^7 - 2$ = 126	16,777,214 ($2^{24} - 2$)
B	128 – 191	10	N.N.H.H	255.255.0.0	2	16 – 2 = 14	$2^{14} - 2$ = 16382	65,534 ($2^{16} - 2$)
C	192 – 223	110	N.N.N.H	255.255.255.0	3	24 – 3 = 21	$2^{21} - 2$ = 2097150	254 ($2^8 - 2$)
D	224 – 239	1110	Reserved for Multicasting					
E	240 – 254	1111	Experimental; used for research					

4.2 JAVA AND THE NET (NETWORKING CLASSES AND INTERFACES)

- Java supports TCP/IP both by extending the already established stream I/O interface and by adding the features required to build I/O objects across the network.
- Java supports both the TCP and UDP protocol families. TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model.
- Java is a premier language for network programming. The java.net package encapsulate large number of classes and interface that provides an easy-to use means to access network resources. Here are some important classes and interfaces of java.net package.

Classes:

- Network classes in Java are given in Table 4.2.

Table 4.2: Network Classes

CacheRequest	InetAddress	Proxy	URL
CookieManager	Socket	Datagrampacket	URLConnection
CookieHandler	ServerSocket	DatagramSocket	

Interfaces:

- Network interfaces in Java are given in Table 4.3.

Table 4.3: Network Interfaces

CookiePolicy	CookieStore	SocketImplFactory
FileNameMap	SocketOption	ProtocolFamily

- Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket, and MulticastSocket classes are for use with UDP.
- In this chapter we will study about below mentioned classes:

1. **InetAddress:** This class represents an Internet Protocol (IP) address.
2. **URL and URLConnection:** Java URL Class present in java.net package, deals with URL (Uniform Resource Locator) which uniquely identify or locate resources on internet.
3. **Socket:** The java.net.Socket class represents the socket that both the client and the server use to communicate with each other.
4. **ServerSocket:** The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests.
5. **DatagramPacket:** It creates a datagram packet.
6. **DatagramSocket:** Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

4.3 INETADDRESS

- Whether we are making a phone call, sending mail, or establishing a connection across the Internet, addresses are fundamental.
- The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address. Java.net package provides different addressing – related classes as shown in Fig. 4.9.
- We interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside.
- The java.net.InetAddress class represents an IP address. The InetAddress class provides methods to get the IP of any host name.
- The java.net.InetAddress class does not define any constructor. To create an InetAddress object, you have to use Factory methods.

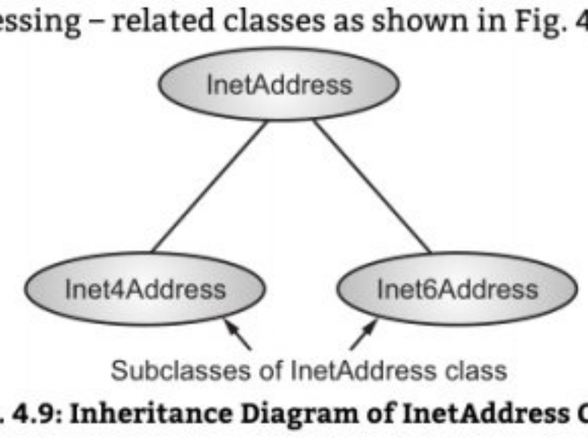


Fig. 4.9: Inheritance Diagram of InetAddress Class

4.3.1 Factory Methods

- We have already know the InetAddress class has no visible constructors. To create an InetAddress object, we have to use one of the available factory methods.
- Factory methods are static methods in a class which return an instance of that class. Three commonly used InetAddress factory methods are shown below:

Sr. No.	Methods	Description
1.	`static InetAddress getByName(String host) throws UnknownHostException`	It returns the instance of InetAddress containing LocalHost IP and name.
2.	`static InetAddress getLocalHost() throws UnknownHostException`	It returns the instance of InetAdddress containing local host name and address.
3.	`static InetAddress getByAddress(byte[] addr) throws UnknownHostException`	Creates an InetAddress object with an IP address and no hostname.
4.	`static InetAddress getByAddress(String hostname, byte[] addr) throws UnknownHostException`	Creates an `InetAddress` object with an IP address and a hostname.
5.	`static InetAddress[] getAllByName (String hostname) throws UnknownHostException`	Determines all the IP addresses of a host, given the host's name.

Program 4.1: Using Factory methods of InetAddress class.

```
import java.net.*;
public class NetDemo1
{
    public static void main(String[] args)
   {
        try {
            InetAddress ip = InetAddress.getByName("www.google.com");
            System.out.println("Host Name: " + ip.getHostName());
            System.out.println("IP Address: " + ip.getHostAddress());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
Host Name: www.google.com
IP Address: 172.217.166.68
```

Program 4.2: Get all the IP addresses associated with host.

```
import java.net.*;
public class NetDemo2 {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getLocalHost();
            System.out.println(address);
            address = InetAddress.getByName("www.google.com");
            System.out.println(address);
            InetAddress sw[] = InetAddress.getAllByName("www.yahoo.com");
            for (int i = 0; i < sw.length; i++) {
                System.out.println(sw[i]);
            }
        } catch (Exception e) {
```

```
            System.out.println(e);
        }
    }
```

Output:

```
Meenakshi/192.168.76.1
www.google.com/172.217.166.68
www.yahoo.com/106.10.250.10
www.yahoo.com/106.10.250.11
```

4.3.2 Instance Methods

- The InetAddress class also has several other methods, which can be used on the objects returned by the methods just discussed.
- Some commonly used InetAddress instance methods are:

Sr. No.	Methods	Description
1.	`boolean equals(Object obj)`	Compare this object with specified object and returns true if this object has the same Internet address as other.
2.	`byte[] getAddress()`	This method returns the IP address associated with this object as an array of bytes in network order.
3.	`String getHostAddress()`	This method returns a string representation of the IP address associated with this object. For example: "206.175.64.78".
4.	`String getHostName()`	Returns the hostname associated with this object.
5.	`int hashCode()`	Returns the hashcode based on the IP address of the object.
6.	`boolean isMulticastAddress()`	True if this object represents a multicast address; false otherwise.
7.	`String toString()`	This method returns a String that contains both the hostname and IP address of this object.

Program 4.3: Using Instance methods of InetAddress class.

```
import java.net.*;
public class NetDemo3 {
    public static void main(String[] args) {
        try {
          InetAddress local = InetAddress.getLocalHost();
          String address=local.getHostAddress();
          String hostName=local.getHostName();
          System.out.println("Local Host IP Address :"+address);
          System.out.println("Local Host Name :"+hostName);
            } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
Local Host IP Address :192.168.76.1
Local Host Name :Meenakshi
```

4.4 TCP/IP SOCKETS

- A socket is one endpoint of a two-way communication link between two programs running on the computer network.
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.
- Java offers good support for TCP sockets, in the form of two socket classes, java.net.Socket and java.net.ServerSocket.
- The java.net.Socket class represents a TCP client socket, which connects to a server socket and infinite protocol exchanges.
- The java.net.ServerSocket class is used to create a server that listens for incoming connections from one or more clients.

4.4.1 Socket Class

- TCP/IP sockets are used to implement to reliable, bi-directional, point-to-point, and stream-based connections between hosts on the Internet.
- A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.
- There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients.
- The ServerSocket class is designed to be a listener, which waits for clients to connect before doing anything.
- The Socket class is designed to connect to server sockets and initiate protocol exchanges.

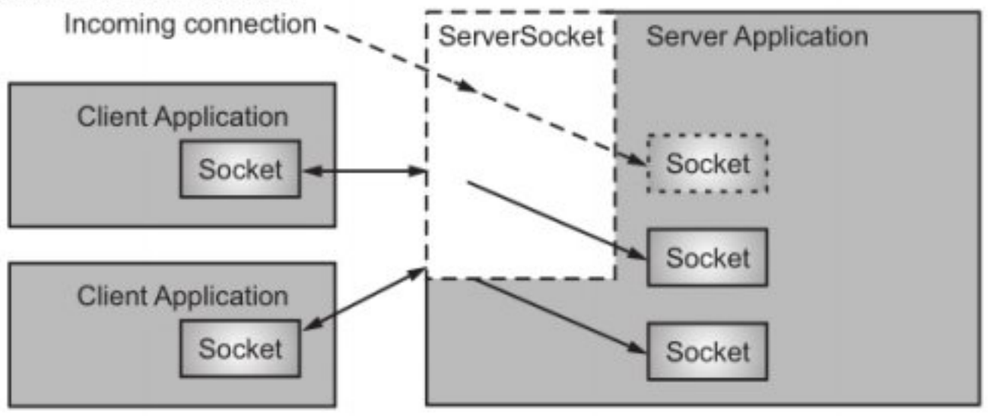


Fig. 4.10: Socket and ServerSocket Connectivity

Constructors of Socket Class: Clients uses following constructors to connect to a server.

Sr. No.	Constructors	Description
1.	`Socket(String host, int port) throws UnknownHostException, IOException`	This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2.	`Socket(InetAddress host, int port) throws IOException`	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
3.	`Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException`	Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4.	`Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException`	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
5.	`Socket()`	Creates an unconnected socket. Use the connect() method to connect this socket to a server.

Methods of Socket Class:

Sr. No.	Methods	Description
1.	`void connect(SocketAddress host, int timeout) throws IOException`	This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.
2.	`InetAddress getInetAddress()`	This method returns the address of the other computer that this socket is connected to.
3.	`int getPort()`	Returns the port the socket is bound to on the remote machine.
4.	`int getLocalPort()`	Returns the port the socket is bound to on the local machine.
5.	`SocketAddress getRemoteSocketAddress()`	Returns the address of the remote socket.
6.	`InputStream getInputStream() throws IOException`	Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7.	`OutputStream getOutputStream() throws IOException`	Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
8.	`void close() throws IOException`	Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Program 4.4: Get a socket's information.

```
import java.net.*;
import java.io.*;
public class WindowDemo {
    public static void main(String args[]) {
            try {
                Socket theSocket = new Socket("www.google.com", 80);
                System.out.println("Connected to "
                        + theSocket.getInetAddress()
                        + " on port " + theSocket.getPort() + " from port "
                        + theSocket.getLocalPort() + " of "
                        + theSocket.getLocalAddress());
            } // end try
            catch (UnknownHostException e) {
                System.err.println("I can't find https://www.google.com" );
            } catch (SocketException e) {
                System.err.println("Could not connect to https://www.google.com " );
            } catch (IOException e) {
                System.err.println(e);
            }
        } // end main
    } // end SocketInfo
```

Output:

```
Connected to www.google.com/172.217.27.196 on port 80 from port 53771 of /192.168.1.4
```

4.4.2 ServerSocket Class

- Java has a different socket class that must be used for creating server applications.
- The ServerSocket class is used to create servers that listen for either local or remote client programs to connect to them on published ports.
- ServerSockets are quite different from normal Sockets. When we create a ServerSocket, it will register itself with the system as having an interest in client connections.
- The constructors for ServerSocket reflect the port number that we wish to accept connections.
- In Java, the basic life cycle of a server program is:
  1. A new ServerSocket is created on a particular port using a ServerSocket() constructor.
  2. The ServerSocket listens for incoming connection attempts on that port using its accept( ) method. A ServerSocket operates in a loop that repeatedly accepts connections. Each pass through the loop invokes the accept( ) method. The accept( ) method returns a Socket object representing the connection between the remote client and the local server.
  3. Depending on the type of server, either the Socket's getInputStream() method, getOutputStream( ) method, or both are called to get input and output streams that communicate with the client.
  4. The server and the client interact according to an agreed-upon protocol until it is time to close the connection. The server the client, or both can close the connection.
  5. The server returns to step 2 and waits for the next connection, if connection not closed by server.

Constructors of ServerSocket Class:

Sr. No.	Constructors	Description
1.	`ServerSocket(int port) throws IOException`	Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2.	`ServerSocket(int port, int backlog) throws IOException`	Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3.	`ServerSocket(int port, int backlog, InetAddress address) throws IOException`	Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.
4.	`ServerSocket() throws IOException`	Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

Methods of ServerSocket Class:

Sr. No.	Methods	Description
1.	`int getLocalPort()`	Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2.	`Socket accept() throws IOException`	Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
3.	`void setSoTimeout(int timeout)`	Sets the time-out value for how long the server socket waits for a client during the accept().
4.	`void bind(SocketAddress host, int backlog)`	Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

Program 4.5: Scanner for the server ports. In following example, if you try to connect server socket on a particular port which is already occupied by other applications then code will throw the exception and you will get the message that "There is a server..."

```
import java.net.*;
import java.io.*;
public class WindowDemo {
    public static void main(String[] args) {
        for (int port = 1; port <= 65535; port++) {
            try {
                ServerSocket server = new ServerSocket(port);
            } catch (IOException ex) {
                System.out.println("There is a server on port " + port + ".");
            }
        }
    }
}
```

Output:

```
There is a server on port 135.
There is a server on port 139.
There is a server on port 443.
There is a server on port 445.
There is a server on port 902.
There is a server on port 912.
```

Program 4.6: TCP/IP communication between client and server. Sending a single message from server to client.

SimpleServer.Java:

```
import java.net.*;
import java.io.*;
public class SimpleServer
{
    public static void main(String args[]) throws IOException
      {
        ServerSocket s = new ServerSocket(1254);      // service on port 1254
        System.out.println("Server Started, waiting for client connection....");
        Socket s1=s.accept();                         // Wait and accept a connection
        OutputStream out = s1.getOutputStream();      //geta stream
        DataOutputStream dos = new DataOutputStream (out);
        dos.writeUTF("Hi there");                     // Send a string!
        // Close the connection, but not the server socket
        dos.close();
        out.close();
        s1.close();
    }
}
```

SimpleClient.Java:

```
import java.net.*;
import java.io.*;
```

```
public class SimpleClient
{
      public static void main(String args[]) throws IOException
      {
         // Open your connection to a server, at port 1254
         Socket s1 = new Socket("localhost",1254);
         // Get an input file handle from the socket and read the input
             InputStream In = s1.getInputStream();
             DataInputStream dis = new DataInputStream(In);
             String st = new String (dis.readUTF());
             System.out.println(st);
             dis.close();
             In.close();
             s1.close();
       }
}
```

Output:

```
Run the server first
Server Started, waiting for client connection....
Run the client
hi there
```

Program 4.7: Program to send number from client and server will reply number as even or odd.

OddEvenServer.Java:

```
import java.net.*;
import java.io.*;
public class OddEvenServer
{
public static void main(String[] args)
{
    try
    {
        ServerSocket ss=new ServerSocket(1000);
        System.out.println("Searching..");
        Socket s=ss.accept();
        BufferedReader br =new BufferedReader(new
        InputStreamReader(s.getInputStream()));
        int n=Integer.parseInt(br.readLine());
        OutputStream o=s.getOutputStream();
        PrintStream ps=new PrintStream(o);
        System.out.println("Checking");
        if(n%2==0)
            ps.println("No is Even");
        else
            ps.println("No is Odd");
    }
         catch(Exception e){}
    }
}
```

OddEvenClient.Java:

```
import java.net.*;
import java.io.*;
public class OddEvenClient {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 1000);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter no:");
            int n = Integer.parseInt(br.readLine());
            OutputStream o = s.getOutputStream();
            PrintStream ps = new PrintStream(o);
            ps.println(n);
            BufferedReader br1 =
                        new BufferedReader(new InputStreamReader(s.getInputStream()));
            int c = br1.read();
            while (c != -1) {
                System.out.print((char) c);
                c = br1.read();
            }
        } catch (Exception e) {
        }
    }
}
```

Output:

```
Run OddEvenServer class
Searching..
Checking
Run OddEvenClient class
Enter no:
56
No is Even
```

Program 4.8: TCP/IP client/server communication, exchange of message between client and server.

TCP_Server.Java:

```
import java.io.*;
import java.net.*;
public class TCP_Server
{
    public static void main(String args[]) throws IOException
    {
        ServerSocket ss = null;
        Socket socket =null;
        String message = null;
        ss = new ServerSocket(8002);
System.out.println("Server socket is created and waiting for client");
DataOutputStream ostream;
    DataInputStream istream;
    socket = ss.accept();
```

```
            ostream = new DataOutputStream(socket.getOutputStream());
            istream = new DataInputStream(socket.getInputStream());
            message = istream.readUTF();
            System.out.println("Client Says: "+message);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            // To read from console
            System.out.println("Enter Server Message:");
            message = br.readLine();
            ostream.writeUTF(message);
            System.out.println("Reply to Client:" +message);
               socket.close();
            ostream.close();
            istream.close();
         }
      }
```

TCP_Client.Java:

```
      import java.io.*;
      import java.net.*;
      public class TCP_Client
      {
         public static void main(String args[]) throws IOException
         {
            Socket cs = null;
            String message = null;
            DataOutputStream ostream;
            DataInputStream istream;
            cs = new Socket(InetAddress.getLocalHost(),8002);
             System.out.println("Client socket is created and waiting for server");
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            // To read from console
            System.out.println("Enter Client Message:");
            message = br.readLine();
            ostream = new DataOutputStream(cs.getOutputStream());
            istream = new DataInputStream(cs.getInputStream());
            ostream.writeUTF(message);
            ostream.flush();
            message = istream.readUTF();
            System.out.println("Client Says: "+message);
            cs.close();
            ostream.close();
            istream.close();
         }
      }
```

Output:

```
      Run the simpleServer Class
      Server socket is created and waiting for client
      Client Says: hello server
      Enter Server Message:
```

```
hello client
Reply to Client:hello client
Run the SimpleClient class
Client socket is created and waiting for server
Enter Client Message:
hello server
Client Says: hello client
```

4.4.3 Whois

- Whois is a simple directory service protocol defined in RFC 954; it was originally designed to keep track of administrators responsible for Internet hosts and domains.
- A whois client connects to one of several central servers and requests directory information for a person or persons; it can usually give you a phone number, an email address, and a U.S. mail address.
- In java whois is a query and response protocol that is widely used for querying databases that store the registered users or assignees of an Internet resource, such as a domain name, an IP address block, or an autonomous system, but is also used for a wider range of other information.
- Let's begin with a simple client to connect to a whois server. The basic structure of the whois protocol is:
  1. The client opens a TCP socket to port 43 on the server whois.internic.net. When you're using whois, you almost always connect to this server.
  2. The client sends a search string terminated by a carriage return/linefeed pair (\r\n). The search string can be a name, a list of names, or a special command.
  3. The server sends an unspecified amount of human-readable information in response to the command and closes the connection.
  4. The client displays this information to the user.
- Following program make use of "Apache Commons Net" library to get the WHOIS data of a domain. The Apache Commons Net 3.6 API does not come bundled with the JDK. You can download it from here: http://commons.apache.org/proper/commons-net/download_net.cgi.
- After downloading the zip file. Extract it and Add the jar file in the libraries of project (Right click Project → properties → libraries → add jar).

Program 4.9: Program for whois.

```
import java.io.IOException;
import java.net.SocketException;
import org.apache.commons.net.whois.*;
public class WhoisDemo {
       public static void main(String[] args) {
       WhoisDemo obj = new WhoisDemo();
       System.out.println(obj.getWhoisInfo("google.com"));
    }
    public String getWhoisInfo(String domainName) {
       StringBuilder result = new StringBuilder("");
       WhoisClient whois = new WhoisClient();
       try {
            //default is internic.net
            whois.connect(WhoisClient.DEFAULT_HOST);
            String s = whois.query("=" + domainName);
            result.append(s);
            whois.disconnect();
    } catch (SocketException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
            }
            return result.toString();
        }
    }
```

Output:

```
Domain Name: GOOGLE.COM
   Registry Domain ID: 2138514_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.markmonitor.com
   Registrar URL: http://www.markmonitor.com
   Updated Date: 2018-02-21T18:36:40Z
   Creation Date: 1997-09-15T04:00:00Z
   Registry Expiry Date: 2020-09-14T04:00:00Z
   Registrar: MarkMonitor Inc.
   Registrar IANA ID: 292
   Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
   Registrar Abuse Contact Phone: +1.2083895740
   Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
   Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
   Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
   Domain Status: serverDeleteProhibited https://icann.org/epp#serverDeleteProhibited
   Domain Status: serverTransferProhibited https://icann.org/epp#serverTransferProhibited
   Domain Status: serverUpdateProhibited https://icann.org/epp#serverUpdateProhibited
   Name Server: NS1.GOOGLE.COM
   Name Server: NS2.GOOGLE.COM
   Name Server: NS3.GOOGLE.COM
   Name Server: NS4.GOOGLE.COM
   DNSSEC: unsigned
   URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2019-04-19T16:03:45Z <<<
```

4.5 URL

- The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet.
- In fact, the Web is really just that same old Internet with all of its resources addressed as URLs plus HTML. Within Java's network class library, the URL class provides a simple, concise API to access information across the Internet using URLs.

4.5.1 Format of URL

- Two examples of URLs are http://www.rediff.com/ and http://www.rediff.com:80/index.htm/.
- A URL specification is based on four components, (See Fig. 4.11).
  1. The first is the protocol to use, separated from the rest of the locator by a colon (:). Common protocols are http, ftp, gopher, and file, although these days almost everything is being done via HTTP.
  2. The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (//) and on the right by a slash (/) or optionally a colon (:).
  3. The third component, the port number, is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/). (It defaults to port 80, the predefined HTTP port; thus:80 is redundant.)

4. The fourth part is the actual file path. Most HTTP servers will append a file named index.html or index.htm to URLs that referdirectly to a directory resource. Thus, http://www.rediff.com/ is the same as http://www.rediff.com/index.htm.

- Java URL Class in java.net package, deals with URL (Uniform Resource Locator) which uniquely identify or locate resource on internet.

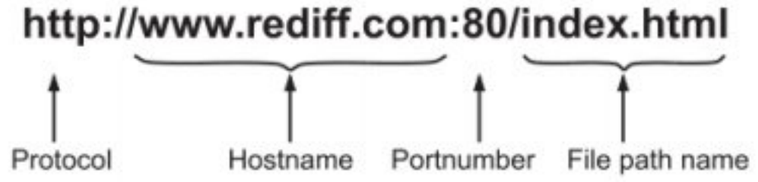


Fig. 4.11: Example of URL

4.5.2 URL Class

- URL is a reference that is an address to a resource on the Internet. The simplest way for a Java program to locate and retrieve data from the network is touse the java.net.URL class.
- The java.net.URL class represents a URL and has complete set of methods to manipulate URL in Java.

Constructors of URL Class:

Sr. No.	Constructor	Description
1.	`URL(String spec)`	Creates a URL object from the String representation.
2.	`URL(String protocol, String host, int port, String file)`	Creates a URL object from the specified protocol, host, port number and file.
3.	`URL(String protocol, String host, int port, String file, URLStreamHandler handler)`	Creates a URL object from the specified protocol, host, port number, file and handler.
4.	`URL(String protocol, String host, String file)`	Creates a URL from the specified protocol name, host name and file name.
5.	`URL(URL context, String spec)`	Creates a URL by parsing the given spec within a specified context.
6.	`URL(URL context, String spec, URLStreamHandler handler)`	Creates a URL by parsing the given spec with the specified handler within a specified context.

Methods of URL Class: Used for retrieving individual components of the represented URL.

Sr. No.	Methods	Description
1.	`void set(String protocol, String host, int port, String file, String ref)`	Sets the fields of the URL. This is not a public method so that only URLStreamHandlers can modify URL fields. URLs are otherwise constant.
2.	`int getPort()`	Gets the port number. Returns –1 if the port is not set.
3.	`String getProtocol()`	Gets the protocol name.
4.	`String getHost()`	Gets the host name.
5.	`String getFile()`	Gets the file name.
6.	`String getPath ()`	Gets the path part of URL.
7.	`String getQuery ()`	Gets the query part of URL.
8.	`boolean equals(Object obj)`	Compares two URLs. Returns true if and only if they are equal, false otherwise.
9.	`URLConnection openConnection() throws IOException`	Creates (if not already in existance) an URLConnection object that contains a connection to the remote object referred to by the URL.
10.	`final InputStream openStream() throws IOException`	Opens an input stream.
11.	`final Object getContent() throws IOException`	Gets the contents from this opened connection.

Program 4.10: Display the parts of a URL.

```
import java.net.*;
class Test
{
   public static void main(String[] arg)
   {
      try
       {
         URL hp = new URL("http://www.yahoo.com:80/index");
         System.out.println(hp.getProtocol());
         System.out.println(hp.getPath());
         System.out.println(hp.getHost());
         System.out.println(hp.getFile());
         System.out.println(hp.getPort());
       }
      catch(MalformedURLException e){}
      }
 }
```

Output:

```
http
/index
www.yahoo.com
/index
80
```

Program 4.11: Determining which protocols you machine supports.

```
import java.net.*;
public class WindowDemo {
    public static void testProtocol(String url) {
        try {
            URL u = new URL(url);
            System.out.println(u.getProtocol() + " is supported");
        } catch (MalformedURLException e) {
        }
    }
    public static void main(String[] arg) {
        testProtocol("http://www.adc.org");
        testProtocol("https://www.xyz.com/");
        testProtocol("ftp://msbte.org/languages/java/javafaq/");
        testProtocol("mailto:thalor.meenakshi@gmail.com");
        testProtocol("telnet://msbte.org/");
        testProtocol("file:///etc/passwd");
    }
}
```

Output:

```
http is supported
https is supported
ftp is supported
mailto is supported
file is supported
```

4.6 URLCONNECTION

- URLConnection is an abstract class that represents an active connection of a resource specified by a URL.
- The URLConnection class has following two different but related purposes.
  1. First, it provides more control over the interaction with a server (especially an HTTP server) than the URL class. With an URLConnection, we can inspect the header sent by the server and respond accordingly. We can set the header fields used in the client request. We can use an URLConnection to download binary files.
  2. An URLConnection lets us send data back to a web server with POST or PUT and use other HTTP request methods.
- The biggest differences between the URL and URLConnection classes are:
  1. URLConnection provides access to the HTTP header.
  2. URLConnection can configure the request parameters sent to the server.
  3. URLConnection can write data to the server as well as read data from theserver.

Constructors of URLConnection Class:

Sr. No.	Constructors	Description
1.	`URLConnection(URL url)`	Constructs a URL connection to the specified URL. A connection to the object referenced by the URL is not created.

Methods of URLConnection Class:

Sr. No.	Methods	Description
1.	`abstract void connect() throws IOException`	Opens a communications link to the resource referenced by this URL.
2.	`URL getURL()`	Returns the value of this URLConnection's URL field.
3.	`int getContentLength()`	Returns the value of the content-length header field.
4.	`String getContentType()`	Returns the value of the content-type header field.
5.	`long getDate()`	Returns the value of the date header field.
6.	`long getLastModified()`	Returns the value of the last-modified header field. The result is the number of milliseconds since January 1, 1970 GMT.
7.	`Object getContent() throws IOException`	Retrieves the contents of this URL connection.
8.	`Object getContent(Class[] classes)throws IOException`	Retrieves the contents of this URL connection.
9.	`Permission getPermission() throws IOException`	Returns a permission object representing the permission necessary to make the connection represented by this object.
10.	`InputStream getInputStream() throws IOException`	Returns the input stream of the URL connection for reading from the resource.
11.	`OutputStream getOutputStream() throws IOException`	Returns the output stream of the URL connection for writing to the resource.

- A program that uses the URLConnection class directly follows this basic sequence of steps:

  Step 1 : Construct a URL object.

  Step 2 : Invoke the URL object's openConnection( ) method to retrieve a URLConnection object for that URL.

Step 3 : Get an input stream and read data.
Step 4 : Get an output stream and write data.
Step 5 : Close the connection.

Program 4.12: Read the content from a URL.

```
import java.net.*;
import java.io.*;
public class Demo
{
    public static void main(String[] arg)
    {
        try
        {
            URL url=new URL("http://www.google.com");
            URLConnection con=url.openConnection();
            InputStream stream=con.getInputStream();
            int i;
            while((i=stream.read())!=-1)
            {
                System.out.print((char)i);
            }
        }
    catch(Exception e){}
    }
}
```

Output:

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN">
.....
</body></html>
```

4.6.1 TCP/IP Server Sockets

- As we all know that Java has a different socket class that must be used for creating server applications.
- The ServerSocket class is used to create servers that listen for either local or remote client programs to connect to them on published ports. ServerSocket class is explained in Section 4.4.2.

4.7 DATAGRAMS (DATAGRAM PACKET, DATAGRAM SERVER AND CLIENT)

- As we are aware that TCP/IP communication provides a serialized, predictable, reliable stream of packet data where clients and servers communicate via a reliable channel, such as a TCP socket.
- During TCP/IP communicate, all data sent over the channel is received in the same order in which it was sent.
- In contrast, UDP communication provides a unreliable stream of packets called as datagram. In UDP communication the clients and servers do not have and do not need a dedicated point-to-point channel hence the delivery and order of datagrams to their destinations is not guaranteed.
- A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.
- Java supports datagram communication through the following classes:
  1. DatagramPacket, and 2. DatagramSocket.

4.7.1 DatagramPackets Class

- Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet.
- That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.
- The **format of datagram packet** is given below:

  `Msg | length | Host | serverPort`
- Java provides a DatagramPacket class to provide facility for connection less transfer of messages from one system to another.

Constructors of DatagramPacket Class:

Sr. No.	Constructors	Description
1	`DatagramPacket(byte[ ] buf, int len)`	Constructs a DatagramPacket for receiving packets of length len.
2.	`DatagramPacket(byte[] buf, int off, int len)`	Constructs a DatagramPacket for receiving packets of length len, specifying an offset of off bytes into the buffer.
3.	`DatagramPacket((byte[] buf, int len, InetAddress addr, int port)`	Constructs a datagram packet for sending packets of length len to the specified port number on the specified host.
4.	`DatagramPacker(byte[] buf, int off, int len, InetAddress addr, int port)`	Constructs a datagram packet for sending packets of length len with offset off to the specified port number on the specified host.
5.	`DatagramPacket(byte[] buf, int off, int len, SocketAddress addr)`	Constructs a datagram packet for sending packets of length len with offset off to the specified port number on the specified host.

Methods of DatagramPacket Class:

Sr. No.	Methods	Description
1.	`InetAddress getAddress()`	Returns the address of the source (for datagrams being received) or destination (for datagrams being sent).
2.	`byte[ ] getData()`	Returns the byte array of data contained in the datagram. Mostly used to retrieve data from the datagram after it has been received.
3.	`int getLength()`	Returns the length of the valid data contained in the byte array that would be returned from the getData( ) method. This may not equal the length of the whole byte array.
4.	`int getPort()`	Returns the port number.
5.	`void setData(byte[ ] data)`	Sets the data to data, the offset to zero, and the length to number of bytes in data
6.	`void setLength(int size)`	Sets the length of the packet to size.

4.7.2 DatagramSocket Class

- In addition to DatagramPacket, java supports DatagramSocket class for sending and receiving the packet.

Constructors of DatagramSocket Class:

Sr. No.	Constructors	Description
1.	`DatagramSocket()`	Constructs a datagram socket and binds it to any available port on the local host.
2.	`DatagramSocket (DatagramSocketImpl impl)`	Creates an unbound datagram socket with the specified DatagramSocketImpl.
3.	`DatagramSocket(int port)`	Constructs a datagram socket and binds it to the specified port on the local host.
4.	`DatagramSocket(int port, InetAddress iaddr)`	Creates a datagram socket, bound to the specified local address.
5.	`DatagramSocket (SocketAddress bindaddr)`	Creates a datagram socket, bound to the specified local socket address.

Methods of DatagramSocket Class:

Sr. No.	Methods	Description
1.	`void close()`	Closes this datagram socket.
2.	`inetAddress getLocalAddress()`	Gets the local address to which the socket is bound.
3.	`int getLocalPort()`	Returns the port number on the local host to which this socket is bound.
4.	`void receive(DatagramPacket p)`	Receives a datagram packet from this socket.
5.	`void send(DatagramPacket p)`	Sends a datagram packet from this socket.

4.7.3 Datagrams Server and Client

- We know that, datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The DatagramPacket and DatagramSocket classes implement system-independent datagram communication using UDP.
- The example where the messages are typed into the window at the server and written across the network to the client side, where they are displayed, explains the concept of datagram server and client.

Program 4.13: UDP communication between client and Server, Exchange of message between Client and Server.

UDPClient.Java:

```
import java.io.*;
import java.net.*;
public class UDPClient
{
 public static void main(String args[]) throws IOException
 {
    String message = null;
    DatagramSocket cs = null;
    cs = new DatagramSocket();
    byte[] receiveData = new byte[512];
    byte[] sendData  = new byte[512];
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    System.out.println(" UDP Client socket is created and waiting for server");
        InetAddress IPAddress = InetAddress.getLocalHost();
    int port = 9000;
    while(true)
    {
System.out.println("Client Says:");
message = br.readLine();
sendData = message.getBytes();
DatagramPacket sendPacket
                  = new DatagramPacket(sendData, sendData.length, IPAddress,port);
```

```
            cs.send(sendPacket);
            DatagramPacket receivePacket =new DatagramPacket(receiveData, receiveData.length);
            cs.receive(receivePacket);
              message = new String(receivePacket.getData(), 0, receivePacket.getLength());
              System.out.println("Server Says: "+message);
            }
          }
      }
```

UDPServer Java:

```
    import java.io.*;
    import java.net.*;
    public class UDPServer
    {
      public static void main(String args[]) throws IOException
      {
        DatagramSocket ss = null;
        ss = new DatagramSocket(9000);
        byte[] receiveData = new byte[512];
        byte[] sendData  = new byte[512];
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println(" UDP Server socket is created and waiting for client");
        while(true)
        {
         DatagramPacket receivePacket =new DatagramPacket(receiveData, receiveData.length);
         ss.receive(receivePacket);
         String message = new String(receivePacket.getData(), 0, receivePacket.getLength());
         System.out.println("Client Says: "+message);
         if(message.equals("end")) break;
         System.out.println("Enter Server Message:");
         message = br.readLine();
         InetAddress IPAddress = receivePacket.getAddress();
         int port = receivePacket.getPort();
         sendData = message.getBytes();
         DatagramPacket sendPacket
                            = new DatagramPacket(sendData, sendData.length, IPAddress,port);
         ss.send(sendPacket);
        }
         ss.close();
         System.out.println("Server Stopped by User program");
      }
    }
```

Output:

```
Run the UDPServer class and see the following output:
UDP Server socket is created and waiting for client
Client Says: hello
Enter Server Message:
Hi
```

```
Run the UDPClient class and see the following output:
UDP Client socket is created and waiting for server
Client Says:
hello
Server Says: hi
Client Says:
```

Program 4.14: Program to check a number is even or odd using UDP connectivity in java.

OddEvenServer.Java:

```
import java.io.*;
import java.net.*;
public class OddEvenServer {
    public static void main(String args[]) {
        try {
            DatagramSocket ds = new DatagramSocket(2000);
            System.out.println("Searching..");
            byte b[] = new byte[1024];
            DatagramPacket dp = new DatagramPacket(b, b.length);
            ds.receive(dp);
            String str = new String(dp.getData(), 0, dp.getLength());
            System.out.println(str);
            int a = Integer.parseInt(str);
            String s = new String();
            System.out.println("Checking");
            if (a % 2 == 0) {
                s = "Number is even";
            } else {
                s = "Number is odd";
            }
            byte b1[] = new byte[1024];
            b1 = s.getBytes();
            DatagramPacket dp1 = new DatagramPacket(b1, b1.length,
                                                    InetAddress.getLocalHost(), 1000);
            ds.send(dp1);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OddEvenClient.Java:

```
import java.io.*;
import java.net.*;
public class OddEvenClient {
    public static void main(String args[]) {
        try {
            DatagramSocket ds = new DatagramSocket(1000);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter a number : ");
            String num = br.readLine();
            byte b[] = new byte[1024];
```

```
            b = num.getBytes();
            DatagramPacket dp = new DatagramPacket(b, b.length,
                                                   InetAddress.getLocalHost(), 2000);
                ds.send(dp);
                byte b1[] = new byte[1024];
                DatagramPacket dp1 = new DatagramPacket(b1, b1.length);
                ds.receive(dp1);
                String str = new String(dp1.getData(), 0, dp1.getLength());
                System.out.println(str);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Output:

```
Run OddEvenServer class
Searching..
11
Checking
Run OddEvenClient class
Enter a number:
11
Number is odd
```

Practice Questions

1. What is networking?
2. What are the advantages of networking?
3. What is protocol? List out at least two protocols.
4. Explain with suitable diagram the HTTP communication between web server and web client.
5. What is client-server model?
6. What is socket? Define it. How it works?
7. Describe client/server networking in detail.
8. What is port?
9. Explain use of Socket class.
10. Explain TCP/IP client sockets with suitable example.
11. Explain TCP/IP server sockets with suitable example.
12. What is the difference between TCP/IP and UDP?
13. Explain the use of ServerSocket class.
14. Explain the use of InetAddress class.
15. Write a program to retrieve hostname using methods in InetAddress class.
16. Write a program that demonstrates TCP/IP based communication between client and server.
17. Write a program that demonstrates UDP based communication between client and server.
18. Write a program to demonstrate use of URL and URLConnection class for communication.
19. What is datagram?
20. Explain datagram packets in detail.
21. Describe datagram server and client with example.

■■■

5...

Interacting with Database

Chapter Outcomes...

- Choose JDBC or ODBC depending on the given application requirement.
- Explain function of the given tier of JDBC architecture for two tier/three tier models.
- Use relevant type of JDBC Driver for the specified environment.
- Elaborate steps with example to establish connectivity with the specified database.

Learning Objectives...

- To learn Basic Database Concepts in Java
- To study JDBC with its Architecture and Types
- To learn Concept of ODBC
- To understand Driver Interfaces and Driver Manager Class

5.0 INTRODUCTION

- Java offers several benefits to the software developer creating a front-end application for a database server. Java is 'Write Once Run Everywhere' language.
- Java interacts with database using a common database application programming interface called as JDBC (Java DataBase Connectivity).
- The specifications required a JDBC driver to be a translator that converted low-level DBMS messages to low-level messages understood by JDBC API and vice-versa. This meant that Java programmer could use high level Java data-objects defined in the JDBC API to write a routine that interacted with the DBMS.
- Java data objects convert the routine into low-level message that conform to the JDBC driver specification and send them to the JDBC driver. The JDBC driver translates the routine into low-level messages that understood and processed by DBMS.
- Open DataBase Connectivity (ODBC) is an open standard Application Programming Interface (API) for accessing a database. By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel and so on.

5.1 INTRODUCTION TO JDBC AND ODBC

- In this section we study basic concepts of JDBC and ODBC.

5.1.1 JDBC

- JDBC stands for Java DataBase Connectivity.
- It refers to several things, depending on context:
  1. It is a specification for using data sources in Java applets and applications.
  2. It is an API for using low-level JDBC drivers.
  3. It is an API for creating the low-level JDBC drivers, which do the actual connecting/transacting with data sources.

- The JDBC defines every aspect of making data-aware Java applications and applets. The low-level JDBC drivers perform the database-specific translation to the high-level JDBC interface.
- This interface is used by the developer so he does not need to worry about the database-specific syntax when connecting to and querying different databases.
- The JDBC is a package, much like other Java packages such as java.awt. The exciting aspect of the JDBC is that the drivers necessary for connection to their respective databases do not require any pre installation on the clients.

5.1.1.1 JDBC API

- JDBC is a database connectivity API that is a part of java enterprise APIs provided by Sun Microsystems. JDBC defines a set of API.
- The JDBC API is contained in two packages named java.sql and javax.sql.
- The java.sql package contains core Java objects of JDBC API.
- There are two distinct layers within the JDBC API; the application layer, which database application developers use and driver layer which the driver's vendors implement.
- The connection between application and driver layers is illustrated in Fig. 5.1.
- There are four main interfaces that every driver layer must implement and one class that bridges the application and driver layers.
- The four interfaces are Driver, Connection, Statement and ResultSet.

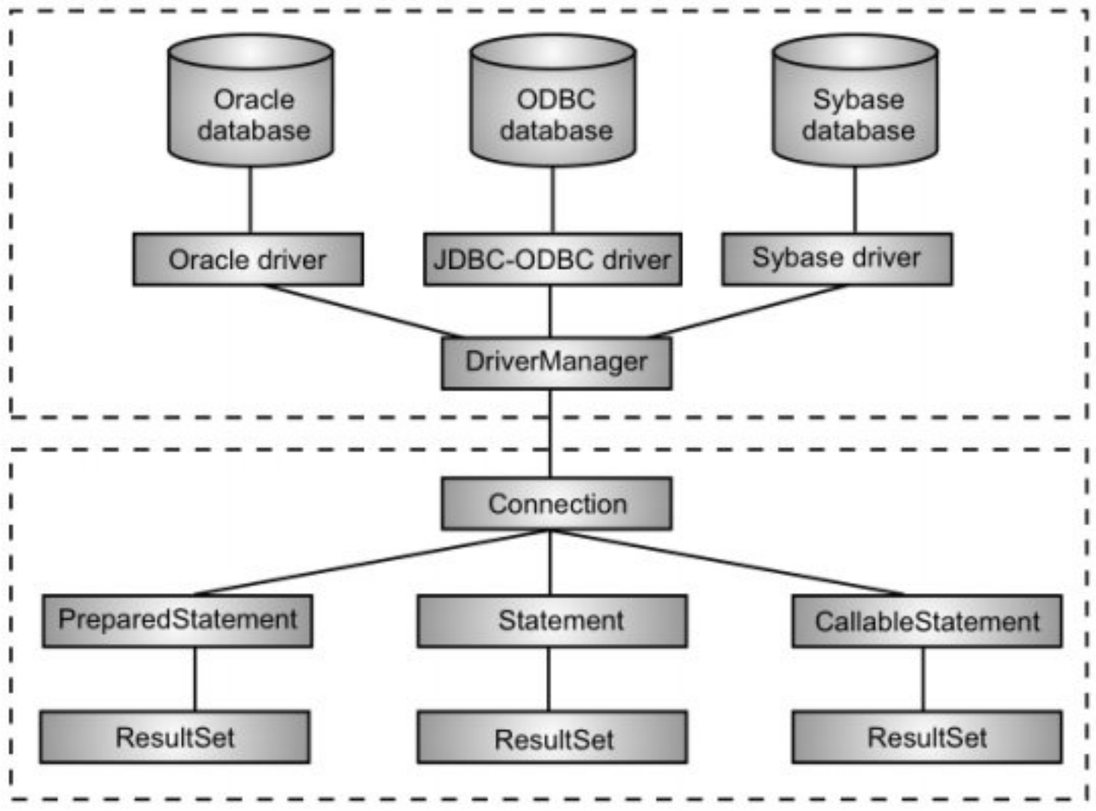


Fig. 5.1: JDBC API

- The Driver interface implementation is where the connection to the database is made. In most applications, Driver is accessed through DriverManager class.
- Let us look objects in of JDBC API in detail:
  1. **Date Object:** The Date object is inherited from the normal Java Date object, but provides methods for accessing the various values within the Data object.
  2. **DriverManager Object:** The DriverManager object provides another way to make a connection to the database. The object is mainly used to manage JDBC Driver objects and can be used to create a connection to the database.
  3. **Time Object:** The Time object is inherited from the Java Date object. It provides various methods for getting and setting the values from the object. It can be used to get Time data values from the database.
  4. **Timestamp Object:** The Timestamp object can be used to get data values from the database that are of the timestamp data type. The object provides various methods for comparing the values of two different Timestamp objects.
  5. **Types Object:** The Types object contains a listing of predefined integer values that identify each of the different data types available for use in JDBC applications.

- The three main **functions of JDBC** are as follows:
  1. Establishing a connection with a database or other tabular data source.
  2. Sending SQL commands to the database.
  3. Processing the results.
- The Java JDBC API enables Java applications to connect to relational databases via a standard API, so your Java applications become independent (almost) of the database the application uses.
- Fig. 5.2 shows Java application using JDBC to connect to a database.

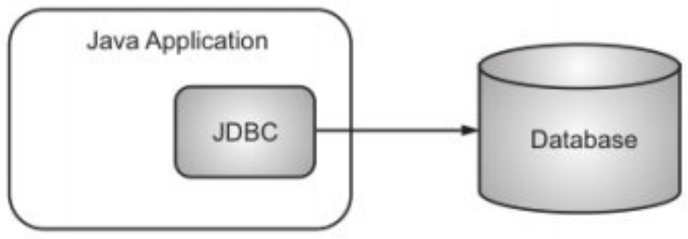


Fig. 5.2: Java Application using JDBC

5.1.2 ODBC

- ODBC (Open Database Connectivity) is a standard programming language middleware API for accessing DataBase Management Systems (DBMS).
- The designers of ODBC aimed to make it independent of database systems and operating systems.
- An application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code.
- ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS.
- The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS.
- Fig. 5.3 shows the ODBC architecture. The ODBC architecture has following components:
  1. The **Application** performs processing and calls ODBC functions to submit SQL statements and retrieve results.
  2. The **Driver Manager** is a DLL provided on a Windows platform as part of the ODBC components.
  3. The **Driver processes** ODBC function calls, submits SQL requests to a specific data source, and returns results to the application.
  4. The **Data source** consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

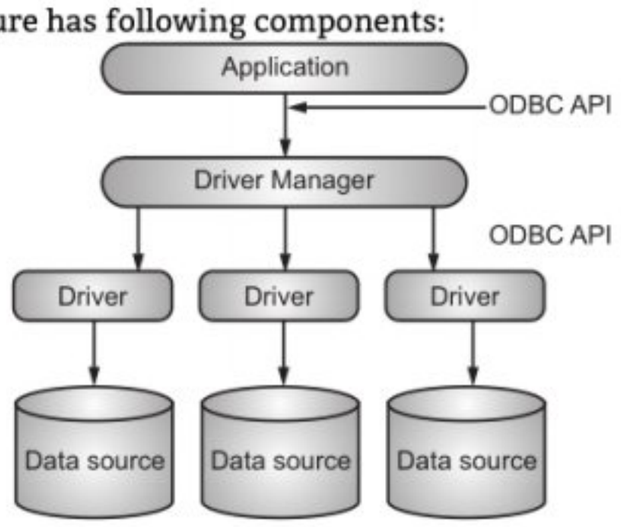


Fig. 5.3: ODBC Architecture

5.1.2.1 ODBC API

- ODBC API uses structured Query Language or SQL as its database language. It provides functions to insert modify and delete data and obtain information regarding the database.
- Fig. 5.4 shows ODBC connection.
- The application could be a GUI program written in java, VC++ or in any other language. The application makes use of ODBC to interact with the database.
- The driver manager is a part of Microsoft ODBC and is used to manage various drivers in the system including loading etc.

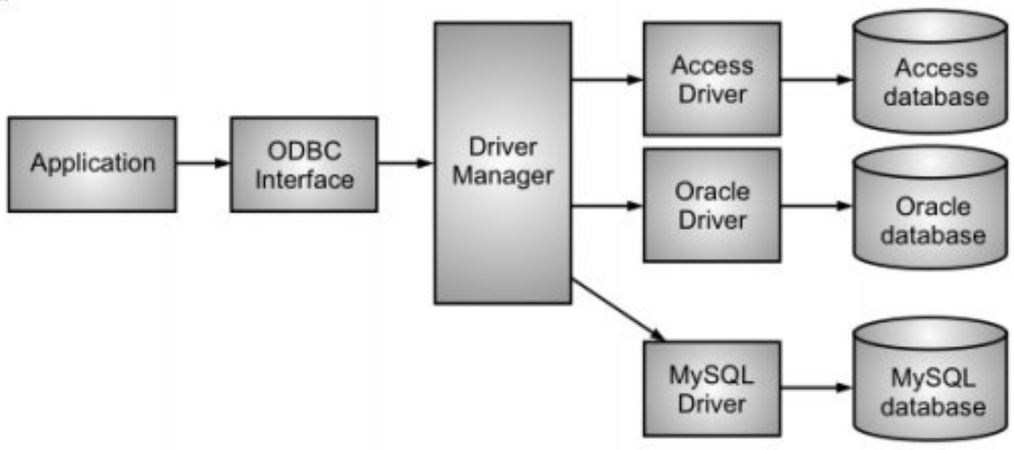


Fig. 5.4: ODBC Connection

- In cases where the application uses multiple databases, it is the function of driver manager to make sure that the function calls gets diverted to the correct database management system.

5.1.3 Difference between JDBC and ODBC

- Following table compares JDBC and ODBC:

Sr. No.	ODBC	JDBC
1.	ODBC Stands for Open Database Connectivity.	JDBC Stands for java database connectivity.
2.	Introduced by Microsoft in 1992.	Introduced by SUN Micro Systems in 1997.
3.	We can use ODBC for any language like C, C++, Java etc.	We can use JDBC only for Java languages.
4.	We can choose ODBC only windows platform.	We can use JDBC in any platform.
5.	Mostly ODBC Driver developed in native languages like C, C++.	JDBC Stands for java database connectivity.
6.	For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform dependent.	For Java application it is highly recommended to use JDBC because there we no performance and platform dependent problem.
7.	ODBC is procedural.	JDBC is object oriented.

5.2 JDBC ARCHITECTURE

- JDBC architecture supports two-tier and three-tier processing models for accessing a database. The detail of these models are explained in subsequent sections.
- The implementation of the actual connection to the data source/ database is left entirely to the JDBC driver. The structure of the JDBC includes following key concepts:
  1. The goal of the JDBC is a DBMS independent interface, a "generic SQL database access framework," and a uniform interface to different data sources.
  2. The programmer writes only one database interface; using JDBC, the program can access any data source without recoding.
- Fig. 5.5 shows architecture of JDBC.

JDBC Components:

- Various JDBC components in Fig. 5.5 are explained below:
  1. **Application:** JDBC API provides the application to manage JDBC connection. The application where the JDBC methods are used to execute the SQL statements and get the results.
  2. **JDBC Driver API:** This supports JDBC manager-to-driver connection.
  3. **Driver Manager:** The main purpose of this component is to load the specific drivers for the user application.
  4. **Driver:** We have explain the different types of drivers in Section 5.3.

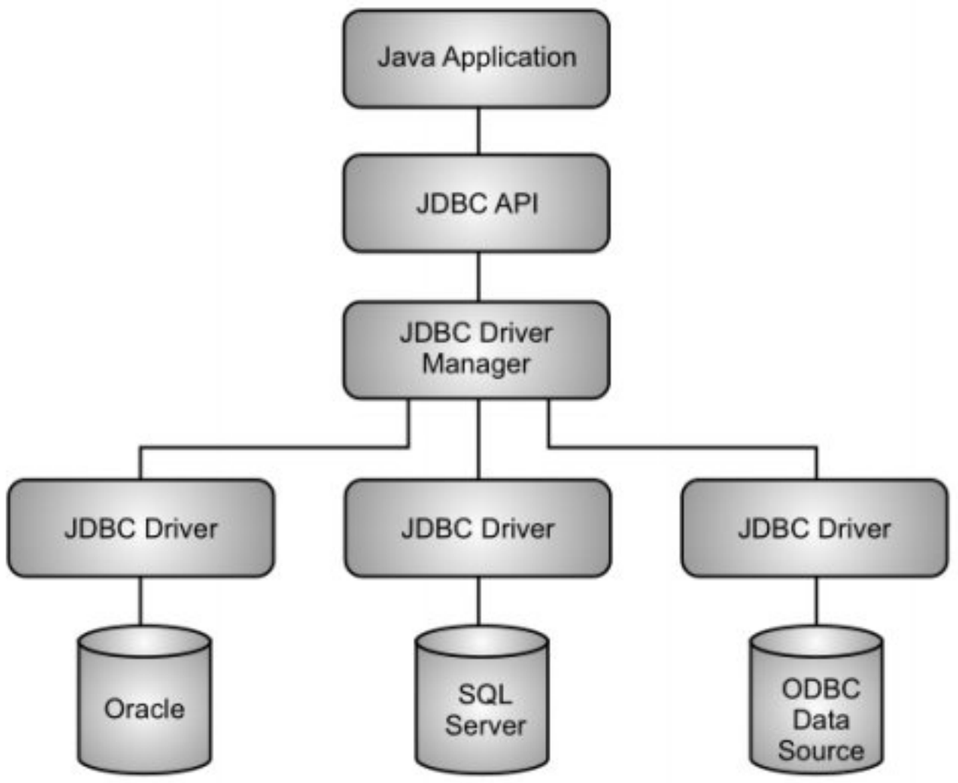


Fig. 5.5: JDBC Architecture

  5. **Data Source:** The data source is what user application interacts with to get the results. This is generally the database.

5.2.1 Two Tier Model

- The first generation of client-server architectures is called two-tiered. It contains two active components i.e., the client which requests data, and the server which delivers data.
- Usually the network binds the back end to the front end, although both tiers could be present on the same hardware.
- For example, hundreds or thousands of airline seat reservation applications can connect to a central DBMS to request, insert, or modify data. While the clients process the graphics and data entry validation, the DBMS does all the data processing.
- Actually, it is inadvisable to overload the database engine with data processing that is irrelevant to the server, thus some processing usually also happens on the clients. Fig. 5.6 shows two tier architecture.
- Preferably, the application's processing should be done separately for database queries and updates, and for user interface presentations.

Advantages:

1. It is simple in design.
2. Client-side scripting offloads work onto the client.

Disadvantages:

1. It is not scalable, because each client requires its own database session.
2. It is inflexible.

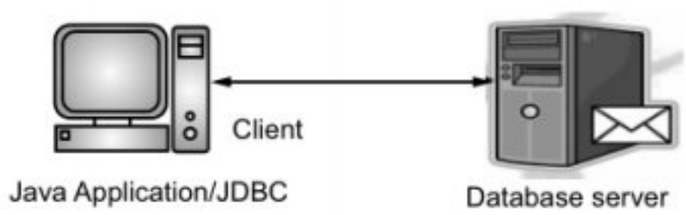


Fig. 5.6: Two Tier Client/Server Architecture

5.2.2 Three Tier Model

- Although the two-tiered architecture is common, another design is starting to appear more frequently. To avoid embedding the application's logic at both the database side and the client side, a third software tier may be inserted.
- In three-tiered architectures, most of the business logic is frozen in the middle tier. In this architecture, when the business activity or business rules change, only the middle-ware must be modified.

Fig. 5.7: Three Tier Client/Server Architecture

- Fig. 5.7 illustrates the three-tier architecture.
- Following the main components of a three-tier architecture:
  1. **Client Tier:** Typically, this is a thin presentation layer that may be implemented using a Web browser.
  2. **Middle Tier:** This tier handles the business or application logic. This may be implemented using a servlet engine such as Tomcat or an application server such as JBOSS. The JDBC driver also resides in this layer.
  3. **Data Source Layer:** This component includes the RDBMS.

Advantages:

1. It is flexible because it can change one part without affecting others.
2. It can connect to different databases without changing code.
3. In this architecture, business logic is clearly separated from the database.
4. Performance can be improved by separating the application server and database server.

Disadvantage:

1. Higher complexity.
2. Higher maintenance.
3. Lower network efficiency.
4. More parts to configure and buy. So time consuming and costly.

5.3 TYPES OF JDBC DRIVERS

- Sun Microsystems has defined four categories of JDBC drivers. The categories delineate the differences in architecture for the drivers. One difference between architectures lies in whether a given driver is implemented in native code or in Java code.
- Native code means whatever machine code is supported by a particular hardware configuration. For example, a driver may be written in C and then compiled to run on a specific hardware platform.
- Fig. 5.8 provides a high-level overview of the various types of drivers.
- Types 1 and 2 rely heavily on additional software, (typically C/C++ DLLs) installed on the client computer to provide database connectivity.
- Types 3 and 4 are pure Java implementations and require no additional software to be installed on the client, except for the JDBC driver.

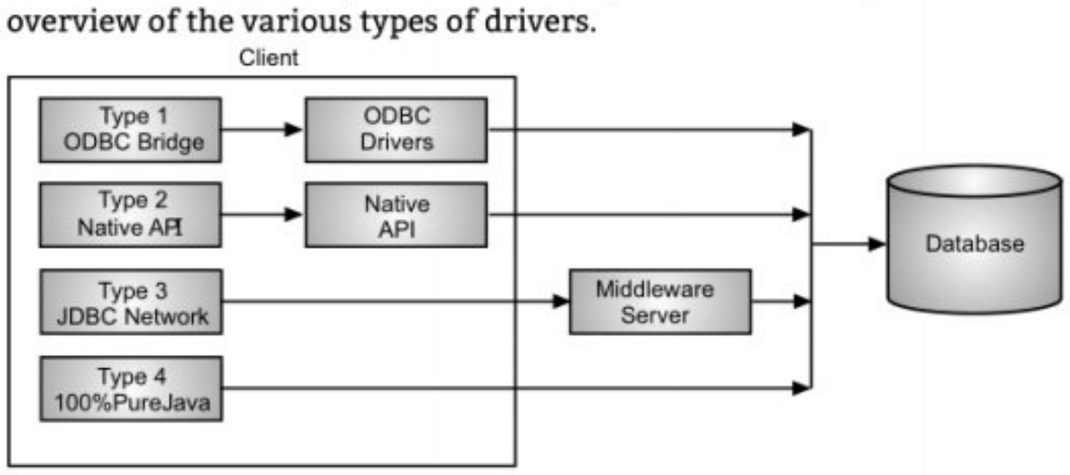


Fig. 5.8: JDBC Driver Types

5.3.1 Type 1 Driver: JDBC/ODBC Bridge

- This type uses bridge technology to connect a Java client to a third-party API such as Open DataBase Connectivity (ODBC).
- Sun's JDBC-ODBC bridge is an example of a Type 1 driver.
- These drivers are implemented using native code. This driver connects Java to a Microsoft ODBC data source.
- Fig. 5.9 shows JDBC Type 1 Driver.
- This driver typically requires the ODBC driver to be installed on the client computer and normally requires configuration of the ODBC data source.
- The bridge driver was introduced primarily to allow Java programmers to build data-driven Java applications before the database vendors had Type 3 and Type 4 drivers.
- Java 8 has removed the JDBC ODBC bridge driver class "sun.jdbc.odbc.jdbcodbcdriver" from JDK and JRE.
- This class is required to connect any database using Object database connectivity driver e.g. Microsoft Access.

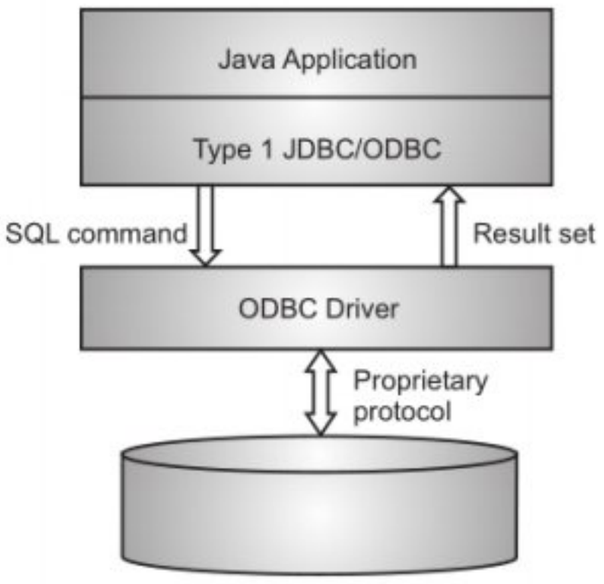


Fig. 5.9: Type 1 Driver

Advantages:

1. Easy to use.
2. Allow easy connectivity to all database supported by the ODBC Driver.
3. It is free there is no any money charge.

Disadvantages:

1. Slow execution time.
2. Dependent on ODBC Driver.
3. Uses Java Native Interface (JNI) to make ODBC call.
4. ODBC drivers must also be loaded on the target machine.
5. Translation between JDBC and ODBC affects performance.
6. Consumes more time and It is not a platform independent.

5.3.2 Type 2 Driver: Native API Driver

- This type of driver wraps a native API with Java classes.
- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver. Because a Type 2 driver is implemented using local native code, it is expected to have better performance than a pure Java driver.
- These drivers enable JDBC programs to use database-specific APIs (normally written in C or C++) that allow client programs to access databases via the Java Native Interface.
- This driver type translates JDBC into database-specific code. Type 2 drivers were introduced for reasons similar to the Type 1 ODBC bridge driver.
- Fig. 5.10 shows Type 2 JDBC Driver.

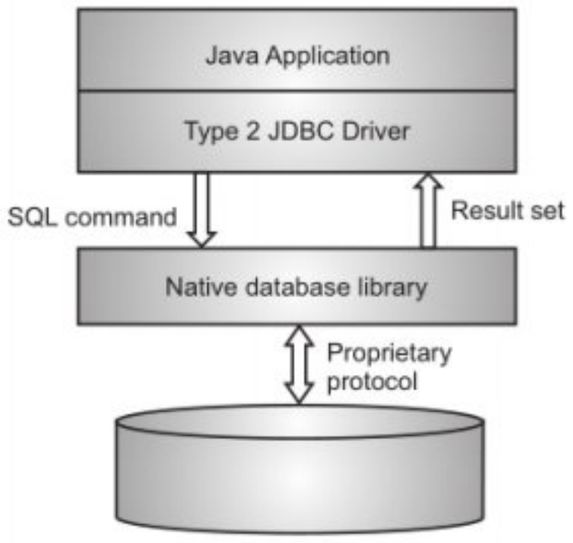


Fig. 5.10: Type 2 Driver

Advantages:

1. Faster as compared to Type-1 driver.
2. Contains additional features.

Disadvantages:

1. Requires native library.
2. Increased cost of application.

5.3.3 Type 3 Driver: Network Protocol, Pure Java Driver

- These drivers take JDBC requests and translate them into a network protocol that is not database specific.
- These requests are sent to a server, which translates the database requests into a database-specific protocol.
- Fig. 5.11 shows Type 3 JDBC Driver.
- This type of driver communicates using a network protocol to a middle tier server. The middle tier in turn communicates to the database.
- Oracle does not provide a Type 3 driver. They do, however, have a program called Connection Manager that, when used in combination with Oracle's Type 4 drivers, acts as a Type 3 driver in many respects.

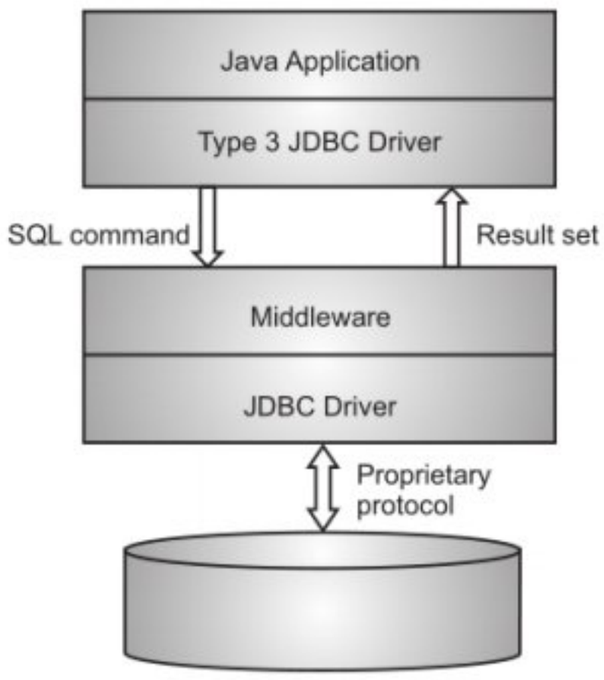


Fig. 5.11: Type 3 Driver

Advantages:

1. It does not require any native library to be installed.
2. It has database independency.
3. It provides facility to switch over from one database to another database.
4. Type 3 drivers do not require any native binary code on the client.
5. They do not need client installation.
6. They support several networking options, such as HTTP tunneling.

Disadvantages:

1. Slow due to increase number of network call.
2. They can be difficult to set up since, the architecture is complicated by the network interface.
3. It is costliest JDBC driver.

5.3.4 Type 4 Driver: Native Protocol, Pure Java Driver

- These convert JDBC requests to database-specific network protocols, so that Java programs can connect directly to a database.
- This type of driver, written entirely in Java, communicates directly with the database. No local native code is required.
- Oracle's thin driver is an example of a Type 4 driver.
- Fig. 5.12 shows Type 4 JDBC Driver.

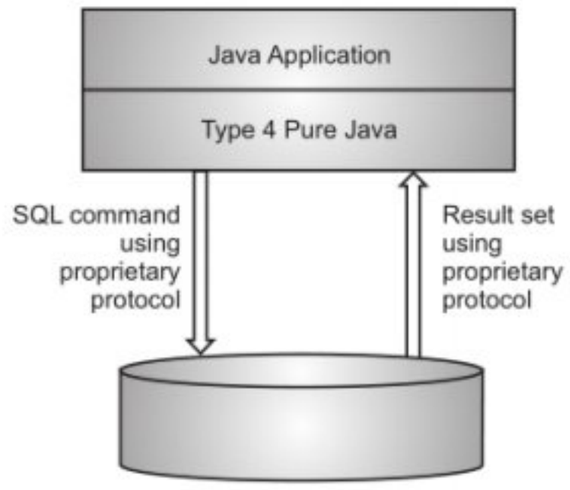


Fig. 5.12: Type 4 Driver

Advantages:

1. They are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
2. Number of translation layers is very less i.e., type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
3. You do not need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantage:

1. With type 4 drivers, the user needs a different driver for each database.

5.4 JAVA INTERFACES AND DRIVER MANAGER CLASS

- Java provides various interfaces for connecting to the database and executing SQL statements.
- Using the JDBC API interfaces, you can execute normal SQL statements, dynamic SQL statements.
- Table 5.1 displays the interfaces provided in the JDBC API.

Table 5.1: Interfaces in JDBC API

Sr. No.	Interfaces	Description
1.	`Connection`	The interface connects your Java application to the database.
2.	`Driver`	An interface provides vendor specific implementations of the abstract classes provided by JDBC API.
3.	`Statement`	An interface for executing normal SQL statements.
4.	`PreparedStatement`	An interface used to execute dynamic SQL statements.
5.	`ResultSet`	An interface that accepts results from a SQL Select statement.

- Let us explain above interfaces in detail:
  1. **Connection Interface:** The Connection interface is the object that provides your Java applications with a connection to the database. This object can be used to create all of the various Statement objects for executing SQL statements.
  2. **Driver Interface:** Driver interface object is a database-specific Driver object provided by the JDBC vendor. It contains specific information about connecting your Java application.
  3. **Statement Interface:** The Statement interface is created from the Connection object and can be used to execute standard SQL statements. The object provides three main methods: execute(), executeQuery() and executeUpdate().
  4. **PreparedStatement Interface:** The PreparedStatement interface enables you to execute dynamic SQL statements.
  5. **ResultSet Interface:** The ResultSet interface is the object that is created and used to get information from SQL Select statements. A SQL Select statement return, a cursor, that is, used by the ResultSet interface to navigate through the result returned by the Select statements.It provides various methods for getting information from the different columns contained in the cursor.

5.4.1 Driver Interface

- A JDBC driver is a software component enabling a Java application to interact with a database.
- To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database.
- The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.
- The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API.
- Through java code we can get connect to any type of database. Following are the widely used drivers:

```
oracle.jdbc.driver.OracleDriver        //for oracle connection
com.mysql.jdbc.Driver                  //for Connection with SQL server
sun.jdbc.odbc.JdbcOdbcDriver;          // for connection with MS-Access
```

- The Driver interface is one of most important interfaces that every driver class must implement. We can load as many database drivers as we want, but all of them must implement the driver interface.
- When a Driver class is loaded, it should create an instance of itself and register it with the DriverManager. This means that a user can load and register a driver by calling,

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- If your application runs on JDK 6 or higher, you do not need to use Class.forName() method as the driver will load automatically when your application asks for its first connection.

5.4.2 DriverManager Class

- The JDBC Driver Manager is a very important class that defines objects which connect Java applications to a JDBC driver.
- Usually Driver Manager is the backbone of the JDBC architecture. The main responsibility of JDBC database driver is to load all the drivers found in the system properly as well as to select the most appropriate driver from opening a connection to a database.
- The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

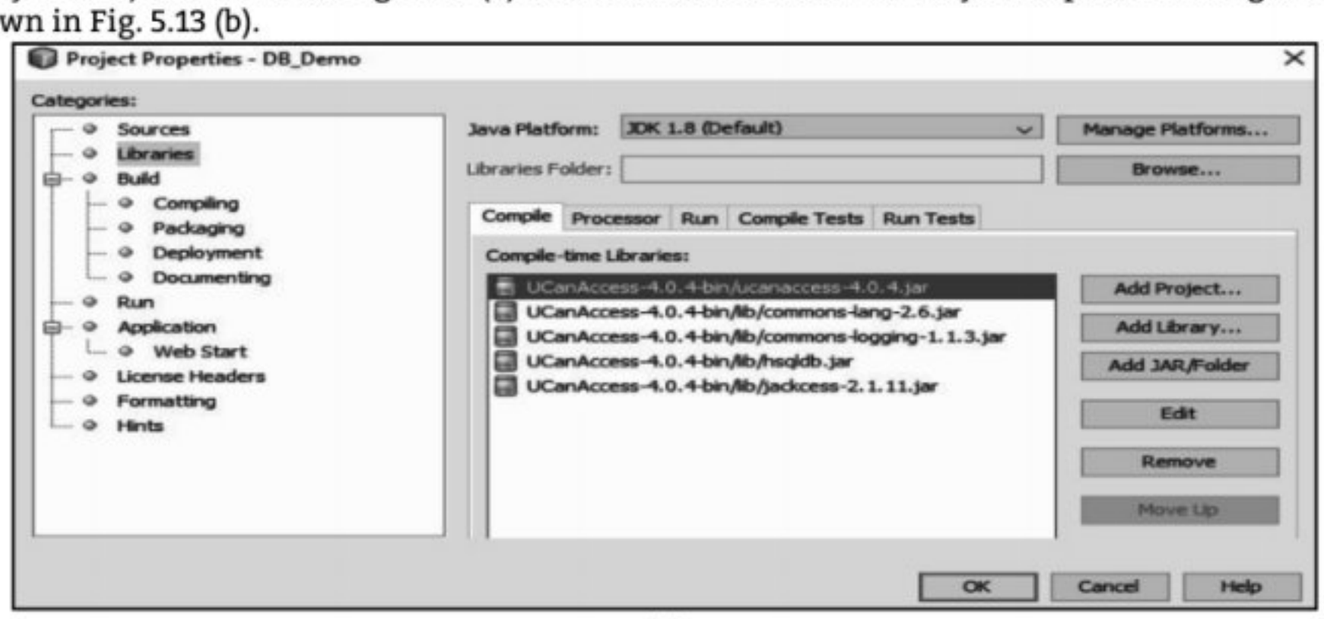
Methods of DriverManager Class:

Sr. No.	Methods	Description
1.	`static void registerDriver(Driver driver)`	It is used to register the given driver with DriverManager.
2.	`static void deregisterDriver(Driver driver)`	It is used to deregister the given driver (drop the driver from the list) with DriverManager.
3.	`static Connection getConnection(String url)`	It is used to establish the connection with the specified url.
4.	`static Connection getConnection (String url,String userName,String password)`	It is used to establish the connection with the specified url, username and password.

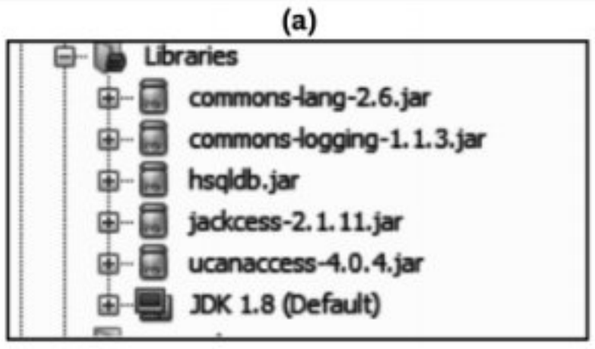
Example:

- Java 8 has removed the JDBC ODBC bridge driver class "sun.jdbc.odbc.jdbcodbcdriver" from JDK and JRE which is required to connect any database using Object database connectivity driver e.g. Microsoft Access.
- In order to access MS-Access Database, in Java 8, download the latest version of UCanAccess through http://ucanaccess.sourceforge.net/site.html website.

- Extract Zip folder and get listed 5 jar files from sub folders: (1) hsqldb.jar, (2) jackcess 2.0.4.jar, (3) commons-lang-2.6.jar, (4) commons-logging-1.1.1.jar, (5) ucanaccess-2.0.8.jar
- Add these jar files in the libraries of your project (right click on Project → Properties → libraries → add jar files) as shown in Fig. 5.13 (a). Confirm the libraries in Project Explorer once get added as shown in Fig. 5.13 (b).



(a)



(b)

Fig. 5.13

- Let Database2.accdb is MS access databse with one Student table which consist of two columns, (ID and Name).
- Following code will check the connection to Database2 established successfully or not. Write following code in DB_Demo.java file.

```
import java.sql.*;
public class DB_Demo
{
    public static void main(String[] args)
        {
            ResultSet rs;
            try
            {
                String url =
                   "jdbc:ucanaccess://C:\\Users\\Meenakshi\\Documents\\Database2.accdb";
                Connection con = DriverManager.getConnection(url);
                System.out.println("connection to database created");
                con.close();
             }
             catch(Exception e)
             {
                 e.printStackTrace();
             }
        }
    }
```

Output:

```
Connection to database created
```

5.4.3 Connection Interface

- A Connection is the session between java application and database.
- The JDBC Connection interface provides methods to handle transaction processing, create objects for executing SQL statements.
- Because the Connection interface is an interface, it cannot directly be created in code; instead, the Connection interface is created by the DriverManager.getConnection() method.
- When the getConnection() method is called, it returns a Connection object that instantiates your Connection object as demonstrated in the following code:

```
Connection con = DriverManager.getConnection(URL);
```

- When you call the getConnection() method of the Driver object, you create a permanent connection to the database.
- The Connection interface provides many methods to handle, information retrieving processing, transaction processing and the creation of SQL statements. It also provides some basic error-handling methods.

Methods of Connection Interface:

Sr. No.	Methods	Description
1.	`Statement createStatement()`	Creates a statement object that can be used to execute SQL queries.
2.	`Statement createStatement (int resultSetType, int resultSetConcurrency)`	Creates a Statement object that will generate ResultSet objects with the given type and concurrency. The Statement object is used to execute only static SQL statements.
3.	`void setAutoCommit(boolean status)`	The setAutoCommit() method enables you to determine whether transaction processing statements such as, commit and rollback will be handled by you or handled automatically by the database. If AutoCommit is turned on (true), then after every SQL statement a commit command will be sent to the database.
4.	`void commit()`	The commit command tells the database to make permanent changes to the database since, the last commit or rollback. If AutoCommit is set to Boolean true, then a commit is implicitly applied after the execution of every SQL statement.
5.	`void rollback()`	Drops all changes made since the previous commit/rollback.
6.	`void close()`	It closes the connection and Releases a JDBC resources immediately.
7.	`PreparedStatement prepareStatement(String sql)`	The PreparedStatement object is used to execute dynamic SQL statements against the database. A dynamic SQL statement is a statement in which some of the parameters(values) in the statement are unknown when the statement is created.

Program 5.1: Program for creation of connection and statement objects.

```
import java.sql.*;
public class DB_Demo
{
   public static void main(String[] args)
   {
     ResultSet rs;
     try
    {
    String url = "jdbc:ucanaccess://C:\\Users\\Meenakshi\\Documents\\
    Database2.accdb";
    Connection con = DriverManager.getConnection(url);
    System.out.println("connection to database created");
    Statement st =con.createStatement();
     System.out.println("Statement object created");
    con.close();
   }
   catch(Exception e)
   {
     e.printStackTrace();
    }
  }
}
```

Output:

```
Connection to database created
Statement object created
```

5.4.4 Statement Interface

- The Connection object just connects you to the database. To execute SQL statements and get results back from the database, you must use the Statement object.
- The Statement object, much like the Connection object but cannot be directly created. Creating a Statement object is a very simple procedure.
- The createStatement() method of Connection interface is used for creation of Statement object . The createStatement() method returns a Statement object as its return value.
- To create a Statement object, you must first import all of the necessary classes that will be used. Then create the connection as,

  ```
  Connection con =DriverManager.getConnection(URL);
  ```
- After creation of the connection object i.e. establishing the connection to the database, you can create the Statement object by calling the createStatement() method of the Connection object.
- The method returns a Statement object as shown below:

  ```
  Statement st =con.createStatement();
  ```
- The Statement interface provides many methods as given in following table:

Sr. No.	Method	Description
1.	`ResultSet executeQuery(String sql)`	The executeQuery() method of the Statement object enables you to send SQL select statements to the database and to receive results from the database. Executing a query, in effect, sends a SQL select statement to the database and returns the appropriate results back in a ResultSet object. The executeQuery() method takes a SQL select statement as a parameter and returns a ResultSet object that contains all of the records that match the select statement's criteria.

contd. ...

2.	`int executeUpdate(String sql)`	The executeUpdate() method of the Statement object enables you to execute SQL update statements such as delete, insert and update. The method takes a String containing the SQL update statement and returns an integer that determines how many records were affected by the SQL statement.
3.	`boolean execute(String sql)`	The execute() method of the Statement object will execute the passed SQL statement against the database using the JDBC driver. The execute() method is used primarily when a SQL statement will return true or false.
4.	`int[] executeBatch()`	Used to execute batch of commands.
5.	`void close()`	The close() method of the Statement object enables you to explicitly close the Statement object.

Program 5.2: Program for insertion of record.

```
import java.sql.*;
public class DB_Demo
{
    public static void main(String[] args)
   {
        ResultSet rs;
        try {
        String url = "jdbc:ucanaccess://C:\\Users\\Meenakshi\\
                                                    Documents\\ Database2.accdb";
        Connection con = DriverManager.getConnection(url);
          System.out.println("connection to database created");
          Statement st = con.createStatement();
          System.out.println("Statement object created");
          st.executeUpdate("insert into Student(Rollno, Name) values(333,'ccc')");
          System.out.println("One record inserted successfully");
          st.close();
          con.close();
       } catch (Exception e) {
          e.printStackTrace();
       }
    }
}
```

Output:

```
connection to database created
Statement object created
One record inserted successfully
```

5.4.5 ResultSet Interface

- A ResultSet provides access to a table of data. The java.sql.ResultSet interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

- A ResultSet object is usually generated by executing a Statement. The ResultSet object is actually a tabular data set; that is, it consists of rows of data organized in uniform columns.
- In JDBC, the Java program can see only one row of data at one time. The program uses the next() method to go to the next row.
- A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.
- The 'next' method moves the cursor to the next row. So in this program we are using while (rs.next()).
- The method rs.next() returns a Boolean value depending on the record set. If it reaches last record, false is returned and loop lapses.
- For maximum portability, ResultSet columns within each row should be read in left-to-right order and each column should be read only once.
- We use getxxx() method of appropriate type to retrieve the value in each column.

Methods of ResultSet Interface:

Sr. No.	Methods	Description
1.	`boolean next()`	It is used to move the cursor to the one row next from the current position.
2.	`boolean previous()`	It is used to move the cursor to the one row previous from the current position.
3.	`boolean first()`	It is used to move the cursor to the first row in result set object.
4.	`boolean last()`	It is used to move the cursor to the last row in result set object.
5.	`boolean absolute(int row)`	It is used to move the cursor to the specified row number in the ResultSet object.
6.	`boolean relative(int row)`	It is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7.	`int getInt(int columnIndex)`	It is used to return the data of specified column index of the current row as int.
8.	`int getInt(String columnName)`	It is used to return the data of specified column name of the current row as int.
9.	`String getString(int columnIndex)`	It is used to return the data of specified column index of the current row as String.
10.	`String getString(String columnName)`	It is used to return the data of specified column name of the current row as String.

Program 5.3: Program for displaying all the records.

```
import java.sql.*;
public class DB_Demo
 {
    public static void main(String[] args)
   {
        ResultSet rs;
        try {
            String url
                 = "jdbc:ucanaccess://C:\\Users\\Meenakshi\\Documents\\Database2.accdb";
Connection con = DriverManager.getConnection(url);
System.out.println("connection to database created");
System.out.println("****Records****");
Statement st1 = con.createStatement();
```

```
            rs = st1.executeQuery("select * from Student");
            while (rs.next()) {
                System.out.println(rs.getInt("Rollno") + "\t\t" + rs.getString("Name"));
            }
            st1.close();
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
connection to database created
****Records****
111    aaa
222    bbb
333    ccc
```

5.4.6 PreparedStatement Interface

- The PreparedStatement interface provides you with a means to execute dynamic SQL statements.
- The PreparedStatement interface provides you with various methods for executing SQL statements as well as setting the dynamic parameters for these SQL.
- The PreparedStatement interface cannot be created directly – it must be instantiated using a method call of another object.
- To create a PreparedStatement object, you can call the prepareStatement() method of the Connection object.
- To create a PreparedStatement object that contains a dynamic SQL statement to get a listing of student based on a specified college, you would use the following **syntax**:

```
String sql = "select * from StudentInfo where college = ?";
PreparedStatement prepare = connection.prepareStatement(sql);
```

- Notice that you specify ? in place of the college name that the SQL statement will be using.
- This indicates to the PreparedStatement interface that you will specify the parameter to use for ? at a later time, using the methods provided by the PreparedStatement interface.

Methods of PreparedStatement Interface:

Sr. No.	Methods	Description
1.	`void setInt(int paramIndex, int value)`	It sets the integer value to the given parameter index. `prepare.setInt(1, 1);` Above code sets the value of first parameter in the SQL statement to the integer value of 1.
2.	`void setString(int paramIndex, String value)`	It sets the String value to the given parameter index. `prepare.setString (1, "Java is hot");` Above code sets the value of first parameter to the string "Java is hot".
3.	`void setFloat(int paramIndex, float value)`	Sets the float value to the given parameter index. `prepare.setFloat (1,10.5);` Above code sets the value of first parameter to the float value of 10.5.
4.	`void setDouble(int paramIndex, double value)`	Sets the double value to the given parameter index. `prepare.setDouble (1, 24.5);` Above code sets the value of first parameter to the double value of 24.5.

contd. ...

5.	`boolean execute()`	The execute() method enables you to execute a SQL statement, usually Create Drop and Alter.
6.	`int executeUpdate()`	It is used to execute the query specially used with insert, update, delete Sql queries . This method will not return a result set, but it will return the number of records that were affected by the executed update statement.
7.	`ResultSet executeQuery()`	It is used to execute the query specially used with Select Sql queries. It returns an instance of ResultSet.

Program 5.4: Program for insertion, deletion, updation and displaying records of table.

```
import java.sql.*;
public class DB_Demo {
    public static void main(String[] args) {
        ResultSet rs;
        try {
            String url =
                   "jdbc:ucanaccess://C:\\Users\\Meenakshi\\Documents\\Database2.accdb";
            Connection con = DriverManager.getConnection(url);
            System.out.println("****Records****");
            Statement st1 = con.createStatement();
            rs = st1.executeQuery("select * from Student");
            while (rs.next()) {
                System.out.println(rs.getInt("Rollno") + "\t\t" + rs.getString("Name"));
            }
PreparedStatement pst1 =
       con.prepareStatement("insert into Student(Rollno,Name) values(?,?)");
pst1.setInt(1,444);
pst1.setString(2,"ddd");
int i = pst1.executeUpdate();
System.out.println(i + " records inserted");
PreparedStatement pst2
            = con.prepareStatement("delete from Student where Rollno =? ");
pst2.setInt(1, 111);
int j = pst2.executeUpdate();
System.out.println(j + " records deleted");

PreparedStatement pst3
      = con.prepareStatement("update  Student  set Name=? where Rollno =?");
pst3.setString(1, "bbbb");
pst3.setInt(2, 222);
int k = pst3.executeUpdate();
System.out.println(k + " records updated");
System.out.println("****Records****");
Statement st2 = con.createStatement();
rs = st2.executeQuery("select * from Student");
while (rs.next()) {
    System.out.println(rs.getInt("Rollno") + "\t\t" + rs.getString("Name"));
}
st1.close();
```

```
            st2.close();
            pst1.close();
            pst2.close();
            pst3.close();
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
****Records****
111    aaa
222    bbb
333    ccc
1 records inserted
1 records deleted
1 records updated
****Records****
333    ccc
444    ddd
222    bbbb
```

5.5 THE ESSENTIAL JDBC PROGRAM

- Upto now we have seen the different operation on database in segment. Now it is time to put all these queries all together in a single program. Following example creates Connection and Statement objects and perform following operations:
  1. Insertion of a record.
  2. Deletion of a record.
  3. Updation of record.
  4. Retrieval of all records.
- Following example considers a MS-Access database (name as Databse2.accdb) with Student table with following columns and data.

Rollno	Name
111	aaa
222	bbb

```
import java.sql.*;
public class DB_Demo {
    public static void main(String[] args) {
        ResultSet rs;
        try {
            String url =
                    "jdbc:ucanaccess://C:\\Users\\Meenakshi\\Documents\\Database2.accdb";
            Connection con = DriverManager.getConnection(url);
            System.out.println("connection to database created");
            Statement stmt1 = con.createStatement();
            System.out.println("Statement object created");
            stmt1.executeUpdate("insert into Student(Rollno, Name) values(333,'ccc')");
```

```
            System.out.println("One record inserted successfully");
            Statement stmt2 = con.createStatement();
            stmt2.executeUpdate("delete from Student where Rollno=111");
            System.out.println("One record deleted successfully");
            Statement stmt3 = con.createStatement();
            stmt3.executeUpdate("update  Student  set Name='xyz' where Rollno=222");
            System.out.println("One record updated successfully");
            System.out.println("****Records****");
            Statement stmt4 = con.createStatement();
            rs = stmt4.executeQuery("select * from Student");
            while (rs.next()) {
                System.out.println(rs.getInt("Rollno") + "\t\t" + rs.getString("Name"));
            }
            stmt1.close();
            stmt2.close();
            stmt3.close();
            stmt4.close();
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
connection to database created
Statement object created
One record inserted successfully
One record deleted successfully
One record updated successfully
****Records****
333    ccc
222    xyz
```

Connection with Java DB (Derby DB):

- In all previous examples, we have used the MS Access database, in addition to MSAccess database we can access other databases like Oracle, SQL and Java DB database.
- JavaDB database is available in NetBeans IDE. Once a connection is made, you can begin working with the database in the Netbeans IDE, allowing you to create tables, populate them with data, run SQL statements and queries, and more.
- The Java DB database is Sun's supported distribution of Apache Derby. Java DB is a fully transactional, secure, standards-based database server, written entirely in Java, and fully supports SQL, JDBC API, and Java EE technology.
- The most popular Relational DBMS's are Oracle, DB2, Microsoft SQL Server, and MySQL Server. We'll use here JavaDB (Derby DB), which is included with your JDK. Java contains classes required for work with DBMS in packages java.sql .
- Follow the following steps to create a Derby Database and use in Java application:

Step 1 : First Add java DB library in java project.
Right click on java Project and select properties .Go to Libraries categories and click on "Add Library.." button and add Java DB driver from list and click ok button as shown in Fig. 5.14.

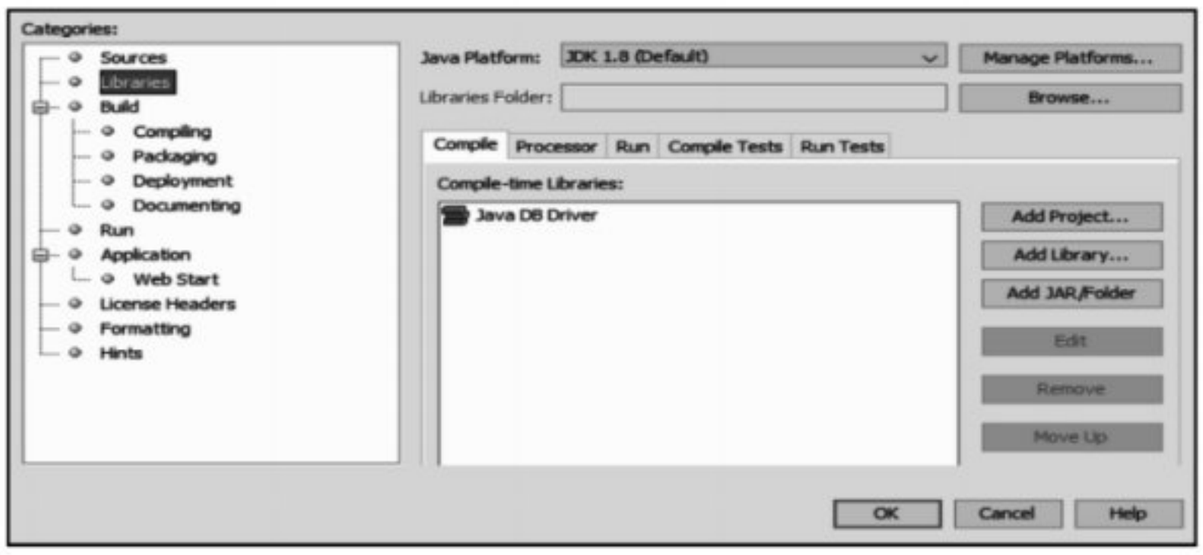
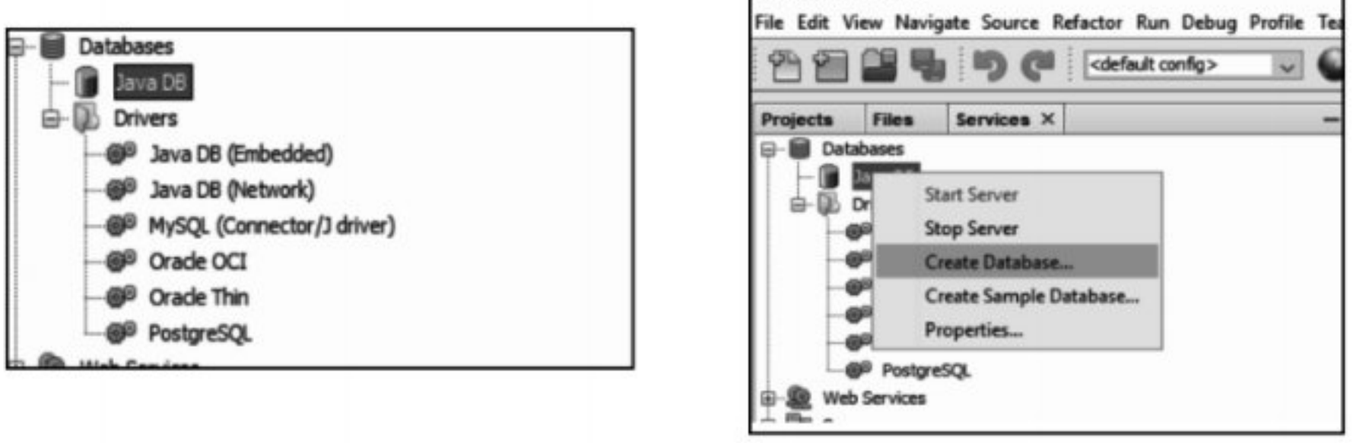


Fig. 5.14

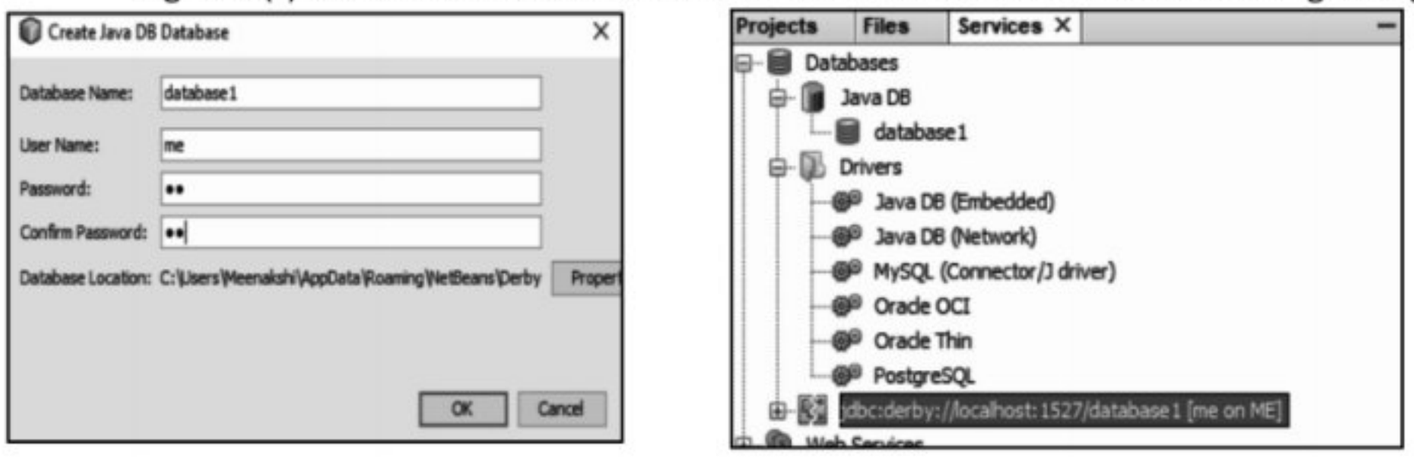
Go to Project explorer and in services tab confirm the addition of java DB in java project (See Fig. 5.15 (a)). Right click on java DB and select "Create Database...", (See Fig. 5.15 (b)).



(a) (b)

Fig. 5.15

Step 2 : Specify the name (database1), username (me) and password (me) for database as shown in Fig. 5.16 (a) and confirm the addition of database1 in services tab as shown in Fig. 5.16 (b).



(a) (b)

Fig. 5.16

Step 3 : Right click on created database1 option and select connect option (See Fig. 5.17 (a)). Once database1 get connected you will see database1 is populated with tables, Views and Procedures. Right click on Tables and select "Create Table.." option as shown in Fig. 5.17 (b).

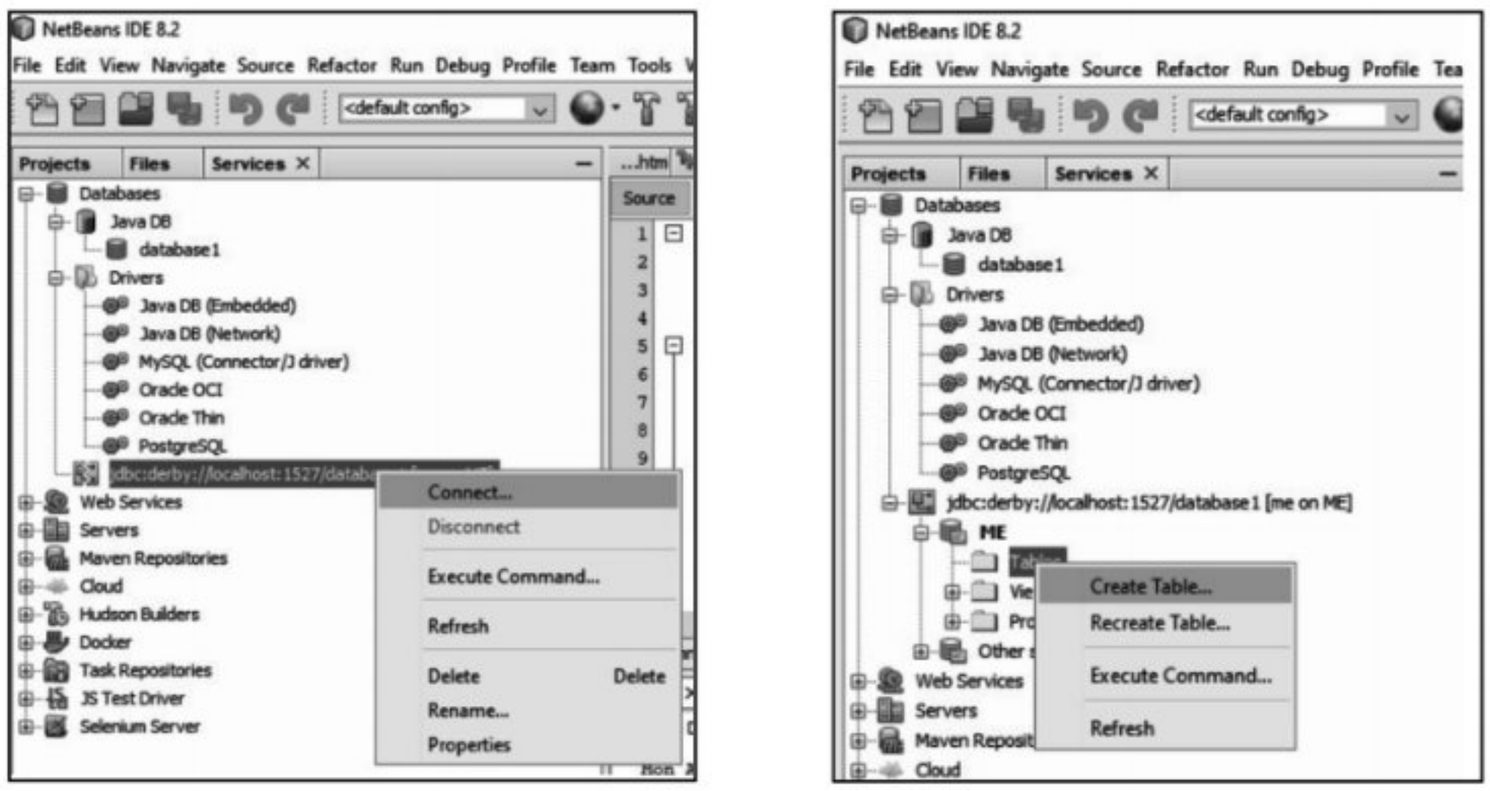


Fig. 5.17

Step 4 : Specify then name of table (Student (See Fig. 5.18 (a)) and add columns (ID and Name) as shown in Fig. 5.18 (b).

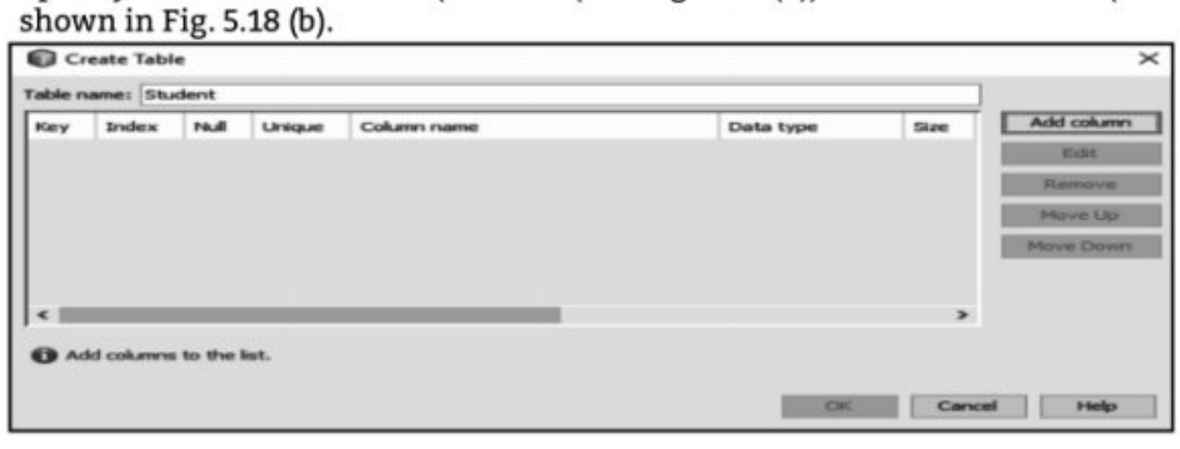


Fig. 5.18

Step 5 : Once the Student table is created inside the tables, Right Click on "Student" table and select "View Data" to add the data in table.

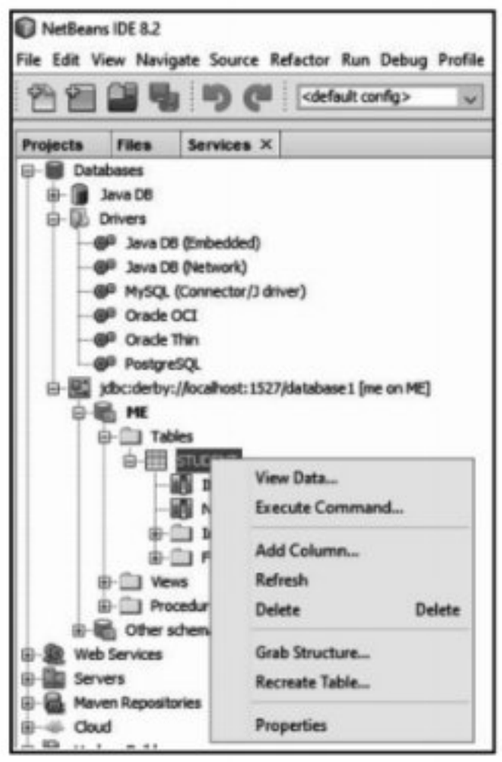


Fig. 5.19

- To add the data in table select the Insert Record icon on top left side (See Fig. 5.20) and add data of interest as shown Fig. 5.21.

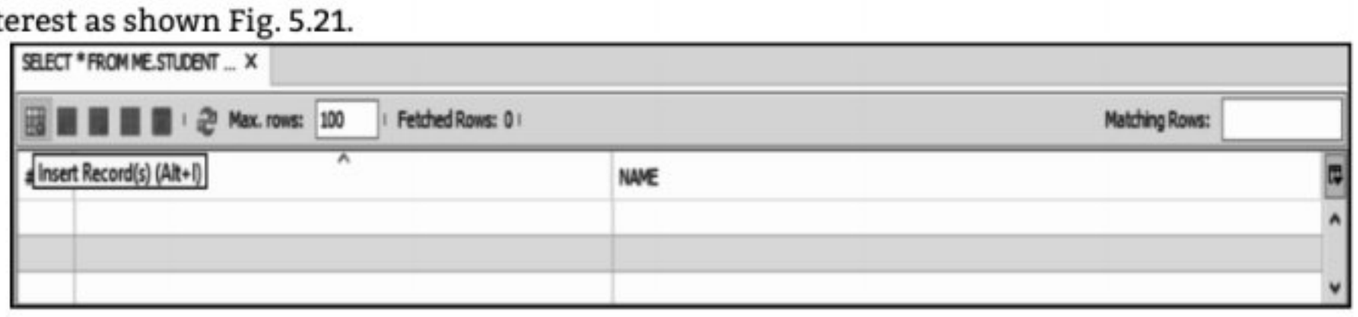


Fig. 5.20

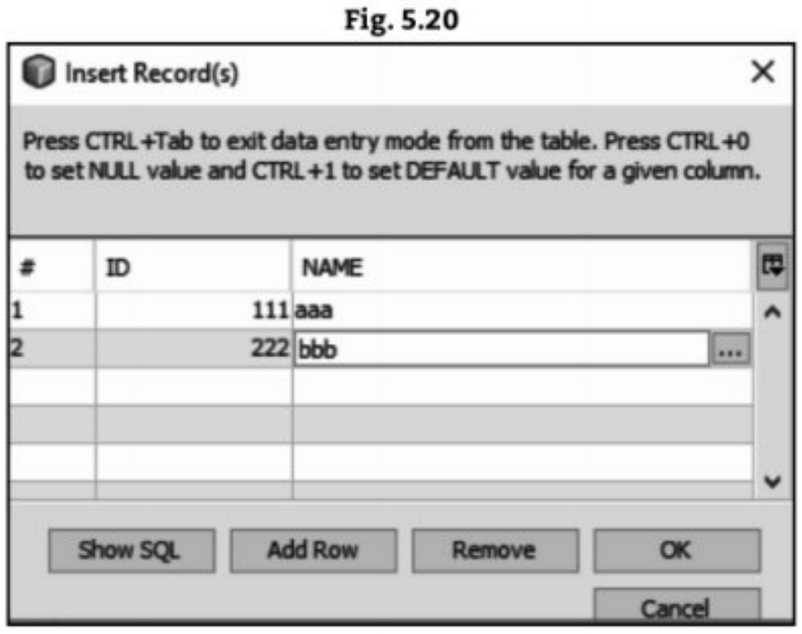
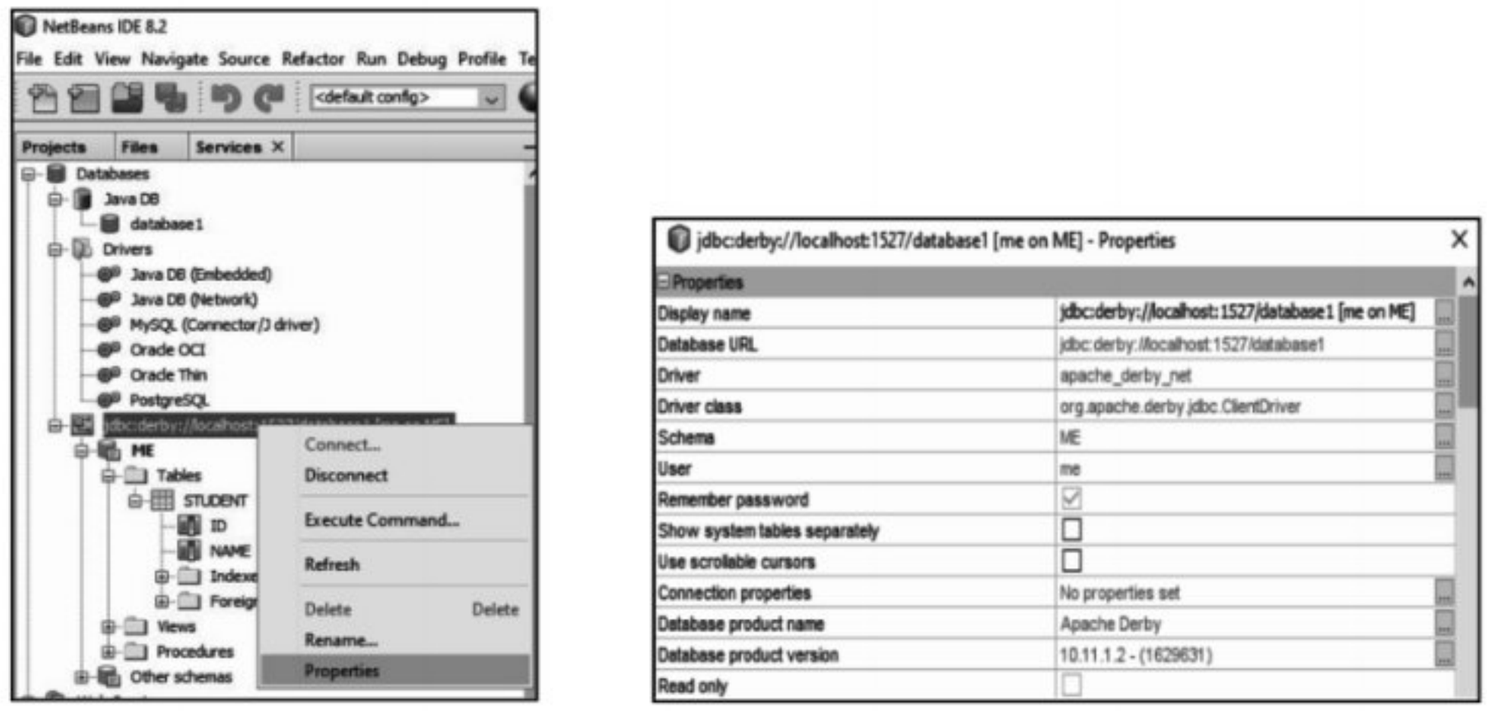


Fig. 5.21

- Once the database1 is ready with Student table and its data,Right click on it and get the properties of database1 as showin in Fig. 5.22. Note the following:
  - Database URL (jdbc:derby://localhost:1527/database1)
  - Driver class (org.apache.derby.jdbc.ClientDriver) to make use in code.



(a) (b)

Fig. 5.22

Progam 5.5: Program to retrieve the records of Student table from database1.

```
import java.sql.*;
public class ConnectionTest
{
    public static void main(String[] args)
    {
        ResultSet rs;
        try
        {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            Connection con=DriverManager.getConnection("jdbc:derby://localhost:1527/
                                                    database1;user=me;password=me");
            System.out.println("Connection established successfully");
            Statement stmt1 = con.createStatement();
            rs = stmt1.executeQuery("select * from Student");
           while(rs.next())
           {
               System.out.println(rs.getInt("ID") + "\t\t" + rs.getString("NAME"));
           }
             stmt1.close();
             con.close();
            }
            catch(Exception e)
            {
                e.printStackTrace();
             }
    }
}
```

Output:

```
Connection established successfully
111    aaa
222    bbb
```

Program 5.6: Program to Insert a record in Student table in database1.

```
import java.sql.*;
public class ConnectionTest
{
    public static void main(String[] args)
        {
            ResultSet rs;
            try
            {
                Class.forName("org.apache.derby.jdbc.ClientDriver");
                Connection con=DriverManager.getConnection("jdbc:derby://localhost:1527
                                                        /database1;user=me;password=me");
                System.out.println("Connection established successfully");
                Statement stmt1 = con.createStatement();
                stmt1.executeUpdate("insert into Student values(333,'ccc')");
                System.out.println("One record inserted successfully");
                stmt1.close();
                con.close();
                }
                catch(Exception e)
                    {
                    e.printStackTrace();
                    }
        }
}
```

Output:

```
Connection established successfully
One record inserted successfully
```

Program 5.7: Program to delete a record from Student table in database1.

```
import java.sql.*;
public class ConnectionTest
{
    public static void main(String[] args)
        {
            ResultSet rs;
            try
            {
                Class.forName("org.apache.derby.jdbc.ClientDriver");
                Connection con=DriverManager.getConnection("jdbc:derby:
                                        //localhost:1527/database1;user=me;password=me");
                System.out.println("Connection established successfully");
               Statement stmt1 = con.createStatement();
               stmt1.executeUpdate("delete from Student  where ID=111");
               System.out.println("One record deleted successfully");
stmt1.close();
 con.close();
}
```

```
                catch(Exception e)
                    {
                    e.printStackTrace();
                    }
            }
    }
```

Output:

```
Connection established successfully
One record deleted successfully
```

Practice Questions

1. What is JDBC?
2. Explain two and three tier architectures diagrammatically.
3. With the help of diagram describe architecture of JDBC.
4. What is the difference between a connection and statement?
5. Differentiate between Statement and PrepareStatement interface.
6. Create an application that opens two connections to the database.
7. Write small program that opens a connection to database.
8. Explain various types of JDBC drivers.
9. Explain some functions of Statement Interface.
10. Describe ResultSet interface in detail.
11. What is ODBC? Explain ODBC API in detail.
12. Describe essential JDBC program in detail.
13. Compare JDBC and ODBC.
14. Explain Driver Manager class in detail.

■■■

6...

Servlets

Chapter Outcomes...

- Explain function of the given method of Servlet life cycle.
- Use relevant Generic Servlet to develop given web based application.
- Use relevant HTTP Servlet to develop specified web based application.
- Develop Servlet for cookies and session tracking to implement the given problem.

Learning Objectives...

- To learn Basic Concepts of Servlets
- To understand Servlet API, javax.servlet Package and javax.servlet.http Package
- To study various Servlet Interfaces
- To understand GenericServlet Class, HttpServlet Class and HttpSessionEvent Class

6.0 INTRODUCTION

- Since, the emergence of the Internet, web technologies have become more and more important and web applications become more and more common.
- In the earlier days web pages were static i.e. a user request a resource and the server returns it.
- However, with the growth of commercial activities on the web, companies wanted to deliver dynamic content to their customers such as Performing bank transactions online, Airline or Railway ticket booking, News, Marketing and so on. To deliver dynamic contents servlet technology came into existence.
- Java Servlets are programs that run on a web or application server and act as a middle layer between a request coming from a Web browser and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Web applications are helper application that resides at web server and build dynamic web pages.

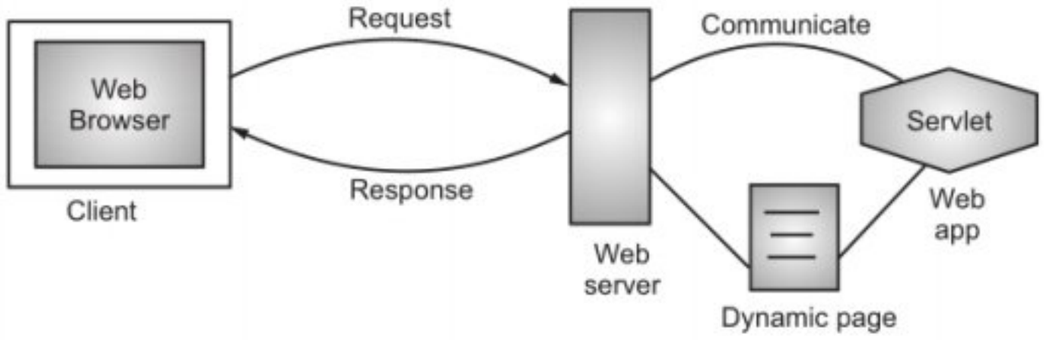


Fig. 6.1: Servlet Technology

- A dynamic page could be anything like a page that randomly chooses picture to display or even a page that display current time.
- As Servlet technology uses Java hence web application made using Servlet are secured, scalable and robust.

6.1 SERVLET

- Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.
- Servlets provide a component-based, platform-independent method for building web-based applications, without the performance limitations of CGI programs. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

6.1.1 Java Servlet Technology

- Java Servlet technology is used to create web application (resides at server side and generates dynamic web page).
- A java servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- A servlet:
  - is a program written using java that runs in a server to answer clients requests.
  - is supported by virtually all web servers and application servers.
  - solves the performance problem by executing all requests as thread in one process.
  - is a technology i.e. used to create web application.
- When a user issues a request for a URL that corresponds to a java servlets, the server hands the request off to the servlet (program on server side) for processing. The servlet dynamically produces a response to the request, typically an HTML web page and sends it back to the requesting web browser as shown in Fig. 6.2.

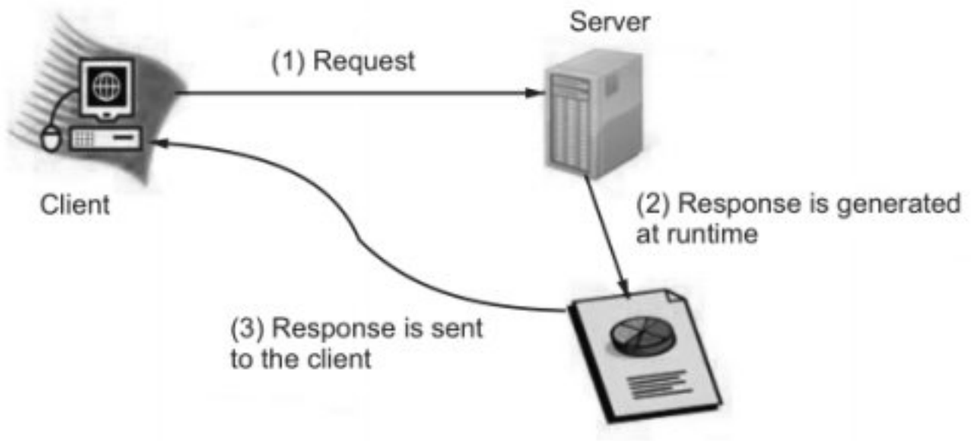


Fig. 6.2: A Java Servlet

- Web server uses servlets ability to access a database table, extract the data and convert this data into a format (HTML) acceptable to a web server for delivery to a client browser through the Internet.
- There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.
- Servlet is a web component that is deployed on the server to create dynamic web page.
- Fig. 6.3 shows the position of Servlets in a web application.

Tasks of Servlets:

1. Read the Explicit data sent by the clients (browsers). This includes an HTML form on a Web page or a custom HTTP client program.
2. Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
3. Process the data and generate the results. This process may require talking to a database and computing the response directly.

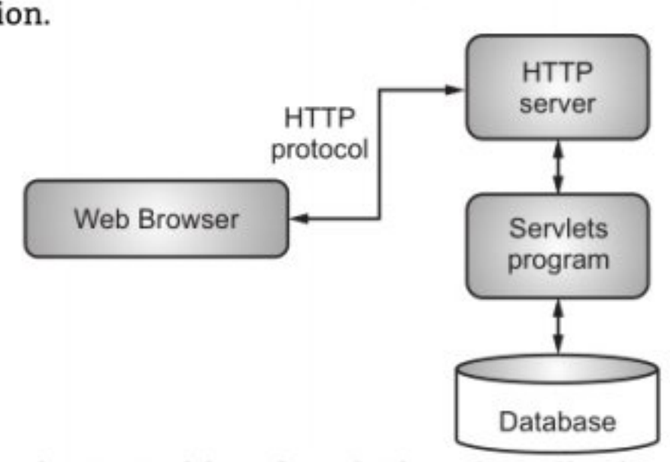


Fig. 6.3: Position of Servlet in Web Application

4. Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
5. Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

6.1.2 Servlet vs. CGI

- CGI (Common Gateway Interface) was the very first attempt to provide dynamic contents to users.
- It allows users to execute a program that resides in the server to process data and even access databases in order to produce the relevant content as shown in Fig. 6.4.
- Since, these are programs, they are written in the native operating system and then stored in a specific directory.

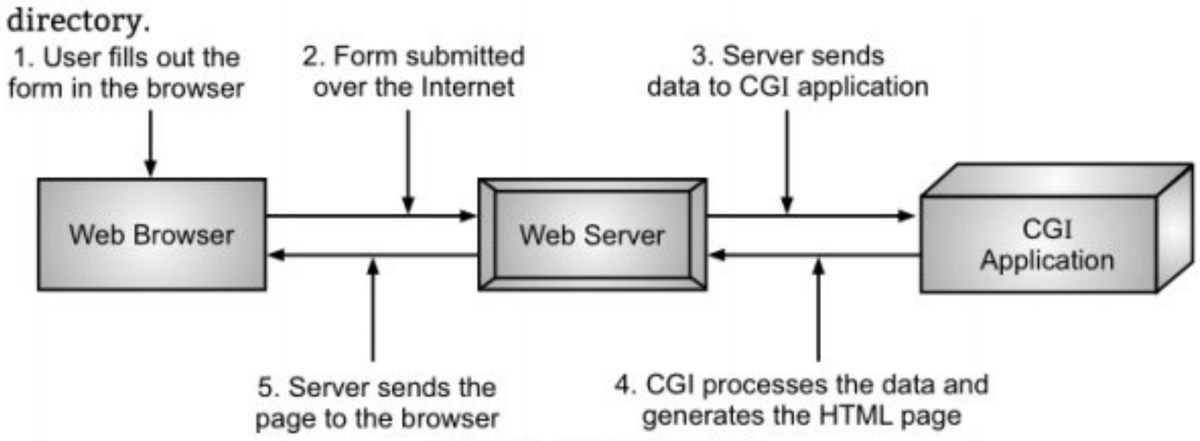


Fig. 6.4: CGI Processing

- A servlet is an implementation of Java that aims to provide the same service as CGI does, but instead of programs compiled in the native operating system, it compiles into the Java bytecode which is then run in the Java virtual machine as shown in Fig. 6.5.

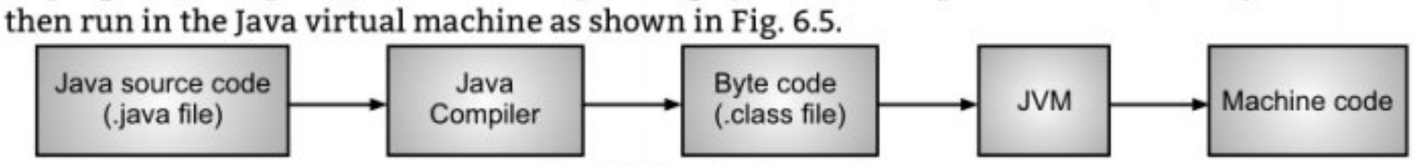


Fig. 6.5: Servlet Processing

- Following table demonstrate the differences between CGI and Servlet:

Sr. No.	CGI	Servlet
1.	CGI creates a new process for each request.	Servlet creates a thread for each request and services the request in that thread.
2.	For each process created by CGI the process is assigned separate address space.	No separate address space is created for every thread created by servlet. All threads operate in the same parent process address space
3.	CGI cannot directly link to Web server.	Servlets can link directly to the Web server.
4.	CGI does not provide sharing property.	Servlets can share data among each other.
5.	CGI cannot perform session tracking and caching of previous computations.	Servlets can perform session tracking and caching of previous computations.
6.	CGI programs are platform dependent and not portable	Servlets are platform independent and portable
7.	In CGI, each request is handled by a heavyweight operating system process.	In Servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread.
8.	CGI cannot automatically parse and decode the HTML form data.	Servlets automatically parse and decode the HTML form data.
9.	CGI cannot read and set HTTP headers, handle cookies, tracking sessions.	Servlets can read and set HTTP headers, handle cookies, tracking sessions.
10.	CGI is more expensive than Servlets	Servlets is inexpensive than CGI.

6.1.3 Advantages of Servlets

- Servlet offers the following benefits (advantages) that are not necessarily available in other technologies:
  1. **Performance:** Each request is handled by the servlet container process. After a servlet is finished processing a request, it stays resident in memory, waiting for another request.

2. **Portability:** Similar to other Java technologies, servlet applications are portable.
3. **Rapid Development Cycle:** As a Java technology, servlets have access to the rich Java library, which helps speed up the development process.
4. **Robustness:** Servlets are managed by the Java Virtual Machine (JVM). As such, you do not need to worry about memory leak or garbage collection, which helps you write robust applications.
5. **Widespread Acceptance:** Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies.
6. **Secure:** Servlet technology is very secured because of it uses java language.

6.1.4 Servlet Application Architecture

- A servlet is a Java class that can be loaded dynamically, (when the request will come from client side) and run by a special web server.
- This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology.
- Servlets interact with clients via a request-response model based on HTTP.

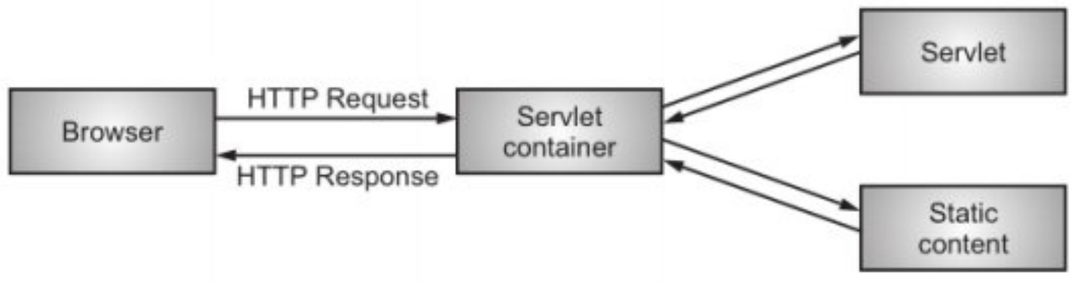


Fig. 6.6: Servlet Application Architecture

- Fig. 6.6 provides the architecture of a servlet application. In Fig. 6.6, a servlet application also can include static content, such as HTML pages and image files.
- Allowing the servlet container to serve static content is not preferable because the content is faster if served by a more robust HTTP server, such as the Apache web server or Microsoft Internet Information Server.
- As such, it is common practice to put a web server at the front to handle all client requests. The web server serves static content and passes to the servlet containers all client requests for servlets.

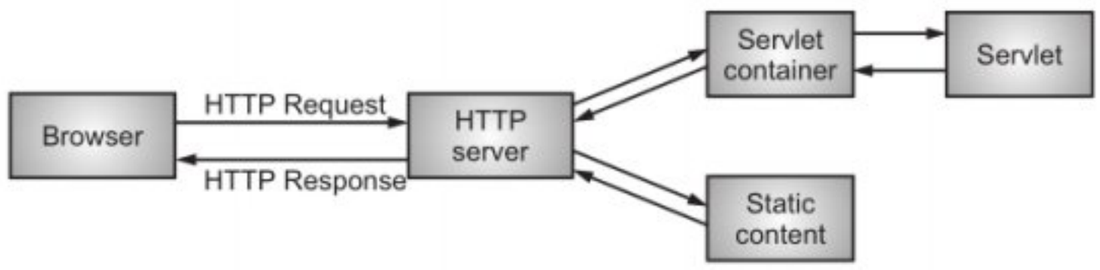


Fig. 6.7: Servlet Application Architecture Employing an HTTP Server

- Fig. 6.7 shows a more common architecture for a servlet application.

6.1.5 Types of Servlets

- There are two types of servlets, GenericServlet and HttpServlet. GenericServlet defines the generic or protocol independent servlet. HttpServlet is subclass of GenericServlet and provides some HTTP specific functionality like doGet and doPost methods.

6.1.5.1 Generic Servlets

- It extend javax.servlet.GenericServlet.
- For example,

```
public class NewServlet1 extends GenericServlet
```

- Generic servlets are protocol independent. They contain no inherent HTTP support or any other transport protocol.

6.1.5.2 HTTP Servlets

- It extends javax.servlet.HttpServlet.
- For example,

```
public class HelloWorldServlet extends HttpServlet
```

- They have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.
- For both servlet types, you implement the constructor method init() and the destructor method destroy() to initialize or deallocate resources.
- All servlets must implement a service() method, which is responsible for handling servlet requests. For generic servlets, simply override the service method to provide routines for handling requests.
- HTTP servlets provide a service method that automatically routes the request to another method in the servlet based on which HTTP transfer method is used.
- For HTTP servlets, override doPost() to process POST requests, doGet() to process GET requests.

6.2 LIFE CYCLE OF A SERVLET

- A java program is converted to servlet by implementing servlet interface. This interface is the source of all activities in servlet programming.
- Every servlet (java program) you write must implement this javax.servlet.Servlet interface, either directly or indirectly.
- A servlet life cycle can be defined as "the entire process from its creation till the destruction". The following are the paths followed by a servlet,
  1. The servlet is initialized by calling the init() method.
  2. The servlet calls service() method to process a client's request.
  3. The servlet is terminated by calling the destroy() method.
  4. Finally, servlet is garbage collected by the garbage collector of the JVM.
- Fig. 6.8 depicts a typical servlet life-cycle scenario:
  1. First the HTTP requests coming to the server are delegated to the servlet container.
  2. The servlet container loads the servlet before invoking the service() method.
  3. Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

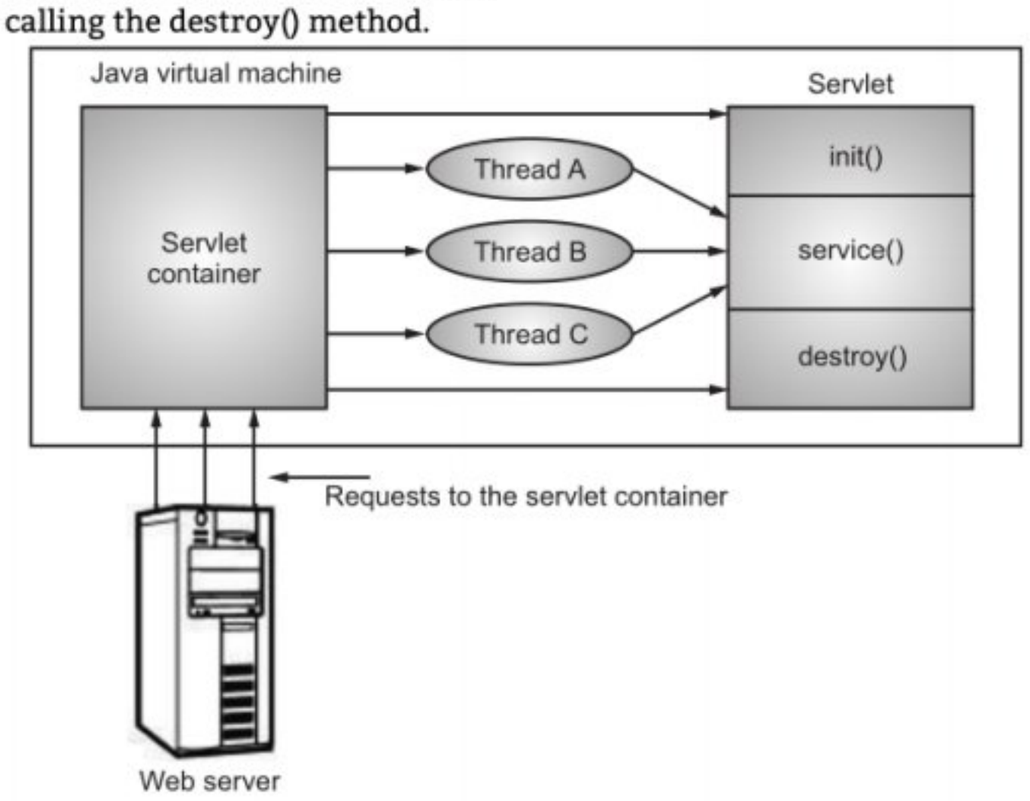


Fig. 6.8: Servlet Procesing

- The life cycle of a servlet is determined by three of its methods i.e., init, service and destroy, (See Fig. 6.8).

1. init() Method:

- The init method is called by the servlet container after the servlet class has been instantiated.
- The servlet container calls this method exactly once to indicate to the servlet that the servlet is being placed into service.
- The **signature/syntax** of this method is as follows:

```
public void init(ServletConfig config) throws ServletException
```

- The init() method is important because the servlet container passes a ServletConfig object, which contains the configuration values stated in the web.xml file for this application.

2. service() Method:

- The service() method is called by the servlet container after the servlet's init method to allow the servlet to respond to a request.
- This method has the following **signature/syntax**:

```
public void service(ServletRequest request,ServletResponse response)
   throws ServletException, IOException
{ // Servlet code }
```

- The server creates ServletRequest, ServletResponse interface objects. Where ServletRequest interface object is used for sending client information to the server and ServletResponse interface object is used for sending response to the client.
- For HttpServlet, service() method dispatches doGet(), doPost() to handle HTTP GET, POST, request respectively.
- The **syntax** of service() method of an HttpServlet is as follows:

```
public void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{ // Servlet code }
```

- The service() method is called by the container which will invokes doGet, doPost, doPut, doDelete, etc. methods depending on what type of request. So you have to override either doGet() or doPost() etc. methods.

(i) doGet() Method:

- When user submit some form data and that form is using GET method then this type of request is handled by doGet() method or when a HTML form has no METHOD specified then also it is handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException { // Servlet code }
```

(ii) doPost() Method:

- When user submit some form data and that form is using POST method then this type of request is handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException { // Servlet code }
```

3. destroy() Method:

- The servlet container calls the destroy method before removing a servlet instance from service. This normally happens when the servlet container is shut down or the servlet container needs some free memory.
- This method is called only after all threads within the servlet's service method have exited or after a timeout period has passed.
- The **signature/syntax** of this method is as follows:

```
public void destroy()
```

6.3 CREATING SIMPLE SERVLET

- There are three different ways to create a servlet.
  1. By implementing Servlet interface.
  2. By extending GenericServlet class.
  3. By extending HttpServlet class.
- The detail of all these ways to create servlet is explained in following sections. Here we will see the steps for writing servlet by using netbeans IDE which is using apache tomcat as Server.
- To create a servlet application in Netbeans IDE, you will need to follow the following steps:

  Step 1 : Open Netbeans, Select file → New Project.

  Step 2 : Select Java Web → Web Application, then click on next as shown in Fig. 6.9.

Fig. 6.9

Step 3 : Give a name to your project and click next, (See Fig. 6.10).

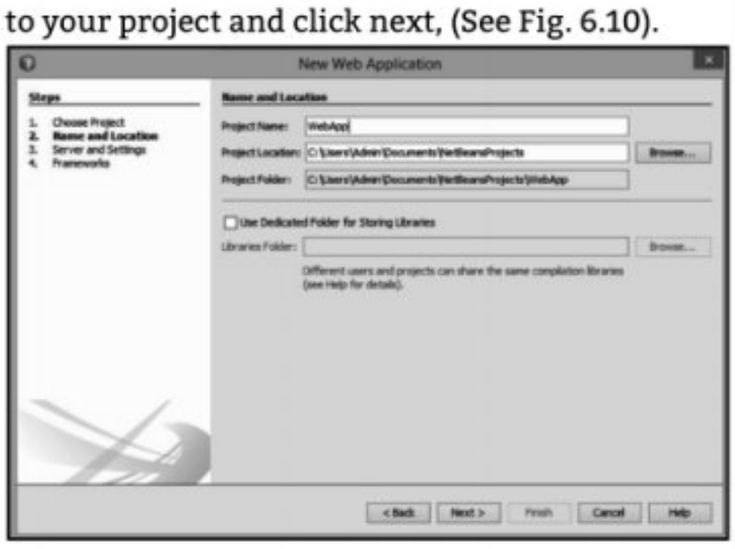


Fig. 6.10

Step 4 : As you finished the wizard, you will see WebApp Project directory is created in project tab. To create a servlet, right click on Source Package → New → Servlet as shown in Fig. 6.11.

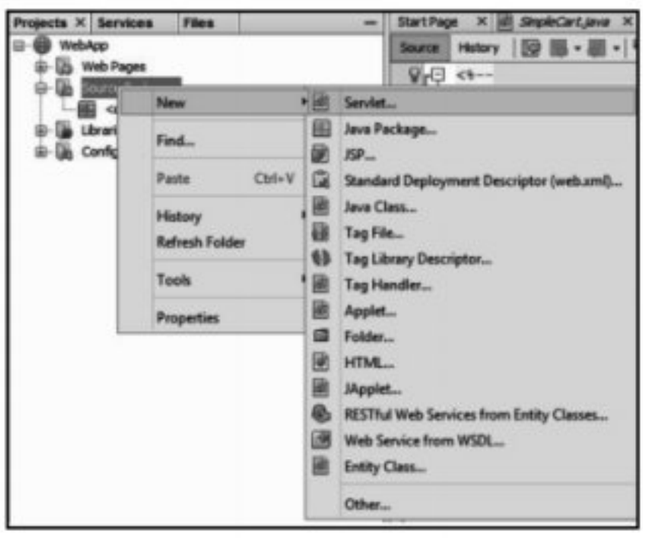


Fig. 6.11

Step 5 : Give a name to your servlet class file and click on next, (See Fig. 6.12). On next screen select the check box "Add information to deployment descriptor(web.xml)" and select finish as shown in Fig. 6.13.

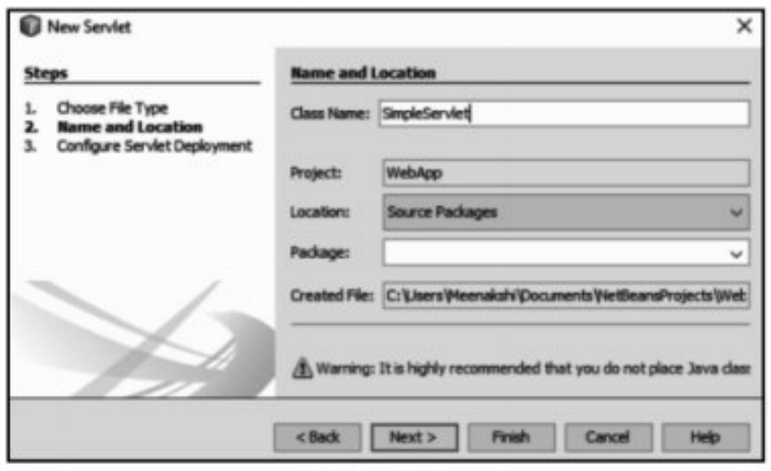


Fig. 6.12

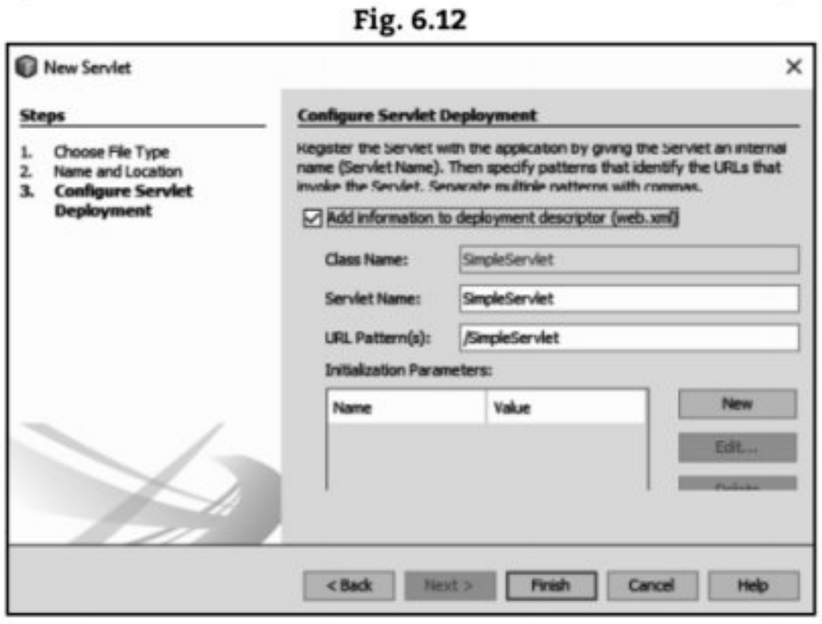


Fig. 6.13

Step 6 : Above step would create a web.xml file in WEB-INF folder and Format of **web.xml** file given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  version="3.1"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <servlet>
    <servlet-name>SimpleServlet</servlet-name>
    <servlet-class>SimpleServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SimpleServlet</servlet-name>
    <url-pattern>/SimpleServlet</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Step 7 : Write the following code in **SimpleServlet.java** file:

```
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.io.*;
import java.util.*;
public class SimpleServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
            HttpServletResponse response)throws ServletException, IOException
  {
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Servlet Testing</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Welcome to the Servlet Programming");
    out.println("</BODY>");
    out.println("</HTML>");
  }
}
```

Output:

Here, you can see that a SimpleServlet class is extended from HTTPServlet class and overriding doGet() methods. To run the file select "SimpleServlet.java" in Project tab and right click on it and select "Run File" and you will get appropriate outcome.

Welcome to the Servlet Programming

Program 6.1: Demonstrating life cycle of a Servlet.

```
import javax.servlet.*;
import java.io.IOException;
public class PrimitiveServlet implements Servlet
{
   public void init(ServletConfig config) throws ServletException
   {
      System.out.println("init");
   }
   public void service(ServletRequest request, ServletResponse
                                          response) throws ServletException, IOException
   {
      System.out.println("service");
   }
   public void destroy()
   {
   System.out.println("destroy");
   }
   public String getServletInfo()
  {
    return null;
   }
   public ServletConfig getServletConfig()
   {
    return null;
    }
}
```

- To run the file select "PrimitiveServlet.java" in Project tab and right click on it and select "Run File" or You can call this servlet from your browser by typing the following URL:

```
http://localhost:8080/WebApp/PrimitiveServlet
```

- The first time the servlet is called, the console displays these two lines:

```
init
service
```

- This tells you that the init method is called, followed by the service method. However, on subsequent requests(page refresh), only the service method is called. The servlet adds the following line to the console:

```
service
```

- This proves that the init method is called only once.

6.3.1 Servlet API

- The Servlet API comes in two Java packages. The packages are as follows:

1. **javax.servlet Package:**

- The javax.servlet package is used for serving protocol-less requests. It contains the interfaces and classes that every servlet must implements and extends respectively.
- Every servlet must implement the Servlet interface in one form or another. The abstract GenericServlet class provides the framework for developing basic servlets.

2. **javax.servlet.http Package:**

- The javax.servlet.http package is used for the development of servlets that use the HTTP protocol. The abstract HttpServlet class extends javax.servlet.GenericServlet and serves as the base class for HTTP servlets.
- HttpServletRequest and HttpServletResponse interfaces allow additional interaction with the client. This package also includes HttpSession and some related classes to support session tracking.

6.3.1.1 Javax.servlet Package

- The javax.servlet package is the core of the servlet API.
- It includes the basic servlet interface which all servlets must implement in one form or another and an abstract GenericServlet class for developing basic servlets.
- As shown in Fig. 6.14, this package includes server interfaces and three classes. For communicating with the host server and client.
- The javax.servlet package is used for serving protocol-less requests. It contains the interfaces and classes that every servlet must implements and extends respectively. Every servlet must implement the Servlet interface in one form or another. The abstract GenericServlet class provides the framework for developing basic servlets.
- Following tables lists some of the interfaces and classes of javax.servlet package.

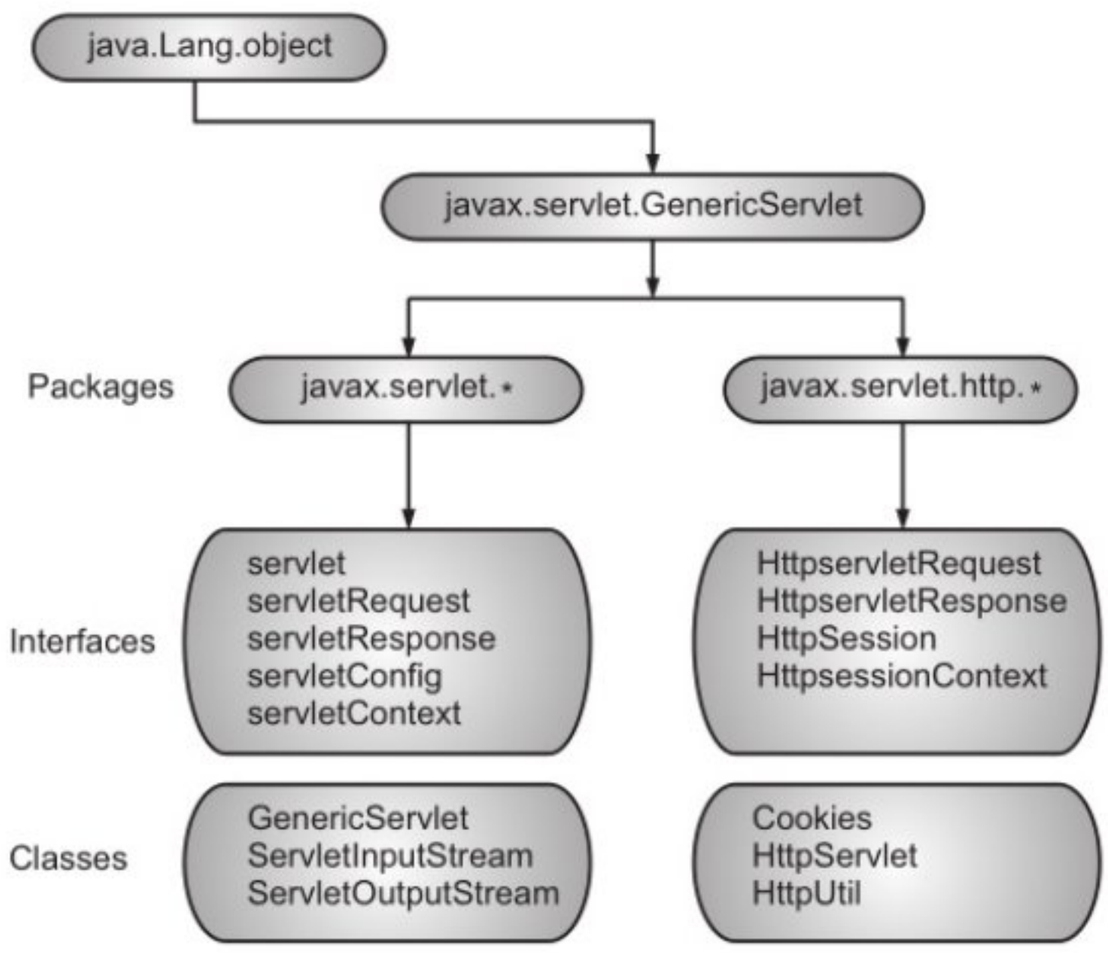


Fig. 6.14: Servlet API

Sr. No.	Interfaces	Sr. No.	Classes
1.	Servlet	1.	GenericServlet
2.	ServletRequest	2.	ServletInputStream
3.	ServletResponse	3.	ServletOutputStream
4.	RequestDispatcher	4.	ServletRequestWrapper
5.	ServletConfig	5.	ServletResponseWrapper
6.	ServletContext	6.	ServletRequestEvent
7.	SingleThreadModel	7.	ServletContextEvent
8.	Filter	8.	ServletRequestAttributeEvent
9.	FilterConfig	9.	ServletContextAttributeEvent
10.	ServletRequestListener	10.	ServletException

6.3.1.2 Servlet Interface

- Servlet interface is a collection of empty method signatures.
- A servlet must directly or indirectly, (by extending the generic class or httpServletclass) implement the servlet interface.
- This interface hold method signature that bring the following servlet functionalities.
  - Initializing the servlet.
  - Handling a client request.
  - Destroying a servlet.
- Following are the **methods** available in Servlet interface:

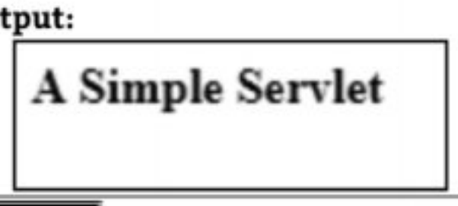
Sr. No.	Methods	Description
1.	void init(ServletConfig config)	init() is used for initializing the Servlet parameters provided by the ServletConfig object. init() called only once when the Servlet is first loaded. It is commonly used to initialize resources to be used by a Servlet when requests are received.
2.	void destroy()	destroy() used to clear all retained resources such as database connection, threads, file handles and so on. This method is overridden in order to free up any resources being used by the Servlet.
3.	void service(ServletRequest request, ServletResponse response)	service() is the actual heart of the HTTP Request-Response model. service() called to handle a client request.
4.	ServletConfig getServletConfig()	getServletConfig() object for initializing the Servlet's parameters.
5.	String getServletInfo()	Provides the Servlet metadata such as author, Servlet version and other copyright information.

Program 6.2: Implementing Servlet interface.

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.IOException;
import java.io.PrintWriter;
public class SimpleServlet implements Servlet{
  public void init(ServletConfig config) {
  System.out.println("servlet is initialized");
  }
```

```
    public void service(ServletRequest req, ServletResponse res)
      throws IOException, ServletException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.print("<html><body>");
      out.print("<b>A Simple Servlet</b>");
      out.print("</body></html>");
    }
    public void destroy() {
      System.out.println("servlet is destroyed");
    }
    public ServletConfig getServletConfig() {
      return null;
    }
    public String getServletInfo() {
      return null;
    }
  }
```

Output:



6.3.1.3 ServletContext Interface

- All servlets belongs to one servlet context.ServletContext can only be called at context initialization time.
- For every Web application a ServletContext object is created by the web container. The ServletContext object is used to get configuration information from Deployment Descriptor (web.xml) which will be available to any servlet or JSPs that are part of the web app.
- There are many methods in ServletContext interface. These are as follows:
  1. `Object getAttribute(String name)` method returns the container attribute with the given name, or null if there is no attribute by that name.
  2. `String getInitParameter(String name)` method returns parameter value for the specified parameter name, or null if the parameter does not exist.
  3. `Enumeration getInitParameterNames()` method returns the names of the context's initialization parameters as an Enumeration of String objects.
  4. `void setAttribute(String name,Objectobj)` method set an object with the given attribute name in the application scope.
  5. `void removeAttribute(String name)` method removes the attribute with the specified name from the application context.

Program 6.3:

1. Change the content of Deployment Descriptor (web.xml) file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" >
<servlet>
  //some code here…….
</servlet>
<servlet-mapping>
  //some code here……
</servlet-mapping>
```

```
<session-config>
 //some code here……
</session-config>
<context-param>
   <param-name>driverName</param-name>
   <param-value>sun.jdbc.JdbcOdbcDriver</param-value>
</context-param>
</web-app>
```

2. Write following code Inside Servlet class:

```
import java.io.*;
importjavax.servlet.*;
importjavax.servlet.http.*;
public class MyServlet2 extends HttpServlet
{
private String message;
public void doGet(HttpServletRequestrequest,HttpServletResponse response)
throwsServletException, IOException
   {
   response.setContentType("text/html;charset=UTF-8");
   PrintWriter out = response.getWriter();
   ServletContext sc=getServletContext();
   out.println(sc.getInitParameter("driverName"));
    }
}
```

- This will display "sun.jdbc.JdbcOdbcDriver " on web page.

6.3.1.4 ServletConfig Interface

- When the Web Container initializes a servlet, it creates a ServletConfig object for the servlet It allows a servlet to obtain configuration data when it is loaded. The configuration information is obtained from web.xml file (Deployment Descriptor).
- There are many methods in ServletConfig interface. These are as follows:
  1. `String getInitParameter(String name)` method returns a String value initialized parameter, or null if the parameter does not exist.
  2. `Enumeration getInitParameterNames()` method returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
  3. `ServletContext getServletContext()` method returns a reference to the ServletContext.
  4. `String getServletName()` method returns the name of this servlet instance.

Program 6.4:

1. Change the content of Deployment Descriptor (web.xml) file as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" >
<servlet>
<servlet-name>MyServlet1</servlet-name>
<servlet-class>MyServlet1</servlet-class>
<init-param>
<param-name>name</param-name>
<param-value>Meenakshi</param-value>
</init-param>
</servlet>
```

```
<servlet-mapping>
//some code here….
</servlet-mapping>
<session-config>
//some code here….
</session-config>
</web-app>
```

2. Write following code Inside a Servlet class:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet1 extends HttpServlet
{
private String message;
public void doGet(HttpServletRequestrequest,HttpServletResponse response)
throwsServletException, IOException
    {
    response.setContentType("text/html ");
    PrintWriter out = response.getWriter();
    ServletConfigsc=getServletConfig();
    out.println(sc.getInitParameter("name"));
     }
}
```

- This will display “Meenakshi” on web page.

6.3.1.5 ServletRequest Interface

- When a servlet accepts a call from a client, it receives two objects, one is ServletRequest and other is ServletResponse.
- ServletRequest interface encapsulates the communication from the client to the server. ServletRequest interface allows the servlet to access information such as:
  - Names of the parameters passed by the client.
  - The protocol such as http POST and PUT methods being used by client.
  - The names of the remote host that made the request.
  - The server that received it.
  - An InputStream for reading binary data from the request body.
- The following are the commonly used methods available in ServletRequest interface:

Sr. No.	Methods	Description
1.	`Object getAttribute()`	This method returns the value of the named attribute as an object.
2.	`Enumeration getAttributeNames()`	Returns an Enumeration containing the names of the attributes to this request.
3.	`String getParameter()`	Returns the value of a request parameter as a String.
4.	`Enumeration getParameterNames()`	Returns an Enumeration of String objects containing the names of the parameters to this request.
5.	`String[] getParameterValues()`	Returns an array of String objects containing all of the values the given request parameter has.

- The ServletRequest interface provides important methods that enable you to access information about the user.

- For example, the getParameterNames method returns an Enumeration containing the parameter names for the current request.
- To get the value of each parameter, the ServletRequest interface provides the getParameter method.

6.3.1.6 ServletResponse Interface

- ServletResponse interface provides methods to the servlet for replying to the client.
- ServletResponse interface allows servlet:
  1. to set the content length and type of the reply.
  2. provides an output stream and a writer.
- Through servletResponse interface, the servlet can send the reply data.
- Subclasses of ServletResponse provide the servlet with more protocol-specific capabilities for e.g. HttpServletResponse contains methods that allow the servlet to manipulate HTTP-specific header information.
- The following are the commonly used methods available in ServletResponse interface:

Sr. No.	Methods	Description
1.	`Locale getLocale()`	Returns the locale assigned to the response.
2.	`PrintWriter getWriter()`	Returns a ServletOutputStream suitable for writing binary data in the response.
3.	`void reset()`	Returns a PrintWriter object that can send character text to the client.
4.	`void setLocale()`	Clears any data that exists in the buffer as well as the status code and headers.
5.	`void setContentType(String)`	This method is used to set the content type to be sent to user as output.

- The ServletResponse interface represents the response to the user.
- The most important method of this interface is getWriter, from which you can obtain a java.io.PrintWriter object that you can use to write HTML tags and other text to the user, (send output of servlet to user).

Program 6.5: Using ServletResponse and ServletResponse. The example consists of an HTML file(index.html) and a servlet (RequestDemoServlet).

index.html:

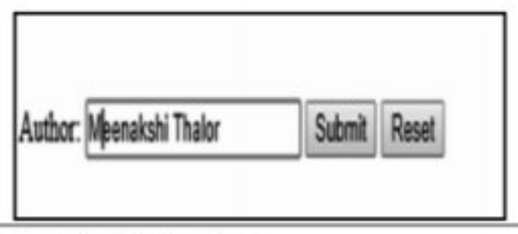
```
<html>
  <head>
    <title>Sending a request</title>
  </head>
  <body>
    <form action=" RequestDemoServlet">
      <br><br>
      Author: <input type="text" name="Author">
      <input type="submit" name="Submit">
      <input type="reset" value="Reset">
    </form>
  </body>
</html>
```

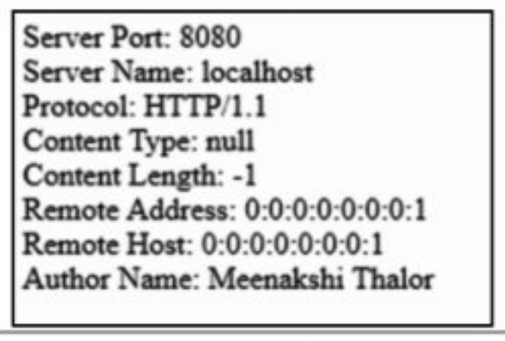
RequestDemoServlet.java:

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.IOException;
import java.io.PrintWriter;
public class RequestDemoServlet implements Servlet {
```

```
public void init(ServletConfig config) throws ServletException {
}
public void destroy() {
}
public void service(ServletRequest request, ServletResponse response)
                                                    throws ServletException, IOException {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  out.println("Server Port: " + request.getServerPort()+"<br>");
  out.println("Server Name: " + request.getServerName()+"<br>");
  out.println("Protocol: " + request.getProtocol()+"<br>");
  out.println("Content Type: " + request.getContentType()+"<br>");
  out.println("Content Length: " + request.getContentLength()+"<br>");
  out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
  out.println("Remote Host: " + request.getRemoteHost()+"<br>");
  Enumeration parameters = request.getParameterNames();
  out.println("Author Name: " + request.getParameter("Author")+"<br>");
}
public String getServletInfo() {
  return null;
}
public ServletConfig getServletConfig() {
  return null;
}
```

- Run the index.html file and enter author detail. When you submit the form, the request will be forwarded from browser to RequestDemoServlet.





Problem with Servlet Interface:

- Everything works fine, but there are two annoying things that you have probably noticed:
  1. You have to provide implementations for all five methods of the Servlet interface, eventhough most of the time you only need one. This makes your code look unnecessarily complicated.
  2. The ServletConfig object is passed to the init method. You need to preserve this object to use it from other methods. This is not difficult, but it means extra work. This Problem is solved by GenericServlet.

6.3.2 GenericServlet Class

- This is first type of Servlet. javax.servlet.GenericServlet class provides the basic implementation of the servlet interface and servletConfig interface. GenericServlet makes writing servlets easier and simpler.
- The javax.servlet package provides a wrapper class called GenericServlet that implements two important interfaces from the javax.servlet package i.e., Servlet and ServletConfig, as well as the java.io.Serializable interface.

- The GenericServlet class provides implementations for all methods, most of which are blank. You can extend GenericServlet and override only methods that you need to use. Clearly, this looks like a better solution.
- Once, you convert your program from servlet to GenericServlet you have to only write service method.
- The code in Program 6.5 is a servlet called GenericServletDemo that extends GenericServlet.
- The code provides the implementation of the service method that sends some output to the browser. Because the service method is the only method you need, only this method needs to appear in the class.
- Following table lists some of the **methods of GenericServlet** class:

Sr. No.	Methods	Description
1.	`void init(ServletConfigconfig)`	Invoked only once .Used to initialize the servlet.
2.	`abstract void service (ServletRequest request, ServletResponse response)`	Provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3.	`void destroy()`	Invoked only once .Indicates that servlet is being destroyed.
4.	`ServletConfig getServletConfig()`	Returns the object of ServletConfig.
5.	`String getServletInfo()`	returns information about servlet like writer, copyright, version etc.
6.	`ServletContext getServletContext()`	Returns the object of ServletContext.
7.	`String getInitParameter(String name)`	Returns the parameter value for the given parameter name.
8.	`Enumeration getInitParameterNames()`	Returns all the parameters defined in the web.xml file.
9.	`String getServletName()`	Returns the name of the servlet object.
10.	`void log(String msg)`	Writes the given message in the servlet log file.
11.	`void log(String msg,Throwable t)`	Writes the explanatory message in the servlet log file and a stack trace.

Program 6.6: Extending GenericServlet class.

```
import java.io.*
import javax.servlet.*;
public class GenericServletDemo extends GenericServlet
{
  public void service(ServletRequest req,ServletResponse res)
                                                    throws IOException,ServletException
  {
    res.setContentType("text/html");
    PrintWriter out=res.getWriter();
    out.print("<html><body>");
    out.print("<b>A Generic servlet</b>");
    out.print("</body></html>");
  }
}
```

Output:

A Generic servlet

Program 6.7: Extending GenericServlet class.

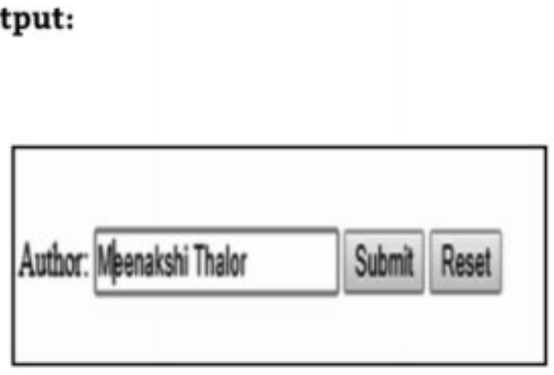
index.html

```
<html>
  <head>
    <title>Sending a request</title>
  </head>
  <body>
    <form action="GenericServletDemo">
      <br><br>
      Author: <input type="text" name="Author">
      <input type="submit" name="Submit">
      <input type="reset" value="Reset">
    </form>
  </body>
</html>
```

GenericServletDemo.java :

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.*;
public class GenericServletDemo extends GenericServlet
{
  public void service(ServletRequest request, ServletResponse response)
                                                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("Server Port: " + request.getServerPort()+"<br>");
    out.println("Server Name: " + request.getServerName()+"<br>");
    out.println("Protocol: " + request.getProtocol()+"<br>");
    out.println("Content Type: " + request.getContentType()+"<br>");
    out.println("Content Length: " + request.getContentLength()+"<br>");
    out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
    out.println("Remote Host: " + request.getRemoteHost()+"<br>");
    Enumeration parameters = request.getParameterNames();
    out.println("Author Name: " + request.getParameter("Author")+"<br>");
  }
}
```

Output:



Server Port: 8080
Server Name: localhost
Protocol: HTTP/1.1
Content Type: null
Content Length: -1
Remote Address: 0:0:0:0:0:0:0:1
Remote Host: 0:0:0:0:0:0:0:1
Author Name: Meenakshi Thalor

Problem with GenericServlet:

1. The GenericServlet is protocol depended servlet.
2. Protocol depended means it works with all protocol.
3. It gives services when the request is coming from any protocol ex FTP, SMTP, POP3.
4. But most of the time the user sends request through http protocol.
5. GenericServlet provide services to http protocol also but doesnot support all functionality (methods) of http protocol. So we need a servlet that support full functionality of http protocol.

6.4 javax.servlet.http PACKAGE

6.4.1 HTTP Methods

- HttpServlet servlet is used for such cases where we require all the functionality (methods) of http protocol.
- Each HTTP request can use one of the many request methods as specified in the HTTP standards.
- The HTTP request methods and the descriptions of each method are given in the following table:

Sr. No.	Methods	Description
1.	GET	GET is the simplest and probably, most used HTTP method. GET simply retrieves the data identified by the URL.
2.	HEAD	The HEAD method provides the same functionality as GET, but HEAD only returns HTTP headers without the document body.
3.	POST	Like GET, POST is also widely used. Typically, POST is used in HTML forms. POST is used to transfer a block of data to the server in the entity body of the request.
4.	OPTIONS	The OPTIONS method is used to query a server about the capabilities it provides. Queries can be general or specific to a particular resource.
5.	DELETE	The DELETE method is used to delete a document from the server. The document to be deleted is indicated in the URI (Uniform Resource Identifier) section of the request.
6.	PUT	The PUT method is a complement of a GET request and PUT stores the entity body at the location specified by the URI. It is similar to the PUT function in FTP.
7.	TRACE	The TRACE method is used to trace the path of a request through firewall and multiple proxy servers. TRACE is useful for debugging complex network problems and is similar to the trace route tool.

6.4.2 The javax.servlet.http Package

- The javax.servlet.http package is used for the development of servlets that use the HTTP protocol. The abstract HttpServlet class extends javax.servlet.GenericServlet and serves as the base class for HTTP servlets.
- HttpServletRequest and HttpServletResponse interfaces allow additional interaction with the client. This package also includes HttpSession and some related classes to support session tracking.
- Following table lists some of the interfaces and classes of javax.servlet.http package:

Sr. No.	Interfaces	Sr. No.	Classes
1.	HttpServletRequest	1.	HttpServlet
2.	HttpServletResponse	2.	Cookie
3.	HttpSession	3.	HttpServletRequestWrapper
4.	HttpSessionListener	4.	HttpServletResponseWrapper
5.	HttpSessionAttributeListener	5.	HttpSessionEvent
6.	HttpSessionBindingListener		

6.4.3 HttpServlet Class

- The HttpServlet class extends the javax.servlet.GenericServlet class. The HttpServlet class also adds a number of methods for use.
- The most important are the different doxxx methods that get called when a related HTTP request method is used.
- The six methods are doPost, doPut, doGet, doDelete, doOptions and doTrace.
- Each doxxx() method is invoked when a corresponding HTTP method is used. For instance, the doGet method is invoked when the servlet receives an HTTP request that was sent using the GET method.
- Among these doxxx methods, the doPost and the doGet methods are the most frequently used. The doPost method is called when the browser sends an HTTP request using the POST method.
- Consider the following HTML form at the client side:

```
<form action="Register" method="post">
<input type=text name="firstName">
<input type=text name="lastName">
<input type=submit>
</form>
```

- When the user clicks the Submit button to submit the form, the browser sends an HTTP request to the server using the POST method.
- The web server then passes this request to the Register servlet and the doPost method of the servlet is invoked.
- Using the POST method in a form, the parameter name/value pairs of the form are sent in the request body.
- For example, if you use the preceding form as an example and enter Tanmay as the value for firstName and Gurunani as the value for lastName, you will get the following result in the request body:

```
firstName=Tanmay
lastName=Gurunani
```

- An HTML form can also use the GET method. The doGet method is invoked when an HTTP request is sent using the GET method. GET is the default method in HTTP. If you use the GET method in a form, the parameter name/value pairs are appended to the URL.
- Upon receiving a GET method, the servlet will call its doGet method. The HTTP method used by the client request. Knowing the HTTP method, the service method simply calls the corresponding doxxx method.
- Following table lists some of the **methods of HttpServlet** class:

Sr. No.	Methods	Description
1.	`void service(ServletRequest req,ServletResponse res)`	Dispatches the request to the protected service method by converting the request and response object into http type.
2.	`void service(HttpServletRequest req, HttpServletResponse res)`	Receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3.	`void doGet(HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the GET request.
4.	`void doPost (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the POST request.
5.	`void doHead (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the HEAD request.
6.	`void doOptions (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the OPTIONS request.

contd. ...

7.	`void doPut (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the PUT request.
8.	`void doTrace (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the TRACE request.
9.	`void doDelete (HttpServletRequest req, HttpServletResponse res)`	Invoked by the web container and handles the DELETE request.
10.	`long getLastModified (HttpServletRequest req)`	Returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Program 6.8: Program for HttpServlet class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
    {
    PrintWriter out = response.getWriter();
    out.println("<h1>Hello world </h1>");
    }
}
```

Output:

Hello World

6.4.4 HttpServletRequest Interface

- This interface is implemented by server. It enables servlet request to obtain information about a client request.
- This interface provides method for extracting HTTP parameters from the query string or the request body depending on the types of request such as GET or POST.
- HttpServletRequest Interface extends ServletRequest Interface to provide request information for HTTP servlet.
- This interface includes support for:
  - Cookies,
  - Session tracking, and
  - Access to HTTP header information.
- Some of the commonly available **methods in HttpServletRequest** interface are:

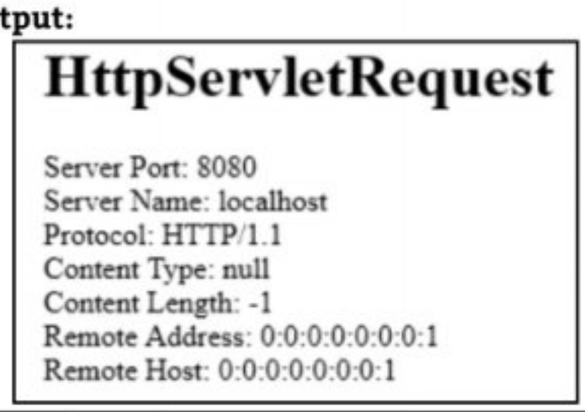
Sr. No.	Methods	Description
1.	`Cookie[] getCookies()`	Returns an array containing all of the Cookie objects the client sent with this request.
2.	`String getQueryString()`	Returns the query string that is contained in the request URL after the path.
3.	`HttpSession getSession()`	Returns the current session associated with this request or if the request does not have a session, creates one.

Program 6.9: Program for HttpServletRequest interface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
```

```
{
public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("<h1>HttpServletRequest</h1>");
        out.println("Server Port: " + request.getServerPort()+"<br>");
        out.println("Server Name: " + request.getServerName()+"<br>");
        out.println("Protocol: " + request.getProtocol()+"<br>");
        out.println("Content Type: " + request.getContentType()+"<br>");
        out.println("Content Length: " + request.getContentLength()+"<br>");
        out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
        out.println("Remote Host: " + request.getRemoteHost()+"<br>");
    }
}
```

Output:



6.4.5 HttpServletResponse Interface

- The HttpServletResponse interface is implemented by the server. It enables a servlet to formulate an HTTP response to a client.
- HttpServletResponse interface extends ServletResponse interface to provide HTTP protocol specific functionality.
- The HttpServletResponse interface provides several protocol-specific methods not available in the javax.servlet.ServletResponse interface.
- The HttpServletResponse interface extends the javax.servlet.ServletResponse interface.
- Some of the important **methods of HttpServletResponse interface** are given below:

Sr. No.	Methods	Description
1.	`void addCookie(Cookie cookie)`	Adds the specified cookie to the response.
2.	`void sendRedirect(String location)`	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
3.	`int getStatus()`	Gets the current status code of this response.
4.	`String getHeader(String name)`	Gets the value of the response header with the given name.
5.	`String getHeaderNames()`	Gets the value of the response header.
6.	`void setHeader(String name, String value)`	Sets a response header with the given name and value.
7.	`void setStatus(int sc)`	Sets the status code for this response.
8.	`void sendError(int sc, String msg)`	Sends an error response to the client using the specified status and clears the buffer.

Program 6.10: Program for HttpServletResponse interface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
public void doGet(HttpServletRequest request,HttpServletResponse response)
                                                    throws ServletException, IOException
   {
   PrintWriter out = response.getWriter();
   out.println("<h1>HttpServletResponse</h1>");
    out.println("Status Code:" + response.getStatus());
    out.println("<br>Header:"+response.getHeaderNames());
    //response.sendRedirect("http://www.google.com");
    out.close();
   }
}
```

Output:

HttpServletResponse

Status Code:200
Header:[Server, X-Powered-By]

6.4.6 Cookie Class

- HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Still there are following ways to maintain session between web client and web server:
  1. Cookies.
  2. HttpSession Object.
- A web server can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the recieved cookie.This may not be an effective way because many time browser does not support a cookie.
- javax.servlet.http.Cookie class provides the functionality of using cookies. It provides following constructors and methods for creation and manipulation of cookies.

Sr. No.	Constructors	Description
1.	`Cookie()`	Constructs a cookie.
2.	`Cookie(String name, String value)`	Constructs a cookie with a specified name and value.

Sr. No.	Methods	Description
1.	`void setMaxAge(int expiry)`	Sets the maximum age of the cookie in seconds.
2.	`String getName()`	Returns the name of the cookie. The name cannot be changed after creation.
3.	`String getValue()`	Returns the value of the cookie.
4.	`void setName(String name)`	Changes the name of the cookie.
5.	`void setValue(String value)`	Changes the value of the cookie.

- Other methods are also required for using Cookies in any java program. For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. These methods are:
  1. `public void addCookie(Cookie ck)` method of HttpServletResponse interface is used to add cookie in response object.
  2. `public Cookie[] getCookies()` method of HttpServletRequest interface is used to return all the cookies from the browser.

Program 6.11: Program for cookies class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class NewServlet2 extends HttpServlet
 {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException
  {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      Cookie cookie1 = new Cookie("name","Meenakshi");
      Cookie cookie2 = new Cookie("name","Anurag");
      response.addCookie(cookie1);
      cookie1.setMaxAge(60*60);
      response.addCookie(cookie2);
      cookie2.setMaxAge(60*60);
      pw.println("Cookies created");
      pw.println(cookie1.getName() + " : " + cookie1.getValue());
      pw.println(cookie2.getName() + " : " + cookie2.getValue());
   }
}
```

Output:

```
Cookies created
name : Meenakshi
name : Anurag
```

6.4.7 HttpSession Interface

- HttpSession interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.

Methods of HttpSession Interface:

Sr. No.	Methods	Description
1.	`Object getAttribute(String name) throws IllegalStateException`	This method retrieves an attribute from the HttpSession object. The return value is an object of type Object.
2.	`java.util.Enumeration getAttributeNames() throws IllegalStateException`	The getAttributeNames method returns a java.util.Enumeration containing all attribute names in the HttpSession object.

contd. ...

3.	`long getCreationTime() throws IllegalStateException`	The getCreationTime method returns the time that the HttpSession object was created, in milliseconds.
4.	`String getId()`	The getID method returns the session identifier.
5.	`long getLastAccessedTime()`	The getLastAccessedTime method returns the time the HttpSession object was last accessed by the client.

Program 6.12: Program for HttpSession interface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    HttpSession s = request.getSession();
    if (s != null)
    {
      pw.println("Session Id : ");
      String s_id = s.getId();
      pw.println(s_id);
    } else
    {
      pw.println("Your session is not created yet");
    }
  }
}
```

Output:

```
Session Id : a5b7fba8ad37bc397ec79de89d79
```

6.4.8 HttpSessionEvent Class

- The purpose of HttpSessionEvent class is to give notifications for any changes to sessions within a web application.This class work in pair with HTTPSessionListerner.
- HttpSessionListener receives notifications of changes to the list of active sessions in a web application and perform some action.
- There are following two methods declared in the HttpSessionListener interface which must be implemented by the servlet programmer to perform some action.

Sr. No.	Methods	Description
1.	`void sessionCreated(HttpSessionEvent e)`	This method invokes when session object is created.
2.	`void sessionDestroyed(ServletContextEvent e)`	This method invokes when session is invalidated.

Program 6.13: Program for HttpSessionEvent class.

SimpleServlet.java:

```
import java.io.*;
import javax.servlet.*;
MySessionListener.java:
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    HttpSession s = request.getSession();
    if (s != null)
    {
      pw.println("Session Id : ");
      String s_id = s.getId();
      pw.println(s_id);
    } else
    {
      pw.println("Your session is not created yet");
    }
  }
}
import javax.servlet.http.*;
public class MySessionListener implements HttpSessionListener
{
  public void sessionCreated(HttpSessionEvent se)
   {
    System.out.println(" HttpSession Created ");
    HttpSession session = se.getSession();
    System.out.println("session id: " + session.getId());
    session.setMaxInactiveInterval(5);//in seconds
  }
  public void sessionDestroyed(HttpSessionEvent se) {
    System.out.println("HttpSession Destroyed");
  }
}
```

Output:

Session Id : a7b20fc83f36a2e0010f17d34b8f

On Console (After some time):

```
HttpSession Created
session id: a7b20fc83f36a2e0010f17d34b8f
HttpSession Destroyed
```

6.4.9 HttpSessionBindingEvent Class

- A javax.servlet.http.HttpSessionBindingEvent object is used for two purposes:
  1. A javax.servlet.http.HttpSessionBindingEvent object act as input to calls HttpSessionAttributeListener methods.
     The HttpSessionAttributeListener interface is implemented to get notifications of changes to the attribute lists of sessions within this web application.

2. A javax.servlet.http.HttpSessionBindingEvent object act as input to calls HttpSessionBindingListener methods.
   The HttpSessionBindingListener interface is implemented to give notifications to an object when it is bound to or unbound from a session.

- The HttpSessionBindingEvent class includes the following methods, which your listener can call:

Sr. No.	Methods	Description
1.	`String getName()`	Use this method to get the name of the attribute that was added, removed, or replaced.
2.	`Object getValue()`	Use this method to get the value of the attribute that was added, removed, or replaced. In the case of an attribute that was replaced, this method returns the old value, not the new value.
3.	`HttpSession getSession()`	Use this method to retrieve the session object that had the attribute change.

Progeram 6.14: Program for HttpSessionBindingEvent class.

SimpleServlet.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    HttpSession s = request.getSession();
     if (s != null)
    {
      pw.println("Session Id : ");
      String s_id = s.getId();
      pw.println(s_id);
      s.setAttribute("FName", "Meenakshi"); //attributeAdded() is executed
      s.setAttribute("FName", "Thalor"); //attributeReplaced() is executed
      s.removeAttribute("FName"); //attributeRemoved() is executed
    } else
    {
      pw.println("Your session is not created yet");
    }
  }
}
```

User.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
public class User implements HttpSessionAttributeListener{
public void attributeAdded(HttpSessionBindingEvent event){
   System.out.println("Attribute added " + event.getName() + " : " + event.getValue());
  }
   public void attributeRemoved(HttpSessionBindingEvent event){
     System.out.println("Attribute removed " + event.getName() + " : " + event.getValue());
   }
   public void attributeReplaced(HttpSessionBindingEvent event){
    System.out.println("Attribute replaced " + event.getName() + " : " + event.getValue());
   }
}
```

Output:

```
Session Id : ac31a00de090244009e8b52a6242
```

On Console:

```
Attribute added FName : Meenakshi
Attribute replaced FName : Meenakshi
Attribute removed FName : Thalor
```

6.5 HANDLING HTTP REQUEST AND RESPONSE

- For HttpServlet, service()method dispatches doGet(), doPost() to handle HTTP GET, POST request respectively, (For detail Refer Section 6.2, Point (2)).

6.5.1 Handling httpRequest

- The purpose of the HttpRequest object is to represent the HTTP request a browser sends to your web application. Thus, anything the browser may send, is accessible via the HttpRequest.
- The HttpRequest object is generally used to retrieve the request parameter values. The request parameters are parameters that are sent from the browser along with the request.
- Request parameters are typically sent as part of the URL (in the "query string"), or as part of the body of an HTTP request. For instance:

```
http://google.com/somePage.html?param1=hello & m2=world
```

- Notice the "query string" part of the URL: ?param1=hello&m2=world This part contains two parameters with parameter values:

```
param1=hello
param2=world
```

- You can access these parameters from the HttpRequest object like this:

```
protected void doGet( HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String param1 = request.getParameter("param1");
    String param2 = request.getParameter("param2");
}
```

- You would use the same code, if the request parameters were sent in the body part of the HTTP request.
- If no parameter exists with the given name, null is returned. If the browser sends an HTTP GET request, the parameters are included in the query string in the URL. If the browser sends an HTTP POST request, the parameters are included in the body part of the HTTP request.
- The HttpRequest object has a lot of methods, few of them are listed below:

Sr. No.	Methods	Description
1.	`Cookie[] getCookies()`	Returns an array containing all of the Cookie objects the client sent with this request.
2.	`java.lang.String getQueryString()`	Returns the query string that is contained in the request URL after the path.
3.	`HttpSession getSession()`	Returns the current session associated with this request, or if the request does not have a session, creates one.

6.5.2 Handling httpResponse

- The purpose of the HttpResponse object is to represent the HTTP response your web application sends back to the browser, in response to the HTTP request.
- To send HTML back to the browser, you have to obtain the a PrintWriter from the HttpResponse object. For this following code is used:

```
PrintWriter writer = response.getWriter();
writer.write("<html><body>GET/POST response</body></html>");
```

6.5.3 Handling HTTP GET Request

- The doGet() method is called when the browser sends an HTTP request using the Get method.
- The servlet is invoked when a form on webpage is submitted.Here we have two files, one html file and other is java source file.

Program 6.15: Calling doGet() method using form GET method.

1. Write following code in HTML page.

```
<html>
<head>
</head>
<body>
<form method=GET action="MyServlet3">
       Enter your Name: <input type=text size=45 name=user>
       <input type=submit value=SUBMIT>
</form>
</body>
</html>
```

2. Write following code in MyServlet3 class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet3 extends HttpServlet
{
   public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
      out.println("<h2> Welcome "+user+"</h2>");
    }
      public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
      out.println("<h2> Hello "+user+"</h2>");
    }
}
```

Here, as the <form Method=GET > so the doGet() method in MyServlet3 get executed.

Output:

```
Hello <Username entered by user>
```

6.5.4 Handling HTTP POST Request

- The doPost() method is called when the browser sends an HTTP request using the post method.
- Now lets develop one servlet that will handle the HTTP post request. The servlet is invoked when a form on a web page is submitted.

- Following program using two files, one html file and other is Java source file.

Program 6.16: Calling doPost() method using form POST method.

1. Write following code in HTML page.

```
<html>
<head>
</head>
<body>
<form method=POST action=" MyServlet4">
    Enter your Name: <input type=text size=45 name=user>
<input type=submit value=SUBMIT>
</form>
</body>
</html>
```

2. Write following code in MyServlet4 class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet4 extends HttpServlet
{
   public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
      out.println("<h2> Welcome "+user+"</h2>");
   }
   public void doGet(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
      out.println("<h2> Hello "+user+"</h2>");
    }
}
```

Here, as <form Method=POST >the doPost() method in MyServlet4 get executed.

Output:

```
Welcome <username entered by user>
```

Program 6.17: Calling doGet() method and doPost() method.

1. Write following code in HTML page.

```
<html>
<head>
</head>
<body>
<form method=GET action=" MyServlet5">
      Enter your Name: <input type=text size=45 name=user>
      <input type=submit value=SUBMIT>
</form>
</body>
</html>
```

2. Write following code in Servlet.

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
public class MyServlet5 extends HttpServlet
{
   public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
         out.println("<h2> Welcome "+user+"</h2>");
    }
      public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws ServletException, IOException
   {
         response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user=request.getParameter("user");
         out.println("<h2> Hello "+user+"</h2>");
         doPost(request,response);
    }
}
```

- In this as <form Method=GET> so doGet() method of servlet class will called and at the end of this method we are calling doPost method. So both going to execute and you will get following output:

6.6 COOKIES AND SESSION TRACKING

- Cookies are a way for a server (or a Servlet, as part of a server) to send some information to a client to store, and for the server to later retrieve its data from that client.
- Session tracking is a mechanism that Servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.
- Session management is a mechanism used by the Web container to store session information for a particular user.
- There are some techniques used by Servlet application for session management. They are Cookies and Session Object.

6.6.1 Concept of Session

- A session is defined as, "a collection of HTTP requests shared between a client and web server over a period of time". Session simply means a particular interval of time.
- Session is used to store everything that we can get from the client from all the requests the client makes.
- Fig. 6.15 shows how sessions work.

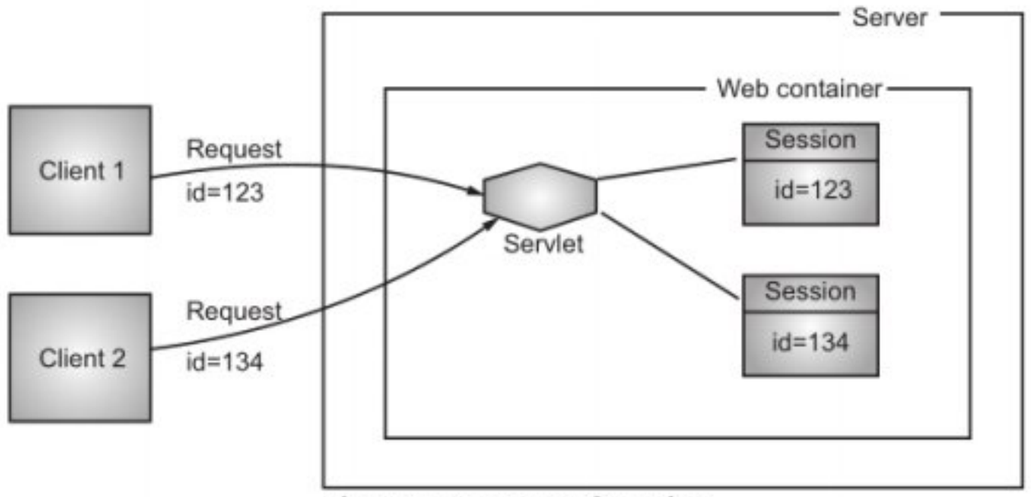


Fig. 6.15: Concept of Session

6.6.2 Session Tracking

- Session tracking is a way to maintain state (data) of an user. It is also known as session management in Servlet.
- Session tracking is a process to gather the detailed information from the web pages and to store user generated information.
- HTTP protocol is a stateless so we need to maintain state using session tracking techniques.
- Each time user requests to the server, server treats the request as the new request (Fig. 6.16).

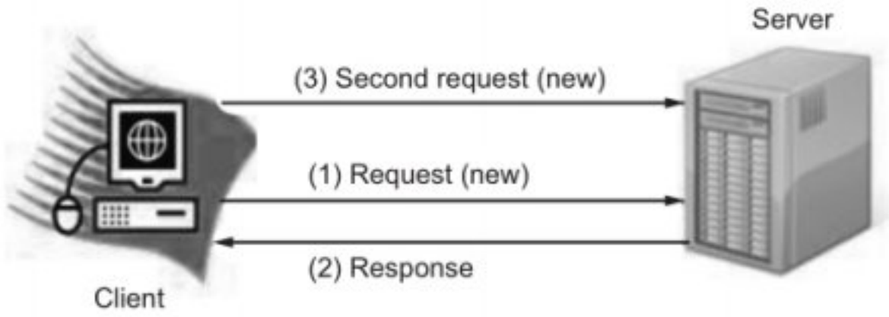


Fig. 6.16: Session Tracking

- So we need to maintain the state of an user to recognize to particular user.

6.6.3 Session Management with HttpSession

- The servlet container uses HttpSession interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user. Fig. 6.17 shows how it works. These steps are given below:
  1. On client's first request, the Web Container generates a unique session ID and gives it back to the client with response.
  2. The client sends back the session ID with each request.
  3. The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

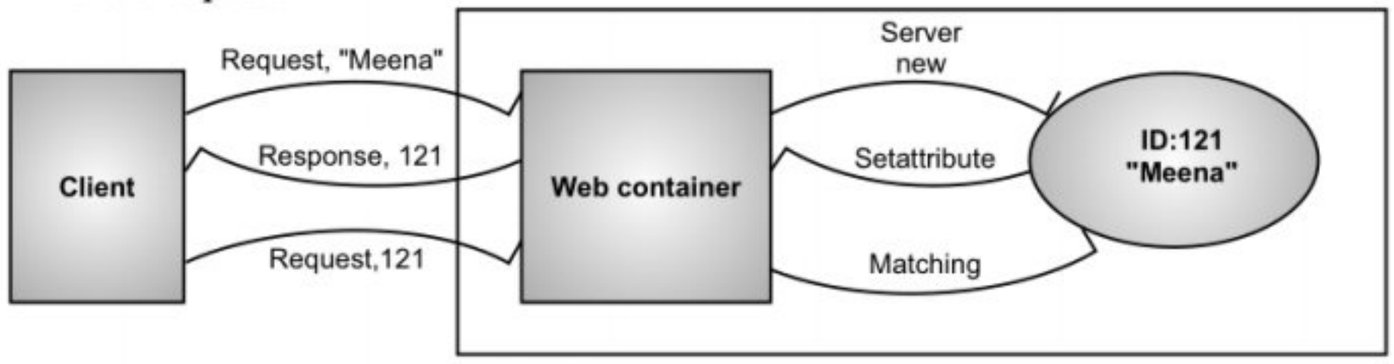


Fig. 6.17: Session Management

- An object that implements HttpSession interface is created by the servlet container and returned when the getSession() method of HttpServletRequest is called.
- The HttpSession object provides methods to read, add, and remove various session data as well as methods to view session information such as the session identifier, creation time, and the time the session was last accessed.
- The **syntax of getSession() methods** are as follows:
  1. `HttpSession getSession()` method always returns an object of HttpSession. It returns the current session associated with this request, if the request has no session attached, then it creates a new session and return it.
  2. `HttpSession getSession(boolean flag)` method returns HttpSession object if request has session else it returns null.

Program 6.18: Session Tracking: In following example we are passing the value of username and password to ValidateServlet class, which then check the password value if it is "1234" then response will be redirected to WelcomeServlet page, which will show hello <username>. So in this way a session is maintained in between the pages.

1. Write following code in HTML file.

```
<html>
<body>
<form method="post" action="ValidateServlet">
      User:<input type="text" name="user" /><br/>
      Password:<input type="text" name="pass" ><br/>
      <input type="submit" value="submit">
</form>
</body>
</html>
```

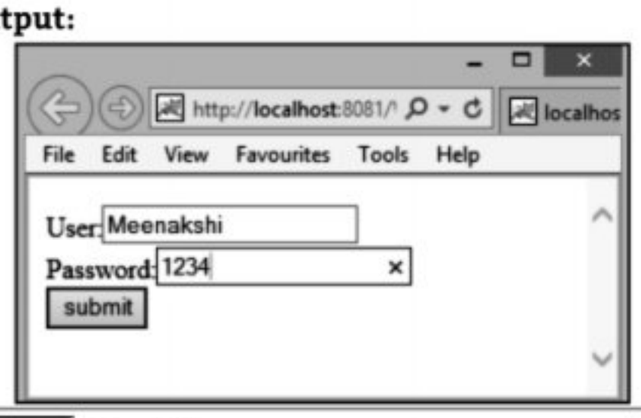
2. Write following code in ValidateServlet class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ValidateServlet extends HttpServlet
{
   public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException, IOException
    {
      response.setContentType("text/html");
      String name = request.getParameter("user");
      String pass = request.getParameter("pass");
      if(pass.equals("1234"))
      {
         //creating a session
         HttpSession session = request.getSession();
         session.setAttribute("user", name);
         response.sendRedirect("WelcomeServlet");
      }
      else
      {
         PrintWriter out = response.getWriter();
         out.println("Wrong password ");
         out.close();
      }
  }
}
```

3. Write Following code in WelcomeServlet class.

```
public class WelcomeServlet extends HttpServlet
{
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
    {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      HttpSession session = request.getSession();
      String user = (String)session.getAttribute("user");
      out.println("Hello "+user);
      out.close();
    }
}
```

Output:



6.6.4 Cookies

- The second technique that you can use to manage user sessions is by using cookies. A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- Eventhough a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.
- The cookies sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

How Cookie Works?

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser.
- After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

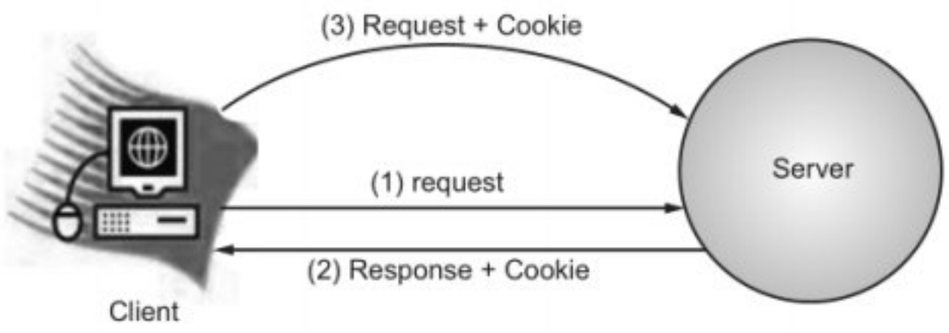


Fig. 6.18: Working of Cookies

Types of Cookies in Servlets:

1. **Non-persistent Cookie:** It is valid for single session only. It is removed each time when user closes the browser.
2. **Persistent Cookie:** It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or signout.

- In servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package.
- You can create a cookie by calling the Cookie class constructor and passing two String objects i.e., the name and value of the cookie.
- For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":

```
Cookie c1 = new Cookie("userName", "Meenakshi");
```

- You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

```
response.addCookie(c1);
```

Advantages of Cookies:

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantages of Cookies:

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Program 6.19: Sending and receiving non-persistent cookies.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class SimpleServlet extends HttpServlet {
  /*Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
    Cookie c1 = new Cookie("userName", "Meenakshi");
    Cookie c2 = new Cookie("password", "abc");
    response.addCookie(c1);
    response.addCookie(c2);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("Please click the button to see the cookies sent to you.<br> ");
    out.println("<form method=POST><input type=Submit value=Submit></form>");
  }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<H2> List of All cookies.</H2>");
    Cookie[] cookies = request.getCookies();
    int length = cookies.length;
    for (int i = 0; i < length; i++) {
      Cookie cookie = cookies[i];
      out.println("<B>Cookie Name:</B> " + cookie.getName() + "<BR>");
      out.println("<B>Cookie Value:</B> " + cookie.getValue() + "<BR>");
    }
  }
}
```

Output:

Please click the button to see the cookies sent to you.
Submit

List of All cookies.

Cookie Name: JSESSIONID
Cookie Value: ac31a00de090244009e8b52a6242
Cookie Name: userName
Cookie Value: Meenakshi
Cookie Name: password
Cookie Value: abc
Cookie Name: _xsrf
Cookie Value:
2|3959bd4e|c1161164eb693b9e1988c9231262250f|1553664571

Persisting Cookies:

- The cookies you created in the previous last as long as the browser is open. When the browser is closed, the cookies are deleted.
- You can choose to persist cookies so that they last longer.
- The javax.servlet.http.Cookie class has the setMaxAge() method that sets the maximum age of the cookie in seconds.

Additional Programs:

Program 1: Servlet to find factorial of number.

- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

```
<html>
  <body>
    <form name = "Form1" action = "FactServlet" method =GET>
      <input type = text name ="number">
      <input type = submit value = "SUBMIT">
    </form>
  </body>
</html>
```

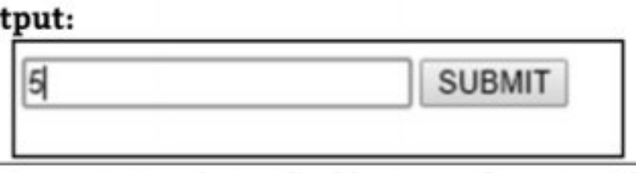
- The java source code that handles get request is as follows:

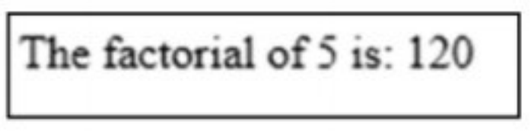
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class FactServlet extends HttpServlet {

public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException, IOException {
    int fact = 1;
    String F1 = req.getParameter("number");
    int num = Integer.parseInt(F1);
    System.out.println(num);
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    for (int i = num; i > 1; --i) {
      fact = fact * i;
    }
    out.println("The factorial of "+num+" is:");
    out.println(fact);
    out.close();
  }
}
```

Output:





Program 2: Servlet to find largest of two number.

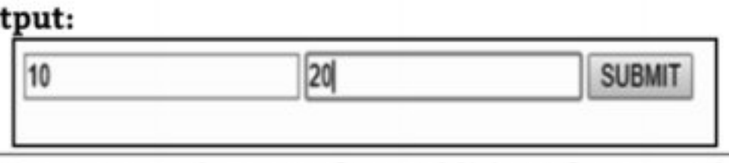
- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

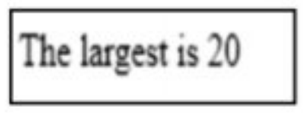
```
<html>
  <body>
    <form name = "Form1" action = "LargeServlet" method =GET>
      <input type = text name =”first”>
      <input type = text name =”second”>
      <input type = submit value = "SUBMIT">
    </form>
  </body>
</html>
```

- The java source code that handles get request is as follows:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class LargeServlet extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    String F1 = req.getParameter("first");
    int num = Integer.parseInt(F1);
    String F2 = req.getParameter("second");
    int num1 = Integer.parseInt(F2);
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    if (num < num1)
      out.println("The largest is "+num1);
   else
      out.println("The largest is "+num);
   out.close();
  }
}
```

Output:





Program 3: Servlet to perform addition of two number.

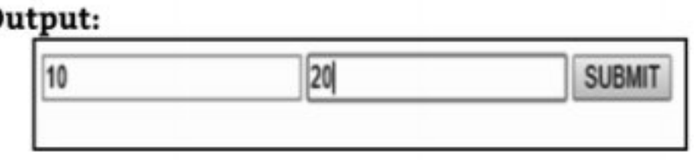
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

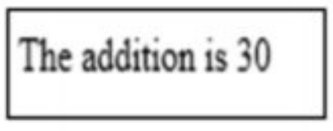
```
<html>
  <body>
    <form name = "Form1" action = "AddServlet" method = GET>
      <input type = text name = "first">
      <input type = text name = "second">
      <input type = submit value = "SUBMIT">
    </form>
  </body>
</html>
```

- The java source code that handles get request is as follows:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class AddServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    String F1 = req.getParameter("first");
    int num = Integer.parseInt(F1);
    String F2 = req.getParameter("second");
   int num1 = Integer.parseInt(F2);
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("The addition is "+(num1+num));
   out.close();
  }
}
```

Output:





Practice Questions

1. What is Servlet?
2. Explain GenericServlet in detail?
3. Explain Servlet interface with example?
4. Describe HttpServlet in detail.
5. How the servlet works? Explain diagrammatically.
6. Define the terms: (i) Cookies, (ii) Session.
7. Explain various mechanisms for Session Management or Session Tracking?
8. Explain HttpSession class?
9. Explain cookie? Explain persistent cookies with example.
10. Write any three methods of HttpSession class.
11. Write a Servlet program which will display Login Form and verify the login.
12. Write a Servlet program that will display the total number of hit to user.
13. Explain the advantages of Servlet technology over other technology.
14. Explain the benefits of Servlet.
15. Write a Servlet program which will take a number from user and display the factorial of that on client side.
16. Write a Servlet program that will take a number from the user and display the table of that number to user.
17. Write a Servlet program that will take two number from the user and display the addition, subtraction, multiplication of that number to user.
18. With the help of diagram describe servlet life cycle.
19. Explain the life cycle method of Servlet.
20. Write a Servlet program that will display the details of Student table using JDBC in tabular form on user side?
21. Compare CGI and Servlet.
22. Write a Servlet program that will take data of student and store it in of Student table using JDBC?
23. Write a Servlet program that will search particular data in Student table using JDBC, by taking some key from user.
24. Write a Servlet for demonstrating the generic Servlet class.
25. Write a Servlet for demonstrating the generic HTTP servlet class.
26. Write a Servlet to demonstrate the Http Servlet class using doGet().
27. Write a Servlet to demonstrate the Http Servlet class using doPost().
28. Write a Servlet to demonstrate the cookie.
29. Explain the term session tracking in detail.

■■■

@SumitBiranje