

SUBJECT CODE : 304195(C)

As per Revised Syllabus of

SAVITRIBAI PHULE PUNE UNIVERSITY

Choice Based Credit System (CBCS)
T.E. (E & Tc) Semester - VI (Elective - II)

ADVANCED JAVA PROGRAMMING

Mrs. Anuradha A. Puntambekar

M.E. (Computer)
Formerly Assistant Professor in
P.E.S. Modern College of Engineering,
Pune.

Santosh B. Dhekale

Ph.D. (Pursuing),
M.E. (E&TC) VLSI & Embedded System, B.E.(Electronics)
Assistant Professor,
AISSMS College of Engineering,
Pune



ADVANCED JAVA PROGRAMMING

Subject Code : 304195(C)

T.E. (E & Tc) Semester - VI (Elective - II)

© Copyright with A. A. Puntambekar

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,

Pune - 411030, M.S. INDIA Ph.: +91-020-24495496/97

Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders

Sr. No. 10/1A,

Ghule Industrial Estate, Nanded Village Road,

Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-5585-013-3



9 789355 850133

SPPU 19

PREFACE

The importance of **Advanced JAVA Programming** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Advanced JAVA Programming**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement was provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors
Mrs. A. A. Puntambekar
Santosh B. Dhekate

Dedicated to God.

SYLLABUS

Advanced JAVA Programming - 304195(C)

Credit	Examination Scheme :
03	In-Sem (Theory) : 30 Marks End-Sem (Theory) : 70 Marks

Unit I Applet

Applet Basics - Introduction, limitations of AWT, Applet architecture - HTML APPLET tag - Passing parameter to Appletget, DocumentBase() and getCodeBase(), Japplet : Icons and Labels Text Fields Buttons, Combo Boxes , Checkboxes, Tabbed Panes, Scroll Panes, Trees : Tables. **(Chapter - 1)**

Unit II Event Handling using AWT / Swing Components

Event Handling : Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes, inner classes. The AWT class hierarchy, user interface components - labels, button, canvas, scrollbars, text components, checkbox, checkbox groups, choices, lists panels - scroll pane, dialogs, menu bar, graphics, layout manager - layout manager types - boarder, grid, flow, card and grib bag.

(Chapter - 2)

Unit III GUI Programming

Designing Graphical User Interfaces in Java, Components and Containers, Basics of Components, Using Containers, Layout Managers, AWT Components, Adding a Menu to Window, Extending GUI Features Using Swing Components, Java Utilities (java.util Package) The Collection Framework : Collections of Objects, Collection Types, Sets, Sequence, Map, Understanding Hashing, and Use of Array List and Vector. **(Chapter - 3)**

Unit IV Database Programming using JDBC

The Concept of JDBC, JDBC Driver Types and Architecture, JDBC Packages, A Brief Overview of the JDBC process, Database Connection, Connecting to non-conventional Databases Java Data Based Client/server, Basic JDBC program Concept, Statement, Result Set, Prepared Statement, Callable Statement, Executing SQL commands, Executing queries.

(Chapter - 4)

Unit V Remote Method Invocation (RMI)

Remote Method Invocation : Architecture, RMI registry, the RMI Programming Model; Interfaces and Implementations; Writing distributed application with RMI, Naming services,

Naming and Directory Services, Setting up Remote Method Invocation - RMI with Applets, Remote Object Activation; The Roles of Client and Server, Simple Client/Server Application using RMI. **(Chapter - 5)**

Unit VI Networking

The java.net package, Connection oriented transmission - Stream Socket Class, creating a Socket to a remote host on a port (creating TCP client and server), Simple Socket Program Example. InetAddress, Factory Methods, Instance Methods, Inet4Address and Inet6Address, TCP/IP Client Sockets. URL, URLConnection, HttpURLConnection, The URI Class, Cookies, TCP/IP Server Sockets, Datagrams, DatagramSocket, DatagramPacket, A Datagram Example. Connecting to a Server, Implementing Servers, Sending EMail, Servlet overview - the Java web server - The Life Cycle of a Servlet, your first servlet.

(Chapter - 6)

TABLE OF CONTENTS

Unit I

Chapter - 1	Applet	(1 - 1) to (1 - 22)
1.1	Introduction	1 - 2
1.2	Applet Architecture.....	1 - 3
1.3	HTML APPLETTAG	1 - 5
1.4	Creating and Executing Applet	1 - 6
1.4.1	Creating an Applet.....	1 - 6
1.4.2	Executing an Applet.....	1 - 8
1.4.2.1	Adding Applet to HTML File.....	1 - 8
1.4.2.2	Embedding Applet Code in Java	1 - 9
1.5	Passing Parameter to Applet	1 - 11
1.6	getDocumentBase() and getCodeBase().....	1 - 15
1.7	Adding Controls to Applet	1 - 16
1.7.1	Buttons	1 - 16
1.7.2	Text Fields.....	1 - 17
1.7.3	Combo Boxes.....	1 - 18
1.7.4	Checkboxes.....	1 - 19
1.8	Multiple Choice Questions with Answers.....	1 - 20

Unit II

Chapter - 2	Event Handling using AWT / Swing Components	(2 - 1) to (2 - 72)
--------------------	--	----------------------------

Part I : Event Handling

2.1	Events.....	2 - 2
2.2	Event Sources.....	2 - 2

2.3	Event Classes.....	2 - 2
2.3.1	ActionEvent Class.....	2 - 3
2.3.2	ItemEvent Class	2 - 3
2.3.3	KeyEvent Class	2 - 4
2.3.4	MouseEvent Class.....	2 - 4
2.3.5	TextEvent Class	2 - 5
2.3.6	WindowEvent Class	2 - 5
2.4	Event Listeners.....	2 - 6
2.4.1	ActionListener Interface	2 - 6
2.4.2	ItemListener Interface	2 - 6
2.4.3	KeyListener Interface.....	2 - 7
2.4.4	MouseListener Interface.....	2 - 7
2.4.5	MouseMotion Interface.....	2 - 7
2.4.6	TextListener Interface.....	2 - 7
2.4.7	WindowListener Interface	2 - 7
2.5	Delegation Event Model	2 - 8
2.6	Handling Mouse Events	2 - 10
2.7	Handling Keyboard Events	2 - 13
2.8	Adapter Classes.....	2 - 15
2.9	Inner Classes	2 - 18
2.9.1	Static Member Classes.....	2 - 18
2.9.2	Member Classes.....	2 - 19
2.9.3	Local Classes	2 - 20
2.9.4	Anonymous Classes	2 - 21
Part II : AWT		
2.10	What is Abstract Windowing Toolkit ?	2 - 21
2.11	The AWT Class Hierarchy	2 - 22
2.12	Limitations of AWT	2 - 22
2.13	User Interface Components.....	2 - 23

2.13.1 Labels.....	2 - 23
2.13.2 Buttons	2 - 24
2.13.3 Canvas.....	2 - 25
2.13.4 Scrollbars	2 - 27
2.13.5 Text Components.....	2 - 28
2.13.6 Checkbox	2 - 29
2.13.7 Checkbox Group	2 - 30
2.13.8 Choices.....	2 - 31
2.13.9 List Panels	2 - 32
2.14 Dialogs.....	2 - 33
2.14.1 File Dialog	2 - 35
2.15 Menu bar	2 - 37
2.16 Programming Examples based on AWT Components and Event Handling.....	2 - 41
2.17 Graphics	2 - 46
2.17.1 Lines.....	2 - 47
2.17.2 Rectangle	2 - 48
2.17.3 Oval.....	2 - 49
2.17.4 Arc.....	2 - 51
2.17.5 Polygons.....	2 - 52
2.18 Layout Manager	2 - 54
2.18.1 FlowLayout	2 - 54
2.18.2 BorderLayout	2 - 57
2.18.3 GridLayout	2 - 61
2.18.4 CardLayout.....	2 - 63
2.18.5 GridBagLayout	2 - 67
2.19 Multiple Choice Questions with Answers.....	2 - 70

Unit III

Chapter - 3	GUI Programming	(3 - 1) to (3 - 80)
3.1	Designing Graphical User Interfaces in Java	3 - 2
3.1.1	Difference between AWT and Swing.....	3 - 2
3.2	Components and Containers	3 - 2
3.3	Basics of Components.....	3 - 3
3.4	Extending GUI Features Using Swing Components	3 - 3
3.4.1	JApplet	3 - 3
3.4.2	Creating Frames.....	3 - 6
3.4.3	Label and ImageIcon.....	3 - 8
3.4.4	TextField	3 - 9
3.4.5	TextArea	3 - 10
3.4.6	Buttons	3 - 12
3.4.7	Checkboxes.....	3 - 15
3.4.8	Radio Buttons	3 - 16
3.4.9	Lists	3 - 18
3.4.10	Choices.....	3 - 19
3.4.11	ScrollPane	3 - 25
3.4.12	Scrollbar	3 - 28
3.4.13	Menus.....	3 - 29
3.4.14	Dialog Boxes	3 - 37
3.4.15	Tabbed Pane	3 - 42
3.4.16	JTree	3 - 45
3.4.17	JTable.....	3 - 47
3.5	Java Utilities (java.util Package).....	3 - 49
3.6	The Collection Framework.....	3 - 49
3.7	Collections of Objects and Types.....	3 - 51
3.8	List Interface	3 - 54

3.8.1 ArrayList.....	3 - 54
3.8.2 LinkedList.....	3 - 56
3.9 Vector.....	3 - 61
3.10 Set Interface.....	3 - 64
3.10.1 HashSet.....	3 - 65
3.10.2 TreeSet.....	3 - 66
3.11 Map Interface	3 - 68
3.11.1 Hashtable.....	3 - 68
3.11.2 HashMap.....	3 - 69
3.11.3 TreeMap	3 - 70
3.12 Multiple Choice Questions with Answers.....	3 - 71

Unit IV

Chapter - 4 Database Programming using JDBC (4 - 1) to (4 - 38)

4.1 The Concept of JDBC.....	4 - 2
4.2 Types of JDBC Drivers	4 - 2
4.3 JDBC Architecture	4 - 6
4.3.1 Two Tier Model.....	4 - 6
4.3.2 Three Tier Model	4 - 6
4.4 JDBC Packages.....	4 - 6
4.5 A Brief Overview of the JDBC Process	4 - 8
4.6 Executing SQL Commands	4 - 8
4.7 Database Connection.....	4 - 14
4.8 Basic JDBC Program Concept	4 - 15
4.9 Executing Queries	4 - 16
4.9.1 CREATE Statement.....	4 - 17
4.9.2 SELECT Statement.....	4 - 18
4.9.3 UPDATE Statement.....	4 - 20
4.10 Statement	4 - 24
4.10.1 Prepared Statement	4 - 25

4.11 Result Set	4 - 30
4.11.1 Navigating Methods	4 - 31
4.11.2 Reading the Result using ResultSet	4 - 31
4.11.3 Updating ResultSets.....	4 - 32
4.12 Multiple Choice Questions with Answers.....	4 - 32

Unit V

Chapter - 5	Remote Method Invocation (RMI)	(5 - 1) to (5 - 30)
5.1 Remote Method Invocation.....		5 - 2
5.2 Architecture		5 - 2
5.3 RMI Registry		5 - 4
5.4 The RMI Programming Model.....		5 - 5
5.5 Interfaces and Implementations.....		5 - 5
5.6 Writing Distributed Application with RMI		5 - 6
5.7 Naming Services.....		5 - 6
5.8 Naming and Directory Services.....		5 - 7
5.9 RMI with Applets		5 - 7
5.10 Remote Object Activation		5 - 10
5.11 The Roles of Client and Server		5 - 11
5.12 Simple Client / Server Application using RMI.....		5 - 12
5.13 Multiple Choice Questions with Answers		5 - 29

Unit VI

Chapter - 6	Networking	(6 - 1) to (6 - 62)
6.1 The java.net Package		6 - 2
6.1.1 The Networking Classes and Interfaces.....		6 - 2
6.2 Socket Class.....		6 - 3
6.2.1 Client Server		6 - 4
6.2.2 Reserved Sockets.....		6 - 5
6.2.3 Proxy Servers		6 - 6

6.2.4 Internet Addressing	6 - 6
6.3 InetAddress	6 - 7
6.3.1 Factory Methods.....	6 - 7
6.3.2 Instance Method	6 - 12
6.3.3 Inet4Address and Inet6 Address	6 - 12
6.4 URL.....	6 - 13
6.4.1 Format	6 - 13
6.4.2 The URL Path	6 - 13
6.4.3 The URL Class.....	6 - 14
6.5 URLConnection	6 - 15
6.6 HttpURLConnection	6 - 18
6.7 The URI Class.....	6 - 19
6.8 Cookies.....	6 - 20
6.9 TCP,IP and UDP	6 - 21
6.10 TCP/IP Client Sockets	6 - 23
6.11 TCP/IP Server Sockets	6 - 25
6.12 Datagrams	6 - 35
6.12.1 Datagram Packet	6 - 35
6.12.2 Datagram Server and Client.....	6 - 36
6.13 Sending Email	6 - 43
6.14 Servlet Overview	6 - 45
6.14.1 The Java Web Server	6 - 45
6.14.2 Advantages of using Servlets.....	6 - 46
6.14.3 The Life Cycle of a Servlet.....	6 - 46
6.14.4 Your First Servlet	6 - 47
6.15 Handling HTTP Requests and Response.....	6 - 52
6.16 Multiple Choice Questions with Answers	6 - 57

UNIT I

1

Applet

Syllabus

Applet Basics - Introduction, limitations of AWT, Applet architecture - HTML APPLET tag - Passing parameter to Appletget, DocumentBase() and getCodeBase() , Japplet : Icons and Labels Text Fields Buttons, Combo Boxes, Checkboxes, Tabbed Panes, Scroll Panes, Trees : Tables.

Contents

- 1.1 Introduction
- 1.2 Applet Architecture
- 1.3 HTML APPLET Tag
- 1.4 Creating and Executing Applet
- 1.5 Passing Parameter to Applet
- 1.6 `getDocumentBase()` and `getCodeBase()`
- 1.7 Adding Controls to Applet
- 1.8 Multiple Choice Questions

1.1 Introduction

- Applets are the small Java programs that can be used in internetworking environment.
- These programs can be transferred over the internet from one computer to another and can be displayed on various web browsers.
- Various applications of applets are in performing arithmetic operations, displaying graphics, playing sounds, creating animation and so on.
- Following are the **situations in which we need to use applet** -
 1. For displaying the dynamic web pages we need an applet. The dynamic web page is a kind of web page on which the contents are constantly changing. For example an applet that can represent the sorting process of some numbers. During the process of sorting the positions of all the elements is changing continuously.
 2. If we want some special effects such as sound, animation and much more then the applets are used.
 3. If we want a particular application should be used by any user who might be located remotely. Then in such situation the applets are embedded into the web pages and can be transferred over the internet.

Difference between Applet and Application

Sr. No.	Applets	Applications
1.	Applets do not have main method . On loading of applets some methods of applet class get called automatically.	Application programs have main method . Within the main method the call to another methods of Java class is given.
2.	Applets cannot run independently . They can be either embedded in web page or can be run using appletviewer.	Applications program run independently .
3.	Applets cannot be read from file . Similarly applets can not write to files.	Application programs make use of I/O functions and can read a file or write to file.
4.	Applets cannot communicate with others on the network.	Java programs can communicate with other programs in distributed environment.
5.	Applets cannot execute any program on local computer.	Applications can execute a program on local computer.

Review Question

1. Differentiate applet and application with any four points.

1.2 Applet Architecture

- There are various methods which are typically used in applet for initialization and termination purpose. These methods are :
 1. Initialization
 2. Running state
 3. Idle state
 4. Dead or destroyed state (Refer Fig. 1.2.1 for applet's life cycle)

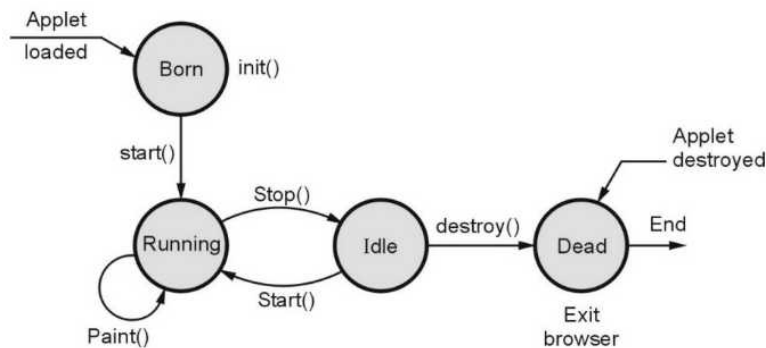


Fig. 1.2.1 Applet's life cycle

- When applet begins, the AWT calls following methods in sequence -
 - a) `init()`
 - b) `start()`
 - c) `paint()`
- When applet is terminated following method are invoked in sequence.
 - a) `stop()`
 - b) `destroy()`

1. Initialization state

- When applet gets loaded it enters in the initialization state. For this purpose the **`init()`** method is used. In this method you can initialize the required variables. This method is called only once initially at the execution of the program. The syntax can be,

```

public void init()
{
  //initialization of variables
}
  
```

- In this method various tasks of initialization can be performed such as -
 1. Creation of objects needed by applet
 2. Setting up of initial values
 3. Loading of image
 4. Setting up of colors.

2. Running state

- When the applet enters in the running state, it invokes the **start()** method of Applet class.
- This method is called only after the init method. After stopping the applet when we restart the applet at that time also this method is invoked. The syntax can be

```
public void start()  
{  
...  
}
```

3. Display state

- Applet enters in the display state when it wants to display some output. This may happen when applet enters in the running state. The **paint()** method is for displaying or drawing the contents on the screen. The syntax is

```
public void paint(Graphics g)  
{  
...  
}
```

- An instance of Graphics class has to be passed to this function as an argument. In this method various operations such as display of text, circle, line are invoked.

4. Idle state

- This is an idle state in which applet becomes idle. The **stop()** method is invoked when we want to stop the applet. When an applet is running if we go to another page then this method is invoked. The syntax is

```
public void stop()  
{  
...  
}
```


5. Dead state

- When applet is said to be dead then it is removed from memory. The method **destroy()** is invoked when we want to terminate applet completely and want to remove it from the memory.

```
public void destroy()
{
...
}
```

- It is a good practice to call stop prior to destroy method.

Review Questions

1. Describe following states of applet life cycle :
 - a) Initialization state.
 - b) Running state.
 - c) Display state.
2. Explain applet life cycle with suitable diagram.

1.3 HTML APPLETTAG

- The <APPLET> tag can be specified with the help of various attributes. These attributes help to integrate the applet into overall design of the web page.
- Various attributes of APPLETTAG are -

Attribute	Description
CODE = appletfilename	The specified applet can be loaded in the web page. This attribute must be specified in order to embed the applet in the HTML file.
WIDTH = pixels HEIGHT = pixels	This attribute specifies the width and height of the applet.
ALIGN = alignment	This attribute is for specifying the alignment. Various alignments are - TOP, BOTTOM, LEFT, RIGHT, MIDDLE, ABSMIDDLE, ABSBOTTOM, TEXTTOP and BASELINE. This is an optional attribute.
CODEBASE = codebase_URL	It specifies the name of the directory in which the applet is stored. This attribute is required when the attribute is not there in the current working directory. This is an optional attribute.

HSPACE = pixels	When ALIGN is either LEFT or RIGHT this attribute is used. It specifies the amount of horizontal blank spaces the browser should leave around the applet. This is an optional attribute.
VSPACE = pixels	When ALIGN is either TOP or BOTTOM this attribute is used. It specifies the amount of vertical blank spaces the browser should leave around the applet. This is an optional attribute.
ALT = alternate text	The non Java browser can display the alternate text in place of applet. This is an optional attribute.

Review Questions

1. Explain all attributes available in < applet > tag.
2. Explain <applet> tag with its major attributes only.
3. Describe the following attributes of applet :
(i) Codebase (ii) Alt (iii) Width (iv) Code
4. Explain any four applet tag.

1.4 Creating and Executing Applet

1.4.1 Creating an Applet

- Normally the applet code makes use of two classes - **Applet** and **Graphics**.
- For the **Applet** class the package **java.applet** is required. This class provides the applet's life cycle method such as **init()**, **start()** and **paint()**.
- The **Graphics** class is supported by the package **java.awt**.
- Here is a simple applet.

Example Program

```

/*
This is my First Applet program
*/
import java.awt.*;
import java.applet.*;
public class FirstApplet extends Applet
{
public void paint(Graphics g)
{
g.drawString("This is my First Applet",50,30);
}
}

```

Program explanation :

- In above given small applet program, the main intension is to display the message “This is my First Applet”. Let us start from the beginning of the program -
- The first three lines represent a comment statement.
- Then next comes

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

- We always need these two packages to be imported in the program. The **java.awt** package consists of **java awt** classes. Here **awt** stands for **abstract window toolkit**. The **AWT** provides the support for window based graphical interface such as for drawing screen, windows, buttons, text boxes, menus and so on. The other imported package is **java.applet**. This is essential because we need to use **Applet** class in our applet program which is included in **java.applet** package.
- The functionalities that are required to run applet inside the web browser are supported by **java.applet**.
- Then comes

```
public class FirstApplet extends Applet
```

- The class *FirstApplet* is a subclass of class *Applet*. Hence the keyword *extends* is used. Java requires that your applet subclass (here it is *FirstApplet*) should be declared as public. Hence is the declaration !
- We have then defined a method

```
public void paint(Graphics g)
```

- This method is used to paint something on the screen and it can be text, line, circle, rectangle or anything. Thus **paint** is a method which provides actual appearance on the screen. And this method requires a parameter to be passed as an object of class **graphics**. Therefore an object **g** of class **Graphics** is passed.
- Using this object the method **drawString** of *Graphics* class is invoked. [Note that *Graphics* class is a part of *java.awt* package].

```
g.drawString("This is my First Applet",50,30);
```

- To method **drawString**, firstly we have passed a string which we want to be displayed, then 50 and 30 represents the position of the string on the screen i.e. x and y positions respectively.

1.4.2 Executing an Applet

- There are two methods of executing an applet program -
 1. Adding Applet to HTML File
 2. Using Appletviewer tool
- Let us discuss these methods in detail.

1.4.2.1 Adding Applet to HTML File

- Following are the steps that need to be followed for adding applet to HTML file.

Step 1 : Write a Java applet program in a notepad. The code is as follows :

Java Program[FirstApplet.java]

```
/*
This is my First Applet program
*/
import java.awt.*;
import java.applet.*;
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("This is my First Applet",50,30);
    }
}
```

Step 2 : Compile your applet source program using **javac** compiler, i.e.

```
D:\test>javac FirstApplet.java
```

Step 3 : Write following code in notepad/wordpad and save it with filename and extension **.html**. For example following code is saved as

Exe_FirstApplet.html, The code is

```
<html>
<body>
    <applet code="FirstApplet" width=300 height=100>

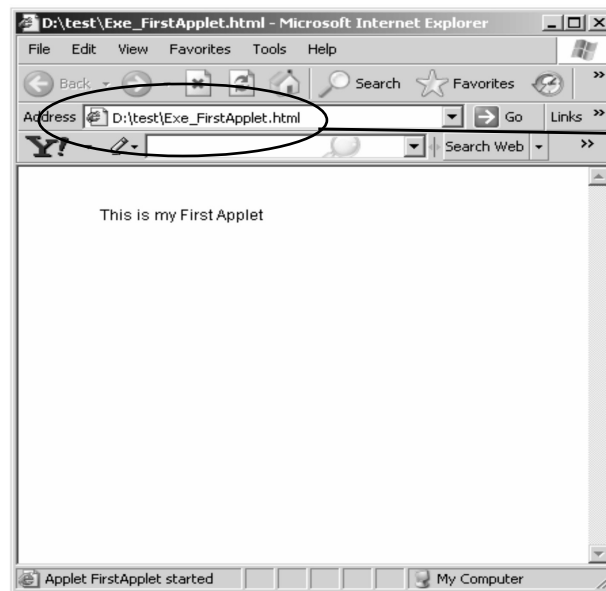
    </applet>
</body>
</html>
```

- For embedding the applet in the HTML document the **<APPLET>** tag is used. The **<APPLET>** tag supplies the name of the applet to be loaded. Suppose we want to load the applet **FirstAppletprogram** in the web page then the applet tag will be,

```
<APPLET CODE= FirstAppletprogram.class
  WIDTH = 500
  HEIGHT = 300
</APPLET>
```

- The HTML code tells the web browser to load the compiled code of applet FirstAppletprogram.class. Note that this class file must be present in the same directory where the web page is stored.

Step 4 : Load html file with some web browser, This will cause to execute your html file. It will look like this –



Note this
URL

1.4.2.2 Embedding Applet Code in Java

- We can embed the applet code in Java using comment statements. It is as shown in following illustration.

Step 1 : Write Java Program as given below.

```
/*
This is my First Applet program
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="FirstApplet" width=300 height=100>
</applet>
*/
public class FirstApplet extends Applet
```

Embedded Applet in
Java

```
{
    public void paint(Graphics g)

    {
        g.drawString("This is my First Applet",50,30);
    }
}
```

- In above code we have added applet code at the beginning of the program.

```
<applet code="FirstApplet" width=300 height=100>
</applet>
```

- This will help to understand Java that the source program is an applet with the name FirstApplet.

Step 2 : By this edition you can run your applet program merely by **Appletviewer** command.

```
D:\test>javac FirstApplet.java
D:\test>Appletviewer FirstApplet.java
```

And we will get



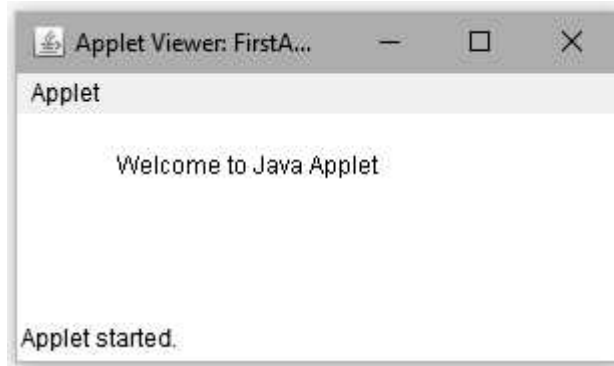
Example 1.4.1 *Define applet. Write a program to create an applet to display message "Welcome to java applet".*

Solution : FirstApplet.java

```
/*
This is my First Applet program
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="FirstApplet" width=300 height=100>
</applet>
*/
public class FirstApplet extends Applet
```

```
{
public void paint(Graphics g)
{
g.drawString("Welcome to Java Applet",50,30);
}
}
```

Output



1.5 Passing Parameter to Applet

- Parameters are passed to applets in NAME=VALUE pairs in <PARAM> tags between the opening and closing APPLET tags.
- There can be any number of <PRAM> tags inside APPLET tag.
- Inside the applet, you read the values passed through the PARAM tags with the **getParameter()** method of the **Applet** class.
- The program below demonstrates this idea -

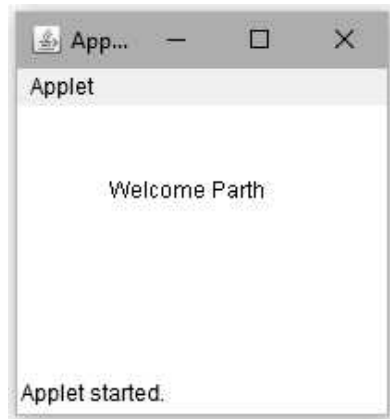
Example 1.5.1 *How can parameters be passed to an applet ? Write an applet to accept user name in the form of parameter and print 'Hello < username >'.*

Solution :

```
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="WelcomeParam" width=200 height=200>
<param name="Username" value="Parth">
</applet>
*/
public class WelcomeParam extends Applet
{
String msg="";
public void init()
{
```

```
    msg = getParameter("Username");
    msg = "Welcome " + msg;
}
public void paint(Graphics g)
{
    g.drawString(msg,50,50);
}
}
```

Output



Example 1.5.2 Write an applet program that accepts two input, strings using `<Param>` tag and concatenate the strings and display it in status window.

Solution : TwoStrings.java

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

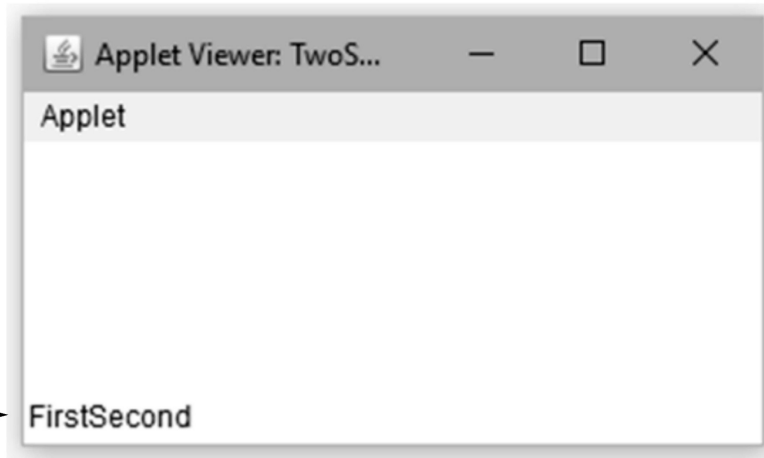
/*
<APPLET code="TwoStrings" width="300" height="100">
<PARAM name="str1" value="First">
<PARAM name="str2" value="Second">
</APPLET>
*/
public class TwoStrings extends Applet
{
    public void paint(Graphics g)
    {
        String s1 = this.getParameter("str1");
        String s2 = this.getParameter("str2");
        String s3;
```



```
s3=s1+s2;
showStatus(s3);
}
}
```

Output

Note that the strings are displayed here



Example 1.5.3 Design an applet which accepts username as a parameter for html page and display number of characters from it.

Solution : Step 1 : Create HTML page as follows -

test.html

```
<html>
<head>
<title> Parameter Demo </title>
</head>
<applet code = "LengthDemo.class" width =300 height =300>
<PARAM name="uname" value="Chitra">
</applet>
</html>
```

Step 2 : Create Java program for the applet class as follows -

LengthDemo.java

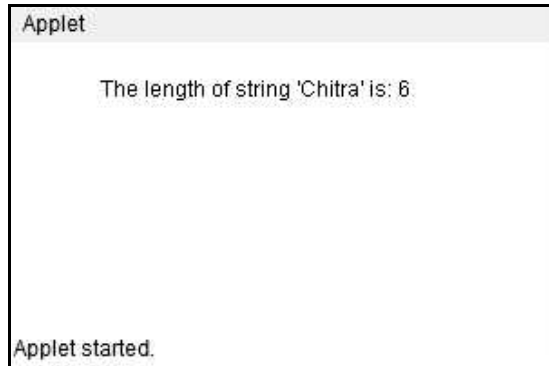
```
import java.applet.*;
import java.awt.*;
public class LengthDemo extends Applet
{
    public void paint(Graphics g)
    {
        String str = this.getParameter("uname");
        int len;
        len=str.length();
    }
}
```

```
String msg="The length of string 'Chitra' is: "+len;
g.drawString(msg,50,30);
}
}
```

Step 3 : Compile the program created in step 2 using following command on command prompt

```
D:\>javac LengthDemo.java
```

Step 4 : Open the web browser and type the name of the html file created in step 1. The output will be -

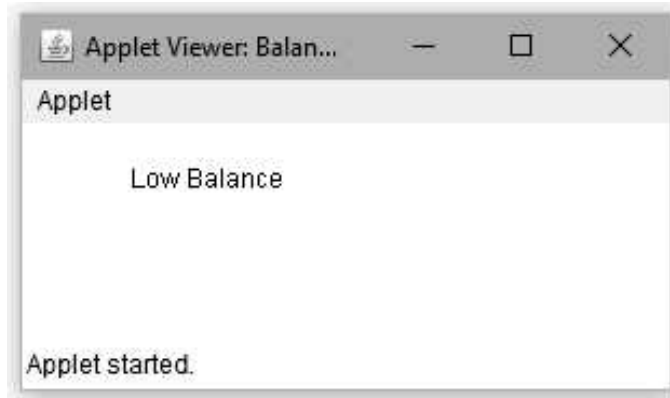


Example 1.5.4 How to pass parameter to an applet ? Write an applet to accept Account No and balance in form of parameter and print message "low balance" if the balance is less than 500.

Solution :

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

/*
<APPLET code="BalanceChk" width="300" height="100">
<PARAM name="AccNo" value=1001>
<PARAM name="Balance" value=300>
</APPLET>
*/
public class BalanceChk extends Applet
{
    public void paint(Graphics g)
    {
        int balance = Integer.parseInt(this.getParameter("Balance"));
        if(balance<500)
            g.drawString("Low Balance",50,30);
        else
            g.drawString("Sufficient Balance",50,30);
    }
}
```

Output**Review Questions**

1. Explain <PARAM> Tag of applet with suitable example.
2. Explain following methods for applet with an example :
 - 1) Passing parameter to applet
 - 2) Embedding <applet> tags in java code.

1.6 **getDocumentBase() and getCodeBase()**

- The **getDocumentBase()** and **getCodeBase()** are the two methods that return the names of the directories. The **getDocumentBase()** is the name of the directory which contains the HTML file that starts the applet. The **getCodeBase()** is the name of the directory that contains the class file of the applet is loaded.
 - **URL getCodeBase()** : Gets the base URL.
 - **URL getDocumentBase()** : Gets the URL of the document in which the apple is embedded.
- Following applet program shows the demonstration of these two directories.

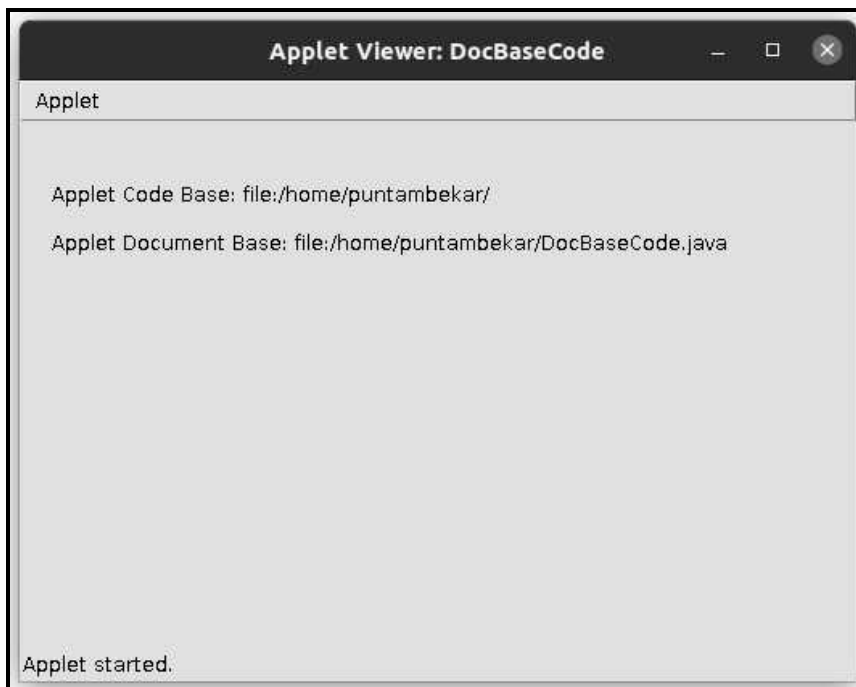
Java Program

```
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code="DocBaseCode" width=500 height=300>
</applet>
*/
public class DocBaseCode extends Applet
{
    public void paint(Graphics g)
    {
```

```
String msg;
//getCodeBase Method
URL appletCodeDir = getCodeBase();
msg = "Applet Code Base: "+appletCodeDir.toString();
g.drawString(msg, 20, 50);

//getDocumentBase Method
URL appletDocDir = getDocumentBase();
msg = "Applet Document Base: "+appletDocDir.toString();
g.drawString(msg, 20, 80);
}
}
```

Output



1.7 Adding Controls to Applet

- We can add various controls such as textbox, Push Button, Radio button, Check box to the applet. Let us understand this with illustrative examples :

1.7.1 Buttons

- The **Button** class is used to add the labeled button to the applet. The caption of the button is passed as a parameter to it. The syntax is

```
Button button_object=new Button(caption string)
```

- Using **add** method we can add the button control on the applet window. Following program shows how to add button to the applet.

Example Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Button1" width=350 height=200>
</applet>
*/
public class Button1 extends Applet
{
    Button button=new Button("Click Me");
    public void init()
    {
        add(button);
    }
}
```

Output



1.7.2 Text Fields

- The **TextField** class allows to add the single line text. Following program shows how to add a textfield control to the applet.

Example Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="TextFieldDemo" width=350 height=200>
</applet>
*/
```

```
public class TextFieldDemo extends Applet
{
    TextField tf=new TextField("Hello, How are you?");
    public void init()
    {
        add(tf);
    }
}
```

Output



1.7.3 Combo Boxes

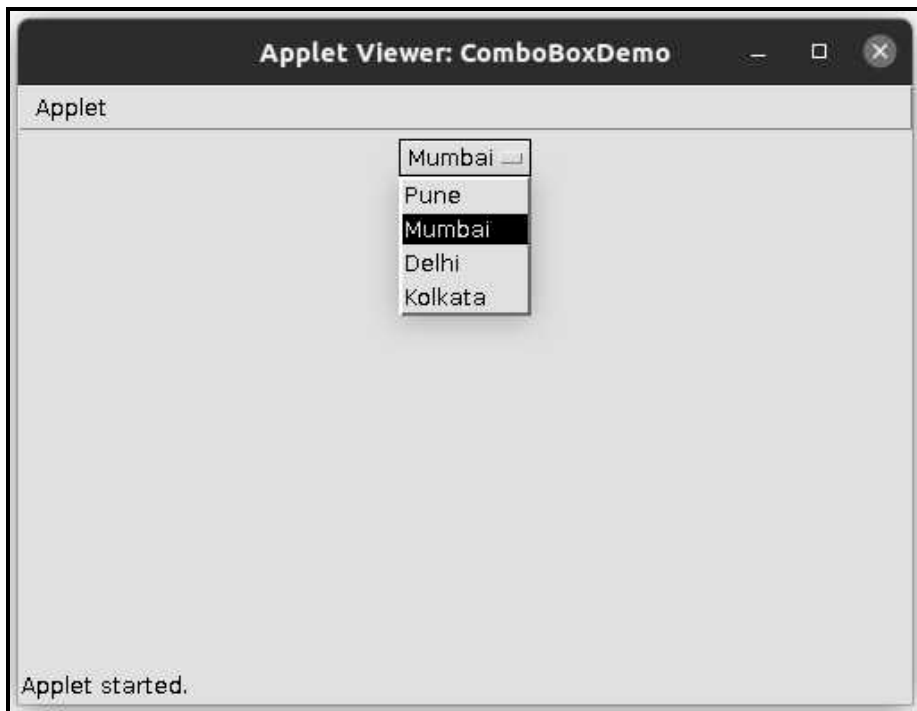
- Combobox is a dropdown list. It forces the user to select only one element from the list. It is easier to change the element of the list. It is basically a Choice control. Hence it is created using **Choice()** class.
- Following program demonstrates the use of combox control.

Java Program[ComboBoxDemo.java]

```
import java.applet.*;
import java.awt.*;
/*
<applet code="ComboBoxDemo" width=350 height=200>
</applet>
*/
public class ComboBoxDemo extends Applet
{
    Choice city=new Choice();
    public void init()
    {
```

```
city.add("Pune");
city.add("Mumbai");
city.add("Delhi");
city.add("Kolkata");
add(city);
}
}
```

Output



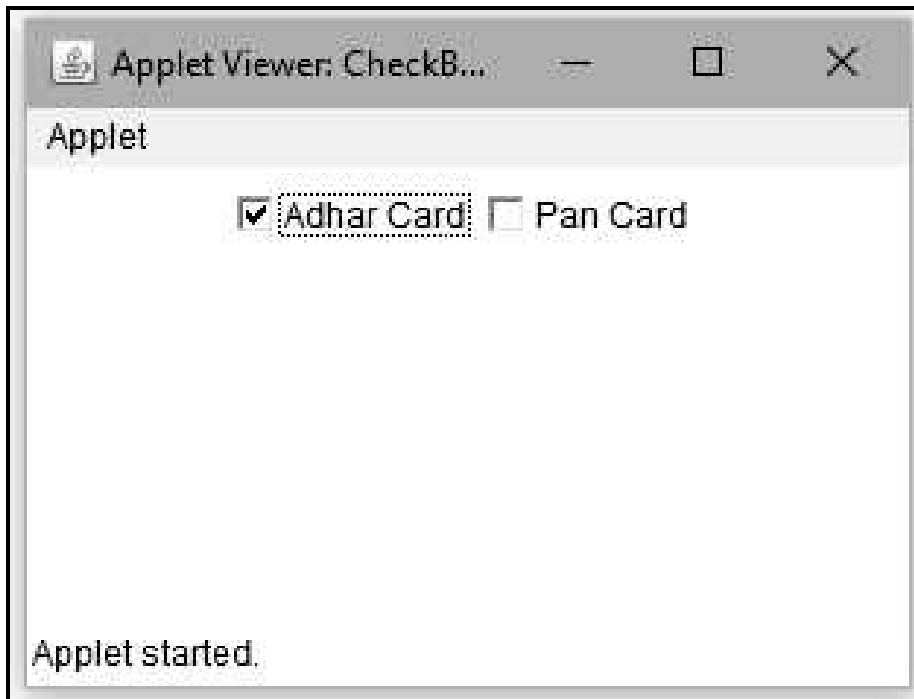
1.7.4 Checkboxes

- The Checkbox class is used to create check boxes. It is used to turn an option true or false. Following program shows the use of checkbox.

Example Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CheckBoxDemo" width=350 height=200>
</applet>
*/
public class CheckBoxDemo extends Applet
{
```

```
Checkbox cb1=new Checkbox("Adhar Card",true);
Checkbox cb2=new Checkbox("Pan Card");
public void init()
{
    add(cb1);
    add(cb2);
}
}
```

Output**1.8 Multiple Choice Questions**

Q.1 On which side applet always executed ?

- a Server side
- b Client side

Q.2 Which method of the Applet class displays the result of applet code on screen ?

- a run() method
- b paint() method
- c drawString() method
- d main() method

Q.3 Applet can be embedded in _____.

- a HTML document
- b word document
- c gif file
- d rtf file

Q.4 Which of the following is true about applet ?

- a Applets do not have main() method.
- b Applets must run under appletviewer or web browser.
- c The user I/O is not performed using Java's stream I/O class.
- d All of these.

Q.5 Executable applet is _____.

- a .applet file
- b .java.html
- c .java file
- d .class file

Q.6 Which object can be constructed to show any number of choices in the visible window ?

- a Labels
- b Choice
- c List
- d Checkbox

Q.7 The drawString method of defined in _____.

- a java.awt
- b java.applet
- c java.io
- d java.util

Q.8 Which class provides many methods for graphics programming ?

- a java.awt
- b java.Graphics
- c java.awt.Graphics
- d None of these

Q.9 When we invoke repaint() for a java.awt.Component object, the AWT invokes the method :

- a draw()
- b update()
- c show()
- d paint()

Q.10 Which method executes only once ?

- a start()
- b stop()
- c init()
- d destroy()

Q.11 What does the following line of code do ?

Textfield text = new Textfield(20);

- a Creates text object that can hold 20 rows of text.
- b Creates text object that can hold 20 columns of text.
- c Creates the object text and initializes it with the value 20.
- d This is invalid code

Answer Keys for Multiple Choice Questions :

Q.1	b	Q.2	b	Q.3	a
Q.4	d	Q.5	d	Q.6	c
Q.7	a	Q.8	c	Q.9	b
Q.10	c	Q.11	b		



UNIT II

2

Event Handling using AWT / Swing Components

Syllabus

Event Handling : Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes, inner classes. The AWT class hierarchy, user interface components - labels, button, canvas, scrollbars, text components, checkbox, checkbox groups, choices, lists panels - scroll pane, dialogs, menu bar, graphics, layout manager - layout manager types - boarder, grid, flow, card and grib bag.

Contents

- 2.1 *Events*
- 2.2 *Event Sources*
- 2.3 *Event Classes*
- 2.4 *Event Listeners*
- 2.5 *Delegation Event Model*
- 2.6 *Handling Mouse Events*
- 2.7 *Handling Keyboard Events*
- 2.8 *Adapter Classes*
- 2.9 *Inner Classes*
- 2.10 *What is Abstract Windowing Toolkit ?*
- 2.11 *The AWT Class Hierarchy*
- 2.12 *Limitations of AWT*
- 2.13 *User Interface Components*
- 2.14 *Dialogs*
- 2.15 *Menu bar*
- 2.16 *Programming Examples based on AWT Components and Event Handling*
- 2.17 *Graphics*
- 2.18 *Layout Manager*
- 2.19 *Multiple Choice Questions*

Part I : Event Handling

2.1 Events

Event is a changing state of an object. For example - The user clicks the mouse button or user presses the key on the keyboard.

2.2 Event Sources

- An event source is the object that generates the event. For example if you click a button an **ActionEvent** object is generated.
- The object of the **ActionEvent** class has all corresponding information about this event. Here button is an event source.
- Various event sources can be **button, checkbox, textbox, scrollbars** and so on.

2.3 Event Classes

- Event classes are the classes responsible for handling events in the event handling mechanism.
- The **EventObject** class is at the top of the event class hierarchy. It belongs to the **java.util** package. And other event classes are present in **java.awt.event** package.
- The **getSource()** and **toString()** are the methods of the **EventObject** class.
- There is **getId()** method belonging to **java.awt.event** package returns the **nature of the event**.
- For example, if a keyboard event occurs, you can find out whether the event was key press or key release from the event object.
- Various event classes that are defined in **java.awt.event** class are -
 1. An **ActionEvent** object is generated when a component is activated. For example if a button is pressed or a menu item is selected then this event occurs.
 2. An **AdjustmentEvent** object is generated when scrollbars are used.
 3. A **TextEvent** object is generated when text of a component or a text field is changed.
 4. A **ContainerEvent** object is generated when component are added or removed from container.
 5. A **ComponentEvent** object is generated when a component is resized, moved, hidden or made visible.
 6. An **ItemEvent** is generated when an item from a list is selected. For example a choice is made or if checkbox is selected.

7. A **FocusEvent** object is generated when component receives keyboard focus for input.
8. A **KeyEvent** object is generated when key on keyboard is pressed or released.
9. A **WindowEvent** object is generated when a window activated, maximized or minimized.
10. A **MouseEvent** object is generated when a mouse is clicked, moved, dragged, released.

2.3.1 ActionEvent Class

Description	When an event gets generated due to pressing of button, or by selecting menu item or by selecting an item, this event occurs.
Constructors	ActionEvent(Object source, int id, string command, long when, int modifier) The <i>source</i> indicates the object due to which the event is generated. The <i>id</i> which is used to identify the type of event. The <i>command</i> is a string that specifies the command that is associated with the event. The <i>when</i> denotes the time of event. The <i>modifier</i> indicates the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.
Methods	<ul style="list-style-type: none"> • String getActionCommand() : This method is useful for obtaining the <i>command</i> string which is specified during the generation of event. • int getModifiers() : This method returns the value which indicates the type of key being pressed at the time of event. • long getWhen() : It returns the time at which the event occurs.
Constants	There are four constants that are used to indicate the modifier keys being pressed. These constants are CTRL_MASK, SHIFT_MASK, META_MASK and ALT_MASK.

2.3.2 ItemEvent Class

Description	This event gets caused when an item is selected or deselected.
Constructors	ItemEvent(ItemSelectable, int, Object, int) Constructs a ItemSelectEvent object with the specified ItemSelectable source, type, item and item select state.
Methods	<ol style="list-style-type: none"> 1) getItem() : It returns the item where the event occurred. 2) getItemSelectable() : Returns the ItemSelectable object where this event originated. 3) getStateChange() : Returns the state change type which generated the event.
Constants	<ol style="list-style-type: none"> 1) ITEM_FIRST : Marks the first integer id for the range of item 2) ITEM_LAST : Marks the last integer id for the range of item

- | |
|--|
| <ul style="list-style-type: none"> 3) ITEM_STATE_CHANGED : The item state changed event type 4) SELECTED : The item selected state change type 5) DESELECTED : The item de-selected state change type |
|--|

2.3.3 KeyEvent Class

Description	This event is generated when key on the keyboard is pressed or released.
Constructors	<p>KeyEvent(Component source, int type, long t, int modifiers, int code)</p> <p>The <i>source</i> is a reference to the component that generates the event.</p> <p>The <i>type</i> specifies the type of event.</p> <p>The <i>t</i> denotes the system time at which the event occurs.</p> <p>The <i>modifiers</i> represent the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.</p> <p>The <i>code</i> represents the virtual keycode such as VK_UP, VK_ESCAPE and so on.</p>
Methods	<ul style="list-style-type: none"> • char getKeyChar() : It returns the character when a key is pressed.
Constants	<ul style="list-style-type: none"> • KEY_PRESSED : This event is generated when the key is pressed. • KEY_RELEASED : This event is generated when the key is released. • KEY_TYPED : This event is generated when the key is typed.

2.3.4 MouseEvent Class

Description	<p>This event occurs when mouse action occurs in a component. It reacts for both mouse event and mouse motion event.</p> <ul style="list-style-type: none"> • Mouse Events <ul style="list-style-type: none"> ○ A mouse button is pressed ○ A mouse button is released ○ A mouse button is clicked (pressed and released) ○ The mouse cursor enters a component ○ The mouse cursor exits a component • Mouse Motion Events <ul style="list-style-type: none"> ○ The mouse is moved ○ The mouse is dragged
Methods	<ol style="list-style-type: none"> 1) int getButton() : It returns which of the mouse button has changed the state. 2) int getClickCount() : It returns number of mouse clicks. 3) Point getLocationOnScreen() : Returns the absolute x, y position at that event. 4) Point getPoint() : Returns the x,y position of the event relative to the source component. 5) int getX() : Returns the horizontal x position of the event relative to the source component. 6) int getY() : Returns the vertical y position of the event relative to the source component.

Fields	<p>Following are the fields for java.awt.event.MouseEvent class :</p> <ul style="list-style-type: none"> • static int BUTTON1 --Indicates mouse button #1; used by getButton() • static int BUTTON2 --Indicates mouse button #2; used by getButton() • static int BUTTON3 --Indicates mouse button #3; used by getButton() • static int MOUSE_CLICKED --The "mouse clicked" event • static int MOUSE_DRAGGED --The "mouse dragged" event • static int MOUSE_ENTERED --The "mouse entered" event • static int MOUSE_EXITED --The "mouse exited" event • static int MOUSE_FIRST --The first number in the range of ids used for mouse events • static int MOUSE_LAST -- The last number in the range of ids used for mouse events • static int MOUSE_MOVED --The "mouse moved" event • static int MOUSE_PRESSED -- The "mouse pressed" event • static int MOUSE_RELEASED --The "mouse released" event • static int MOUSE_WHEEL --The "mouse wheel" event • static int NOBUTTON --Indicates no mouse buttons; used by getButton() <p>static int VK_WINDOWS --Constant for the Microsoft Windows "Windows" key.</p>
---------------	---

2.3.5 TextEvent Class

Description	This event indicates that object's text is changed.
Constructor	<p>TextEvent(Object source, int id)</p> <p>It constructs a TextEvent object.</p>
Methods	String paramString() : Returns a parameter string identifying this text event.
Fields	<ul style="list-style-type: none"> • TEXT_FIRST : The first number in the range of ids used for text events. • TEXT_LAST : The last number in the range of ids used for text events. • TEXT_VALUE_CHANGED : This event id indicates that object's text changed.

2.3.6 WindowEvent Class

Description	This is an event that indicates that a window has changed its status.
Constructor	<p>WindowEvent(Window source, int id) : Constructs a WindowEvent object.</p> <p>WindowEvent(Window source, int id, int oldState, int newState) : Constructs a WindowEvent object with the specified previous and new window states.</p>
Methods	<ol style="list-style-type: none"> 1) int getNewState() : It returns the new state of the window. 2) int getOldState() : It returns the previous state of the window. 3) Window getOppositeWindow() : Returns the other Window involved in this focus or activation change. 4) Window getWindow() : Returns the originator of the event.

Fields

- **WINDOW_ACTIVATED** : The window-activated event type.
- **WINDOW_CLOSED** : The window closed event.
- **WINDOW_DEACTIVATED** : The window-deactivated event type.
- **WINDOW_FIRST** : The first number in the range of ids used for window events.
- **WINDOW_LAST** : The last number in the range of ids used for window events.
- **WINDOW_OPENED** : The window opened event.
- **WINDOW_STATE_CHANGED** : The window-state-changed event type.

2.4 Event Listeners

- The task of handling an event is carried out by **event listener**.
- When an event occurs, first of all an event object of the appropriate type is created. This object is then passed to a **Listener**.
- A listener must **implement the interface** that has the method for event handling.
- The **java.awt.event** package contains definitions of all event classes and **listener interface**.
- **An Interface** contains constant values and method declaration.
- The methods in an interface are only declared and not implemented, i.e. the methods do not have a body.
- The interfaces are used to define behavior on occurrence of event that can be implemented by any class anywhere in the class hierarchy.

2.4.1 ActionListener Interface

- This interface defines the method **actionPerformed()** which is invoked when an **ActionEvent** occurs.
- The **syntax** of this method is

```
void actionPerformed(ActionEvent act)
```

where **act** is an object of **ActionEvent** class.

2.4.2 ItemListener Interface

- This interface is used when an item from a list is selected. For example a choice is made or if checkbox is selected.
- The **itemStateChanged()** is the only method defined by the **ItemListener** interface.
- The **syntax** is

```
void itemStateChanged(ItemEvent It)
```


2.4.3 KeyListener Interface

- This interface is defining the events such as `keyPressed()`, `keyReleased()` and `keyTyped()` are used.
- These methods are useful for key press, key release and when you type some characters.

```
void keyPressed(KeyEvent k)
void keyReleased(KeyEvent k)
void keyTyped(KeyEvent k)
```

2.4.4 MouseListener Interface

- This interface defines five important methods for various activities such as mouse click, press, released, entered or exited. These are

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

2.4.5 MouseMotion Interface

- For handling mouse drag and mouse move events the required methods are defined by `MouseMotionListener` interface. These methods are

```
void mouseDragged(MouseEvent m)
void mouseMoved(MouseEvent m)
```

2.4.6 TextListener Interface

- An event of type `TextEvent` is generated when a value in textfield or textarea is entered or edited.
- The methods used by `TextListener` Interface are

```
public String paramString():
public void textValueChanged(TextEvent e):
```

2.4.7 WindowListener Interface

- There are seven methods in which are related to windows activation and deactivation.

```
void windowOpened(WindowEvent w)
void windowClosed(WindowEvent w)
void windowClosing(WindowEvent w)
void windowActivated(WindowEvent w)
void windowDeactivated(WindowEvent w)
```

```
void windowIconified(WindowEvent w)
void windowDeiconified(WindowEvent w)
```

2.5 Delegation Event Model

- **Basic Concept : Event delegation model** is used for understanding the event and for processing it. The event-handler method takes the Event object as a parameter. For handling particular event specific object of event must be mentioned.
- There are four main components based on this model are

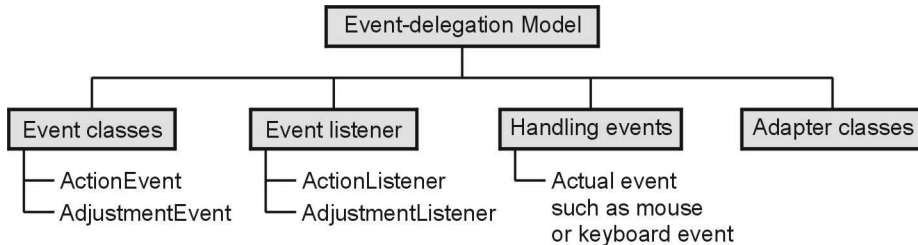


Fig. 2.5.1 Components of event delegation model

Advantages of Event Delegation Model

Following are the advantages of event delegation model -

1. In event delegation model the events are handled using objects. This allows a clear separation between the usage of the components and the design.
 2. It accelerates the performance of the application in which multiple events are used.
- Let us see how an event gets handled with the help of programming example.

Example : Handling Button Click

- The buttons are sometimes called as push buttons. We can associate some event on button click. That means on clicking the button, certain event gets triggered to perform required task. Following example illustrates this idea.

Example 2.5.1 Write a Java program to toggle the background color on every click of button.

Solution :

Java Program[Button1.java]

```
//This program alternatively changes the background color
//After every click of button
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Button1" width=350 height=200>
```

```

</applet>
*/
public class Button1 extends Applet

implements ActionListener
{

    Button button=new Button("change the color");
    boolean flag=true;
    public void init()
    {
        add(button);
        button.addActionListener(this);
    }
    public void paint(Graphics g)
    {
        if(flag)
            setBackground(Color.yellow);

        else

            setBackground(Color.red);
    }
    public void actionPerformed(ActionEvent e)
    {
        String str=e.getActionCommand();
        if(str.equals("change the color"))
        {
            flag=!flag;
            //toggle the flag values on every click of button
            repaint();
        }
    }
}

```

Event Listener Interface

Placing the button control

Invoking the button click Event

ActionListener Interface defines action performed method. Using this method event is handled

Event gets handled here

Output

D:\>Appletviewer Button1.java



Program Explanation

In above program,

- 1) When a button is clicked the action of changing the background color occurs. Hence this program must implement the **ActionListener** interface.
- 2) This listener class must have a method called **actionPerformed** which has a parameter an object-**ActionEvent**.
- 3) By invoking the method **getActionCommand** we can recognise that the event has occurred by clicking the button.

2.6 Handling Mouse Events

- In order to handle any event, we need to use Listener interfaces. For handling the mouse events the **MouseListener** and **MouseMotionListener** interfaces are used.

Methods of MouseListener interface

- There are 5 abstract methods used under **MouseListener** interface and those are -
 1. public abstract void mouseClicked(MouseEvent e);
 2. public abstract void mouseEntered(MouseEvent e);
 3. public abstract void mouseExited(MouseEvent e);
 4. public abstract void mousePressed(MouseEvent e);
 5. public abstract void mouseReleased(MouseEvent e);

Methods of MouseMotionListener interface

- There are 2 abstract methods used under **MouseMotionListener** interface and those are -
 1. public abstract void mouseDragged(MouseEvent e);
 2. public abstract void mouseMoved(MouseEvent e);

Example Program

Java Program[MouseEventDemo.java]

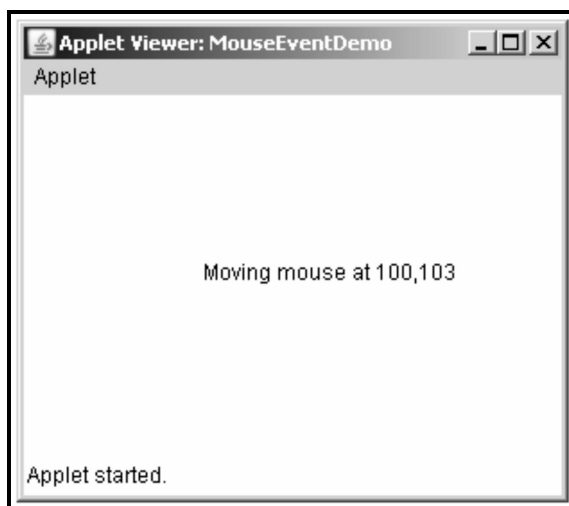
```
/*
This is a Java program which is for handing mouse events
*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="MouseEventDemo" width=300 height=200>
```

```
</applet>
*/
public class MouseEventDemo extends Applet
implements MouseListener,MouseMotionListener
{
String msg="";
int xposition=0,yposition=0;
public void init()
{
addMouseListener(this);
addMouseMotionListener(this);
}
public void mouseClicked(MouseEvent m)
{
xposition=m.getX();
yposition=m.getY();
msg="mouse Clicked";
repaint();
}
public void mousePressed(MouseEvent m)
{
xposition=m.getX();
yposition=m.getY();
msg="Pressing mouse button";
repaint();
}
public void mouseReleased(MouseEvent m)
{
xposition=m.getX();
yposition=m.getY();
msg="Releasing mouse button";
repaint();
}
public void mouseEntered(MouseEvent m)
{
xposition=0;
yposition=190;
msg="mouse Entered";
repaint();
}

public void mouseExited(MouseEvent m)
{
xposition=0;
```

```
yposition=190;
msg="mouse Exited";
repaint();
}
public void mouseDragged(MouseEvent m)
{
    xposition=m.getX();
    yposition=m.getY();
    msg="Dragging mouse at "+xposition+","+yposition;
    repaint();
}
public void mouseMoved(MouseEvent m)
{
    xposition=m.getX();
    yposition=m.getY();
    msg="Moving mouse at "+xposition+","+yposition;
    repaint();
}
public void paint(Graphics g)
{
    g.drawString(msg,xposition,yposition);
}
}
```

Output



Program Explanation

In above program, we have used

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

because various commonly used events are defined in the package **java.awt.event**.

- The applet has to register itself as a listener to various events (*here the same applet acts as a event source as well as event listener*). Hence inside **init()** method applet registers itself as a listener by following statements

```
addMouseListener(this);
addMouseMotionListener(this);
```

- And then simple methods of mouse events are defined. The **getX()** and **getY()** methods return the current x and y positional values. To each of these methods an object of **MouseEvent** is passed which is shown by a variable 'm'.

Example 2.6.1 *What method is used to distinguish between single, double and triple mouse clicks ? Illustrate.*

Solution : **public int getClickCount()** : This method returns the number of mouse clicks.

Hence we can check -

```
if (mouseEvent.getClickCount() == 1)
{
    System.out.println("Single Click");
}
else if (mouseEvent.getClickCount() == 2)
{
    System.out.println("Double Click");
}
else if (mouseEvent.getClickCount() == 3)
{
    System.out.println("Triple Click");
}
```

2.7 Handling Keyboard Events

- When key from a keyboard is pressed then it causes an event.
- There are three commonly used methods from **KeyListener** interface and those are
 1. **keyPressed()**,
 2. **keyReleased()** and
 3. **keyTyped()**.
- The use of these methods is shown by following Java program -

Java Program [KeyboardDemo.java]

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="KeyboardDemo" width=500 height=300>
```

```
</applet>
*/
public class KeyboardDemo extends Applet
implements KeyListener
{
String msg="";
public void init()
{
addKeyListener(this);
requestFocus();
}
public void keyPressed(KeyEvent k)
{
showStatus("Key Pressed");
}
public void keyReleased(KeyEvent k)
{
showStatus("Key Released");
}
public void keyTyped(KeyEvent k)
{
Font f;
f=new Font("Monotype Corsiva",Font.BOLD,30);
msg+=k.getKeyChar();
setFont(f);
repaint();
}
public void paint(Graphics g)
{
g.drawString(msg,30,70);
}
}
```

Output



Program Explanation : In above program, we have used -

- 1) The **keyPressed()** event is for handling the event of pressing a key, similarly **keyReleased()** event is for handling the event of release of key.
- 2) Using **keyTyped()** method we can display the character typed on the screen. The only needed method for this is **getKeyChar()** which returns the currently typed character.
- 3) Last but not the least, all these methods require one important method that has to be invoked in **init()** function and that is **requestFocus()**. This method has to be invoked to gain the focus in the **init** method and whenever you want to implement **KeyListener** interface.

2.8 Adapter Classes

- It is basically a class in Java that implements an interface with a set of dummy methods.
- The famous adapter classes in Java API are **WindowAdapter**, **ComponentAdapter**, **ContainerAdapter**, **FocusAdapter**, **KeyAdapter**, **MouseAdapter** and **MouseMotionAdapter**.
- Whenever your class implements such interface, you have to implement all of the seven methods.
- **WindowAdapter** class implements **WindowListener** interface and make seven empty implementation.
- When you class subclass **WindowAdapter** class, you may choose the method you want without restrictions.
- The following give such an example.

```
public interface WindowListener {
    public void windowClosed(WindowEvent e);
    public void windowOpened(WindowEvent e);
    public void windowIconified(WindowEvent e);
    public void windowDeiconified(WindowEvent e);
    public void windowActivated(WindowEvent e);
    public void windowDeactivated(WindowEvent e);
    public void windowClosing(WindowEvent e);
}
public class WindowAdapter implements WindowListener{
    public void windowClosed(WindowEvent e){}
    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
```

```
public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowClosing(WindowEvent e){}
}
```

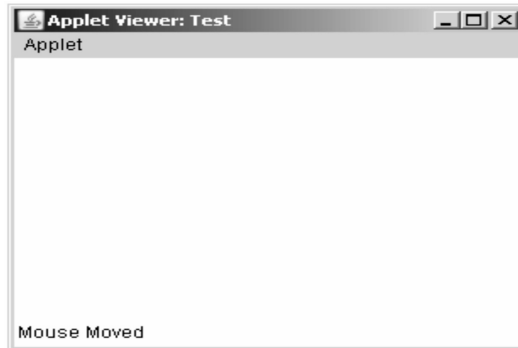
- You can add adapter class as subclass and override just the methods you need.
- Here is a simple Java program for handling mouse events.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Test" width=300 height=200>
</applet>
*/
public class Test extends Applet
{
    public void init()
    {
        //defining the adapter classes
        //adapter1 is for MouseListener
        //adapter2 is for MouseMotionListener

        addMouseListener(new adapter1(this));
        addMouseMotionListener(new adapter2(this));
    }
}
class adapter1 extends MouseAdapter
{
    //object of main Test class
    Test obj;
    public adapter1(Test obj)
    {
        this.obj=obj;
    }
    //method belonging to MouseListener interface
    public void mouseClicked(MouseEvent m)
    {
        obj.showStatus("Mouse clicked");
    }
}
class adapter2 extends MouseMotionAdapter
{
    Test obj;
    public adapter2(Test obj)
    {
```

```
    this.obj=obj;
}
//method belonging to MouseMotionListener interface
public void mouseMoved(MouseEvent m)
{
    obj.showStatus("Mouse Moved");
}
}
```

Output



Program Explanation :

- 1) In this program we have used two interfaces **MouseListener** and **MouseMotionListener**.
- 2) Two adapter classes are created. The **adapter1** class is for **MouseListener** interface and the **adapter2** class is for **MouseMotionListener** interface.
- 3) Note that, in the class **Test** we have written **init** method in which two adapter classes are created and registered as Listener using following statements

```
addMouseListener(new adapter1(this));
addMouseMotionListener(new adapter2(this));
```
- 4) Then we have defined **adapter1** class in which object for adapter class **adapter1** is initialised. Note that we have written **mouseClicked** method in adapter1 class because **mouseClicked** method is belonging to the **MouseListener** interface and the adapter class **adapter1** is for **MouseListener**.
- 5) Same is true for the **adapter2** class which is an adapter class for **MouseMotionListener**. Note that in the definition of **adapter2** class **mouseMoved** method is written because **mouseMoved** method is belonging to the **MouseMotionListener**.
- 6) On running the above code, appropriate status will be shown on applet on encountering particular mouse event (either mouse click or mouse move).

Review Question

1. What are the various adapter classes that implements commonly used Listener interfaces ?
Write a sample Java program to demonstrate an Adapter.

2.9 Inner Classes

Definition : Inner classes are the nested classes. That means these are the classes that are defined inside the other classes.

The syntax of defining the inner class is -

```
Access_modifier class OuterClass
{
    //code
    Access_modifier class InnerClass
    {
        //code
    }
}
```

Properties of inner classes

Following are some properties of inner class -

- The outer class can inherit as many number of inner class objects as it wants.
- If the outer class and the corresponding inner class both are **public** then any other class can create an instance of this inner class.
- The inner class objects do not get instantiated with an outer class object.
- The outer class can call the private methods of inner class.
- Inner class code has free access to all elements of the outer class object that contains it.
- If the inner class has a variable with same name then the outer class's variable can be accessed like this -

```
outerclassname.this.variable_name
```

There are **four types of inner classes** -

2.9.1 Static Member Classes

- This inner class is defined as the static member variable of another class.
- Static members of the outer class are visible to the **static inner** class.
- The non-static members of the outer class are not available to inner class.

Syntax

```
Access_modifier class OuterClass
{
    //code
    public static class InnerClass
    {
        //code
    }
}
```

2.9.2 Member Classes

This type of inner class is non-static member of outer class.

Syntax

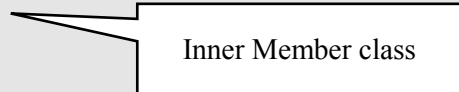
```
Access_modifier class outerclass
{
    //code
    Access_modifier class interclass
    {
        //code
    }
}
```

Example

```
class A
{
    private String name="Ankita";

    class B
    {
        void display()
        {
            System.out.println("Name is: "+name);
        }
    }

    public static void main(String args[])
    {
        A obj=new A();
        A.B in_obj=obj.new B();
        in_obj.display();
    }
}
```

A diagram consisting of a rectangular box with the text "Inner Member class" inside. A line extends from the left side of the box, branching into two lines that point to the "class B" declaration and the opening curly brace of its class body in the code above.

Output

```
Name is : Ankita
```

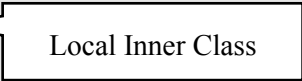
2.9.3 Local Classes

- This class is defined within a Java code just like a local variable.
- Local classes are never declared with an access specifier.
- The scope inner classes is always restricted to the block in which they are declared.
- The local classes are completely hidden from the outside world.

Syntax

```
Access_modifier class OuterClass
{
    //code
    Access_modifier return_type methodname(arguments)
    {
        class InnerClass
        {
            //code
        }
        //code
    }
}
```

Example

```
public class A
{
    private String name="Ankita";//instance variable
    void display()
    {
        class Local 
        {
            void show()
            {
                System.out.println("From Local class: "+name);
            }
        }
        Local obj=new Local();
        obj.show();
    }
    public static void main(String args[])
    {
        A a=new A();
        a.display();
    }
}
```

Output

```
From Local class: Ankita
```

2.9.4 Anonymous Classes

- Anonymous class is a local class without any name.
- Anonymous class is a one-shot class- created exactly where needed.
- The anonymous class is created in following situations -
 - When the class has very short body.
 - Only one instance of the class is needed.
 - Class is used immediately after defining it.
- The anonymous inner class can extend the class, it can implement the interface or it can be declared in method argument.

Programming Example

```
class MyInnerClass implements Runnable
{
    public void run()
    {
        System.out.println("Hello");
    }
}
class DemoClass
{
    public static void main(String[] arg)
    {
        MyInnerClass my=new MyInnerClass();
        Thread th=new Thread(my);
        my.start();
    }
}
```

Review Question

1. What is inner classes ? Explain its types.

Part II : AWT

2.10 What is Abstract Windowing Toolkit ?

- The AWT stands for Abstract Window Toolkit.
- The AWT package contains large number of classes which help to include various **graphical components** in the Java Program.
- The graphical components include text box, buttons, labels, radio buttons and so on.

2.11 The AWT Class Hierarchy

- The AWT classes are arranged in hierarchical manner which is known as AWT Hierarchy. Refer Fig. 2.11.1.

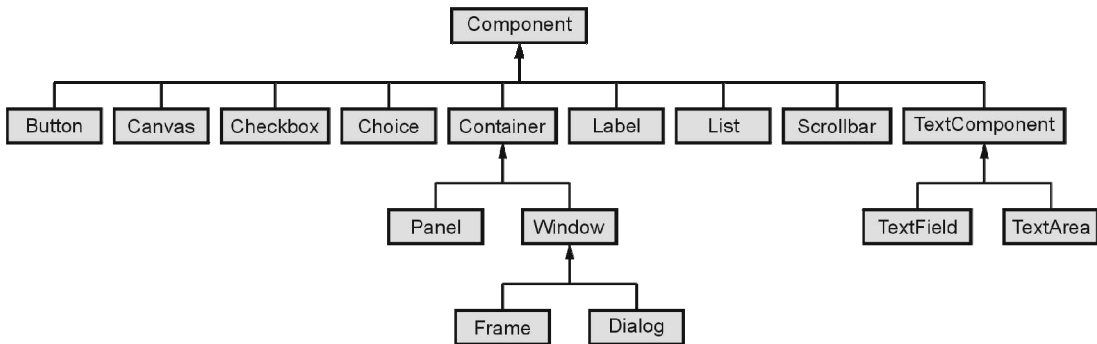


Fig. 2.11.1 AWT hierarchy

- The hierarchy components classes are -
 - Component** : This is the super class of all the graphical classes from which variety of graphical classes can be derived. It helps in displaying the graphical object on the screen. It handles the mouse and keyboard events.
 - Container** : This is a graphical component derived from **the component** class. It is responsible for managing the layout and placement of graphical components in the container.
 - Window** : The top level window without border and without the menu bar is created using the window class. It decides the layout of the window.
 - Panel** : The panel class is derived from the **container** class. It is just similar to window - without any border and without any menu bar, title bar.
 - Frame** : This is a top-level window with a border and menu bar. It supports the common window events such as window open, close, activate and deactivate.

Review Question

1. Explain AWT hierarchy in detail

2.12 Limitations of AWT

Following are some limitations of AWT

1. The AWT components are heavyweight in nature.
2. It does not support some advanced components such as trees, tables and so on.
3. It is platform dependable.
4. The buttons of AWT does not support pictures.

2.13 User Interface Components

- There are various graphical components that can be placed on the frame. These components have the classes. These classes have the corresponding methods.
- When we place the components on the frame we need to set the layout of the frame.
- The commonly used layout is **FlowLayout**. The **FlowLayout** means the components in the frame will be placed from left to right in the same manner as they get added.
- Various components that can be placed for designing user interface are -
 1. Label
 2. Buttons
 3. Canvas
 4. Scroll bars
 5. Text Components
 6. Checkbox
 7. Choices
 8. Lists Panels
 9. Dialogs
 10. Menubar
- Let us discuss these components one by one, but before that let us understand how to create a frame on which we can place the components.

2.13.1 Labels

- The syntax of this control is

```
Label (String s)
Label(String s, int style)
```

- where the **s** of String type represent the string contained by the label similarly in the other label function style is a constant used for the style of label. It can be Label.LEFT, Label.RIGHT and Label.CENTER. Here is a JAVA program which makes use Label.

Example 2.13.1 Write a simple Java program to demonstrate the use of label components.

Solution :

Java Program[Use_Label.java]

```
import java.awt.*;
class Use_Label
{
    public static void main(String[] args)
    {
        Frame fr=new Frame("This Program is for Displaying the Label");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("OK");
        Label L2=new Label("CANCEL");
        fr.add(L1);
        fr.add(L2);
    }
}
```

```
}  
}
```

Output

```
C:\>javac Use_label.java
```

```
C:\>java Use_label
```



2.13.2 Buttons

- Buttons are sometimes called as **push buttons**. This component contains a label and when it is pressed it generates an event.
- The syntax of this control is

```
Button (String s)
```

Java Program

```
import java.awt.*;  
class Use_Button  
{  
    public static void main(String[] args)  
    {  
        Frame fr=new Frame("This Program is for Displaying the Button");  
        fr.setSize(400,200);  
        fr.setLayout(new FlowLayout());  
        fr.setVisible(true);  
        Button B1=new Button("OK");  
        Button B2=new Button("CANCEL");  
        fr.add(B1);  
        fr.add(B2);  
    }  
}
```

Output



- We can create an array of buttons. Following is a simple Java program which illustrates this ideas -

Java Program[Use_Button_Arr.java]

```
import java.awt.*;
class Use_Button_Arr
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Buttons");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Button buttons[]=new Button[5];
        String Fruits[]={"Mango","Orange","Banana","Apple","Strawberry"};
        for(i=0;i<5;i++)
        {
            buttons[i]=new Button(" "+Fruits[i]);
            fr.add(buttons[i]);
        }
    }
}
```

Output



2.13.3 Canvas

- Canvas is a **special area** created on the frame.
- The canvas is specially used for drawing the graphical components such as oval, rectangle, line and so on.

- Various methods of this class are

Method	Description
void setSize(int width, int height)	This method sets the size of the canvas for given width and height.
void setBackground(Color c)	This method sets the background color of the canvas.
void setForeground(Color c)	This method sets the color of the text.

- Following is a simple Java program which shows the use of canvas -

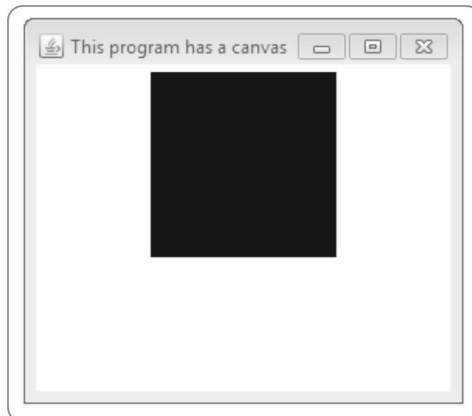
Java Program[Use_Canvas.java]

```
import java.awt.*;
class Use_Canvas
{
    public static void main(String[] args)
    {
        Frame Fr = new Frame("This program has a canvas");
        Canvas C1 = new Canvas();
        C1.setSize(120,120);
        C1.setBackground(Color.blue);
        Fr.setLayout(new FlowLayout());
        Fr.setSize(250,250);
        Fr.setVisible(true);
        Fr.add(C1);
    }
}
```

Output

```
C:\>javac Use_Canvas.java
```

```
C:\>java Use_Canvas
```



2.13.4 Scrollbars

- Scrollbar can be represented by the slider widgets.
- There are two styles of scroll bars - Horizontal scroll bar and vertical scroll bar.
- Following program shows the use of this component.

Java Program

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

        Frame Fr = new Frame("This program has a scrollbars");
        Scrollbar HSelector = new Scrollbar(Scrollbar.HORIZONTAL);
        Scrollbar VSelector = new Scrollbar(Scrollbar.VERTICAL);

        Fr.setLayout(new FlowLayout());
        Fr.setSize(300,300);
        Fr.setVisible(true);
        Fr.add(HSelector);
        Fr.add(VSelector);
    }
}
```

Output



2.13.5 Text Components

- In Java there are two controls used for text box - One is **TextField** and the other one is **TextArea**.
- The **TextField** is a slot in which one line text can be entered. In the **TextField** we can enter the string, modify it, copy, cut or paste it. The syntax for the text field is

```
int TextField(int n)
```

where n is total number of characters in the string.

- The **TextArea** control is used to handle multi-line text. The syntax is -

```
TextArea(int n,int m)
```

where n is for number of lines and m is for number of characters.

Following is a Java program which illustrates the use of **TextField** and **TextArea**.

Java Program[Use_TxtFld.java]

```
import java.awt.*;
class Use_TxtFld
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the TextField");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("Enter your name here");
        TextField input1=new TextField(10);
        Label L2=new Label("Enter your address here");
        TextArea input2=new TextArea(10,20);
        fr.add(L1);
        fr.add(input1);
        fr.add(L2);
        fr.add(input2);
    }
}
```

Output**2.13.6 Checkbox**

- Checkbox is basically a small box which can be ticked or not ticked.
- In Java we can select particular item using checkbox control.
- This control appears as small box along with label. The label tells us the name of the item to be selected.
- The syntax of checkbox is as given below -

```
Checkbox(String label)
```

where *label* denotes the label associated with each checkbox.

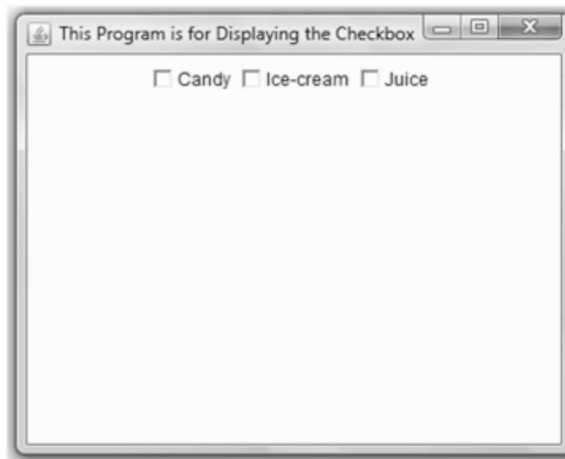
- To get the state of the checkbox the **getState()** method can be used.

Java Program[Use_ChkBox.java]

```
import java.awt.*;
class Use_ChkBox
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Checkbox");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Checkbox box1=new Checkbox("Candy");
```

```
Checkbox box2=new Checkbox("Ice-cream");
Checkbox box3=new Checkbox("Juice");
fr.add(box1);
fr.add(box2);
fr.add(box3);
}
}
```

Output



2.13.7 Checkbox Group

- The Checkbox Group component allows the user to make one and only one selection at a time.
- These checkbox groups is also called as **radio buttons**. The syntax for using checkbox groups is -

```
Checkbox(String str ,CheckboxGroup cbg , Boolean val);
```

- Following is a simple Java program which makes use of this control.

Java Program[Use_CheckBoxGr.java]

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

        Frame Fr = new Frame("This program uses checkbox groups");
        Fr.setLayout(new FlowLayout());
        Fr.setSize(300,300);
        Fr.setVisible(true);
```



```
CheckboxGroup cbg=new CheckboxGroup();
Checkbox box1=new Checkbox("Candy",cbg,true);
Checkbox box2=new Checkbox("Ice-cream",cbg,false);
Checkbox box3=new Checkbox("Juice",cbg,false);
Fr.add(box1);
Fr.add(box2);
Fr.add(box3);
}
}
```

Output



2.13.8 Choices

- This is a simple control which allows the popup list for selection.
- We have to create an object of type **choice** as follows -

```
Choice obj=new Choice();
```

Java Program[Use_Choice.java]

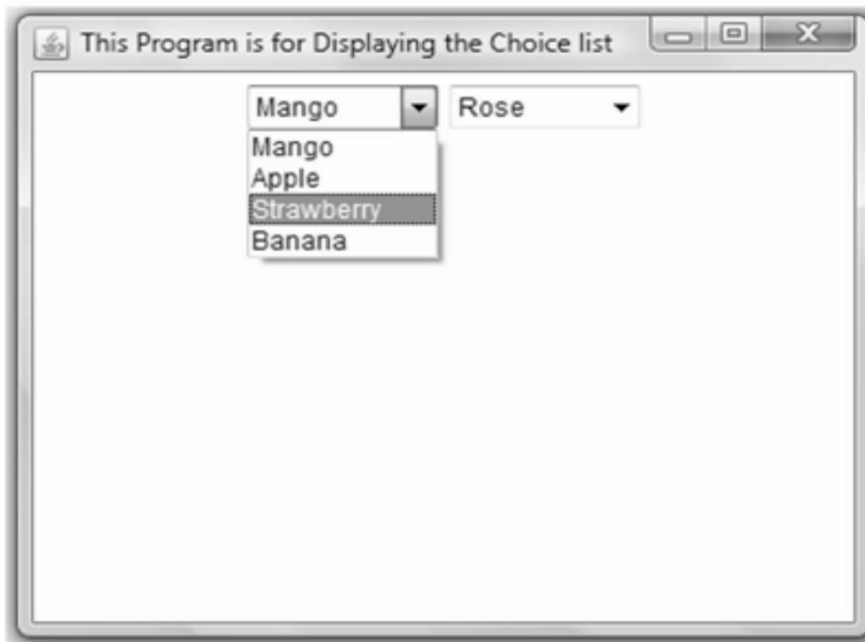
```
import java.awt.*;
class Use_Choice
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Choice list");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Choice c1=new Choice();
        Choice c2=new Choice();
        c1.add("Mango");
```

```
c1.add("Apple");
c1.add("Strawberry");
c1.add("Banana");

c2.add("Rose");
c2.add("Lily");
c2.add("Lotus");
fr.add(c1);
fr.add(c2);

}
}
```

Output



2.13.9 List Panels

- **List** is a collection of many items.
- By double clicking the desired item we can select it. Following Java program makes use of this control -

Java Program[Use_List.java]

```
import java.awt.*;
class Use_List
{
```

```
public static void main(String[] args)
{
    int i;
    Frame fr=new Frame("This Program is for Displaying the List");
    fr.setSize(350,300);
    fr.setLayout(new FlowLayout());
    fr.setVisible(true);
    List flower=new List(4,false);
    flower.add("Rose");
    flower.add("Jasmine");
    flower.add("Lotus");
    flower.add("Lily");
    fr.add(flower);
}
}
```

Output



2.14 Dialogs

- Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
- The dialog boxes does not have maximize and minimize buttons.

Constructor for using Dialog Box

Constructor	Description
Dialog(Dialog owner)	Creates modeless dialog window with a frame owner.
Dialog(Dialog owner, String title)	Creates modeless dialog window with a frame owner and string title.
Dialog(Dialog owner, String title, boolean modal)	Creates modeless dialog window with a frame owner, string title and modality to be true or false i.e. if the dialogbox is modal or modeless

What is modal and modeless Dialog Window ?**Modal Dialog Box**

- A Modal dialog box is one that the user must first close in order to have access to any other framed window or dialog box of the same application.

Modeless Dialog Box

- A dialog box is referred to as modeless if the user does not have to close it in order to continue using the application that owns the dialog box.

Java Program[DialogBoxProg.java]

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class DialogBoxProg extends Frame
{
    public static void main(String[] args)
    {
        Dialog d;
        Frame frame = new Frame();
        d=new Dialog(frame,"Dialog Box Demo",true);
        d.add( new Label ("This is a simple dialog box."));
        d.setSize(300,300);
        d.setVisible(true);
        frame.setSize(330,250);
        frame.setVisible(true);
    }
}
```

output



2.14.1 File Dialog

- The `FileDialog` class displays a dialog window from which the user can select a file.
- Since it is a modal dialog, when the application calls its `show` method to display the dialog, it blocks the rest of the application until the user has chosen a file.
- **Signature for File Dialogbox is**

```
public class FileDialog extends Dialog
```

Methods used by FileDialog

Name	Purpose
<code>getDirectory()</code>	gets the directory of file dialog
<code>getFile()</code>	gets selected file of file dialog
<code>getMode()</code>	indicates whether file dialog box is for loading from a file or saving to a file.
<code>setDirectory(String)</code>	Sets the directory of this file dialog window to be the specified directory.
<code>setFile(String)</code>	Sets the selected file for this file dialog window to be the specified file.
<code>setMode(int)</code>	Sets the mode of the file dialog

Variables used by FileDialog

Name	Purpose
LOAD	This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
SAVE	This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

Syntax

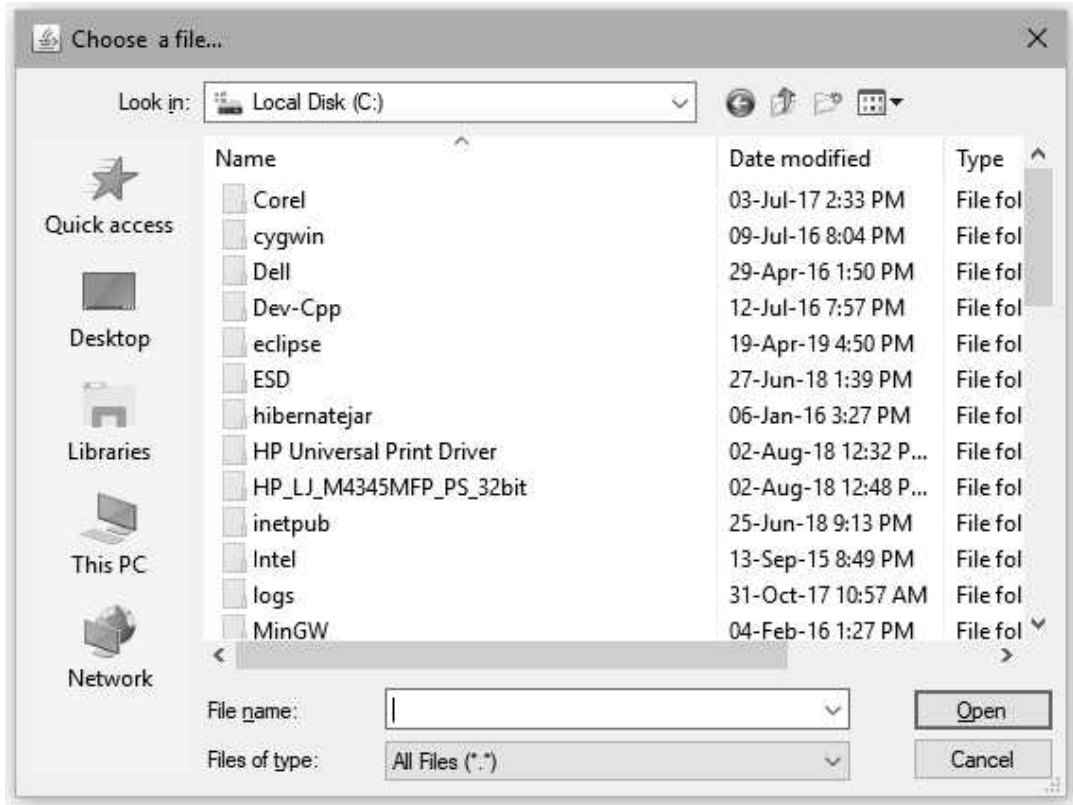
```
public FileDialog(Frame parent, String title, int mode)
```

- Creates a file dialog window with the specified title for loading or saving a file.
- If the value of mode is LOAD, then the file dialog is finding a file to read. If the value of mode is SAVE, the file dialog is finding a place to write a file.

Java Program[FileDialog.java]

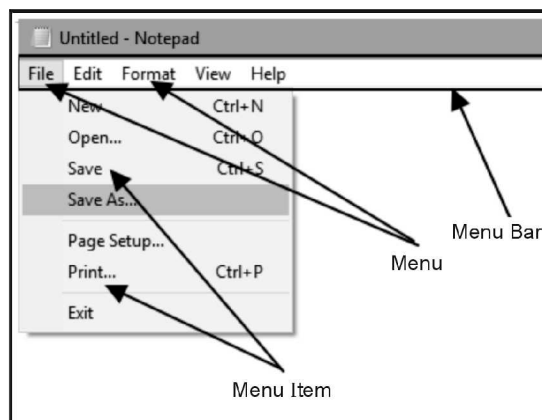
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class FileDialogBoxProg extends Frame
{
    public static void main(String[] args)
    {
        FileDialog fd;
        Frame frame = new Frame();
        fd=new FileDialog(frame,"Choose a file...",FileDialog.LOAD);
        fd.setDirectory("C:\\");
        fd.setSize(300,300);
        fd.setVisible(true);
        frame.setSize(330,250);
        frame.setVisible(true);
    }
}
```

output



2.15 Menu bar

- Menus are essential components of any Window based GUI. It allows the user to choose one of several options.
- Menus are created with the help of Menu items and these menus are placed on menubar. Following figure represents Menu, Menubar and Menuitems.



- **Constructor**

For creating Menu, menu items and Menu bar using AWT we use three constructors

Constructor	Description
public MenuBar()	It helps in creating the menubar on which the menus can be added.
public Menu(String title)	The menu items can be created using some title.
public MenuItem(String title)	The menu items are created with suitable title for particular menu.

- **Steps for creating menu**

Creation of menus involves many steps to be followed in an order. Following are the steps.

Step 1 : Create menu bar

Step 2 : Create menus

Step 3 : Create menu items

Step 4 : Add menu items to menus

Step 5 : Add menus to menu bar

Step 6 : Add menu bar to the frame

Following program demonstrates these steps in a simple Java program

Java Program[MenuDemo.java]

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MenuDemo extends Frame
{
    public static void main(String[] args)
    {
        MenuBar menuBar;
        Menu menu1;
        MenuItem mItem1, mItem2, mItem3;
        Frame frame = new Frame("MenuBar and Menu Demo");

        //Creating a menu bar
        menuBar= new MenuBar();

        //Creating menu
        menu1 = new Menu("File");

        //creating menu items
```



```
mItem1 = new MenuItem("New");
mItem2 = new MenuItem("Open");
mItem3 = new MenuItem("Save");

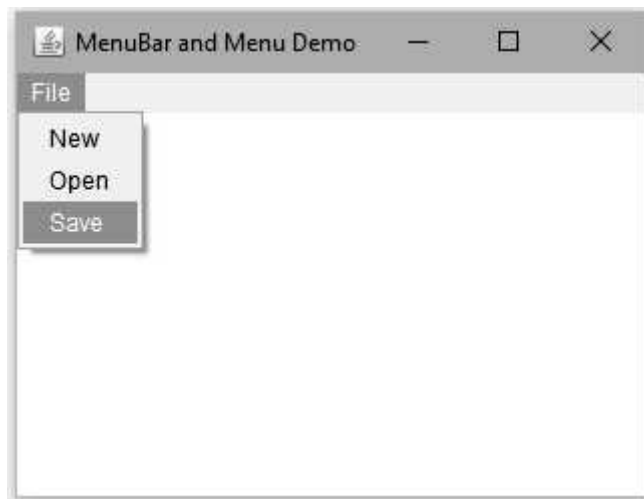
//Adding menu items to the menu
menu1.add(mItem1);
menu1.add(mItem2);
menu1.add(mItem3);

//Adding our menu to the menu bar
menuBar.add(menu1);

//Adding my menu bar to the frame by calling setMenuBar() method
frame.setMenuBar(menuBar);

frame.setSize(330,250);
frame.setVisible(true);
}
}
```

Output



Program Explanation : In above program,

- 1) We have created a menubar and a menu **File** is added to this menu bar.
- 2) The menu items **New, Open, Save** are added to the **File** menu.
- 3) This set of Menubar, Menu and Menu Item is then added on the frame.

How to add submenus ?

We can add submenu, to a menu. It is illustrated by following program.

Java Program[SubMenuDemo.java]

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class SubMenuDemo extends Frame
{
    public static void main(String[] args)
    {
        MenuBar menuBar;
        Menu menu1,menu2;
        MenuItem mItem1, mItem2, mItem3,mItem4,mItem5;
        Frame frame = new Frame("MenuBar and Menu Demo");

        //Creating a menu bar
        menuBar= new MenuBar();

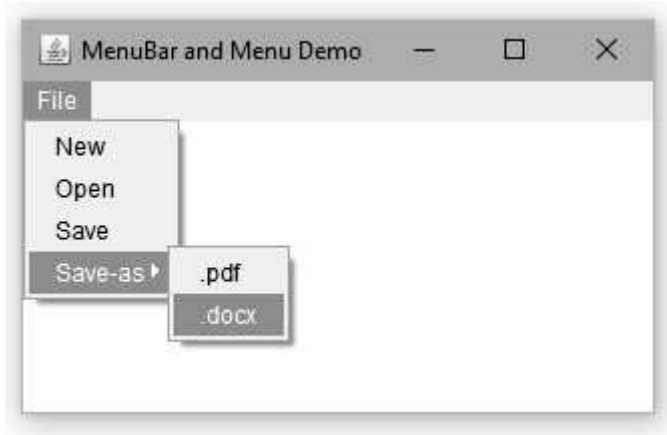
        //Creating menu
        menu1 = new Menu("File");

        //creating menu items
        mItem1 = new MenuItem("New");
        mItem2 = new MenuItem("Open");
        mItem3 = new MenuItem("Save");
        //Adding menu items to the menu
        menu1.add(mItem1);
        menu1.add(mItem2);
        menu1.add(mItem3);

        //creating sub menu
        menu2 = new Menu("Save-as");
        //Adding menu items to the submenu
        mItem4 = new MenuItem(".pdf");
        mItem5 = new MenuItem(".docx");
        //Adding menu items to the submenu
        menu2.add(mItem4);
        menu2.add(mItem5);
        //adding submenu to menu
        menu1.add(menu2);
        //Adding menu to the menu bar
        menuBar.add(menu1);
    }
}
```

```
//Adding my menu bar to the frame by calling setMenuBar() method
frame.setMenuBar(menuBar);
frame.setSize(330,250);
frame.setVisible(true);
}
}
```

Output



Review Question

1. Write a program in Java AWT to create Menu and Menu items.

2.16 Programming Examples based on AWT Components and Event Handling

Example 2.16.1 Develop an applet that receives two numerical values as input from the user and then displays the sum of these numbers on the screen. Write the HTML code that calls the applet.

Solution :

Step 1 :

test.html

```
<html>
<body>
<applet code="NumOperations" width=300 height=300>
</applet>
</body>
</html>
```

Step 2 :**NumOperations.java**

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

public class NumOperations extends Applet implements ActionListener
{
    Label L1,L2,L3;
    TextField T1,T2,T3;
    Button B1;
    public void init()
    {

        L1=new Label("Enter Num1 :");
        add(L1);
        T1=new TextField(15);           //TextField for Num1
        add(T1);

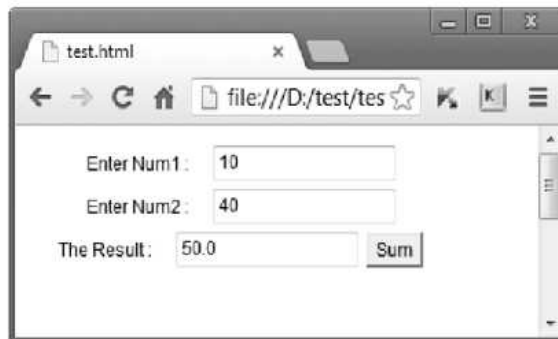
        L2=new Label("Enter Num2 :");
        add(L2);
        T2=new TextField(15);           //TextField for Num2
        add(T2);

        L3=new Label("The Result :");
        add(L3);
        T3=new TextField(15);           //TextField for result
        add(T3);

        B1=new Button("Sum");
        add(B1);                       //Button to invoke addition operation
        B1.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource()==B1)
        {
            int a=Integer.parseInt(T1.getText());
            int b=Integer.parseInt(T2.getText());
            Float c=Float.valueOf(a+b);
            T3.setText(String.valueOf(c));
        }
    }
}
```

Output

Open **test.html** file on web browser.



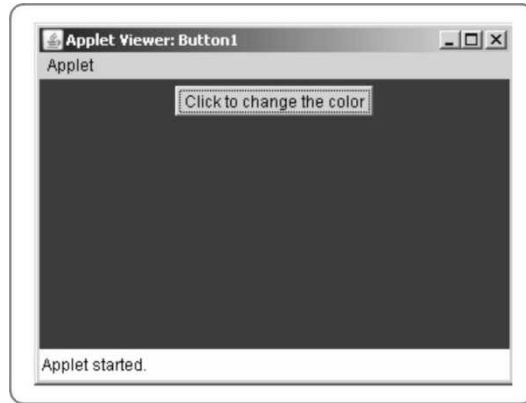
Example 2.16.2 Write applet program to that alternatively changes the background color after every click of button.

Solution :

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Button1" width=350 height=200>
</applet>
*/
public class Button1 extends Applet
{
    Button button=new Button("Click to change the color");
    boolean flag=true;
    public void init()
    {
        add(button);
    }
    public void paint(Graphics g)
    {
        if(flag)
            setBackground(Color.yellow);
        else
            setBackground(Color.red);
    }
    public boolean action(Event e,Object o)
    {
        if(e.target==button)
        {
```

```
flag=!flag;
//toggle the flag values on every click of button
repaint();
return true;
}
return false;
}
}
```

Output



Program Explanation

In above program,

- First of all we have created a button object 'button' using class **Button**. The string "Click to change the color" is written on the button.

```
Button button=new Button("Click to change the color");
```

- Then in the **init** method we have added button using **add()**.
- Then there is a special method called **action()** which is used to deal with the button clicks. There is no specific call to this method rather it is called automatically when you press a button.
- There are two parameters that are passed to the **action** method, first parameter is an object e of type **Event** and other one is an object of type **Object**.
- The property **target** of the **Event** e is compared with the **button**.

```
if(e.target==button)
```

- This helps in finding whether the button is pressed or not.
- We have maintained one variable **flag** which is complemented on each button click. However simply toggling the value of the variable is not sufficient to change the color of the screen, that is why we have called the paint method repeatedly using **repaint()** method.

- In **paint()** method red and yellow background colors are set.

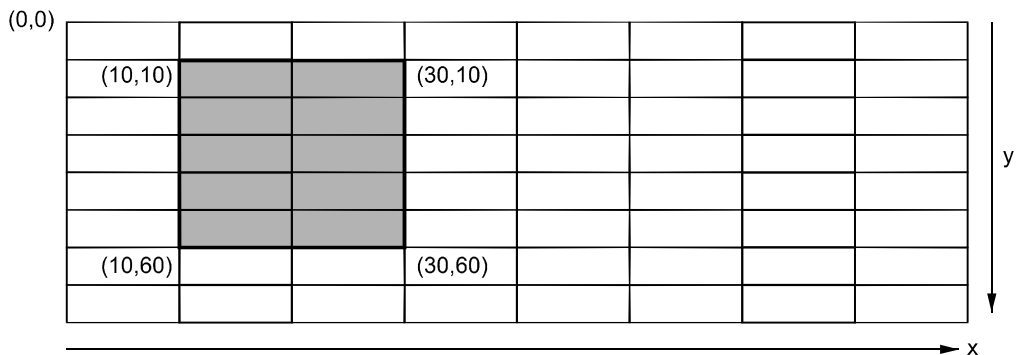
Example 2.16.3 Write a Java program to create AWT radio buttons using check box group.
Explain various event listener interface.

Solution :

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="checkgroup" width=300 height=250>
</applet>
*/
public class checkgroup extends Applet implements ItemListener
{
String msg=" ";
CheckboxGroup gr=new CheckboxGroup();
Checkbox box1=new Checkbox("Candy",gr,true);
Checkbox box2=new Checkbox("Ice-cream",gr,false);
Checkbox box3=new Checkbox("Juice",gr,false);
public void init()
{
=
add(box1);
add(box2);
add(box3);
box1.addItemListener(this);
box2.addItemListener(this);
box3.addItemListener(this);
}
public void itemStateChanged(ItemEvent e)
{
repaint();
}
public void paint(Graphics g)
{
msg="I like ";
msg +=gr.getSelectedCheckbox().getLabel();
g.drawString(msg,10,100);
}
}
```

Output**2.17 Graphics**

- Java has an ability to draw various graphical shapes. The applet can be used to draw these shapes.
- These objects can be colored using various colors.
- Every applet has its own area which is called **canvas** on which the display can be created.
- The co-ordinate system of Java can be represented by following Fig. 2.17.1.

**Fig. 2.17.1 Co-ordinate system**

- Before drawing the 2D shapes we will need a special area on the frame. This area is called **canvas**. Various methods of this class are -

void setSize(int width,int height) - Sets the size of the canvas of given width and height.

void setBackground(Color c) - Sets the background color of the canvas.

void setForeground(Color c) - Sets the color of the text.

2.17.1 Lines

- Drawing the line is the simplest thing in Java. The syntax is,

```
void drawLine(int x1,int y1,int x2,int y2);
```

Where x1 and y1 represents the starting point of the line and x2 and y2 represents the ending point of the line.

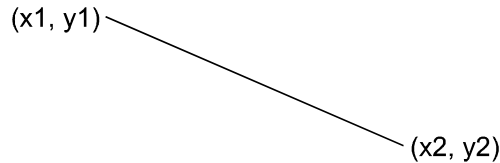
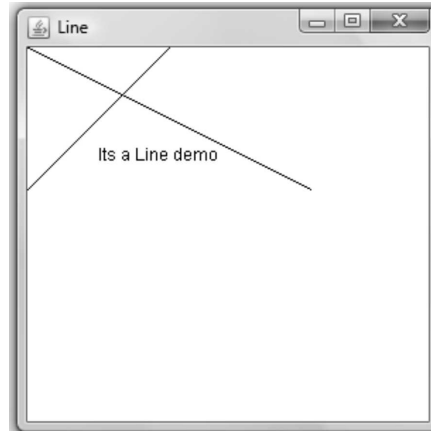


Fig. 2.17.2 Line

Here is a demonstration -

Java Program[LineDemo.java]

```
import java.awt.*;
class LineDemo extends Canvas
{
    public LineDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        LineDemo obj=new LineDemo();
        Frame fr=new Frame("Line");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawLine(0,0,200,100);//diagonal line from top-left
        g.drawLine(0,100,100,0);//another diagonal line
        g.drawString("Its a Line demo",50,80);
    }
}
```

Output**2.17.2 Rectangle**

In Java we can draw **two types of rectangles** : normal rectangle and the rectangle with round corners. The syntax of these methods are -

```
void drawRect(int top,int left,int width,int height)
```

```
void drawRoundRect(int top,int left,int width,int height,int xdimeter,int ydimeter)
```

For example

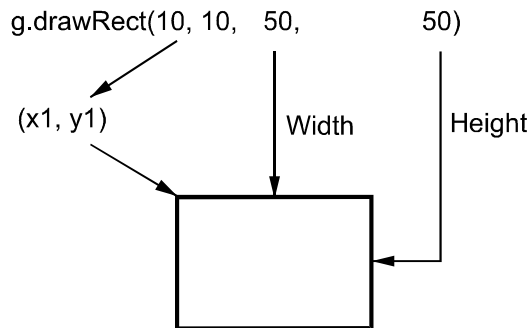


Fig. 2.17.3 Rectangle

The rectangle can be filled up with following methods,

```
void fillRect(int top,int left,int width,int height)
```

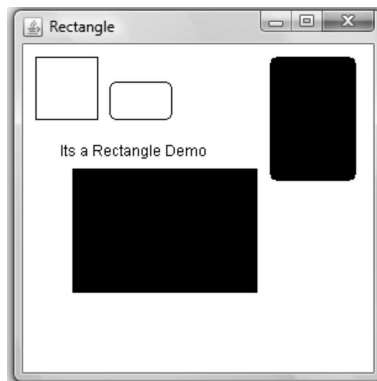
```
void fillRoundRect(int top,int left,int width,int height,int xdimeter,int ydimeter)
```

Java Program[RectangleDemo.java]

```
import java.awt.*;  
class RectangleDemo extends Canvas  
{  
    public RectangleDemo()  
    {
```

```
setSize(200,200);
setBackground(Color.white);
}
public static void main(String[] args)
{
    RectangleDemo obj=new RectangleDemo();
    Frame fr=new Frame("Rectangle");
    fr.setSize(300,300);
    fr.add(obj);
    fr.setVisible(true);
}
public void paint(Graphics g)
{
    g.drawRect(10,10,50,50);
    g.drawRoundRect(70,30,50,30,10,10);
    g.fillRect(40,100,150,100);
    g.fillRoundRect(200,10,70,100,10,10);
    g.drawString("Its a Rectangle Demo",30,90);
}
}
```

Output



2.17.3 Oval

To draw circle and ellipse we can use the function **drawOval()** method. The syntax of this method is ,

```
void drawOval(int top, int left, int width, int height)
```

To fill this oval we use **fillOval()** method. The syntax of this method is,

```
void fillOval(int top, int left, int width, int height)
```

We can specify the color for filling up the rectangle using **setColor** method. For example

```
g.drawOval(10,10,200,100);
g.setColor(Color.blue);
g.fillOval(30,50,200,100);
```

The *top* and *left* values specify the upper left corner and *width* and *height* is for specifying the width and heights respectively.

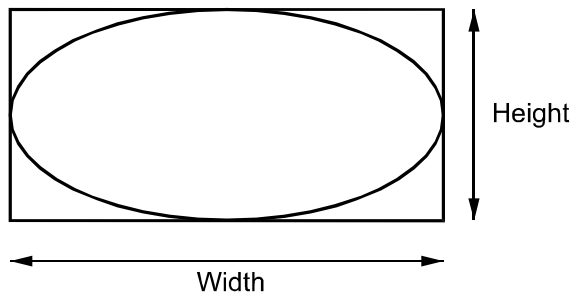
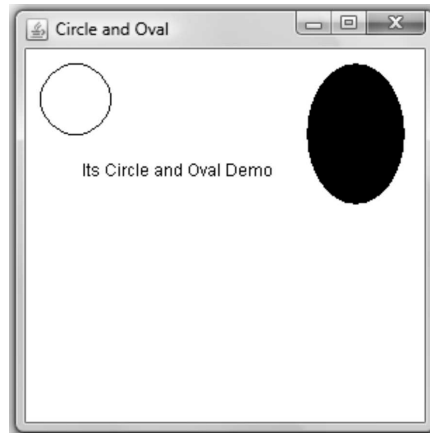


Fig. 2.17.4 Ellipse

Let us understand the functioning of these methods with the help of some example

Java Program[OvalDemo.java]

```
import java.awt.*;
class OvalDemo extends Canvas
{
    public OvalDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        OvalDemo obj=new OvalDemo();
        Frame fr=new Frame("Circle and Oval");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawOval(10,10,50,50);
        g.fillOval(200,10,70,100);
        g.drawString("Its Circle and Oval Demo",40,90);
    }
}
```

Output**2.17.4 Arc**

To draw an arc the **drawArc()** and to fill an arc **fillArc()** are the functions. The syntax of these methods is as follows -

```
void drawArc(int top,int left,int width,int height,int angle1,int angle2)
```

```
void fillArc(int top,int left,int width,int height,int angle1,int angle2)
```

The *angle1* represents the starting angle and the *angle2* represents the angular distance.

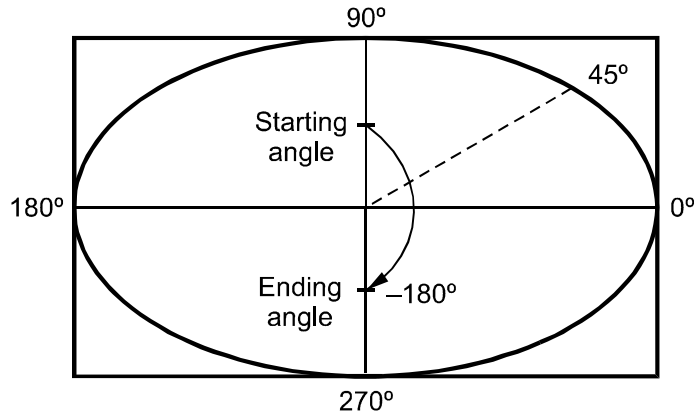


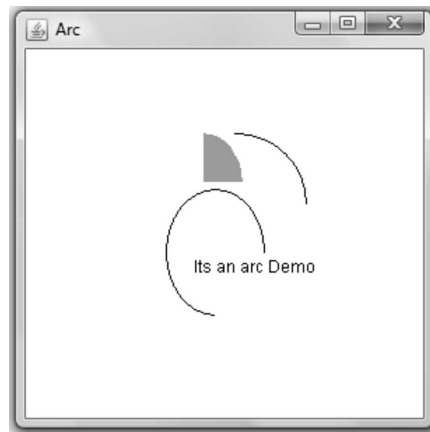
Fig. 2.17.5

Java Program[arcDemo.java]

```
import java.awt.*;
class arcDemo extends Canvas
{
    public arcDemo()
    {
        setSize(200,200);
    }
}
```

```
setBackground(Color.white);
}
public static void main(String[] args)
{
    arcDemo obj=new arcDemo();
    Frame fr=new Frame("Arc");
    fr.setSize(300,300);
    fr.add(obj);
    fr.setVisible(true);
}
public void paint(Graphics g)
{
    g.drawArc(100,60,100,100,0,90);
    g.setColor(Color.green);//fills the arc with green
    g.fillArc(100,60,55,70,0,90);
    g.setColor(Color.black);
    g.drawArc(100,100,70,90,0,270);
    g.drawString("Its an arc Demo",120,160);
}
}
```

Output



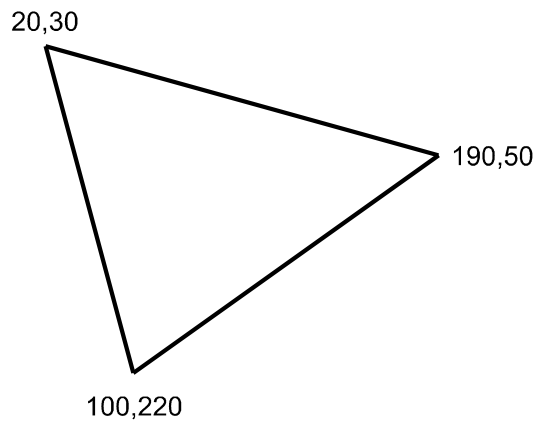
2.17.5 Polygons

In order to draw polygons following function is used

```
Polygon(int[] xpoints,int[] ypoints,int npoints)
```

The *xpoints* represent the array of *x* co-ordinates. The *ypoints* represent the array of *y* co-ordinates. And the *npoints* represents the number of points.

For filling up the polygon the **fillPolygon** method is used.



Java Program[Polyg.java]

```
import java.awt.*;
class Polyg extends Canvas
{
    public Polyg()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        Polyg obj=new Polyg();
        Frame fr=new Frame("Polygon");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        int xpt[]={50,20,20,20,130};
        int ypt[]={80,30,200,200,30};
        int num=5;
        g.drawPolygon(xpt,ypt,num);
        g.setColor(Color.magenta);
        g.fillPolygon(xpt,ypt,num);
        g.setColor(Color.black);
        g.drawString("Its a polygon Demo",100,100);
    }
}
```

Output**Review Questions**

1. List the methods available in the draw shapes.
2. Write the short note on - Graphics programming.
3. How will you draw the following graphics in a window ?
i) Arcs ii) Ellipses and circles in Java.
4. With an example describe in detail about how to work with 2D Shapes in Java.

2.18 Layout Manager**Definition :**

- A **Layout manager** is an interface which automatically arranges the controls on the screen.
- Thus using layout manager the **symmetric** and **systematic arrangement** of the **controls** is possible. In this section we will discuss following layout managers.

2.18.1 FlowLayout

- FlowLayout manager is the simplest Layout manager.
- Using this Layout manager components are arranged from top left corner lying down from left to right and top to bottom.
- Between each component there is some space left.
- The **syntax** of FlowLayout manager is as given below -

```
FlowLayout(int alignment)
```

Where *alignment* denotes the alignment of the components on the applet windows. The alignment can be denoted as :

FlowLayout.LEFT
FlowLayout.RIGHT
FlowLayout.CENTER

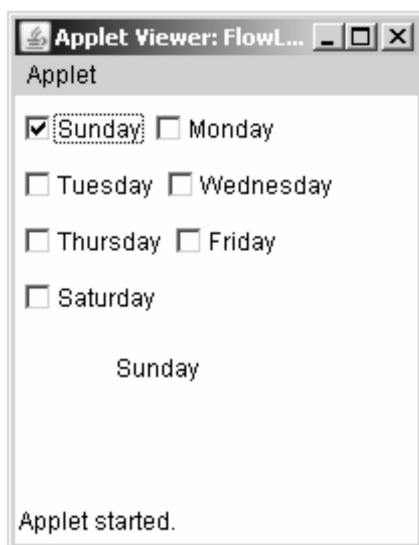
- Here is a Java program which makes use of seven checkboxes which are aligned on the applet window using FlowLayout manager.

Java Program[ItemListener.java]

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="FlowLDemo" width=200 height=200>
</applet>
*/
public class FlowLDemo extends Applet
implements ItemListener
{
String msg=" ";
Checkbox box1=new Checkbox("Sunday");
Checkbox box2=new Checkbox("Monday");
Checkbox box3=new Checkbox("Tuesday");
Checkbox box4=new Checkbox("Wednesday");
Checkbox box5=new Checkbox("Thursday");
Checkbox box6=new Checkbox("Friday");
Checkbox box7=new Checkbox("Saturday");
public void init()
{
//creating FlowLayout manager
setLayout(new FlowLayout(FlowLayout.LEFT));
//adding the components with Left alignment
add(box1);
add(box2);
add(box3);
add(box4);
add(box5);
add(box6);
add(box7);
//registering the checkboxes to EventListener
box1.addItemListener(this);
box2.addItemListener(this);
box3.addItemListener(this);
box4.addItemListener(this);
box5.addItemListener(this);
```

```
box6.addItemListener(this);
box7.addItemListener(this);
}
public void paint(Graphics g)
{
    //if box1 checkbox is clicked
    if(box1.getState())
        msg="Sunday";//then print the corresponding day
    if(box2.getState())
        msg="Monday ";
    if(box3.getState())
        msg="Tuesday ";
    if(box4.getState())
        msg="Wednesday ";
    if(box5.getState())
        msg="Thursday ";
    if(box6.getState())
        msg="Friday ";
    if(box7.getState())
        msg="Saturday ";
    g.drawString(msg,50,140);
}
public void itemStateChanged(ItemEvent e)
{
    repaint();
}
}
```

Output



Program Explanation :

In above program, since we are using Check box control the **ItemListener** interface is used. And for registering these controls to receive the events we have written following lines in the **init** method.

```
box1.addItemListener(this);
box2.addItemListener(this);
box3.addItemListener(this);
box4.addItemListener(this);
box5.addItemListener(this);
box6.addItemListener(this);
box7.addItemListener(this);
```

We have written a method **public void itemStateChanged (ItemEvent e)** because **ItemListener** interface is used.

2.18.2 BorderLayout

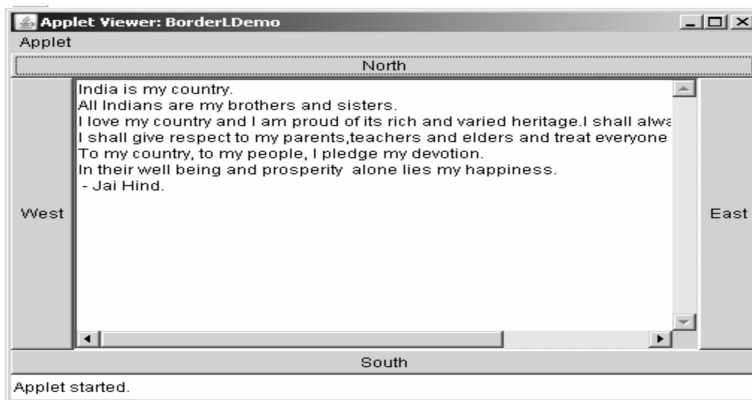
- In BorderLayout there are **four components** at the four sides and one component occupying large area at the centre.
- The central area is called CENTER and the components forming four sides are called LEFT,RIGHT, TOP and BOTTOM.
- Following program consists of one big message stored in variable **msg** which is to be displayed at the centre. And in the **init** method it shows that four sides are occupied by the **Buttons**.

Java Program[BorderLDemo.java]

```
import java.applet.*;
import java.awt.*;
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
String msg="India is my country.\n"+"All Indians are my
    brothers and sisters.\n"+
    "I love my country and I am proud of its rich
    and varied heritage."+
    "I shall always strive to be worthy of it.\n"+
    "I shall give respect to my parents,teachers and
    elders and treat everyone with courtesy.\n"+
```

```
"To my country, to my people, I pledge my
devotion.\n"+
"In their well being and prosperity alone lies my
happiness.\n"+
" - Jai Hind.";
public void init()
{
setLayout(new BorderLayout());
add(new Button("North"),BorderLayout.NORTH);
add(new Button("South"),BorderLayout.SOUTH);
add(new Button("East"),BorderLayout.EAST);
add(new Button("West"),BorderLayout.WEST);
add(new TextArea(msg),BorderLayout.CENTER);
}
}
```

Output



Program Explanation :

In the above program, we have created object for BorderLayout manager using
`setLayout(new BorderLayout());`

Then using

```
BORDER.NORTH
BORDER.SOUTH
BORDER.EAST
BORDER.WEST
```

- The four sides are set with the help of **Button** control. The central large area is formed using **TextArea** control which is called as **BORDER.CENTER**.
- The concept of BorderLayout can then be clearly understood with the help of above given output.

- In this Layout manager, we can add one more method called **getInsets()**. This method allows us to leave some space between underlying window on the applet and Layout manager.
- We have used this method in the following program. The syntax of **Insets** method is

```
Insets(int top,int left,int bottom,int right)
```

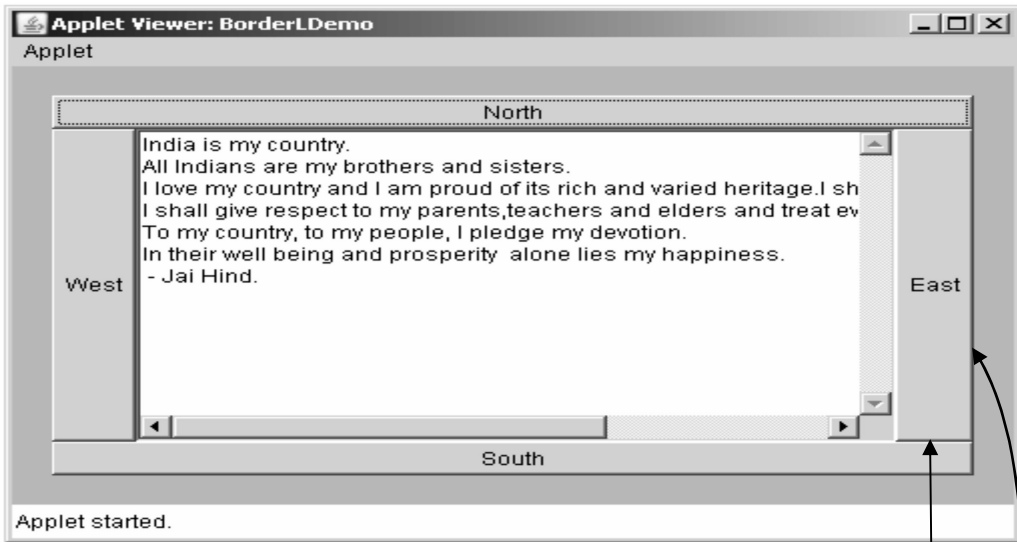
The *top, left, bottom* and *right* parameters specify the amount of space to be left.

Java Program

```
import java.applet.*;
import java.awt.*;
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
String msg="India is my country.\n"+"All Indians are my
    brothers and sisters.\n"+
    "I love my country and I am proud of its rich
    and varied heritage."+
    "I shall always strive to be worthy of it.\n"+
    "I shall give respect to my parents,teachers
    and elders and treat everyone with
    courtesy.\n"+
    "To my country, to my people, I pledge my
    devotion.\n"+
    "In their well being and prosperity alone lies
    my happiness.\n"+
    "- Jai Hind.";
public void init()
{
setBackground(Color.green);
setLayout(new BorderLayout());
add(new Button("North"),BorderLayout.NORTH);
add(new Button("South"),BorderLayout.SOUTH);
add(new Button("East"),BorderLayout.EAST);
add(new Button("West"),BorderLayout.WEST);
add(new TextArea(msg),BorderLayout.CENTER);
}
public Insets getInsets()
{
return new Insets(20,20,20,20);
}
```

```
}
}
```

Output



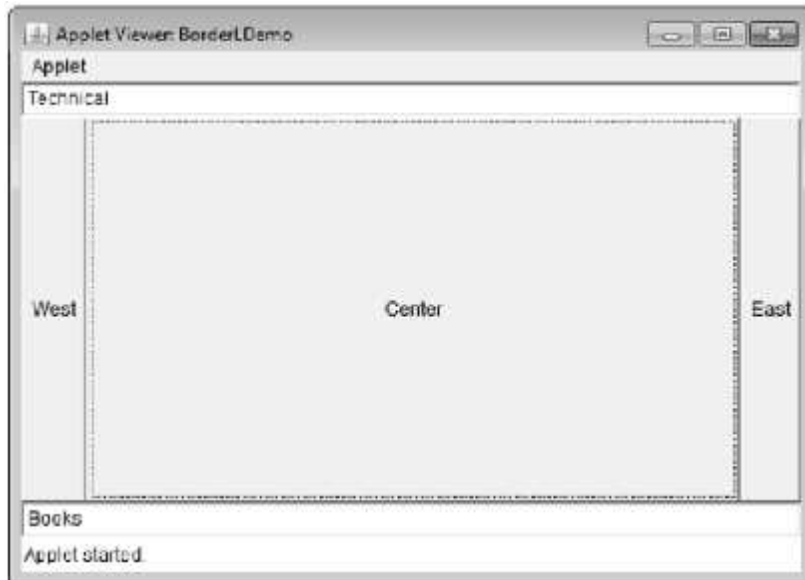
This Space is left in between

Example 2.18.1 Write a Java program which create border layout and adds two text boxes to it.

Solution :

BorderLDemo.java

```
import java.applet.*;
import java.awt.*;
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
    public void init()
    {
        setLayout(new BorderLayout());
        add(new Button("Center"),BorderLayout.CENTER);
        add(new Button("East"),BorderLayout.EAST);
        add(new Button("West"),BorderLayout.WEST);
        add(new TextField("Technical"),BorderLayout.NORTH);
        add(new TextField("Books"),BorderLayout.SOUTH);
    }
}
```

Output**2.18.3 GridLayout**

- GridLayout is a Layout manager used to arrange the components in a grid. The syntax of GridLayout manager is

```
GridLayout(int n,int m)
```

Where n represents total number of rows and m represents total number of columns.

- In the following program we have arranged **Buttons** in a grid form.

Java Program[GridLDemo.java]

```
import java.awt.*;
import java.applet.*;
/*
<applet code="GridLDemo" width=400 height=400>
</applet>
*/
public class GridLDemo extends Applet
{
    int n=4,m=3;
    public void init()
    {
        setLayout(new GridLayout(n,n));
        for(int i=0;i<n;i++)
```

```
{
for(int j=0;j<m;j++)
{
switch(i)
{
case 0:if(j==0) //Button[0,0]
add(new Button("Red"));
else if(j==1) //Button[0,1]
add(new Button("Green"));
else if(j==2) //Button[0,2]
add(new Button("Blue"));
break;
case 1:if(j==0) //Button[1,0]
add(new Button("Orange"));
else if(j==1) //Button[1,1]
add(new Button("Pink"));
else if(j==2) //Button[1,2]
add(new Button("Magenta"));
break;
case 2:if(j==0) //Button[2,0]
add(new Button("Cyan"));
else if(j==1) //Button[2,1]
add(new Button("Gray"));
else if(j==2) //Button[2,2]
add(new Button("Yellow"));
break;
case 3:if(j==0) //Button[3,0]
add(new Button("Black"));
else if(j==1) //Button[3,1]
add(new Button("White"));
else if(j==2) //Button[3,2]
add(new Button("LightGray"));
break;
} //end of switch
} //end of inner for
} //end of outer for
} // end of function init
} //end for class
```


Output**2.18.4 CardLayout**

- Sometimes we want to perform various sets of graphical controls at a time then CardLayout is used.
- Thus CardLayout manager allows us to have more than one layouts on the applet.
- The CardLayout is conceptually thought as a collection of cards lying on a **panel**.
- We have to follow following steps -

Step 1 : We have to create two objects

1. Panel object
2. CardLayout object

Step 2 : Then we have to add the cards on the panel using **add()** method. For example -

```
panel_obj.setLayout(layout_obj);
```

where *panel_obj* is an object of panel and *layout_obj* is an object of CardLayout

Step 3 : Finally we have to add the object of panel to main applet. For example -

```
add(panel_obj);
```

These all stages seem to be complicated. Hence let us understand following program which implements CardLayout.

Java Program[cardDemo.java]

```
//This program demonstates cardLayout
//The dynamic selection of fruit/flower/colour can be made
//using cardlayout component
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="cardDemo" width=300 height=100>
</applet>
*/
public class cardDemo extends Applet implements
ActionListener,MouseListener
{
    Checkbox mango,apple,rose,lotus,Red,Green;
    Panel panel_obj;
    CardLayout layout_obj;
    Button fruit,flower,colour;
    public void init()
    {
        fruit=new Button("Fruit");
        flower=new Button("Flower");
        colour=new Button("Colour");
        //adding the button controls
        add(fruit);
        add(flower);
        add(colour);
        //getting object of Cardlayout
        layout_obj=new CardLayout();
        //getting object of Panel
        panel_obj=new Panel();
        panel_obj.setLayout(layout_obj);
        //adding checkbox controls for fruits
        mango=new Checkbox("Mango");
        apple=new Checkbox("Apple");
        //adding checkbox controls for flowers
        rose=new Checkbox("Rose");
        lotus=new Checkbox("Lotus");
        //adding checkbox controls for colors
```

```
Red=new Checkbox("Red");
Green=new Checkbox("Green");

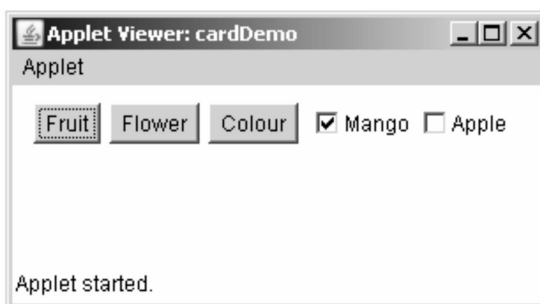
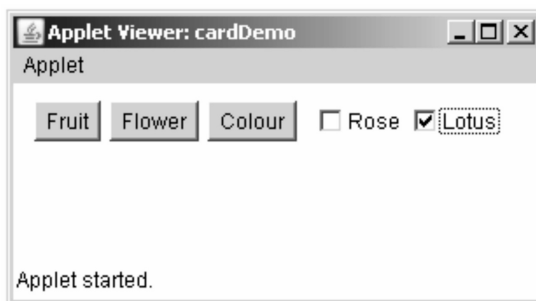
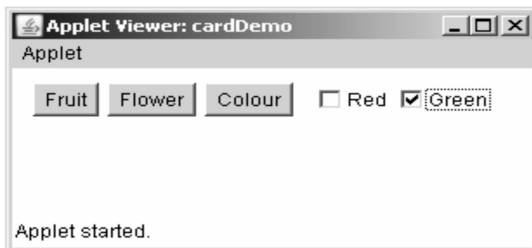
Panel fruit_pan=new Panel();
fruit_pan.add(mango);
fruit_pan.add(apple);

Panel flower_pan=new Panel();
flower_pan.add(rose);
flower_pan.add(lotus);

Panel colour_pan=new Panel();
colour_pan.add(Red);
colour_pan.add(Green);

panel_obj.add(fruit_pan,"Fruit");
panel_obj.add(flower_pan,"Flower");
panel_obj.add(colour_pan,"Colour");
add(panel_obj);
//register the components to event listener
fruit.addActionListener(this);
flower.addActionListener(this);
colour.addActionListener(this);
addMouseListener(this);
}
//following empty methods are necessary for mouse events
public void mousePressed(MouseEvent m)
{
    layout_obj.next(panel_obj);
}
public void mouseClicked(MouseEvent m)
{
}
public void mouseEntered(MouseEvent m)
{
}
public void mouseExited(MouseEvent m)
{
}
public void mouseReleased(MouseEvent m)
{
}
public void actionPerformed(ActionEvent e)
{
```

```
if(e.getSource()==fruit)
{
    layout_obj.show(panel_obj,"Fruit");
}
else if(e.getSource()==flower)
{
    layout_obj.show(panel_obj,"Flower");
}
else if(e.getSource()==colour)
{
    layout_obj.show(panel_obj,"Colour");
}
} //end for actionPerformed method
} //end of class
```

Output(Run 1)**Output(Run 2)****Output (Run 3)**

Program Explanation :

- It is clear from the output that we can have a combination of various components lying on the same applet and can be invoked as per need. That mean if we click on *Fruit* button then we should get two checkboxes namely : *Mango* and *Apple*.
- If we click on *Flower* button then we should get two checkboxes namely : *Rose* and *Lotus*. Similarly, if we click on *Colour* button we should get two checkboxes namely : *Red* and *Green*.
- In above program, we have used to event listener interfaces : **ActionListener** and **MouseListener**.
- We have created **panel object** and **Layout object**.
- In the **init** method we have first created and added the **Button** controls. Then **CardLayout** is placed on the panel.
- The panel is then set for the applet window.
- Various components such as **checkboxes** and **Buttons** are added to the panel.
- In order to understand mouse events some necessary empty methods are written.
- In the **actionPerformed()** method, on the click of **Fruit** button, two corresponding check boxes are shown. Same is true for **Flower** and **Colour** buttons.

2.18.5 GridBagLayout

- The GridBagLayout is the most flexible and complex layout manager.
- The GridBagLayout manager places the components in rows and columns allowing the components to occupy multiple rows and columns. This is called **display area**.
- GridBagLayout performs three functions using values from the **GridBagConstraints** parameter in the **add()** method.
 1. Grid position, width and height describe the display area using **gridx**, **gridy**, **Gridwidth** and **gridheight** values.
 2. Position within the display area using fill, **ipadx** and **ipady**.
 3. Identifying rows and columns which receive extra space on expansion using **weightx**, and **weighty**.
- The layout can be set as follows

```
Container pane=frame.getContentPane();  
pane.setLayout(new GridBagLayout());
```

- The component can be added as

```
pane.add(component,constraintObject);//pane is a container pane
```

- The following values are then set -
 - **gridx and gridy** : These values denote the integer column and row value of the component.
 - **gridwidth and gridheight** : They denote the number of columns and rows the component occupies.
 - **weightx and weighty** : They denote the extra space occupied by the component horizontally or vertically when the output window is resized.
 - **fill** : The fill value denotes how the component should expand within the display area.

Typical values are -

GridBagConstraints.NONE // Can not expand (Default)

GridBagConstraints.VERTICAL // Expand vertically

GridBagConstraints.HORIZONTAL // Expand horizontally

GridBagConstraints.BOTH // Expand vertically and horizontally

- **ipadx and ipady** : These values denote increase and decrease in horizontal or vertical preferred size of the component. Default value is 0.

Example 2.18.2 Write a simple Java program that illustrates the use of GridBagConstraints.

Solution :

```
import java.awt.*;
import java.applet.*;
/*
<applet code="GridBagLayoutDemo" width=400 height=400>
</applet>
*/
public class GridBagLayoutDemo extends Applet
{
    public void init()
    {
        Button B;
        setLayout(new GridBagConstraints());
        GridBagConstraints gBC = new GridBagConstraints();
        gBC.fill = GridBagConstraints.HORIZONTAL;//placing the components horizontally
        B = new Button("Button 1");//first component
        gBC.weightx = 0.5;
        gBC.gridx = 0;
        gBC.gridy = 0;
        add(B, gBC);

        B = new Button("Button 2");//second component
        gBC.gridx = 2;
        gBC.gridy = 0;
        add(B, gBC);
    }
}
```

```

B = new Button("Button 3"); //third component
gBC.ipady = 40; //This component is broad
gBC.weightx = 0.0;
gBC.gridwidth = 3;
gBC.gridx = 0;
gBC.gridy = 1;
add(B, gBC);

TextField T = new TextField("Hello Friends!!!");//forth component
gBC.ipady = 0;
gBC.weightx = 0.0;
gBC.gridx = 1;
gBC.gridwidth = 2;
gBC.gridy = 2;
T.setEditable(false);//text field is not editable
add(T, gBC);
}
}

```

Output**Before Expanding****After Expanding****Review Questions**

1. What is the function of layout manager ? Describe in detail about the different layout in Java GUL.
2. What is layout management ? State the various types of layout supported by JAVA. Which layout is default one ? Discuss the components of swing.

2.19 Multiple Choice Questions

Q.1 Give the Abbreviation of AWT :

- a Applet Windowing Toolkit b Abstract Windowing Toolkit
 c Absolute Windowing Toolkit d None of these

Q.2 In which places can put the event handling code _____.

- a same class b other class
 c anonymous class d all of the above

Q.3 In Java an event is an _____ which specifies the change of state in the source.

- a class b object
 c int d string

Q.4 The Following steps are required to perform 1) Implement the listener interface and overrides its methods 2) Register the component with the listener

- a Exception Handling b String Handling
 c Event Handling d None of the above

Q.5 Various event classes and listener interfaces for event handling are present in following package :

- a java.awt b java.awt.graphics
 c java.awt.event d None of these

Q.6 The ActionListener interface is used for handling action events by _____.

- a JButton b JCheckbox
 c JMenuItem d All of these

Q.7 Which is the container that doesn't contain title bar and MenuBars. It can have other components like button, textfield etc ?

- a Window b Frame
 c Panel d Container

Q.8 Clicking a mouse button will always generate which event ?

- a MouseButtonEvent b ActionEvent
 c MouseClickEvent d MouseEvent

Q.9 By which method we can set or change the text in a Label in AWT ?

- a setText() b getText()
 c addText() d all of these

UNIT III

3

GUI Programming

Syllabus

Designing Graphical User Interfaces in Java, Components and Containers, Basics of Components, Using Containers, Layout Managers, AWT Components, Adding a Menu to Window, Extending GUI Features Using Swing Components, Java Utilities (java.util Package) The Collection Framework : Collections of Objects, Collection Types, Sets, Sequence, Map, Understanding Hashing, and Use of Array List & Vector.

Contents

- 3.1 *Designing Graphical User Interfaces in Java*
- 3.2 *Components and Containers*
- 3.3 *Basics of Components*
- 3.4 *Extending GUI Features Using Swing Components*
- 3.5 *Java Utilities (java.util Package)*
- 3.6 *The Collection Framework*
- 3.7 *Collections of Objects and Types*
- 3.8 *List Interface*
- 3.9 *Vector*
- 3.10 *Set Interface*
- 3.11 *Map Interface*
- 3.12 *Multiple Choice Questions*

3.1 Designing Graphical User Interfaces in Java

- Swing is another approach of graphical programming in Java.
- Swing creates highly interactive GUI applications.
- It is the most flexible and robust approach.

3.1.1 Difference between AWT and Swing

Sr. No.	AWT	Swing
1.	The Abstract Window ToolKit is a heavy weight component because every graphical unit will invoke the native methods.	The Swing is a light weight component because it's the responsibility of JVM to invoke the native methods.
2.	The look and feel of AWT depends upon platform.	As Swing is based on Model View Controller pattern, the look and feel of swing components in independent of hardware and the operating system.
3.	AWT occupies more memory space.	Swing occupies less memory space.
4.	AWT is less powerful than Swing.	Swing is extension to AWT and many drawbacks of AWT are removed in Swing.

3.2 Components and Containers

The most commonly used component classes are -

Component	Purpose
AbstractButton	Abstract class for the buttons
ButtonGroup	It creates the group of buttons so that they behave in mutually exclusive manner.
JApplet	The swing applet class
JButton	The swing button class
JCheckBox	The swing check box
JComboBox	The swing combo box
JLabel	The swing label component

JRadioButton	The radio button component
JTextArea	The text area component
JTextField	The text field component
JSlider	The slider component

We will learn and understand these components with the help of programming examples.

3.3 Basics of Components

In order to display any JComponent on the GUI, it is necessary to add this component to the container first. If you do not add these components to container then it will not be displayed on the GUI. The swing class component hierarchy is as shown by following Fig. 3.3.1. (See Fig. 3.3.1 on next page).

There are two important features of swing components - Firstly, all the component classes begin with the letter J and secondly all these GUI components are descendant of JComponent class. Let us now learn how to place these components on the GUI.

Review Question

1. List and briefly discuss the swing components in Java.

3.4 Extending GUI Features Using Swing Components

- Swing is a large system. It is one part of Java Foundation Classes (JFC).
- It has got good **look** and **feel**.
- There is a rich collection of swing components that can be used to provide the advanced user interface.
- Various classes that can be used for placing the components are JApplet, JLabel, JTextField, JButton, JCheckBox, JTree, JTabbedPane and so on.

3.4.1 JApplet

The **JApplet** is a fundamental swing class. It extends the Applet class. This is much more powerful than the Applet class. It supports the **pane**. Various panes are content pane, glass pane and root pane. The **add** method can be used to add the **content pane**. The add method of **Container** class can be used to add the components on the GUI.

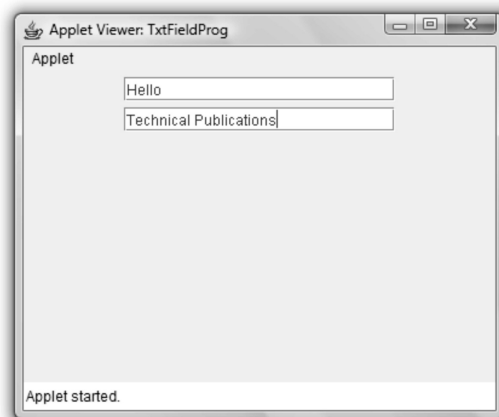
Example program

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="TxtFieldProg" width=300 height=200>
</applet>
*/
public class TxtFieldProg extends JApplet
{
    public void init()
    {
        JTextField T;
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        T=new JTextField("Hello",20);
        contentPane.add(T);
        T=new JTextField(20);
        contentPane.add(T);
    }
}
```

For getting the output of the above program following commands can be given on the command prompt -

```
F:\SwingProg>javac TxtFieldProg.java
F:\SwingProg>AppletViewer TxtFieldProg.java
```

The Applet viewer will display the GUI as follows -

Output

Program explanation

To place the control textfield on the GUI we have followed following steps -

1. Using the **getContentPane** method an object for the **Container** class is created. This object is taken in the variable *contentPane*.

2. The Layout is set using the *contentPane* object by the statement

```
contentPane.setLayout(new FlowLayout());
```

The default layout is **FlowLayout** but you can set other layout managers such as **GridLayout**, **BorderLayout** and so on.

3. After setting the layout manager, the **TextField** component can be created and placed using **add** method. In above program, we have created two text fields. In the first text field, the string "Hello" is already written during its creation but the second text field is kept blank and some string can be written into it during the execution.

3.4.2 Creating Frames

- A Frame is a top-level window with a title and a border. The Frames in java works like the main window where the components like JButton, JLabel, JComboBox and so on can be placed. In Java swing the top-level windows are represented by the **JFrame** class. For creating the frame following statement can be used -

```
JFrame f=new JFrame("My First Frame Program");
```



- The frames are not visible initially, hence we have to make them visible by `f.setVisible(true);`
- The close button of the frame by default performs the hide operation for the JFrame. Hence we have to change this behavior to window close operation by setting the `setDefaultCloseOperation()` to **EXIT_ON_CLOSE** value.
- Any suitable size of the frame can be set by **setSize(int height,int width)** function.

Example 3.4.1 Write a JFrame with a hello world program.

Solution :

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



```
public class FrameProg extends JFrame
{
    public static void main(String[] args)
    {
        new FrameProg();
    }
    FrameProg()
    {
        JFrame f=new JFrame("Frame Demo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
        f.setSize(300,300);
        Container content=f.getContentPane();
        content.setLayout(new FlowLayout());
        content.add(new JLabel("Hello World"));
    }
}
```

Output

3.4.3 Label and ImageIcon

- The icons are encapsulated by **ImageIcon** class. The constructors are -

```
ImageIcon(string fname)
ImageIcon(URL url)
```

The *fname* denotes the name of the file which contains the icon. The *url* denotes the resource of image.

- The **JLabel** is a component for placing the label component. The **JLabel** is a subclass of **JComponent** class. The syntax for the JLabel is -

```
JLabel(Icon ic);
Label(String s);
JLabel(String s, Icon ic, int alignment)
```

- The alignment can be LEFT, RIGHT, CENTER, LEADING and TRADING. The icons and text methods associated with the label are -

```
Icon getIcon()
void setIcon(Icon ic) ← ic represents the icon file
String getText();
void setText(String s); ← s represents the string
```

- Following simple Java Program illustrates the use of JLabel component -

Java Program[LabelProg.java]

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="LabelProg" width=300 height=200>
</applet>
*/
public class LabelProg extends JApplet
{
    public void init()
    {
        Container contentPane=getContentPane();
        ImageIcon i=new ImageIcon("innocence.gif");
        JLabel L1=new JLabel("Innocence",i,JLabel.LEFT);
        contentPane.add(L1);
    }
}
```

Output**3.4.4 TextField**

- The **JTextField** is extended from the **JComponent** class. The **JTextField** allows us to add a single line text.
- The syntax of using **JTextField** is -

```
JTextField();  
JTextField(int col_val);  
JTextField(String s,int col_val);  
JTextField(String s);
```

- The program making use of TextField is as given below -

Java Program[TxtFieldProg.java]

```
import java.awt.*;  
import javax.swing.*;  
/*  
<applet code="TxtFieldProg" width=300 height=200>  
</applet>  
*/  
public class TxtFieldProg extends JApplet  
{  
    public void init()  
    {  
        JTextField T;  
        Container contentPane=getContentPane();  
        contentPane.setLayout(new FlowLayout());
```

```
T=new JTextField("Hello",20);
contentPane.add(T);
T=new JTextField(20);
contentPane.add(T);
}
}
```

- For getting the output of the above program following commands can be given on the command prompt -

```
F:\SwingProg>javac TxtFieldProg.java
```

```
F:\SwingProg>AppletViewer TxtFieldProg.java
```

The Applet viewer will display the GUI as follows -

Output



Program explanation

To place the control textfield on the GUI we have followed following steps -

1. Using the `getContentPane` method an object for the container class is created. This object is taken in the variable `contentPane`.
2. The layout is set using the `contentPane` object by the statement

```
contentPane.setLayout(new FlowLayout());
```

The default layout is `FlowLayout` but you can set other layout managers such as `GridLayout`, `BorderLayout` and so on.

3. After setting the layout manager, the `TextField` component can be created and placed using `add` method. In above program, we have created two text fields. In the first text field, the string "Hello" is already written during its creation but the second text field is kept blank and some string can be written into it during the execution.

3.4.5 TextArea

The `JTextArea` is a GUI control which allows us to write multi-line text. We can set the desired number of rows and number of columns so that a long big message can be written using this control.

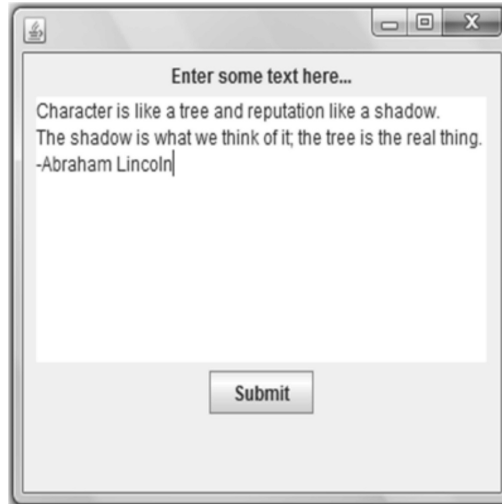
The syntax for JTextArea is

```
JTextArea(String str,int rows,int cols);
```

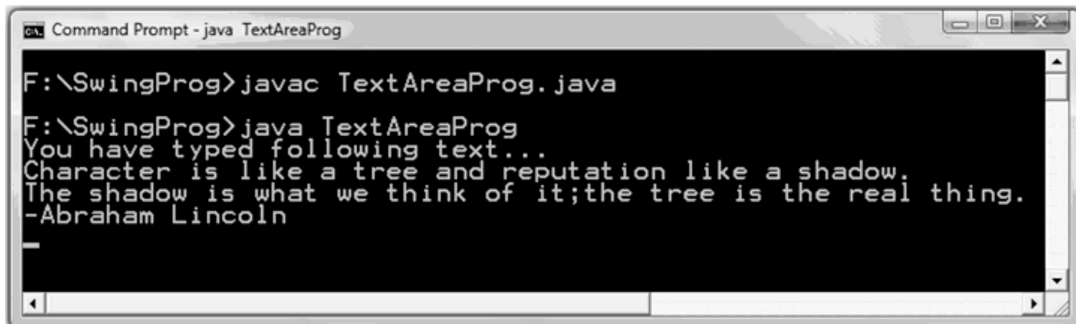
In the following Java program, there are two GUI controls on the window - the textarea and the push button. The idea is that : just type something within the text area and then click the push button. Whatever text you type will get displayed on the console window.

Java Program[TextAreaProg.java]

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class TextAreaProg
implements ActionListener
{
JTextArea TA;
public static void main(String[] args)
{
new TextAreaProg(); //invoke the function for demonstrating TextArea control
}
TextAreaProg() //definition of the function
{
JFrame f=new JFrame(); //create a Frame window
Container content=f.getContentPane();//get the Container class object
content.setLayout(new FlowLayout());//set the layout manager
JLabel L=new JLabel("Enter some text here...");//create a Label control
content.add(L); //using add method add the label on the GUI
TA=new JTextArea("",10,20); //create a TextArea control with rows=10 and col=20
content.add(TA);//using add method add the TextArea control on GUI
JButton B=new JButton("Submit");//create a push button control
content.add(B);//place this control on the GUI using add method
B.addActionListener(this);//invoke action listener for button click event
f.setSize(300,300);//set the size of the frame window
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//exit on closing
f.setVisible(true);//set the visibility of frame as true
}
public void actionPerformed(ActionEvent e)//handling the event when button is clicked
{
String s=TA.getText();//get the string typed in textarea
System.out.println("You have typed following text...\n"+s);//display it on the consol
}
}
```

Output

Now if we click **Submit** button then on the command-prompt window we can see following text-

**3.4.6 Buttons**

- The swing push button is denoted by using **JButton** class.
- The swing button class provides the facilities that can not be provided by the applet button class. For instance you can associate some image file with the button.
- The swing button classes are subclasses of **AbstractButton** class.
- The **AbstractButton** class generates action events when they are pressed. These events can be associated with the Push buttons.
- We can associate an icon and/or string with the **JButton** class. The syntax of JButton is

```
JButton(Icon ic);  
JButton(String s);  
JButton(String s,Icon ic);
```

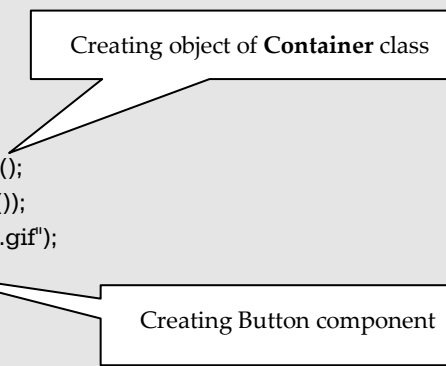
- Following program illustrates the use of Icon and the string for the JButton class -

Java Program[ButtonProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="ButtonProg" width=500 height=300>
</applet>
*/
public class ButtonProg extends JApplet//inherited from JApplet
implements ActionListener
{
    JTextField T;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        ImageIcon apple=new ImageIcon("apple.gif");
        JButton B1=new JButton(apple);
        B1.setActionCommand("Apple");
        B1.addActionListener(this);
        contentPane.add(B1);//adding button on GUI

        ImageIcon orange=new ImageIcon("orange.gif"); //Creating image icon
        JButton B2=new JButton(orange); //associating image with button
        B2.setActionCommand("Orange");
        B2.addActionListener(this); //button pressed event
        contentPane.add(B2);

        ImageIcon grapes=new ImageIcon("grapes.gif");
        JButton B3=new JButton(grapes);
        B3.setActionCommand("Grapes");
        B3.addActionListener(this);
        contentPane.add(B3);
        T=new JTextField(20);
        contentPane.add(T);//placing the textfield on the GUI
    }
    public void actionPerformed(ActionEvent e)
    {
        T.setText(e.getActionCommand()); //retrieving the text associated with button
    }
}
```

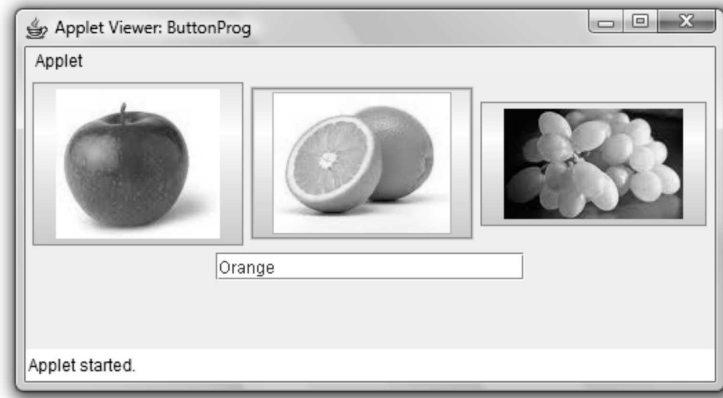


For getting the output open the **command-prompt** and give the following commands -

```
F:\SwingProg>javac ButtonProg.java
F:\SwingProg>AppletViewer ButtonProg.java
```

and you will get the applet as follows -

Output



Program explanation

The above program is inherited from the **JApplet** class. In the init method, we have written the code -

1. Create a container object by using the **getContentPane** method. This object is now in the variable **contentPane**.
2. A layout manager such as **FlowLayout** is used to set the layout of the GUI.
3. The button components are then placed on the GUI using the **add** method.
4. These push buttons are associated with some images using **ImageIcon** class. To this **ImageIcon** class appropriate gif file is passed as an argument. Thus corresponding image gets associated with each corresponding push button.
5. We have associated an **ActionListener** event with this program. This is necessary to handle the events when a push button gets pressed. We have associated some command string when the particular button is pressed. This can be done using **setActionCommand** method. The event handler can be invoked using the methods **addActionListener(this)**. When this a call to this method is given the control goes to the function **actionPerformed**. In this method we are simply displaying the appropriate command string in the textbox.

3.4.7 Checkboxes

- The Check Box is also implementation of **AbstractButton** class. But the immediate superclass of **JCheckBox** is **JToggleButton**.
- The **JCheckBox** supports two states true or false.
- We can associate an icon, string or the state with the checkboxes. The syntax for the Check Box will be -

```
JCheckBox(Icon ic);
JCheckBox(Icon ic, boolean state);
JCheckBox(String s);
JCheckBox(String s,boolean state);
JCheckBox(String s,Icon ic,boolean state);
```

- Following program illustrates the use of check box -

Java Program[CheckBoxProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="CheckBoxProg" width=300 height=300>
</applet>
*/
public class CheckBoxProg extends JApplet
implements ItemListener
{
    JTextField T;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        JCheckBox chk1=new JCheckBox("Apple");
        chk1.addItemListener(this);// invoking the event handler
        contentPane.add(chk1);
        JCheckBox chk2=new JCheckBox("Orange");
        chk2.addItemListener(this); // invoking the event handler
        contentPane.add(chk2);
        JCheckBox chk3=new JCheckBox("Grapes");
        chk3.addItemListener(this); // invoking the event handler
        contentPane.add(chk3);
        T=new JTextField(5);
        contentPane.add(T);
        ButtonGroup bg=new ButtonGroup();
```

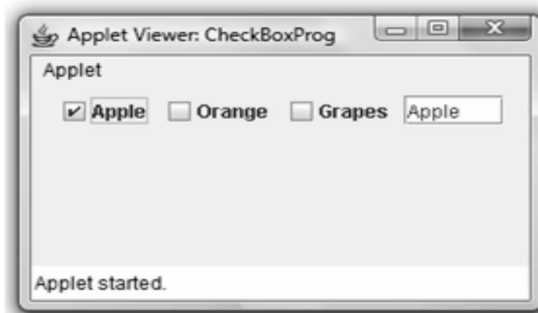
```

    bg.add(chk1);
    bg.add(chk2);
    bg.add(chk3);
}
public void itemStateChanged(ItemEvent e)
{
    JCheckBox chk=(JCheckBox)e.getItem();
    T.setText(chk.getText());//displaying the appropriate string in textbox
}
}

```

ButtonGroup helps the objects to behave mutually exclusive.

Output



Program explanation

In above program, the **ItemListener** event is implemented. Using the **addItemListener(this)** method the event handler function **itemStateChanged** is invoked. Thus on the clicking corresponding checkbox, the appropriate string will be displayed in the text field.

3.4.8 Radio Buttons

- The **JRadioButton** is a subclass of **JToggleButton**. This control is similar to the checkboxes.
- The syntax for the Radio Box will be -

```

JRadioButton(Icon ic);
JRadioButton (Icon ic, boolean state);
JRadioButton (String s);
JRadioButton (String s,boolean state);
JRadioButton (String s,Icon ic,boolean state);

```

- Following program illustrates the use of radio buttons.

Java Program[RadioButProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="RadioButProg" width=300 height=300>
</applet>
*/
public class RadioButProg extends JApplet
implements ActionListener
{
    JTextField T;
    public void init()
    {

        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());

        JRadioButton rb1=new JRadioButton("Apple");
        rb1.addActionListener(this);
        contentPane.add(rb1);

        JRadioButton rb2=new JRadioButton("Orange");
        rb2.addActionListener(this);
        contentPane.add(rb2);
        JRadioButton rb3=new JRadioButton("Grapes");
        rb3.addActionListener(this);

        contentPane.add(rb3);
        T=new JTextField(5);
        contentPane.add(T);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);
        bg.add(rb3);
    }
    public void actionPerformed(ActionEvent e)
    {

        T.setText(e.getActionCommand());
    }
}
```

Output**Program explanation**

We have used **ActionListener** event to handle the radio button click. The **addActionListener** method is invoked when the button gets clicked. The appropriate command string can be displayed in the textfield.

3.4.9 Lists

JList is a component that displays list of text items. User can select one or multiple items from this list.

Various components of JList component are

JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Various methods of this component are -

Method	Description
int getSelectedIndex()	It returns the index of selected item of the list.
void setListData(Object[] list)	It is used to create a read-only ListModel from an array of objects.

Java Program

```
import javax.swing.*;
import java.awt.*;
/*
<applet code="ListDemo" width=200 height=200>
</applet>
*/
public class ListDemo extends JApplet
{
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new BorderLayout());
        String [] str={"Windows98\n","Windows2000\n","Windows7\n","Windows8\n","Windows10"};
        JList list = new JList(str);
        list.setBounds(100,100, 75,75);
        contentPane.add(list);
    }
}
```

Output



3.4.10 Choices

- **JList** and **JComboBox** are very similar components but the **JList** allows to select multiple selections whereas the **JComboBox** allows only one selection at a time.
- A combo box is a combination of text field and the drop down list. The **JComboBox** is a subclass of **JComponent** class.

Java Program[ComboBoxProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="ComboBoxProg" width=300 height=300>
</applet>
*/
public class ComboBoxProg extends JApplet
implements ItemListener
{
    JLabel L;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        JComboBox co=new JComboBox();
        co.addItem("apple");
        co.addItem("orange");
        co.addItem("grapes");
        co.addItemListener(this);
        contentPane.add(co);

        L=new JLabel(new ImageIcon("apple.gif"));
        contentPane.add(L);

    }
    public void itemStateChanged(ItemEvent e)
    {
        String str=(String)e.getItem();
        L.setIcon(new ImageIcon(str+".gif"));//invokes the appropriate image file
    }
}
```

Output

**Program explanation**

Using the **new JComboBox()** the object for the combo box can be created. This object then invokes the method **addItem** for adding the items in the combo box. In the event handler function **itemStateChanged** the **getItem** method is used to retrieve the selected string.

Example 3.4.2 *Create a phonebook look-up application that displays the phone number for the person selected and looks up the name for an entered phone number. Create a GUI that consists of a pull-down list of names, a text field for entry of phone numbers to look up and a command button to activate the look-up operation. It is your job to add the event handling to the application. There are three different events that must be handled. When the user changes the name in the pull-down list, the application should look up the appropriate telephone number in the phonebook. When the user enters a phone number in the text field and presses return or selects the button, the application should look up the number in the reverse phonebook and change the name in the pull-down list. Finally, the application should restrict input in the text field to digits and the '-' character. If a look-up operation fails, the application should simply beep.*

Solution :

```
//Program for handling the telephone Directory using event handling
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="TelephoneDir" width=300 height=300>
</applet>
*/
public class TelephoneDir extends JApplet
implements ItemListener,ActionListener
{
    JTextField T;
    JComboBox co;
    JButton B;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        co=new JComboBox();
        co.addItem("Select an Item");
        co.addItem("Archana");
        co.addItem("Supriya");
        co.addItem("Jayashree");
        co.addItem("Shivraj");
        co.addItem("Sandip");
        contentPane.add(co);
        co.addItemListener(this);
        T=new JTextField(10);
        contentPane.add(T);
        T.addActionListener(this);
        B=new JButton("Submit");
        contentPane.add(B);
        B.addActionListener(this);
    }
    public void itemStateChanged(ItemEvent e)
    {
        String str=(String)e.getItem();
        if(str=="Archana")
            T.setText("11-11111111");
        else if(str=="Supriya")
            T.setText("11-22222222");
        else if(str=="Jayashree")
```

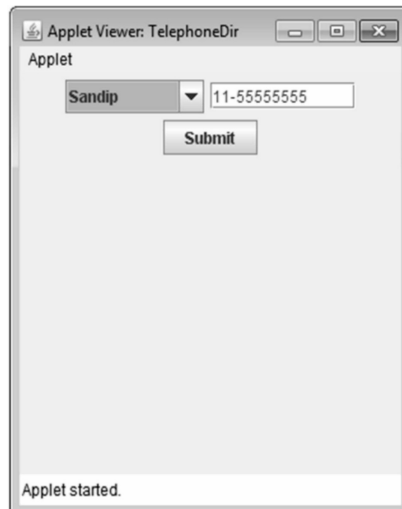


```

T.setText("11-33333333");
else if(str == "Shivraj")
T.setText("11-44444444");
else if(str == "Sandip")
T.setText("11-55555555");
else
{
if((str == "") || (T.getText() == ""))//generating Beep
System.out.println("\007");
}
}
public void actionPerformed(ActionEvent e)
{
String str=(String)T.getText();
co.setSelectedItem(str);
if(co.getSelectedItem() == "Select an Item");//no item selected
System.out.println("\007\007");//generating Beep
}
}

```

Output



Example 3.4.3 Write a program to create product enquiry form using frames.

Solution :

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrameProg extends JFrame
{

```

```
public static void main(String[] args)
{
    new FrameProg();
}
FrameProg()
{
    JFrame f=new JFrame("Enquiry Form");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
    f.setSize(800,300);
    Container content=f.getContentPane();
    content.setLayout(new FlowLayout(FlowLayout.CENTER));
    content.add(new JLabel("Name:"));
    content.add(new JTextField(10));
    content.add(new JLabel("Address:"));
    content.add(new JTextField(20));
    content.add(new JLabel("Product:"));
    JComboBox co=new JComboBox();
    co.addItem("Mobile Phones");
    co.addItem("Laptops");
    co.addItem("ipods");
    co.addItem("Tablet PC");
    content.add(co);
    JButton b=new JButton("Submit");
    content.add(b);
}
}
```

Example 3.4.4 *How will you display an image on the frame in a window using java ?*

Solution :

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

        JFrame frame = new JFrame("Displaying Image On Frame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        frame.setResizable(false);
        frame.setLocationRelativeTo(null);
        // Inserts the image icon
```

```
ImageIcon image = new ImageIcon("img1.jpg");
JLabel label1 = new JLabel(" ", image, JLabel.CENTER);
frame.getContentPane().add(label1);
frame.validate();
frame.setVisible(true);
}
public static void main(String[] args)
{
    ImgFrameDemo obj = new ImgFrameDemo();
    obj.display();
}
}
```

Output



3.4.11 ScrollPane

- The ScrollPane is a rectangular area in which some component can be placed.
- The component can be viewed with the help of **horizontal** and **vertical** scroll bars.
- Using the **JScrollPane** class the component can be added in the program.
- The **JScrollPane** class extends the **JComponent** class.
- There are **three constructors** that can be used for this component -

```
JScrollPane(Component component)
```

```
JScrollPane(int vscrollbar, int hscrollbar)
```

```
JScrollPane(Component component, int vscrollbar, int hscrollbar)
```

The *component* represents the reference to the component. The *vscrollbar* and *hscrollbar* are the integer values for the vertical and horizontal scroll bars. These values can be defined by the constants such as

Constant	Meaning
HORIZONTAL_SCROLLBAR_ALWAYS	It always displays the horizontal scroll bar.
HORIZONTAL_SCROLLBAR_NEEDED	It displays the horizontal scrollbar if required.
VERTICAL_SCROLLBAR_ALWAYS	It always displays the vertical scroll bar.
VERTICAL_SCROLLBAR_NEEDED	It displays the vertical scrollbar if required.

- Following is a simple program which illustrates the use of scrollpane.

Step 1 : Create a label.

Step 2 : Create a panel.

Step 3 : Use an image with the help of label.

Step 4 : Add the label on the panel.

Step 5 : Create a content pane.

Step 6 : Create a scrollpane component by passing the image (within a panel) as a *component* to it.

Step 7 : Add the the scrollpane component to the content pane component.

Java Program

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="ScrollPaneDemo" width=150 height=150>
</applet>
*/
public class ScrollPaneDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        JPanel mypanel = new JPanel();
        JLabel L1 = new JLabel();
        ImageIcon i = new ImageIcon("img.jpg");
        L1.setLocation(20, 100);
        L1.setSize(120, 120);
        L1.setIcon(i);
        mypanel.add(L1);
    }
}
```

```

int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(mypanel, vscrollbar, hscrollbar);
contentPane.add(jsp, BorderLayout.CENTER);
}
}

```

Output



Example 3.4.5 *List does not support scrolling. Why? How this can be remedied? Explain with an example.*

Solution : List is a component which simply displays the list of items. But this component does not support scrolling. The scrolling facility can be added to this component by using the scroll pane. Following are the steps that can be carried out for that purpose -

Step 1 : Create the content pane.

Step 2 : Create a list of items. (Preferably list of lots of items)

Step 3 : Create a scroll pane component by setting the values to vertical and horizontal component `ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED` and `ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED`;

Step 4 : Then pass the JList component to scroll pane component.

Step 5 : Then add this scroll pane component to content pane.

Following program illustrates this idea -

Java Program

```

import javax.swing.*.*;
import java.awt.*.*;
import javax.swing.tree.*;

```

```

/*
< applet code="ListDemo" width=150 height=90>
</ applet>
*/

public class ListDemo extends JApplet
{
public void init()
{
Container contentPane = getContentPane();

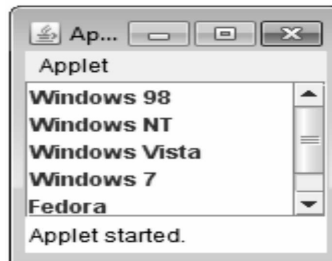
contentPane.setLayout(new BorderLayout());
String[] str = { "Windows 98\n", "Windows NT\n", "Windows Vista\n", "Windows
7\n", "Fedora\n", "Ubuntu"};
JList list = new JList(str);

int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

JScrollPane mypane = new JScrollPane(list, vscrollbar, hscrollbar);
contentPane.add(mypane, BorderLayout.CENTER);
}
}

```

Output



3.4.12 Scrollbar

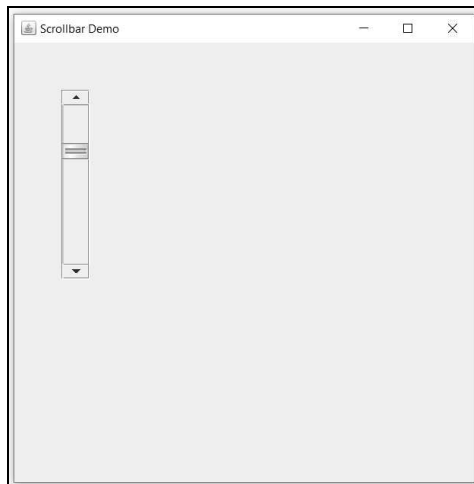
- The JScrollPane class is used to implement the horizontal and vertical scrollbars. This class is inherited from JComponent class.
- The commonly used constructors are :

JScrollbar()	Creates a vertical scrollbar with the initial values.
JScrollbar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollbar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum and maximum.

- Following is a simple java program that displays the scrollbar component.

```
import javax.swing.*;
class ScrollBarDemo
{
    ScrollBarDemo()
    {
        JFrame fr= new JFrame("Scrollbar Demo");
        JScrollBar sb=new JScrollBar();
        sb.setBounds(50,50,30,200);
        fr.add(sb);
        fr.setSize(500,500);
        fr.setLayout(null);
        fr.setVisible(true);
    }
    public static void main(String args[])
    {
        new ScrollBarDemo();
    }
}
```

Output



Difference between Scrollbar and ScrollPane :

- 1) Scrollbar is a component whereas scrollPane is container.
- 2) Scrollbar cannot handle its own events whereas the ScrollPane can handle its events.
- 3) Scrollbar can not contain ScrollPane but scrollPane can have scrollbar.

3.4.13 Menus

- The menu bar is the most standard GUI which is present in almost all the applications.

- We can create a menu bar which contains several menus.
- Each menu can have several menu items.
- The separator can also be used to separate out these menus.
- Various methods and APIs that can be used for creating the Menus in Swing are -

JMenuBar

This class creates a menu bar. A menu bar normally contains several menus.

JMenu(String *str*)

This class creates several menus. The menus contain the menu items. The string *str* denotes the names of the menus.

JMenuItem(String *str*)

The menu items are the parts of menus. The string *str* denotes the names of the menu items.

setMnemonic(char *ch*)

The mnemonic key can be set using this method. The character which is passed to it as an argument becomes the mnemonic key. Hence using alt + *ch* you can select that particular menu.

JSeparator

This is the constructor of the class JSeparator which adds separating line between the menu items.

setJMenuBar

- This method is used to set menu bar to a specific frame.
- In the following Java program, we have created a frame on which a menu bar is placed. The menu bar contains three menus - File, Edit and Help. Each of these menus have their own set of menu items. The separators and mnemonic keys are also used in this program.

Java Program

```
import java.awt.*;
import javax.swing.*;
class MenuProg
{
    public static void main(String[] args)
    {
        JMenu menu1=new JMenu("File"); ← Creating Menu
        menu1.setMnemonic('F');
        menu1.add(new JMenuItem("New"));
        menu1.add(new JMenuItem("Open"));
        menu1.add(new JMenuItem("Save"));
        menu1.add(new JMenuItem("Save as..."));
    }
}
```

Adding menu items
to menus


```

menu1.add(new JSeparator());
menu1.add(new JMenuItem("Close"));
JMenu menu2=new JMenu("Edit");
menu2.setMnemonic('E');
menu2.add(new JMenuItem("Undo"));
menu2.add(new JMenuItem("Redo"));
menu2.add(new JSeparator());
menu2.add(new JMenuItem("Cut"));
menu2.add(new JMenuItem("Copy"));
menu2.add(new JMenuItem("Paste"));

```

```

JMenu menu3=new JMenu("Help");
menu3.setMnemonic('H');
JMenuBar menubar=new JMenuBar(); ← Creating menubar
menubar.add(menu1);
menubar.add(menu2);
menubar.add(menu3);

```

Adding menus to menubar

```

JFrame f=new JFrame(); ← Creating frames
f.setJMenuBar(menubar); ← placing menubar on the frame
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
f.setSize(200,200);

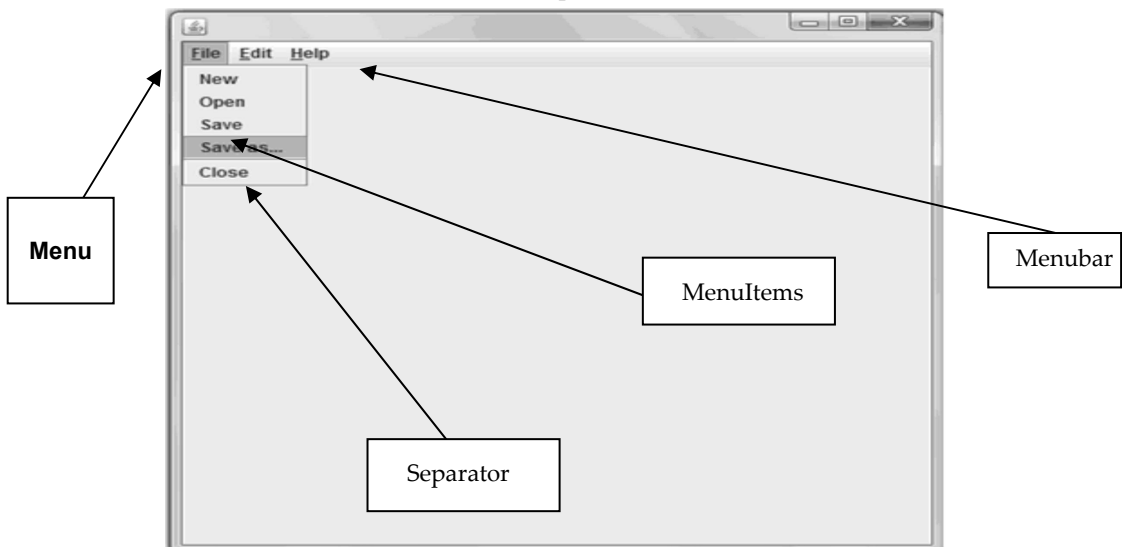
```

```

}

```

Output



In order to try out the mnemonic keys just press Alt+f key and the File menu gets selected.

Example 3.4.6 Write a program to create a frame with the following menus, such that the corresponding geometric object is created when a menu is clicked. i) Circle ii) Rectangle iii) Line iv) Diagonal for the rectangle.

Solution :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class ShapesMenu extends JFrame implements ActionListener
{
    JMenuBar mb;
    JMenu menu;
    JMenuItem rect,line,oval;
    ShapesMenu()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        mb=new JMenuBar();

        menu=new JMenu("Shapes");
        mb.add(menu);

        rect=new JMenuItem("Rectangle");
        rect.addActionListener(this);
        menu.add(rect);

        line=new JMenuItem("Line");
        line.addActionListener(this);
        menu.add(line);

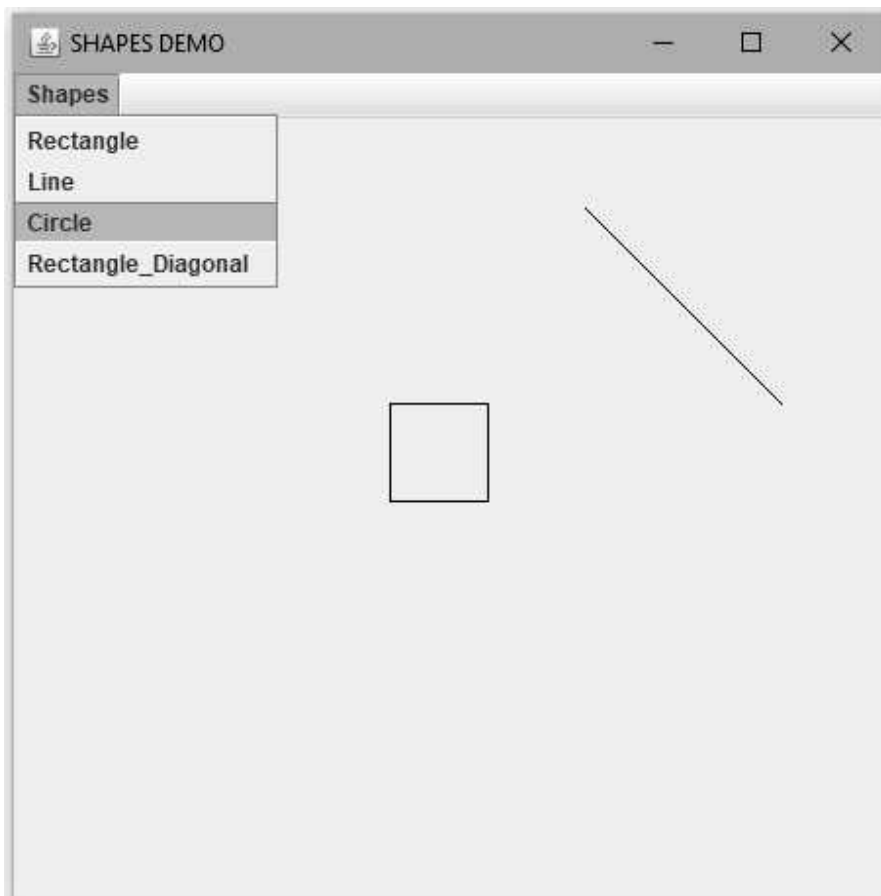
        oval=new JMenuItem("Circle");
        oval.addActionListener(this);
        menu.add(oval);

        line=new JMenuItem("Rectangle_Diagonal");
        line.addActionListener(this);
        menu.add(line);

        setJMenuBar(mb);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String str=ae.getActionCommand();
        Graphics g=getGraphics();
```

```
if(str=="Rectangle")
    g.drawRect(200,200,50,50);
if(str=="Line")
    g.drawLine(300,100,400,200);
if(str=="Circle")
    g.drawOval(200,300,100,100);
if(str=="Rectangle_Diagonal")
    g.drawLine(200,200,250,250);
}
public static void main(String args[])
{
    ShapesMenu f=new ShapesMenu();
    f.setTitle("SHAPES DEMO");
    f.setSize(500,500);
    f.setVisible(true);
}
}
```

Output



Example 3.4.7 *Create a simple menu application that enables a user to select one of the following items :*

- Radio 1*
- Radio 2*
- Radio 3*
- Radio 4*
- Radio 5*
- Red Dragon Radio*

i) From the menu bar of the application ii) From a pop up menu iii) From a toolbar.
Add tooltips to each menu item that indicates some information about the Radio station such as type of music and its broadcast frequency.

Solution :

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

public class MenuApplication
{
    public static void main(String args[])
    {
        JFrame f = new JFrame("Simple Menu Application");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /* Creating Menu Bar */
        JMenuBar bar = new JMenuBar();
        JMenu menu = new JMenu("Options");
        menu.setMnemonic(KeyEvent.VK_O);
        ButtonGroup group = new ButtonGroup();
        JRadioButtonMenuItem menuItem = new JRadioButtonMenuItem("Radio 1");
        menuItem.setToolTipText("Music Masala 94.5KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 2");
        menuItem.setToolTipText("Retro Melody 100.1KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 3");
        menuItem.setToolTipText("Classical Khajana 98.3KHz");
        group.add(menuItem);
        menu.add(menuItem);
    }
}
```

```
menuItem = new JRadioButtonMenuItem("Radio 4");
menuItem.setToolTipText("Music on Demand 91.1KHz");
group.add(menuItem);
menu.add(menuItem);

menuItem = new JRadioButtonMenuItem("Radio 5");
menuItem.setToolTipText("Disco Station 95.3KHz");
group.add(menuItem);
menu.add(menuItem);

menuItem = new JRadioButtonMenuItem("Red Dragon Radio");
group.add(menuItem);
menu.add(menuItem);

bar.add(menu);
f.setJMenuBar(bar);

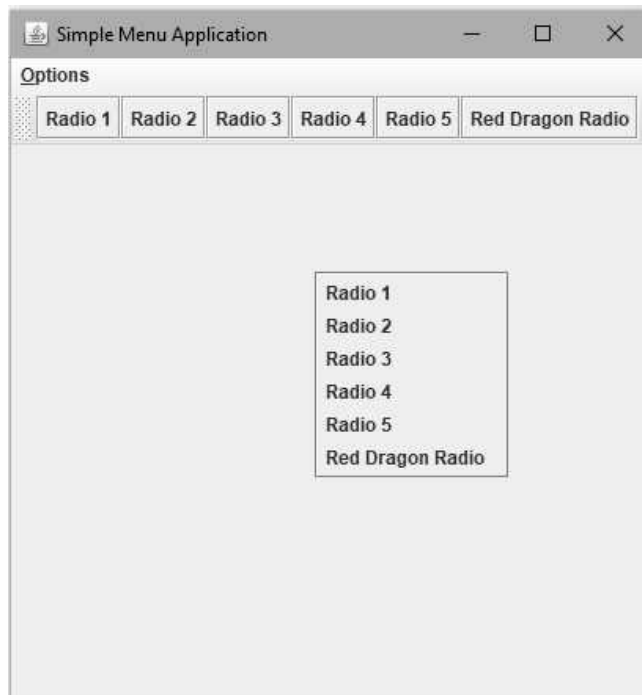
/* Creating ToolBar */
JToolBar toolbar = new JToolBar();
toolbar.setRollover(true);
JButton button = new JButton("Radio 1");
toolbar.add(button);
button = new JButton("Radio 2");
toolbar.add(button);
button = new JButton("Radio 3");
toolbar.add(button);
button = new JButton("Radio 4");
toolbar.add(button);
button = new JButton("Radio 5");
toolbar.add(button);
button = new JButton("Red Dragon Radio");
toolbar.add(button);
Container contentPane = f.getContentPane();
contentPane.add(toolbar, BorderLayout.NORTH);

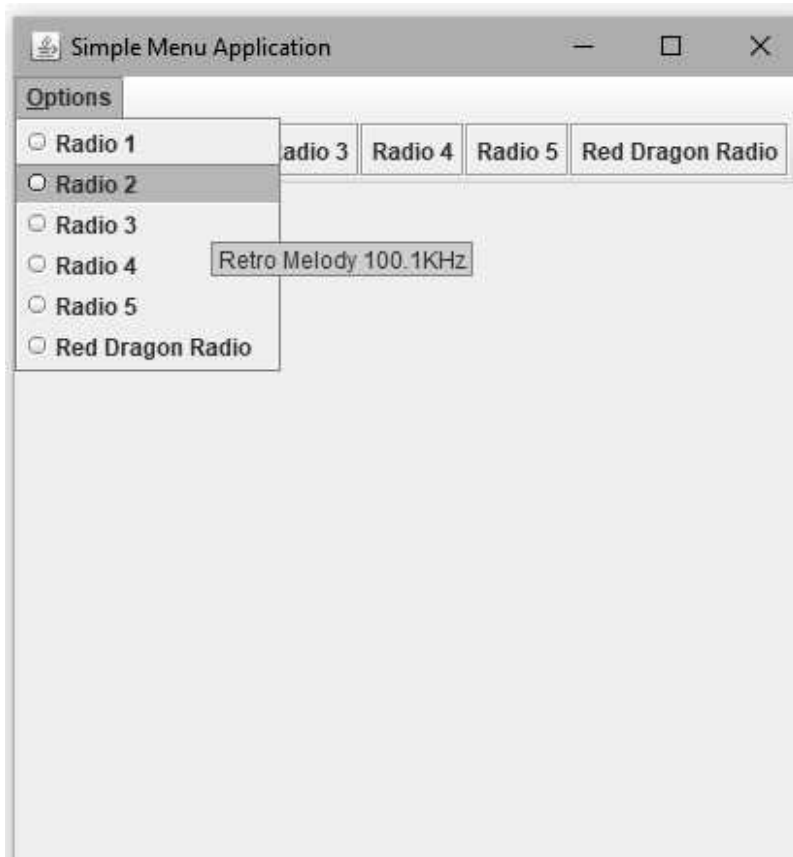
/* Creating Popup menu*/
JPopupMenu popup = new JPopupMenu();
JMenuItem mItem=new JMenuItem("Radio 1");
popup.add(mItem);
mItem=new JMenuItem("Radio 2");
popup.add(mItem);
mItem=new JMenuItem("Radio 3");
popup.add(mItem);
mItem=new JMenuItem("Radio 4");
popup.add(mItem);
mItem=new JMenuItem("Radio 5");
popup.add(mItem);
mItem=new JMenuItem("Red Dragon Radio");
```

```
popup.add(mItem);

f.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        showPopup(e);
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        showPopup(e);
    }
    private void showPopup(MouseEvent e) {
        if (e.isPopupTrigger()) {
            popup.show(e.getComponent(),
                e.getX(), e.getY());
        }
    }
});
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setSize(500, 500);
f.setVisible(true);
}
```

Output





3.4.14 Dialog Boxes

- The dialog box is useful to display some useful messages to the user. Using Swing we can create three types of dialog boxes -
 1. Simple message dialog box
 2. Confirm message dialog box
 3. Input dialog box
- These dialog boxes are created using **JOptionPane** class. This class is present in the **javax.swing.*** package. Hence we must import this file at the beginning.

Let us understand these dialog boxes with the help of programming examples -

Simple message dialog box

- This type of message box simply displays the informative message to the user.
- It has only OK button.

- It is expected that after reading out the message the user must click the OK button to return.
- The **JOptionPane** class provides the method **showMessageDialog()** in order to display the simple message box.
- Following is the simple Java program which shows how to create simple message dialog box -

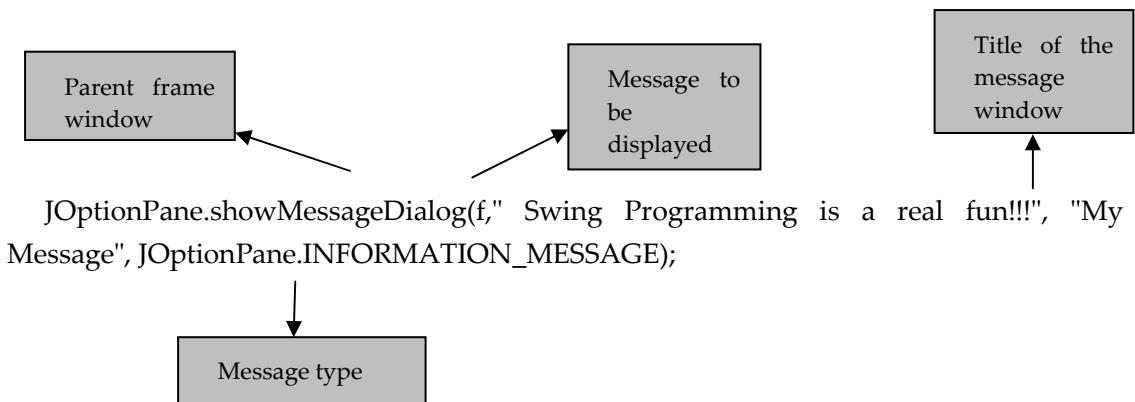
Java Program[DialogBox1Prog.java]

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class DialogBox1Prog
implements ActionListener
{
    JFrame f;
    public static void main(String[] args)
    {
        new DialogBox1Prog();
    }
    public DialogBox1Prog()
    {
        f=new JFrame("Dialog Box Demo");
        JButton B=new JButton("Click Me!!");
        Container container=f.getContentPane();
        container.setLayout(new FlowLayout());
        container.add(B);
        B.addActionListener(this);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        JOptionPane.showMessageDialog(f,"Swing Programming is a real fun!!!","My
        Message",JOptionPane.INFORMATION_MESSAGE);
    }
}
```


Output**Program explanation**

In above Java program, we have created one frame window, on which one push button is placed. When we click the button, then the simple message box will be displayed. This message box has only one OK button.

To display the dialog box we have

**Confirm message dialogbox**

- This type of dialog box will display some message to the user and will get the confirmation from him.
- User can give his confirmation either by clicking OK, CANCEL, YES or NO button.

- The confirm message dialog box will display the message either along with the OK and CANCEL button or with the YES, NO or CANCEL button.
- In the following Java program we have used three types of message boxes - simple message box, confirm message box with OK and CANCEL and the confirm message box with YES, NO and CANCEL.

Java Program[DialogBox2Prog.java]

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class DialogBox2Prog
implements ActionListener
{
    JFrame f;
    public static void main(String[] args)
    {
        new DialogBox2Prog();
    }
    public DialogBox2Prog()
    {
        f=new JFrame("Dialog Box Demo");
        Container container=f.getContentPane();
        container.setLayout(new FlowLayout());
        JButton B=new JButton("Simple Message DialogBox");
        container.add(B);
        B.addActionListener(this);

        B=new JButton("OK and Cancel DialogBox");
        container.add(B);
        B.addActionListener(this);

        B=new JButton("Yes/No/Cancel DialogBox");
        container.add(B);
        B.addActionListener(this);

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) //event handler function
```

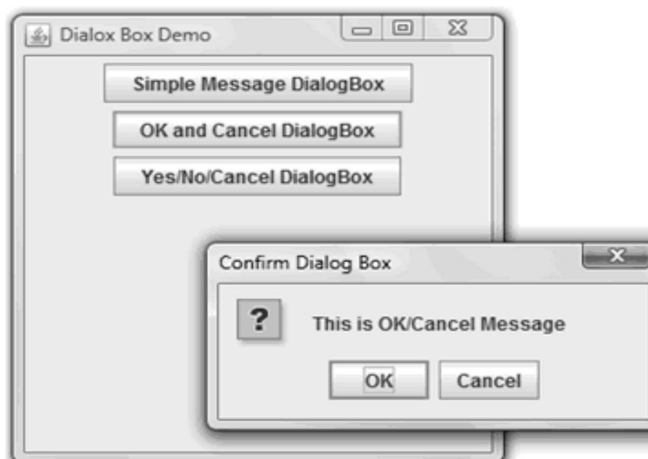
On this button click simple message box will be displayed

On this button click OK/Cancel dialog box will be displayed

On this button click Yes/No/Cancel dialog box will be displayed

```
{
String s=e.getActionCommand(); //getting the string present on the button
if(s.equals("Simple Message DialogBox"))
    JOptionPane.showMessageDialog(f,"Simple Message");
else if(s.equals("OK and Cancel DialogBox"))
{
if(JOptionPane.showConfirmDialog(f,"This is OK/Cancel Message","Confirm Dialog
Box",JOptionPane.OK_CANCEL_OPTION)==0)
    JOptionPane.showMessageDialog(f,"You have clicked OK button");
else
    JOptionPane.showMessageDialog(f,"You have clicked Cancel button");
}
else if(s.equals("Yes/No/Cancel DialogBox"))
{
if(JOptionPane.showConfirmDialog(f,"This is Yes/No/Cancel Message","Confirm Dialog
Box",JOptionPane.YES_NO_CANCEL_OPTION)==0)
{
    JOptionPane.showMessageDialog(f,"You have clicked Yes button");
}
else
{
    JOptionPane.showMessageDialog(f,"You have clicked OK or Cancel button");
}
}
}
}
}
```

Output



Program explanation

For the confirm dialog box, the option type could be `OK_CANCEL`, `YES_NO_OPTION`, `OK_OPTION`, `NO_OPTION` which are associated with some constant values.

In above program, when the user clicks the button for OK and Cancel message then appropriate message will be displayed. If user clicks the OK button then the message "You have clicked OK button will be displayed" otherwise the message "You have clicked Cancel button" will be displayed.

3.4.15 Tabbed Pane

Tabbed pane is a type of component in which group of folders can be represented together and particular folder can be selected from the tab.

Each folder has a title and when the user selects the particular folder on the tab then only it will be displayed. One folder can be displayed at a time.

For displaying the tabbed pane there exists a **JTabbedPane** class which extends the **JComponent** class.

Using the **addTab** method the folders can be added to the tabbed pane. The syntax for this method is

```
void addTab(Strin str,Component comp)
```

The *str* represents the string which will be displayed as a title to the tab.

The *comp* is a reference to the component which should be added to the tabbed pane.

Following program illustrates the use of tabbed pane.

Step 1 : Create an applet.

Step 2 : Create the tabbed pane.

Step 3 : Create a contentpane.

Step 4 : Place the tabbed pane on the content pane.

Step 5 : Define the components of tabbed pane.

Java program

```
import javax.swing.*;
/*
<applet code="TabbedPaneDemo" width=500 height=400>
</applet>
*/

public class TabbedPaneDemo extends JApplet
```

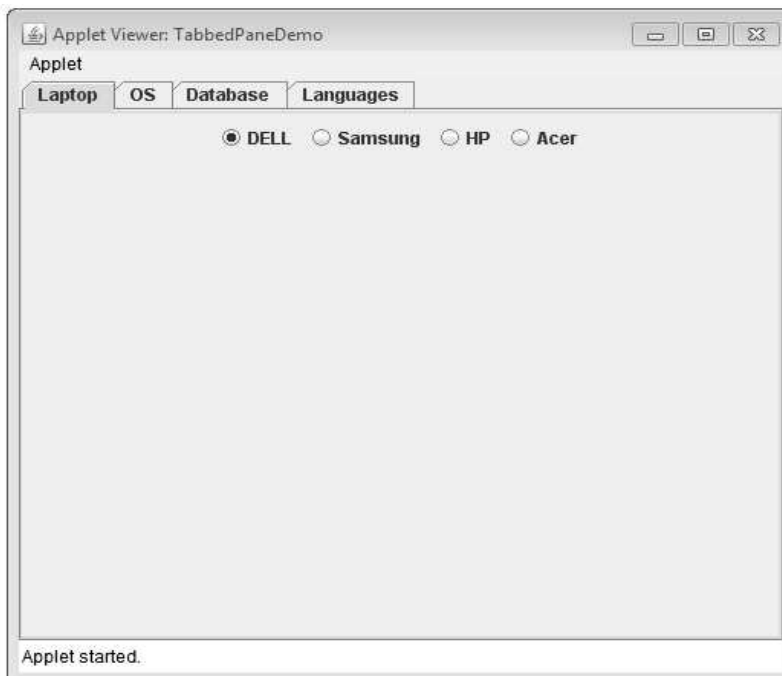
```
{
public void init()
{
    JTabbedPane mytpane = new JTabbedPane();
    mytpane.addTab("Laptop", new LaptopPanel());
    mytpane.addTab("OS", new OSPanel());
    mytpane.addTab("Database", new DatabasePanel());
    mytpane.addTab("Languages", new LangPanel());
    getContentPane().add(mytpane);
}
}
class LaptopPanel extends JPanel
{
    public LaptopPanel()
    {
        JRadioButton jrb1 = new JRadioButton("DELL");
        add(jrb1);
        JRadioButton jrb2 = new JRadioButton("Samsung");
        add(jrb2);
        JRadioButton jrb3 = new JRadioButton("HP");
        add(jrb3);
        JRadioButton jrb4 = new JRadioButton("Acer");
        add(jrb4);
        ButtonGroup bg=new ButtonGroup();
        bg.add(jrb1);
        bg.add(jrb2);
        bg.add(jrb3);
        bg.add(jrb4);
    }
}
class OSPanel extends JPanel
{
    public OSPanel()
    {
        JComboBox jcb = new JComboBox();
        jcb.addItem("Windows XP");
        jcb.addItem("Windows Vista");
        jcb.addItem("Windows 7");
        jcb.addItem("Ubuntu");
        add(jcb);
    }
}
class DatabasePanel extends JPanel
```

The diagram shows a Java code snippet with several callout boxes. A large right-facing curly bracket groups the first five lines of the `init()` method, with a callout box labeled "Adding the folders on the tabbed pane". A callout box labeled "Languages Component defined" points to the `LangPanel` class definition. A callout box labeled "OS Component defined" points to the `OSPanel` class definition. A callout box labeled "Database Component defined" points to the `DatabasePanel` class definition.

```
{
public DatabasePanel()
{
    JCheckBox cb1 = new JCheckBox("Oracle");
    add(cb1);
    JCheckBox cb2 = new JCheckBox("MySQL");
    add(cb2);
    JCheckBox cb3 = new JCheckBox("Microsoft Access");
    add(cb3);
}
}
class LangPanel extends JPanel
{
public LangPanel()
{
    JButton b1 = new JButton("JSP");
    add(b1);
    JButton b2 = new JButton("ASP");
    add(b2);
    JButton b3 = new JButton("PHP");
    add(b3);
}
}
```

Languages Component defined

Output



3.4.16 JTree

Tree is a type of component that gives the hierarchical view of data. User can expand or shrink the nodes of the tree. In swing the trees are implemented using the **JTree** class. This class extends the **JComponent**.

The most commonly used constructor for this class is -

```
JTree(TreeNode root)
```

The *root* represents the root node of the tree.

Using the **DefaultMutableTreeNode**, it creates a tree node with no root node, the child of root node, specified by user object and it allows only children that have to be specified. It takes boolean types values either 'true' or 'false'. If you will take 'true' that means children node are allowed.

Step 1 : Make use of applet for implementation of the tree program.

Step 2 : Create a tree with root, child and grand child nodes.

Step 3 : Create a content pane object.

Step 4 : Add the JTree component on the content pane.

Java Program

```
import javax.swing.*;
import java.awt.*;
import javax.swing.tree.*;
/*
<applet code="TreeDemo" width=300 height=200>
</applet>
*/

public class TreeDemo extends JApplet
{
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new BorderLayout());

        DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root", true);

        DefaultMutableTreeNode c1 = new DefaultMutableTreeNode("Child 1");
        root.add(c1);

        DefaultMutableTreeNode c2 = new DefaultMutableTreeNode("Child 2");
        root.add(c2);
    }
}
```

```
DefaultMutableTreeNode gc1 = new DefaultMutableTreeNode("GrandChild 1");
DefaultMutableTreeNode gc2 = new DefaultMutableTreeNode("GrandChild 2");
DefaultMutableTreeNode gc3 = new DefaultMutableTreeNode("GrandChild 3");
c2.add(gc1);
c2.add(gc2);
c2.add(gc3);

DefaultMutableTreeNode c3 = new DefaultMutableTreeNode("Child 3");
root.add(c3 );

DefaultMutableTreeNode c4 = new DefaultMutableTreeNode("Child 4");
root.add(c4);

DefaultMutableTreeNode c5 = new DefaultMutableTreeNode("Child 5");
root.add(c5);

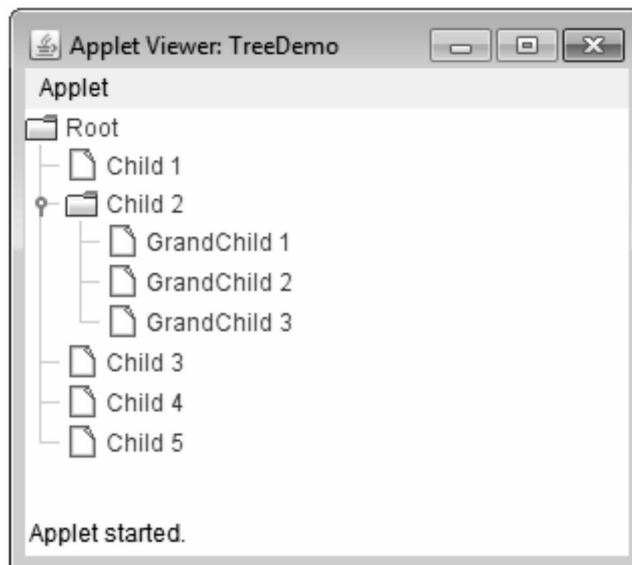
JTree tree = new JTree(root);
contentPane.add(tree);
}
}
```

Following commands can be used to execute the above code.

```
D:\JavaPrograms>javac TreeDemo.java
```

```
D:\JavaPrograms>AppletViewer TreeDemo.java
```

Output



3.4.17 JTable

Table is a component that arranges the data in rows and columns. The **JTable** class extends the **JComponent** class. The constructor used for table component is -

```
JTable( object[][] tablevalues object [] columnheader)
```

The *tablevalues* indicate the data that can be arranged in tabular fashion.

The *columnheader* denotes the header for each column.

Following is a simple program that shows the use of JTable component.

Java Program

```
import javax.swing.*;
import java.awt.*;
import javax.swing.tree.*;
/*
<applet code="TableDemo" width=300 height=100>
</applet>
*/

public class TableDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();

        contentPane.setLayout(new BorderLayout());

        final String[] th = { "Name", "City", "Salary","Designation" };
        final Object[][] mytable = {
            { "Arun", "Pune", "5000","Accountant"},
            { "Archana", "Mumbai", "7000","Executive"},
            { "Shivani", "Banglore", "10000","Manager"},
            { "Priyanka", "Chennai", "8000","Programmer"},
            { "Monika", "Hyderabad", "10000","Designer"},
            { "Shilpa", "Hyderabad", "12000","Director"},
            { "Anuja", "Delhi", "17000","Director"},
            { "Kumar", "Pune", "10000","Manager"}
        };

        JTable table = new JTable(mytable,th);

        int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
```

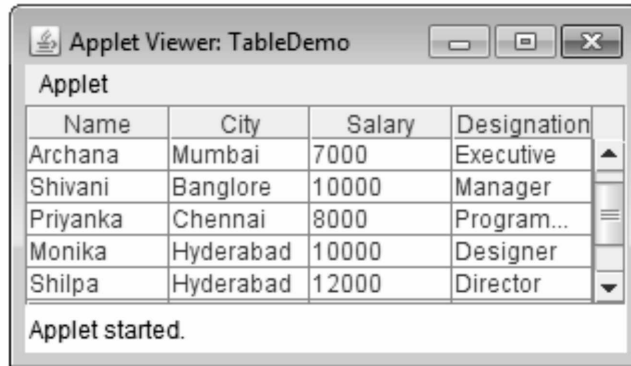
```
JScrollPane mypane = new JScrollPane(table, vscrollbar, hscrollbar);
contentPane.add(mypane, BorderLayout.CENTER);
}
}
```

Following commands are used to display the output.

```
D:\JavaPrograms>javac TableDemo.java
```

```
D:\JavaPrograms>AppletViewer TableDemo.java
```

Output



Example 3.4.8 Write a Java program to display the 3×3 magic square using `JTable`.

Solution :

```
import javax.swing.*.*;
import java.awt.*.*;
import javax.swing.tree.*.*;
/*
<applet code="TableDemo" width=100 height=100>
</applet>
*/

public class TableDemo extends JApplet
{
public void init()
{
Container contentPane = getContentPane();

contentPane.setLayout(new BorderLayout());
final String[] th = { "", "", "" };
final Object[][] mytable = {
{ "8", "1", "6" },
{ "3", "5", "7" },
{ "4", "9", "2" }
}
```

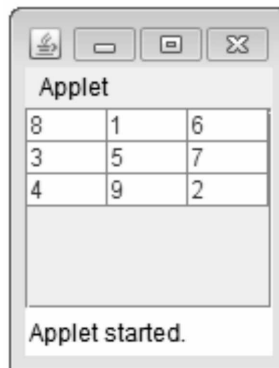
```
};

JTable table = new JTable(mytable,th);

int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

JScrollPane mypane = new JScrollPane(table, vscrollbar, hscrollbar);
contentPane.add(mypane, BorderLayout.CENTER);
}
}
```

Output



3.5 Java Utilities (java.util Package)

Package **java.util** contains the collections framework, legacy collection classes, event model, date and time facilities, internationalisation and miscellaneous utility classes.

Importing java.util package

The **import** is a java keyword which is used for importing a Java class or entire Java package. For example `import java.util.Calendar;` means you are importing a single calendar class. If you want to import all the classes from any Java package, your import statement must be like this **`import java.util.*`**. That means you are importing the entire java.util package.

3.6 The Collection Framework

- The standard data structures can be implemented in Java using some library classes and methods.
- These classes are present in the **java.util** package.
- The collection framework is comprised of **collection classes** and **collection interfaces**.

- Basically collection is a group of objects which are designed to perform certain task. These tasks are associated with alteration of data structures.
- The collection classes are the group of classes used to implement the collection interfaces.
- Various collection classes are

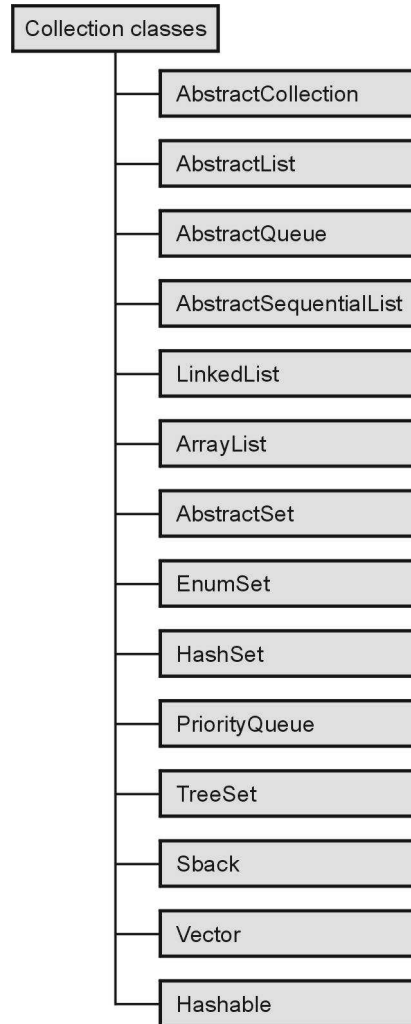


Fig. 3.6.1 Collection class hierarchy

Review Question

1. Write short note on - collection framework.

3.7 Collections of Objects and Types

Collection classes provide the implementation of different collection interfaces.

Fig. 3.7.1 shows different collection interfaces.

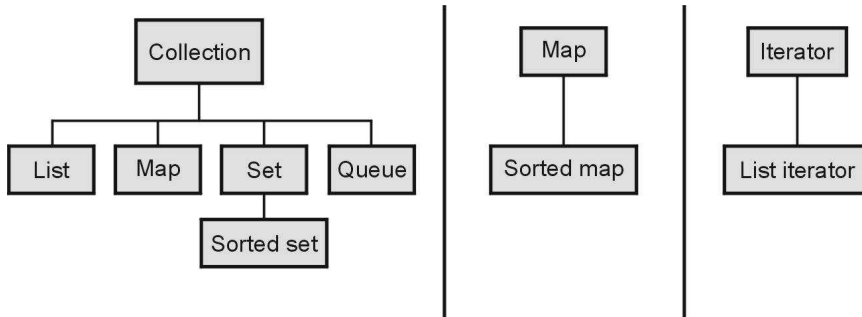


Fig. 3.7.1 Interfaces in collection framework

The collection interfaces contain several useful methods by which we can modify the collections. Various methods that are supported by collection are as given below -

Method	Description
boolean add(Object obj)	Objects are added using this method. This method takes arguments of type object.
boolean addAll(Collection collection)	Entire content of one collection can be added to another using this method.
void clear()	In order to clear the collection, this method is used.
boolean contains(Object obj)	For checking whether the collection contains specific object or not this method is used.
boolean containsAll(Collection collection)	If all the elements of the collection are present in the collection then this method returns true.
boolean isEmpty()	It's a Boolean method which determines whether collection is empty or not. If a collection is empty then it return true otherwise false.
Iterator iterator()	Returns iterator to the collection.
boolean remove(Object obj)	An object can be removed by this method.
boolean removeAll(Collection collection)	It helps in removing a group of objects.
int size()	It returns the total number of elements in the collection.

Object[] toArray()	Returns an array of elements to the invoking collection. Basically these array elements are the copies of the collection that calls the toArray method.
boolean equals(Object obj)	For comparing two collections this method is used.

The methods in collection interface throws has two common exceptions -

Unsupported Operation Exception for illegal supporting operation and **ClassCastException** when one object is incompatible with other object.

List Interface

The List interface extends the Collection interface. It is basically the sequence of elements in which the elements can be inserted and accessed by their position in the list. The duplicate elements are allowed in this interface. Various methods used in the **List** interface are enlisted in following table -

Method	Description
void add(int i, Object obj)	This method inserts the object obj at specified location in the list. This location is denoted by index i.
void add(int i, Collection collection)	This method inserts elements of collection collection at specified location in the list. This location is denoted by index i.
Object get(int i)	To get the object stored at specified location in the list, this method is used. This location can be obtained with the help of index i.
Object set(int i, Object obj)	The value of the element can be set using this method.
int indexOf(Object obj)	If object is an element of the list then its index will be returned by this method.
int lastIndexOf(Object obj)	This method returns the index of the last object present in the list.
ListIterator listIterator()	It returns an iterator of the element to the start of the list.
ListIterator listIterator(int i)	It returns an iterator to the list which begins at specified index.

Object remove(int i)	This method is for removing the element present at the specific position.
List subList(int starting,int end)	It returns the sublist specified within the starting index and ending index.

Set Interface

The set interface is used to define the set of elements. It extends the collection interface. This interface defined unique elements. Hence if any duplicate elements is tried to insert in the set then the add() method returns false.

SortedSet Interface

This interface is inherited from the set interface and allows the elements to be arranged in ascending order. The methods defined in this interface normally throw the exception such as **NoSuchElementException**, **NullPointerException** and **ClassCastException**.

Various methods used by this interface are as given below -

Method	Description
Comparator comparator()	This method returns the comparator object. If the elements in the SortedSet are present in ascending order then this method returns null.
Object first()	It returns the first element present in the SortedSet.
Object last()	It returns the last element present in the SortedSet.
SortedSet headset(Object target)	This method returns all those elements which are less than the target element.
SortedSet subset(Object source,Object target)	This method returns all those elements of SortedSet which are present between source and target.
SortedSet headset(Object target)	This method returns all those elements which are greater than or equal to the target element.

Map Interface

This interface maps a unique key element to the value. Thus map interface represents a key-value pair.

SortedMap Interface

The **SortedMap** is inherited from the Map interface. In this interface the elements are stored in ascending order. This sorted order is based on the key.

Review Question

1. Explain the collection interface with example.

3.8 List Interface

- Lists interface extends the **collection** interface.
- It is basically the sequence of elements in which the elements can be inserted and accessed by their position in the list. The duplicate elements are allowed in this interface.
- There are two concrete classes for defining the **List** interface.

1. ArrayList

2. LinkedList

3.8.1 ArrayList

- The **ArrayList** class implements the List interface. It is used to implement the dynamic array.
- The dynamic array means the size of array can be created as per requirement. Hence **ArrayList** is a variable length array of object references.
- Initially **ArrayList** is created with some initial size and then as per requirement we can extend the size of array. When the objects are removed then the size of array can be reduced.
- The syntax of using **ArrayList** is as given below -

ArrayList() ← Creates an empty list

ArrayList(Collection collection) ← Creates a list in which the collection elements are added

ArrayList(int c) ← Creates a list with specified capacity c, the capacity represents the size of the underlying array

Let us see simple Java program which demonstrates the **ArrayList**

Java Program [ArrayListProg.Java]

```

/*****
Program uses the ArrayList collection class to
implement ArrayList data structure
*****/
import java.util.*;
class ArrayListProg
{
public static void main(String[] args)
{
System.out.println("\n\t\t Program for Implementing Array List");
}
}

```



```
ArrayList obj=new ArrayList();           //creation of ArrayList
System.out.println("\n Inserting some elements in the array");
obj.add(10);
obj.add(20);
obj.add(30);
obj.add(40);
obj.add(50);
System.out.println("The array elements are... "+obj);
System.out.println("\n Inserting some elements in the array in between");
obj.add(4,45);                          //inserting after element 40
System.out.println("The array elements are... "+obj);
System.out.println("\n Removing some elements from the array");
obj.remove(1);                           //array index is passed to remove method
System.out.println("The array elements are... "+obj);
System.out.println("The array size is... "+obj.size());
}
}
```

Output

Program for Implementing Array List

Inserting some elements in the array

The array elements are... [10, 20, 30, 40, 50]

Inserting some elements in the array in between

The array elements are... [10, 20, 30, 40, 45, 50]

Removing some elements from the array

The array elements are... [10, 30, 40, 45, 50]

The array size is... 5

Program explanation

In above program we have created an array list of integer numbers we can also pass some character elements to the array list. While removing the element from the list we should pass the index position of the element. Note that the elements are placed from 0th index in the ArrayList. The above array is a dynamic array. As we insert the elements it grows and as we remove the elements it gets shrunk.

Example 3.8.1 Write a program which stores the list of strings in an ArrayList and then displays the contents of the list.

Solution :

```
import java.util.*;
class ArrayListProg
{
```

```

public static void main(String[] args)
{
    System.out.println("\n\t\t Program for Implementing Array List for List of Strings");
    ArrayList obj=new ArrayList();           //creation of ArrayList
    System.out.println("\n Inserting some elements in the array");
    obj.add("AAA");
    obj.add("BBB");
    obj.add("CCC");
    obj.add("DDD");
    obj.add("EEE");
    System.out.println("The array elements are... "+obj);
    System.out.println("The array size is... "+obj.size());
}
}

```

Output

```

Program for Implementing Array List for List of Strings
Inserting some elements in the array
The array elements are... [AAA, BBB, CCC, DDD, EEE]
The array size is... 5

```

3.8.2 LinkedList

- Linked List is a collection of nodes in which every node contains two fields Data and Next pointer fields.
- The link list can be graphically represented as follows -

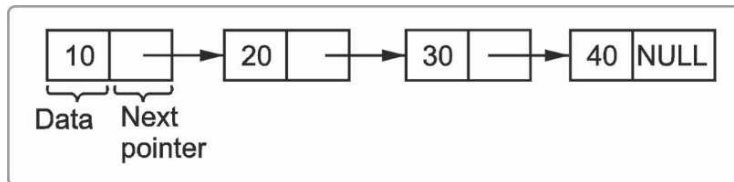


Fig. 3.8.1 Linked list

- The java.util package provides the collection class LinkedList in order to implement the List interface.
- The syntax of using this class is

LinkedList() ← creates an empty linked list

LinkedList(Collection collection) ← creates a linked list having the elements of *collection*

Programming example :

Following program makes use of various methods such as **add()**, **remove()**, **addFirst()**, **addLast()**, **removeFirst()**, **removeLast()** for manipulating the contents of linked list.

```
import java.io.*;
import java.util.*;
class LinkedListProg
{
    public static void main(String[] args) throws IOException
    {
        char ans='y',ch='y';
        int choice,val,position;
        String str;
        LinkedList obj=new LinkedList();
        Scanner s=new Scanner(System.in);
        do
        {
            System.out.println("\n\t\t Program for Implementing Linked List");
            System.out.print("\n1.Create\n2.Display \n3.Insert First\n4. Insert Last");
            System.out.print("\n5.Delete First\n6.Delete Last");
            System.out.print("\n7.Insert At any Position");
            System.out.println("\n8.Delete From any Position");
            System.out.println("Enter Your choice");
            choice=s.nextInt();
            switch(choice)
            {
                case 1:
                    do
                    {
                        System.out.println("\n Enter the element to be inserted in the list");
                        val=s.nextInt();
                        obj.add(val);
                        System.out.println(" Do u want to insert more elements?");
                    } while(ans == 'y');
                    str =s.next();
                    ans =str.charAt(0);

                    break;
                case 2:
                    System.out.println("\t The List elements are... "+obj);
                    System.out.println("\t The size of linked list is... "+obj.size());
                    break;
                case 3:
                    System.out.println("\n Enter the element to be inserted in the list");
                    val=s.nextInt();
                    obj.addFirst(val);
                    System.out.println("\n The element inserted!!!");
                    break;
                case 4:
                    System.out.println("\n Enter the element to be inserted in the list");
```

```
        val=s.nextInt();
        obj.addLast(val);
        System.out.println("\n The element inserted!!!");
        break;
    case 5: obj.removeFirst();
        System.out.println("\n The element deleted!!!");
        break;
    case 6: obj.removeLast();
        System.out.println("\n The element deleted!!!");
        break;
    case 7: System.out.println("\n Enter the element to be inserted in the list");
        val=s.nextInt();
        System.out.println("\n Enter the position at which the element is to be inserted");
        position=s.nextInt();
        obj.add(position, val);
        System.out.println("\n The element inserted!!!");
        break;
    case 8: System.out.println("\n Enter the position of element to be deleted");
        position=s.nextInt();
        obj.remove(position);
        System.out.println("\n The element deleted!!!");
        break;
    }
    System.out.println("\n Do u want to go to main menu?");
    str=s.next();
    ch=str.charAt(0);
    }while(ch != 'y');
}
}
```

Output

Program for Implementing Linked List

```
1. Create
2. Display
3. Insert First
4. Insert Last
5. Delete First
6. Delete Last
7. Insert At any Position
8. Delete From any Position
Enter your choice
1
Enter the element to be inserted in the list
10
Do u want to insert more elements?
```

```
y
Enter the element to be inserted in the list
20
Do u want to insert more elements?
y
Enter the element to be inserted in the list
30
Do u want to insert more elements?
y
Enter the element to be inserted in the list
40
Do u want to insert more elements?
n
Do u want to go to main menu?
y
        Program for Implementing Linked List
1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
2
        The List elements are... [10, 20, 30, 40]
        The size of linked list is... 4
Do u want to go to main menu?
y
        Program for Implementing Linked List
1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
3
Enter the element to be inserted in the list
9
The element inserted!!!
```

```

Do u want to go to main menu?
y
    Program for Implementing Linked List
1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
2
    The List elements are... [9, 10, 20, 30, 40]
    The size of linked list is... 5
Do u want to go to main menu ?

```

Difference between ArrayList and LinkedList

Sr. No.	ArrayList	LinkedList
1.	ArrayList internally uses dynamic array to store the elements.	LinkedList internally uses doubly linked list to store the elements.
2.	It internally uses array. If any element is removed from the array, all the bits are shifted in memory.	It uses doubly linked list so no bit shifting is required in memory.
3.	Manipulation with ArrayList is slow.	Manipulation with LinkedList is faster than ArrayList.
4.	ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
5.	ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

Review Question

1. What is the difference between ArrayList and LinkedList ?

3.9 Vector

Vector is similar to the **Array List** class which implements the **dynamic array**. It implements the list interface. This class contains various additional methods which are enlisted below -

Method	Description
void addElement(object)	For adding some element in the vector this method is used.
void insertElementAt(object obj,int pos)	For inserting the element in the vector specified by its position.
boolean removeElement(object ele)	This method removes the specified element.
void removeAllElements()	This method is for removing all the elements from the vector.
void removeElementAt(int pos)	The element specified by its position gets deleted from the vector.
int capacity()	It returns the capacity of the vector.
int size()	It returns the total number of elements present in the vector.
boolean isEmpty()	It returns true if the vector is empty.
object firstElement()	Returns the first element of the vector.
int indexOf(object ele)	It returns the index of the corresponding element in the vector.
void setSize(int size)	This method is for setting the size of the vector.

Following program illustrates the implementation of vector class.

Java Program [VectorProg.Java]

```

/*****
Program for implementing vector class
*****/
import java.io.*;
import java.util.*;
class VectorProg
{
public static void main(String[] args)throws IOException
{

```

```

Vector obj=new Vector(3);
int ival;
double dval;
char ans='y';
System.out.println("The capacity of vector is "+obj.capacity());
System.out.println("The total number of elements are "+obj.size());
System.out.println("\t Inserting the integers");
do
{
System.out.println("\n Enter some integer value");
ival=getInt();
obj.addElement(ival);
System.out.println("\n Do u want to enter more?");
ans=getChar();
}while(ans=='y');
System.out.println("The elements in the vector are "+obj);
System.out.println("\t Inserting the double values");
do
{
System.out.println("\n Enter some double value");
dval=getDouble();
obj.addElement(dval);
System.out.println("\n Do u want to enter more?");
ans=getChar();
}while(ans=='y');
System.out.println("The elements in the vector are "+obj);
System.out.println("\t The size of vector is "+obj.size());
System.out.println("\t The first element of the vector is "+(Integer)obj.firstElement());
System.out.println("\t The last element of the vector is "+(Double)obj.lastElement());
System.out.println("Removing 2 the elements");
obj.remove(0);
obj.remove(2);
System.out.println("Now The elements in the vector are "+obj);

}
////////////////////////////////////
//Following functions are used to handle the inputs entered
//by the user using keyboard
////////////////////////////////////
public static String getString() throws IOException
{
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader b = new BufferedReader(input);
String str = b.readLine();//reading the string from console

```



```
    return str;
}

public static char getChar() throws IOException
{
    String str = getString();
    return str.charAt(0);//reading first char of console string
}

public static int getInt() throws IOException
{
    String str = getString();
    return Integer.parseInt(str);//converting console string to numeric value
}

public static double getDouble() throws IOException
{
    String str = getString();
    return Double.parseDouble(str);//converting console string to numeric value
}
}
```

Output

```
The capacity of vector is 3
The total number of elements are 0
    Inserting the integers

Enter some integer value
11

Do u want to enter more?
y

Enter some integer value
22

Do u want to enter more?
y

Enter some integer value
33

Do u want to enter more?
y

Enter some integer value
44
```

```

Do u want to enter more?
n
The elements in the vector are [11, 22, 33, 44]
    Inserting the double values

Enter some double value
1.1

Do u want to enter more?
y

Enter some double value
2.22

Do u want to enter more?
y

Enter some double value
3.333

Do u want to enter more?
n
The elements in the vector are [11, 22, 33, 44, 1.1, 2.22, 3.333]
    The size of vector is 7
    The first element of the vector is 11
    The last element of the vector is 3.333
Removing 2 the elements
Now The elements in the vector are [22, 33, 1.1, 2.22, 3.333]

```

3.10 Set Interface

- The set interface is used to define the set of elements.
- It extends the collection interface.
- This interface defined unique elements. Hence if any duplicate elements is tried to insert in the set then the add() method returns false.
- Some commonly used functionalities in Set interface are -

Method	Description
add()	Adds an object to the collection.
clear()	Removes all objects from the collection.
contains()	Returns true if a specified object is an element within the collection.
isEmpty()	Returns true if the collection has no elements.

iterator()	Returns an Iterator object for the collection, which may be used to retrieve an object.
remove()	Removes a specified object from the collection.
size()	Returns the number of elements in the collection.

There are two commonly used ways of implementing set interface.

3.10.1 HashSet

- HashSet is a collection class that implements the set interface.
- It creates the hash table.
- The hash table is a data structure in which the data is stored using hashing function. Hence the elements get stored in the hash table based on the hash key returned by hash function.
- The syntax of using HashSet class is -

```
HashSet()
HashSet(Collection collection)
HashSet(int c)
HashSet(int c,float fillRatio)
```

- The fillRatio ranges from 0.0 to 1.0. This parameter determines how full the Hash set can be before getting resized.
- Various methods supported by the Set class are supported by the HashSet class. Following program implements the hash table -

Java Program[HashSetProg.java]

```
import java.io.*;
import java.util.*;
class HashSetProg
{
public static void main(String[] args)throws IOException
{
char ans='y';
int val;
System.out.println("\n\t\t Program to create Hash Set");
HashSet obj=new HashSet();
Scanner s=new Scanner(System.in);
System.out.println("\n Creation of hash set");
do
{
System.out.println(" Enter the element ");
```

```
    val=s.nextInt();
    obj.add(val);
    System.out.println("\n Do u want to enter more elements?");
    String str=s.next();
    ans=str.charAt(0);
}while(ans=='y');
    System.out.println("The elements are ..." +obj);
}
}
```

Output

```
        Program to create Hash Set
Creation of hash set
Enter the element
10
Do u want to enter more elements?
y
Enter the element
20
Do u want to enter more elements?
y
Enter the element
30
Do u want to enter more elements?
y
Enter the element
40
Do u want to enter more elements?
y
Enter the element
50
Do u want to enter more elements?
n
The elements are ...[50, 20, 40, 10, 30]
```

3.10.2 TreeSet

- The TreeSet class implements the **Set** collection class.
- This class is basically for creating the Tree data structure.
- The binary tree gets created and when we try to display the tree then we get the nodes arranged in sorted manner.
- The following program makes use of TreeSet collection class.

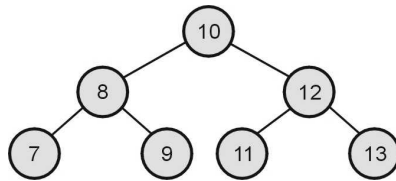


Fig. 3.10.1 Binary tree

- We will create following tree in the program -

Java Program[TreeSetProg.java]

```

import java.io.*;
import java.util.*;
class TreeSetProg
{
    public static void main(String[] args)throws IOException
    {
        char ans='y';
        int val;
        System.out.println("\n\t Program to create Tree data structure");
        TreeSet obj=new TreeSet();
        Scanner s=new Scanner(System.in);
        System.out.println("\n Creation of Tree");
        do
        {
            System.out.println(" Enter the element ");
            val=s.nextInt();
            obj.add(val);
            System.out.println("\n Do u want to enter more elements?");
            String str=s.next();
            ans=str.charAt(0);
        }while(ans == 'y');
        System.out.println("The tree is... "+obj);
    }
}
  
```

Output

```

    Program to create Tree data structure
    Creation of Tree
    Enter the element
    10
    Do u want to enter more elements?
    y
    Enter the element
    8
    Do u want to enter more elements?
    y
  
```

```

Enter the element
9
Do u want to enter more elements?
y
Enter the element
12
Do u want to enter more elements?
y
Enter the element
11
Do u want to enter more elements?
y
Enter the element
13
Do u want to enter more elements?
n
The tree is... [8, 9, 10, 11, 12, 13]

```

3.11 Map Interface

Map is a kind of data structure which associates the keys to the values. If there is a unique value present for each unique key then the mapping is called **one to one mapping**.

If for a unique key there are multiple values associated with it then it is called **many to one mapping**.

There is a difference between **Maps** and the **Sets**. The maps contain keys and values whereas the sets contain only values.

There are three classes which implement maps - **Hashtable**, **HashMap** and **TreeMap**.

3.11.1 Hashtable

The instance of **Hashtable** is created using which the methods **keys()** and **elements()** are invoked in order to get the keys and corresponding values. Then using **Enumerator** we can access every key and corresponding value of the **Map**.

Java Program[HashtableProg.java]

```

/*****
Program to display all the keys and corresponding values of the map
using Hashtable
*****/
import java.util.*;
public class HashtableProg
{

```

```

public static void main(String[] args)
{
    Hashtable h=new Hashtable();
    h.put("Accounts","Priyanka");
    h.put("Proof","Lekhana");
    h.put("Graphics","Nilesh");
    h.put("DTP","Archana");
    System.out.println("\n Displaying the Keys...");
    Enumeration E=h.keys();
    while(E.hasMoreElements())
        System.out.println(" "+E.nextElement());

    System.out.println("\n Displaying the values...");
    E=h.elements();
    while(E.hasMoreElements())
        System.out.println(" "+E.nextElement());
}
}

```

Output

Displaying the Keys...

Proof

DTP

Graphics

Accounts

Displaying the values...

Lekhana

Archana

Nilesh

Priyanka

3.11.2 HashMap

The instance of **HashMap** is created using which the methods **keySet()** and **values()** are invoked in order to get the keys and corresponding values. Then using iterator we can access every key and corresponding value of the **Map**.

Java Program[HashMapProg.java]

```

/*****
Program to display all the keys and corresponding values of the map
using HashMap
*****/
import java.util.*;

```

```
public class HashMapProg
{
    public static void main(String[] args)
    {
        HashMap h=new HashMap();
        h.put("Accounts","Priyanka");
        h.put("Proof","Lekhana");
        h.put("Graphics","Nilesh");
        h.put("DTP","Archana");
        System.out.println("\n Displaying the Keys...");
        Collection c=h.keySet();
        Iterator i=c.iterator();
        while(i.hasNext())
            System.out.println(" "+i.next());
        System.out.println("\n Displaying the values...");
        c=h.values();
        i=c.iterator();
        while(i.hasNext())
            System.out.println(" "+i.next());
    }
}
```

Output

Displaying the Keys...

Accounts
Graphics
Proof
DTP

Displaying the values...

Priyanka
Nilesh
Lekhana
Archana

3.11.3 TreeMap

Displaying the map using TreeMap is similar to the HashMap. We use the **TreeMap** class instead of the **HashMap** class.

Java Program[TreeMap.java]

```
/******  
Program to display all the keys and corresponding values of the map  
using TreeMap  
*****/
```



```
import java.util.*;
public class TreeMapProg
{
    public static void main(String[] args)
    {
        TreeMap t=new TreeMap();
        t.put("Accounts","Priyanka");
        t.put("Proof","Lekhana");
        t.put("Graphics","Nilesh");
        t.put("DTP","Archana");
        System.out.println("\n Displaying the Keys...");
        Collection c=t.keySet();
        Iterator i=c.iterator();
        while(i.hasNext())
            System.out.println(" "+i.next());
        System.out.println("\n Displaying the values...");
        c=t.values();
        i=c.iterator();
        while(i.hasNext())
            System.out.println(" "+i.next());
    }
}
```

Output

Displaying the Keys...

Accounts

Graphics

Proof

DTP

Displaying the values...

Priyanka

Nilesh

Lekhana

Archana

3.12 Multiple Choice Questions

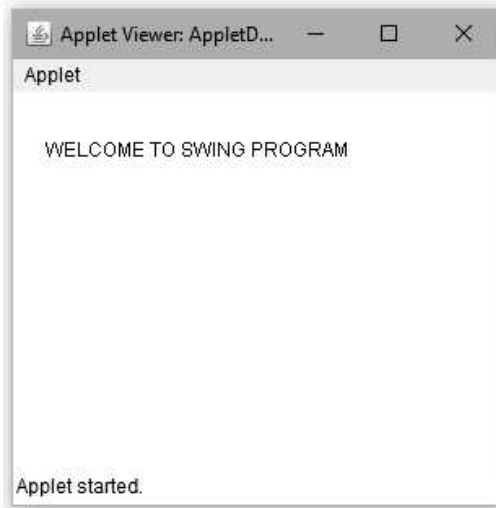
Q.1 Pluggable look and feel and lightweight components are the features supported by _____.

- a) swing
- c) core Java

- b) AWT
- d) none of these

- Q.2** Swing is based on ____ architecture.
- a client server b model view controller
 c layered d none of these
- Q.3** Swing is not a part of JFC (Java Foundation Classes) that is used to create GUI application.
- a True b False
- Q.4** The Java Foundation Classes (JFC) is a set of GUI components which simplify the development of desktop applications.
- a True
 b False
- Q.5** Following letter used as a prefix to swing component ____.
- a A b S
 c G d J
- Q.6** ____ is one of the features of object oriented programming that allows the creation of hierarchical classifications.
- a Polymorphism b Class
 c Inheritance d Object
- Q.7** In Swing JButton class is derived from ____.
- a AbstractButton b JToggleButton
 c JComponent d none of these
- Q.8** The JTextComponent derives two components JTextField and ____.
- a JComboBox
 b JTextArea
 c JSlider
 d all of the above
- Q.9** In Swing class hierarchy the class present at the root is ____.
- a component b window
 c container d object
- Q.10** _____ pane can be used to add component to container.
- a Glass b Content
 c Container d All of above

Q.11 Select the correct source code using swing for generating following output.



a

```
public class AppletDemo extends JApplet
{
    public void paint(Graphics g)
    {
        g.msg("WELCOME TO SWING PROGRAM",20,40);
    }
}
```

b

```
public class AppletDemo extends JApplet
{
    public void paint(Graphics g)
    {
        g.drawString("WELCOME TO SWING PROGRAM",20,40);
    }
}
```

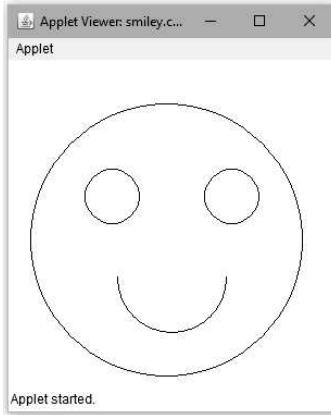
c

```
public class AppletDemo
{
    public void paint(Graphics g)
    {
        g.drawString("WELCOME TO SWING PROGRAM",20,40);
    }
}
```

d

```
public class AppletDemo extends JApplet
{
    public void paint(Graphics g)
    {
        g.display("WELCOME TO SWING PROGRAM",20,40);
    }
}
```

Q.12 The components used to display following image are _____.



- a two circle, two arcs
 b three circle, one arc and one rectangle
 c four circles
 d three circles and one arc

Q.13 To represent Icon file in swing label we use _____.

- a setimg
 b setIcon
 c serLabelIcon
 d none of this

Q.14 Which of the following component allows multiple selection ?

- a JList
 b JComboBox
 c JLabel
 d All of the above

Q.15



To generate above output we need _____.

- a List, choiceButton, ImageIcon
- b ComboBox, Image Icon, Label
- c List and Image Icon
- d ComboBox and Image Icone

Q.16 To generate following output the components that are used are _____.



- a Checkbox, Textbox
- b Radiobutton, Textbox
- c Checkbox, button
- d List, Textbox

Q.17 The subclass of JToggleButton is _____.

- a JButton
- b JCheckBox
- c JRadioButton
- d both b and c

Q.18 The Swing Component classes that are used in Encapsulates a mutually exclusive set of buttons ?

a) AbstractButton

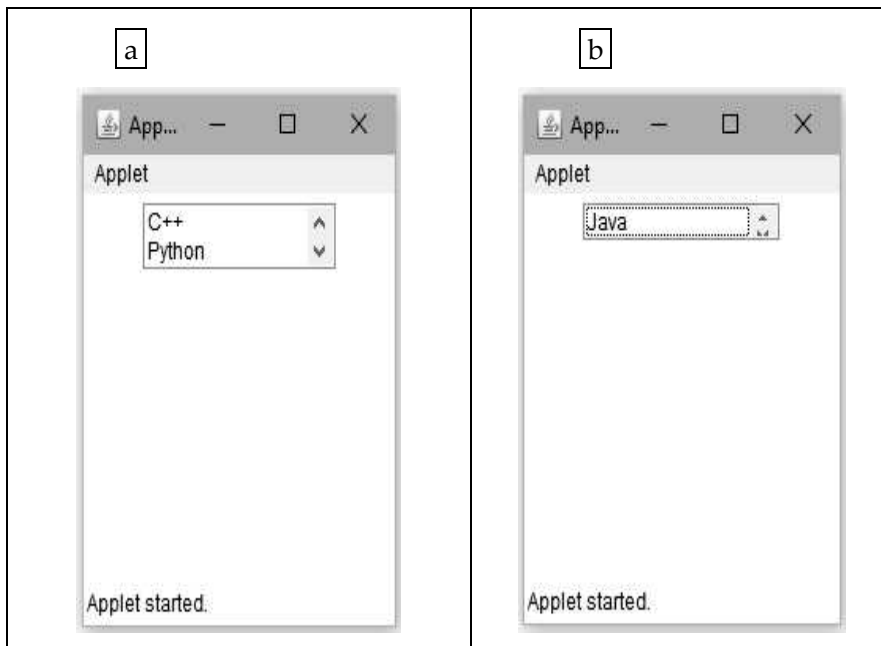
b) ButtonGroup

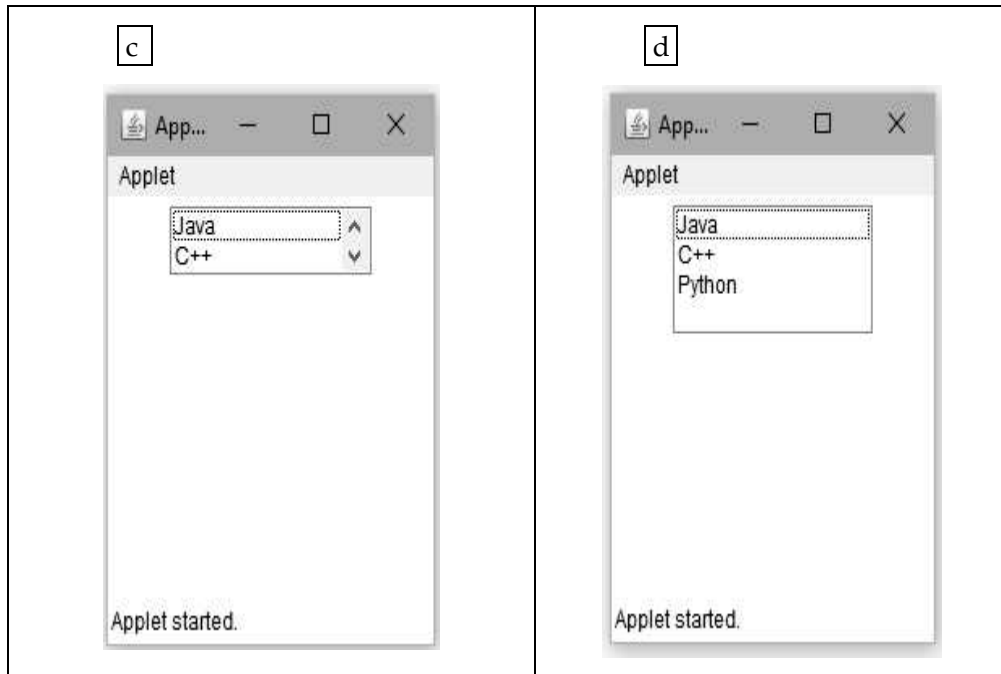
c) JButton

d) ImageIcon

Q.19 Select the correct output generated by following code.

```
import java.awt.*;
import java.applet.*;
/*<applet code=Test.class height=200 width=200>
</applet>*/
public class Test extends Applet
{
    public void init()
    {
        List l= new List(2,true);
        l.add("Java");
        l.add("C++");
        l.add("Python");
        add(l);
    }
}
```





Q.20 Which method of the component class is used to set the position and size of a component ?

- a) setPosition b) setBounds
 c) setSize d) none of these

Q.21 Select the correct option.

I. Canvas is a component.

II. ScrollPane is a container.

- a) I is True and II is False b) I is False and II is True
 c) I and II both are False d) I and II both are true

Q.22 The difference between Scrollbar and Scrollpane is _____.

- a) Scrollbar is component and Scrollpane is container
 b) Scrollbar is container and Scrollpane is component
 c) Scrollbar and Scrollpane both are components and not containers
 d) Scrollbar and Scrollpane both are container and not components

Q.23 Frame class Extends Window.

- a) True b) False

Q.24 Which is the container class ?

a Window

c Dialog

b Frame

d All of the above

Q.25 Following is uneditable control.

a Button

c Label

b Textfield

d List

Q.26 Debug the following program

```
import javax.swing.*;
import java.awt.*;
import javax.swing.tree.*;
/*
<applet code="TableDemo" width=300 height=100>
</applet>
*/
public class TableDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        final String[] th = { "Name", "City", "Salary", "Designation" };
        final Object[][] mytable = {
            { "Arun", "Pune", "5000", "Accountant" },
            { "Archana", "Mumbai", "7000", "Executive" },
            { "Shivani", "Banglore", "10000", "Manager" },
            { "Priyanka", "Chennai", "8000", "Programmer" },
            { "Monika", "Hyderabad", "10000", "Designer" },
            { "Shilpa", "Hyderabad", "12000", "Director" },
            { "Anuja", "Delhi", "17000", "Director" },
            { "Kumar", "Pune", "10000", "Manager" }
        };

        JTable table = new JTable(mytable);

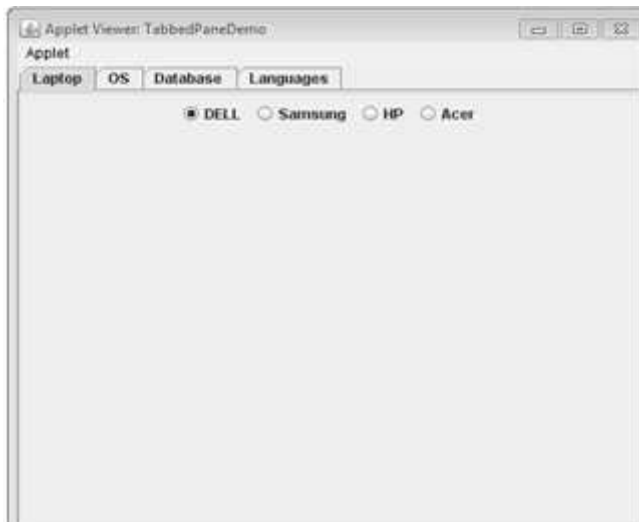
        int vscrollbar = ScrollPaneConstants.
        VERTICAL_SCROLLBAR_AS_NEEDED;
        int hscrollbar = ScrollPaneConstants.
        HORIZONTAL_SCROLLBAR_AS_NEEDED;
        JScrollPane mypane = new JScrollPane(table, vscrollbar, hscrollbar);
        contentPane.add(mypane, BorderLayout.CENTER);
    }
}
```


- a Error in statement in which jTable is created.
- b Error in statement in which JScrollPane is created.
- c Error in statement in which applet tag is declared.
- d None of these

Q.27 JPanel and Applet use _____ as their default layout.

- a FlowLayout
- b GridLayout
- c BorderLayout
- d GridBagLayout

Q.28 Which components are used to generate following output ?



- a Panel, TabbedPane, Radio button
- b TabbedPane, List
- c TabbedPane, Panel
- d Label, TabbedPane, Checkbox

Q.29 MVC stands for _____.

- a Model Version Control
- b Model View Controller
- c Mini View Controller
- d Major View Controller

Q.30 MVC architecture is used by swing.

- a True
- b False

Q.31 In swing ___ gives the visual representation of the component.

- a model
- b view
- c controller
- d none of these

Q.32 In swing the event handling task is carried out by ____.

- a model
- b view
- c controller
- d none of these

Q.33 _____ represents enterprise data and the business rules that gives access to enterprise data.

- a Model
- b View
- c Controller
- d None of these

Answer Keys for Multiple Choice Questions :

Q.1	a	Q.2	b	Q.3	b	Q.4	a
Q.5	d	Q.6	c	Q.7	a	Q.8	b
Q.9	d	Q.10	b	Q.11	b	Q.12	d
Q.13	b	Q.14	a	Q.15	b	Q.16	a
Q.17	d	Q.18	b	Q.19	c	Q.20	b
Q.21	d	Q.22	a	Q.23	a	Q.24	d
Q.25	c	Q.26	a	Q.27	a	Q.28	a
Q.29	b	Q.30	a	Q.31	b	Q.32	c
Q.33	a						

Explanations :

Q.19 : In above code we have pass 2 as a first parameter to the constructor **list**. Hence first two items of the list will be displayed. The second parameter to **list** is true that means this list is a scrollable. Hence the correct option is c.

Q.21 : Canvas is a rectangular area where the application can draw or trap input events. ScrollPane implements horizontal and vertical scrolling.

Q.26 : While creating the JTable we need to pass both table data and header data. The correct statement is

```
JTable table = new JTable(mytable,th);
```



UNIT IV

4

Database Programming using JDBC

Syllabus

The Concept of JDBC, JDBC Driver Types & Architecture, JDBC Packages, A Brief Overview of the JDBC process, Database Connection, Connecting to non-conventional Databases Java Data Based Client / server, Basic JDBC program Concept, Statement, Result Set, Prepared Statement, Callable Statement, Executing SQL commands, Executing queries.

Contents

- 4.1 The Concept of JDBC
- 4.2 Types of JDBC Drivers
- 4.3 JDBC Architecture
- 4.4 JDBC Packages
- 4.5 A Brief Overview of the JDBC Process
- 4.6 Executing SQL Commands
- 4.7 Database Connection
- 4.8 Basic JDBC Program Concept
- 4.9 Executing Queries
- 4.10 Statement
- 4.11 Result Set
- 4.12 Multiple Choice Questions

4.1 The Concept of JDBC

- ODBC stands for **Open Database Connectivity**. It is basically API i.e. Application Programming Interface.
- Using ODBC driver interface created by Microsoft, an application can access the data present in Database Management System (DBMS).
- For accessing the data from DBMS, we normally make use of Structured Query Language (SQL) statements which is popularly known as **SQL Queries**.
- JDBC stands for **Java DataBase Connectivity**. JDBC is nothing but an API (i.e. Application Programming Interface).
- It consists of various classes, interfaces, exceptions using which Java application can send SQL statements to a database. The SQL is a Structured Query Language used for accessing the database.
- JDBC is useful for both application developers and JDBC driver vendors.
- The JDBC specification is prepared by Sun Microsystems. Any third party vendor can design their own JDBC drivers using this specification. These JDBC drivers are then used by the application developers for getting connected to the database.
- JDBC is specially used for having connectivity with the RDBMS packages (such as Oracle or MYSQL) using corresponding JDBC driver.

Review Question

1. What is JDBC - ODBC ?

4.2 Types of JDBC Drivers

- There are four types of JDBC drivers and those are -
 1. **Type 1** : JDBC-ODBCBridge
 2. **Type 2** : Native-API/Partly Java Driver
 3. **Type 3** : All JAVA/ Net Protocol driver for accessing middleware server.
 4. **Type 4** : All JAVA/ Native-Protocol Pure driver
- Let us discuss them in detail -

Type 1 : JDBC-ODBCBridge

This driver translates all the JDBC calls into ODBC (Open Database Connectivity) calls and send them to ODBC driver. Thus JDBC access is via ODBC driver. The ODBC is a generic API. In this scenario the client database code must be present on the client machine.

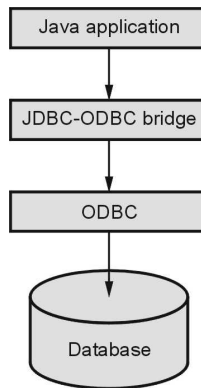


Fig. 4.2.1 JDBC-ODBC bridge

Merit

1. Using the JDBC-ODBC bridge access to any database is possible.

Demerits

1. This is slowest driver because the calls are sent to ODBC driver and then to the native database connectivity interface.
2. This type of driver is not suitable for large scale applications.
3. For using this type of driver the native database must be present on the client machine and the ODBC driver must be installed on the client's machine.

Type 2 : Native-API/Partly Java Driver

This driver translates all the JDBC calls into database-specific calls. This driver works specifically for particular database. For example MYSQL will have native MYSQL API. This type of driver directly communicates with the database server. Hence some binary code must be present on the client machine.

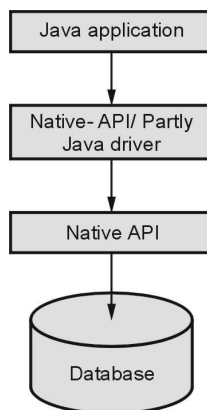


Fig. 4.2.2 Type - 2 driver

Merit

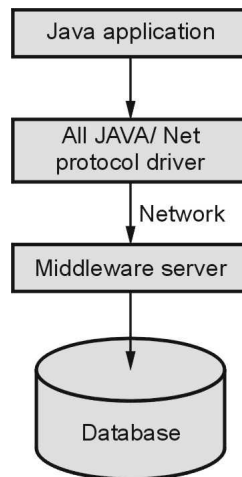
1. It gives better performance as comparison with type -1 driver because the JDBC call is directly converted to database specific call.

Demerits

1. The library of required databases must be loaded on the client machine.
2. This type of driver is not useful for the internet.
3. If some modifications in made in the database then the native API must also be modified because it is specific to a database.

Type 3 : All JAVA/ Net Protocol driver for accessing middleware server

- In this type of driver all the JDBC calls are passed through the network to the middle-ware server. The middleware server then translates the request to the database-specific native-connectivity interface and then the request is sent to the database server.
- This driver is a server-based driver. This is also known as a pure Java driver.

**Fig. 4.2.3 Type - 3 driver****Merits**

1. As it is server-based driver there is no need to keep library of required databases on the client machine.
2. This driver is fully written in JAVA (hence is the name all Java) and hence it is portable and can be used on internet.
3. The performance of this driver can be optimized.

4. This driver supports many advanced features such as load balancing, caching and logging.
5. For this driver it is possible to access multiple databases using one driver.

Demerits

1. The middleware server application needs to be installed and maintained.
2. The record set has to traverse through the backend server.

Type 4 : All JAVA / Native-Protocol Pure driver

This type of driver converts the JDBC calls to network protocol used by the database directory so that the client application can directly communicate to the database server. This driver is also completely implemented in Java and hence it is referred as Pure Java driver.

Merits

1. As this driver is completely written in Java, it is platform independent and can be used on Internet.
2. There is no translation layer in between such as to ODBC or to native API. Neither there is a need to send the call to middle ware server. Hence the performance of this type of driver is typically good.
3. There is no need to install specific software on the client machine.
4. These drivers can be downloaded dynamically.

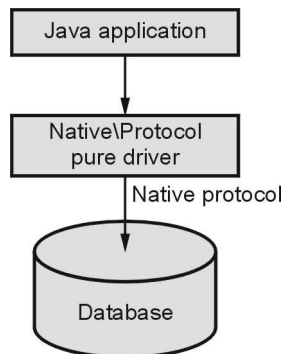


Fig. 4.2.4 Type - 4 driver

Demerit

1. When the type 4 driver is used then for each database a specific driver is needed.

Review Question

1. Explain driver types of JDBC.

4.3 JDBC Architecture

- JDBC API supports both two-tier and three-tier processing model.

4.3.1 Two Tier Model

- In two tier model, Java application can directly communicate with database. For this communication, the JDBC driver API is required.
- The Two Tier model is represented by following Fig. 4.3.1.

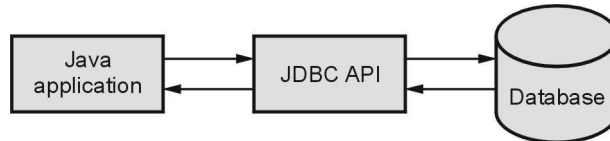


Fig. 4.3.1

4.3.2 Three Tier Model

In this architecture, the HTML browser will send the command to Java Application. The Java application will then communicate with database through JDBC API. The architecture is as shown below -

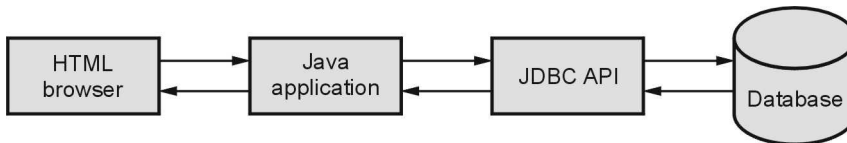


Fig. 4.3.2

Review Question

1. Explain JDBC architecture.

4.4 JDBC Packages

- The `javax.sql` is an API which contains several classes and interfaces for data source access.
- Various useful classes in `javax.sql.*` package are :

Class name	Use
ConnectionEvent	It is useful for getting the information of the source of connection related event.
PooledConnection	It allows an access to a connection pool management.

RowSet	It extends the ResultSet interface for supporting disconnected result sets.
XAConnection	This interface is useful for distributed transactions.

- The **javax.sql.*** is a supplementary package for **java.sql** package. Various features of this package are -
 1. The **DataSource** interface serves as an alternative to **DriverManager** class. The main advantage of using DataSource is that the applications do not need to hard code the driver class.
 2. It provides support for distributed transactions. In distributed transactions, using single transaction, data sources on multiple servers can be used by an application. The classes and interfaces used in distributed transactions are -
 - XADataSource
 - XAConnection
 3. The connection pooling is possible. The connection pooling is a mechanism by which connections can be reused. This ultimately improves the overall performance.
 4. It allows the use of RowSet.

Mapping between JAVA and SQL

- When we wish to move the application from JAVA to SQL or vice versa then we require JDBC driver to map the data types.
- SQL data type for corresponding JAVA primitive is given by following table -

JAVA	SQL
boolean	BIT
byte	TINYINT
int	INTEGER
short	SMALLINT
long	BIGINT
float	REAL
double	DOUBLE

String	VARCHAR
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
Byte []	BINARY

- Some databases treat INTEGER data type as NUMERIC. We can access the SQL datatype of each column in the database using `getColumnType()` function.

4.5 A Brief Overview of the JDBC Process

- Following is a way by which JDBC works -
 - 1) First of all Java application establishes connection with the data source.
 - 2) Then Java application invokes classes and interfaces from JDBC driver for sending queries to the data source.
 - 3) The JDBC driver connects to corresponding database and retrieves the result.
 - 4) These results are based on SQL statements which are then returned to Java application.
 - 5) Java application then uses the retrieved information for further processing.

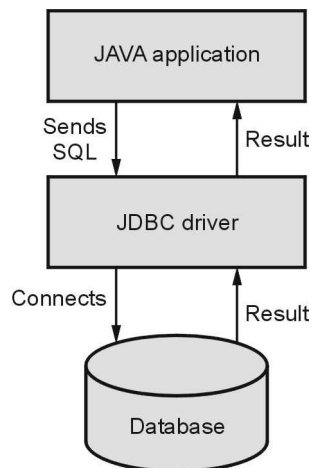


Fig. 4.5.1

4.6 Executing SQL Commands

- The MYSQL is a open source database product which can be downloaded from the web site.

- After getting installed on the machine the command prompt window for MYSQL can appear. The screenshot is as follows -



- Now let us go through some MYSQL query statements which are required while handling the database.

1. Creating database

```
mysql> CREATE DATABASE mydb;  
Query OK, 1 row affected (0.15 sec)
```

2. Displaying all the databases

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| mydb     |  
| mysql    |  
| students |  
| test     |  
+-----+  
4 rows in set (0.06 sec)
```

3. Selecting particular database

```
mysql> USE MYDB;  
Database changed
```

4. Creating table

- We must create a table inside a database hence it is a common practice to use create table command after USE database command. While creating a table we must specify the table fields.

```
mysql> CREATE TABLE my_table(id INT(4),name VARCHAR(20));
Query OK, 0 rows affected (0.28 sec)
```

- **Use of Primary Key :** The primary key contains the unique value. The primary key column can not contain NUL value. Each table can have only one primary key. Following is a SQL statement used to create PRIMARY KEY.

```
CREATE TABLE student_table(
roll_no INT(4) NOT NULL AUTO_INCREMENT,
name VARCHAR(50) NOT NULL,
address VARACHAR(50) NOT NULL,
PRIMARY KEY(roll_no)
);
```

- In above example the roll_no acts as a primary key for the student_table.

5. Displaying a table

- After creating the table using SHOW command we can see all the existing tables in the current database.

```
mysql> SHOW TABLES;
+-----+
| Tables_in_mydb |
+-----+
| my_table      |
+-----+
1 row in set (0.00 sec)
```

6. Displaying the table fields

- For knowing the various fields of the table we may use following command.

```
mysql> DESCRIBE my_table;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(4)    | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.07 sec)
```

7. Inserting values into the table

- We can insert only one complete record at a time. It is as shown below -

```
mysql> INSERT INTO my_table
-> VALUES(1,'SHILPA');
Query OK, 1 row affected (0.05 sec)
```

8. Displaying the contents of the table

```
mysql> SELECT * FROM my_table;
+-----+-----+
| id | name |
+-----+-----+
| 1 | SHILPA |
+-----+-----+
1 row in set (0.06 sec)
```

- We can also write SELECT statement for selecting particular row by specifying some condition such as -

```
mysql> SELECT * FROM my_table where id=1;
or
mysql> SELECT * FROM my_table where name='SHILPA';
```

- Thus we can insert the rows into the table by repeatedly giving the INSERT command.
- If we want to get the records in sorted manner then we use ORDER BY clause

```
mysql> SELECT * FROM my_table;
+-----+-----+
| id | name |
+-----+-----+
| 1 | SHILPA |
| 2 | SUPRIYA |
| 3 | YOGESH |
| 4 | MONIKA |
+-----+-----+
4 rows in set (0.00 sec)
mysql> SELECT * FROM my_table ORDER BY name;
+-----+-----+
| id | name |
+-----+-----+
| 4 | MONIKA |
| 1 | SHILPA |
| 2 | SUPRIYA |
| 3 | YOGESH |
+-----+-----+
4 rows in set (0.00 sec)
```

9. Updating the record

- For updating the record in the database following command can be used -

```
mysql> UPDATE my_table
-> SET name='PRIYANKA'
-> WHERE id=4;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> SELECT * FROM my_table;
+-----+-----+
| id | name      |
+-----+-----+
| 1 | SHILPA   |
| 2 | SUPRIYA  |
| 3 | YOGESH   |
| 4 | PRIYANKA |
+-----+-----+
4 rows in set (0.00 sec)
```

10. Deleting record

- For deleting particular record from a database

```
mysql> DELETE FROM my_table
-> WHERE id=3;
Query OK, 1 row affected (0.04 sec)
```

- Then use SELECT statement for displaying the contents of the table we use following command

```
mysql> SELECT * FROM my_table;
+-----+-----+
| id | name      |
+-----+-----+
| 1 | SHILPA   |
| 2 | SUPRIYA  |
| 4 | PRIYANKA |
+-----+-----+
3 rows in set (0.00 sec)
```

11. For deleting the table

- The table can be deleted using the command.

```
mysql> drop table my_table;
```

12. Inner join

- This query used to display the result from both the tables when at least one column from both the tables is matching -

For example

Consider table **Customer**

Cust_Id	Name	City
1	Rahul	Bombay
2	Priyanka	Pune
3	Supriya	Banglore

Consider table **Order**

Order_Id	Cust_Id	Order_Number
10	3	1234
20	1	5678
30	1	8900

- Then we can make use of inner join query as follows -

```
SELECT Customer.Name, Customer.city, Order.Order_Number
FROM Customer
INNER JOIN Order
ON Customer.Cust_Id=Order.Order_Id
```

Name	City	Order_Number
Rahul	Bombay	5678
Rahul	Bombay	8900

13. Order by clause

- If we want to get the records in sorted manner then we use ORDER BY clause

```
mysql> SELECT * FROM my_table;
```

id	name
1	SHILPA
2	SUPRIYA
3	YOGESH
4	MONIKA

```
mysql> SELECT * FROM my_table ORDER BY name;
```

id	name
4	MONIKA
1	SHILPA
2	SUPRIYA
3	YOGESH

4.7 Database Connection

- **Prerequisite** : We need following things for connecting to database Using JDBC
 1. MySQL is already Installed and required database is created.
 2. JDK is installed.
 3. A database connector `mysql-connector-java-X.X.XX-bin` must be downloaded and present at the path.

```
C:\your_tomcat_directory\common\lib
```

4. Tomcat is running.
- For connecting java application with the mysql database, Following steps are needed to follow -
 1. **Driver class** : The driver class for the mysql database is **`com.mysql.jdbc.Driver`**. Here we can instantiate the object for JDBC driver by using following statement.

```
Class.forName("com.mysql.jdbc.Driver");
```

2. **Connection URL** : The connection URL for the mysql is

```
databases jdbc:mysql://localhost:3306/my_database
```

Where

- `jdbc` is the API,
- `mysql` is the database,
- `localhost` is the server name on which mysql is running, we may also use IP address, `3306` is the port number
- **`my_database`** is the database name. (We may use any database, in such case, you need to replace the `my_database` with your database name.)
- **Username** : The default username for the mysql database is **`root`**. If you haven't set any user name it could be blank.
- **Password** : Password is given by the user at the time of installing the mysql database.

The connection URL can be obtained using the method **`getConnection`**. This method is defined in class named **`DriverManager`**. Note that this **`DriverManager`** class is defined in **`java.sql.*`** package. We can obtain the instance for connection using following statement

```
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/my_database","","");
```

- The list of common drivers along with JDBC URL is,

1. ODBC Driver

Syntax :

```
jdbc:odbc:<data source name>
```


Example :

```
jdbc:odbc:MyDataSource
```

2. MYSQL**Syntax :**

```
jdbc:mysql://[host][:port]/[database]
```

Example :

```
jdbc:mysql://localhost:3306/Mydatabase
```

3. ORACLE**Syntax :**

```
jdbc:oracle:<drivertype>:<user>/<password>@<database>
```

Example :

```
jdbc:oracle:thin:myuser/mypassword@localhost:mydatabase
```

4.8 Basic JDBC Program Concept

- Following steps are used to connect JDBC to MYSQL.

Step 1 : Import java.sql.* package in the JDBC program

Following line can be included in your JDBC program at the beginning.

```
import java.sql.*
```

Step 2 : Load JDBC Driver.

The JDBC driver for MYSQL can be loaded using following statement

```
Class.forName("com.mysql.jdbc.Driver");
```

Step 3 : Get connection using the Driver Manager

```
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/my_database","root","password");
```

Step 4 : Create Statement

```
stmt = conn.createStatement();
```

Step 5 : Execute Query

```
String sql = "SELECT Roll,StudName FROM my_table";  
ResultSet rs = stmt.executeQuery(sql);
```

Step 6 : Display the result.

4.9 Executing Queries

The executeQuery() Method

- The **executeQuery()** method is belonging to `java.sql.Statement` interface of JDBC.
- This method is used for SQL statements which retrieve some data from the database. For example is **SELECT** statement.
- This method is basically used for **SELECT** queries which fetch some data from the database. This method returns one **java.sql.ResultSet** object which contains the data returned by the query.
- This method throws the exception **SQLException**.
- The general syntax is :

```
ResultSet executeQuery(String SQL)
```

The executeUpdate() Method

- The **executeUpdate()** method is used to update the database.
- This method makes used of all the Data Manipulation Language (DML) statements such as **INSERT**, **DELETE**, **UPDATE** or Data Definition Language (DDL) such as **CREATE**, **ALTER** and so on.
- This method returns an **int value** which represents the number of rows affected by the query. This value will be 0 for the statements which return nothing.
- This method throws the exception **SQLException**.
- The general syntax of this method is

```
int executeUpdate(String sql)
```

Difference between executeQuery() and executeUpdate() Method

executeQuery()	executeUpdate()
This method is used to execute SQL statements in order to retrieve data from the database.	This method is used to execute SQL statements which are intended to update or modify the database.
This method returns a ResultSet object in which the result of the query is stored.	This method returns integer value that represent number of rows affected by the query.
This method is normally used to execute SELECT queries.	This method is used to execute non SELECT queries. That is- DML as INSERT , DELETE , UPDATE or DDL as CREATE , DROP

4.9.1 CREATE Statement

- The database can be created using SQL query as follows -

```
CREATE DATABASE databasename;
```

- For example we can create a database named My_database as follows -

```
CREATE DATABASE My_database;
```

- After creating the database, a table can be created within it. The query used for creating a table is

```
CREATE TABLE table_name  
(  
  column_name1 data_type(size),  
  column_name2 data_type(size),  
  column_name3 data_type(size),  
  ....  
);
```

- The *column_name* specifies the name of the columns of the table, *data_type* specifies the type of data the column field has. The *size* field specifies the *size* of the column in the table.
- For example

```
CREATE TABLE My_table  
(  
  Roll INT,  
  StudName VARCHAR(20)  
);
```

- With the help of Java we can execute the query using following statement.

```
Statement stat = con.createStatement();  
ResultSet rs = stat.executeQuery("CREATE TABLE MY_table  
(Roll INT, StudName VARCHAR(20))");
```

- Here is a simple Java program that illustrates how to create a table using SQL statement.

Java Program

```
import java.sql.*;  
public class JDBCdemo1  
{  
  public static void main(String [ ] args)  
  {  
    Connection con = null;  
    try  
    {  
      Class.forName("com.mysql.jdbc.Driver");
```

```
con = DriverManager.getConnection("jdbc:mysql://localhost/my_database","root","");
System.out.println("Connection Successful!");
Statement stat=con.createStatement();
    int result=stat.executeUpdate("CREATE TABLE My_table(Roll INT,StudName
    VARCHAR(20))");
System.out.println("Table Created");
stat.close();
con.close();
}
catch (ClassNotFoundException e)
{
    System.err.println("Exception: "+e.getMessage());
}
catch (SQLException e)
{
    System.err.println("Exception: "+e.getMessage());
}
}
}
```

Program Explanation : In the above program,

- 1) We are first establishing the connection with the database and then using following statements particular SQL statement can be executed.

```
Statement stat=con.createStatement();
int result= stat.executeUpdate("CREATE TABLE My_table(Roll INT, StudName
VARCHAR(20))");
```

- 2) First of all the object or the instance of **Statement** named *stat* is created.
- 3) Then the **executeUpdate()** method of that object is written in which the complete SQL query is passed. Thus we can create the table in our database.

Output

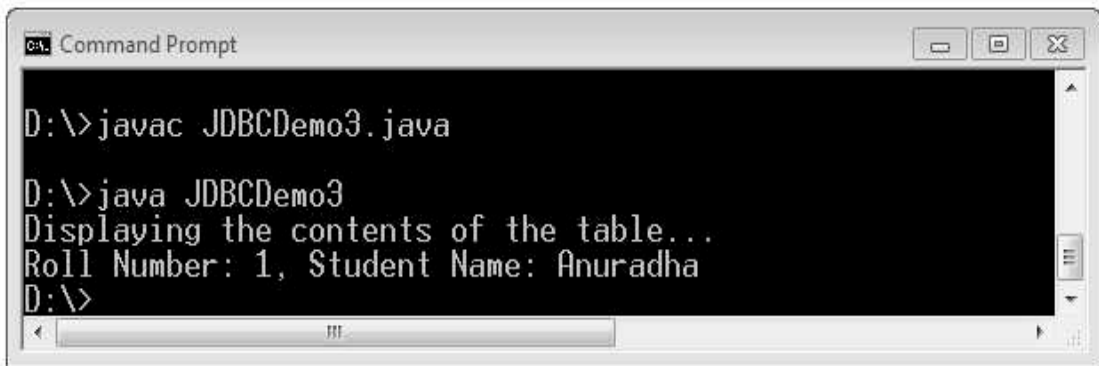
```
c:\test>javac JBDCDemo1.java
c:\test>java JBDCDemo1
Connection Successful!
Table Created
```

4.9.2 SELECT Statement

- Using the **select** statement we can get the result from the database the result is obtained in the instance of **ResultSet**. The **executeQuery** function returns the **ResultSet** Object. The syntax of **executeQuery** is

```
ResultSet executeQuery(Strin SQL);
```


Output



```
Command Prompt
D:\>javac JDBCdemo3.java
D:\>java JDBCdemo3
Displaying the contents of the table...
Roll Number: 1, Student Name: Anuradha
D:\>
```

4.9.3 UPDATE Statement

- For updating data in the database the UPDATE query must be executed. The syntax of UPDATE command is

```
UPDATE table_name
SET column1 = value1, column2 = value2..., columnN = valueN
WHERE [condition];
```

- The **condition** can be combined using AND and OR operators.
- The steps to update the data in the database are as follows -

Step 1 : Create the instance for Connection class using **DriverManager's getConnection** method.

Step 2 : Then invoke the createStatement method in order to create the object for **Statement** class.

Step 3 : With the help of Statement class object the method **executeUpdate** will be invoked. The SQL query for UPDATE record can be passed as a parameter to **executeUpdate** method. The general syntax for executeUpdate is

```
public int executeUpdate(java.lang.String sql)
```

The **sql** represents the Query string using INSERT, DELETE or UPDATE.

The return value is integer which indicates number of rows affected.

Following is a Java program that updates the record stored in the table of a database.

JDBCdemo4.java

```
import java.sql.*;
public class JDBCdemo4 {
public static void main(String [ ] args) {
```

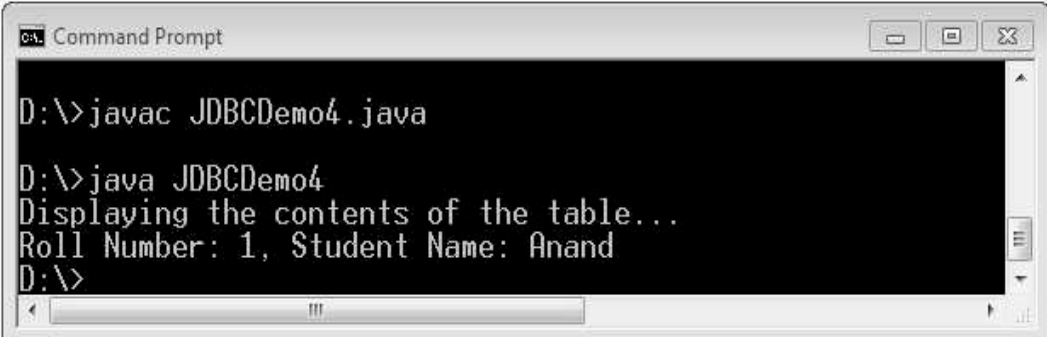
```
Connection con = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://localhost/my_database","root","");
    Statement stat=con.createStatement();

    String sql="UPDATE My_table " + "SET StudName='Anand' WHERE Roll=1";
    stat.executeUpdate(sql);

    sql="SELECT * FROM My_table";
    ResultSet rs=stat.executeQuery(sql);

    System.out.println("Displaying the contents of the table...");
    while(rs.next())
    {
        int RollNo = rs.getInt("Roll");
        String Name = rs.getString("StudName");
        //Display values
        System.out.print("Roll Number: " + RollNo);
        System.out.print(", Student Name: " + Name);
    }
    rs.close();
    stat.close();
    con.close();
}
catch (ClassNotFoundException e) {
System.err.println("Exception: "+e.getMessage());
}
catch (SQLException e) {
    System.err.println("Exception: "+e.getMessage());
}
}
```

Output



```
Command Prompt
D:\>javac JDBCdemo4.java
D:\>java JDBCdemo4
Displaying the contents of the table...
Roll Number: 1, Student Name: Anand
D:\>
```

Example 4.9.1 Consider bank table with attributes AccountNo, CustomerName, Balance, Phone and Address. Write a database application which allows insertion, updation and deletion of records in Bank table. Print values of all customers whose balance is greater than 20,000.

Solution : Step 1 : Create a database using suitable database such as MySQL/Microsoft Access/Oracle. The name of the database is **bankdb** and the name of the table created is **banktable**.

The sample database table will be as follows -

AccountNo	CustomerName	Balance	Phone	Address
11	AAA	10000	1111111111	Pune
22	BBB	20000	2222222222	Chennai
44	DDD	30000	4444444444	Delhi
0	0	0	0	0

Step 2 : The java program for handling this database for given operations can be written as follows. Here we have taken the file name as **test.java**.

```
import java.sql.*;
public class test {
    public static void main(String [] args) {
        Connection con = null;
        int result;
        ResultSet rs = null;
        Statement stat=null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection("jdbc:odbc:bankdb", " ", " ");
            stat = con.createStatement();
            result = stat.executeUpdate("INSERT INTO banktable" +
            "(AccountNo,CustomerName,Balance,Phone,Address)" +
            "VALUES(55,'EEE',40000,555555555,'Pune')");
            System.out.println("Values inserted in table!");
            result = stat.executeUpdate("DELETE FROM banktable WHERE AccountNo=33");
            System.out.println("Values deleted from table!");
            result = stat.executeUpdate("UPDATE banktable set AccountNo=100 WHERE
            AccountNo=11");
            System.out.println("Values updated from table and updated record is as follows....");
        }
    }
}
```



```
rs = stat.executeQuery("SELECT * FROM banktable WHERE AccountNo=100");

while (rs.next())
{
    System.out.println("AccountNo: " + rs.getObject(1).toString());
    System.out.println("CustomerName: " + rs.getObject(2).toString());
    System.out.println("Balance: " + rs.getObject(3).toString());
    System.out.println("Phone: " + rs.getObject(4).toString());
    System.out.println("Address: " + rs.getObject(5).toString());
}
rs = null;
System.out.println("Following records having salary>20000...");
rs = stat.executeQuery("SELECT * FROM banktable WHERE Balance>20000");
System.out.println("AccountNo  Name  Balance");
while (rs.next())
{
    if (rs != null)
        System.out.println(rs.getObject(1).toString() + " " + rs.getObject(2).toString() +
            " " + rs.getObject(3).toString());
}
}
catch (ClassNotFoundException e)
{
    System.err.println("Exception: " + e.getMessage());
}
catch (SQLException e)
{
    System.err.println("Exception: " + e.getMessage());
}
finally
{
    try
    {
        if(rs!=null)
        {
            rs.close();
            rs=null;
        }
        if(stat!=null)
        {
            stat.close();
            stat=null;
        }
        if (con != null)
        {
            con.close();
        }
    }
}
```

```
        con = null;
    }

    }catch (SQLException e) { }
}
}
```

4.10 Statement

- A **Statement** object is used for executing a static SQL statement and obtaining the results produced by it.
- The **Statement** interface can not accept parameters.

Creation of Statement Object

- The statement object can be created using the **createStatement()** method. Following is a illustrative Java code.

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://localhost/my_database","root","");
    Statement stat=con.createStatement();
    ...
}
catch(SQLException ex) {
    ...
}
```

- After creating the statement object one of the following three methods can be invoked.

1. **execute** : It Returns true if a **ResultSet** object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statement.

The general syntax is

```
bool execute(String Q)
```

2. **executeUpdate** : It returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements using the INSERT, UPDATE or DELETE statement. The general syntax is

```
int executeUpdate(String Q)
```

3. **executeQuery** : It returns a **ResultSet** object. Use this method when you expect to get a result set. Normally this statement is used with a SELECT statement. The general syntax is

```
ResultSet executeQuery(String Q)
```

Closing Statement Object

Using a call to **close** method the Statement object can be closed. For instance

```
Stat.close();
```

Types of Statements :

There are two types of statements.

1. Prepared statement
2. Callable statement

Let us discuss them in detail.

4.10.1 Prepared Statement

- The **java.sql.PreparedStatement** interface object represents a precompiled SQL statement.
- This interface is used to efficiently execute SQL statements multiple times. That is when we want to insert a record in a table by putting different values at runtime.
- This statement is derived from the Statement class.
- The PreparedStatement interface can be created by calling `PreparedStatement()` method.
- The `prepareStatement()` is available in **java.sql.Connection** interface.
- The `prepareStatement()` method takes SQL statement in java format.

```
prepareStatement("insert into student values(?,?)").
```

where each ? represents the column index number in the table. If table **student** has **rollnumber** and **name** columns, then ? refers to **rollnumber**, ? refers to **name**.

- After that we need to set the value to each ? by using the setter method from PreparedStatement interface as follows :

```
setXXX(ColumnIndex,value)
```

- Various setter methods are

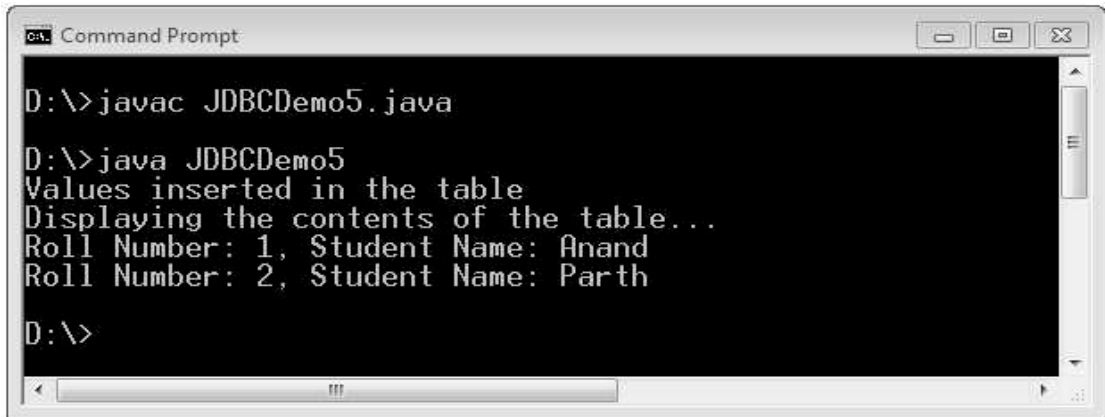
SQL datatype	Method used
char/varchar/varchar2	setString()
int/number	setInt()
float/number	setFloat()
double/Float	setDouble()
long/int	setLong()
int/short	setShort()
time	setTime()
datetime/date	setDate()

Following is a Java Program that makes use of Prepared Statement for inserting data in the table.

JDBCDemo5.java

```
import java.sql.*;
public class JDBCDemo5 {
    public static void main(String [ ] args) {
        Connection con = null;
        try {
            int rollnum=2;
            String name="Parth";
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost/my_database","root","");
            PreparedStatement ps=con.prepareStatement("INSERT INTO My_table
            VALUES(?,?)");
            ps.setInt(1,rollnum);
            ps.setString(2,name);
            int result=ps.executeUpdate();
            if(result!=0)
                System.out.println("Values inserted in the table");
            else
                System.out.println("Values are not inserted in the table");
            String sql="SELECT * FROM My_table";
            ps=con.prepareStatement(sql);
            ResultSet rs=ps.executeQuery();
            System.out.println("Displaying the contents of the table...");
            while(rs.next()) {
                int RollNo = rs.getInt("Roll");
                String Name = rs.getString("StudName");
                //Display values
                System.out.print("Roll Number: " + RollNo);
                System.out.println(", Student Name: " + Name);
            }
            rs.close();
            ps.close();
            con.close();
        }
        catch (ClassNotFoundException e) {
            System.err.println("Exception: "+e.getMessage());
        }
        catch (SQLException e) {
            System.err.println("Exception: "+e.getMessage());
        }
    }
}
```

Output



```

C:\> Command Prompt
D:\> javac JDBCdemo5.java
D:\> java JDBCdemo5
Values inserted in the table
Displaying the contents of the table...
Roll Number: 1, Student Name: Anand
Roll Number: 2, Student Name: Parth
D:\>

```

The callable statement is used when we want to access the **database stored procedures**. The stored procedure is basically a block of code which is identified by unique name. Let us first understand how to create procedure.

Creating Procedure

- The procedure can be created using following syntax.

```

DELIMITER //
CREATE PROCEDURE procedureName(parameters_list)
BEGIN
SQL Statements to be executed
END //

```

Calling Procedure

When calling the stored procedure, the **CallableStatement** object is used. For this object three types of parameters are used.

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. Then the values can be associated with IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. These values are obtained in OUT parameters with the getXXX() methods.
INOUT	A parameter that supplies input as well as accepts output parameter requires a call to the appropriate setXXX method.

You can create an instance of a **CallableStatement** by calling the **prepareCall()** method on a connection object. Here is an example :

```
CallableStatement callableStatement = connection.prepareCall("{call myprocedure(?, ?)}");
```

Setting the values to parameters

We can set the values to the paramters using setXX method. For example :

```
connection.prepareCall("{call myprocdeures(?, ?)}");
callableStatement.setInt(1, 10);
callableStatement.setString (2, "AAA");
```

Executing the CallableStatement

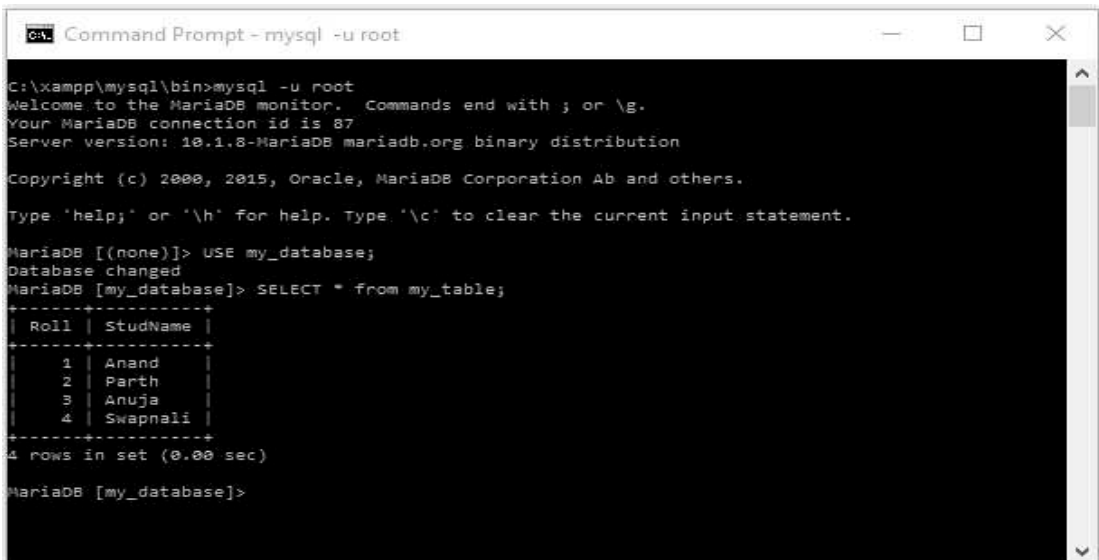
Once you have set the parameter values you need to set, you are ready to execute the CallableStatement. Here is how that is done :

```
ResultSet result = callableStatement.executeQuery();
```

Then using the instance of **ResultSet** we can display the result.

Following is a simple Java application program (illustrated in stepwise manner) that makes use of callable statement.

Step 1 : Create a database in MySQL with the table. Insert some values inside the table. The contents of the table are represented by following Screenshot.



```

C:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 87
Server version: 10.1.8-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE my_database;
Database changed
MariaDB [my_database]> SELECT * from my_table;
+----+-----+
| Roll | StudName |
+----+-----+
| 1    | Anand    |
| 2    | Parth    |
| 3    | Anuja    |
| 4    | Swapnali |
+----+-----+
4 rows in set (0.00 sec)

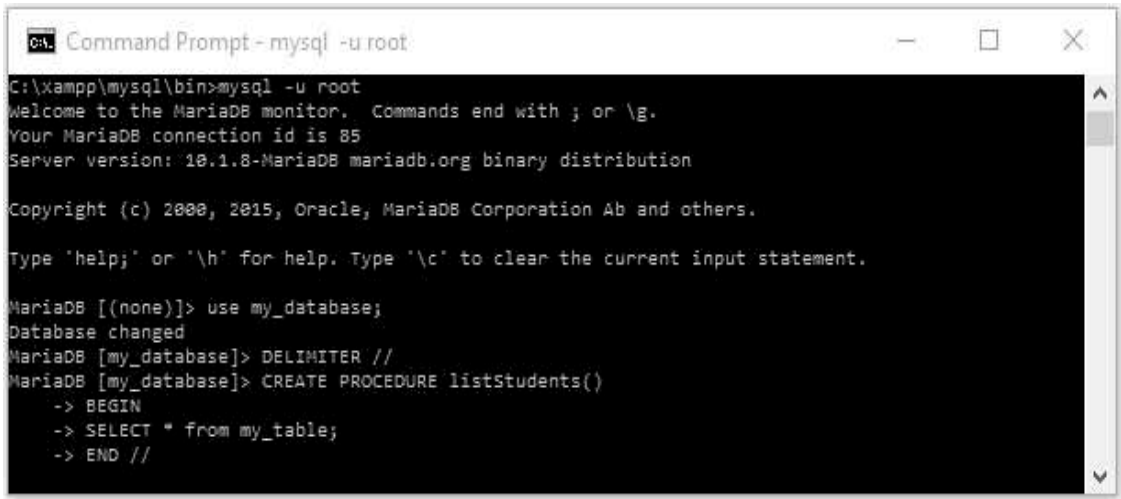
MariaDB [my_database]>

```

- The sample table **my_table** is as follows -

Roll	StudName
1	Anand
2	Parth
3	Anuja
4	Swapnali

Step 2 : Then Create a procedure as follows -



```
Command Prompt - mysql -u root
C:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 85
Server version: 10.1.8-MariaDB mariadb.org binary distribution
Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]> use my_database;
Database changed
MariaDB [my_database]> DELIMITER //
MariaDB [my_database]> CREATE PROCEDURE listStudents()
-> BEGIN
-> SELECT * from my_table;
-> END //
```

The procedure is as follows -

```
>use my_database
> DELIMITER //
>CREATE PROCEDUTR listStudents()
-> BEGIN
-> SELECT * FROM my_table
-> END //
```

Step 3 :

- Before you start writing the application make sure that the driver for MySQL is present in the /tomcat/lib directory. It should be **mysql-connector-java-5.xx.xx-bin.jar** file.
- If it is not present then it can be downloaded from the site <https://dev.mysql.com/downloads/connector>
- Go for **Platform independent option** and select Zip or Tar file option depending upon your need. Then get is downloaded and save it to tomcat's **lib** directory.
- The Java code that makes use of Callable Statement for invoking the created procedure is as given below.

JDBCDemo6.java

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
public class JDBCdemo6 {
    public static void main(String[] args)
    {
        Connection con=null;
        try {
            //Getting the driver for MySQL
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost/my_database","root","");
            //Creating instance of CallableStatement using prepare call
            //The procedure listStudents() created in Step 2 is invoked here
            CallableStatement cs=con.prepareCall("CALL listStudents()");
            ResultSet rs=cs.executeQuery();
            while(rs.next()){
                System.out.println(rs.getInt(1)+"\t"+rs.getString(2));
            }
        }catch(Exception e){
            e.printStackTrace();
        }
        finally {
            try {
                con.close();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
    }
}
```

4.11 Result Set

- The ResultSet interface is an important interface which is used to access the database table with general width and unknown length.
- The table rows are retrieved in sequence using **ResultSet** object. Within a row its column values can be accessed in any order.
- 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.
- The ResultSet object can be created using **executeQuery()** method. For example

```
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery("select * from my_table");
```


4.11.1 Navigating Methods

- Various commonly used navigating methods for ResultSet are as given in the following table.

Sr. No.	Methods	Description
1.	public boolean first() throws SQLException	Moves the cursor to the first row.
2.	public void last() throws SQLException	Moves the cursor to the last row.
3.	public boolean previous() throws SQLException	Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
4.	public boolean next() throws SQLException	Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
5.	public int getRow() throws SQLException	Returns the row number that the cursor is pointing to.

4.11.2 Reading the Result using ResultSet

- There are various methods using which the data can be retrieved using the ResultSet object. These methods can be used with either column Name or with column Index. Few commonly used methods are

Sr. No.	Methods	Description
1.	public int getInt(int index)	Returns the integer value in the current row specified by the index.
2.	public int getInt(String Name)	Returns the integer value in the current row specified by the column Name.
3.	public Date getDate(int index)	Returns the Date value in the current row specified by index.
4.	Public Date getDate(String Name)	Returns the Date value in the current row specified by Name of the column.

- Similar to **getInt** there are methods such as **getString**, **getBoolean**, **getBytes**, **getFloat**, **getDouble** and so on.

4.11.3 Updating ResultSets

- There are various methods for updating the ResultSets denoted by **updateXXX()**. Just similar to **getXX** method the update method makes use of Column Index and Column Name. These are as given below -

Sr. No.	Methods	Description
1.	Public void updateString(int <i>index</i> , String <i>new_val</i>)	Updates the string by <i>new_val</i> which is specified by <i>index</i> .
2.	Public void updateString(int <i>Name</i> , String <i>new_val</i>)	Updates the string by <i>new_val</i> which is specified by <i>Name</i> of the column.

- The row values can be updated using the methods as given below.

Sr. No.	Methods	Description
1.	public void updateRow()	Updates current row from database.
2.	public void deleteRow()	Deletes the current row from database.
3.	Public void insertRow()	Inserts the row in the database.

4.12 Multiple Choice Questions

Q.1 JDBC stands for _____.

a) Java Database Connectivity b) Java Database Control

c) Java Database Components d) None of these

Q.2 Which statements about JDBC are true ?

a) JDBC is an API to connect to relational-, object and XML data sources.

b) JDBC stands for Java DataBase connectivity.

c) JDBC is an API to access relational databases, spreadsheets and flat files.

d) JDBC is an API to bridge the object-relational mismatch between OO programs and relational databases.

Q.3 Which packages contain the JDBC classes ?

a) java.jdbc and javax.jdbc b) java.jdbc and java.jdbc.sql

c) java.sql and javax.sql d) java.rdb and javax.rdb

Q.4 JDBC technology-based drivers generally fit into how many categories ?

a) 4 b) 3

c) 2 d) 5

Q.14 Which of the following JDBC drivers is known as a partially java driver ?

- a JDBC-ODBC bridge driver b Native-API driver
 c Network protocol driver d Thin driver

Q.15 Which class has strong support of the JDBC architecture ?

- a The JDBC driver manager b The JDBC driver test suite
 c The JDBC-ODBC bridge d All of these

Q.16 In order to transfer data between a database and an application written in the Java programming language, the JDBC API provides which of these methods ?

- a Methods on the ResultSet class for retrieving SQL SELECT results as Java types.
 b Methods on the PreparedStatement class for sending Java types as SQL statement parameters.
 c Methods on the CallableStatement class for retrieving SQL OUT parameters as Java types.
 d All of these.

Q.17 The JDBC API has always supported persistent storage of objects defined in the Java programming language through the methods getObject and setObject.

- a True b False

Q.18 What is, in terms of JDBC, a DataSource ?

- a A DataSource is the basic service for managing a set of JDBC drivers.
 b A DataSource is the Java representation of a physical data source.
 c A DataSource is a registry point for JNDI-services.
 d A DataSource is a factory of connections to a physical data source.

Q.19 Which of the following describes the correct sequence of the steps involved in making a connection with a database.

1. Loading the driver.
2. Process the results.
3. Making the connection with the database.
4. Executing the SQL statements.

- a 1,3,4,2 b 1,2,3,4
 c 2,1,3,4 d 4,1,2,3

Q.20 Which of the following methods are needed for loading a database driver in JDBC ?

- a registerDriver() method b Class.forName()
 c Both a and b d getConnection()

Q.21 Which type of statement can execute parameterized queries ?

- a PreparedStatement b ParameterizedStatement
 c CallableStatement d All of these

Q.22 What is used to execute parameterized query ?

- a Statement interface b PreparedStatement interface
 c ResultSet interface d None of the above

Q.23 Which of the following encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed ?

- a DriverManager b JDBC driver
 c Connection d Statement

Q.24 Which of the following is used to call a stored procedure ?

- a Statement b PreparedStatement
 c CallableStatement d CallableStatement

Q.25 What happens if you call deleteRow() on a ResultSet object ?

- a The row you are positioned on is deleted from the ResultSet, but not from the database.
 b The row you are positioned on is deleted from the ResultSet and from the database.
 c The result depends on whether the property synchronizeWithDataSource is set to true or false.
 d You will get a compile error : The method does not exist because you can not delete rows from a ResultSet.

Q.26 The JDBC-ODBC bridge supports multiple concurrent open statements per connection ?

- a True b False

Q.27 All raw data types (for instance-data for images) should be read and uploaded to the database as an array of_____.

- a byte b int
 c boolean d char

Q.28 Are prepared statements actually compiled ?

- a Yes, they compiled
- b No, they are bound by the JDBC driver

Q.29 When the message “No Suitable Driver” occurs ?

- a When the driver is not registered by Class.forName() method.
- b When the user name, password and the database does not match.
- c When the JDBC database URL passed is not constructed properly.
- d When the type 4 driver is used.

Q.30 Database system compiles query when it is ___.

- a executed
- b initialized
- c prepared
- d invoked

Q.31 _____ is an open source DBMS product that runs in window as well as Linux.

- a JSP/SQL
- b MySQL
- c Microsoft access
- d SQL server

Q.32 To execute a statement, we invoke method ____.

- a executeUpdate method
- b executeRel method
- c executeStmt method
- d executeConn method

Q.33 Method on resultset that tests whether or not there remains at least one unfetched tuple in result set, is said to be _____.

- a fetch method
- b current method
- c next method
- d access method

Q.34 The ResultSet.next method is used to move to the next row of the ResultSet, making it the current row.

- a True
- b False

Q.35 ResultSet object can be moved forward only and it is updatable.

- a True
- b False

Q.36 Which JDBC drivers will run your program ?

- a The JDBC-ODBC bridge.
- b The JDBC driver manager.
- c The JDBC driver test suite.
- d None of the above.

Q.37 JDBC is a Java API that is used to connect and execute query to the database.

- a True
- b False

Answer Keys for Multiple Choice Questions :

Q.1	a	Q.2	b	Q.3	c	Q.4	a	Q.5	a	Q.6	d	Q.7	d
Q.8	b	Q.9	d	Q.10	b	Q.11	a	Q.12	c	Q.13	d	Q.14	b
Q.15	a	Q.16	d	Q.17	a	Q.18	d	Q.19	a	Q.20	c	Q.21	a
Q.22	b	Q.23	d	Q.24	d	Q.25	b	Q.26	a	Q.27	a	Q.28	a
Q.29	c	Q.30	c	Q.31	b	Q.32	a	Q.33	c	Q.34	a	Q.35	b
Q.36	c	Q.37	a										



UNIT V

5

Remote Method Invocation (RMI)

Syllabus

Remote Method Invocation : Architecture, RMI registry, the RMI Programming Model; Interfaces and Implementations; Writing distributed application with RMI, Naming services, Naming and Directory Services, Setting up Remote Method Invocation - RMI with Applets, Remote Object Activation; The Roles of Client and Server, Simple Client/Server Application using RMI.

Contents

- 5.1 *Remote Method Invocation*
- 5.2 *Architecture*
- 5.3 *RMI Registry*
- 5.4 *The RMI Programming Model*
- 5.5 *Interfaces and Implementations*
- 5.6 *Writing Distributed Application with RMI*
- 5.7 *Naming Services*
- 5.8 *Naming and Directory Services*
- 5.9 *RMI with Applets*
- 5.10 *Remote Object Activation*
- 5.11 *The Roles of Client and Server*
- 5.12 *Simple Client/Server Application using RMI*
- 5.13 *Multiple Choice Questions*

5.1 Remote Method Invocation

- RMI stands for Remote Method Invocation. It is a mechanism in Java which allows an object running on one system to access an object running on another system.
- This technology consists of a server and a client.
- It is used to build distributed applications.
- For making the remote communication between two java programs the mechanism of RMI is used.
- While implementing the RMI applications the **java.rmi** package is used.
- There are few commonly used terminologies in RMI and those are -
 - **Remote Object** : In Java based distributed computing environment, remote object is the one whose methods can be invoked from another Java Virtual Machine. This JVM can be on different host. The remote object is described by the remote interfaces. The Remote Method Invocation (RMI) is a mechanism used to invoke the remote object via the method defined in remote interface.
 - **Server** : The remote server means the single remote object having the methods that can be remotely invoked.
 - **Client** : The remote client means the remote object that invokes the remote methods on a remote object.
 - **Remote Interface** : The remote object gets accessed via its remote interface.

5.2 Architecture

- A RMI server program creates some remote objects, makes references to these objects. These references are accessible from remote machine. The server then waits for clients to invoke methods with the help of these objects.
- The client program executes and invokes the method on the server. In any RMI application server and the client communicate with the help of stub and skeleton. **Stub** is a client side entity used to invoke the remote object and skeleton is a server side program which dispatches call to the method on the server.
- The **RMI Registry** is used to obtain the reference to a remote object. The task of server is to call the registry and associate a name with a remote object. Using this name from server's registry the client can find (looks up) the remote object and then invokes a required method.

- Fig. 5.2.1 illustrates the RMI flow.

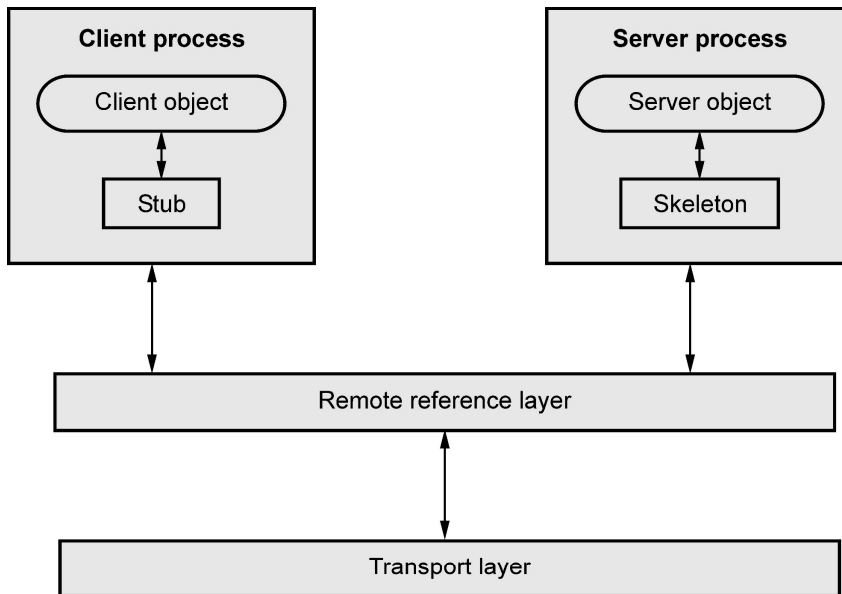


Fig. 5.2.1 Architecture of RMI

Step 1 : Server creates a remote object and registers it in the registry.

Step 2 : Client makes a request for an object from the registry. The registry then returns the remote reference to the client.

Step 3 : Client invokes the method to the stub, the stub in turn talks to the skeleton and skeleton in turn invokes method from the server.

- The RMI implementation is based on three layers and those are :
 - o The Stub / Skeleton layer
 - o Remote Reference layer
 - o Transport layer

1. The Stub/Skeleton layer

- This layer is responsible to route the method calls made by the client to the interface reference and redirect these calls to the remote object.

2. Remote reference layer

- The remote reference layer defines and support the invocation semantics of RMI.
- This layer provide the object that represents the handle to the remote object using the JRMP (Java Remote Method Protocol) protocol.

- The object **RemoteStub** is used to carry out remote method invocation.
- The remote reference layer helps to add another kind of semantics to RMI. That is with the help of this layer, single proxy can send method request to multiple implantations simultaneously.

3. Transport Layer

- The transport layer is a binary data protocol that sends the remote object requests over the wire.
- The transport layer makes use of TCP/IP for the communication between two JVMs. This is connection oriented communication.
- For the communication, the JRMP i.e. Java Remote Method Protocol is on the top of TCP/IP. The JRMP is basically a wire - level protocol.
- Generally the transport layer prefers to use multiple TCP/IP connections between the client and server. But in RMI mechanism, the transport layer multiplexes the multiple virtual connections within the single TCP/IP connection.

5.3 RMI Registry

- RMI registry is a simplified name service where all the server objects are placed.
- When a server creates an object, it registers this object with RMI registry. This registered object has unique name which is called as **bind name**.
- When client wants to refer this object which is present on the server, it actually needs the reference of that object. At that time, the client fetches that object from the registry by using its bind name.
- RMI registry acts a broker between RMI servers and the clients. Essentially the RMI registry is a place for the server to register services it offers and a place for clients to query for those services.

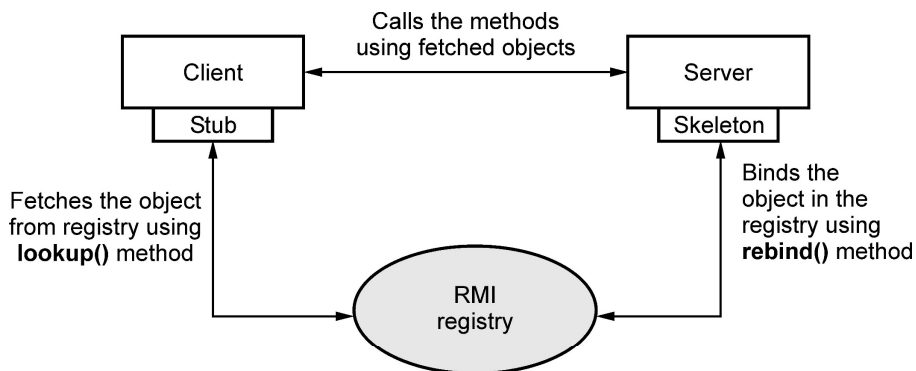


Fig. 5.3.1 RMI registry

5.4 The RMI Programming Model

- For having Remote Method Invocation following three programs are created -
 1. Interface program in which the remote method is declared.
 2. Server program in which remote method is defined.
 3. Client program from which the remote method is called.
- For running the following program, you open two command-prompt windows one for running server and other for running client.

5.5 Interfaces and Implementations

- In RMI, a remote interface is an interface that declares a set of methods that may be invoked from a remote java virtual machine. A remote interface must satisfy the following requirements :
 1. A remote interface must at least extend, either directly or indirectly, the interface **java.rmi.Remote**.
 2. **Remote method** declaration in a remote interface.
 3. A remote method declaration must include the exception **java.rmi.RemoteException** in its throws clause, in addition to any application - specific exceptions.
 - In a remote method declaration, a remote object declared as a parameter or return value (either declared directly in the parameter list or embedded within a non-remote object in a parameter) must be declared as the remote interface, not the implementation class of that interface.
 - The interface **java.rmi.Remote** is a marker interface that defines no methods :
- public interface Remote { }
- In the RMI server-client communication, the interface program is written. Interface program is a program in which remote method is declared. For example - following is a Java program that implements the interface.

```
import java.rmi.*;
public interface FirstRMI extends Remote
{
    void MyHello() throws RemoteException;
}
```

- Note that the method named **MyHello()** will be defined in RMI server and RMI client will invoke this method.

5.6 Writing Distributed Application with RMI

- It is not possible to directly refer to a remote object. It requires an active distributed framework.
- For the Java distributed object model, a remote object is an object whose methods can be invoked from another system which might be running on a different host.
- The remote object is described by remote interface that declares available methods.

5.7 Naming Services

- When multiple servers are involved in your application, the naming service allows you to specify logical server names rather than server addresses. For example, instead of connecting to your database server at host using some address and port number, you can specify the name of that server, such as MYHost/MyCompany/MyServer.
- Similarly components on that server can be identified by specifying an initial server name context plus the package and component name.

C:\My package \ Payroll \ index.html

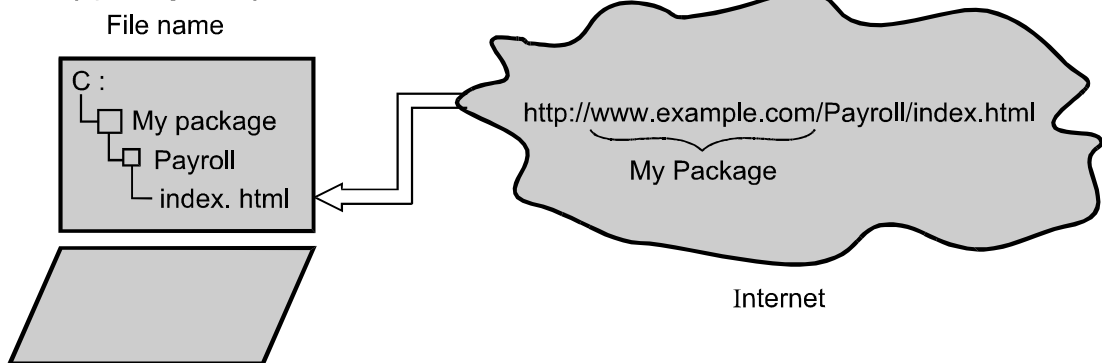


Fig. 5.7.1 Naming system

- **Name Server** consists of a program that implements a **naming service** protocol. This protocol maps a **human-recognizable** name/address to a system-internal, numeric, identification. Thus naming services are used to specify logical server name rather than server addresses.
- The most prominently used name servers are Domain Name Servers / Domain Name Service (DNS).

5.8 Naming and Directory Services

- **Directory service** is a kind of naming service which can perform searching of the objects based on search filter criteria.
- Directory services also allow the modification of object attributes in the directory service tree.
- The **Object Management Group (OMG)** provides a standard classification of naming services.
- Directory services have a typical hierarchical structure each contained **directory object** corresponding to some **specific resources** such as a file, a printer, or a user's profile. This directory object itself is actually manifested in a directory service as a collection of attributes describing the kind of resource to which it is associated.
- A directory object in a directory service tree can be retrieved by its name.
- The directory objects can be **added, deleted, searched** or **modified** by using the directory services. These operations are similar to database operations.
- Example of Directory service provider is Lightweight Directory Access Protocol (LDAP) service.

5.9 RMI with Applets

- The applet is a client side programming. Hence RMI with applet can be implemented for the client program.
- In this applet, the applet class extends **Applet**. Inside this class - there are two methods **init** and **paint**.
- In the **init** method, we try to connect to RMI server program and access the function written in the RMI server program. In the following example we try to access the function **MyHello()** which is present in RMI Server program.
- Following example shows both the server and client programs.

Interface (FirstRMI.java)

```
import java.rmi.*;
public interface FirstRMI extends Remote
{
    void MyHello() throws RemoteException;
}
```

Server Program(FirstRMI_Server.java)

```
*
*****
RMI Server program [FirstRMI_Server.java]
*****
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.net.*;

public class FirstRMI_Server extends
java.rmi.server.UnicastRemoteObject implements FirstRMI
{
String address;
Registry registry;
int port=1234; //this is a server port
//use the same port number for client
//so that the communication can be possible

public void MyHello() //Definition of remote method
{
System.out.println("Hello Friends!!!");

}

public FirstRMI_Server() throws RemoteException
{
try
{
address = (InetAddress.getLocalHost()).toString();
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
System.out.println("Server started at: " + address + ", "+port);
System.out.println("[Now run client with same IP address]");
//creating the registry
registry = LocateRegistry.createRegistry(port);
registry.rebind("rmiServer", this);
}
static public void main(String args[] )
{
```

Remote method **MyHello**
is defined here.


```
try
{
    FirstRMI_Server server = new FirstRMI_Server();
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
}
```

Client Program (FirstRMI_Client.java)

```
/*
*****
RMI Client program [FirstRMI_Client.java]
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;
import java.applet.*;
import java.awt.*;
import java.net.URL;
/*
<applet code="FirstRMI_Client" width=300 height=100>
</applet>
*/
public class FirstRMI_Client extends Applet
{
    FirstRMI rmiServer;
    Registry registry;
    int port=1234;//this should be same as Server
    String msg="";
    public void init(){
        try
        {
            registry=LocateRegistry.getRegistry(getCodeBase().getHost(),1234);
            rmiServer=(FirstRMI)(registry.lookup("rmiServer"));
            // call the remote method with the message
            msg=rmiServer.MyHello();
        }
        catch(RemoteException e){
            System.out.println(e.getMessage());
        }
    }
}
```

```
    catch(NotBoundException e){
        System.out.println(e.getMessage());
    }
}
public void paint(Graphics g) {
    g.drawString(msg, 25, 50);
}
}
```

5.10 Remote Object Activation

- The important task in RMI communication is to locate the remote objects.
- The remote objects can be located with the help of **naming and directory services**. These services are run on the host machines whose name and port numbers are known to the clients.
- RMI makes use of **Java Naming and Directory Interface (JNDI)** for locating the remote clients.
- The **RMI Registry** exists for storing the names of the remote objects and **RMI URL** for identifying the locations of objects over the network. Clients can acquire a reference and get connected to a remote object. Using **java.rmi.Naming class**, client connects to RMI Registries for the purposes of finding remote objects.
- The most important method that the class **java.rmi.Naming** which exposes is the **lookup()** method. This method takes an RMI URL, connects to an RMI Registry on a target java machine and returns a **java.lang.Object** representing the remote object. The returned object is actually the remote stub. This stub can then be typecast into our remote object's interface type. Refer following Fig. 5.10.1.

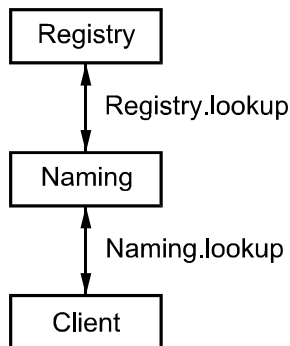


Fig. 5.10.1 Locating remote objects

- The URL is of the form

```
rmi://<host>[:name_service_port]/<servicename>
```

Where :

- o <host> is the name of the host. It can be DNS name on internet.
- o <name_service_port> specifies the name of the port on which the service is running.
- o <servicename> is the name of the service to which the remote object is associated in the registry.
- Following are the steps followed in order to locate the remote object -
 - o Create a local object.
 - o Export that object to the listening service. The listening services are those services that waits for listening the clients (these clients are those who demand for some service).
 - o Registers the object in the RMI registry.
- The code for above these steps can be as follows -

```
MyServer obj=new MyServer();  
Naming.rebind("MyServer",obj);
```

- Just similar to **Naming** class **java.rmi.registry.LocateResitry** using which the reference for the registry can be obtained and then the registry can be started. Following code is shows this -

```
registry = LocateRegistry.createRegistry(port);  
registry.rebind("rmiServer", this);
```

- For granting the permission to the code to execute the **security policy** file is created by Java.

5.11 The Roles of Client and Server

- The stubs are the client side components and the skeletons are the server side components.
- The stub/skeleton layer helps in transport of the data to remote reference layer by marshalling and unmarshalling. The **marshalling** is a process of converting data or object to byte stream and **unmarshalling** is the process of converting byte stream to data or object. This conversion is done with the help of **object serialization**.
- The stub and skeleton layer is based on **proxy design pattern**. In the proxy design pattern, the stub class acts as a proxy for the remote service implementation. The skeleton class is a **helper class** and it allows the object to communicate with the stub.

- The skeleton communicates with the stub. It reads the parameters for method call made by the client stub, then skeleton makes to the remote service implementation object. Then the return value obtained from the remote service implementation is accepted by the skeleton. The skeleton then writes back this value to the stub. The proxy design pattern method calls occur through the proxy.
- **Stub** : The stub is a client side object that acts as a proxy for the remote object. Following are the steps performed by the stub.
 - o Initiates the connection with the remote virtual machine. This virtual machine contains the desired remote object.
 - o Marshals the parameters to the remote virtual machine. That means writes and transmit data in the form of bytes to the remote VM.
 - o Unmarshals the return value or exception returned. That means the return value returning from the remote object is taken back from byte stream to the data stream format.
 - o This data value is then returned to the caller.
- The stub hides all these serialization of method and network communication. The above scenario is presented to the caller as a simple invocation mechanism.
- **Skeleton** : The skeleton object communicates with the **remote object**. Following are the steps performed by the skeleton.
 - o The stub marshals the parameters of remote method and these method parameters are unmarshaled (read) by the skeleton.
 - o The skeleton then invokes the method on the actual remote object of the implementation.
 - o Marshals the result obtained from the remote method implementation. It can be return value or exception. That means the return value is converted to the byte stream and transmitted over the RMI link to the client stub.

5.12 Simple Client / Server Application using RMI

- For having remote method invocation following three programs are created -
 1. **Interface program** in which the remote method is declared.
 2. **Server program** in which remote method is defined.
 3. **Client program** from which the remote method is called.
- For running the following program, you open two command-prompt windows one for running server and other for running client.

Example 5.12.1 Write a simple RMI application in which the client invokes the method of the server.

Solution : Step 1 : We will write a simple interface in which the signature of method is given. Following code can be written in a notepad and saved as **FirstRMI.java**.

```
import java.rmi.*;
public interface FirstRMI extends Remote
{
    void MyHello() throws RemoteException;
}
```

Note that the method named **MyHello()** will be defined in RMI server and RMI client will invoke this method.

Step 2 : Following is a simple code that can be written in notepad for RMI Server. The filename is **FirstRMI_Server.java**.

```
/*
*****
RMI Server program [FirstRMI_Server.java]
*****
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.net.*;

public class FirstRMI_Server extends
java.rmi.server.UnicastRemoteObject implements FirstRMI
{
    String address;
    Registry registry;
    int port= 1234; //this is a server port
    //use the same port number for client
    //so that the communication can be possible

    public void MyHello() //Definition of remote method
    {
        System.out.println("Hello Friends!!!");
    }

    public FirstRMI_Server() throws RemoteException
    {
        try
        {
```

Remote method
MyHello is defined
here.

```

    address = (InetAddress.getLocalHost()).toString();
}
catch(Exception e)
{
    System.out.println(e.getMessage());
}
System.out.println("Server started at: " + address + ", "+port);
System.out.println("[Now run client with same IP address]");
//creating the registry
registry = LocateRegistry.createRegistry(port);
registry.rebind("rmiServer", this);
}
static public void main(String args[] )
{
    try
    {
        FirstRMI_Server server = new FirstRMI_Server();
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
}
}

```

Step 3 : The RMI client code is basically to invoke the server method. Following is a client code that invokes the method stored in RMI server. This code can also be written in Notepad and the file name for this code is **FirstRMI_Client.java**.

```

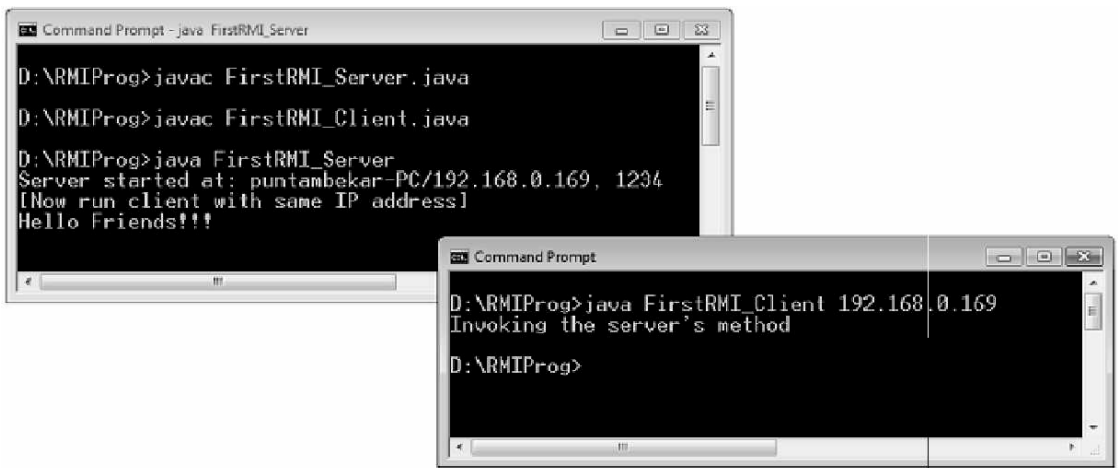
/*
*****
RMI Client program [FirstRMI_Client.java]
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;

public class FirstRMI_Client
{
    static public void main(String args[])
    {
        FirstRMI rmiServer;
        Registry registry;
        String IP_Add=args[0];
    }
}

```

```
int port=1234;//this should be same as Server
System.out.println("Invoking the server's method");
try
{
    registry=LocateRegistry.getRegistry(IP_Add,port);
    rmiServer=(FirstRMI)(registry.lookup("rmiServer"));
    // call the remote method with the message
    rmiServer.MyHello();//remote method is called
}
catch(RemoteException e)
{
    System.out.println(e.getMessage());
}
catch(NotBoundException e)
{
    System.out.println(e.getMessage());
}
}
```

Step 4 : Now open the two command prompts. On and get the output as follows -



The image shows two overlapping Windows Command Prompt windows. The top window, titled 'Command Prompt - java FirstRMI_Server', shows the following commands and output:

```
D:\RMIProg>javac FirstRMI_Server.java
D:\RMIProg>javac FirstRMI_Client.java
D:\RMIProg>java FirstRMI_Server
Server started at: puntambekar-PC/192.168.0.169, 1234
[Now run client with same IP address]
Hello Friends!!!
```

The bottom window, titled 'Command Prompt', shows the following commands and output:

```
D:\RMIProg>java FirstRMI_Client 192.168.0.169
Invoking the server's method
D:\RMIProg>
```

Program explanation :

- In this application we have first created an interface file. In this interface the signature of the remote method is stored. Note that every RMI interface file in Java imports **java.rmi.***;
- The interface class is inherited from the **Remote** interface of java.rmi package. Then the method is declared by throwing the **RemoteException**.

- The server class extends the **UnicastRemoteObject** class. This is a class in the **java.rmi.server** package that extends the **java.rmi.server.RemoteServer**, which itself extends **java.rmi.server.RemoteObject**. This is the base class for all RMI objects.
- In the RMI server program the registry is created using the code. The **createRegistry** method is invoked using the object **LocateRegistry**.

```
registry = LocateRegistry.createRegistry(port);
```

- Then registry is bind to the RMI server.
- The RMI Client program obtains the reference to the registry by following command -

```
registry=LocateRegistry.getRegistry(IP_Add,port);
```

- Then the bound RMI server is searched for using the lookup method.

```
rmiServer=(FirstRMI)(registry.lookup("rmiServer"));
```

- Then the actual method stored in the server is invoked.

```
rmiServer.MyHello();//remote method is called
```

- Thus client invokes the server's method using RMI.

Example 5.12.2 Write a RMI application in which the client can send a message to the server.

Solution : Java Program [My_Interface.java]

```

/*
*****
        RMI Interface program
*****
*/
import java.rmi.*;
public interface My_Interface extends Remote
{
    void My_Message(String str) throws RemoteException;
}
//this method is defined in server and called from the client
}
/*
*****
        RMI Server program [RMI_Server.java]
*****
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

```

The interface must extend **Remote**

And must throw an exception


```
import java.net.*;
public class RMI_Server extends
java.rmi.server.UnicastRemoteObject implements My_Interface
{
String address;
Registry registry;
int port= 1234; //this is a server port
//use the same port number for client
//so that the communication can be possible
public void My_Message(String str) //Definition of remote method
{
//This function converts the client's message to upper case
String Capital_str;
Capital_str=str.toUpperCase(); //received message is capitalized
System.out.println(Capital_str);//an displayed on server console
}
public RMI_Server() throws RemoteException
{
try
{
address = (InetAddress.getLocalHost()).toString();
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
System.out.println("Server started at: " + address + ", "+port);
System.out.println("[Now run client with same IP address]");
//creating the registry
registry = LocateRegistry.createRegistry(port);
registry.rebind("rmiServer", this);
}
static public void main(String args[ ])
{
try
{
RMI_Server server = new RMI_Server();
}
catch (Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

Created object for registry

Getting and IP address of Local host

```

/*
*****
        RMI Client program [RMI_Client.java]
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;

public class RMI_Client
{
    static public void main(String args[])
    {
        My_Interface rmiServer;
        Registry registry;
        BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
        //Created for reading the input string from the console
        String IP_Add=args[0];
        int port=1234;//this should be same as Server
        String text="";
        try
        {
            System.out.println("enter some message...");
            text=input.readLine();
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
        System.out.println("sending " + text + " to " +IP_Add + ":" + port);
        try
        {
            registry=LocateRegistry.getRegistry(IP_Add,port);
            rmiServer=(My_Interface)(registry.lookup("rmiServer"));
            // call the remote method with the message
            rmiServer.My_Message(text);//remote method is called
        }
        catch(RemoteException e)
        {
            System.out.println(e.getMessage());
        }
        catch(NotBoundException e)

        {
            System.out.println(e.getMessage());
        }
    }
}

```

Getting interface object

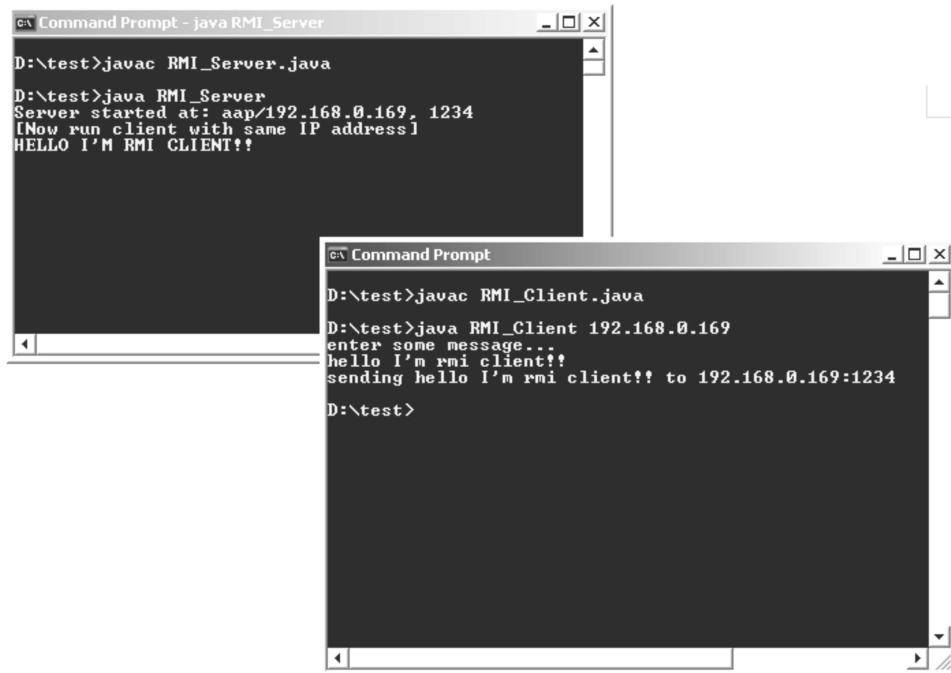
Object for registry

Asking user to enter some message on console

Using an interface object, by having registry lookup the message is sent to the server.

```
}  
}  
}
```

Output



```
Command Prompt - java RMI_Server  
D:\test>javac RMI_Server.java  
D:\test>java RMI_Server  
Server started at: aap/192.168.0.169, 1234  
[Now run client with same IP address]  
HELLO I'M RMI CLIENT!?
```

```
Command Prompt  
D:\test>javac RMI_Client.java  
D:\test>java RMI_Client 192.168.0.169  
enter some message...  
hello I'm rmi client!?  
sending hello I'm rmi client!! to 192.168.0.169:1234  
D:\test>
```

Example 5.12.3 *What is RMI ? Create a remote phone book server that maintains a file of names and phone numbers and client allows the user to scroll through the file using RMI concept.*

Solution : RMI : The Remote Method Invocation is a technique in which the method present on one machine can be invoked by another machine using some interface. Java has a strong support for RMI feature.

In Java, the RMI Server stores the definition of the method and a client can call this method using some interface. The interface is again a Java program in which the declaration of this method is given.

Following is a simple **phonebook** application in which, we have created an interface file named **My_Interface.java**. In this file we have declared a function **GetPhNumber**. While declaring this function it is a must to throw **RemoteException**.

```
/*
*****
                RMI Interface program
*****
*/
import java.rmi.*;
public interface My_Interface extends Remote

{
    public String GetPhNumber(String str) throws RemoteException;
    //this method is defined in server and called from the client
}

```

Explanation :

- We have to compile this file in order to generate **My_Interface.class** file now the RMI server program can be written in the following file. It is named as **RMIServer.java**.
- In this program the definition of the function **GetPhNumber** is written. We maintained two arrays for storing phone numbers and corresponding names in **Phone_number** and **name** array respectively.
- Client sends the name which is compared by the server in the **name** array. If the entry for that name is found then corresponding phone number is retrieved from the **Phone_number** array and is returned to the client.

```
/*
*****
                RMI Server program
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.net.*;

public class RMIServer extends
java.rmi.server.UnicastRemoteObject implements My_Interface
{

```

```
String address;
int flag=0;
int i;
String[] Phone_number=new
String[]{"9812345671","9812345672","9812345673","98123456714","98123456715"};
String[] name=new String[]{"Archana","Supriya","Shilpa","Sagar","Shivraj"};
Registry registry;

int port=1234; //this is a server port
public String GetPhNumber(String str) //Definition of remote method
{
try

for(i=0;i<5;i++)
{
if(str.equals(name[i]))
{
flag=1;
System.out.println("Record is present");//an displayed on server console
break;
}
}
}
catch(ArrayIndexOutOfBoundsException e){System.out.println(e.getMessage()); }
if(flag==0)
{
// displayed on server console
System.out.println("The Record is not present on the server");
String msg="Record is not present in the phone book";
i=0;
return msg;
}
else
return Phone_number[i];

} //end of function GetPhNumber
```

```
public RMIServer() throws RemoteException
{
    try
    {
        address = (InetAddress.getLocalHost()).toString();
    }
    catch(Exception e)
    {
        System.out.println(e.getMessage());
    }
    System.out.println("Server started at: " + address + ", "+port);
    System.out.println("[Now run client with same IP address]");

    //creating the registry
    registry = LocateRegistry.createRegistry(port);
    registry.rebind("rmiServer", this);
}
static public void main(String args[ ])
{
    try
    {
        RMIServer server = new RMIServer();
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
} //end of main
} //end of class
```

Explanation :

The client asks to enter the name of the person whose phone number needs to be known. When user submits the name of the person, server returns it the phone number of desired person. If the corresponding entry is not present then it simply returns “record is not present” message. The client program is written in **RMIClient.java** file and compiled to generate the **RMIClient.class** file.

```
/*
*****
          RMI Client program
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;
public class RMIClient

{
    static public void main(String args[])
    {
        My_Interface rmiServer;
        Registry registry;
        BufferedReader input=new BufferedReader(new InputStreamReader(System.in));

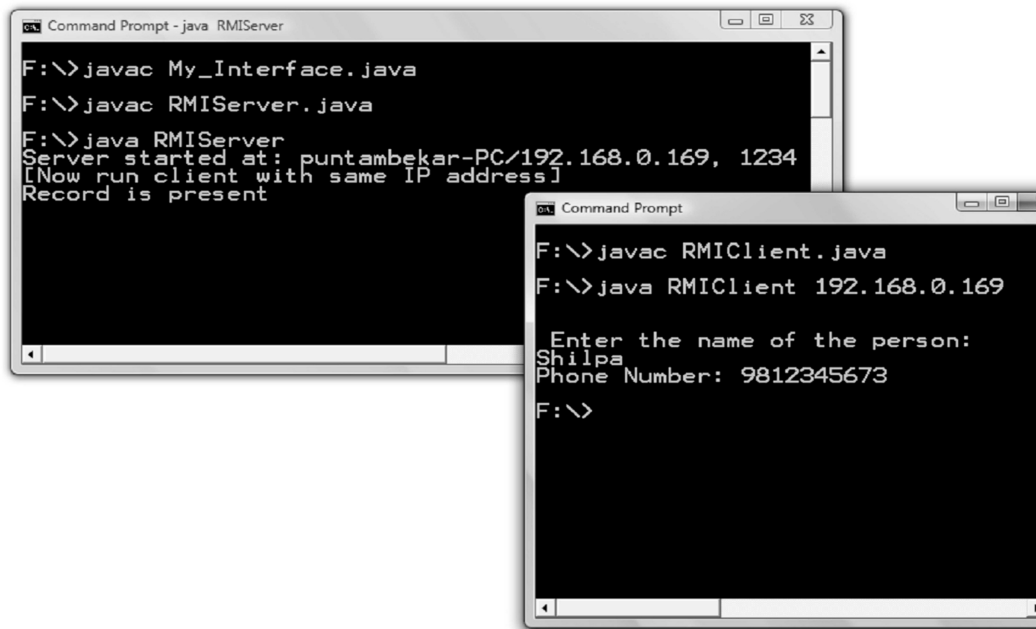
        //Created for reading the input string from the console
        String IP_Add=args[0];
        int port=1234;//this should be same as Server
        String text=" ";
        try
        {
            System.out.println("\n\n Enter the name of the person: ");
            text=input.readLine();
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
        try
        {
            registry=LocateRegistry.getRegistry(IP_Add,port);
            rmiServer=(My_Interface)(registry.lookup("rmiServer"));

            // call the remote method with the message
            System.out.println("Phone Number: "+rmiServer.GetPhNumber(text));//remote method is called
        }
        catch(RemoteException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

```
catch(NotBoundException e)
{
    System.out.println(e.getMessage());
}
}
```

[Note : For the sake of understanding the output of the above application is as given below. First execute the server and then start the client execution.]

Output



The image shows two overlapping Command Prompt windows. The background window, titled 'Command Prompt - java RMIServer', shows the following commands and output:

```
F:\>javac My_Interface.java
F:\>javac RMIServer.java
F:\>java RMIServer
Server started at: puntambekar-PC/192.168.0.169, 1234
[Now run client with same IP address]
Record is present
```

The foreground window, titled 'Command Prompt', shows the following commands and output:

```
F:\>javac RMIClient.java
F:\>java RMIClient 192.168.0.169

Enter the name of the person:
Shilpa
Phone Number: 9812345673
F:\>
```

Example 5.12.4 Explain the use of RMI in examination control system in which the server has all the student information and the student objects can be accessed from any client.

Solution : This is a RMI application in which there are two entities RMI server and RMI client. There is one more entity which is used by both the client and server and that is **interface**.

We will write declarations of two methods in the **interface** file. One method is returning the roll number of the student when client submits the student name. The other method will return the marks of the student whose roll number is submitted by the client. All the records of the students such as roll number, name and marks are stored on the server.

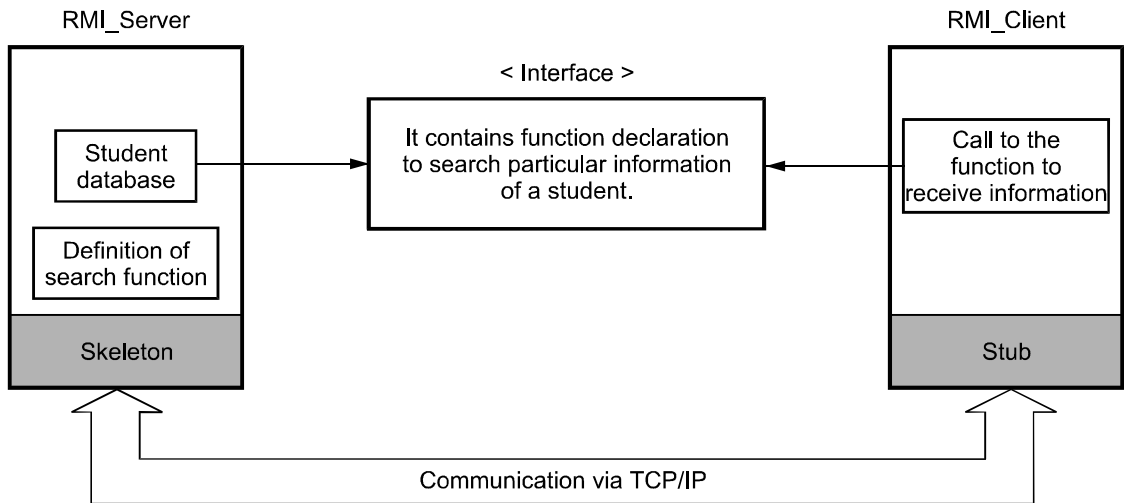


Fig. 5.12.1 RMI application

The code for this will be like this -

Step 1 : Java Program[RMI_Server.java]

```

/*
*****
        RMI Server program
*****
*/
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.net.*;

public class RMI_Server extends
java.rmi.server.UnicastRemoteObject implements My_Interface
{
    String address;
    int flag=0;
    int i;
    int[] roll_number=new int[]{1234,1235,1236,1237,1238};
    String[] name=new String[]{"Archana","Supriya","Shilpa","Sagar","Shivraj"};
    double[] marks=new double[]{73.33,66.44,55.50,44.2,67.99};
    Registry registry;
    int port=1234; //this is a server port
    public int My_Message(String str) //Definition of remote method
    {
        for(i=0;i<5;i++)

```

```
{
    if(str.equals(name[i]))
    {
        flag=1;
        System.out.println("Record is present");//an displayed on server console
        break;
    }
}
if(flag==0)
System.out.println("The Record is not present");//an displayed on server console
return roll_number[i];
} //end of MyMessage
```

//This is another function which returns marks of the student

```
public double My_Message1(int Roll) //Definition of remote method
{
    double stud_marks;
    int i,flag=0;
    for(i=0;i<5;i++)
    {
        if(Roll==roll_number[i])
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
    {
        System.out.println("The record is not present");
        return -99; //means record is not present
    }

    else
    {
        stud_marks=marks[i];
        return stud_marks;
    }
} //end of MyMessage1
```

```
public RMI_Server() throws RemoteException
{
    try
    {
        address = (InetAddress.getLocalHost()).toString();
```

```
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
System.out.println("Server started at: " + address + ", "+port);
System.out.println("[Now run client with same IP address]");
//creating the registry
registry = LocateRegistry.createRegistry(port);
registry.rebind("rmiServer", this);
}

static public void main(String args[] )
{

try
{
RMI_Server server = new RMI_Server();
}
catch (Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

Step 2 :

```
/*
*****
RMI Client program
*****
*/

import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;

public class RMI_Client
{
static public void main(String args[])
{
My_Interface rmiServer;
Registry registry;
```

```
BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
//Created for reading the input string from the console
String IP_Add=args[0];
int port=1234;//this should be same as Server
String text=" ";
String roll_text=" ";
try
{
    System.out.println("\n\n Enter the name of the student whose roll number is need to be known ...");
    text=input.readLine();
}
catch(IOException e)
{
    System.out.println(e.getMessage());
}
try
{
    registry=LocateRegistry.getRegistry(IP_Add,port);
    rmiServer=(My_Interface)(registry.lookup("rmiServer"));
    // call the remote method with the message
    System.out.println("The Roll Number of the student is "+rmiServer.My_Message(text));//remote
method is called
    try
    {
        System.out.println("\n\n Enter the Roll number of the student whose marks need to be known ...");
        roll_text=input.readLine();
    }
    catch(IOException e)
    {
        System.out.println(e.getMessage());
    }
    System.out.println("The marks the student are
"+rmiServer.My_Message1(Integer.parseInt(roll_text));//remote method is called

}
catch(RemoteException e)
{
    System.out.println(e.getMessage());
}
catch(NotBoundException e)
{
    System.out.println(e.getMessage());
}
}
```

Step 3 :

```

/*
*****
                RMI Interface program
*****
*/
import java.rmi.*;
public interface My_Interface extends Remote
{
    public int My_Message(String str) throws RemoteException;
    public double My_Message1(int Roll) throws RemoteException;
    //this method is defined in server and called from the client
}

```

Step 4 : Open up two separate command prompt and compile server program first and then the client program. Following sort of output can be obtained -

```

Command Prompt - java RMI_Server
F:\> javac RMI_Server.java
F:\> java RMI_Server
Server started at: puntambekar-PC/192.168.0.169, 1234
[Now run client with same IP address]
Record is present

Command Prompt
F:\> javac RMI_Client.java
F:\> java RMI_Client 192.168.0.169
Enter the name of the student whose roll number is need to be known ...
Shilpa
The Roll Number of the student is 1236
Enter the Roll number of the student whose marks need to be known ...
1236
The marks the student are 55.5
F:\>

```

Review Question

1. What are the different ways of parameter passing in RMI ?

5.13 Multiple Choice Questions

1. RMI stands for ____.

- a) Routine modification Interface
- b) Remote Method Interval
- c) Remote Method Interface
- d) Remote Method Invocation

UNIT VI

6

Networking

Syllabus

The java.net package, Connection oriented transmission - Stream Socket Class, creating a Socket to a remote host on a port (creating TCP client and server), Simple Socket Program Example. InetAddress, Factory Methods, Instance Methods, Inet4Address and Inet6Address, TCP/IP Client Sockets. URL, URLConnection, HttpURLConnection, The URI Class, Cookies, TCP/IP Server Sockets, Datagrams, DatagramSocket, DatagramPacket, A Datagram Example. Connecting to a Server, Implementing Servers, Sending EMail, Servlet overview - the Java web server - The Life Cycle of a Servlet, your first servlet.

Contents

- 6.1 *The java.net Package*
- 6.2 *Socket Class*
- 6.3 *InetAddress*
- 6.4 *URL*
- 6.5 *URLConnection*
- 6.6 *HttpURLConnection*
- 6.7 *The URI Class*
- 6.8 *Cookies*
- 6.9 *TCP,IP and UDP*
- 6.10 *TCP/IP Client Sockets*
- 6.11 *TCP/IP Server Sockets*
- 6.12 *Datagrams*
- 6.13 *Sending Email*
- 6.14 *Servlet Overview*
- 6.15 *Handling HTTP Requests and Response*
- 6.16 *Multiple Choice Questions*

6.1 The java.net Package

6.1.1 The Networking Classes and Interfaces

The **java.net** package is for providing the useful classes and interfaces for networking applications which are used in sockets and URL. These are as given below -

Classes

Name	Description
ContentHandler	This class is a superclass of all the classes that read the data from a class URLConnection. It also builds the appropriate local object based on MIME types.
DatagramSocket	This class represents the Datagram Socket (UDP socket).
DatagramPacket	This class represents the Datagram Packet (UDP packets for containing data).
InetAddress	This class represents the IP address.
InetSocketAddress	The IP socket address is a combination of IP address and port number. To implement such IP socket address this class is used.
ServerSocket	For implementing serverside sockets this class is useful.
Socket	For implementing client side sockets this class is useful.
SocketAddress	This class helps to represent the socket address without specification of protocol.
URL	This class is for creating a reference of uniform resource locator which points to WWW.
URI	This class provides the object of uniform resource identifier.
URLConnection	For establishing a communication between application program and URL this class is used.

Interfaces

Name	Description	Some methods
ContentHandlerFactory	This interface is for defining the factory for content handlers.	<i>createContentHandler(string MIME_type)</i> -Creates ContentHandler.

SocketImplFactory	This interface is for defining the factory for implementing the sockets.	<i>createSocketImpl()</i> - Creates a new instance for implementing socket.
URLConnectionHandlerFactory	For implementing URL stream protocol handlers the factory can be defined by this interface.	<i>createURLConnectionHandler (String p)</i> - For specific protocol 'p' a new URLStreamHandler instance can be created by this method.

Review Question

1. Explain networking classes and interfaces.

6.2 Socket Class

- Socket is the most commonly used term in network programming. Socket provides the way by which the computers can communicate using **connection oriented** or **connection less communication**.
- **Definition of socket** : A socket is basically an **endpoint** of a two-way communication link between two programs running on the network. It is **OS-controlled interface** into which the applications can send or receive messages to and fro from another application.
- The **java.net.Socket** class represents the socket.
- Two key classes are used to create the socket.
 - i) ServerSocket
 - ii) Socket
- A server program creates a specific type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams.
- The **ServerSocket** can be created with the help of port number. For example

```
ServerSocket server_socket=new Socket(1234);
```

- The **Socket** object can be created with the help of hostname and port number. The general syntax of socket instantiation is

```
Socket client= new Socket(serverName, portNumber);
```

For example

```
Socket client_socket=new Socket("myserver",1234);
```

The **ServerSocket** listens the client using the **accept** method. For example

```
Socket Listen_socket=server_socket.accept();
```

There are two types of sockets -

- i) **TCP sockets** : These are denoted by **streams**.
- ii) **UDP sockets** : These are denoted by **datagrams**.

6.2.1 Client Server

- In client server communication, there are 3 components.
 - 1) Client PC or web client
 - 2) An application server or web server
 - 3) A database server.

Working

Step 1 : The client PC or web client submits the request for desired web page to the web server.

Step 2 : The work of server is distributed among application server and database servers. Application server possess the required communication functions.

Step 3 : The data required by this business logic is present on database server. The required data is returned to application servers.

Step 4 : The web server or application server prepares the response page and sends it to the web client.

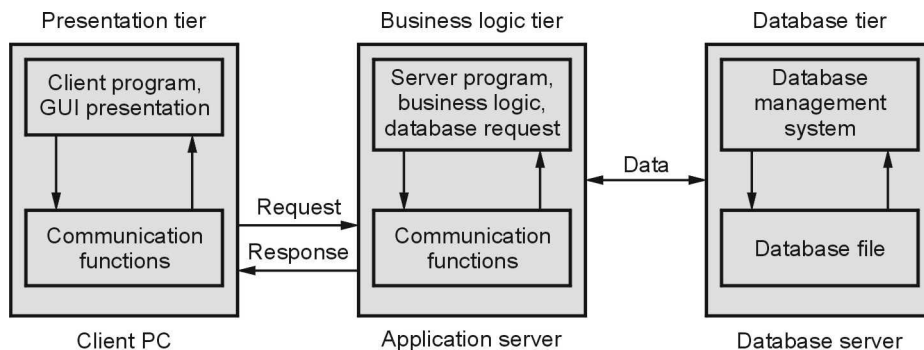


Fig. 6.2.1 Client server communication

6.2.2 Reserved Sockets

- In network programming, the **port** is a medium through which one application establishes connection with other application by binding socket using **port number**.
- With the help of port number and some additional information data is transferred from client to server or from server to client.
- There are some ports which are reserved for specific service. These ports are called as reserved ports.
- Various port numbers specifying their services are given in the following table

Port number	Service
21	FTP
23	Telnet
25	SMTP
80	HTTP
110	POP3

- **User level processes** or services generally use port numbers ≥ 1024 .

Role of port in socket programming

- A server runs on specific computer and has a socket that is bound to specific port. Here the port number must be other than the reserved port numbers. Usually it is ≥ 1024 . For example - If I use port number 1122 for client server communication then it is a valid port number. But if I make use of port number 110 then it is not valid as POP3 service makes use of this number.
- The server waits and listen to the socket for a client to listen.
- The client makes a connection request knowing the hostname and port number on which server is listening.
- The client binds to its local port number that it will use during this connection.

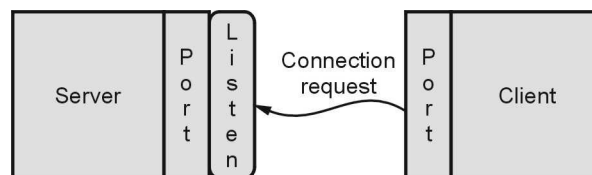


Fig. 6.2.2

6.2.3 Proxy Servers

What is proxy server ? : A proxy server is a server that sits between a client application and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.

What is the purpose of having proxy server ?

- 1) It **improves the performance** by caching the information. To understand the concept of caching consider a scenario -

Suppose there are two users - user X and user Y access the World Wide Web through a proxy server. First user X requests a certain web page, which we'll call page 1. Sometime later, user Y requests the same page. Instead of forwarding the request to the web server where page 1 resides, which can be a time-consuming operation, the proxy server simply returns the page 1 that it already fetched for user X. Since the proxy server is often on the same network as the user, this is a much faster operation.

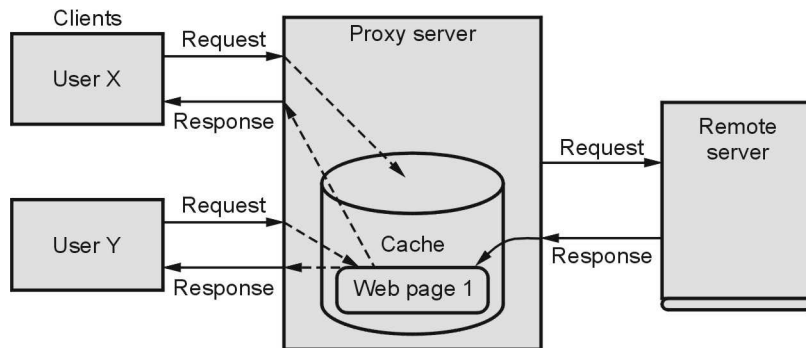
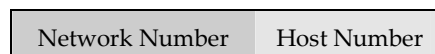


Fig. 6.2.3 Working of proxy server

- 2) It provides security and privacy. Because it prevents clients to access sensitive information directly from the server.

6.2.4 Internet Addressing

- Every computer on internet has address. This is called IP address or Internet Protocol Address.
- This is unique 32 bit logical address divided into two main parts - Network Number and Host number



- There are 5 classes based on two categories viz. A, B, C, D and E.

IP address class	Format	Range	Purpose
Class A	N.H.H.H	1 to 126	Very few large organizations use this class addressing.
Class B	N.N.H.H	127 to 191	Medium size organizations use this addressing.
Class C	N.N.N.H	192 to 223	Relatively small organizations use this class.
Class D	–	224 to 239	This class address is used for multicast groups.
Class E	–	240 to 254	This class addressing is reserved for experimental purpose.

- Here N stands for network number and H stands for host number. For instance in class C first three octets are reserved for network address and last 8-bits denote host address.
- IP address is assigned to the devices participating in computer network.
- The IP protocol makes use of this address for communication between two computers.
- Using IP address particular node can be identified in the network.

Review Questions

1. Give an overview of socket.
2. What is reserved port ?
3. What is proxy server ? Explain working of proxy server.
4. Explain internet addressing scheme.

6.3 InetAddress

- The InetAddress class from **java.net** package represents the IP addresses.
- It works with either **host name or numerical IP address** of corresponding host.
- **InetAddress** class offers many useful methods for handling IP addresses and host names.

6.3.1 Factory Methods

- InetAddress class has no visible constructor. Hence to make use of InetAddress object we need to use the factory methods.
- **What is factory method ?** Factory method is simply a convention whereby static methods in a class return instance of that class.

- There are three important factory **methods** from InetAddress class and those are
 - getLocalHost
 - getByname
 - getAllByName

Following program illustrates the use of **getLocalHost()** method -

Example 6.3.1 Write a Java program to find the IP address of your machine.

Solution :

```
/******  
This program make use of the method getLocalHost()  
method for finding the IP address of local host machine.  
*****/  
import java.net.*;  
class InetAdd1  
{  
    public static void main(String args[]) throws UnknownHostException  
    {  
        InetAddress local_add=InetAddress.getLocalHost();  
        System.out.println("Local Host is: "+local_add);  
    }  
}
```

How to run above program ?

Open command prompt window and type following command

```
D:\test>javac InetAdd1.java  
D:\test>java InetAdd1
```

Output

```
Local Host is: aap/192.168.0.166
```

Program explanation :

- When you run the above program you get the IP address of the local machine on which you are running your program along with the host name. In above case,
 - 1) *aap* is the name my machine on which I am running this program and 192.168.0.166 is my machine's IP address.
 - 2) Note that while running this program your machine should be in network. If it is not in the networking (i.e. standalone machine) then you will get the output in the following manner

```
D:\test>javac InetAdd1.java  
D:\test>java InetAdd1  
Local Host is: aap/127.0.0.1
```

- Here 127.0.0.1 is a reserved IP address corresponding to the host computer which is also known as loop-back address. 127.0.0.1 is used whenever program needs to access a network running on the local computer itself.
- 3) One more important thing about the above Java program and that is definition of main function. It should be

```
public static void main(String args[]) throws UnknownHostException
```

That means, it is a must to throw an exception for unknown host in order to handle unknown host situation. Hence precisely **UnknownHostException** should be thrown.

- 4) Our programs which makes use of the methods in InetAddress class must have following statement at the beginning -

```
Import java.net.*
```

Remember that it is **java.net** package only which deals with InetAddress class functionalities.

Example 6.3.2 Write a Java program to illustrate `getByName()` factory method.

Solution :

```

/*****
This program shows the use of the method getByName
for getting the IP address for the corresponding host name
*****/
import java.net.*;
class InetAdd2
{
public static void main(String args[]) throws UnknownHostException
{
    InetAddress addr =InetAddress.getLocalHost();
    System.out.println(addr);
    Address=InetAddress.getByName("vtubooks.com");
    System.out.println(addr);
}
}

```

Output

```
D:\test>java InetAdd2
aap/192.168.0.166
vtubooks.com/202.137.237.142
```

While running the above program if your computer is not connected to internet then it will generate following output [In fact here is the real use of throwing an exception].

Output

```
D:\test>javac InetAdd2.java
```

```
D:\test>java InetAdd2
aap/127.0.0.1
```

```
Exception in thread "main" java.net.UnknownHostException: vtubooks.com
at java.net.Inet4AddressImpl.lookupAllHostAddr(Native Method)
at java.net.InetAddress$1.lookupAllHostAddr(Unknown Source)
at java.net.InetAddress.getAddressFromNameService(Unknown Source)
at java.net.InetAddress.getAllByName0(Unknown Source)
at java.net.InetAddress.getAllByName(Unknown Source)
at java.net.InetAddress.getAllByName(Unknown Source)
at java.net.InetAddress.getByName(Unknown Source)
at InetAdd2.main(InetAdd2.java:8)
```

To avoid such dirty output we can slightly modify the above program as follows -

Java Program

```
/******
This program shows the use of the method getByName
for getting the IP address for the corresponding host name
*****/
import java.net.*;
class InetAdd2
{
public static void main(String args[]) throws UnknownHostException
{
try
{
InetAddress addr=InetAddress.getLocalHost();
System.out.println(addr);
Address=InetAddress.getByName("vtubooks.com");
System.out.println(addr);
}
catch(UnknownHostException e)
{
System.out.println(e);
}
}
}
```

Output

```
D:\test>javac InetAdd2.java
```

```
D:\test>java InetAdd2
aap/127.0.0.1
java.net.UnknownHostException: vtubooks.com
```


Example 6.3.3 Write a Java program to illustrate the factory method `getAllByName()`.

Solution : This function is used to find the several machines that are associated with single several IP addresses. There are some cases in which single domain name may be associated with several machines. Here is an illustration

```

/*****
This program shows the use of the method getAllByName
for getting all the IP addresses related to the same host name
*****/
import java.net.*;
class InetAdd3
{
public static void main(String args[]) throws UnknownHostException
{
    InetAddress[]
    addr=InetAddress.getAllByName("www.microsoft.com");
    for(int i=0;i<addr.length;i++)
    System.out.println(addr[i]);
}
}

```

Output

```

D:\test>javac InetAdd3.java
D:\test>java InetAdd3
www.microsoft.com/207.46.19.254
www.microsoft.com/207.46.192.254
www.microsoft.com/207.46.193.254
www.microsoft.com/207.46.19.190

```

We can obtain the host name from the IP address as well. In the following program the IP address 192.168.0.166 is given as a string to the method `getByName`. And then using `getHostName` method we can obtain the host name of corresponding machine.

Example 6.3.4 Write a Java program to obtain name of the host machine using corresponding IP address.

Solution :

```

/*
This program shows the use of IP address from which
the host name can be obtained
*/
import java.net.*;
class InetAdd4
{
public static void main(String args[]) throws UnknownHostException

```

```

{
  InetAddress
    addr=InetAddress.getByName("192.168.0.166");
    System.out.println(addr.getHostName());
}
}

```

Output

```

D:\test>javac InetAdd4.java
D:\test>java InetAdd4
aap

```

6.3.2 Instance Methods

Instance methods are the methods that return something which can be used for object or instances

InetAddress class encapsulates several useful instance methods that are listed below -

Method	Description
String getHostName()	It returns the name of the host.
boolean equals(object obj)	If the address of obj equals to the address obtained from InetAddress class, then this function returns true.
byte [] getAddress()	It represents object's internet address in the array of bytes form.
String toString()	Returns a string that shows the host name and IP address.
String getHostAddress()	Returns the address of host associated with object InetAddress class.

6.3.3 Inet4Address and Inet6Address

- There are two new classes **Inet4Address** and **Inet6Address** introduced in **Java 1.4** which are inheriting the basic **InetAddress** class.
- These classes are for supporting **IPv4** and **IPv6** respectively.
- Typically Java programmer is not concerned with IPv4 and IPv6 stuff because his work area is restricted to application layer.
- For example - Consider the method

public boolean isIPv4CompatibleAddress()

this return true if **InetAddress** is an **IPv4** compatible **IPv6** address.

Review Questions

1. Explain factory and instance methods.
2. What is IP4 and IP6 ?

6.4 URL

- For identifying the documents on the internet the uniform or universal resource locator i.e. URL is used.
- There is variety of URL depending upon the type of resources.

6.4.1 Format

- The general format of URL is -

Scheme:Address

That is

```
protocol://username@hostname/path/filename
```

- The scheme specifies the **communication protocol**. Different schemes have different schemes have different forms of addresses.
- Various schemes that are used are http, ftp, gopher, file, mailto, news and so on.
- The most commonly used protocol for web browser and web server communication is **Hypertext Transfer Protocol(HTTP)**. This protocol is based on request-response mechanism. This protocol handles the documents that are created using eXtensible **Hypertext Markup Language (XHTML)**. Using **http** the **Address** part of URL can be written as follows -

```
//Domain_name/path_to_Document
```

- **file** is another most commonly used scheme in URL. This protocol allows to reside the document in the client's machine from which the web browser is making out the demand. Due to this the actual document is not available only for its visibility but it can be tested. Using **file** the Address part of URL can be written as follows -

```
file://path-to-document
```

- The **hostname** is the name of the server computer that stores the web documents.
- The default port number for the http protocol is 80.
- Any URL does not allow the spaces in it. But there are some special characters that can be present in the URL. These characters are - ampersand & or percentage % .

6.4.2 The URL Path

- The path to the web document is similar to the path to the particular file present in the folder. In this path the directory names and files are separated by the separator characters. The character that is used as a separator is **slash**. Unix system uses forward slash whereas the windows system makes use of backward slash.

For example -

```
http://www.mywebsite.com/mydocs/index.html
```

- The URL path that includes all the directories along the path to the file is called the **complete path**.
- Sometimes the base URL path is specified in the configuration file of the server. In such a case we need not have to specify the complete path for accessing the particular file such a path is called the **partial path**.
- For example -

```
http://www.mywebsite.com
```

This indicates that the file mydocs/index.html is specified in the configuration file.

6.4.3 The URL Class

- The Java's URL class has various constructors. These constructors throw the exception **MalformedURLException**.
- The syntax to specify the URL constructor is as follows -

```
URL(String urlString);
```

```
URL(String protocolName, String hos, int port, String path)
```

```
URL(String protocolName, String host, String path)
```

- Various methods in URL class that are commonly used in Java programs are -

Method	Description
getProtocol()	This method returns the name of the protocol which is typically used. Generally http is the protocol being used.
getPort()	It returns the port number. For http the port number is 80.
getHost()	This method returns the host name.
getFile()	This method returns the name of the file which we want to access.

Following is a simple Java program which makes use of URL class for obtaining the protocol host name, file name being used in the URL.

Example 6.4.1 Write a Java program to obtain the name of the protocol, port number, host name and file name of the URL.

Solution :

```
import java.net.*;
class MyURLDemo
{
    public static void main(String args[])throws MalformedURLException
    {
```

```
URL obj=new URL("http://technicalpublications.org/index.php/");
System.out.println("Protocol: " + obj.getProtocol());
System.out.println("Port: " + obj.getPort());
System.out.println("Host: " + obj.getHost());
System.out.println("File: " + obj.getFile());
}
}
```

Output

```
Protocol: http
Port: -1
Host: technicalpublications.org
File: /index.php/
```

Program explanation : We get the protocol name as http, the host name as the name of the website and the corresponding file name. The port value as -1 indicates that it is not set explicitly.

Review Question

1. What is URL ?

6.5 URLConnection

- The class `URLConnection` is used for accessing the attributes of remote resource. These attributes are exposed by the HTTP protocol.
- Using the **`OpenConnection()`** method of `URL` class we can examine the contents. This method is used to establish the connection with some specific web site. Hence to get the contents of the web page we used following statement -
- `URLConnection handle_connection=handle.openConnection();`
- Here **`handle_connection`** is an object of class **`URLConnection`**. Thus the **`openConnection()`** method returns object of **`URLConnection`**.
- Using this object we can further get the access for desired webpage.
- Here is a sample java program which accepts the URL of some web site. Then using **`openConnection()`** method we can get the information about that web page. This information could be current date or content type (such as text/html or image/gif or audio/midi and so on). In the following program we are displaying the contents of the web page on command prompt. Just observe the output of this program carefully.

Example 6.5.1 Write a Java program to display the date, content-type, content length and contents of a web page on command prompt.

Solution :

```
import java.net.*;

import java.io.*;
import java.util.Date;
class Get_Web_Page
{
public static void main(String args[] ) throws Exception
{
int ch;
URL handle=new URL("http://www.yahoo.com");
URLConnection handle_connection=handle.openConnection();
long date_info;
int length,i;
date_info =handle_connection.getDate();
System.out.println("Date: "+new Date(date_info));//Printing the current Date
System.out.println("Content-Type: "+handle_connection.getContentType());
//printing the content types
length=handle_connection.getContentLength();
System.out.println("Content length: "+length);//printing the length of contents

if(length!=0)
{
System.out.println("\n\t*****Contents of web page are*****\n\n");

InputStream input_string=handle_connection.getInputStream();
while((ch=input_string.read())!=-1)
{

System.out.print((char)ch); //printing web page contents

} //end of while
input_string.close();
} //end of if
else
{
System.out.println("There are no Contents for this site");
}
} //end of main
} //end of class
```

Output

```

Date: Wed Feb 04 11:38:56 IST 2015
Content-Type: text/html
Content length: 1450
*****Contents of web page are*****

<!DOCTYPE html>
<html lang="en-us"><head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <meta charset="utf-8">
  <title>Yahoo</title>
....
....
....
  
  <h1 style="margin-top:20px;">Will be right back...</h1>
  <p id="message-1">Thank you for your patience.</p>
  <p id="message-2">Our engineers are working quickly to resolve the issue.</p>
  </td>
</tr>
</tbody> </table>

</body></html>

```

Program explanation :

In above program we have used

```

URL handle=new URL("http://www.yahoo.com");
URLConnection handle_connection=handle.openConnection();

```

In order to connect to the web site "*www.yahoo.com*".

The **handle_connection** is the object variable being used for this purpose. Then using **getDate()** and **getContentType()** methods we can get the current date and content type and this information is displayed on the console.

Then in order to read the contents of the web site we have to create one input stream using **InputStream** with the help of following code -

```

InputStream input_string = handle_connection.getInputStream();

```

Then using the **read()** function we can read the contents character by character manner in the while loop.

```

ch=input_string.read()

```

The output of this program shows the contents of the web page "*www.yahoo.com*"

6.6 HttpURLConnection

- The **HttpURLConnection** class is a subclass of the **URLConnection** class. It provides support for HTTP-specific features. It allows you to read the web page content, the return code, content type or MIME type and so on. The syntax for the constructor of **HttpURLConnection** is as follows -

HttpURLConnection(URL url)

Various methods that can be defined in **HttpURLConnection** class are as follows -

Method	Meaning
void connect()	Connects to the server and issues the request.
void disconnect()	Closes all the connections to this server.
protected HttpURLConnection getConnection(URL url)	Returns an HttpURLConnection.
static String getDefaultRequestProperty(String name)	Gets the value for a given default request header.
InputStream getInputStream()	Gets an input stream from which the data in the response may be read.
OutputStream getOutputStream()	Gets an output stream which can be used send an entity with the request.
String getRequestMethod()	Return the request method used.
String getRequestProperty(String name)	Gets the value of a given request header.
int getResponseCode()	Get the response code.
String getResponseMessage()	Get the response message describing the response code.
URL getURL()	Gets the url for this connection.
void setRequestMethod(String method)	Sets the request method .
void setRequestProperty(String name, String value)	Sets an arbitrary request header.
String toString()	produces a string.

Following is a simple Java program that makes use of **HttpURLConnection** class to display the return code.


```
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;

public class ReturnCodeDemo
{
    public static void main(String[] args) throws IOException
    {
        String urlstring = "http://www.yahoo.com";
        URL url = new URL(urlstring);
        int responseCode = ((HttpURLConnection) url.openConnection()).getResponseCode();
        System.out.println(responseCode);
    }
}
```

Various return code and their meanings are enlisted in the following table.

Return Code	Meaning
200	Ok
301	Permanent redirect to another webpage
400	Bad request
404	Not found

6.7 The URI Class

- URI stands for Uniform Resource Identifier. It is a string of characters used to identify a name of a resource.
- The most common form of URI is the Uniform Resource Locator (URL), frequently referred to informally as a *web address*.
- URLs constitute a subset of URIs. A URI represents a standard way to identify a resource. A URL also describes how to access the resource.
- Following program makes use of URI class

```
import java.net.URI;
import java.net.URISyntaxException;

public class URIClassDemo
{
    public static void main(String[] args) throws NullPointerException, URISyntaxException
    {
        URI uri = new URI("http://www.techicalpublications.org");
        System.out.println("URI : " + uri);
        System.out.println("Authority : " + uri.getAuthority());
    }
}
```

```

System.out.println("RawUserInfo : " + uri.getRawUserInfo());

}
}

```

Output

```

URI : http://www.techicalpublications.org
Authority : www.techicalpublications.org
RawUserInfo : null

```

Review Question

1. Write short note on - URI class.

6.8 Cookies

Cookies are some little information that can be left on your computer by the other computer when we access an internet.

- Generally this information is left on your computer by some advertising agencies on the internet. Using the information stored in the cookies these advertising agencies can keep track of your internet usage. For the applications like on-line purchase systems once you enter your personal information such as your name or your e-mail ID then it can be remembered by these systems with the help of cookies. Sometimes cookies are very much dangerous because by using information from your local disk some malicious data may be passed to you. So it is upto you how to maintain your own privacy and security.
- A cookie is a **name-value pair** information. This information is passed from server to browser in response header. The browser then returns these cookies unchanged to the server by including the **state**. By returning a cookie to a web server, the browser provides the server a means of connecting the current page view with previous page views. Use of session-ID in session tracking using the cookies can be illustrated as follows -

Suppose we want to access web page

<http://www.mywebpage.com/introduction.html>, then the browser connects to the server www.mywebpage.com by making a request.

```

GET /introduction.html HTTP P.1

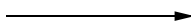
HOST www.mywebpage.com

```

Browser

Requests

Server



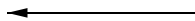
The server then replies in the form of **HTTP response**. This packet contains a line requesting browser to store cookies.

HTTP/P.1	200 OK
Set-Cookie :	sid = xf1234ad

Browser

Response

Server



Using set-cookie statement server is requesting browser to store the sid=xf1234ad. So if browser supports cookies then every subsequent page request to the same server will contain the cookie.

GET	/Chapter1.html	HTTP1.1
Host	www.mywebpage.com	
Cookie	sid = xf1234ad	

Browser

Request

Server



This is another request to the same server. By including cookies which contain **sid=xf1234ad** server knows that this request is related to the previous one. Thus server-browser can keep track of current session.

6.9 TCP,IP and UDP

TCP

- Transmission Control Protocol(TCP) is a **connection-oriented, reliable** protocol which supports the transfer of data in continuous **streams**.
- This is called connection-oriented protocol because **control information** is sent before transmitting any data. This process is sometimes called as **handshaking**.
- This is a reliable protocol because any data which when gets lost or corrupted, the TCP has a provision to retransmit it. Because of these characteristics, TCP is used by most internet applications. However, TCP requires a lot of overhead while coding it.

- The **addressing scheme** used in TCP is by means of **ports**. On separate ports the communication can be established concurrently by the system. During this communication, the server waits for the client to get connected to a specific port for establishing communication. This type of communication is called as **socket programming**.

IP

- Internet protocol is a major protocol in the TCP/IP suit.
- It is a **connectionless, unreliable** protocol which supports the transfer of data in the form of **packets**.
- This is called connectionless protocol because it does not exchange any **control information** before transmitting any data. The data is just sent to the destination with a hope that it will reach at appropriate place.
- IP is known as an unreliable protocol because it does not have the provision of retransmitting the lost packets or detect the corrupted data.
- The **addressing scheme** used in IP is by means of **IP addresses**. An IP address is a 32-bit unique number. IP addresses are generally written as four numbers, between 0 and 255, separated by period. Using the IP address defined in IP packet header, the IP packets can be routed to its destination.

UDP

- The User Datagram Protocol (UDP) is a low-overhead protocol which can be used as an alternative to TCP protocol.
- The UDP is a **connectionless, unreliable** protocol in which the data is passed in the form of **datagrams**.
- This is called connectionless protocol because it does not exchange any **control information** before transmitting any data. The data is just sent to the destination with a hope that it will reach at appropriate place.
- UDP is known as an unreliable protocol because it does not have the provision of retransmitting the lost datagram or detect the corrupted data.
- The addressing scheme used UDP is by means of **ports**. On separate ports the communication can be established concurrently by the system. UDP ports are distinct from the TCP port.

Difference between TCP and UDP

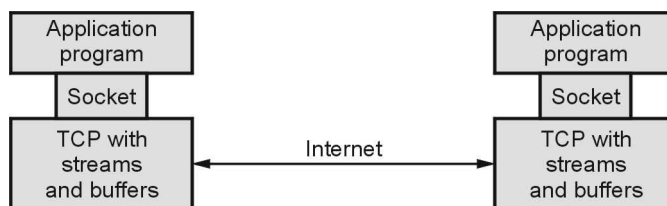
Sr. No.	TCP	UDP
1.	TCP stands for transmission control protocol.	UDP stands for user datagram protocol.
2.	It is a connection oriented protocol with acknowledgement . When a file or message send it will get delivered unless connections fails. If connection lost, the server will request the lost part. Hence it is called reliable protocol .	It is a connectionless protocol without any acknowledgement . When you send a data or message, you don't know if it'll get there, it could get lost on the way. Hence it is called unreliable protocol .
3.	The message will get transferred in an orderly manner .	The message transfer have no order .
4.	It is slower than UDP.	It is a faster protocol.
5.	When the low level parts of the TCP stream arrive in the wrong order resend requests have to be sent and all the out of sequence parts have to be put back together so this protocol is called heavyweight protocol .	No ordering of messages no tracking connections. Hence UDP is called lightweight protocol .
6.	Examples : Email, FTP, Secure Shell protocol makes use of TCP.	Example : Streaming media applications such as movies, Voice Over IP(VOIP), online multiplayer games makes use of UDP.

Review Question

1. Differentiate TCP and UDP.

6.10 TCP/IP Client Sockets

- A socket is bound to a port number so that the TCP/UDP from transport layer can identify the corresponding application at destination.

**Fig. 6.10.1**

- TCP sockets are denoted by streams and server is a device which has resources and from which the services can be obtained. For example : There are various types of servers such as web server which is for storing the web pages, there are print servers for managing the printer services or there are database servers which store the databases.
- Client is a device which wants to get service from particular server.
- First of all server starts and gets ready to receive the client connections. The server-client communications occurs in following steps

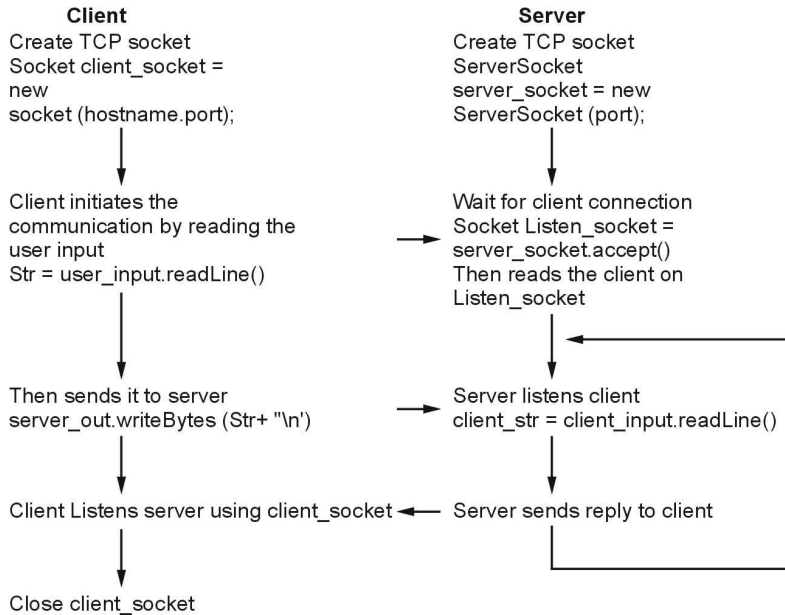


Fig. 6.10.2 Client - Server communication

- There are two constructors used to create **client socket** -

Constructor	Meaning
Socket(String <i>hostName</i> , int <i>port</i>)	Creates a socket connecting the local host to the named host and port.
Socket(InetAddress <i>ipAddress</i> , int <i>port</i>)	Creates a socket using a pre-existing InetAddress object and a port.

- The methods used for examining the socket are -

InetAddress getInetAddress()	Returns the InetAddress associated with the Socket object.
int getPort()	Returns the remote port to which this Socket object is connected.
int getLocalPort()	Returns the local port to which this.

Whois

- Whois is a query and response protocol that is widely used for querying databases that store the registered users of an Internet resource, such as a domain name, an IP address block or an autonomous system.
- The protocol stores and delivers database content in a human readable format.
- The WHOIS protocol is a TCP-based protocol designed to work on the port 43.

Review Question

1. Explain steps used in socket programming.

6.11 TCP/IP Server Sockets

- The **ServerSocket** class is used to create servers that listen to its clients.
- When **ServerSocket** is created, it will register itself with the system so that clients can connect to it. This class throws exception **IOException**.
- Various ways by which **ServerSocket** can be created is as follows -

```
ServerSocket(int port)
```

```
ServerSocket(int port, int maxQueue)
```

```
ServerSocket(int port, int maxQueue, InetAddress localAddress)
```

- Using the **accept()** method, the server initiates the communication with the client.

TCP socket programming

The socket programming includes two programs- one at the server side which is called as **server program** and other at the client side which is called as **client program**.

Let us discuss various application programs based on server client communication using TCP.

Example 6.11.1 Write a TCP socket programming application in which client sends 'Hello' message to the server.

Solution :

Step 1 : The server program can be written on Notepad as follows -

```
/*
*****
Server Program
*****
*/
import java.io.*;
import java.net.*;
class Server
```

```

{
public static void main(String args[]) throws Exception
{
ServerSocket server_socket=new ServerSocket(1234); //Step 1:Creating serversocket
while(true)
{
Socket Listen_socket=server_socket.accept(); //Step 2: Server is ready to Listen on
//ServerSocket

BufferedReader client_input=new BufferedReader(new
InputStreamReader (Listen_socket.getInputStream()));
String client_str;
client_str=client_input.readLine(); //Step 3: reading data from client in Client_str
System.out.println(client_str); //Step 4:Displaying 'Hello' obtained from client
}
}
}

```

Step 2 : The client program can be written on the Notepad as follows -

```

/*
*****
Client Program
*****
*/
import java.io.*;
import java.net.*;
class Client
{
public static void main(String args[]) throws Exception
{

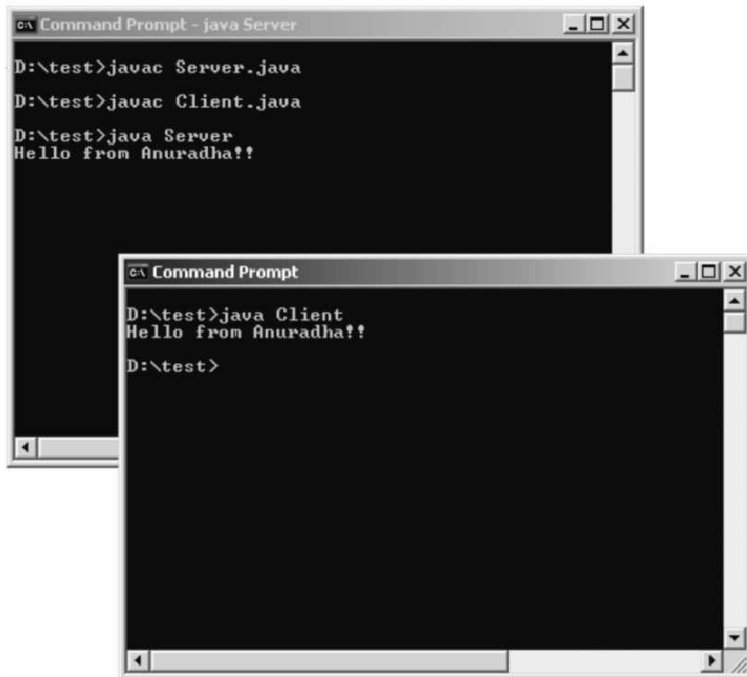
Socket client_socket=new Socket("aap",1234); //Step 1: Creating socket for client
BufferedReader user_input = new BufferedReader(new
InputStreamReader(System.in));
DataOutputStream server_out=new
DataOutputStream(client_socket.getOutputStream());

String Str;
Str=user_input.readLine(); //step 2: Client reading 'Hello' message typed by user
server_out.writeBytes(Str+"\n"); //Step 3: Client sending 'Hello' over the output stream of
//server
client_socket.close(); //Step 4: Closing client socket
}
}

```


Step 3 : These two programs can be run on separate command-prompt windows so that we can see the client-server communication getting established. Note that the **server program must be running before starting the client**. Here is an illustrative output

Output



```
Command Prompt - java Server
D:\test>javac Server.java
D:\test>javac Client.java
D:\test>java Server
Hello from Anuradha!!

Command Prompt
D:\test>java Client
Hello from Anuradha!!

D:\test>
```

Program explanation :

Server program

- In the server program first two lines are

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

- The **java.io package** is required to support I/O operations. And for socket programming **java.net package** is required because it contains the class **Socket** which is required in our program. **Socket** class creates a stream socket and connects it to a specific port at a specific IP address/hostname. Hence in the main() we have written

```
ServerSocket server_socket=new ServerSocket(1234);
```

Here *aap* is a host name of the machine on which the server is running and 1234 is the port number. That means the server creates a socket *server_socket* for establishing the connection with the client.

- The server creates another separate socket for listening to the client

```
Socket Listen_socket=server_socket.accept();
```

- Thus only on Listen_socket server can listen to the client. Then comes

```
BufferedReader client_input=new  
BufferedReader(newInputStreamReader(Listen_socket.getInputStream()));
```

- The *BufferedReader* is for creating the object *client_input*. We have used **getInputStream()** method which returns an input stream from the other side of the socket. The *InputStreamReader* takes **getInputStream()** as a parameter.
- Then in the string *client_str* the client message can be collected by a using a method *readLine()* as follows

```
String client_str;  
client_str=client_input.readLine();
```

- Then the message obtained from client is printed on the console using following code

```
System.out.println(client_str);
```

Client program

- Now let us discuss the client program. In client program we have to create a socket for client as follows

```
Socket client_socket=new Socket("aap",1234);
```

- When the client runs something must be typed on the console and that message will be sent to the server.
- Using *System.in* we can get the input written on the console. Hence to get the message typed by the user following declaration must be made

```
BufferedReader user_input = new BufferedReader(new  
InputStreamReader(System.in));
```

- The **BufferedReader** object is created to read the input from keyboard.
- The **InputStreamReader** is for reading the stream input. We have send *System.in* as a parameter to **InputStreamReader**. Then an output stream has to be created on which the data can be sent to the server. Hence we declare *server_out* as an output stream.

```
DataOutputStream server_out=new  
DataOutputStream(client_socket.getOutputStream());
```

- The method *getOutputStream()* returns an output stream to the other side the socket. Then using *readLine()* function we can read the message written on the console.

```
Str=user_input.readLine();
```

- The client then sends this message (collected in string Str) to the server using **writeBytes()** method as follows

```
server_out.writeBytes(Str+"\n");
```

- Finally the client closes its connection by

```
client_socket.close();
```

- Last but not the least, we have to throw some exception to handle the unknown host or connection not getting established situation. Hence the **main()** method throws a general exception **Exception**.

Example 6.11.2 Write a TCP socket programming application in which client sends some message to the server and server sends the acknowledgement to the client.

Solution :

Step 1 :

Server program

```

/*****
    Server Program which sends acknowledgement to the client
    *****/
import java.io.*;
import java.lang.*;
import java.net.*;
class S
{
    public static void main(String args[]) throws Exception
    {
        String str3;
        String str4;
        ServerSocket s2=new ServerSocket(1234);
        while(true)
        {
            Socket s3=s2.accept();
            BufferedReader in_client=new BufferedReader(new InputStreamReader(s3.getInputStream()));
            DataOutputStream out_client=new DataOutputStream(s3.getOutputStream());
            str3=in_client.readLine();
            str4=str3+"—>Received"+"\n";
            out_client.writeBytes(str4);
        }
    }
}

```

Input Stream is created to send the acknowledgment to the client.

Using **writeBytes** method the acknowledgment is sent to client

Step 2 :

Client program

```

/*****
    Client Program
    *****/

```

```
import java.io.*;
import java.net.*;
class C
{
public static void main(String args[]) throws Exception
{
String Str;
String Str1;
BufferedReader input_user = new BufferedReader(new
InputStreamReader(System.in));
Socket s1=new Socket("aap",1234);
DataOutputStream out_server=new
DataOutputStream(s1.getOutputStream());
BufferedReader in_server=new BufferedReader(new
InputStreamReader(s1.getInputStream()));
Str=input_user.readLine();
out_server.writeBytes(Str+"\n");
Str1=in_server.readLine();
System.out.println("From server: "+ Str1);
s1.close();
}
}
```

Client reading the acknowledgment from the server and printing it on its console

Step 3 :

Output

```
Command Prompt - java S
D:\test>javac S.java
D:\test>javac C.java
D:\test>java S

Command Prompt
D:\test>java C
Hello how,are you?
From server: Hello how,are you?-->Received
D:\test>_
```

Example 6.11.3 Write a client program to send any string from its standard input to the server program. The server program reads the string, finds number of characters and digits and sends it back to client program. Use connection-oriented or connection-less communication.

Solution : Following application makes use of **connection oriented socket programming**.

Step 1 : We will write a client program which accepts some string containing alphanumeric string. This string is then sent to the server. The program is as follows -

```

/*****
Client Program
*****/
import java.io.*;
import java.net.*;
class Client
{
public static void main(String args[]) throws Exception
{
String Str=" ";
String Str1;
String Str2;
Socket s1=new Socket("127.0.0.1",1234);
try
{
System.out.println("Enter Some string...");
while(true)
{
BufferedReader input_user = new BufferedReader(new InputStreamReader(System.in));
DataOutputStream out_server=new DataOutputStream(s1.getOutputStream());
BufferedReader in_server=new BufferedReader(new InputStreamReader(s1.getInputStream()));
Str=input_user.readLine();
out_server.writeBytes(Str+"\n");
Str1=in_server.readLine(); //obtaining the from server
Str2=in_server.readLine(); //obtaining the from server
//displaying this count on client console
System.out.println("Number of digits is "+Str1);
System.out.println("Number of characters is "+Str2);
s1.close();
} //end of while
} catch (Exception e) {System.out.println(e.getMessage());}
}
}

```

Step 2 : Now we will write down some server program which will accept the string from the client and count the total number of characters and digits into it. These counts will be then returned to the client. The client program will display these counts. The server program is follows -

```

/*****
Server Program
*****/
import java.io.*;
import java.lang.*;
import java.net.*;
class Server
{
public static void main(String args[]) throws Exception
{
ServerSocket s2=new ServerSocket(1234);
String text=" ";
int counts=0;
int charcounts=0;
System.out.println("\t\tServer started.....");
while(true)
{
Socket s3=s2.accept();
BufferedReader in_client=new BufferedReader(new InputStreamReader(s3.getInputStream()));
DataOutputStream out_client=new DataOutputStream(s3.getOutputStream());
text=in_client.readLine();//getting data from client
System.out.println("Receiving data...");
for(int i=0;i<text.length();i++)
{
if((text.charAt(i)>='0')&&(text.charAt(i)<='9'))
counts++;
}
}
out_client.writeBytes(counts+"\n");//sending number of digits to client
charcounts=text.length()-counts;
out_client.writeBytes(charcounts+"\n");//sending number of characters to client
System.out.println("Data is sent...");
}
}
}
}

```

Step 3 : Now open two command prompt windows to get the output. First execute the server program and then the client program. The output is as follows -

```

Command Prompt - java Server
D:\Bank> javac Server.java
D:\Bank> javac Client.java
D:\Bank> java Server
Server started....
Receiving data...
Data is sent...

Command Prompt
D:\Bank> java Client
Enter Some string...
Hello123Friends
Number of digits is 3
Number of characters is 12
Socket is closed
D:\Bank>

```

Example 6.11.4 Write a client server program using TCP where client sends a string and server checks whether that string is palindrome or not and responds with appropriate message.

Solution :

Step 1 :

```

/*****
Client Program
*****/
import java.io.*;
import java.net.*;
class Client_Pal
{
public static void main(String args[]) throws Exception
{
String Str=" ";
String Str1;
int flag;
Socket s1=new Socket("127.0.0.1",5000);
try
{
System.out.println("Enter Some string...");
while(true)
{
BufferedReader input_user = new BufferedReader(new InputStreamReader(System.in));
DataOutputStream out_server=new DataOutputStream(s1.getOutputStream());
BufferedReader in_server=new BufferedReader(new InputStreamReader(s1.getInputStream()));
Str=input_user.readLine();
out_server.writeBytes(Str+"\n");
Str1=in_server.readLine(); //obtaining the from server
flag=Integer.parseInt(in_server.readLine()); //obtaining the from server
}
}
}
}

```

```

    if(flag== 1)
        System.out.println(Str1+" is Palindrome");
    else
        System.out.println(Str1+" is not palindrome");
    s1.close();
} //end of while
} catch(Exception e){System.out.println(e.getMessage());}
}
}

```

Step 2 :

```

/*****
Server Program
*****/
import java.io.*;
import java.lang.*;
import java.net.*;
class Server_Pal
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket s2=new ServerSocket(5000);
        String str="",rev="";
        int flag;
        System.out.println("\t\tServer started.....");
        while(true)
        {
            Socket s3=s2.accept();
            BufferedReader in_client=new BufferedReader(new InputStreamReader(s3.getInputStream()));
            DataOutputStream out_client=new DataOutputStream(s3.getOutputStream());
            str=in_client.readLine();//getting data from client
            System.out.println("Receiving data...");
            int length = str.length();
            for ( int i = length - 1; i >= 0; i-- )
                rev = rev + str.charAt(i);
            if (str.equals(rev))
                flag=1;
            else
                flag=0;
            out_client.writeBytes(str+"\n");//sending string to client
            out_client.writeBytes(flag+"\n");//sending status of palindrome or not
            System.out.println("Data is sent...");
        } //end of while
    } //end of main
} //end of class

```


6.12 Datagrams

- A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time and content are not guaranteed.
- It is a basic transfer unit associated with a packet-switched network. The applications that communicate via datagrams send and receive completely independent packets of information.
- The **java.net** package contains three classes to help you write Java programs that use datagrams to send and receive packets over the network : **DatagramSocket**, **DatagramPacket**, and **MulticastSocket**.
- An application can send and receive **DatagramPackets** through a **DatagramSocket**. In addition, **DatagramPackets** can be broadcast to multiple recipients all listening to a **MulticastSocket**.

6.12.1 Datagram Packet

- The **DatagramPacket** is a message that can be sent or received.
- An application can send and receive **DatagramPackets** through a **DatagramSocket**.
- If you send multiple packets then these packets may arrive in any order. Moreover, the delivery of packets is also not guaranteed.

Constructors used for DatagramPacket

- **DatagramPacket(byte[] barr, int length)** : It creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port)** : It creates a datagram packet. This constructor is used to send the packets.

Methods

Here are some useful methods for UDP socket programming

Method	Description
<code>InetAddress getByName(String hostname)</code>	This method returns the IP address when the <i>hostname</i> is given.
<code>InetAddress getAddress()</code>	This method returns the IP address.
<code>int getPort()</code>	It returns the port number.
<code>Byte []getData()</code>	It returns the data containing in the datagram. This is stored in the array of bytes.
<code>int getLength()</code>	Returns the length of data obtained by <code>getData</code> method.

6.12.2 Datagram Server and Client

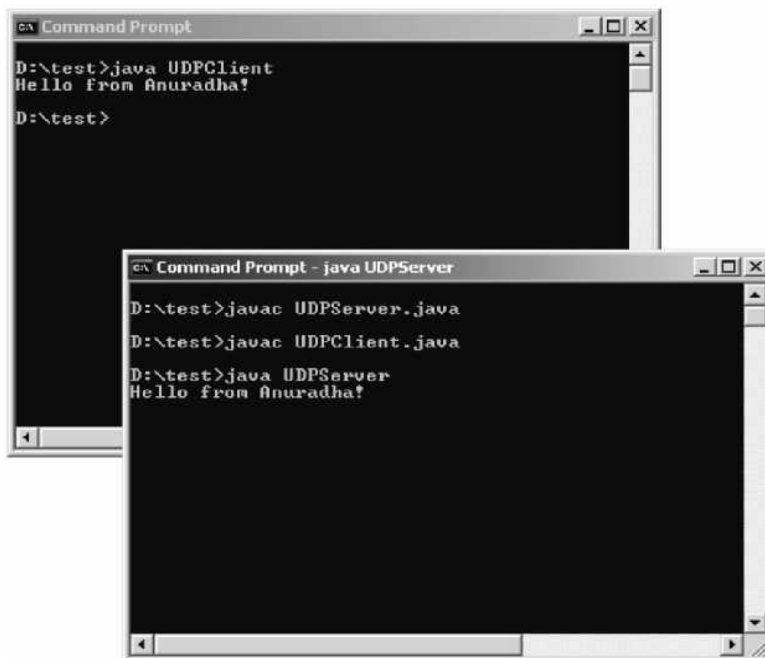
Example 6.12.1 Write an UDP client and server program to send some message to the server and server displays the received message on its console.

Solution :

```
/*
*****
                        UDP Client Program
*****
*/
import java.io.*;
import java.net.*;
class UDPClient
{
public static void main(String args[]) throws Exception
{
    BufferedReader user_input=new BufferedReader(new
    InputStreamReader(System.in));
    //creating the client socket
    DatagramSocket client_socket=new DatagramSocket();
    //getting the IP address of host "aap"
    InetAddress IP_add=InetAddress.getByName("aap");
    //creating the buffer for sending the data
    byte out_data[]=new byte[1024];
    //reading the input through keyboard
    String str=user_input.readLine();
    out_data=str.getBytes();
    //creating the datagram packet in which data is
    //encapsulated
    DatagramPacket Packet1=new
    DatagramPacket(out_data,out_data.length,IP_add,1234);
    //sending the packet to the server
    client_socket.send(Packet1);
    //closing the client's connection
    client_socket.close();
}
}
/*
*****
                        UDP Server Program
*****
*/
import java.io.*;
import java.net.*;
```

```
class UDPServer
{
public static void main(String args[]) throws Exception
{
//creating the socket for server
DatagramSocket server_socket=new DatagramSocket(1234);
//array which reserves the input data getting from
//client
byte in_data[]=new byte[1024];
while(true)
{
//creating the datagram packet
DatagramPacket Packet2=new
DatagramPacket(in_data,in_data.length);
//the data from the client is received in the packet
server_socket.receive(Packet2);
String str=new String(Packet2.getData());
//printing the received data on the console of server
System.out.println(str);
}
}
}
```

Output



```
cmd Command Prompt
D:\test>java UDPClient
Hello from Anuradha!
D:\test>

cmd Command Prompt - java UDPServer
D:\test>javac UDPServer.java
D:\test>javac UDPClient.java
D:\test>java UDPServer
Hello from Anuradha!
```

Example 6.12.2 Write UDP client and server program in which UDP client sends some message to the server and server sends some another message to the client.

Solution :

```

/*
*****
                UDP Client Program
*****
*/
import java.io.*;
import java.net.*;
class UDPClient
{
public static void main(String args[]) throws Exception
{
    BufferedReader user_input=new BufferedReader(new
    InputStreamReader(System.in));
    //creating the client socket
    DatagramSocket client_socket=new DatagramSocket();
    //getting the IP address of host "aap"
    InetAddress IP_add=InetAddress.getByName("aap");
    //creating the buffer for sending the data
    byte out_data[]=new byte[1024];
    byte in_data[]=new byte[1024];
    //reading the input through keyboard

    String str=user_input.readLine();
    out_data=str.getBytes();
    //creating the datagram packet in which data is
    //encapsulated
    DatagramPacket Packet1=new
    DatagramPacket(out_data,out_data.length,IP_add,1234);
    //sending the packet to the server
    client_socket.send(Packet1);
        DatagramPacket Packet4=new
    DatagramPacket(in_data,in_data.length);
    //the data from the server is received in the packet
    client_socket.receive(Packet4);
    String receive_str=new String(Packet4.getData());
    //printing the received data on the console of client
    System.out.println(receive_str);
    //closing the client's connection
    client_socket.close();
}
}

```

Sending the data stored
in array **out_data** to the
server in **Packet1**

Receiving the data stored
in array **in_data** from the
server in **Packet4**

```

/*
*****
                UDP Server Program
*****
*/
import java.io.*;
import java.net.*;
class UDPServer
{
public static void main(String args[]) throws Exception
{
//creating the socket for server
DatagramSocket server_socket=new DatagramSocket(1234);
BufferedReader server_input=new BufferedReader(new InputStreamReader(System.in));
InetAddress IP_add=InetAddress.getByName("aap");
//array which reserves the input data getting from client
byte in_data[]=new byte[1024];
byte out_data[]=new byte[1024];
while(true)
{
//creating the datagram packet
DatagramPacket Packet2=new
DatagramPacket(in_data,in_data.length);
//the data from the client is received in the packet
server_socket.receive(Packet2);
String str=new String(Packet2.getData());
//printing the received data on the console of server
System.out.println(str);

InetAddress IP_add1=Packet2.getAddress();
int port=Packet2.getPort();
String send_str=server_input.readLine();
out_data=send_str.getBytes();
DatagramPacket Packet3=new
DatagramPacket(out_data,out_data.length,IP_add1,port);
//sending the packet to the client
server_socket.send(Packet3);
}
}
}

```

Receiving the data stored
in array **in_data** from the
client in **Packet2**

Sending the data stored
in array **out_data** to the
client in **Packet3**

Output

```

C:\> Command Prompt
D:\test> java UDPClient
from client to server
from server to client

C:\> Command Prompt - java UDPServer
D:\test> java UDPServer
from client to server
from server to client
=

```

Example 6.12.3 Write an UDP client and server program to do the following :

Client send any string and server respond with its capital string.

Solution :

```

/*
*****
UDP Client Program[UDPClient.java]
*****
*/
import java.io.*;
import java.net.*;
class UDPClient
{
public static void main(String args[]) throws Exception
{
    BufferedReader user_input=new BufferedReader(new
    InputStreamReader(System.in));
    //creating the client socket
    DatagramSocket client_socket=new DatagramSocket();
    //getting the IP address of host "aap"
    InetAddress IP_add=InetAddress.getByName("localhost");
    //creating the buffer for sending the data
    byte out_data[]=new byte[1024];
    //reading the input through keyboard
    String str=user_input.readLine();

```

```
    out_data=str.getBytes();
    //creating the datagram packet in which data is
    //encapsulated
    DatagramPacket Packet1=new
    DatagramPacket(out_data,out_data.length,IP_add,1234);
    //sending the packet to the server
    client_socket.send(Packet1);
    //closing the client's connection
    client_socket.close();
}
}
/*
*****
UDP Server Program[UDPServer.java]
*****
*/
import java.io.*;
import java.net.*;
class UDPServer
{
public static void main(String args[]) throws Exception
{
//creating the socket for server
DatagramSocket server_socket=new DatagramSocket(1234);
//array which reserves the input data getting from
//client
byte in_data[]=new byte[1024];
while(true)
{
//creating the datagram packet
DatagramPacket Packet2=new
DatagramPacket(in_data,in_data.length);
//the data from the client is received in the packet
server_socket.receive(Packet2);
String str=new String(Packet2.getData());
//printing the received data on the console of server

    System.out.println(str.toUpperCase());
}
}
}
```

Example 6.12.4 Write a UDP client-server program in which the client sends any string and server responds with reserve string.

Solution : Client program

```
import java.io.*;
import java.net.*;
class UDPClient
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader user_input=new BufferedReader(new InputStreamReader (System.in));
        //creating the client socket
        DatagramSocket client_socket=new DatagramSocket();
        //getting the IP address of host "aap"
        InetAddress IP_add=InetAddress.getByName("localhost");
        //creating the buffer for sending the data
        byte out_data[]=new byte[1024];
        //reading the input through keyboard
        String str=user_input.readLine();
        out_data=str.getBytes();
        //creating the datagram packet in which data is
        //encapsulated
        DatagramPacket Packet1=new DatagramPacket(out_data,out_data.length,IP_add,1234);
        //sending the packet to the server
        client_socket.send(Packet1);
        //closing the client's connection
        client_socket.close();
    }
}
```

Server Program

```
import java.io.*;
import java.net.*;
class UDPServer
{
    public static void main(String args[]) throws Exception
    {
        //creating the socket for server
        DatagramSocket server_socket=new DatagramSocket(1234);
        //array which reserves the input data getting from
        //client
        byte in_data[]=new byte[1024];
        while(true)
        {
```



```

//creating the datagram packet
DatagramPacket Packet2=new DatagramPacket(in_data,in_data.length);
//the data from the client is received in the packet
server_socket.receive(Packet2);
String str=new String(Packet2.getData());
//printing the received data on the console of server
String reverse="";
int length = str.length();
for ( int i = length - 1 ; i >= 0 ; i--)
    reverse = reverse + str.charAt(i);
System.out.println("Reverse of entered string is: "+reverse);
}
}
}

```

6.13 Sending Email

The email can be sent in a three step process.

1. Get the session object

- There are two methods of **javax.mail.Session** class that can be used to get the session object.
- These are - **Session.getDefaultInstance()** and **Session.getInstance()** method.
- The **getDefaultInstance** method returns the default session and the **getInstance** returns the new session.

2. Compose the message

- For composing the message the **javax.mail.Message** class is used. But it is an abstract class so its subclass **javax.mail.internet.MimeMessage** class is used.
- While creating the message, we need to pass session object in **MimeMessage** class constructor. The following code can be used for that purpose
- `MimeMessage message=new MimeMessage(session);`
- Thus the message object is created. We will now use the methods of **MimeMessage** class that allow is to build the message for email.

1.	<code>public void setFrom()</code>	This method is used to set the address of the sender.
2.	<code>public void addRecipient()</code>	This method is used to add the address of the recipient.
3.	<code>public void setSubject()</code>	This method is used to set the subject header field.
4.	<code>public void setText()</code>	This method is used to set the text as the message content.

3. Send the message

- The **send()** method of **javax.mail.Transport** class is used to send the email.
- For executing the following program we need to use two jar files namely, **activation.jar** and **mail.jar**

Example Program

```
import java.util.*;  
import javax.mail.*;  
import javax.mail.internet.*;  
import javax.activation.*;
```

```
public class SendEmailDemo  
{  
    public static void main(String [] args){  
        String to = "puntambekar.a@gmail.com";  
        String from = "puntambekar.a@gmail.com";  
        String host = "localhost";
```

//Get the session object

```
Properties properties = System.getProperties();  
properties.setProperty("mail.smtp.host", host);  
Session session = Session.getDefaultInstance(properties);
```

//compose the message

```
try{  
    MimeMessage message = new MimeMessage(session);  
    message.setFrom(new InternetAddress(from));  
    message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));  
    message.setSubject("This is a subject");  
    message.setText("Hello, how are you?");
```

// Send message

```
Transport.send(message);  
System.out.println("message sent successfully....");  
  
}catch (MessagingException ex) {ex.printStackTrace();}  
}  
}
```

6.14 Servlet Overview

- Servlets are basically the Java programs that run on server. These are the programs that are requested by the XHTML documents and are displayed in the browser window as a response to the request.
- The servlet class is instantiated when web server begins the execution.
- The execution of servlet is managed by **servlet container**, such as Tomcat.
- The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.
- Servlets are most commonly used with HTTP (i.e. Hyper Text Transfer Protocol) hence sometimes servlets are also called as "HTTP Servlet".
- The main purpose of servlets is to add up the functionality to a web server.

6.14.1 The Java Web Server

- Before learning the actual servlet programming it is very important to understand how servlet works. (Refer Fig. 6.14.1)

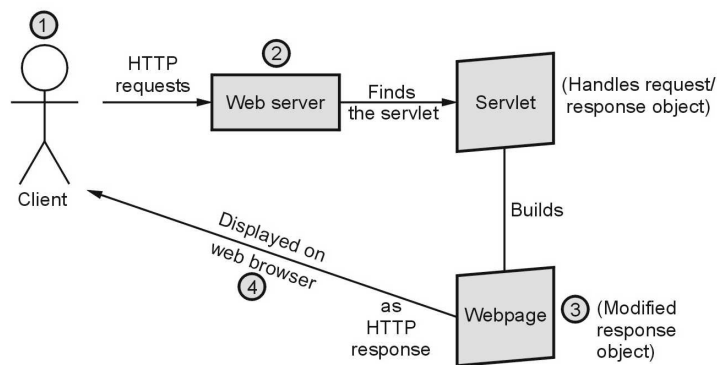


Fig. 6.14.1 How servlet works ?

1. When a client make a request for some servlet, he/she actually uses the **Web browser** in which request is written as a URL.
2. The web browser then sends this request to **Web server**. The web server first finds the requested servlet.
3. The obtained servlet gathers the relevant information in order to satisfy the client's request and builds a **web page** accordingly.
4. This web page is then **displayed** to the client. Thus the request made by the client gets satisfied by the servlets.

6.14.2 Advantages of using Servlets

- The servlets are very **efficient** in their performance and get executed in the address space of the belonging web server.
- The servlets are **platform independent** and can be executed on different web servers.
- The servlets working is based on **Request-Response**. Any HTML form can take the user input and can forward this input to the servlet. The servlets are then responsible to communicate with the back-end database and manipulate the required business logic. These servlets embedded on the web servers using **Servlets API**.
- Servlets provide a way to generate the **dynamic document**. For instance : A servlet can display the information of current user logged in, his logging time, his last access, total number of access he made so far and so on.
- **Multiple users** can keep a co-ordination for some application among themselves using servlets.
- Using servlets **multiple requests** can be synchronized and then can be concurrently handled.

6.14.3 The Life Cycle of a Servlet

- In the life cycle of servlet there are three important methods. These methods are,
1. Init 2. Service 3. Destroy

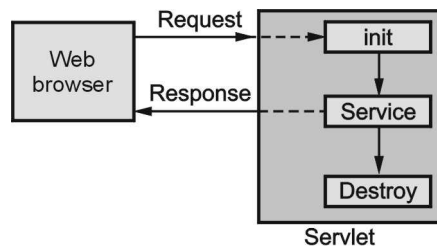


Fig. 6.14.2 Life cycle of servlet

- The client enters the URL in the web browser and makes a request. The browser then generates the HTTP request and sends it to the Web server. (Refer Fig. 6.14.2)
- Web server maps this request to the corresponding servlet.
 - 1. Init () Method :** The server basically invokes the **init()** method of servlet. This method is called only when the servlet is loaded in the memory for the first time. Using this method initialization parameters can also be passed to the servlet in order to configure itself.

2. **service() Method** : Server can invoke the service for particular HTTP request using **service()** method. The servlets can then read the data provided by the HTTP request with the help of **service()** method.
3. **destroy() Method** : Finally server unloads the servlet from the memory using the **destroy()** method.

6.14.4 Your First Servlet

- When we write a servlet program, it is necessary to -
 - i) Either implement Servlet interface or
 - ii) Extend a class that implements **Servlet** interface.
- While implementing **Servlet** interface we must include **javax.servlet** package. Hence the first line in our servlet program must be

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

- **GenericServlet** class is a predefined implementation of **Servlet** interface. Hence we can extend **GenericServlet** class in our servlet program. Similarly, the **HttpServlet** class is a child class of **GenericServlet** class, hence we can extend this class as well while writing the servlet program.
- Hence following are the two ways by which we can write servlet program

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Test extends
GenericServlet
{
    //body of servlet
}
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Test extends HttpServlet
{
    //body of servlet
}
```

- The servlet gets the request from the client for some service. The servlet then processes the request and sends the response back to the client. In order to handle these issues **HttpServletRequest** and **HttpServletResponse** are used in servlet program.
- These requests are handled with the help of some methods that are popularly known as **methods of HttpServlet**. These methods are as follows

Method	Purpose
doGet	This method handles the HTTP GET request
doPost	This method handles the HTTP POST request

doPut	This method handles the HTTP Put request.
doDelete	This method handles the DELETE request.

The doGet and doPost methods

- The **doGet** method requests the data from the source.
- The **doPost** method submits the processed data to the source.
- The protocol of **doGet** method is as follows

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

throws ServletException,IOException

- The **ServletException** and **IOException** are thrown to handle the Servlet problems gracefully.
- The **HttpServletRequest** request : Contain the client request made by client.
- The **HttpServletResponse** response : Contains the response made by servlet back to the client.
- The protocol of doPost method is same as doGet method. It is as follows -

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException
```

- The GET request is more efficient than the POST request.
- The GET request is less secure than the POST request.

How to Write Servlet Program ?

Open Notepad and write the first servlet code to display Greeting messages. It is as follows

FirstServlet.java

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

public class FirstServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws IOException,ServletException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>My First Servlet</title>");
    }
}
```

```
out.println("<body>");
out.println("<h1>Hello How are U?</h1>");
out.println("<h2>I am enjoying this Servlet Application</h2>");
out.println("<h3>See You later!</h3>");
out.println("</body>");
out.println("</html>");
}
}
```

Program Explanation :

- In the above program, we have imported following files,

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

- Out of these files **java.io** package is useful for taking care of I/O operations.
- The **javax.servlet** and **javax.servlet.http** are important packages containing the **classes** and **interfaces** that are required for the operation of **servlets**. The most commonly used interface from **javax.servlet** package is **Servlet**. Similarly most commonly used class in this package is **GenericServlet**. The **ServletRequest** and **ServletResponse** are another two commonly used interfaces defined in **javax.servlet** package.
- In the **javax.servlet.http** package **HttpServletRequest** and **HttpServletResponse** are two commonly used interfaces. The **HttpServletRequest** enables the servlet to read data from the HTTP request and **HttpServletResponse** enables the servlet to write data to HTTP response. The **cookie** and **HttpServlet** are two commonly used classes that are defined in this package.
- We have given class name **FirstServlet** which should be derived from the class **HttpServlet**. (Sometimes we can derive our class from **GenericServlet**).
- Then we have defined **doGet method** to which the HTTP request and response are passed as parameters. The commonly used basic exceptions for the servlets are **IOException** and **ServletException**.
- The MIME type is specified as using the **setContentType()** method. This method sets the content type for the HTTP response to type. In this method "text/html" is specified as the MIME type. This means that the browser should interpret the contents as the HTML source code.
- Then an output stream is created using **PrintWriter()**. The **getWriter()** method is used for obtaining the output stream. Anything written to this stream is sent to the

client as a response. Hence using the object of output stream 'out', we can write the HTML source code in **println** method as HTTP response.

How to execute Servlet program ?

Step 1 : Compile the above program using the javac command at command prompt.

```
D:\test>javac FirstServlet.java
```

The class file for this program gets generated.

Step 2 : Before running any servlet program, it is necessary to have

1. JDK installed
2. Tomcat installed.
3. Class path for above two packages must be set.

For Tomcat installation, I prefer to install the package XAMPP. The XAMP contains a directory for tomcat. The XAMPP package contains Apache Web server, MySQL, PHP and Perl support. It can work on both Windows and Linux operating System.

Step 3 : Copy the class file generated in Step 1 to the path

```
C : \xampp\tomcat\webapps\examples\WEB-INF\classes
```

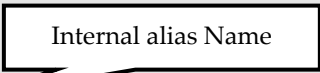
Step 4 : Go to the directory

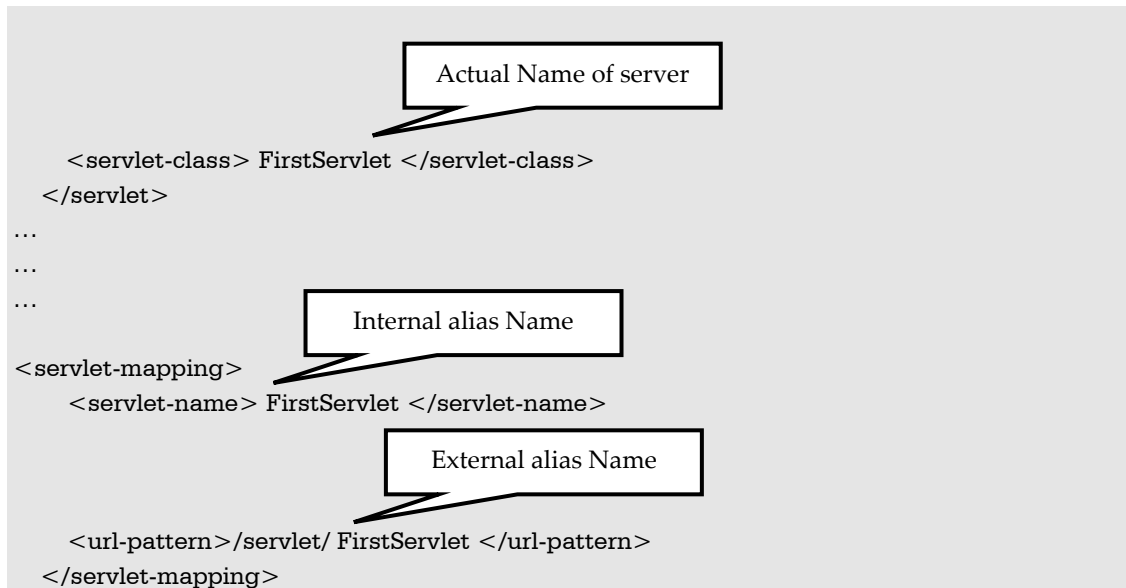
```
C : \xampp\tomcat\webapps\examples\WEB-INF
```

Open **web.xml** file and edit it as follows

```
<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class> FirstServlet </servlet-class>
</servlet>
...
...
...
<servlet-mapping>
  <servlet-name> FirstServlet </servlet-name>
  <url-pattern>/servlet/ FirstServlet </url-pattern>
</servlet-mapping>
```

The **web.xml** file is popularly known as **deployment descriptor**. This is basically the configuration file that describes how to map the URL of web application to servlet.

```
<servlet>
  
  <servlet-name>FirstServlet</servlet-name>
</servlet>
```

The Servlet comes with two alias names, internal and external. The internal name is used by the Tomcat and the external name is given (to be written in `<FORM>` tag of HTML file) to the client to invoke the Servlet on the server. That is, there exists alias to alias. All this is for security. Observe, the names are given in two different XML tags, in the `web.xml` file, to make it difficult for hacking.

To invoke the **FirstServlet** Servlet, the client calls the server with the name **servlet/FirstServlet**.

When **servlet/FirstServlet** call reaches the server, the Tomcat server opens the **web.xml** file to check the deployment particulars. Searches such a `<servlet-mapping>` tag that matches **servlet/FirstServlet**. **servlet/FirstServlet** is exchanged with **FirstServlet**. Then, searches such a `<servlet>` tag that matches **FirstServlet** and exchanges with **FirstServlet** class. Now the server, loads **FirstServlet** Servlet, executes and sends the output of execution as response to client.

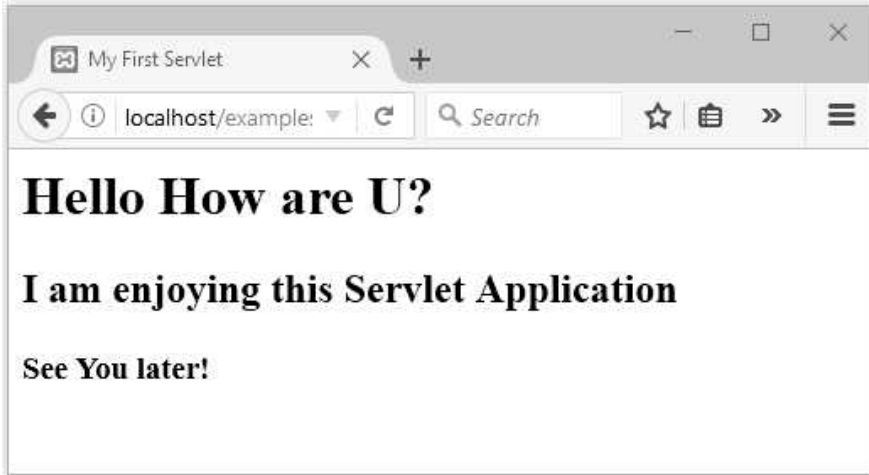
Step 5 : Start tomcat and xampp

Step 6 : Open web browser and type the command

```
http://localhost/examples/servlet/FirstServlet
```

The output will be

Output

**Review Questions**

1. What is servlets ? Explain how it works ?
2. Explain the advantages of servlets.
3. Explain the life cycle methods of servlets.

6.15 Handling HTTP Requests and Response

- Many times we need to pass some information from web browser to Web server. In such case, the HTML document containing the FORM is written which sends the data to the servlet present on the web server.
- To make the form works with Java servlet, we need to specify the following attributes for the <form> tag :
 - i) **method="method name"**: To send the form data as an HTTP POST request to the server.
 - ii) **action="URL/address of the servlet"**: Specifies relative URL of the servlet which is responsible for handling data posted from this form.
- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.
 - i) **GET method** : The GET method sends user information along with ? symbol called query string. For instance
`http://localhost/hello?user = aaaa&age = 20`
In servlet, this information is processed using **doGet()** method.

ii) POST method : This is the most reliable method of sending user information to the server from HTML form. Servlet handles this request using **doPost** method.

Difference between GET and POST

GET	POST
Using GET request limited amount of information can be sent.	Using POST large amount of information can be sent.
GET request is not secured as information is visible in URL.	This is a secured request.
This request is can be bookmarked.	This request can not be bookmarked.
This request is more efficient.	This request is less efficient.

How does servlet read form data ?

- Servlet makes use of following three methods to read the data entered by the user on the HTML form
 1. **getParameter()** - You call `request.getParameter()` method to get the value of a form parameter.
 2. **getParameterValues()** - Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
 3. **getParameterNames()** - Call this method if you want a complete list of all parameters in the current request.

Programming Examples

Example 6.15.1 Write a HTML that shows the following list :

C,C++,JAVA,C#

Define a form that contains a select statement and submit button. If the user selects the java and press the submit the web page displays "The selected language is Java".

Write a servlet program using `HttpServlet` and `doGet` method.

Solution : We will write two source files first one is the HTML file named **test.html** in which the list of all the desired languages is displayed along with the submit button. User has to select the language of his choice and then press the submit button.

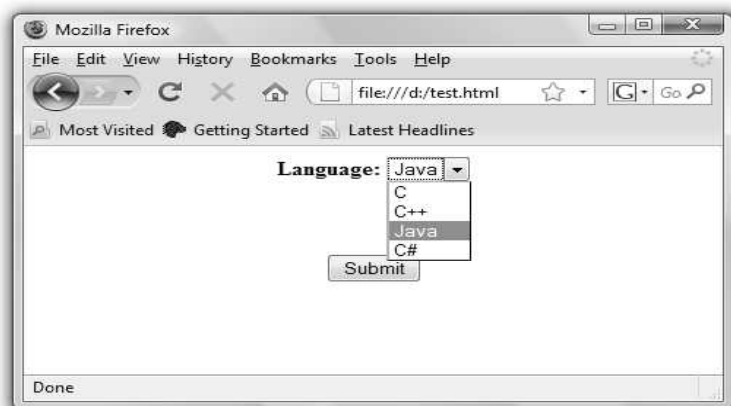
This data made by this request will be received by the servlet named **my_choiceservlet.java** which reads the selected parameter and displays the message about the selection made. The source files are as follows -

test.html

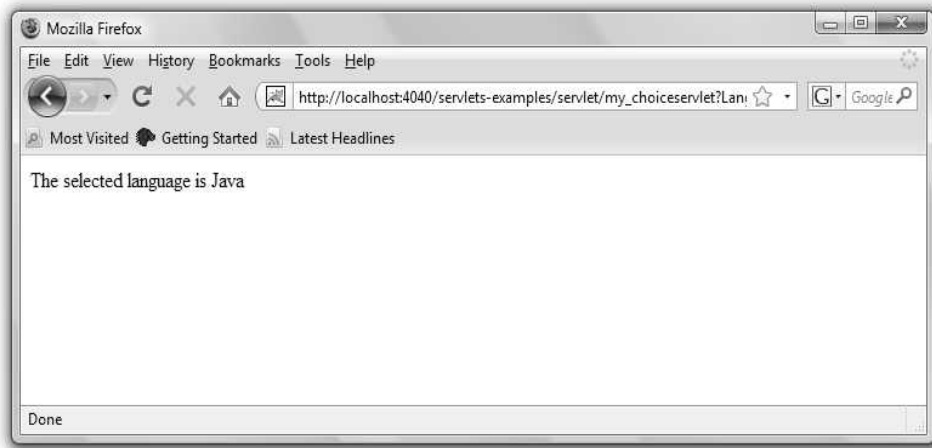
```
<html>
  <body>
    <center>
      <form name="form1" method=GET
action="http://localhost:4040/servlets-examples/servlet/my_choiceservlet">
        <b>Language:</b>
        <select name="Language" size="1">
          <option value="C">C</option>
          <option value="C++">C++</option>
          <option value="Java">Java</option>
          <option value="C#">C#</option>
        </select>
        <br><br>
        <input type="submit" value="Submit">
      </form>
    </body>
  </html>
```

my_choiceservlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class my_choiceservlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
        String lang=req.getParameter("Language");
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("The selected language is "+lang);
        out.close();
    }
}
```



On clicking the submit button we will get following output.



Example 6.15.2 Write HTML form to read user name and password. This data is sent to the servlet. If the correct user name and password is given then welcome him/her by his/her name otherwise display the message for invalid user.

Solution :

Step 1 : Create HTML form for accepting user name and password

Input.html

```
<html>
<head>
</head>
<body>
<form action="http://localhost/examples/servlets/servlet/Welcome" method="get">
User Name:<input type="text" name="uname"/>
<br/>
Password:<input type="password" name="pwd"/>
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```

Step 2 : Create the servlet program to read user name and password and validate it.

Welcome.java

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

public class Welcome extends HttpServlet
{
```

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
{
    PrintWriter out=res.getWriter();
    res.setContentType("text/html");

    String username=req.getParameter("uname");
    String password=req.getParameter("pwd");
    if ((username=="Ankita")&&(password=="1234"))
        out.print("Welcome "+username);
    else
        out.println("Invalid username");
}
}
```

Example 6.15.3 Write a servlet which accept two numbers using POST methods and display the maximum of them.

Solution :

Step 1 : The HTML document for inputting two numbers is as follows -

NumbersInput.html

```
<html>
<head>
<body>
<div align="center">
    <br> <br>
    <form action="http://localhost/examples/servlets/servlet/MaxNumber" method="post">
        Enter First Number :
        <input type="text" value="" name="Number1" size='5'>
        <br/><br/> Enter Second Number :
        <input type="text" value="" name="Number2" size='5'>
        <br/> <br/> <br/>
        <input type="submit" value="Submit">
    </form>
</div>
</body>
</html>
```

Step 2 : The servlet code that handles the Post method and finds the maximum of the two input numbers is as follows -

MaxNumber.java

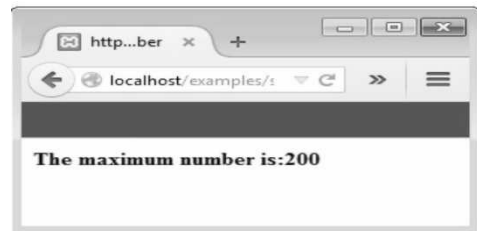
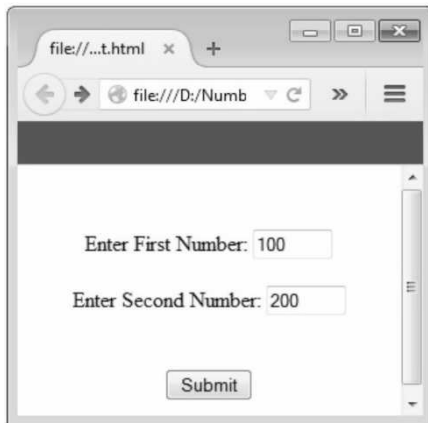
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class MaxNumber extends HttpServlet
{
    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();

        // get request parameters for userID and password
        int a = Integer.parseInt(req.getParameter("Number1"));
        int b = Integer.parseInt(req.getParameter("Number2"));

        if (a>b)
            out.println("<h4>The maximum number is:" +a+"</h4>");
        else
            out.println("<h4>The maximum number is :"+b+"</h4>");
    }
}
```

Step 3 : The output is as follows –



Review Question

1. Differentiate between GET and POST request.

6.16 Multiple Choice Questions

Q.1 Which package contains the classes and interfaces required for Java networking ?

- | | |
|--------------------------------------|---------------------------------------|
| <input type="checkbox"/> a) java.io | <input type="checkbox"/> b) java.util |
| <input type="checkbox"/> c) java.net | <input type="checkbox"/> d) java.awt |

Q.2 Which methods are commonly used in ServerSocket class ?

- a public OutputStream getOutputStream()
- b public Socket accept()
- c public synchronized void close()
- d none of the above

Q.3 Which class is used to create servers that listen for either local client or remote client programs ?

- a ServerSockets
- b httpServer
- c httpResponse
- d none of the above

Q.4 Which methods are commonly used in ServerSocket class ?

- a public OutputStream getOutputStream()
- b public Socket accept()
- c public synchronized void close()
- d none of the above

Q.5 Which of these is a protocol for breaking and sending packets to an address across a network ?

- a TCP/IP
- b DNS
- c Socket
- d Proxy server

Q.6 How many ports of TCP/IP are reserved for specific protocols ?

- a 10
- b 1024
- c 2048
- d 512

Q.7 How many bits are in a single IP address ?

- a 8
- b 16
- c 32
- d 64

Q.8 URL stands for Uniform Resource Locator and represents a resource on the World Wide Web, such as a Web page.

- a True
- b False

Q.9 Which of these class is used to encapsulate IP address and DNS ?

- a DatagramPacket
- b URL
- c InetAddress
- d ContentHandler

Q.10 The DatagramSocket and DatagramPacket classes are not used for connection-less socket programming.

a True

b False

Q.11 Which of these is a full form of DNS ?

a Data Network Service

b Data Name Service

c Domain Network Service

d Domain Name Service

Q.12 What is the output of this program ?

```
import java.net.*;
class networking {
    public static void main(String[] args) throws UnknownHostException {
        InetAddress obj1 = InetAddress.getByName("www.google.com");
        InetAddress obj2 = InetAddress.getByName("www.facebook.com");
        boolean x = obj1.equals(obj2);
        System.out.print(x);
    }
}
```

a 0

b 1

c True

d False

Q.13 The client in socket programming must know which informations ?

a IP address of server

b Port number

c Both a and b

d None of the above

Q.14 Datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

a True

b False

Q.15 What is the output of this program ?

```
import java.net.*;
class networking {
    public static void main(String[] args) throws UnknownHostException {
        InetAddress obj1 = InetAddress.getByName("www.google.com");
        InetAddress obj2 = InetAddress.getByName("www.google.com");
        boolean x = obj1.equals(obj2);
        System.out.print(x);
    }
}
```

a 0

b 1

c True

d False

- Q.16** Port number 80 is reserved for ____ protocol
- a FTP b HTTP
 c SMTP d Telnet
- Q.17** While using the getLocalHost() method the exception ____ is thrown
- a UnknownHostException b NullPointerException
 c LostHostException d IOException
- Q.18** The class _____ is used for accessing the attributes of remote resource.
- a URLconnection b URL
 c URI d none of these
- Q.19** The correct way of using ServerSocket is ____.
- a ServerSocket(int port)
 b ServerSocket(int port, int maxQueue)
 c ServerSocket(int port, int maxQueue, InetAddress localAddress)
 d All of these
- Q.20** Which method of URL class represents a URL and it has complete set of methods to manipulate URL in Java ?
- a java.net.URL b java.net.URLconnection
 c java.net.URI d None of the above
- Q.21** The Java _____ specification defines an application programming interface for communication between the web server and the application program.
- a servlet b randomise
 c applet d script
- Q.22** Which method is used to specify before any lines that uses the PintWriter ?
- a setPageType() b setContextType()
 c setContentType() d setResponseType()
- Q.23** What are the functions of Servlet container ?
- a Lifecycle management b Communication support
 c Multithreading support d All of the above
- Q.24** What is bytecode ?
- a Machine-specific code b Java code
 c Machine-independent code d None of the mentioned

Answer Keys for Multiple Choice Questions :

Q.1	c	Q.2	b	Q.3	a	Q.4	b
Q.5	a	Q.6	b	Q.7	c	Q.8	a
Q.9	c	Q.10	b	Q.11	d	Q.12	d
Q.13	c	Q.14	a	Q.15	c	Q.16	b
Q.17	a	Q.18	a	Q.19	d	Q.20	a
Q.21	a	Q.22	c	Q.23	d	Q.24	c
Q.25	b	Q.26	a	Q.27	c	Q.28	d
Q.29	a	Q.30	b	Q.31	b		



SOLVED MODEL QUESTION PAPER (In Sem)

Advanced JAVA Programming

T.E. (E & Tc) Semester - VI (Elective - II) (As Per 2019 Pattern)

Time : 1 Hour]

[Maximum Marks : 30

N. B. :

- i) Attempt Q.1 or Q.2, Q.3 or Q.4.
- ii) Neat diagrams must be drawn wherever necessary.
- iii) Figures to the right side indicate full marks.
- iv) Assume suitable data, if necessary.

- Q.1** a) Differentiate applet and application with any four points. (Refer section 1.1) [4]
b) Explain any three applet tags. (Refer section 1.3) [3]
c) Explain with suitable example how to create and execute an applet program.
(Refer section 1.4) [8]

OR

- Q.2** a) Explain applet life cycle with suitable diagram. (Refer section 1.2) [5]
b) Explain `getDocumentBase()` and `getCodeBase()` functions with suitable examples
(Refer section 1.6) [4]
c) How can parameters be passed to an applet ? Write an applet to accept user name in the form of parameter and print 'Hello <username >'.
(Refer example 1.5.1) [6]

- Q.3** a) Explain event classes. (Refer section 2.3) [4]
b) What method is used to distinguish between single, double and triple mouse clicks ? Illustrate. (Refer example 2.6.1) [3]
c) Explain event delegation model with some illustrative example.
(Refer section 2.5) [8]

OR

- Q.4** a) What is AWT ? Enlist the limitations of AWT. (Refer sections 2.10 and 2.12) [3]
b) Write a Java program to draw a rectangle using AWT graphics object.
(Refer section 2.17.2) [5]
c) What is mouse event ? Write a Java program to implement the handling of mouse events. (Refer section 2.6) [7]

SOLVED MODEL QUESTION PAPER (End Sem)**Advanced JAVA Programming**

T.E. (E & Tc) Semester - VI (Elective - II) (As Per 2019 Pattern)

Time : 2 $\frac{1}{2}$ Hours]

[Maximum Marks : 70

N. B. :

- i) Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.
- ii) Neat diagrams must be drawn wherever necessary.
- iii) Figures to the right side indicate full marks.
- iv) Assume suitable data, if necessary.

- Q.1** a) Write a JFrame with a hello world program. (Refer example 3.4.1) [5]
- b) Write a program to create product enquiry form using frames.
(Refer example 3.4.3) [8]
- c) Explain JButton class with suitable example. (Refer section 3.4.6) [5]

OR

- Q.2** a) Explain the different types of dialog boxes used in Java. (Refer section 3.4.14) [5]
- b) Write a Java program to demonstrate ArrayList. (Refer section 3.8.1) [5]
- c) With suitable example explain JMenu in Java. (Refer section 3.4.13) [8]
- Q.3** a) Explain JDBC architecture in detail. (Refer section 4.3) [5]
- b) Explain executeQuery() and executeUpdate() method in JDBC.
(Refer section 4.9) [5]
- c) Explain different types of JDBC drivers. (Refer section 4.2) [7]

OR

- Q.4** a) Write a JDBC program to demonstrate CREATE Statement. (Refer section 4.9.2) [5]
- b) Write a short note on - ResultSet interface. (Refer section 4.11) [5]
- c) What is prepared statement ? Explain it in detail with illustrative example.
(Refer section 4.10.1) [7]

- Q.5** a) Explain RMI registry. (Refer section 5.3) [4]
b) What is RMI ? Explain architecture of RMI. (Refer sections 5.1 and 5.2) [8]
c) Write a RMI application in which the client can send a message to the server.
(Refer example 5.12.2) [6]

OR

- Q.6** a) Write short note on - Naming and directory services. (Refer section 5.7) [8]
b) Explain the use of RMI in examination control system in which the server has all the student information and the student objects can be accessed from any client.
(Refer example 5.12.4) [10]
- Q.7** a) Explain networking classes and interfaces. (Refer section 6.1) [6]
b) What is proxy servers ? Explain. (Refer section 6.2.3) [6]
c) Write a Java program to find the IP address of your machine.
(Refer example 6.3.1) [5]

OR

- Q.8** a) What are factory and instance methods ? Explain. (Refer section 6.3) [8]
b) Write a TCP socket programming application in which client sends some message to the server and server sends the acknowledgement to the client.
(Refer example 6.11.2) [9]

□□□

TEXT BOOKS FOR T.E. (E&Tc) SEM VI**Compulsory Subjects**

1. Cellular Networks (*V. S. Bagad*)
2. Project Management (*Rana S. Mahajan, Dr. Dipak P. Patil, Dr. Manoj V. Bhalerao*)
3. Power Devices & Circuits (*Dr. J. S. Chitode, Dr. Shamsundar M. Kulkarni*)

Elective Subjects

4. Digital Image Processing
5. Sensors in Automation
6. Advanced JAVA Programming (*A. A. Puntambekar, Santosh B. Dhekale*)
7. Embedded Processors
8. Network Security (*V. S. Bagad, I. A. Dhotre*)

**FE
SE
TE
BE**For All
Branches**DECODE**[®]

A Guide for Engineering Students

PAPER SOLUTIONS

- Covers Entire Syllabus • Question Answer Format • Exact Answers & Solutions
- Important Points to Remember • Important Formulae
- Chapterwise Solved University Questions • Last 10 Years Solved Papers

... Available at all Leading Booksellers ...