

SRINIVAS UNIVERSITY

INSTITUTE OF COMPUTER & INFORMATION SCIENCE

CITY CAMPUS, PANDESHWAR, MANGALORE - 575 001

BACKGROUND STUDY MATERIAL

ADVANCED JAVA PROGRAMMING

B.C.A V SEMESTER



2023-2024

Compiled by

Faculty

15-06-17 11:35

SYLLABUS

Paper: 21CAC13 Theory/Week: 3 Hrs Credits: 3	Advanced Java Programming	Hours: 30 IA : 50 Exam: 50
Course Objective: <ul style="list-style-type: none"> To train the student in understanding the concepts of advanced Java programming. To create Java programs to connect to the database at the back end To enable the students to create distributed computing applications in Java Course outcome: On the successful completion of this course, student will be able CO1: To recall the concepts of java programming and work with applets and Swings. CO2: To demonstrate database connectivity and build java applications. CO3: To develop distributed applications using remote method invocation		
Module I :		6 Hours
Introduction to Applets and File systems: Applet, building applet code, applet life cycle, applet tag, adding applet to HTML file, running applet, passing parameters to applet, concept of stream, stream classes, byte stream classes, character stream classes, using the File class, I/O exceptions, creation of files, reading/writing characters, reading/writing bytes, reading/writing using random access file Teaching Methodology Chalk and Board Power point Experimental Learning Video Lectures		
Module II:		6 Hours
Using AWT controls, Layout Managers and Menus: Control Fundamentals- Labels, Buttons, CheckBoxes, CheckboxGroup, Choice Controls, Lists, Scroll Bars, TextField, TextArea. Layout Managers- FlowLayout, BorderLayout, GridLayout, Menu Bars and Menus Teaching Methodology Chalk and Board Power point Experimental Learning Video Lectures		

Module III :	6 Hours
Introducing Swing: The Origins and Design Philosophy of Swing, Components and Containers, Layout Managers, Use JButton, Work with JTextField, Create a JCheckBox, Work with Jlist, Use anonymous inner classes to handle events, Create a Swing applet Teaching Methodology: Chalk and Board Power point Experimental Learning Video lectures	
Module IV:	6 Hours
JDBC and RMI: Introduction to JDBC, JDBC Driver, DB Connectivity Steps, Connectivity with MySQL, Access without DSN, DriverManager, Connection, Statement, ResultSet, Introduction to Remote Method Invocation (RMI), Understanding Stub and Skeleton, Architecture of RMI, RMI in server side, RMI in client side, a complete example for RMI, and Meaningful examples of RMI application with database. Teaching Methodology Chalk and Board Power point Experimental Learning	
MODULE V:	6 Hours
Servlets: Introduction to Servlets-Web application, Advantages of Servlets, Life Cycle of a Servlet, Using Tomcat for Servlet Development, The Servlet API, Handling HTTP request and Response, Using Cookies, Session tracking, Database Access-create table, insert records and access table. Web interface for Servlets Teaching Methodology: Chalk and Board. Blended Learning Experimental Learning Power point Seminars	

Text Books	
1	Herbert Schildt, <i>Java - The Complete Reference – 7th Edition</i> , Tata McGraw Hill.
2	Jim Keogh, <i>J2EE - The Complete Reference</i> , Tata McGraw Hill.

Reference Books	
1	A. Lew, H. Mauch, <i>Dynamic Programming - A Computational tool</i> , Springer
2	K. Somasundaram, <i>Introduction to Java Programming</i> , Jaico Publishing House

Teaching Plan

Total Hours : 40

Internal Marks: 50

Hours/Week: 4 hours

External Marks: 50

Course Objective: The objective here is to provide fundamental knowledge and practical exposure to the object oriented programming concepts in Java. It includes study of information concepts and the realization of those advanced concepts using Java. Practical Knowledge is gained through creating programs in Java environment

Learning Outcome: Upon successful completion of the course, the students will be able to

- Perform Applet Programming
- Work with Files in Java
- Work with Swing Components
- JDBC connectivity in Java
- RMI programming

UNIT-1

Chapter 1: Introducing Applet and File System

8Hrs

Session 1: Introduction about Applet, Building applet code

Session 2: Applet life cycle, Applet tag.

Session 3: Adding applet to HTML file, Running Applet

Session 4: Passing parameters to Applets with example

Chapter 2: Managing Input / Output Files in Java

Session 5: Introduction , Concept of Streams, Stream classes , byte stream classes and character stream classes

Session 6: Creating files using the File Class, I/O Exceptions

Session 7: Creation of files, Explanation about Reading /Writing characters

Session 8: Reading/Writing bytes, Reading/ Writing using Random access file

UNIT-2

Chapter 3: AWT Controls

8hrs

Session 1: Control Fundamentals, adding and removing controls, Responding to controls, Headless Exception

Session 2: Labels, Buttons, Handling Labels and Buttons

Session 3: Checkbox , Checkbox Group, Handling Checkbox and Checkbox group

Session 4: Choice controls, Handling Choice Lists, and List, Handling Lists

Session 5: Managing Scrollbars, Handling Scrollbars

Session 6: TextField, Handling a TextField, TextArea

Chapter 4: Layout Managers and Menus

Session 7: Layout Managers- FlowLayout, BorderLayout, GridLayout

Session 8: Menu Bars and Menus**UNIT-3****Chapter 5: Introducing Swing****8hrs**

Session 1: The origins and Design Philosophy of Swing

Session 2: Explanation about Components and Containers

Session 3: Layout Managers, A first Simple Swing program, swing example line by line

Session 4: Use JButton, Work with JTextField

Session 5: Create a JCheckBox

Session 6: Working with JList

Session 7: Use anonymous inner classes to handle events

Session 8: Create a Swing Applet

UNIT-4**Chapter 6: JDBC****8hrs**

Session 1: Introduction to JDBC

Session 2: JDBC Driver

Session 3: DB Connectivity Steps

Session 4: Connectivity with MySQL

Session 5: Connectivity with Access without DSN

Session 6: Driver Manager, Connection Interface

Session 7: Statement Interface, Example of Statement Interface

Session 8: ResultSet Interface

UNIT-5**Chapter 7: RMI****8hrs**

Session 1: Introduction to Remote Method Invocation(RMI)

Session 2: Understanding Stub and Skeleton

Session 3: Explanation about Architecture of RMI

Session 4: RMI in server side, RMI in client side

Session 5: A complete example for RMI

Session 6: Examples of RMI application with database

Session 7: Programs related to RMI

Session 8: Unit Test

Reference Books :

1. Java - The Complete Reference – Herbert Schildt, 7th Edition, Tata McGraw Hill.
2. J2EE - The Complete Reference – Jim Keogh, Tata McGraw Hill.

UNIT-1

Chapter-1

Introducing Applet

Applet Basics

Applets are small programs that are designed for transmission over the Internet and run within a browser. Because Java's virtual machine is in charge of executing all Java programs, including applets, applets offer a secure way to dynamically download and execute programs over the Web.

There are two general varieties of applets: those based on the Abstract Window Toolkit (AWT) and those based on Swing. Both the AWT and Swing support the creation of a graphical user interface (GUI). The AWT is the original GUI toolkit and Swing is Java's lightweight alternative. It is important to understand, however, that Swing-based applets are built upon the same basic architecture as AWT-based applets. Furthermore, Swing is built on top of the AWT. Therefore, the information and techniques presented here describe the foundation of applet programming and most of it applies to both types of applets.

// A Minimal AWT-based applet.

```
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet {
    public void paint (Graphics g) {
        g.drawString ("—Java makes applets easy. —", 20, 20);
    }
}
```

This applet begins with two import statements. The first imports the Abstract Window Toolkit classes. Applets interact with the user (either directly or indirectly) through the AWT, not through the console-based I/O classes. The AWT contains support for a window-based, graphical user interface. The next import statement imports the applet package. This package contains the class Applet. Every applet that you create must be a subclass (either directly or indirectly) of Applet. The next line in the program declares the class SimpleApplet.

This class must be declared as public because it will be accessed by outside code. Inside SimpleApplet, paint() is declared. This method is defined by the AWT Component class (which is a superclass of Applet) and is overridden by the applet. paint() is called each time the applet must redisplay its output. This can occur for several reasons. For example, the window in which the applet is running can be overwritten by another window and then uncovered. Or the applet window can be minimized and then restored. paint() is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint() is called. The paint() method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

Inside paint(), there is a call to drawString(), which is a member of the Graphics class.

This

method outputs a string beginning at the specified X, Y location. It has the following general form:

```
void drawString(String message, int x, int y)
```

Here, message is the string to be output beginning at x, y. In a Java window, the upper-left corner is location 0,0. The call to drawString() in the applet causes the message to be displayed beginning at location 20,20.

The applet does not have a main() method. Unlike the standalone application program, applets do not begin execution at main(). In fact, most applets don't even have a main() method. Instead, an applet begins execution when the name of its class is passed to a browser or other applet-enabled program.

Compiling applet program is similar to stand alone application program. However, running SimpleApplet involves a different process. There are two ways in which you can run an applet: inside a browser or with a special development tool that displays applets. The tool provided with the standard Java JDK is called appletviewer use it to run the applets. We can also run them in your browser, but the appletviewer is much easier to use during development.

One way to execute an applet (in either a Web browser or the appletviewer) is to write a short HTML text file that contains a tag that loads the applet. Currently, Oracle recommends using the APPLET tag for this purpose.

```
<applet code = —simple applet — width=200 height=60>  
</applet>
```

The width and height statements specify the dimensions of the display area used by the applet. To execute SimpleApplet with an applet viewer, you will execute this HTML file. For example, if the preceding HTML file is called StartApp.html, then the following command line will run SimpleApplet:

```
C:\> appletviewer StartApp.html
```

1.1.2 Preparing to Write Applets

Before we try to write applets, we must make sure that Java is installed properly and also ensure that either the Java **appletviewer** or a Java-enabled browser is available. The steps involved in developing and testing in applet are:

1. Building an applet code(.java file)
2. Creating an executable applet(.class file)
3. Designing a Web page using HTML tags
4. Preparing <APPLET> tag
5. Incorporating <APPLET> tag into the Web page
6. Creating HTML file
7. Testing the applet code

Building Applet Code

Applet code uses the services of two classes, namely, Applet and Graphics from the Java class library. The Applet class which is contained in the java.applet package provides life and behaviour to the applet through its methods such as init(), start() and point(). Java calls the main() method directly to initiate the execution of the program. When an applet is loaded, Java automatically calls a series of Applet class methods for starting, running, and stopping the applet code.

The paint() method of the Applet class, when it is called, actually displays the result of the applet code on the screen. The output may be text, graphics, or sound. The paint() method, which requires a Graphics object as an argument, is defined as follows.

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

This requires that the applet code imports the java.awt package that contains the Graphics class. All output operations of an applet are performed using the methods defined in the Graphics class.

Syntax:

```
Import java.awt.*;
Import java.applet.*;
.....
.....

Public class appletclassname extends Applet
{
.....
Public void paint(Graphics g)
{
.....
.....
}
.....
.....
}
```

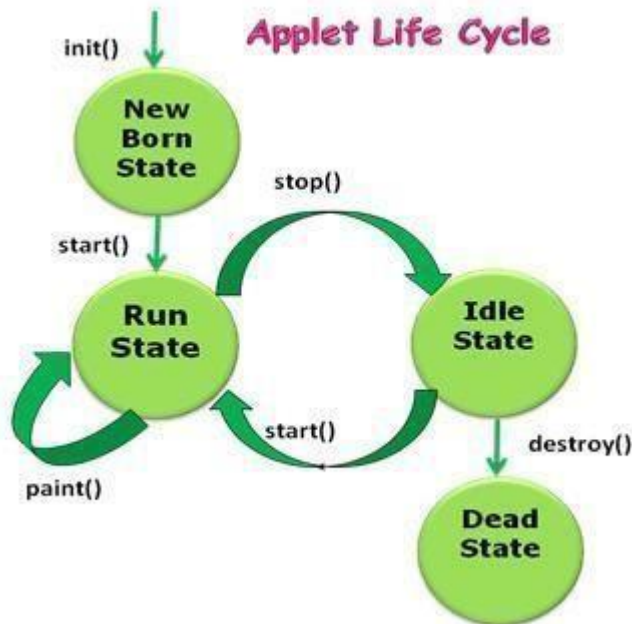
The appletclassname is the main class for the applet. When the applet is loaded, Java creates an instance of this class, and then a series of Applet class methods are called on that instance to execute the code.

Example:

```
Import java.awt.*;
Import java.applet.*;
Public class HelloJava extends Applet
{
    Public void paint(Graphics g)
    {
        g.drawString("Hello Java", 10,100);
    }
}
```

Applet Life Cycle

When an applet is loaded, it undergoes a series of changes in its state as shown in fig.



- Born on initialization state
- Running state
- Idle state
- Dead or destroyed state

Initialization State

Applet enters the initialization state when it is first loaded. This is achieved by calling the `init()` method of Applet class. The applet is born. At this stage, we may do the following.

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

```

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

Running State

Applet enters the running state when the system calls the `start()` method of Applet class. This occurs automatically after the applet is initialized.

```

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

Idle or stopped state

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the `stop()` method.

```

public void stop()
{
  
```

```

.....
.....
}

```

Dead State

An applet is said to be dead when it is removed from memory. This occurs by invoking the `destroy()` method when we quit the browser. Destroying stage occurs only once in the applet's life cycle.

```

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

```

Display State

Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The `paint()` method is called to accomplish this task.

```

public void paint(Graphics g)
{
.....
.....(Display statements)
}

```

The **paint()** method is defined in the Applet class.

Applet Tag

We have included a pair of `<APPLET>` and `</APPLET>` tags in the body section. The `<APPLET...>` tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The ellipsis in the tag `<APPLET..>` indicates that it contains certain attributes that must be specified.

```

<APPLET
  CODE=helloJava.class
  WIDTH=400
  HEIGHT=200>
</APPLET>

```

This HTML code tells the browser to load the compiled Java applet **HelloJava.class**, which is in the same directory as the HTML file. And also specifies the display area for the applet output as 400 pixels width and 200 pixels height. We can make this display area appear in the centre of the screen by using the `CENTER` tags shown as follows.

```

<CENTER>
  <APPLET
    .....
    .....
  </APPLET>
</CENTER>

```

Note that `<APPLET>` tag discussed above specifies three things.

1. Name of the applet
2. Width of the applet(in pixels)

3.
of the applet(in pixels)

Height

Adding Applet to HTML File

We can put together the various components of the Web page and create a file known as HTML file. Insert the <APPLET> tag in the page at the place where the output of the applet must appear. Following is the content of the HTML file that is embedded with the <APPLET> tag of our HelloJava applet.

<HTML>

<! It specifies the applet to be loaded and executed.

>

<HEAD>

<TITLE>

Welcome to Java

</TITLE>

</HEAD>

<BODY>

<CENTER>

<H1> Welcome to the world of Applet </H1>

</CENTER>

<CENTER>

Passing parameters to Applet

We can supply user-defined parameters to an applet using <PARAM ... > tags. Each <PARAM... > tag has a name attribute such as color, and a value attribute such as red. Inside the applet code, the applet can refer to that parameter by name to find its value. For example, we can change the colour of the text displayed to red by an applet by using a <PARAM... > tag as follows.

<APPLET >

<PARAM = color VALUE = "red" >

</APPLET>

Similarly, we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM....>tag as shown below:

<PARAM NAME = text VALUE = "I love Java" >

Passing parameters to an applet code using <PARAM>tag is something similar to passing parameters to the main() method using command line arguments. To set up and handle parameters, we need to do two things:

1. include appropriate <PARAM> tags in the HTML document.
2. Provide Code in the applet to parse these parameters.

Parameters are passed to an applet when it is loaded. We can define the init() method in the applet to get hold of the parameters defined in the <PARAM> tags. This is done using the getParameter() method., which takes one string argument representing the name of the parameter and returns a string containing the value of that parameter.

Applet HellojavaParm

```

import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
String str;
public void init( )
{
str = getParameter("string");    //Receiving parameter value
if (str == null)
    str="Java";
    str= "Hello"+ str           //Using the value
}
public void paint (Graphics g)
{
g.drawString(str, 10, 100);
}
}

```

Now we have to create HTML file that contains this applet. below program shows a web page that passes a parameter whose name is "string" and whose VALUE is "Applet!" to the applet HelloJavaParam.

The HTML file for HelloJavaParam applet

```

<HTML>
<!-- Parameterized HTML file -->
<HEAD>
<TITLE> Welcome to Java Applets </TITLE>
</HEAD>
<BODY>
<APPLET CODE = HelloJavaParam.class
WIDTH = 400
HEIGHT = 200>
<PARAM NAME = "string"
VALUE = "Applet!" >
</APPLET>
</BODY>
</HTML>

```

This will produce output as Hello Applet. if we remove <param> tag from HTML file will produce output as Hello Java.

CHAPTER 2

MANAGING INPUT/OUTPUT FILES IN JAVA

Introduction

A file is a collection of related records placed in a particular area on the disk. A record is composed of several fields and a field is a group of characters. Characters in java are Unicode characters composed of two bytes, each byte containing eight binary digits, 1 or 0.

Storing and managing data using files is known as file processing which includes tasks such as creating files, updating files and manipulation of data.

Concept of Streams

In file processing, input refers to the flow of data into a program and output means the flow of data out of a program. Input to a program may come from the keyboard, the mouse, the memory, the disk, a network, or another program. Similarly, output from a program may go to the screen, the printer, the memory, the disk, a network, or another program.

The Stream Classes

Java's stream-based I/O is built upon four abstract classes: **InputStream**, **OutputStream**, **Reader**, and **Writer**. These classes were briefly discussed in Chapter 13. They are used to create several concrete stream subclasses. Although your programs perform their I/O operations through concrete subclasses, the top-level classes define the basic functionality common to all stream classes.

InputStream and **OutputStream** are designed for byte streams. **Reader** and **Writer** are designed for character streams. The byte stream classes and the character stream classes form separate hierarchies. In general, you should use the character stream classes when working with characters or strings, and use the byte stream classes when working with bytes or other binary objects.

The Byte Streams

The byte stream classes provide a rich environment for handling byte-oriented I/O. A byte stream can be used with any type of object, including binary data. This versatility makes byte streams important to many types of programs. Since the byte stream classes are topped by **InputStream** and **OutputStream**, our discussion will begin with them.

InputStream

InputStream is an abstract class that defines Java's model of streaming byte input. It implements the **Closeable** interface. Most of the methods in this class will throw an **IOException** on error conditions. (The exceptions are **mark()** and **markSupported().**) Table 19-1 shows the methods in **InputStream**.

OutputStream

OutputStream is an abstract class that defines streaming byte output. It implements the **Closeable** and **Flushable** interfaces. Most of the methods in this class return **void** and throw an **IOException** in the case of errors. (The exceptions are **mark()** and **markSupported().**)

The below table gives a brief description of all the methods provided by the **InputStream** class

Method	Description
1. Read()	Reads a byte from the input stream
2. Read(byte b[])	Reads an array of bytes into b
3. Read(byte b[], int n, int m)	Reads m bytes into b starting from nth byte
4. Available()	Gives number of bytes available in the input
5. Skip(n)	skips over n bytes from the input stream
6. Reset()	Goes back to the beginning of the stream
7. Close	closes the input stream

Table gives a brief description of all the methods of outputstream class

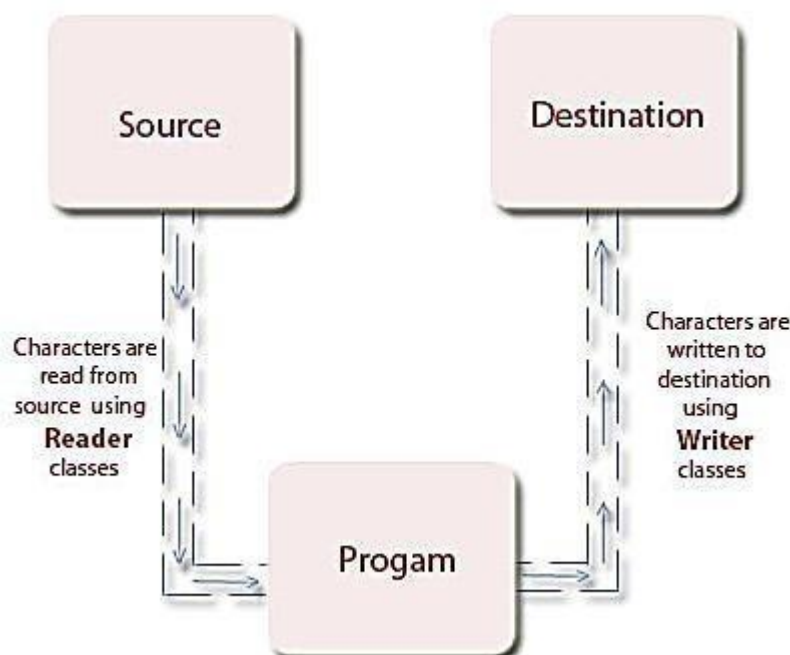
Method	Description
1. Write()	Writes a byte to the output stream
2. write(byte[] b)	Writes m bytes in the array b to the output stream
3. write(byte b[], int n, int m)	Writes m bytes from array b starting from nth byte
4. close()	closes the output stream
5. flush()	flushes the output stream

Character Stream Classes

Character Stream Classes are used to read characters from the source and write **characters** to destination.

There are two kinds of Character Stream classes - Reader classes and Writer classes.

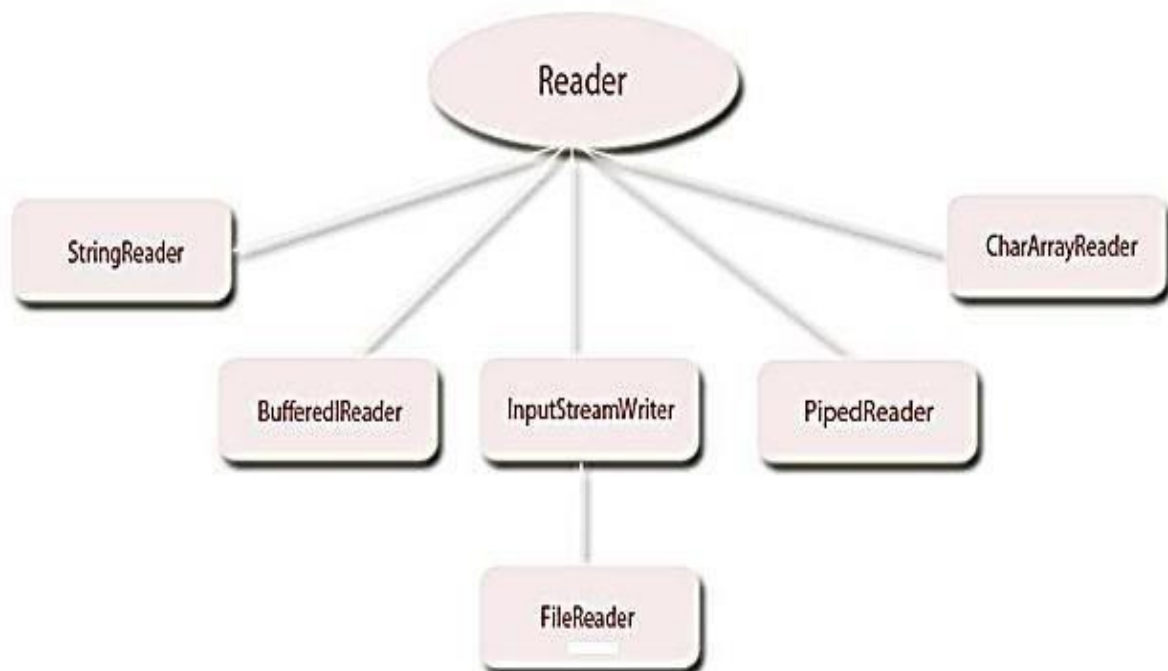
- **Reader Classes** - These classes are subclasses of an abstract class, Reader and they are used to read characters from a source(file, memory or console).
- **Writer Classes** - These classes are subclasses of an abstract class, Writer and they used to write characters to a destination(file, memory or console).



Reader

Reader class and its subclasses are used to read characters from source. Reader class is a base class of all the classes that are used to read characters from a file,

memory or console. Reader is an abstract class and hence we can't instantiate it but we can use its subclasses for reading characters from the input stream. We will discuss subclasses of Reader class with their code, in the next few articles.



Hierarchy of reader stream class

Methods of Reader classes

These methods are of Reader class.

Methods	Description
Int read()	This method reads a character from the input stream
Int read(char[], ch)	This method reads a chunk of characters from the input stream and store them in its char array, ch.
close()	This method closes this output stream and also frees any system resources connected with it.

Writer

Writer class and its subclasses are used to write characters to a file, memory or console. Writer is an abstract class and hence we can't create its object but we can use its subclasses for writing characters to the output stream. We will discuss subclasses of Writer with their code in the next few articles.



Hierarchy of Writer classes

Methods of writer classes

Methods of Writer class provide support for writing characters to the output stream. As this is an abstract class. Hence, some undefined abstract methods are defined in this subclasses of Output Stream.

Methods	Description
abstract void flush()	This method flushes the output stream by forcing out buffered bytes to be written out.
void write(int c)	This method writes a character (contained in an int) to the output stream.
void write(char[] arr)	This method writes a whole char array (arr) to the output stream.
abstract void close()	This method closes this output stream and also frees any resources connected with this output stream.

Using the File Class

The java.io package includes a class known as the **File** class that provides support for creating files and directories. The class includes several constructors for instantiating the File objects. This class also contains several methods for supporting the operations such as

- Creating a file
- Opening a file
- Closing a file
- Deleting a file
- Getting the name of a file
- Getting the size of a file
- Checking the existence of a file
- Renaming a file
- Checking whether the file is writable
- Checking whether the file is readable

Input/output Exceptions

When creating files and performing i/o operations on them, the system may generate i/o related exception.

The basic i/o related exception classes and their functions are given below.

EOFException	Signals that an end of the file or end of stream has been reached unexpectedly during input.
FileNotFoundException	Informs that a file could not be found
InterruptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred

Each i/o statement or group of i/o statements must have an exception handler around it as shown below.

```
try
{
.....
.....//I/O statements
}
Catch(IOException e)
{
.....
.....//message output statement
}
```

Creation of Files

If we want to create and use a disk file, we need to decide the following about the file and its intended purpose.

- Suitable name for the file
- Data type to be stored
- Purpose(reading, writing, or updating)
- Method of creating the file

A filename is a unique string of characters that helps identify a file on the disk.

Example: input.data or input.txt

Data type is to decide the type of file stream classes to be used for handling the data. We should decide whether the data to be handled is in the form of characters, bytes or primitive type.

The purpose of using a file must also be decided before using it. For example, we should know whether the file is created for reading only, or writing only, or both the operations.

For using a file, it must be opened first. This is done by creating a file stream and then linking it to the filename. A file stream can be defined using the classes of **Reader/InputStream** for reading data and **Writer/OutputStream** for writing data.

Common Stream Classes used for I/O Operations

Characters

Read	Write
CharArrayReader	CharArrayWriter
FileReader	FileWriter
PipedReader	PipedWriter

Bytes

Read	Write
ByteArrayInputStream	ByteArrayOutputStream
FileInputStream	FileOutputStream
PipedInputStream	PipedOutputStream

Initializing using a file object

There are two ways of initializing the file stream objects. All of the constructors require that we provide the name of the file either directly, or indirectly by giving a file object that has already been assigned a filename.

The following code segment illustrates the use of **direct** approach.

```
FileInputStream fis;
try
{
    // Assign the filename to the file stream object
    fis=new FileInputStream("test.dat");
    .....
}
Catch(IOException e){ }
```

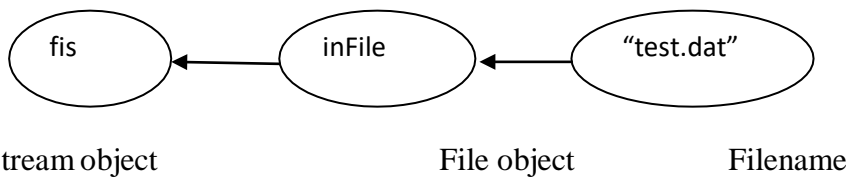
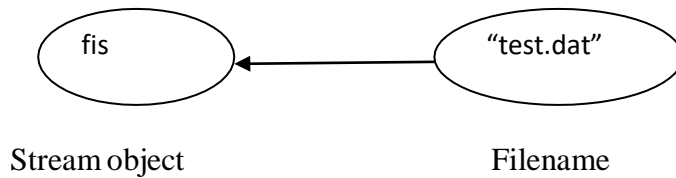
The **indirect** approach uses a file object that has been initialized with the desired filename. This is illustrated by the following code.

```
File infile;                //Declare a file object
infile=new File("test.dat"); //Assign the filename to the file object
FileInputStream fis;
try
{
    //Give the value of the file object
    //to the file stream object
    fis=new FileInputStream(infile);
}
Catch(.....)
{ }
```

The code above includes five tasks:

- Select a filename
- Declare a file object

- Give the selected name to the file object declared
- Declare a file name object
- Connect the file to the file stream object



Reading/Writing Characters

The two subclasses used for handling characters in files are `FileReader`(for reading characters) and `FileWriter`(for writing characters).

The below program uses these two file stream classes to copy the contents of a file named —input.dat into a file called —output.dat.

Import java.io.*;

Class CopyCharacters

```

{
    Public static void main(String args[])
    {
        //Declare and create input and output files
        File inFile=new File(—input.dat);
        File outFile=new File(—output.dat);
        FileReader ins=null; //Creates file stream ins
        FileWriter outs=null; //Creates file stream outs
        try
        {
            ins=new FileReader(inFile); //opens inFile
            outs=new FileWriter(outFile); //Opens outFile
            //Read and write till the end
            int ch;
            While((ch=ins.read())!=-1)
            {
                Outs.write(ch);
            }
        }
        Catch(IOException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
        Finally //close files
    }
}
  
```

```
{
    try
    {
        Ins.close();
        Outs.close();
    }
    Catch(IOException e){ }
}
}
```

Reading or writing Bytes

FileReader and File Writer classes to read and write 16-bit characters. Most file systems use only 8-bit bytes.

Two commonly used classes for handling bytes are FileInputStream and FileOutputStream classes.

Writing Bytes to a file

```
import java. io. *;
class WriteBytes
{
    Public static void main(String args[])
    {
        Byte cities []={_D','E','L','H','I'};
        FileOutputStream outfile=null;
        try
        {
            outfile=new FileOutputStream(-city.txtll);
            outfile.write(cities);
            outfile.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
    }
}
```

Reading bytes from file

```
import java.io.*;
class ReadBytes
{
    Public static void main(String args[])
    {
        FileInputStream infile=null;
        int b;
        try
```

```

        {
            infile=new FileInputStream(args[0]);
            while((b=infile.read())!=-1)
            {
                System.out.println((char)b);
            }
            infile.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
}

```

Random Access File

RandomAccessFile class supported by the java.io package allows us to create files that can be used for reading and writing data with random access.

Ie, we can -jump around in the file while using the file. Such files are known as random access files.

A file can be created and opened for random access by giving a mode string as a parameter to the constructor when we open the file.

We can use one of the following two mode strings.

- -r for reading only
- -rw for both reading and writing

An existing file can be updated using the -rw mode.

Random access files support a pointer known as file pointer that can be moved to arbitrary positions in the file prior to reading or writing.

The file pointer is moved using the method **seek()** in the RandomAccessFile class.

When the file is opened by the statement

File =new RandomAccessFile("rand.dat");

The file pointer is automatically positioned at the beginning of the file.

Reading/Writing using a random access file

Import java.io.*;

Class Random

```

{
    Public static void main(String args[])
    {
        RandomAccessFile file=null;
        try
        {
            file=new RandomAccessFile("-rand.dat", "-rw");
            file.writeChar('_X');
            file.writeInt(555);
            file.writeDouble(3.1412);
            file.seek(0);
            System.out.println(file.readChar());
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
    }
}

```

```
        System.out.println(file.readInt());
        System.out.println(file.readDouble());
        File.seek(2);
        System.out.println(file.readInt());
        file.seek(file.length());
        filewriteBoolean(false);
        file.seek(4);
        System.out.println(file.readBoolean());
        file.close();
    }
    Catch(IOException e)
    {
        System.out.println(e);
    }
}
```

Assignment 1

1. Which of these functions is called to display the output of an applet?

- A. display()
- B. paint()
- C. displayApplet()
- D. PrintApplet()

Answer: Option B

2. Which of these methods can be used to output a string in an applet?

- A. display()
- B. print()
- C. drawString()
- D. transient()

Answer: Option C

3. What does AWT stand for?

- A. All Window Tools
- B. All Writing Tools
- C. Abstract Window Toolkit
- D. Abstract Writing Toolkit

Answer: Option C

4. Which of these is used to perform all input & output operations in Java?

- A. streams
- B. Variables
- C. classes
- D. Methods

Answer: Option A

5. Which of these is a type of stream in Java?

- A. Integer stream
- B. Short stream
- C. Byte stream
- D. Long stream

Answer: Option C

6. Which of these classes are used by Byte streams for input and output operation?

- A. InputStream
- B. OutputStream
- C. Reader
- D. All of the mentioned

Answer: Option A

7. Which of these classes are used by character streams for input and output operations?

- A. InputStream
- B. Writer
- C. ReadStream
- D. InputOutputStream

Answer: Option B

8. Which exception is thrown by read() method?

- A. IOException
- B. InterruptedException
- C. SystemException
- D. SystemInputException

Answer: Option A

9. Which of these packages contain classes and interfaces used for input & output operations of a program?

- A. java.util
- B. java.lang
- C. java.io
- D. All of the mentioned

Answer: Option C

10. Which of these class is not a member class of java.io package?

- A. String
- B. StringReader
- C. Writer
- D. File

Answer: Option A

11. Which of these class is not related to input and output stream in terms of functioning?

- A. File
- B. Writer
- C. InputStream
- D. Reader

Answer: Option A

12. The package contains a large number of stream classes that provide capabilities for processing all types of data.

- A) java.awt
- B) java.io
- C) java.util
- D) java.net

Answer: Option B

13. Which of the following method(s) not included in InputStream class.

- A) available()
- B) reset()
- C) flush()
- D) close()

Answer: Option C

14. Which of the following methods not included in OutputStream class.

- A) write()
- B) skip()
- C) close()
- D) flush()

Answer: Option B

15. The DataInputStream and DataOutputStream classes are.....streams that allow the reading and writing of java primitive data types.

- A) file
- B) sequence
- C) object
- D) filter

Answer: Option D

16. Theclass provides the capacity to read primitive data types from an input stream.

- A) pushbackInputStream
- B) DataInputStream
- C) BufferedInputStream
- D) PipeInputStream

Answer: Option B

17. DataInput is

- A) an abstract class defined in java.io
- B) a class we can use to read primitive data types
- C) an interface that defines methods to open files
- D) an interface that defines methods to read primitives data types

Answer: Option D

18. The following example shows the creation of a

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

```
public class Main extends Applet{  
    public void paint(Graphics g){  
        g.drawString("Welcome in Java Applet.",40,20);  
    }  
}
```

A. Banner Using Applet

- B. Basic Applet
- C. Display Clock
- D. None Of The Above

Answer: Option B

19. An applet can play an audio file represented by the AudioClip interface in the java, applet package Causes the audio clip to replay continually in which method?

- A. Public Void Play()
- B. Public Void Loop()
- C. Public Void Stop()
- D. None Of The Above

Answer: Option B

20. What invokes immediately after the start() method and also any time the applet needs to repaint itself in the browser?

- A. Stop()
- B. Init()
- C. Paint()
- D. Destroy()

Answer: Option C

21. Which method is called only once during the run time of your applet?

- A. stop()
- B. paint()
- C. init()
- D. destroy()

Answer: Option C

22. When an applet is terminated which of the following sequence of methods calls take place?

- A. stop(),paint(),destroy()
- B. destroy(),stop(),paint()
- C. destroy(),stop()
- D. stop(),destroy()

Answer: Option D

23. Which is a special type of program that is embedded in the webpage to generate the dynamic content?

- A. Package
- B. Applet
- C. Browser
- D. None Of The Above

Answer: Option B

24. What is used to run an Applet?

- A. An Html File
- B. An Appletviewer Tool(For Testing Purpose)
- C. Both A & B
- D. None Of The Above

Answer: Option C

25. Which is the correct order of lifecycle in an applet?
- A. Applet Is Started,Initialized,Painted,Destroyed,Stopped
 - B. Applet Is Painted,Started,Stopped,Initilaized,Destroyed
 - C. Applet Is Initialized,Started,Painted,Stopped,Destroyed
 - D. None Of The Above

Answer: Option C

Long Answer questions

1. Explain the life cycle of Applet with neat diagram.
2. How to build the applet code along with the syntax.
3. How to pass the parameters to Applet. Explain with example.
4. Explain about Input/output Exception.
5. Explain about Random Access File along with example.
6. Write a note on reading and writing bytes.
7. Write a note on reading and writing characters.
8. What are the procedures to create the file.
9. Explain about character stream classes.
10. Write all the methods of InputStream and OutputStream classes

UNIT-2

USING AWT CONTROLS, LAYOUT MANAGERS AND MENUS

Control Fundamentals

Controls are components that allow a user to interact with your application in various ways for example, a commonly used control is the push button. A layout manager automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them. The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

These controls are subclasses of Component.

Adding and Removing Controls: To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling `add()`, which is defined by Container. The `add()` method has several forms. The following form is the one that is used for the first part of this chapter:

Component add(Component compObj)

Here, `compObj` is an instance of the control that you want to add. A reference to `compObj` is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call `remove()`. This method is also defined by Container. It has this general form:

void remove(Component obj)

Here, `obj` is a reference to the control you want to remove. We can remove all controls by calling **`removeAll()`**.

Responding to Controls: Except for labels, which are passive, all controls generate events when they are accessed by the user. For example, when the user clicks on a push button, an event is sent that identifies the push button. In general, a program simply implements the appropriate interface and then registers an event listener for each control that we need to monitor.

The HeadlessException: Most of the AWT controls have constructors that can throw a `HeadlessException` when an attempt is made to instantiate a GUI component in a non-interactive environment (such as one in which no display, mouse, or keyboard is present).

Labels

The easiest control to use is a label. A label is an object of type `Label`, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user. `Label` defines the following constructors:

`Label()` throws `HeadlessException`

`Label(String str)` throws `HeadlessException`

`Label(String str, int how)` throws `HeadlessException`

The first version creates a blank label. The second version creates a label that contains the string specified by `str`. This string is left-justified. The third version creates a label that contains the string specified by `str` using the alignment specified by `how`. The value of `how` must be one of these three constants: **`Label.LEFT`**, **`Label.RIGHT`**, or **`Label.CENTER`**. You can set or change the text in a label by using the `setText()` method. You can obtain the current label by calling `getText()`. These methods are shown here:

`void setText(String str)`

`String getText()`

For `setText()`, `str` specifies the new label. For `getText()`, the current label is returned. You can set the alignment of the string within the label by calling `setAlignment()`. To obtain the current alignment, call `getAlignment()`. The methods are as follows:

`void setAlignment(int how)`

`int getAlignment()`

Here, *how* must be one of the alignment constants. The following example creates three labels and adds them to an applet window.

// Demonstrate Labels

```
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=200>
</applet>
*/
public class LabelDemo extends Applet {
    public void init() {
        Label one = new Label("One");
        Label two = new Label("Two");
        Label three = new Label("Three");
        // add labels to applet window
        add(one);
        add(two);
        add(three);
    }
}
```

The labels are organized in the window by the default layout manager.

Buttons

Perhaps the most widely used control is the push button. A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type `Button`. `Button` defines these two constructors:

`Button()` throws `HeadlessException`

`Button(String str)` throws `HeadlessException`

The first version creates an empty button. The second creates a button that contains `str` as a label. After a button has been created, you can set its label by calling `setLabel()`. You can retrieve its label by calling `getLabel()`. These methods are as follows:

`void setLabel(String str)`

`String getLabel()`

Here, `str` becomes the new label for the button.

Handling Buttons: Each time a button is pressed, an action event is generated. This is sent to any listeners that previously registered an interest in receiving action event notifications from that component. Each listener implements the `ActionListener` interface. That interface defines the `actionPerformed()` method, which is called when an event occurs. An `ActionEvent` object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the action command string associated with the button. By default, the action command string is the label of the button. Usually, either the button reference or the action command string can be used to identify the button.

Here is an example that creates three buttons labeled —Yes!, —No!, and —Undecided!. Each time one is pressed, a message is displayed that reports which button has been pressed. In this version, the action command of the button (which, by default, is its label) is used to determine which button has been pressed. The label is obtained by calling the `getActionCommand()` method on the `ActionEvent` object passed to **`actionPerformed()`**.

// Demonstrate Buttons

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

```

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

```

Check Boxes

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

Checkbox() throws `HeadlessException`

Checkbox(String str) throws `HeadlessException`

Checkbox(String str, boolean on) throws `HeadlessException`

`Checkbox(String str, boolean on, CheckboxGroup cbGroup)` throws `HeadlessException`

`Checkbox(String str, CheckboxGroup cbGroup, boolean on)` throws `HeadlessException`

The first form creates a check box whose label is initially blank. The state of the check box is unchecked. The second form creates a check box whose label is specified by `str`. The state of the check box is unchecked. The third form allows you to set the initial state of the check box. If `on` is true, the check box is initially checked; otherwise, it is cleared. The fourth and fifth forms create a check box whose label is specified by `str` and whose group is specified by `cbGroup`. If this check box is not part of a group, then `cbGroup` must be null. The value of `on` determines the initial state of the check box.

To retrieve the current state of a check box, call `getState()`. To set its state, call `setState()`. You can obtain the current label associated with a check box by calling `getLabel()`. To set the label, call `setLabel()`. These methods are as follows:

boolean getState()

void setState(boolean on)

String getLabel()

void setLabel(String str)

Here, if `on` is true, the box is checked. If it is false, the box is cleared. The string passed in `str` becomes the new label associated with the invoking check box.

Handling Checkboxes: Each time a check box is selected or deselected, an item event is generated. This is sent to any listeners that previously registered an interest in receiving item event notifications from that component. Each listener implements the `ItemListener` interface. That interface defines the **`itemStateChanged()`** method. An `ItemEvent` object is supplied as the argument to this method. It contains information about the event (for example, whether it was a selection or deselection)

The following program creates four check boxes. The initial state of the first box is checked. The status of each check box is displayed. Each time you change the state of a check box, the status display is updated.

// Demonstrate check boxes.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CheckboxDemo" width=250 height=200>
</applet>
*/
public class CheckboxDemo extends Applet implements ItemListener {
    String msg = "";
    Checkbox winXP, winVista, solaris, mac;
    public void init() {
        winXP = new Checkbox("Windows XP", null, true);
        winVista = new Checkbox("Windows Vista");
        solaris = new Checkbox("Solaris");
        mac = new Checkbox("Mac OS");
        add(winXP);
        add(winVista);
        add(solaris);
        add(mac);
        winXP.addItemListener(this);
        winVista.addItemListener(this);
        solaris.addItemListener(this);
        mac.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie) {
        repaint();
    }
    // Display current state of the check boxes.
    public void paint(Graphics g) {
        msg = "Current state: ";
        g.drawString(msg, 6, 80);
        msg = " Windows XP: " + winXP.getState();
        g.drawString(msg, 6, 100);
        msg = " Windows Vista: " + winVista.getState();
        g.drawString(msg, 6, 120);
        msg = " Solaris: " + solaris.getState();
        g.drawString(msg, 6, 140);
        msg = " Mac OS: " + mac.getState();
        g.drawString(msg, 6, 160);
    }
}
```



```
}
}
```

Checkbox Group

It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called radio buttons, because they act like the station selector on a car radio—only one station can be selected at any one time. To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes. Check box groups are objects of type `CheckboxGroup`. Only the default constructor is defined, which creates an empty group. You can determine which check box in a group is currently selected by calling `getSelectedCheckbox()`. You can set a check box by calling `setSelectedCheckbox()`. These methods are as follows:

Checkbox `getSelectedCheckbox()`

`void setSelectedCheckbox(Checkbox which)`

Here, `which` is the check box that you want to be selected. The previously selected check box will be turned off. Here is a program that uses check boxes that are part of a group:

// Demonstrate check box group.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CBGroup" width=250 height=200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener {
    String msg = "";
    Checkbox winXP, winVista, solaris, mac;
    CheckboxGroup cbg;
    public void init() {
        cbg = new CheckboxGroup();
        winXP = new Checkbox("Windows XP", cbg, true);
        winVista = new Checkbox("Windows Vista", cbg, false);
        solaris = new Checkbox("Solaris", cbg, false);
        mac = new Checkbox("Mac OS", cbg, false);
        add(winXP);
        add(winVista);
        add(solaris);
        add(mac);
        winXP.addItemListener(this);
        winVista.addItemListener(this);
        solaris.addItemListener(this);
        mac.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie) {
        repaint();
    }
    // Display current state of the check boxes.
    public void paint(Graphics g) {
```

```

        msg = "Current selection: ";
        msg += cbg.getSelectedCheckbox().getLabel();
        g.drawString(msg, 6, 100);
    }
}

```

Choice Controls

The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu. When inactive, a Choice component takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made. Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object. Choice only defines the default constructor, which creates an empty list. To add a selection to the list, call `add()`. It has this general form:

```
void add(String name)
```

Here, name is the name of the item being added. Items are added to the list in the order in which calls to `add()` occur. To determine which item is currently selected, you may call either **`getSelectedItem()`** or **`getSelectedIndex()`**. These methods are shown here:

```
String getItem( )
```

```
int getSelectedIndex( )
```

The `getItem()` method returns a string containing the name of the item. `getSelectedIndex()` returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected.

To obtain the number of items in the list, call `getItemCount()`. You can set the currently selected item using the `select()` method with either a zero-based integer index or a string that will match a name in the list. These methods are shown here:

```
int getItemCount( )
```

```
void select(int index)
```

```
void select(String name)
```

Given an index, you can obtain the name associated with the item at that index by calling `getItem()`, which has this general form:

```
String getItem(int index)
```

Each time a choice is selected, an item event is generated. This is sent to any listeners that previously registered an interest in receiving item event notifications from that component. Each listener implements the `ItemListener` interface. That interface defines the **`itemStateChanged()`** method. An `ItemEvent` object is supplied as the argument to this method. Here is an example that creates two Choice menus. One selects the operating system.

The other selects the browser.

// Demonstrate Choice lists.

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ChoiceDemo" width=300 height=180>

```

```
</applet>
*/
public class ChoiceDemo extends Applet implements ItemListener {
    Choice os, browser;
    String msg = " ";
    public void init() {
        os = new Choice( );
        browser = new Choice();
        // add items to os list
        os.add("Windows XP");
        os.add("Windows Vista");
        os.add("Solaris");
        os.add("Mac OS");
        // add items to browser list
        browser.add("Internet Explorer");
        browser.add("Firefox");
        browser.add("Opera");
        // add choice lists to window
        add(os);
        add(browser);
        // register to receive item events
        os.addItemListener(this);
        browser.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie) {
        repaint();
    }
    // Display current selections.
    public void paint(Graphics g) {
        msg = "Current OS: ";
        msg += os.getSelectedItem();
        g.drawString(msg, 6, 120);
        msg = "Current Browser: ";
        msg += browser.getSelectedItem();
        g.drawString(msg, 6, 140);
    }
}
```

Lists

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

List() throws HeadlessException

List(int numRows) throws HeadlessException

List(int numRows, boolean multipleSelect) throws HeadlessException

The first version creates a List control that allows only one item to be selected at any one time. In the second form, the value of numRows specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed). In the third form, if

multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected. To add a selection to the list, call add(). It has the following two forms:

void add(String name)

void add(String name, int index)

Here, name is the name of the item added to the list. The first form adds items to the end of the list. The second form adds the item at the index specified by index. Indexing begins at zero. You can specify -1 to add the item to the end of the list. For lists that allow only single selection, you can determine which item is currently selected by calling either getSelectedItem() or getSelectedIndex(). These methods are shown here:

String getSelectedItem()

int getSelectedIndex()

The getSelectedItem() method returns a string containing the name of the item. If more than one item is selected, or if no selection has yet been made, null is returned. getSelectedIndex()

returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, -1 is returned.

getSelectedItems() returns an array containing the names of the currently selected items.

getSelectedIndexes() returns an array containing the indexes of the currently selected items.

To obtain the number of items in the list, call getItemCount(). You can set the currently selected item by using the select() method with a zero-based integer index. These methods are shown here:

int getItemCount()

void select(int index)

Given an index, you can obtain the name associated with the item at that index by calling getItem(), which has this general form:

String getItem(int index)

Here, index specifies the index of the desired item.

To process list events, you will need to implement the ActionListener interface. Each time a List item is double-clicked, an(ActionEvent) object is generated. Its getActionCommand() method can be used to retrieve the name of the newly selected item. Also, each time an item is selected or deselected with a single click, an(ItemsEvent) object is generated. Its

getStateChange()

method can be used to determine whether a selection or deselection triggered this event.

getItemSelectable() returns a reference to the object that triggered this event. Here is an example that converts the Choice controls in the preceding section into List components, one multiple choice and the other single choice

// Demonstrate Lists.

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

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

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

```
/*
```

```
<applet code="ListDemo" width=300 height=180>
```

```
</applet>
```

```
*/
```

```
public class ListDemo extends Applet implements ActionListener {
    List os, browser;
    String msg = "";
    public void init() {
        os = new List(4, true);
        browser = new List(4, false);
        // add items to os list
        os.add("Windows XP");
        os.add("Windows Vista");
        os.add("Solaris");
        os.add("Mac OS");
        // add items to browser list
        browser.add("Internet Explorer");
        browser.add("Firefox");
        browser.add("Opera");
        browser.select(1);
        // add lists to window
        add(os);
        add(browser);
        // register to receive action events
        os.addActionListener(this);
        browser.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae) {
        repaint();
    }
    // Display current selections.
    public void paint(Graphics g) {
        int idx[ ];
        msg = "Current OS: ";
        idx = os.getSelectedIndexes();
        for(int i=0; i<idx.length; i++)
            msg += os.getItem(idx[i]) + " ";
        g.drawString(msg, 6, 120);
        msg = "Current Browser: ";
        msg += browser.getSelectedItem();
        g.drawString(msg, 6, 140);
    }
}
```

Scroll Bars

Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. A scroll bar is actually a composite of several individual parts. Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow. The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar. The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value. In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1. Typically, this

action translates into some form of page up and page down. Scroll bars are encapsulated by the Scrollbar class. Scrollbar defines the following constructors:

Scrollbar() throws HeadlessException

Scrollbar(int style) throws HeadlessException

Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

throws HeadlessException

The first form creates a vertical scroll bar. The second and third forms allow you to specify the orientation of the scroll bar. If style is **Scrollbar.VERTICAL**, a vertical scroll bar is created. If style is **Scrollbar.HORIZONTAL**, the scroll bar is horizontal. In the third form of the constructor, the initial value of the scroll bar is passed in initialValue. The number of units represented by the height of the thumb is passed in thumbSize. The minimum and maximum values for the scroll bar are specified by min and max.

If you construct a scroll bar by using one of the first two constructors, then you need to set its parameters by using setValues(), shown here, before it can be used:

void setValues(int initialValue, int thumbSize, int min, int max)

The parameters have the same meaning as they have in the third constructor just described. To obtain the current value of the scroll bar, call getValue(). It returns the current setting. To set the current value, call setValue(). These methods are as follows:

int getValue()

void setValue(int newValue)

Here, newValue specifies the new value for the scroll bar. When you set a value, the slider box inside the scroll bar will be positioned to reflect the new value. You can also retrieve the minimum and maximum values via getMinimum() and getMaximum(), shown here:

int getMinimum()

int getMaximum()

They return the requested quantity. By default, 1 is the increment added to or subtracted from the scroll bar each time it is scrolled up or down one line. You can change this increment by calling setUnitIncrement().

By default, page-up and page-down increments are 10. You can change this value by calling setBlockIncrement(). These methods are shown here:

void setUnitIncrement(int newIncr)

void setBlockIncrement(int newIncr)

To process scroll bar events, you need to implement the AdjustmentListener interface. Each time a user interacts with a scroll bar, an AdjustmentEvent object is generated. Its getAdjustmentType() method can be used to determine the type of the adjustment. The types of adjustment events are as follows:

BLOCK_DECREMENT- A page-down event has been generated.

BLOCK_INCREMENT -A page-up event has been generated.

TRACK -An absolute tracking event has been generated.

UNIT_DECREMENT- The line-down button in a scroll bar has been pressed.

UNIT_INCREMENT-The line-up button in a scroll bar has been pressed.

// Demonstrate scroll bars.

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/*

```
<applet code="SBDemo" width=300 height=200>
</applet>
*/
public class SBDemo extends Applet
    implements AdjustmentListener, MouseMotionListener {
    String msg = " ";
    Scrollbar vertSB, horzSB;
    public void init() {
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, height);
        horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, width);
        add(vertSB);
        add(horzSB);
        // register to receive adjustment events
        vertSB.addAdjustmentListener(this);
        horzSB.addAdjustmentListener(this);
        addMouseMotionListener(this);
    }
    public void adjustmentValueChanged(AdjustmentEvent ae) {
        repaint( );
    }
    // Update scroll bars to reflect mouse dragging.
    public void mouseDragged(MouseEvent me) {
        int x = me.getX();
        int y = me.getY();
        vertSB.setValue(y);
        horzSB.setValue(x);
        repaint();
    }
    // Necessary for MouseMotionListener
    public void mouseMoved(MouseEvent me) {
    }
    // Display current value of scroll bars.
    public void paint(Graphics g) {
        msg = "Vertical: " + vertSB.getValue();
        msg += ", Horizontal: " + horzSB.getValue( );
        g.drawString(msg, 6, 160);
        // show current mouse drag position
        g.drawString("*", horzSB.getValue(),
            vertSB.getValue());
    }
}
```

TextField

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

TextField() throws HeadlessException

TextField(int numChars) throws HeadlessException

TextField(String str) throws HeadlessException

TextField(String str, int numChars) throws HeadlessException

The first version creates a default text field. The second form creates a text field that is numChars characters wide. The third form initializes the text field with the string contained in str. The fourth form initializes a text field and sets its width.

TextField (and its superclass TextComponent) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call getText(). To set the text, call setText(). These methods are as follows:

String getText()

void setText(String str)

Here, str is the new string. The user can select a portion of the text in a text field. Also, you can select a portion of text under program control by using select(). Your program can obtain the currently selected text by calling getSelectedText(). These methods are shown here:

String getSelectedText()

void select(int startIndex, int endIndex)

getSelectedText() returns the selected text. The select() method selects the characters beginning at startIndex and ending at endIndex-1.

You can control whether the contents of a text field may be modified by the user by calling setEditable(). You can determine editability by calling isEditable(). These methods are shown here:

boolean isEditable()

void setEditable(boolean canEdit)

isEditable() returns true if the text may be changed and false if not. In setEditable(), if canEdit is true, the text may be changed. If it is false, the text cannot be altered. There may be times when you will want the user to enter text that is not displayed, such as a password. You can disable the echoing of the characters as they are typed by calling setEchoChar(). This method specifies a single character that the TextField will display when characters are entered (thus, the actual characters typed will not be shown). You can check a text field to see if it is in this mode with the echoCharIsSet() method. You can retrieve the echo character by calling the getEchoChar() method. These methods are as follows:

void setEchoChar(char ch)

boolean echoCharIsSet()

char getEchoChar()

Here, ch specifies the character to be echoed. Since text fields perform their own editing functions, your program generally will not respond to individual key events that occur within a text field. However, you may want to respond when the user presses ENTER. When this occurs, an action event is generated. Here is an example that creates the classic user name and password screen:

// Demonstrate text field.

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



```
<applet code="TextFieldDemo" width=380 height=150>
</applet>
*/
public class TextFieldDemo extends Applet
    implements ActionListener {
        TextField name, pass;
        public void init() {
            Label namep = new Label("Name: ", Label.RIGHT);
            Label passp = new Label("Password: ", Label.RIGHT);
            name = new TextField(12);
            pass = new TextField(8);
            pass.setEchoChar('?');
            add(namep);
            add(name);
            add(passp);
            add(pass);
            // register to receive action events
            name.addActionListener(this);
            pass.addActionListener(this);
        }
        // User pressed Enter.
        public void actionPerformed(ActionEvent ae) {
            repaint( );
        }
        public void paint(Graphics g) {
            g.drawString("Name: " + name.getText(), 6, 60);
            g.drawString("Selected text in name: " + name.getSelectedText(), 6, 80);
            g.drawString("Password: " + pass.getText(), 6, 100);
        }
    }
}
```

TextArea

Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea. Following are the constructors for TextArea:

TextArea() throws HeadlessException

TextArea(int numLines, int numChars) throws HeadlessException

TextArea(String str) throws HeadlessException

TextArea(String str, int numLines, int numChars) throws HeadlessException

TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException

Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters. Initial text can be specified by str. In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values: SCROLLBARS_BOTH, SCROLLBARS_NONE, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_VERTICAL_ONLY.

TextArea is a subclass of TextComponent. Therefore, it supports the getText(), setText(), getSelectedText(), select(), isEditable(), and setEditable() methods described in the preceding section. TextArea adds the following methods

void append(String str)**void insert(String str, int index)****void replaceRange(String str, int startIndex, int endIndex)**

The append() method appends the string specified by str to the end of the current text. insert() method inserts the string passed in str at the specified index. To replace text, call replaceRange(). It replaces the characters from startIndex to endIndex-1, with the replacement text passed in str. Text areas are almost self-contained controls. Your program incurs virtually no management overhead. Text areas only generate got-focus and lost-focus events.

Normally, your program simply obtains the current text when it is needed. The following program creates a TextArea control:

// Demonstrate TextArea.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="TextAreaDemo" width=300 height=250>
</applet>
*/
public class TextAreaDemo extends Applet {
    public void init() {
        String val =
"Java SE 6 is the latest version of the most\n" +
"widely-used computer language for Internet programming.\n" +
"Building on a rich heritage, Java has advanced both\n" +
"the art and science of computer language design.\n\n" +
"One of the reasons for Java's ongoing success is its\n" +
"constant, steady rate of evolution. Java has never stood\n" +
"still. Instead, Java has consistently adapted to the\n" +
"rapidly changing landscape of the networked world.\n" +
"Moreover, Java has often led the way, charting the\n" +
"course for others to follow.";
        TextArea text = new TextArea(val, 10, 30);
        add(text);
    }
}
```

Layout Managers

Layout manager automatically arranges controls within a window by using some type of algorithm. If you have programmed for other GUI environments, such as Windows, then you are accustomed to laying out your controls by hand. While it is possible to lay out Java controls by hand, too, you generally won't want to, for two main reasons. First, it is very tedious to manually layout a large number of components. Second, sometimes the width and height information is not yet available when you need to arrange some control, because the native toolkit components haven't been realized.

Each Container object has a layout manager associated with it. A layout manager is an instance of any class that implements the `LayoutManager` interface. The layout manager is set by the `setLayout()` method. If no call to `setLayout()` is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The `setLayout()` method has the following general form:

`void setLayout(LayoutManager layoutObj)`

Here, `layoutObj` is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass null for `layoutObj`. If you do this, you will need to determine the shape and position of each component manually, using the `setBounds()` method defined by `Component`. Normally, you will want to use a layout manager.

Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time you add a component to a container. Whenever the container needs to be resized, the layout manager is consulted via its `minimumLayoutSize()` and `preferredLayoutSize()` methods. Each component that is being managed by a layout manager contains the `getPreferredSize()` and `getMinimumSize()` methods. These return the preferred and minimum size required to display each component. The layout manager will honor these requests if at all possible, while maintaining the integrity of the layout policy. You may override these methods for controls that you subclass. Default values are provided otherwise.

FlowLayout

`FlowLayout` is the default layout manager. `FlowLayout` implements a simple layout style, which is similar to how words flow in a text editor. The direction of the layout is governed by the container's component orientation property, which, by default, is left to right, top to bottom. Therefore, by default, components are laid out line-by-line beginning at the upper-left corner. In all cases, when a line is filled, layout advances to the next line. A small space is left between each component, above and below, as well as left and right. Here are the constructors for `FlowLayout`:

`FlowLayout()`

`FlowLayout(int how)`

`FlowLayout(int how, int horz, int vert)`

The first form creates the default layout, which centers components and leaves five pixels of space between each component. The second form lets you specify how each line is aligned. Valid values for `how` are as follows:

`FlowLayout.LEFT`

`FlowLayout.CENTER`

`FlowLayout.RIGHT`

`FlowLayout.LEADING`

FlowLayout.TRAILING

These values specify left, center, right, leading edge, and trailing edge alignment, respectively. The third constructor allows you to specify the horizontal and vertical space left between components in `horz` and `vert`, respectively. Here is a version of the `CheckboxDemo` applet shown earlier in this chapter, modified so that it uses left-aligned flow layout:

// Use left-aligned flow layout.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="FlowLayoutDemo" width=250 height=200>
</applet>
*/
public class FlowLayoutDemo extends Applet
    implements ItemListener {
    String msg = "";
    Checkbox winXP, winVista, solaris, mac;
public void init( ) {
    // set left-aligned flow layout
    setLayout(new FlowLayout(FlowLayout.LEFT));
    winXP = new Checkbox("Windows XP", null, true);
    winVista = new Checkbox("Windows Vista");
    solaris = new Checkbox("Solaris");
    mac = new Checkbox("Mac OS");
    add(winXP);
    add(winVista);
    add(solaris);
    add(mac);
    // register to receive item events
    winXP.addItemListener(this);
    winVista.addItemListener(this);
    solaris.addItemListener(this);
    mac.addItemListener(this);
}
// Repaint when status of a check box changes.
public void itemStateChanged(ItemEvent ie) {
    repaint();
}
// Display current state of the check boxes.
public void paint(Graphics g) {
    msg = "Current state: ";
    g.drawString(msg, 6, 80);
    msg = " Windows XP: " + winXP.getState();
    g.drawString(msg, 6, 100);
    msg = " Windows Vista: " + winVista.getState();
    g.drawString(msg, 6, 120);
    msg = " Solaris: " + solaris.getState();
    g.drawString(msg, 6, 140);
    msg = " Mac: " + mac.getState();
```

```
g.drawString(msg, 6, 160);
}
}
```

BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north, south, east, and west. The middle area is called the center. Here are the constructors defined by BorderLayout:

BorderLayout()

BorderLayout(int horz, int vert)

The first form creates a default border layout. The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively. BorderLayout defines constants that specify the regions as BorderLayout.CENTER, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.NORTH.

When adding components, you will use these constants with the following form of add(), which is defined by Container:

void add(Component compObj, Object region)

Here, compObj is the component to be added, and region specifies where the component will be added. Here is an example of a BorderLayout with a component in each layout area:

// Demonstrate BorderLayout.

```
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code="BorderLayoutDemo" width=400 height=200>
</applet>
*/
public class BorderLayoutDemo extends Applet {
    public void init( ) {
        setLayout(new BorderLayout());
        add(new Button("This is across the top."),
            BorderLayout.NORTH);
        add(new Label("The footer message might go here."),
            BorderLayout.SOUTH);
        add(new Button("Right"), BorderLayout.EAST);
        add(new Button("Left"), BorderLayout.WEST);
        String msg = "The reasonable man adapts " + "himself to the world;\n" +
            "the unreasonable one persists in " +
            "trying to adapt the world to himself.\n" +
            "Therefore all progress depends " +
            "on the unreasonable man.\n\n" + " - George Bernard Shaw\n\n";
        add(new TextArea(msg), BorderLayout.CENTER);
    }
}
```

GridLayout

GridLayout lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns. The constructors supported by GridLayout are shown here:

GridLayout()

GridLayout(int numRows, int numColumns)

GridLayout(int numRows, int numColumns, int horz, int vert)

The first form creates a single-column grid layout. The second form creates a grid layout with the specified number of rows and columns. The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively. Either numRows or numColumns can be zero. Specifying numRows as zero allows for unlimited-length columns. Specifying numColumns as zero allows for unlimited-length rows. Here is a sample program that creates a 4×4 grid and fills it in with 15 buttons, each labeled with its index:

// Demonstrate GridLayout

```
import java.awt.*;
import java.applet.*;
/*
<applet code="GridLayoutDemo" width=300 height=200>
</applet>
*/
public class GridLayoutDemo extends Applet {
    static final int n = 4;
    public void init( ) {
        setLayout(new GridLayout(n, n));
        setFont(new Font("SansSerif", Font.BOLD, 24));
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                int k = i * n + j;
                if(k > 0)
                    add(new Button("" + k));
            }
        }
    }
}
```

Menu Bars and Menus

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes:MenuBar,Menu, andMenuItem. In general, a menu bar contains one or more Menu objects. EachMenu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user. SinceMenu is a subclass ofMenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of typeCheckboxMenuItem and will have a check mark next to them when they are selected. To create a menu bar, first create an instance ofMenuBar. This class only defines the default

constructor. Next, create instances of `Menu` that will define the selections displayed on the bar. Following are the constructors for `Menu`:

`Menu()` throws `HeadlessException`

`Menu(String optionName)` throws `HeadlessException`

`Menu(String optionName, boolean removable)` throws `HeadlessException`

Here, `optionName` specifies the name of the menu selection. If `removable` is true, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.) The first form creates an empty menu. Individual menu items are of type `MenuItem`. It defines these constructors:

`MenuItem()` throws `HeadlessException`

`MenuItem(String itemName)` throws `HeadlessException`

`MenuItem(String itemName, MenuShortcut keyAccel)` throws `HeadlessException`

Here, `itemName` is the name shown in the menu, and `keyAccel` is the menu shortcut for this item. You can disable or enable a menu item by using the `setEnabled()` method. Its form is shown here:

`void setEnabled(boolean enabledFlag)`

If the argument `enabledFlag` is true, the menu item is enabled. If false, the menu item is disabled. You can determine an item's status by calling **`isEnabled()`**. This method is shown here:

`boolean isEnabled()`

`isEnabled()` returns true if the menu item on which it is called is enabled. Otherwise, it returns false. You can change the name of a menu item by calling **`setLabel()`**. You can retrieve the current name by using `getLabel()`. These methods are as follows:

`void setLabel(String newName)`

`String getLabel()`

Here, `newName` becomes the new name of the invoking menu item. `getLabel()` returns the current name. You can create a checkable menu item by using a subclass of `MenuItem` called `CheckboxMenuItem`. It has these constructors:

`CheckboxMenuItem()` throws `HeadlessException`

`CheckboxMenuItem(String itemName)` throws `HeadlessException`

`CheckboxMenuItem(String itemName, boolean on)` throws `HeadlessException`

Here, `itemName` is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes. In the first two forms, the checkable entry is unchecked. In the third form, if `on` is true, the checkable entry is initially checked. Otherwise, it is cleared. You can obtain the status of a checkable item by calling `getState()`. You can set it to a known state by using `setState()`. These methods are shown here:

`boolean getState()`

`void setState(boolean checked)`

If the item is checked, `getState()` returns true. Otherwise, it returns false. To check an item, pass true to `setState()`. To clear an item, pass false. Once you have created a menu item, you must add the item to a `Menu` object by using `add()`, which has the following general form:

`MenuItem add(MenuItem item)`

Here, `item` is the item being added. Items are added to a menu in the order in which the calls to `add()` take place. The item is returned. Once you have added all items to a `Menu` object, you can add that object to the menu bar by using this version of `add()` defined by `MenuBar`:

`Menu add(Menu menu)`

Here, menu is the menu being added. The menu is returned. Menus only generate events when an item of type MenuItem or CheckboxMenuItem is selected. They do not generate events when a menu bar is accessed to display a drop-down menu, for example. Each time a menu item is selected, an ActionEvent object is generated. By default, the action command string is the name of the menu item. However, you can specify a different action command string by calling setActionCommand() on the menu item. Each time a check box menu item is checked or unchecked, an ItemEvent object is generated. Thus, you must implement the ActionListener and/or ItemListener interfaces in order to handle these menu events. The getItem() method of ItemEvent returns a reference to the item that generated this event. The general form of this method is shown here:

Object getItem()

Assignment 2

MCQ

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

- A. Window B. Frame C. Panel D. Container

Answer: Option C

2. Which package provides many event classes and Listener interfaces for event handling?

- A. java.awt B. java.awt.Graphics
C. java.awt.event D. None of the above

Answer: Option C

3. In Graphics class which method is used to draw a rectangle with the specified width and height?

- A. public void drawRect(int x, int y, int width, int height)
B. public abstract void fillRect(int x, int y, int width, int height)
C. public abstract void drawLine(int x1, int y1, int x2, int y2)
D. public abstract void drawOval(int x, int y, int width, int height)

Answer: Option A

4. Name the class used to represent a GUI application window, which is optionally resizable and can have a title bar, an icon, and menus.

- A. Window
B. Panel
C. Dialog
D. Frame

Answer: Option D

5. To use the ActionListener interface it must be implemented by a class there are several ways to do that find in the following?

- A. Creating a new class B. Using the class the graphical component
C. An anonymous inner class D. All mentioned above

Answer: Option D

6. Which is a component in AWT that can contain another component like buttons, text fields, labels etc.?

- A. Window
B. Container
C. Panel
D. Frame

Answer: Option B

7. Which is used to store data and partial results, as well as to perform dynamic linking, return values for methods, and dispatch exceptions?

- A. Window
- B. Panel
- C. Frame
- D. Container

Answer: Option C

8. What are the different types of controls in AWT?

- A. Labels
- B. Pushbuttons
- C. Checkboxes
- D. Choice lists
- E. All of these

Answer: Option E

9. Which class provides many methods for graphics programming?

- A. java.awt
- B. java.Graphics
- C. java.awt.Graphics
- D. java. awt.event

Answer: Option C

10. By which method You can set or change the text in a Label?

- A. setText()
- B. getText()
- C. Both A & B
- D. None of the above

Answer: Option A

11. Which class can be used to represent a checkbox with a textual label that can appear in a menu.

- A. MenuBar
- B. MenuItem
- C. CheckboxMenuItem
- D.Menu

Answer: Option C

12. How many types of controls does AWT supports these controls are subclasses of component?

- A. 7
- B. 6
- C. 5
- D. 8

Answer: Option A

13. Which are passive controls that do not support any interaction with the user?

- A. Choice
- B. List
- C. Labels
- D. Checkbox

Answer: Option C

14. Which of the following classes are derived from the Component class.

- A. Container
- B. Window
- C. List
- D. MenuItem

Answer: Option D

15. How many ways can we align the label in a container?

- A. 1
- B. 2
- C. 3
- D. 4

Answer: Option C

16. Which method is used to set the graphics current color to the specified color in the graphics class?

- A. public abstract void setFont(Font font)
- B. public abstract void setColor(Color c)
- C. public abstract void drawString(String str, int x, int y)
- D. None of the above

Answer: Option B

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

- A. Labels
- B. Choice
- C. List
- D. Checkbox

Answer: Option C

18. Which class is used for this Processing Method processActionEvent()?

- A. Button, List, MenuItem
- B. Button, Checkbox, Choice
- C. Scrollbar, Component, Button
- D. None of the above

Answer: Option A

19. Which package provides many event classes and Listener interfaces for event handling?

- A. java.awt
- B. java.awt.Graphics
- C. java.awt.event
- D. None of the above

Answer: Option C

20. Name the class used to represent a GUI application window, which is optionally resizable and can have a title bar, an icon, and menus.

- A. Window
- B. Panel
- C. Dialog
- D. Frame

Answer: Option D

21. What does the following line of code do?

Textfield text = new Textfield(10);

- A. Creates text object that can hold 10 rows of text.
- B. Creates the object text and initializes it with the value 10.
- C. The code is illegal.
- D. Creates text object that can hold 10 columns of text.

Answer: Option D

22. Which of the following methods can be used to remove a java.awt.Component object from the display?

- A. delete()
- B. remove()
- C. disappear()
- D. hide()

Answer: Option D

23. The setBackground() method is part of the following class in java.awt package:

- A. Component
- B. Graphics
- C. Applet
- D. Container

Answer: Option A

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

- A. update()
- B. draw()
- C. show()
- D. paint()

Answer: Option A

25. Where are the following four methods commonly used?

- 1) public void add(Component c)

- 2) `public void setSize(int width,int height)`
- 3) `public void setLayout(LayoutManager m)`
- 4) `public void setVisible(boolean)`

A. Graphics class

B. Component class

C. Both A & B

D. None of the above

Answer: Option B

Long Answer Questions

1. Illustrate the use of Label with suitable example.
2. List and explain various methods associated with Button.
3. Explain the purpose of Choice controls with an example.
4. Explain List control with an example.
5. Explain Scroll Bars.
6. Explain about Layout Managers.
7. With an example explain FlowLayout.
8. With an example explain BorderLayout.
9. What is the purpose of layout managers? With an example explain the use of Grid Layout.
10. Name the controls required for creating a menu. With an example explain the creation of a menu.
11. List and explain different methods associated with check boxes.
12. What is the purpose of TextField class? Explain any three methods of TextField class with syntax and example.

UNIT-3

Introducing Swing

The Origins and Design Philosophy of Swing

The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or peers. This means that the look and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as heavyweight.

The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. This potential variability threatened the overarching philosophy of Java: write once, run anywhere. Second, the look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed. Third, the use of heavyweight components caused some frustrating restrictions.

Two Key Swing Features: Swing was created to address the limitations present in the AWT. It does this through two key features: lightweight components and a pluggable look and feel. Together they provide an elegant, yet easy-to-use solution to the problems of the AWT. More than anything else, it is these two features that define the essence of Swing.

Swing Components Are Lightweight: With very few exceptions, Swing components are lightweight. This means that they are written entirely in Java and do not map directly to platform-specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. Furthermore, because lightweight components do not translate into native peers, the look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.

Swing Supports a Pluggable Look and Feel: Swing supports a pluggable look and feel (PLAF). Because each Swing component is rendered by Java code rather than by native peers, the look and feel of a component is under the control of Swing. This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what Swing does. Separating out the look and feel provides a significant advantage: it becomes possible to change the way that a component is rendered without affecting any of its other aspects. In other words, it is possible to —plug in a new look and feel for any given component without creating any side effects in the code that uses that component. Moreover, it becomes possible to define entire sets of look-and-feels that represent different GUI styles. To use a specific style, its look and feel is simply —plugged in. Once this is done, all components are automatically rendered using that style. Pluggable look-and-feels offer several important advantages. It is possible to define a look and feel that is consistent across all platforms. Conversely, it is possible to create a look and feel that acts like a specific platform. For example, if you know that an application will be running only in a Windows environment, it is possible to specify the Windows look and feel. It is also possible to design a custom look and feel. Finally, the look and feel can be changed dynamically at run time.

Java SE 6 provides look-and-feels, such as metal and Motif, that are available to all Swing users. The metal look and feel is also called the Java look and feel. It is platform-independent and available in all Java execution environments. It is also the default look and feel. Windows environments also have access to the Windows and Windows Classic look and feel.

Components and Containers

A Swing GUI consists of two key items: components and containers. However, this distinction is mostly conceptual because all containers are also components. The difference between the two is found in their intended purpose: As the term is commonly used, a component is an independent visual control, such as a push button or slider. A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components. Furthermore, in order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container. Because containers are components, a container can also hold other containers. This enables Swing to define what is called a containment hierarchy, at the top of which must be a top-level container.

Components: In general, Swing components are derived from the `JComponent` class. `JComponent` provides the functionality that is common to all components. For example, `JComponent` supports the pluggable look and feel. `JComponent` inherits the AWT classes `Container` and `Component`. Thus, a Swing component is built on and compatible with an AWT component. All of Swing's components are represented by classes defined within the package `javax.swing`. The following are the class names for Swing components.

`JApplet`, `JButton`, `JCheckBox`, `JCheckBoxMenuItem`, `JColorChooser`, `JComboBox`, `JComponent`, `JDesktopPane`, `JDialog`, `JEditorPane`, `JFileChooser`, `JFormattedTextField`, `JFrame`, `JInternalFrame`, `JLabel`, `JLayeredPane`, `JList`, `JMenu`, `JMenuBar` and `JMenuItem` etc.

Notice that all component classes begin with the letter J. For example, the class for a label is `JLabel`; the class for a push button is `JButton`; and the class for a scroll bar is `JScrollBar`.

Containers

Swing defines two types of containers. The first are top-level containers: `JFrame`, `JApplet`, `JWindow`, and `JDialog`. These containers do not inherit `JComponent`. They do, however, inherit the AWT classes `Component` and `Container`. Unlike Swing's other components, which are lightweight, the top-level containers are heavyweight. This makes the top-level containers a special case in the Swing component library.

As the name implies, a top-level container must be at the top of a containment hierarchy. A top-level container is not contained within any other container. Furthermore, every containment hierarchy must begin with a top-level container. The one most commonly used for applications is `JFrame`. The one used for applets is `JApplet`. The second type of containers supported by Swing are lightweight containers. Lightweight containers do inherit `JComponent`. An example of a lightweight container is `JPanel`, which is a general-purpose container. Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container. Thus, you can use lightweight containers such as `JPanel` to create subgroups of related controls that are contained within an outer container.

Each top-level container defines a set of panes. At the top of the hierarchy is an instance of `JRootPane`. `JRootPane` is a lightweight container whose purpose is to manage the other panes. It also helps manage the optional menu bar. The panes that comprise the root pane are called the glass pane, the content pane, and the layered pane. The glass pane is the top-level pane. It sits above and completely covers all other panes. By default, it is a transparent instance of `JPanel`. The glass pane enables you to manage mouse events that affect the entire container (rather than an individual control) or to paint over any other component, for example. In most cases, you won't need to use the glass pane directly, but it is there if you need it.

Layout Managers

layout manager automatically arranges controls within a window by using some type of algorithm. If you have programmed for other GUI environments, such as Windows, then you are accustomed to laying out your controls by hand. While it is possible to lay out Java controls by hand, too, you generally won't want to, for two main reasons. First, it is very tedious to manually layout a large number of components. Second, sometimes the width and height information is not yet available when you need to arrange some control, because the native toolkit components haven't been realized.

Each Container object has a layout manager associated with it. A layout manager is an instance of any class that implements the `LayoutManager` interface. The layout manager is set by the `setLayout()` method. If no call to `setLayout()` is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The `setLayout()` method has the following general form:

`void setLayout(LayoutManager layoutObj)`

Here, `layoutObj` is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass null for `layoutObj`. If you do this, you will need to determine the shape and position of each component manually, using the `setBounds()` method defined by `Component`. Normally, you will want to use a layout manager.

Use JButton

The `JButton` class provides the functionality of a push button. `JButton` allows an icon, a string, or both to be associated with the push button. Three of its constructors are shown here:

`JButton(Icon icon)`

`JButton(String str)`

`JButton(String str, Icon icon)`

Here, `str` and `icon` are the string and icon used for the button. When the button is pressed, an `ActionEvent` is generated. Using the `ActionEvent` object passed to the `actionPerformed()` method of the registered `ActionListener`, you can obtain the action command string associated with the button. By default, this is the string displayed inside the button. However, you can set the action command by calling `setActionCommand()` on the button. You can obtain the action command by calling `getActionCommand()` on the event object. It is declared like this:

`String getActionCommand()`

The action command identifies the button. Thus, when using two or more buttons within the same application, the action command gives you an easy way to determine which button was pressed.

Work with JTextField

TextField is the simplest Swing text component. It is also probably its most widely used text component. JTextField allows you to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to Swing text components. JTextField uses the Document interface for its model.

Three of JTextField's constructors are shown here:

```
JTextField(int cols)
```

```
JTextField(String str, int cols)
```

```
JTextField(String str)
```

Here, str is the string to be initially presented, and cols is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string. JTextField generates events in response to user interaction. For example, an ActionEvent is fired when the user presses ENTER. ACaretEvent is fired each time the caret (i.e., the cursor) changes position. (CaretEvent is packaged in javax.swing.event.) Other events are also possible. In many cases, your program will not need to handle these events. Instead, you will simply obtain the string currently in the text field when it is needed. To obtain the text currently in the text field, call getText().

Create a Checkbox

The JCheckBox class provides the functionality of a check box. Its immediate superclass is JToggleButton, which provides support for two-state buttons, as just described. JCheckBox defines several constructors. The one used here is JCheckBox(String str) It creates a check box that has the text specified by str as a label. Other constructors let you specify the initial selection state of the button and specify an icon. When the user selects or deselects a check box, an ItemEvent is generated. You can obtain a reference to the JCheckBox that generated the event by calling getItem() on the ItemEvent passed to the itemStateChanged() method defined by ItemListener. The easiest way to determine the selected state of a check box is to call isSelected() on the JCheckBox instance. In addition to supporting the normal check box operation, JCheckBox lets you specify the icons that indicate when a check box is selected, cleared, and rolled-over. We won't be using this capability here, but it is available for use in your own programs.

Work with Jlist

In Swing, the basic list class is called JList. It supports the selection of one or more items from a list. Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed. JList is so widely used in Java that it is highly unlikely that you have not seen one before. JList provides several constructors. The one used here is

```
JList(Object[ ] items)
```

This creates a JList that contains the items in the array specified by items. JList is based on two models. The first is ListModel. This interface defines how access to the list data is achieved. The second model is the ListSelectionModel interface, which defines methods that determine what list item or items are selected. Although a JList will work properly by itself, most of the time you will wrap a JList inside a JScrollPane. This way, long lists will automatically be scrollable, which simplifies GUI design. It also makes it easy to change the number of entries in a list without having to change the size of the JList component. AJList generates a ListSelectionEvent when the user makes or changes a selection. This event is also generated when the user deselects an item. It is handled by implementing

ListSelectionListener. This listener specifies only one method, called `valueChanged()`, which is shown here:

```
void valueChanged(ListSelectionEvent le)
```

Here, `le` is a reference to the object that generated the event. Although `ListSelectionEvent` does provide some methods of its own, normally you will interrogate the `JList` object itself to determine what has occurred. Both `ListSelectionEvent` and `ListSelectionListener` are packaged in `javax.swing.event`.

By default, a `JList` allows the user to select multiple ranges of items within the list, but you can change this behavior by calling `setSelectionMode()`, which is defined by `JList`. It is shown here:

```
void setSelectionMode(int mode)
```

Here, `mode` specifies the selection mode. It must be one of these values defined by `ListSelectionModel`:

`SINGLE_SELECTION`

`SINGLE_INTERVAL_SELECTION`

`MULTIPLE_INTERVAL_SELECTION`

The default, multiple-interval selection, lets the user select multiple ranges of items within a list. With single-interval selection, the user can select one range of items. With single selection, the user can select only a single item. Of course, a single item can be selected in the other two modes, too. It's just that they also allow a range to be selected.

You can obtain the index of the first item selected, which will also be the index of the only selected item when using single-selection mode, by calling `getSelectedIndex()`, shown here:

```
int getSelectedIndex()
```

Indexing begins at zero. So, if the first item is selected, this method will return 0. If no item is selected, -1 is returned. Instead of obtaining the index of a selection, you can obtain the value associated with the selection by calling `getSelectedValue()`:

```
Object getSelectedValue()
```

It returns a reference to the first selected value. If no value has been selected, it returns null.

Use anonymous inner classes to handle events

Event handling is one important part of programming in swing. Because `JLabel` does not take input from the user, it does not generate events, so no event handling was needed. However, the other Swing components do respond to user input and the events generated by those interactions need to be handled. Events can also be generated in ways not directly related to user input. For example, an event is generated when a timer goes off. Whatever the case, event handling is a large part of any Swing-based application.

The event handling mechanism used by Swing is the same as that used by the AWT. This approach is called the delegation event model. In many cases, Swing uses the same events as does the AWT, and these events are packaged in `java.awt.event`. Events specific to Swing are stored in `javax.swing.event`. Although events are handled in Swing in the same way as they are with the AWT, it is still useful to work through a simple example. The following program handles the event generated by a Swing push button. Event handling program imports both the `java.awt` and `java.awt.event` packages. The `java.awt` package is needed because it contains the `FlowLayout` class, which supports the standard flow layout manager used to lay out components in a frame.

The java.awt.event package is needed because it defines the ActionListener interface and the(ActionEvent) class. The EventDemo constructor begins by creating a JFrame called jfrm. It then sets the layout manager for the content pane of jfrm to FlowLayout. Recall that, by default, the content pane uses BorderLayout as its layout manager. However, for this example, FlowLayout is more convenient. Notice that FlowLayout is assigned using this statement `jfrm.setLayout(new FlowLayout());`

As explained, in the past you had to explicitly call `getContentPane()` to set the layout manager for the content pane.

Create a Swing applet

The second type of program that commonly uses Swing is the applet. Swing-based applets are similar to AWT-based applets, but with an important difference: A Swing applet extends JApplet rather than Applet. JApplet is derived from Applet. Thus, JApplet includes all of the functionality found in Applet and adds support for Swing. JApplet is a top-level Swing container, which means that it is not derived from JComponent. Because JApplet is a top-level container, it includes the various panes described earlier. This means that all components are added to JApplet's content pane in the same way that components are added to JFrame's content pane.

Swing applets use the same four lifecycle methods as `init()`, `start()`, `stop()`, and `destroy()`. Of course, you need override only those methods that are needed by your applet. Painting is accomplished differently in Swing than it is in the AWT, and a Swing applet will not normally override the `paint()` method. All interaction with components in a Swing applet must take place on the event dispatching thread, as described in the previous section. This threading issue applies to all Swing programs.

Here is an example of a Swing applet. It provides the same functionality as the previous application, but does so in applet form.

// A simple Swing-based applet

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

This HTML can be used to launch the applet:

```
<object code="MySwingApplet" width=220 height=90>
</object>
```

```
*/
```

```
public class MySwingApplet extends JApplet {
```

```
    JButton jbtnAlpha;
```

```
    JButton jbtnBeta;
```

```
    JLabel jlab;
```

```
    // Initialize the applet.
```

```
    public void init() {
```

```
        try {
```

```
            SwingUtilities.invokeLater(new Runnable() {
```

```
                public void run() {
```

```
                    makeGUI(); // initialize the GUI
```

```
                }
```

```
});
} catch(Exception exc) {
    System.out.println("Can't create because of "+ exc);
}
}
// This applet does not need to override start(), stop(),
// or destroy().
// Set up and initialize the GUI.
private void makeGUI( ) {
    // Set the applet to use flow layout.
    setLayout(new FlowLayout());
    // Make two buttons.
    jbtnAlpha = new JButton("Alpha");
    jbtnBeta = new JButton("Beta");
    // Add action listener for Alpha.
    jbtnAlpha.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent le) {
            jlab.setText("Alpha was pressed.");
        }
    });
    // Add action listener for Beta.
    jbtnBeta.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent le) {
            jlab.setText("Beta was pressed.");
        }
    });
    // Add the buttons to the content pane.
    add(jbtnAlpha);
    add(jbtnBeta);
    // Create a text-based label.
    jlab = new JLabel("Press a button.");
    // Add the label to the content pane.
    add(jlab);
}
```

There are two important things to notice about this applet. First, `MySwingApplet` extends `JApplet`. As explained, all Swing-based applets extend `JApplet` rather than `Applet`. Second, the `init()` method initializes the Swing components on the event dispatching thread by setting up a call to `makeGUI()`. Notice that this is accomplished through the use of `invokeAndWait()` rather than `invokeLater()`.

Applets must use `invokeAndWait()` because the `init()` method must not return until the entire initialization process has been completed. In essence, the `start()` method cannot be called until after initialization, which means that the GUI must be fully constructed. Inside `makeGUI()`, the two buttons and label are created, and the action listeners are added to the buttons. Finally, the components are added to the content pane. Although this example is quite simple, this same general approach must be used when building any Swing GUI that will be used by an applet.

Assignment-3

1. Where are the following four methods commonly used?

- 1) public void add(Component c)
- 2) public void setSize(int width,int height)
- 3) public void setLayout(LayoutManager m)
- 4) public void setVisible(boolean)

- A. Graphics class
- B. Component class
- C. Both A & B
- D. None of the above

Answer: Option B

2. Implement the Listener interface and overrides its methods is required to perform in event handling.

- A. True
- B. False

Answer: Option A

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

- A. Window
- B. Frame
- C. Panel
- D. Container

Answer: Option C

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

- A. Labels
- B. Choice
- C. List
- D. Checkbox

Answer: Option C

5. Object which can store group of other objects is called

- A. Collection object
- B. Java object
- C. Package
- D. Wrapper

Answer: Option A

6. JFrame myFrame = new JFrame (); Any command (such as the one listed above) which creates a new object of a specific class (in this case a new JFrame object called myFrame) is generally called a ...

- A. Constructor
- B. Layout manager
- C. Parameter

D. GUI

Answer: Option A

7. What are the names of the list layout managers in Java?

A. Flow Layout Manager

B. Grid Layout Manager

C. Box Layout Manager

D. All of the above

Answer: Option D

8. What is the name of the Swing class that is used for frames?

A. Window

B. Frame

C. JFrame

D. SwingFrame

Answer: Option C

9. What is the name of the Swing class that is used for frames?

A. swing

B. Frame

C. applet

D. SwingFrame

Answer: Option A

10. Which method is used to set the text of a Label object?

A. setText()

B. setLabel()

C. setTextLabel()

D. setLabelText()

Answer: Option A

11. The layout of a container can be altered by using which of the following methods.

A. setLayout(aLayoutManager)

B. layout(aLayoutManager)

C. addLayout(aLayoutManager)

D. setLayoutManager(aLayoutManager)

Answer: Option A

12. Which of these methods can be used to know which key is pressed?

A. getActionEvent()

B. getActionKey()

C. getModifier()

D. getKey()

Answer: Option C

13. Which of these event is generated when a button is pressed?

A. WindowEvent

- B. ActionEvent
- C. WindowEvent
- D. KeyEvent

Answer: Option B

14. Which of these packages contains all the classes and methods required for event handling in java?

- A. Java.applet
- B. Java.awt
- C. Java.awt.event
- D. Java.event

Answer: Option C

15. In Java, what do you call an area on the screen that has nice borders and various buttons along the top border?

- A. A window
- B. A screen
- C. A box
- D. A frame

Answer: Option D

16. Suppose you are developing a Java Swing application and want to toggle between various views of the design area. Which of the views given below are present for the users to toggle?

- A. Design View
- B. Requirements View
- C. Source View
- D. Management View

Answer: Option B

17. The size of a frame on the screen is measured in:

- A. Inches
- B. Nits
- C. Dots
- D. Pixels

Answer: Option D

18. The following specifies the advantages of

It is lightweight.

It supports pluggable look and feel.

It follows MVC (Model View Controller) architecture.

- A. Swing
- B. AWT
- C. Both A & B
- D. None of the above

Answer: Option A

19. Which class provides many methods for graphics programming?

- A. java.awt
- B. java.Graphics
- C. java.awt.Graphics
- D) None of the above

Answer: Option C

20. The ActionListener interface is used for handling action events, For example, it's used by a

- A) JButton
- B) JCheckbox
- C) JMenuItem
- D) All of these

Answer: Option D

21. Which class is used to create a pop-up list of items from which the user may choose?

- A. List
- B. Choice
- C. Labels
- D. Checkbox

Answer: Option B

22. Which class is used for this Processing Method processActionEvent()?

- A. Button, List, MenuItem
- B. Button, Checkbox, Choice
- C. Scrollbar, Component, Button
- D. None of the above

Answer: Option A

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

- A. AbstractButton
- B. ButtonGroup
- C. JButton
- D. ImageIcon

Answer: Option B

Long Answer Questions

1. Explain the advantages of Swing.
2. Write a short note on containers.
3. Briefly explain components.
4. Mention the purpose of different layout managers available in Swing.
5. Explain the use of JTextField and any methods associated with it.
6. Explain the use of JList and any methods associated with it.
7. Explain the purpose of JButton and explain any methods associated with it.
8. With an example explain how to create a swing applet.
9. Explain the use of JCheckBox and any methods associated with it.
10. Give one example for swing program

UNIT-4

Java Data Base Connectivity

Introduction to JDBC

Java JDBC is a java API to connect and execute query with the database. JDBC API uses JDBC drivers to connect with the database.

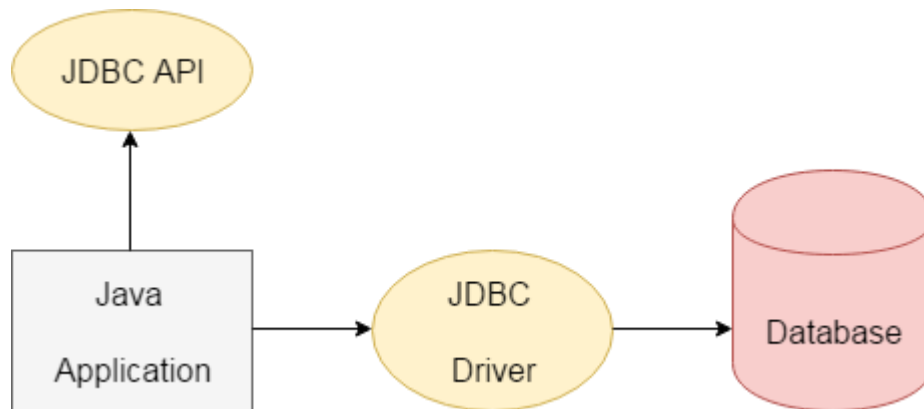


Figure: JDBC Driver

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

Advantages:

- easy to use.

- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

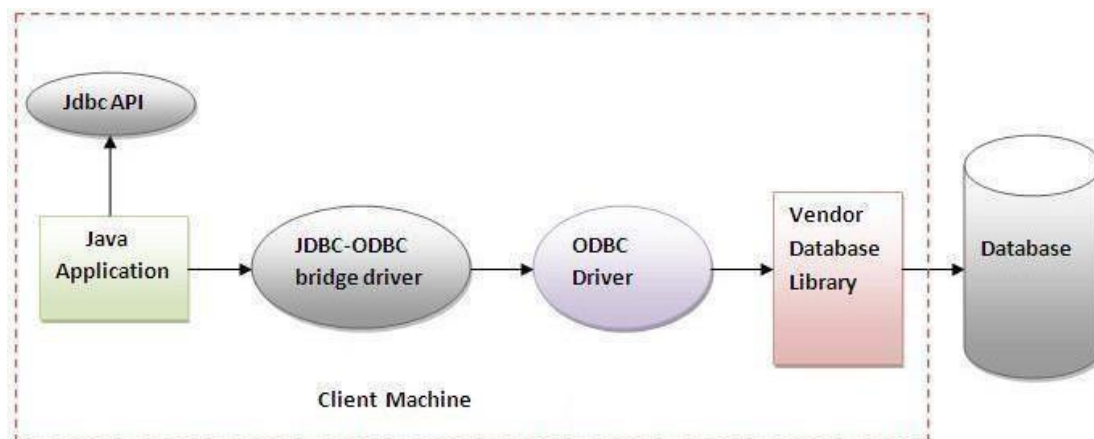


Figure- JDBC-ODBC Bridge Driver

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

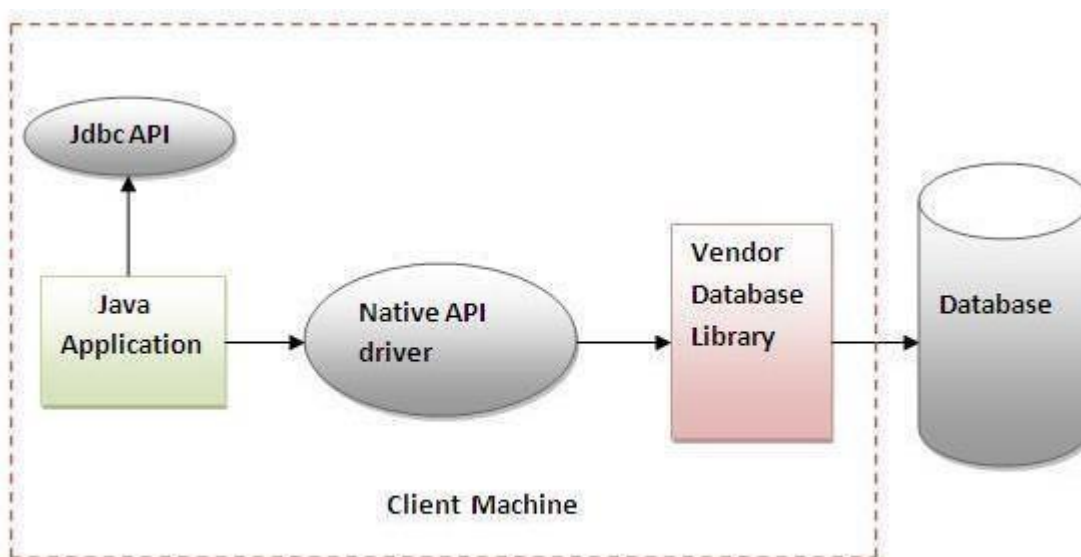


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol Driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

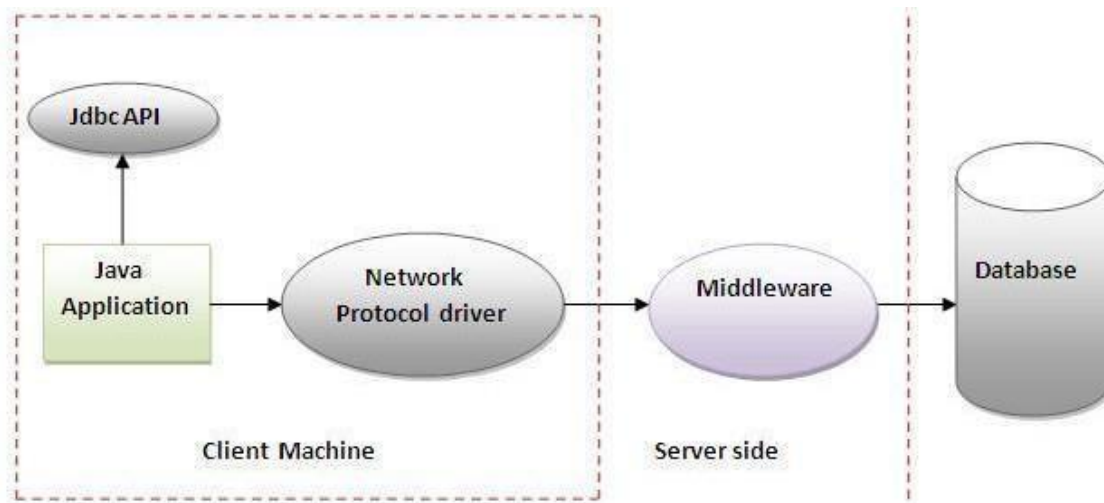


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin Driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

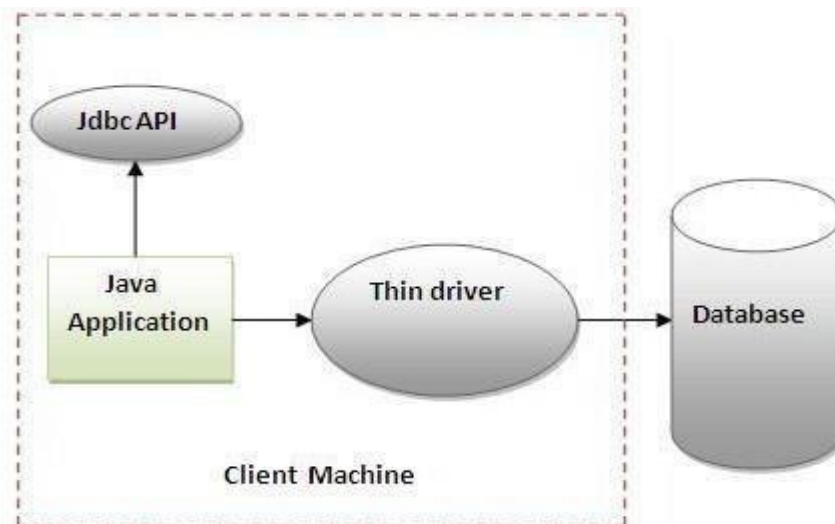


Figure- Thin Driver

+

Database Connectivity Steps

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

1) Register the driver class

The `forName()` method of `Class` is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of `forName()` method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register the `OracleDriver` class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection Object

The `getConnection()` method of `DriverManager` class is used to establish connection with the database.

Syntax for `getConnection()` method

```
1) public static Connection getConnection(String url)throws SQLException
```

```
2) public static Connection getConnection(String url,String name,String password)  
throws SQLException
```

Example to establish connection with Oracle database

1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

3) Create the statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax for createStatement() method

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to wxwcute query

```
ResultSet rs=stmt.executeQuery("select * from emp");  
while(rs.next()){  
System.out.println(rs.getInt(1)+" "+rs.getString(2));  
}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close()
```

Connectivity with MySQL

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

- **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
- **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** 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 and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
- **Username:** The default username for the mysql database is **root**.

Password: Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;  
use sonoo;  
create table emp(id int(10),name varchar(40),age int(3));
```

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;  
class MysqlCon{  
public static void main(String args[]){  
try{  
Class.forName("com.mysql.jdbc.Driver");  
Connection con=DriverManager.getConnection(  
"jdbc:mysql://localhost:3306/sonoo","root","root");  
//here sonoo is database name, root is username and password  
Statement stmt=con.createStatement();  
ResultSet rs=stmt.executeQuery("select * from emp");  
while(rs.next())  
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));  
con.close();  
}catch(Exception e){ System.out.println(e);}  
}  
}
```

Two ways to load the jar file:

1. paste the mysqlconnector.jar file in jre/lib/ext folder

2. set classpath

- 1) paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

- 2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

```
C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar,;
```

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar,; as C:\folder\mysql-connector-java-5.0.8-bin.jar,;.

Connectivity with Access without DSN

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)
2. With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

```
import java.sql.*;
class Test{
public static void main(String ar[]){
try{
    String database="student.mdb";//Here database exists in the current directory

    String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
        DBQ="+ database + ";DriverID=22;READONLY=true";

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection c=DriverManager.getConnection(url);
    Statement st=c.createStatement();
    ResultSet rs=st.executeQuery("select * from login");

    while(rs.next()){
        System.out.println(rs.getString(1));
    }

} catch(Exception ee){System.out.println(ee);}

}}
```

Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

```
import java.sql.*;
class Test{
public static void main(String ar[]){
try{
    String url="jdbc:odbc:mydsn";
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection c=DriverManager.getConnection(url);
    Statement st=c.createStatement();
    ResultSet rs=st.executeQuery("select * from login");

    while(rs.next()){
        System.out.println(rs.getString(1));
    }
} catch(Exception ee){System.out.println(ee);}
}
}
```

DriverManager

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection Interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Commonly use methods of connection interface

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.
2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.
4) public void commit(): saves the changes made since the previous commit/rollback permanent.
5) public void rollback(): Drops all changes made since the previous commit/rollback.
6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql):	is used to execute SELECT query. It returns the object of ResultSet.
2) public int executeUpdate(String sql):	is used to execute specified query, it may be create, drop, insert, update, delete etc.
3) public boolean execute(String sql):	is used to execute queries that may return multiple results.
4) public int[] executeBatch():	is used to execute batch of commands.

Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement();
//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}}
```

ResultSet Interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in

createStatement(int,int) method as well as we can make this object as updatable by:

Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in

	result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");
//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

Assignment 4**MCQ**

Database system compiles query when it is

- A.Executed
- B.Initialized
- C.Prepared
- D.Invoked

Answer: Option C

1. To execute a statement, we invoke method

- A. executeUpdate method
- B. executeRel method
- C. executeStmt method
- D. executeConn method

Answer: Option A

2. Interface ResultSet has a method, getMetaData(), that returns a/an

- A. Tuple
- B. Value
- C. Object
- D. Result

Answer: Option C

3. Older systems allow multiple statements to be executed in a single call, with statements separated by a

- A. Question mark
- B. Colon
- C. Semicolon
- D. \$ sign

Answer: Option C

4. Method on result set 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

Answer: Option C

5. 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 mentioned above

Answer: Option D

6. Which JDBC type represents a "single precision" floating point number that supports seven digits of mantissa?

- A. REAL
- B. DOUBLE
- C. FLOAT
- D. INTEGER

Answer: Option A

7. How many Result sets available with the JDBC 2.0 core API?

- A. 2
- B. 3
- C. 4
- D. 5

Answer: Option B

8. Abbreviate the term UDA?

- A. Unified Data Access
- B. Universal Data Access
- C. Universal Digital Access
- D. Uniform Data Access

Answer: Option B

9. Which method is used to establish the connection with the specified url in a Driver Manager class?

- A. public static void registerDriver(Driver driver)
- B. public static void deregisterDriver(Driver driver)
- C. public static Connection getConnection(String url)
- D. public static Connection getConnection(String url,String userName,String password)

Answer: Option C

10. Which driver Network connection is indirect that a JDBC client makes to a middleware process that acts as a bridge to the DBMS server?

- A. JDBC-Net
- B. JDBC-ODBC bridge
- C. Native API as basis
- D. Native protocol as basis

Answer: Option A

11. JDBC technology-based drivers generally fit into how many categories?

- A. 4
- B. 3
- C. 2
- D. 5

Answer: Option A

12. Which method is used for an SQL statement that is executed frequently?

- A. preparedStatement
- B. prepareCall
- C. createStatement
- D. None of the above

Answer: Option A

13. What is used to execute parameterized query?

- A. Statement interface
- B. PreparedStatement interface
- C. ResultSet interface
- D. None of the above

Answer: Option B

14. Which JDBC product components does the Java software provide?

- A. The JDBC driver manager
- B. The JDBC driver test suite
- C. The JDBC-ODBC bridge
- D. All mentioned above

Answer: Option D

15. JDBC stands for?

- A. Java database connectivity
- B. Java database concept
- C. Java database communications
- D. None of the above

Answer: Option A

16. Which class has traditionally been the backbone of the JDBC architecture?

- A. the JDBC driver manager
- B. the JDBC driver test suite
- C. the JDBC-ODBC bridge
- D. All mentioned above

Answer: Option A

17. How many steps are used to connect any java application using JDBC?

- A. 5
- B. 4
- C. 3
- D. 6

Answer: Option A

18. _____ is an open source DBMS product that runs on UNIX, Linux and Windows.

- A. MySQL
- B. JSP/SQL
- C. JDBC/SQL
- D. Sun ACCESS

Answer: Option A

19. Which JDBC driver Type(s) can you use in a three-tier architecture and if the Web server and the DBMS are running on the same machine?

- A. Type 1 only
- B. Type 2 only
- C. Both Type 3 and Type 4

D. All of Type 1, Type 2, Type 3 and Type 4

Answer: Option D

20. Database system compiles query when it is

- A. Executed
- B. Initialized
- C. Prepared
- D. Invoked

Answer: Option C

21. To execute a statement, we invoke method

- A. executeUpdate method
- B. executeRel method
- C. executeStmt method
- D. executeConn method

Answer: Option A

22. Older systems allow multiple statements to be executed in a single call, with statements separated by a

- A. Question mark
- B. Colon
- C. Semicolon
- D. \$ sign

Answer: Option C

23. Interface ResultSet has a method, getMetaData(), that returns a/an

- A. Tuple
- B. Value
- C. Object
- D. Result

Answer: Option C

24. Method on result set 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

Answer: Option C

Long Answer Questions

1. Explain the architecture of JDBC
2. Explain Database connectivity steps using JDBC
3. Explain Java and MYSQL connection using a code example.
4. Explain the different types of JDBC drivers.
5. Explain about ResultSet Interface with example.
6. Explain about Statement interface with example.
7. Write a note on Thin Driver.
8. Write advantages and disadvantages of JDBC-ODBC bridge driver.
9. Explain the working of a Native-API driver.
10. Explain the working of a Network- Protocol driver.

UNIT-V

Remote Method Invocation

Introduction to RMI

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

There are three entities involved in running a program that uses RMI:

Client: This is the program that you write to access remote methods

Server: This is the program that you write to implement the remote methods - clients connect to the server and request that a method be executed. The remote methods to the client are local methods to the server.

Object registry: This is a program that you use. The object registry runs on a known port (1099 by default). A server, upon starting, registers its objects with a textual name with the object registry. A client, before performing invoking a remote method, must first contact the object registry to obtain access to the remote object.

Two types of objects are useful when dealing with RMI.

A remote object is one whose instances can be used remotely. A handle to a remote object identifies where that object is located and how to contact it remotely (via rmi). When used locally, it works like any other object. When it is passed as a parameter, its handle is passed (as with ordinary java objects).

A serializable object is one whose value can be marshaled. This means that the contents of the object can be represented in a pointer less form as a bunch of bytes and then reconstructed into their respective values as needed. This is useful, for example, in saving the contents of an object into a file and then reading back that object. In the RMI case, it is useful if we want to pass an object as a parameter to a remote method or receive a result from a remote method. In this case, we don't want to pass object handles, because the pointers will make no sense on a different JVM. If an object is defined to implement `java.io.Serializable`, however, passing it as a parameter will allow us to pass the value of the object (marshaled) rather than the handle.

There are two parts to defining a remote class: its interface and the actual class itself. The Remote Interface represents the type of an object handle and tells clients how to invoke remote methods. The remote class defines the interface definition and the remote class. The interface definition must be public, extend the interface `java.rmi.Remote` and every method in the interface must declare that it throws `java.rmi.RemoteException`. The Remote class must implement a Remote interface, that should extend `java.rmi.server.UnicastRemoteObject` class. Objects of this class exist in the address space of the server and can be invoked remotely. (there are other ways of defining a remote class - this is the easiest). Objects in the remote class may have methods that are not in its remote interface. However, these can only be invoked locally.

Understanding Stub and Skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

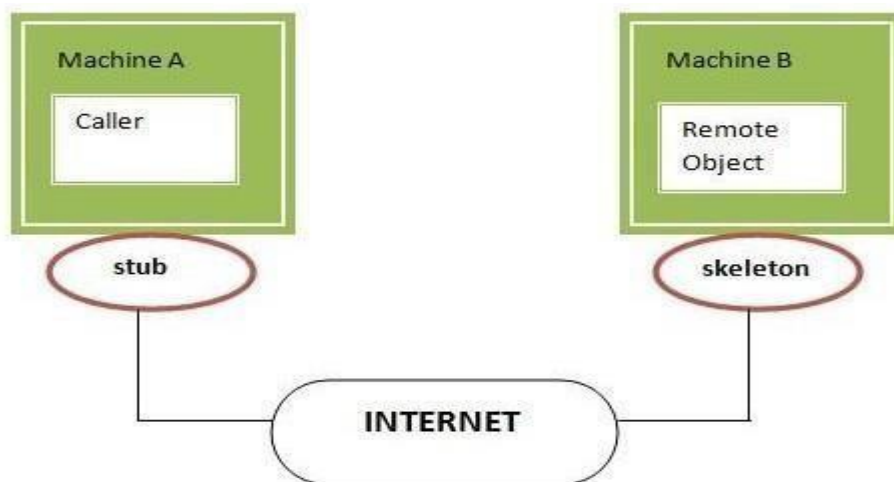


Figure: Stub and Skelton role in RMI

Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

- The application need to locate the remote method
- It need to provide the communication with the remote objects, and
- The application need to load the class definitions for the objects.

The RMI applications have all these features, so it is called the distributed application.

Architecture of Remote Method Invocation

The server must first bind its name to the registry. The client lookup the server name in the registry to establish remote references. The Stub serializing the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.

Each remote object has two interfaces.

Client interface – a stub/proxy of the object

Server interface – a skeleton of the object

When a client invokes a remote method, the call is first forwarded to stub. The stub is responsible for sending the remote call over to the server-side skeleton. The stub opening a socket to the remote server, marshaling the object parameters and forwarding the data stream to the skeleton. A skeleton contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual remote object implementation.

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

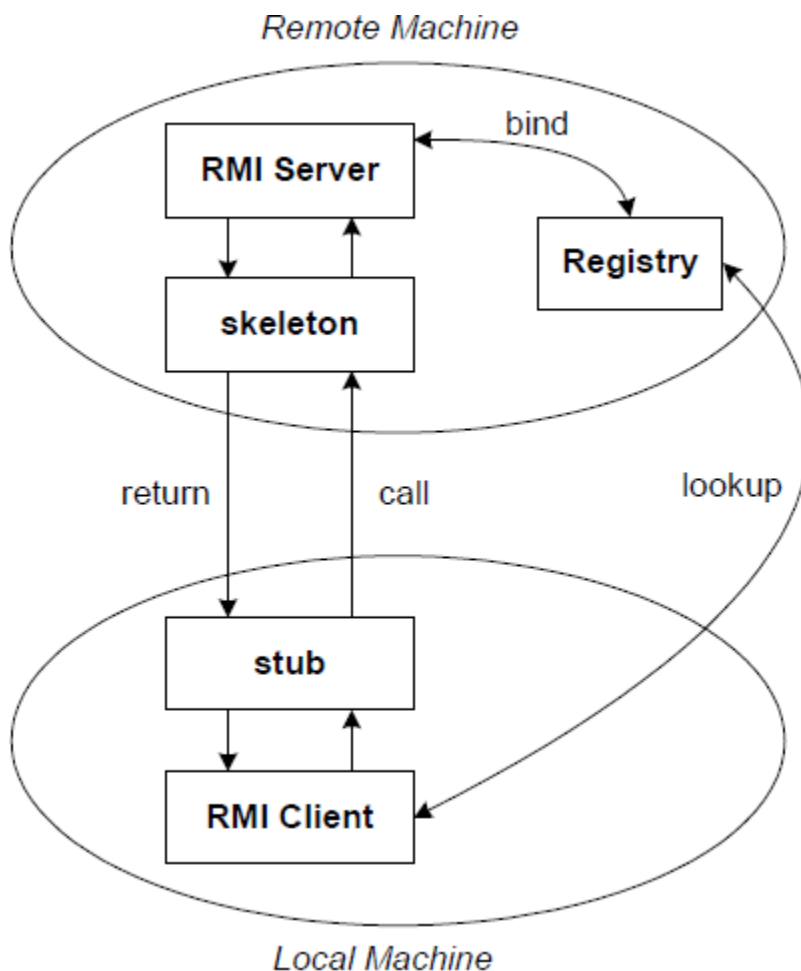


Figure : Architecture of RMI

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result

- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

RMI Server

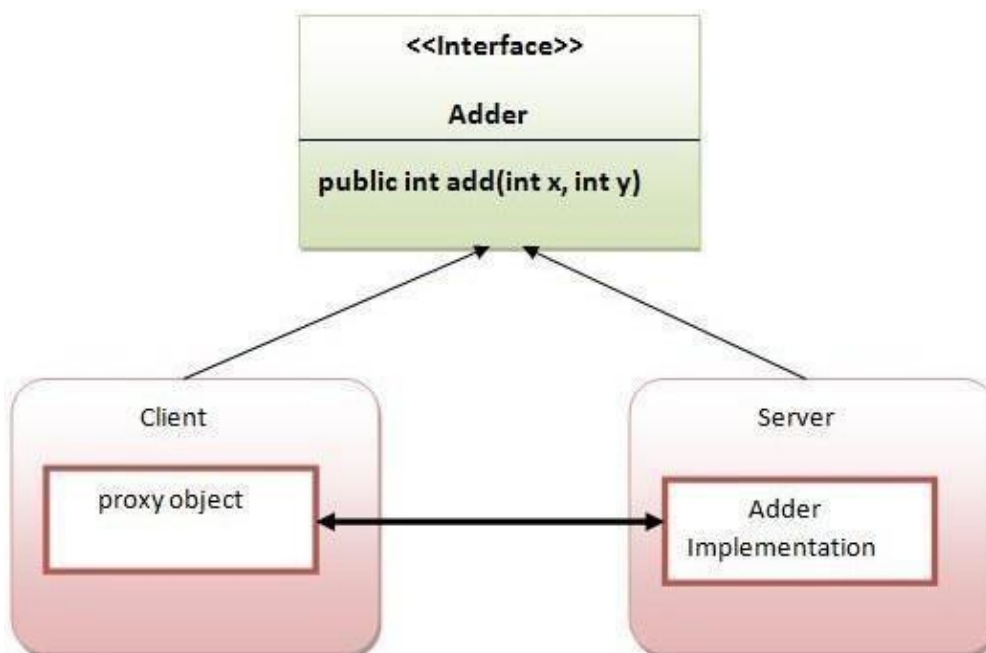
The compute engine server accepts tasks from clients, runs the tasks, and returns any results. The server code consists of an interface and a class. The interface defines the methods that can be invoked from the client. Essentially, the interface defines the client's view of the remote object. The class provides the implementation.

RMI client

The compute engine is a relatively simple program: it runs tasks that are handed to it. The clients for the compute engine are more complex. A client needs to call the compute engine, but it also has to define the task to be performed by the compute engine.

RMI in server side, RMI in client side

In the client side, RMI requires only two programs. The client application needs only two files, remote interface and client application. The server also requires two files as interface implementation and server application. In the rmi application, both client and server interact with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

Either extend the UnicastRemoteObject class,
or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

<pre>public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;</pre>	It returns the reference of the remote object.
<pre>public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;</pre>	It binds the remote object with the given name.

<pre>public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;</pre>	It destroys the remote object which is bound with the given name.
<pre>public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;</pre>	It binds the remote object to the new name.
<pre>public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;</pre>	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){ }
}
}
```

6.6 Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.




1) Create the table

First of all, we need to create the table in the database. Here, we are using Oracle10 database.

CUSTOMER400

TableDataIndexesModelConstraintsGrantsStatisticsUI DefaultsTriggersDependenciesSQL

QueryCount RowsInsert Row

EDIT	ACC_NO	FIRSTNAME	LASTNAME	EMAIL	AMOUNT
	67539876	James	Franklin	franklin1james@gmail.com	500000
	67534876	Ravi	Kumar	ravimalik@gmail.com	98000
	67579872	Vimal	Jaiswal	jaiswalvimal32@gmail.com	9380000
					row(s) 1 - 3 of 3

Download

2) Create Customer class and Remote interface

File: Customer.java

```
package com.javatpoint;
public class Customer implements java.io.Serializable{
    private int acc_no;
    private String firstname,lastname,email;
    private float amount;
    //getters and setters
}
```

Note: Customer class must be Serializable.

File: Bank.java

```
package com.javatpoint;
import java.rmi.*;
import java.util.*;
interface Bank extends Remote{
    public List<Customer> getCustomers()throws RemoteException;
}
```

3) Create the class that provides the implementation of Remote interface

File: BankImpl.java

```
package com.javatpoint;
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
import java.util.*;
class BankImpl extends UnicastRemoteObject implements Bank{
    BankImpl()throws RemoteException{ }

    public List<Customer> getCustomers(){
        List<Customer> list=new ArrayList<Customer>();
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
            PreparedStatement ps=con.prepareStatement("select * from customer400");
            ResultSet rs=ps.executeQuery();
```

```
while(rs.next()){
    Customer c=new Customer();
    c.setAcc_no(rs.getInt(1));
    c.setFirstname(rs.getString(2));
    c.setLastname(rs.getString(3));
    c.setEmail(rs.getString(4));
    c.setAmount(rs.getFloat(5));
    list.add(c);
}

con.close();
}catch(Exception e){System.out.println(e);}
return list;
} //end of getCustomers()
}
```

4) Compile the class rmic tool and start the registry service by rmiregistry tool

5) Create and run the Server

File: MyServer.java

```
package com.javatpoint;
import java.rmi.*;
public class MyServer{
    public static void main(String args[])throws Exception{
        Remote r=new BankImpl();
        Naming.rebind("rmi://localhost:6666/javatpoint",r);
    }
}
```

6) Create and run the Client

File: MyClient.java

```
package com.javatpoint;
import java.util.*;
import java.rmi.*;
public class MyClient{
    public static void main(String args[])throws Exception{
        Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");
        List<Customer> list=b.getCustomers();
        for(Customer c:list){
            System.out.println(c.getAcc_no()+" "+c.getFirstname()+" "+c.getLastname()
            +" "+c.getEmail()+" "+c.getAmount());
        }
    }
}
```

Assignment 6**MCQ**

1. In RMI Architecture which layer Intercepts method calls made by the client/redirects these calls to a remote RMI service?

- A. Stub & Skeleton Layer
- B. Application Layer
- C. Remote Reference Layer
- D. Transport Layer

Answer: Option A

2. Which is an object, acts as a gateway for the client side, all the outgoing requests are routed through it and it resides at the client side and represents the remote object?

- A. Stub
- B. Skeleton
- C. Both A & B
- D. None of the above

Answer: Option A

3. Java supports RMI, RMI Stands for?

- A. Random Method Invocation
- B. Remote Memory Interface
- C. Remote Method Invocation
- D. Random Method Invocation

Answer: Option C

4. An RMI Server is responsible for _____

- A. Creating an instance of the remote object
- B. Exporting the remote object
- C. Binding the instance of the remote object to the RMI registry
- D. All mentioned above

Answer: Option D

5. Abbreviate the term DGC?

- A. Digital Garbage Collection
- B. Distributed Garbage Collection
- C. Distributed Garbage Connection
- D. None of the above

Answer: Option B

6. In RMI, the objects are passed by_____.

- A. Value
- B. Reference
- C. Value and Reference
- D. None of the above

Answer: Option A

7. What are the exceptions which have to be handled in a RMI client program?

- A. RemoteException
- B. NotBoundException

- C. MalformedURLException
- D. All mentioned above

Answer: Option D

8. Which package is used for Remote Method Invocation (RMI)?

- A. java.lang.rmi
- B. java.lang.reflect
- C. java.applet
- D. java.rmi

Answer: Option D

9. Which program obtains a remote reference to one or more remote objects on a server and then invokes methods on them in an RMI application?

- A. Server
- B. Client
- C. Both A & B
- D. None of the above

Answer: Option B

10. Which objects are used by RMI for communicating with the remote object?

- A. Stub
- B. Skeleton
- C. Both A & B
- D. None of the above

Answer: Option C

11. Which method of the Naming class (found in java.rmi) is used to update the RMI registry on the server machine?

- A. rebind ()
- B. lookup()
- C. Both A & B
- D. None of the above

Answer: Option A

12. What is built on top of the socket programming?

- A. EJB
- B. RMI
- C. Both A & B
- D. None of the above

Answer: Option B

13. RMI Architecture consists of how many layers?

- A. 5
- B. 3
- C. 4
- D. 2

Answer: Option C

14. In RMI Distributed object applications need to

- A. Locate remote objects

- B. Communicate with remote objects
- C. Load class definitions for objects that are passed around
- D. All mentioned above

Answer: Option D

15. Which of the function is used to convert string to Number in java program?

- A. to Number()
- B. conString()
- C. valueOf()
- D. toString()

Answer: Option C

16. How many types of protocol implementations does RMI have?

- A. 2
- B. 4
- C. 3
- D. None

Answer: Option C

17. Which of these Exceptions is thrown by a remote method?

- A. RemoteException
- B. InputOutputException
- C. RemoteAccessException
- D. RemoteInputOutputException

Answer: Option A

18. RMI uses which protocol on top of TCP/IP (an analogy is HTTP over TCP/IP)?

- A. Java Remote Method Protocol (JRMP)
- B. Internet Inter-ORB Protocol (IIOP)
- C. Jinni Extensible Remote Invocation (JERI)
- D. All mentioned above

Answer: Option A

19. What is the built on top of the socket programming?

- A. EJB
- B. RMI
- C. Both A & B
- D. None of the above

Answer: Option B

20. In RMI Distributed object applications need to

- A. Locate remote objects
- B. Communicate with remote objects
- C. Load class definitions for objects that are passed around
- D. All mentioned above

Answer: Option D

21. RMI facilitates communication between:

- A. 2 Java programs

- B. 2 JVMs
- C. 2 Machines
- D. DNS

Answer: Option A

22. Which of the following is the tool used to generate the stub and the skeleton?

- A. javac
- B. MI Registry
- C. rmic
- D. Java

Answer: Option C

23. RMI uses a layered architecture; each of the layers could be enhanced or replaced without affecting the rest of the system.

- A. True
- B. False

Answer: Option A

24. RMI is a server-side component; It is not required to be deployed on the server.

- A. True
- B. False

Answer: Option B

25. Which method in naming class specifies a name to the remote object?

- A. bind(string name)
- B. rebind(string name)
- C. Both A & B
- D. None of the above

Answer: Option A

Long Answer Questions

1. Write a short note on stub and skeleton in RMI.
2. Explain architecture of RMI with the help of a diagram.
3. Explain RMI in Server side and Client side.
4. Explain meaningful example of RMI application with database.
5. Explain accessing database without using DSN from Java application.
6. Write a short note on Stub.
7. Write a short note on Skeleton.
8. Create and run the server application with example.
9. Create and run the client application with example.
10. Create and run the interface and implementation code .

Lab List

Advanced Java Programming PART-A

1. Create an applet to implement simple calculator. The integer data are to be entered through text box and the operation that is to be performed (operator) (+,-,*,/) to be given through command buttons. When the user press the compute button result should be displayed. Do the validation for empty text box for numbers.

Handle -Division by Zero

2. Write an applet program to accept the employee name, employee number and basic salary as parameters. Find the gross and net salaries on the following conditions.

if Salary \leq 20000

D.A is 40% Salary

H.R.A is 10% Salary.

P.F 12% of Gross PT is Rs.100

if Salary $>$ 20000

D.A is 50% of salary

H.R.A 15% of salary

P.F 12% of Gross PT is Rs.150

Gross = basic salary + D.A + HRA, Net = Gross - PT - PF

3. Write a program to demonstrate passing parameters to Applet.

4. Write a program to copy a character from one file to another file.

5. Write a program to reading and writing using random access file.

6. Using the swing components, design the frame for shopping a book that accepts book code, book name and price. Calculate the discount on code as follows.

Code

101

102

103

Any other 5%

Find the discount amount and Net bill amount. Display the bill

7. Using the Swing components, design the form for an electricity office for accepting meter no. customer name, previous reading, and current reading . Use data validation to see that current reading is more than previous reading. Produce the bill in neat format.

Bill is calculates as follows:

Compute total number of units consumed and total amount to be paid by each consumer the condition is

If unit consumed is \leq 150 charge is 200.

For next 50 units Rs. 1.50 per units.

For next 100 units charge= Rs 2.00 per unit.

For next additional units charge is Rs. 3.00 per unit or Rs. 500 whichever is maximum.

PART-B

1. Write a program to retrieve data from personal telephone directory in the form of a database table. When the user hits a character the names which start with the character and the telephone number must appear.(For example if names are Anil, Arjun, Ashwin, Ram, Raj, Deepak etc, When 'A' is hit, Anil, Arjun, Ashwin must be displayed with respective no.s, When A is continued with 'r', only Arjun detail to be shown from the above list.)
2. Write a program to create a student database with fields Stud_no, Stud_name, Class, Mark1 and Mark2. Perform these operations a) Input the student information from the keyboard . b) Display Specific student information. C) Delete specific student information.
3. Write a Java class called Temp converter with methods for calculating conversion operations. Have this class as a servant and create a server program and register in the RMI registry. Write a client program to invoke these remote methods of the servant and do the calculations. Accepts input interactively.
4. Write a java class called Tax with methods for calculating Income Tax. Have this class as a servant and create a server program and register in the RMI registry. Write a client program to invoke these remote methods of the servant and do the calculations. Accept inputs interactively.
5. Write a Java class called Simple Interest with methods for calculating Simple Interest. Have this class as a servant and create a server program and register in the RMI registry. Write a client program to invoke these remote methods of the servant and do the calculations. Accept inputs at command prompt.
6. Write a program to convert dollar to rupees.
7. Write a JDBC program to retrieve bank balance of given account number.

1. Q: Write an applet program to display text on the screen by parsing parameters.

```
general@general-desktop ~ $ cd supriya
```

```
general@general-desktop ~/supriya$ gedit HelloJava2.java
```

```
import java.awt.*;
import java.applet.*;
public class HelloJava2 extends Applet
{
    String str;
    public void init()
    {
        str=getParameter("string");
        if(str==null)
            str="java";
        str="hello"+str;
    }
}
```

```
public void paint(Graphics g)
{
    g.drawString(str,10,100);
}
```

```
general@general-desktop ~/supriya$ javac HelloJava2.java
general@general-desktop ~/supriya$ gedit HelloJava2.html
```

```
<html>
  <body>
    <APPLET
      code=HelloJava2.class
      width=100
      height=400>
    <PARAM
      name="string"
      value="applet">
    </PARAM>
    </APPLET>
  </body>
</html>
```

```
general@general-desktop ~/supriya$ appletviewer HelloJava2.html
```

OUTPUT :



2. Q: Create an applet to implement simple calculator. The integer data are to be entered through the text box. And the operation that is to be performed (operator: +, -, *, /) to be given through the command buttons. Then the user press the compute button results should be displayed. Through the validation empty text box for the number and the division by zero.

```
general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ gedit Cal.java
```

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Cal extends Applet implements ActionListener
{
    int num1,num2;
    int result=0;
    String r;
    TextField txt1;
    TextField txt2;
    Label l1;
    Label l2;
    Label l3;
    Label l4;
    Button plus;
    Button minus;
    Button mul;
    Button div;
    Button compute;
    Button clear;
    public void init()
    {
        txt1=new TextField();
        txt2=new TextField();
        l1=new Label("Enter the first number:");
        l2=new Label("Enter the second number:");
        l3=new Label("Result");
        l4=new Label(" ");
        plus=new Button("+");
        minus=new Button("-");
        mul=new Button("*");
        div=new Button("/");
        compute=new Button("compute");
        setLayout(null);
        add(txt1);
        add(txt2);
        add(l1);
        add(l2);
        add(l3);
        add(l4);
        add(plus);
```

```
        add(minus);
        add(mul);
        add(div);
        add(compute);
        txt1.setBounds(300,20,80,20);
        txt2.setBounds(300,50,80,20);
        plus.setBounds(100,80,40,20);
        minus.setBounds(200,80,40,20);
        mul.setBounds(300,80,40,20);
        div.setBounds(400,80,40,20);
        l1.setBounds(100,20,200,25);
        l2.setBounds(100,50,200,25);
        l3.setBounds(100,100,100,25);
        l4.setBounds(200,100,200,25);
        compute.setBounds(290,150,80,25);
        compute.addActionListener(this);
        plus.addActionListener(this);
        minus.addActionListener(this);
        mul.addActionListener(this);
        div.addActionListener(this);
        repaint();
    }
    public void actionPerformed(ActionEvent d)
    {
        try
        {
            num1=Integer.parseInt(txt1.getText());
            num2=Integer.parseInt(txt2.getText());
            if(d.getSource()==plus)
            {
                result=num1+num2;
            }
            if(d.getSource()==minus)
            {
                result=num1-num2;
            }
            if(d.getSource()==mul)
            {
                result=num1*num2;
            }
            if(d.getSource()==div)
            {
                try
                {
                    result=num1/num2;
                }
                catch(ArithmeticException e)
                {
                    l4.setText("Division by zero");
                }
            }
        }
    }
}
```

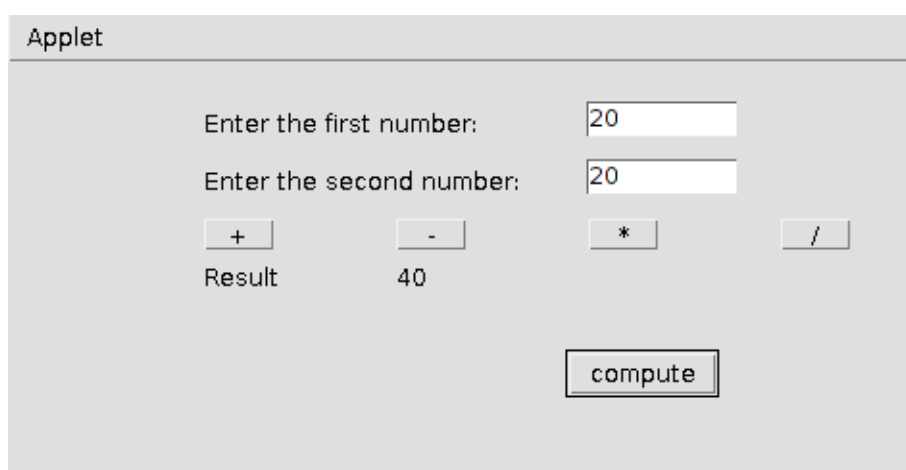
```
    }  
    if(d.getSource()==compute)  
    {  
        r=Integer.toString(result);  
        l4.setText(r);  
    }  
}  
catch(NumberFormatException e)  
{  
    l4.setText("Invalid number");  
}  
}  
}
```

```
general@general-desktop ~/supriya$ javac Cal.java  
general@general-desktop ~/supriya$ gedit Cal.html
```

```
<html>  
  <body>  
    <APPLET  
      code=Cal.class  
      width=300  
      height=600>  
    </APPLET>  
  </body>  
</html>
```

```
general@general-desktop ~/supriya$ appletviewer Cal.html
```

OUTPUT :



3. Q: Write an applet to accept employee name, employee number and basic salary as parameter. Find the gross and net salary on the following condition. If salary ≤ 20000

DA is 40% of salary
HRA is 10% of salary
PF is 12% of gross
PT is Rs.100

If salary is >20000
DA is 50% of salary
HRA is 15% of salary
PF is 12% of gross
PT is Rs.150
Gross=basic salary+DA+HRA
NET=Gross-PT-PF

```
general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ gedit Employee.java
```

```
import java.awt.*;
import java.applet.*;
public class Employee extends Applet
{
    String ename;
    int enumber;
    double bsalary;
    double netsalary;
    double grosssalary;
    double hra,da,pt,pf;
    public void init()
    {
        ename=getParameter("param1");
        enumber=Integer.parseInt(getParameter("param2"));
        bsalary=Double.parseDouble(getParameter("param3"));
        calculate();
    }
    public void paint(Graphics g)
    {
        g.drawString("Employee name:"+ename,20,20);
        g.drawString("Employee number:"+enumber,20,40);
        g.drawString("Basic salary:"+bsalary,20,60);
        g.drawString("da:"+da,20,80);
        g.drawString("hra:"+hra,20,100);
        g.drawString("pf:"+pf,20,120);
        g.drawString("pt:"+pt,20,140);
        g.drawString("gross:"+grosssalary,20,160);
        g.drawString("netsalary:"+netsalary,20,180);
    }
    public void calculate()
    {
        if(bsalary<=20000)
        {
            da=0.40*bsalary;
```

```
        hra=0.10*bsalary;
        grosssalary=da+hra+bsalary;
        pf=0.12*grosssalary;
        pt=100;
        netsalary=grosssalary-(pf+pt);
    }
    if(bsalary>20000)
    {
        da=0.50*bsalary;
        hra=0.15*bsalary;
        grosssalary=da+hra+bsalary;
        pf=0.12*grosssalary;
        pt=150;
        netsalary=grosssalary-(pf+pt);
    }
}
```

```
general@general-desktop ~/supriya$ javac Employee.java
general@general-desktop ~/supriya$ gedit Employee.html
```

```
<html>
  <body>
    <APPLET
      code=Employee.class
      width=300
      height=500>
      <param name="param1" value="Anas">
      <param name="param2" value="1001">
      <param name="param3" value="25000">
      </param>
    </APPLET>
  </body>
</html>
```

```
general@general-desktop ~/supriya$ appletviewer Employee.html
```

OUTPUT :

Applet

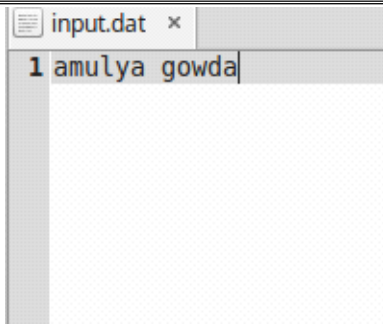
```
Employee name:Anas
Employee number:1001
Basic salary:25000.0
da:12500.0
hra:3750.0
pf:4950.0
pt:150.0
gross:41250.0
netsalary:36150.0
```

4. Q: Write a program to copy a character from one file to another file.

general@general-desktop ~/supriya\$ gedit Copychar.java

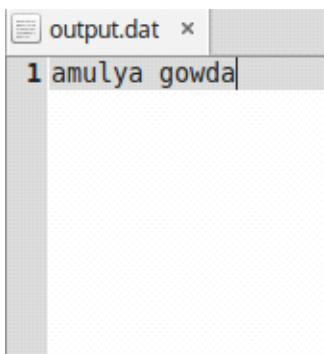
```
import java.io.*;
class Copychar
{
    public static void main(String args[])
    {
        File infile=new File("input.dat");
        File outfile=new File("output.dat");
        FileReader ins=null;
        FileWriter outs=null;
        try
        {
            ins=new FileReader(infile);
            outs=new FileWriter(outfile);
            int ch;
            while((ch=ins.read())!=-1)
            {
                outs.write(ch);
            }
        }
        catch(IOException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
        finally
        {
            try
            {
                ins.close();
                outs.close();
            }
            catch(IOException e)
            {}
        }
    }
}
```

general@general-desktop ~/supriya\$ javac Copychar.java
general@general-desktop ~/supriya\$ gedit input.dat



```
general@general-desktop ~/supriya$ java Copychar  
general@general-desktop ~/supriya$ gedit output.dat
```

OUTPUT :



5. Q: Write a program for reading and writing using a random access file.

```
general@general-desktop ~ $ cd supriya  
general@general-desktop ~/supriya$ gedit Random.java
```

```
import java.io.*;  
class Random  
{  
    public static void main(String args[])  
    {  
        RandomAccessFile file=null;  
        try  
        {  
            file=new RandomAccessFile("rand.dat","rw");  
            file.writeInt(55);  
            file.writeChar('A');  
            file.writeDouble(3.14);  
            file.seek(0);  
            System.out.println(file.readInt());  
            System.out.println(file.readChar());  
            System.out.println(file.readDouble());  
            file.seek(4);  
        }  
    }  
}
```

```

        System.out.println(file.readChar());
        file.seek(file.length());
        file.writeBoolean(true);
        file.seek(5);
        System.out.println(file.readBoolean());
        file.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
        System.exit(-1);
    }
}
}

```

```

general@general-desktop ~/supriya$ javac Random.java
general@general-desktop ~/supriya$ java Random

```

OUTPUT :

```

55
A
3.14
A
true

```

6. Q: Using the swing components design the frame for shopping a book that accepts a book code,name and price.Calculate the discount on codes as follow,

<u>code</u>	<u>Discount rate</u>
101	15%
102	20%
103	25%
Any other	5%

Find the discount amount and net bill amount display the bill

```

general@general-desktop ~/supriya$ gedit Shop.java

```

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Shop implements ActionListener
{
    JLabel lcode;
    JLabel lname;
    JLabel lprice;
    JLabel ldiscount;
    JLabel lnetbill;
    JTextField tcode;
    JTextField tname;

```

```

JTextField tprice;
JTextField tdiscount;
JTextField tnetbill;
JButton bill;
Shop()
{
    JFrame jfrm=new JFrame("Shopping Book");
    jfrm.setLayout(null);
    jfrm.setSize(300,300);
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    lcode=new JLabel("Book code");
    tcode=new JTextField();
    lname=new JLabel("Book name");
    tname=new JTextField();
    lprice=new JLabel("Price");
    tprice=new JTextField();
    ldiscount=new JLabel("Discount");
    tdiscount=new JTextField();
    lnetbill=new JLabel("Net Bill");
    tnetbill=new JTextField();
    bill=new JButton("Bill");
    lcode.setBounds(20,20,100,20);
    tcode.setBounds(100,20,50,20);
    lname.setBounds(20,40,100,20);
    tname.setBounds(100,40,50,20);
    lprice.setBounds(20,60,100,20);
    tprice.setBounds(100,60,50,20);
    ldiscount.setBounds(20,80,100,20);
    tdiscount.setBounds(100,80,50,20);
    lnetbill.setBounds(20,100,100,20);
    tnetbill.setBounds(100,100,50,20);
    bill.setBounds(300,120,200,50);
    jfrm.add(lcode);
    jfrm.add(tcode);
    jfrm.add(lname);
    jfrm.add(tname);
    jfrm.add(lprice);
    jfrm.add(tprice);
    jfrm.add(ldiscount);
    jfrm.add(tdiscount);
    jfrm.add(lnetbill);
    jfrm.add(tnetbill);
    jfrm.add(bill);
    bill.addActionListener(this);
    jfrm.setVisible(true);
    tdiscount.setEditable(false);
    tnetbill.setEditable(false);
}
public void actionPerformed(ActionEvent e)
{

```

```
double discount;
String discountamt,netbillamt;
double price;
double netbill;
try
{
    int codeval=Integer.parseInt(tcode.getText());
    if(codeval==101)
        discount=15;
    else if(codeval==102)
        discount=20;
    else if(codeval==103)
        discount=25;
    else
        discount=5;
    discountamt=Double.toString(discount);
    if(!(tcode.getText().equals(" ")) && !(tname.getText().equals(" ")) &&
!(tprice.getText().equals(" ")))
        tdiscount.setText(discountamt+"%");
    price=Integer.parseInt(tprice.getText());
    netbill=price-(price*discount/100);
    netbillamt=Double.toString(netbill);
    tnetbill.setText(netbillamt);
}
catch(NumberFormatException d)
{
    tnetbill.setText("Invalid number");
}
}
public static void main(String args[])
{
    new Shop();
}
}
```

```
general@general-desktop ~/supriya$ javac Shop.java
general@general-desktop ~/supriya$ javac Shop
```

OUTPUT:

Book code	<input type="text" value="101"/>
Book name	<input type="text" value="java"/>
Price	<input type="text" value="1200"/>
Discount	<input type="text" value="15.0%"/>
Net Bill	<input type="text" value="1020.0"/>

7. Q: Using swing components, design the form for an electricity office for accepting meter number, customer name, previous reading and current reading. Use data validation to see that current reading is more than the previous reading. Produce the bill in need format. Bill is calculated as follows. Compute total number of unit consume and total amount to be paid by each consumer the condition is,

If unit consume ≤ 150 , charge is 200.

For next 50 units Rs.1.5 /unit

For next 100 units Rs.2 /unit

For next additional units charge is Rs.3 /unit or Rs.500 which ever is maximum

general@general-desktop ~/supriya\$ gedit Electricity.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Electricity implements ActionListener
{
    JLabel lmeter;
    JTextField tmeter;
    JLabel lname;
    JTextField tname;
    JLabel lpreading;
    JTextField tpreading;
    JLabel lcreading;
    JTextField tcreading;
    JLabel lamt;
    JTextField tamt;
    JButton bill;
    Electricity()
    {
        JFrame jfrm=new JFrame("Electricity Bill");
        jfrm.setLayout(null);
        jfrm.setSize(300,300);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        lmeter=new JLabel("Meter number:");
        tmeter=new JTextField();
        lname=new JLabel("Customer name:");
        tname=new JTextField();
        lpreading=new JLabel("Previous reading:");
        tpreading=new JTextField();
        lcreading=new JLabel("current reading:");
        tcreading=new JTextField();
        lamt=new JLabel("Amount to be paid:");
        tamt=new JTextField();
        bill=new JButton("Bill");
```



```
lmeter.setBounds(20,20,200,20);
tmeter.setBounds(200,20,50,20);
lname.setBounds(20,40,200,20);
tname.setBounds(200,40,50,20);
lpreading.setBounds(20,60,200,20);
tpreading.setBounds(200,60,50,20);
lcreading.setBounds(20,80,200,20);
tcreading.setBounds(200,80,50,20);
lamt.setBounds(20,100,200,20);
tamt.setBounds(200,100,50,20);
bill.setBounds(100,200,100,50);

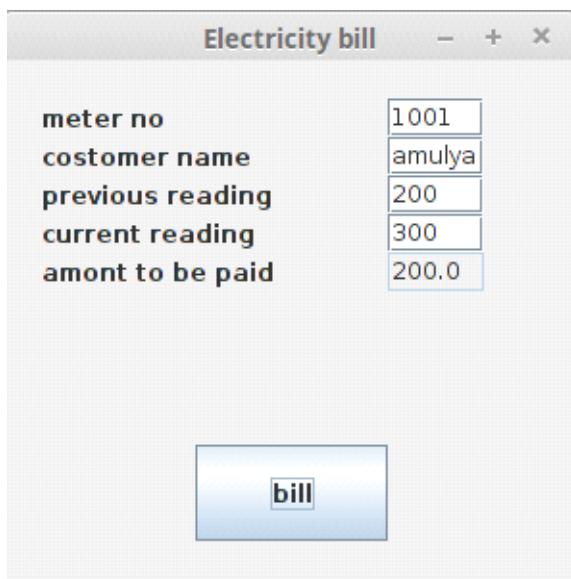
jfrm.add(lmeter);
jfrm.add(tmeter);
jfrm.add(lname);
jfrm.add(tname);
jfrm.add(lpreading);
jfrm.add(tpreading);
jfrm.add(lcreading);
jfrm.add(tcreading);
jfrm.add(lamt);
jfrm.add(tamt);
jfrm.add(bill);
bill.addActionListener(this);
jfrm.setVisible(true);
tamt.setEditable(false);
}
public void actionPerformed(ActionEvent e)
{
    int unit,creading,preading;
    double charge=0;
    String netamt;
    try
    {
        preading=Integer.parseInt(tpreading.getText());
        creading=Integer.parseInt(tcreading.getText());
        if(creading>preading)
        {
            unit=creading-preading;
            if(unit<=150)
                charge=200;
            else if(unit>150&&unit<=200)
                charge=200+(unit*1.5);
            else if(unit>200&&unit<=300)
                charge=200+unit*2;
            else
                charge=Math.max(200+unit*3,500);
            netamt=Double.toString(charge);
            tamt.setText(netamt);
        }
    }
}
```

```
        else
        {
            tamt.setText("Invalid number");
        }
    }

    catch(NumberFormatException d)
    {
        tamt.setText("Invalid number");
    }
}
public static void main(String args[])
{
    new Electricity();
}
}
```

```
general@general-desktop ~/supriya$ javac Electricity.java
general@general-desktop ~/supriya$ javac Electricity
```

OUTPUT:



Label	Value
meter no	1001
costomer name	amulya
previous reading	200
current reading	300
amont to be paid	200.0

bill

1. Q: Write a programe to retrive data from personal telephone directory in the form of a database table. When the user is hits a character the names start with the character and telephone number must appliar (for eg: if names are Anil, Arjun, Ashwin, Ram, Raj, Deepak, etc. When 'A' is hit, Anil, Arjun, Ashwin must be display with the respective number, when 'A' is continued with 'r' only Arjun details to be shown from the above list).

```
mysql -u root -p
```

Enter password:

mysql> use telephone2

Database changed

mysql> create table tele1(cname varchar(50),phone bigint);

Query OK, 0 rows affected (0.10 sec)

mysql> SHOW TABLES;

```
+-----+
| Tables_in_telephone2 |
+-----+
| tele1                |
+-----+
1 row in set (0.00 sec)
```

mysql> insert into tele1 values('Anil',6472890162);

Query OK, 1 row affected (0.07 sec)

mysql> insert into tele1 values('Arjun',2356997289);

Query OK, 1 row affected (0.07 sec)

mysql> insert into tele1 values('Ashwin',7484930201);

Query OK, 1 row affected (0.06 sec)

mysql> insert into tele1 values('Ram',8439320289);

Query OK, 1 row affected (0.06 sec)

mysql> insert into tele1 values('Raj',7393020120);

Query OK, 1 row affected (0.06 sec)

mysql> insert into tele1 values('Deepak',7382190020);

Query OK, 1 row affected (0.07 sec)

mysql> exit;

Bye

general@general-desktop ~/supriya\$ gedit Telephone1.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.io.*;
public class Telephone1
{
    static
    {
        try
        {
```

```
        Class.forName("com.mysql.jdbc.Driver");
    }
    catch(ClassNotFoundException cnf)
    {
        System.out.println("Driver could not be
loaded:"+cnf);
    }
}
public static void main(String args[])
{
    String
ConnectionUrl="jdbc:mysql://localhost:3306/telephone2";
    String dbUser="root";
    String dbPwd="password";
    DataInputStream in=new DataInputStream(System.in);
    String ch;
    int flag=0;
    try
    {
        Connection
con=DriverManager.getConnection(ConnectionUrl,dbUser,dbPwd);
        Statement st=con.createStatement();
        System.out.println("Enter the first few
character of the name of user :");
        ch=in.readLine();
        ResultSet rec=st.executeQuery("Select * from
tele1 where cname like '"+ch+"%'");
        while(rec.next())
        {
            System.out.print(rec.getString("cname"));

            System.out.println("\t"+rec.getString("phone"));
            flag=1;
        }
        if(flag==0)
            System.out.print("No record found");
    }
    catch(SQLException sqle)
    {
        System.out.println("SQLException"+sqle);
    }
    catch(IOException io)
    {
        System.out.println("SQLException"+io);
    }
}
}
```

OUTPUT :

general@general-desktop ~/supriya\$ javac Telephone1.java
Note: Telephone1.java uses or overrides a deprecated API.

```
Note: Recompile with -Xlint:deprecation for details.
general@general-desktop ~/supriya$ java Telephone1
Enter the first few character of the name of user :
A
Anil 6472890162
Arjun      2356997289
Ashwin     7484930201
```

```
Enter the first few character of the name of user :
ar
Arjun      2356997289
```

2. Q: Write a programme to create a student database with fields student number, student name, class, mark1 & mark2. Perform these operations.

- a) Input the student information from the keyboard.**
- b) Display specific student information.**
- c) Delete specific student information.**

```
mysql -u root -p
Enter password:
mysql> create database student1;
Query OK, 1 row affected (0.00 sec)

mysql> use student1
Database changed

mysql> create table stud(stud_no int(4), name varchar(50), class varchar(10), mark1
int(2), mark2 int(2));
Query OK, 0 rows affected (0.12 sec)

mysql> insert into stud values(1001, 'Arjun', 'BCA', 79, 87);
Query OK, 1 row affected (0.07 sec)

mysql> insert into stud values(1002, 'Arun', 'BCA', 97, 78);
Query OK, 1 row affected (0.05 sec)

mysql> insert into stud values(1003, 'Anju', 'BCA', 97, 67);
Query OK, 1 row affected (0.06 sec)

mysql> insert into stud values(1004, 'Malu', 'BCA', 90, 78);
Query OK, 1 row affected (0.08 sec)

mysql> show tables;
+-----+
| Tables_in_student1 |
+-----+
| stud                |
+-----+
```

1 row in set (0.00 sec)

```
mysql> select *from stud;
```

stud_no	name	class	mark1	mark2
1001	Arjun	BCA	79	87
1002	Arun	BCA	97	78
1003	Anju	BCA	97	67
1004	Malu	BCA	90	78

4 rows in set (0.00 sec)

```
mysql> exit
```

Bye

```
general@general-desktop ~/supriya$ gedit Student.java
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.io.*;
public class Student
{
    static
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
        }

        catch(ClassNotFoundException cnf)
        {
            System.out.println("Driver could not be loaded:"+cnf);
        }
    }
    public static void main(String args[])
    {
        String ConnectionUrl="jdbc:mysql://localhost:3306/student2";
        String dbUser="root";
        String dbPwd="password";
        DataInputStream in=new DataInputStream(System.in);
        int no,m1,m2,n,ch,num,i;
        String name,clas;
        Connection con;
        try
        {
            con=DriverManager.getConnection(ConnectionUrl,dbUser,dbPwd);
            Statement st=con.createStatement();
```

```

do
{
    i=0;
    System.out.println("\n Menu \n 1.Input Student Information \n
2.Delete specific student information \n 3.Display \n 4.Exit");
    System.out.println("Enter your choice :");
    ch=Integer.parseInt(in.readLine());
    switch(ch)
    {
        case 1:
            System.out.println("How many students you want
to enter:");

            n=Integer.parseInt(in.readLine());
            while(i<n)
            {
                System.out.println("Enter the student
number:");

                no=Integer.parseInt(in.readLine());
                System.out.println("Enter the student
name:");

                name=in.readLine();
                System.out.println("Enter the student
class:");

                clas=in.readLine();
                System.out.println("Enter the student
marks1:");

                m1=Integer.parseInt(in.readLine());
                System.out.println("Enter the student
marks2:");

                m2=Integer.parseInt(in.readLine());
                num=st.executeUpdate("insert into stud
values("+no+", '"+name+"', '"+clas+"', '"+m1+"', '"+m2+"')");
                i++;
            }
            break;
        case 2:
            System.out.println("Enter the student number
whose details you want to delete:");

            no=Integer.parseInt(in.readLine());
            st.executeUpdate("delete from stud where
stud_no="+no);

            break;
        case 3:
            System.out.println("Enter the student number
whose details you want:");

            num=Integer.parseInt(in.readLine());
            ResultSet rec=st.executeQuery("Select * from
stud where stud_no="+num);

            if(rec.next())
            {

```

```

        number:"+rec.getString("stud_no"));
        name:"+rec.getString("name"));
        class:"+rec.getString("class"));
        mark1:"+rec.getString("mark1"));
        mark2:"+rec.getString("mark2"));
    }
    else

        System.out.println("No record found");

        break;
    case 4:
        System.exit(0);
    }
}

while(ch<=4);
}
catch(SQLException sqle)
{
    System.out.println("SQLException" +sqle);
}
catch(IOException io)
{
    System.out.println("SQLException" +io);
}
}
}

```

OUTPUT :

```

general@general-desktop ~/supriya$ javac Student.java
Note: Student.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
general@general-desktop ~/supriya$ java Student

```

Menu

1. Input Student Information
2. Delete specific student information
3. Display
4. Exit

Enter your choice :

3

Enter the student number whose details you want:

Student name:Arjun
Student class:BCA
Student mark1:79
Student mark2:87

Menu

- 1.Input Student Information
- 2.Delete specific student information
- 3.Display
- 4.Exit

Enter your choice :

1

How many students you want to enter:

1

Enter the student number:

1006

Enter the student name:

amith

Enter the student class:

bca

Enter the student marks1:

67

Enter the student marks2:

78

Menu

- 1.Input Student Information
- 2.Delete specific student information
- 3.Display
- 4.Exit

Enter your choice :

3

Enter the student number whose details you want:

1006

Student number:1006

Student name:amith

Student class:bca

Student mark1:67

Student mark2:78

Menu

- 1.Input Student Information
- 2.Delete specific student information
- 3.Display
- 4.Exit

Enter your choice :

2

Enter the student number whose details you want to delete:

1006

Menu

1. Input Student Information
2. Delete specific student information
3. Display
4. Exit

Enter your choice :

3

Enter the student number whose details you want:

1006

No record found

Menu

1. Input Student Information
2. Delete specific student information
3. Display
4. Exit

Enter your choice :

4

3. Q: Write a java class called temp converter with methods for calculating conversion operations. Have this class as a servent and create a server program and register in the RMI registry. Write a client program to invoke these remote methods of the servent and do the calculations. Accepts input interactively.

//Interface code

```
general@general-desktop ~/supriya$ gedit tempintf.java
```

```
import java.rmi.*;
public interface tempintf extends Remote
{
    double ctof(double c) throws RemoteException;
    double ftoc(double f) throws RemoteException;
}
```

```
general@general-desktop ~/supriya$ javac tempintf.java
```

//Implementation code

```
general@general-desktop ~/supriya$ gedit tempimp.java
```

```
import java.rmi.*;
import java.rmi.server.*;
public class tempimp extends UnicastRemoteObject implements tempintf
{
    public double ctof(double c) throws RemoteException
    {
        double f=c*180.0/100+32.0;
        return f;
    }
}
```

```
}  
public double ftoc(double f)throws RemoteException  
{  
    double c=(f-32)*100/180.0;  
    return c;  
}  
public tempimp() throws RemoteException  
{};  
}
```

general@general-desktop ~/supriya\$ javac tempimp.java

//Server code

general@general-desktop ~/supriya\$ gedit tempserver.java

```
import java.rmi.*;  
import java.net.*;  
public class tempserver  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            tempimp x=new tempimp();  
            Naming.rebind("tempserver",x);  
        }  
        catch(Exception e)  
        {}  
    }  
}
```

general@general-desktop ~/supriya\$ javac tempserver.java

//Client code

general@general-desktop ~/supriya\$ gedit tempclient.java

```
import java.rmi.*;  
import java.io.*;  
public class tempclient  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            DataInputStream in=new DataInputStream(System.in);  
            tempintf t=(tempintf)Naming.lookup("rmi://localhost/tempserver");  
            System.out.println("Temperature converter");
```

```

        System.out.print("Enter the celsius value");
        float c=Float.parseFloat(in.readLine());
        System.out.println("Celsius to fahrenheit is:"+t.ctof(c));
        System.out.print("Enter the fahrenheit");
        float f=Float.parseFloat(in.readLine());
        System.out.println("fahrenheit to is Celsius: "+t.ftoc(f));
    }
    catch(Exception ioe)
    {}
}
}
}

```

```

general@general-desktop ~/supriya$ javac tempclient.java
Note: tempclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

```

OUTPUT :

```

general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ javac tempclient.java
Note: tempclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
general@general-desktop ~/supriya$ javac tempimp.java
general@general-desktop ~/supriya$ javac tempintf.java
general@general-desktop ~/supriya$ javac tempserver.java
general@general-desktop ~/supriya$ rmic tempimp
general@general-desktop ~/supriya$ rmiregistry &
[1] 4533
general@general-desktop ~/supriya$ java tempserver

```

Open New Terminal

```

general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ java tempclient

```

```

Temperature converter
Enter the celsius value100
Celsius to fahrenheit is:212.0
Enter the fahrenheit212
fahrenheit to is Celsius: 100.0

```

4. Q: Write a java class tax with methods for calculating income tax. Have this class as a servant and create a server program and register in the rmiregistry. Write a client program to invoke these remote methods of the servant and do the calculation. Except inputs interactively.

//Interface code

```

general@general-desktop ~/supriya$ gedit taxintf.java

import java.rmi.*;

```

```
public interface taxintf extends Remote
{
    double tax(double t)throws RemoteException;
}
general@general-desktop ~/supriya$ javac taxintf.java
```

//Implementation code

```
general@general-desktop ~/supriya$ gedit taximp.java
```

```
import java.rmi.*;
import java.rmi.server.*;
public class taximp extends UnicastRemoteObject implements taxintf
{
    public double tax(double t)throws RemoteException
    {
        double tax=0;
        if(t<10000)
            tax=0;
        if((t>=10000) && (t<=25000))
            tax=0.025*t;
        if((t>=25000) && (t<=100000))
            tax=0.05*t;
        if((t>100000) && (t<=1000000))
            tax=0.075*t;
        if(t>1000000)
            tax=0.10*t;
        return tax;
    }
    public taximp() throws RemoteException
    {};
}
```

```
general@general-desktop ~/supriya$ javac taximp.java
```

//Server code

```
general@general-desktop ~/supriya$ gedit taxserver.java
```

```
import java.rmi.*;
import java.net.*;
public class taxserver
{
    public static void main(String args[])
    {
        try
        {
            taximp x=new taximp();
            Naming.rebind("taxserver",x);
        }
    }
}
```

```
    }  
    catch(Exception e)  
    {}  
}  
}
```

general@general-desktop ~/supriya\$ javac taxserver.java

//Client code

general@general-desktop ~/supriya\$ gedit taxclient.java

```
import java.rmi.*;  
import java.io.*;  
public class taxclient  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            DataInputStream in=new DataInputStream(System.in);  
            taxintf t=(taxintf)Naming.lookup("rmi://localhost/taxserver");  
            System.out.print("Enter the salary");  
            double s=Double.valueOf(in.readLine());  
            double it=t.tax(s);  
            if(it==0)  
            {  
                System.out.println("Salary is <10000");  
                System.out.print("No tax detected");  
                System.exit(0);  
            }  
            System.out.println("The tax is :"+it);  
        }  
        catch(Exception ioe)  
        {}  
    }  
}
```

general@general-desktop ~/supriya\$ javac taxclient.java

Note: taxclient.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

OUTPUT :

general@general-desktop ~ \$ cd supriya

general@general-desktop ~/supriya\$ javac taxclient.java

Note: taxclient.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

general@general-desktop ~/supriya\$ javac taximp.java

general@general-desktop ~/supriya\$ javac taxintf.java

```
general@general-desktop ~/supriya$ javac taxserver.java
general@general-desktop ~/supriya$ rmic taximp
general@general-desktop ~/supriya$ rmiregistry &
[1] 3533
general@general-desktop ~/supriya$ java taxserver
```

Open New Terminal

```
general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ java taxclient
Enter the salary15000
The tax is :375.0
```

```
general@general-desktop ~/supriya$ java taxclient
Enter the salary10000
The tax is :250.0
```

```
general@general-desktop ~/supriya$ java taxclient
Enter the salary9000
Salary is <10000
No tax detected
```

5. Q: Write a java class called simple interest with methods for calculating simple interest. Have this class as a servant program and register in the rmi registry. Write a client program to invoke these remote methods of the servant and do the calculation. Except input, @ command prompt.

//Interface code

```
general@general-desktop ~/supriya$ gedit simpintf.java

import java.rmi.*;
public interface simpintf extends Remote
{
    double simpint(double p,double t,double r)throws RemoteException;
}
```

```
general@general-desktop ~/supriya$ javac simpintf.java
```

//Implementation code

```
general@general-desktop ~/supriya$ gedit simpimp.java

import java.rmi.*;
import java.rmi.server.*;
public class simpimp extends UnicastRemoteObject implements simpintf
{
    public double simpint(double p,double t,double r)throws RemoteException
    {
        double f=p*t*r/100;
```



```
        return f;
    }
    public simpimp() throws RemoteException
    {};
}
```

general@general-desktop ~/supriya\$ javac simpimp.java

//Server code

general@general-desktop ~/supriya\$ gedit simpserver.java

```
import java.rmi.*;
import java.net.*;
public class simpserver
{
    public static void main(String args[])
    {
        try
        {
            simpimp x=new simpimp();
            Naming.rebind("simpserver",x);
        }

        catch(Exception e)
        {}
    }
}
```

general@general-desktop ~/amulya\$ javac simpserver.java

//Client code

general@general-desktop ~/amulya\$ gedit simpclient.java

```
import java.rmi.*;
import java.io.*;
public class simpclient
{
    public static void main(String args[])
    {
        try
        {
            simpintf s=(simpintf)Naming.lookup("rmi://localhost/simpserver");
            double p=Double.valueOf(args[0]);
            double t=Double.valueOf(args[1]);
            double r=Double.valueOf(args[2]);
            System.out.println("Simple interest is :"+s.simpint(p,t,r));
        }

        catch(Exception e)
        {}
    }
}
```



```
}
}
```

general@general-desktop ~/supriya\$ javac simpclient.java

Note: simpclient.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

OUTPUT :

general@general-desktop ~ \$ cd supriya

general@general-desktop ~/supriya\$ gedit simpintf.java

general@general-desktop ~/supriya\$ javac simpintf.java

general@general-desktop ~/supriya\$ gedit simpimp.java

general@general-desktop ~/supriya\$ javac simpimp.java

general@general-desktop ~/supriya\$ gedit simpserver.java

general@general-desktop ~/supriya\$ javac simpserver.java

general@general-desktop ~/supriya\$ gedit simpclient.java

general@general-desktop ~/supriya\$ javac simpclient.java

general@general-desktop ~/supriya\$

new terminal

general@general-desktop ~ \$ cd supriya

general@general-desktop ~/supriya\$ javac simpclient.java

general@general-desktop ~/supriya\$ javac simpimp.java

general@general-desktop ~/supriya\$ javac simpintf.java

general@general-desktop ~/supriya\$ javac simpserver.java

general@general-desktop ~/supriya\$ rmic simpimp

general@general-desktop ~/supriya\$ rmiregistry &

[1] 3219

general@general-desktop ~/supriya\$ java simpserver

new terminal

general@general-desktop ~ \$ cd supriya

general@general-desktop ~/supriya\$ java simpclient 2000 2 15

Simple interest is :600.0

6. Q: Write a rmi program to convert dollar to rupees accept input as command prompt.

//Interface code

general@general-desktop ~/supriya\$ gedit moneyintf.java

```
import java.rmi.*;
```

```
public interface moneyintf extends Remote
```

```
{
```

```
    double dtor(double d)throws RemoteException;
```

```
}
```

```
general@general-desktop ~/supriya$ javac moneyintf.java
```

//Implementation code

```
general@general-desktop ~/supriya$ gedit moneyimp.java
```

```
import java.rmi.*;
import java.rmi.server.*;
public class moneyimp extends UnicastRemoteObject implements moneyintf
{
    public double dtor(double d)throws RemoteException
    {
        double r=d*68.60;
        return r;
    }
    public moneyimp() throws RemoteException
    {};
}
```

```
general@general-desktop ~/supriya$ javac moneyimp.java
```

//Server code

```
general@general-desktop ~/supriya$ gedit moneyserver.java
```

```
import java.rmi.*;
import java.net.*;
public class moneyserver
{
    public static void main(String args[])
    {
        try
        {
            moneyimp x=new moneyimp();
            Naming.rebind("moneyserver",x);
        }

        catch(Exception e)
        {}
    }
}
```

```
general@general-desktop ~/supriya$ javac moneyserver.java
```

//Client code

```
general@general-desktop ~/supriya$ gedit moneyclient.java
```

```
import java.rmi.*;
import java.io.*;
public class moneyclient
```

```
{
    public static void main(String args[])
    {
        try
        {
            moneyintf
m=(moneyintf)Naming.lookup("rmi://localhost/moneyserver");
            double d=Double.valueOf(args[0]);
            System.out.println("Dollar to rupees conversion is :"+m.dtor(d));
        }
        catch(Exception e)
        {}
    }
}
```

general@general-desktop ~/supriya\$ javac moneyclient.java

//Compiling the program

```
general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ javac moneyclient.java
general@general-desktop ~/supriya$ javac moneyimp.java
general@general-desktop ~/supriya$ javac moneyintf.java
general@general-desktop ~/supriya$ javac moneyserver.java
general@general-desktop ~/supriya$ rmic moneyimp
general@general-desktop ~/supriya$ rmiregistry &
[1] 3329
general@general-desktop ~/supriya$ java moneyserver
```

//Run the program

```
general@general-desktop ~ $ cd supriya
general@general-desktop ~/supriya$ java moneyclient 68.06
Dollar to rupees conversion is :4668.916
general@general-desktop ~/supriya$ java moneyclient 1
Dollar to rupees conversion is :68.6
```

Z. Q: Write a JDBC program to retrieve the bank balance of the given account number.

```
mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 49
Server version: 5.5.35-1ubuntu1 (Ubuntu)

mysql> create database bankbal;
Query OK, 1 row affected (0.00 sec)

mysql> use bankbal
```

Database changed

```
mysql> create table bank1(acno int(6),name varchar(50),balance bigint);
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_bankbal |
+-----+
| bank1              |
+-----+
1 row in set (0.00 sec)
```

```
mysql> insert into bank1 values(546989,'Anil',10000);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into bank1 values(326723,'Anju',12000);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into bank1 values(327853,'Ram',2000);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into bank1 values(684846,'Roy',25000);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into bank1 values(345278,'Deepak',5000);
Query OK, 1 row affected (0.06 sec)
```

```
mysql> select * from bank1;
```

```
+-----+-----+-----+
| acno   | name   | balance |
+-----+-----+-----+
| 546989 | Anil   | 10000   |
| 326723 | Anju   | 12000   |
| 327853 | Ram    | 2000    |
| 684846 | Roy    | 25000   |
| 345278 | Deepak | 5000    |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> exit;
Bye
```

```
general@general-desktop ~/supriya$ gedit bank.java
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
import java.io.*;
public class bank
{
    static
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
        }

        catch(ClassNotFoundException cnf)
        {
            System.out.println("Driver could not be loaded:"+cnf);
        }
    }
    public static void main(String args[])
    {
        String ConnectionUrl="jdbc:mysql://localhost:3306/bankbal";
        String dbUser="root";
        String dbPwd="password";
        DataInputStream in=new DataInputStream(System.in);
        String ch;
        int flag=0;
        try
        {
            Connection
con=DriverManager.getConnection(ConnectionUrl,dbUser,dbPwd);
            Statement st=con.createStatement();
            System.out.println("Enter the account number :");
            ch=in.readLine();
            ResultSet rec=st.executeQuery("Select * from bank1 where acno
like'" +ch+"%'");
            while(rec.next())
            {
                System.out.print(rec.getString("name"));
                System.out.println("\t"+rec.getString("balance"));
                flag=1;
            }
            if(flag==0)
                System.out.print("No record found");
        }
        catch(SQLException sqle)
        {
            System.out.println("SQLException"+sqle);
        }
        catch(IOException io)
        {
            System.out.println("SQLException"+io);
        }
    }
}
```

OUTPUT :

```
general@general-desktop ~/supriya$ javac bank.java
Note: bank.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
general@general-desktop ~/supriya$ java bank
```

```
Enter the account number :
345278
Deepak      5000
```

```
general@general-desktop ~/supriya$ java bank
```

```
Enter the account number :
679242
No record found
```


SRINIVAS UNIVERSITY**IV SEMISTER BCA DEGREE EXAMINATION MAY-2019****ADVANCED JAVA PROGRAMMING**

Time: 2 Hours

Max.Marks: 50

Instructions: Answer any 10 Questions from PART-A and One Full Questions from Each UNIT in PART-B

1. Which of these is used to perform all input & output operations in Java?
A. streams B. Variables C. classes D. Methods
2. What invokes immediately after the start() method and also any time the applet needs to repaint itself in the browser?
A. Stop() B. Init() C. Paint() D. Destroy()
3. Name the class used to represent a GUI application window, which is optionally resizable and can have a title bar, an icon, and menus.
A. Window B. Panel C. Dialog D. Frame
4. What does AWT stands for?
A. All Window Tools B. All Writing Tools
C. Abstract Window Toolkit D. Abstract Writing Toolkit
5. What are the names of the list layout managers in Java?
A. Flow Layout Manager B. Grid Layout Manager
C. Box Layout Manager D. All of the above
6. Interface ResultSet has a method, getMetaData(), that returns a/an
A. Tuple B. Value C. Object D. Result
7. Which of the following is the tool used to generate the stub and the skeleton?
A. javac B. MI Registry C. rmic D. Java
8. Which of these class is not a member class of java.io package?
A. String B. StringReader C. Writer D. File
9. How many ways can we align the label in a container?
A. 1 B. 2 C. 3 D. 4
10. What is the name of the Swing class that is used for frames?

A. Window

B. Frame

C. JFrame

D. SwingFrame

11. The size of a frame on the screen is measured in:

A. Inches

B. Nits

C. Dots

D. Pixels

12. In RMI, the objects are passed by_____.

A. Value

B. Reference

C. Value and Reference

D. None of the above

PART-B

UNIT-1

UNIT

V

SERVLETS, COOKIES AND SESSIONS

Servlets:

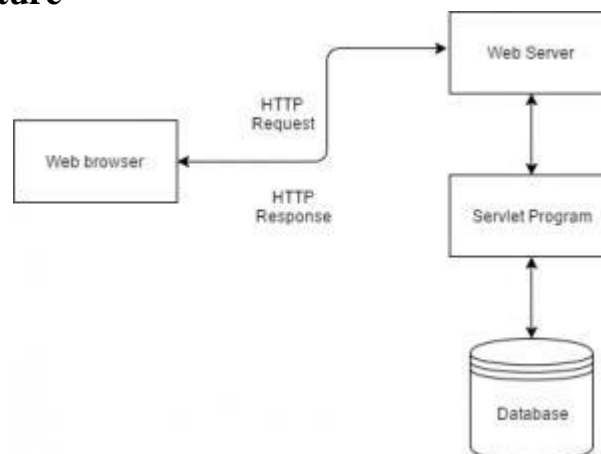
Introduction to Servlets-

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture



Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Advantages of Servlets:

The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area,

lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses Java language.

Web Application:

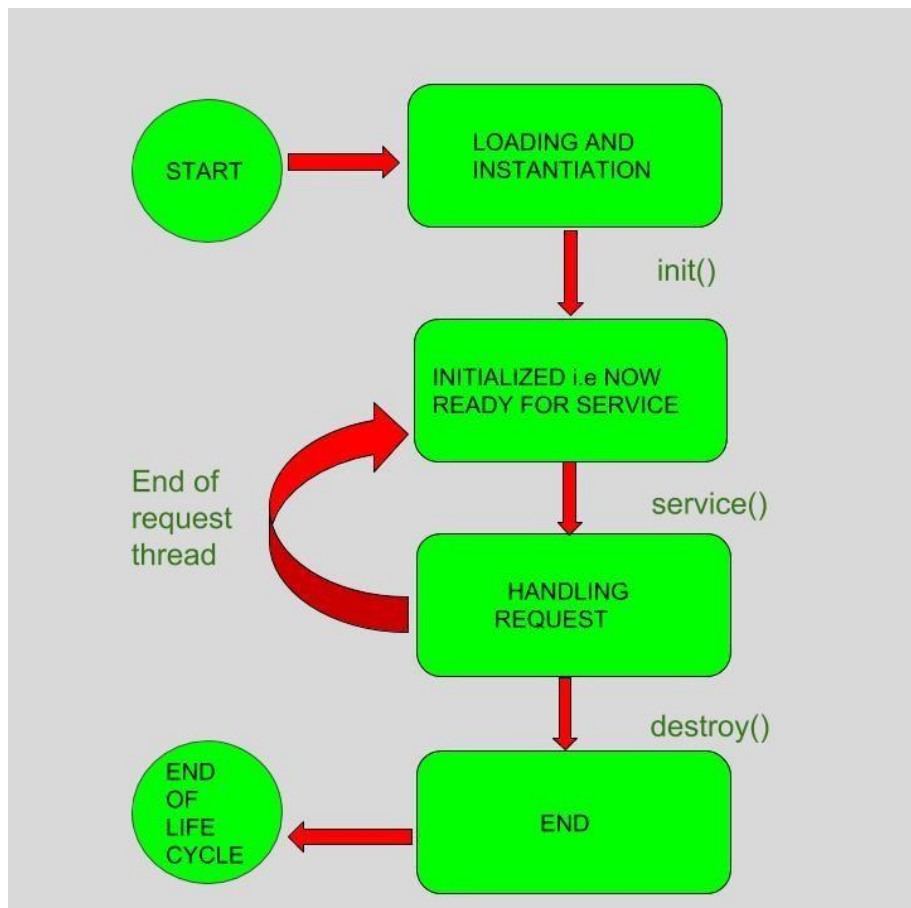
A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

Life Cycle of a Servlet

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it.

Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.



1. **Loading a Servlet:** The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages:

- Initializing the context, on configuring the Servlet with a zero or positive integer value.
- If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

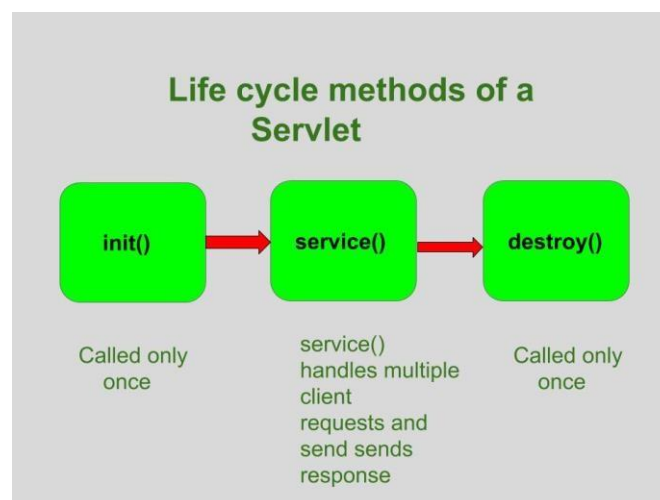
- **Loading:** Loads the Servlet class.
- **Instantiation:** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

2. **Initializing a Servlet:** After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.
The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.
Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.
3. **Handling request:** After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request:
 - a. It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request, then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.
 - b. After creating the request and response objects it invokes the **Servlet.service(ServletRequest, ServletResponse)** method by passing the request and response objects.The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.
4. **Destroying a Servlet:** When a Servlet container decides to destroy the Servlet, it performs the following operations:
 - It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
 - After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

Servlet Life Cycle Methods:

There are three life cycle methods of a Servlet :

- **init()**
- **service()**
- **destroy()**



1. **init() method:** The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

Syntax:

```
//init() method
```

```
public class MyServlet implements Servlet{
    public void init(ServletConfig config) throws ServletException {
        //initialization code
    }
    //rest of code
}
```

2. **service() method:** The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.

- This method uses **ServletRequest** object to collect the data requested by the client.
- This method uses **ServletResponse** object to generate the output content.

Syntax:

```
// service() method
```

```
public class MyServlet implements Servlet{
    public void service(ServletRequest res, ServletResponse res)
        throws ServletException, IOException {
        // request handling code
    }
    // rest of code
}
```

3. **destroy() method:** The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

Syntax:

```
//destroy() method
```

```
public void destroy()
```

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

Using Tomcat for Servlet Development:

Apache Tomcat is a **web container**. It allows the users to run **Servlet and JAVA** Server Pages that are based on the **web-applications**. It can be used as the HTTP server. The performance of the Tomcat server is not as good as the designated web server. It can be used as **separate product** with its own internal Web-server.

The lifecycle of a typical servlet running on Tomcat might look something like this:

1. Tomcat receives a request from a client through one of its connectors.
2. Tomcat maps this request to the appropriate Engine for processing. These Engines are contained within other elements, such as Hosts and Servers, which limit the scope of Tomcat's search for the correct Engine.
3. Once the request has been mapped to the appropriate servlet, Tomcat checks to see if that servlet class has been loaded. If it has not, Tomcat compiles the servlet into Java bytecode, which is executable by the JVM, and creates an instance of the servlet.
4. Tomcat initializes the servlet by calling its `init()` method. The servlet includes code that is able to read Tomcat configuration files and act accordingly, as well as declare any resources it might need, so that Tomcat can create them in an orderly, managed fashion.
5. Once the servlet has been initialized, Tomcat can call the servlet's service method to process the request, which will be returned as a response.
6. During the servlet's lifecycle, Tomcat and the servlet can communicate through the use of listener classes, which monitor the servlet for a variety of state changes. Tomcat can retrieve and store these state changes in a variety of ways, and allow other servlets access to them, allowing state to be maintained and accessed by various components of a given context across the span of a single or multiple user sessions. An example of this functionality in action is an e-commerce application that remembers what the user has added to their cart and is able to pass this data to a checkout process.
7. Tomcat calls the servlet's `destroy` method to smoothly remove the servlet. This action is triggered either by a state change that is being listened for, or by an external command delivered to Tomcat to undeploy the servlet's Context or shut down the server.

The Servlet API

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver. In Java, to create web applications we use Servlets. To create Java Servlets, we need to use Servlet API which contains all the necessary interfaces and classes. Servlet API has 2 packages namely,

- `javax.servlet`
- `javax.servlet.http`

javax.servlet

- This package provides the number of interfaces and classes to support Generic servlet which is protocol independent.
- These interfaces and classes describe and define the contracts between a servlet class and the runtime environment provided by a servlet container.

Classes available in javax.servlet package:

Class Name	Description
<code>GenericServlet</code>	To define a generic and protocol-independent servlet.
<code>ServletContextAttributeEvent</code>	To generate notifications about changes to the attributes of the servlet context of a web application.
<code>ServletContextEvent</code>	To generate notifications about changes to the servlet context of a web application.

ServletInputStream	This class provides an input stream to read binary data from a client request.
--------------------	--

ServletOutputStream	This class provides an output stream for sending binary data to the client.
ServletRequestAttributeEvent	To generate notifications about changes to the attributes of the servlet request in an application.
ServletRequestEvent	To indicate lifecycle events for a ServletRequest.
ServletRequestWrapper	This class provides the implementation of the ServletRequest interface that can be subclassed by developers to adapt the request to a Servlet.
ServletResponseWrapper	This class provides the implementation of the ServletResponse interface that can be subclassed by developers to adapt the response from a Servlet.

Handling HTTP request and Response

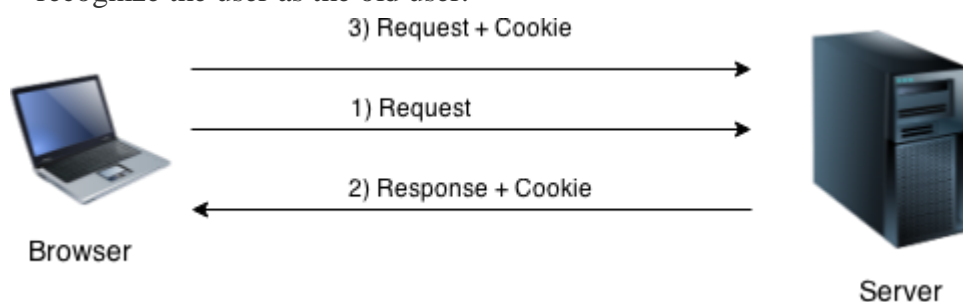
Cookies

Definition: A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Web applications are typically a series of Hypertext Transfer Protocol (HTTP) requests and responses. As HTTP is a stateless protocol, information is not automatically saved between HTTP requests. Web applications use cookies to store state information on the client. Cookies can be used to store information about the user, the user's shopping cart, and so on.

Working of Cookies:

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



Types of Cookies:

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie (Session Cookies)

It is **valid for single session** only. It is removed each time when user closes the browser. Session cookies are stored in memory and are accessible as long as the user is using the web.

application. Session cookies are lost when the user exits the web application. Such cookies are identified by a session ID and are most commonly used to store details of a shopping cart.

Persistent cookie (Permanent Cookies)

It is **valid for multiple session**. It is not removed each time when user closes the browser. It is removed only if user logout or signout. Permanent cookies are used to store long-term information such as user preferences and user identification information. Permanent cookies are stored in persistent storage and are not lost when the user exits the application. Permanent cookies are lost when they expire.

Advantage of Cookies

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

Disadvantage of Cookies

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

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

Sessions:

Definition: The time interval in which two systems(i.e. the client and the server) communicate with each other can be termed as a session. In simpler terms, a session is a state consisting of several requests and response between the client and the server.

It is a known fact that HTTP and Web Servers are both stateless. Hence, the only way to maintain the state of the user is by making use of technologies that implement session tracking. Session tracking in servlets can be implemented by a number of methods. **Sessions are server-side entities that store information (in memory or persisted) that spans multiple requests/responses between the server and the client.** The Servlet HTTP session uses a cookie with the name JSESSIONID and a value that identifies the session.

Types of Sessions:

Session comes in two flavours

1. Stateful session
2. Stateless Session

Stateful session

It creates a channel between client and server, session will be closed if any one end gets disturbed. And they need to re-authenticate again to continue their process.

Example: SSH, FTP, Telnet etc

Stateless session

Once the user is authenticated it maintains a token to do the further process, so there is no need to maintain any channel between client and server, the client needs to re-authenticate if the token is lost or expired.

Example: http protocol, all web based application.

Advantages:

1. It helps maintain user state and data all over the application.
2. It is easy to implement and we can store any kind of object.
3. Stores client data separately.
4. Session is secure and transparent from the user.

Disadvantages:

1. Performance overhead in case of large volumes of data/user, because session data is stored in server memory.
2. Overhead involved in serializing and de-serializing session data

Session Tracking:

In the world of the web, a session is the amount of time in which any two systems interact with each other. Those two systems can have a peer-to-peer or client-server relationship with each other. However, the problem is, in HTTP protocol, the state of the communication is not

maintained, i.e., *HTTP* is a stateless protocol. **Session Tracking in Java** is used to tackle this problem with the help of servlets.

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time. Sessions are shared among the servlets accessed by a client.

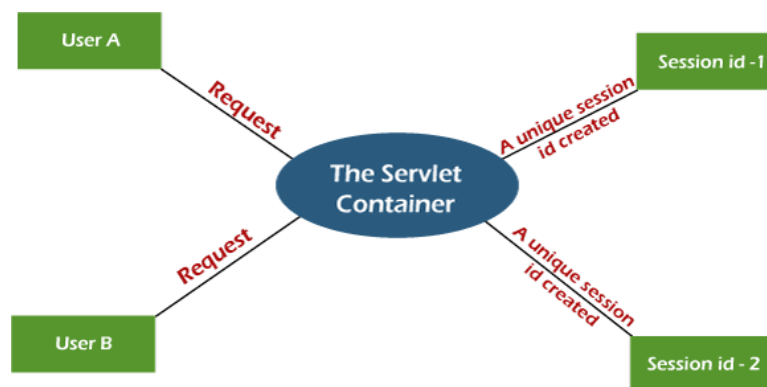
HttpSession Interface

The Java servlets provide the `HttpSession` Interface that gives a way to spot a particular user among multiple page requests or to keep the information about that user. In fact, the Java servlets use the `HttpSession` interface to establish a connection between the *HTTP* server and the *HTTP* client.

The *HttpSession* interface facilitates the servlets to:

- Manipulate and view the information about any session, such as the creation time, the session identifier, and the last accessed time.
- Binding objects to the session, hence; allowing the information about a user to be persistent across the multiple connections.

The following diagram shows the working of the *HttpSession* interface in a session.



User A and User B both are requesting to connect to a server. The servlet container uses the `HttpSession` interface to connect to the server by creating a unique id for each request. The unique id is used to identify a user. The unique id can be stored in a request parameter or in a cookie.

Session Tracking employs Four Different techniques

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. `HttpSession`

1. Cookies

Cookies are little pieces of data delivered by the web server in the response header and kept by the browser. Each web client can be assigned a unique session ID by a web server. Cookies are used to keep the session going. Cookies can be turned off by the client.

2. Hidden Form Field

The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user's view.

3. URL Rewriting

With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.

4. HttpSession

A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.

- 2 a) What are the procedures to create the file.
b) How to build the applet code along with the syntax.

OR

- 3 a) Explain the life cycle of Applet with neat diagram.
b) Explain about Random Access File along with example.

UNIT-2

- 4 a) Explain List control with an example.
b) What is the purpose of layout managers? With an example explain the use of Grid Layout.

OR

- 5 a) Illustrate the use of Label with suitable example.
b) Name the controls required for creating a menu. With an example explain the creation of a menu.

UNIT-3

- 6 a) Explain the use of JTextField and any methods associate with it.
b) Write a short note on containers.

OR

- 7 a) Give one example for swing program
b) Explain the use of JCheckBox and any methods associated with it.

UNIT-4

- 8 a) Explain the architecture of JDBC
b) Explain Java and MYSQL connection using a code example.

OR

- 9 a) Explain the different types of JDBC drivers.
b) Explain about ResultSet Interface with example.

UNIT-5

10. a) Write a short note on stub and skeleton in RMI.
b) Create and run the client application with example.

OR

11. a) Explain architecture of RMI with the help of a diagram.
b) Create and run the interface and implementation code .
