Part II : Web based Application in Java

5.6 Use of JavaScript for Creating Web based Applications in Java ..... 5 - 15

5.7 Introduction to Java Script Frameworks - ReactJS, VueJS, AngularJS ..... 5 - 18

Unit VI

Chapter - 6 Logical and Functional Programming **(6 - 1) to (6 - 24)**

6.1 Understanding Symbol Manipulation ..... 6 - 1

6.2 Basic LISP Functions ..... 6 - 3

6.3 Definitions, Predicates, Conditionals and Scoping ..... 6 - 7

6.4 Recursion and Iteration ..... 6 - 12

6.5 Properties A-List, Array and Access Functions ..... 6 - 13

6.6 Using Lambda Definitions ..... 6 - 13

6.7 Printing, Reading and Atom Manipulation ..... 6 - 14

6.8 An Overview of Prolog ..... 6 - 15

6.9 Syntax and Meaning of Prolog Programs ..... 6 - 17

6.10 Lists, Operators and Arithmetic ..... 6 - 22

6.11 List Structures ..... 6 - 24

Solved SPPU Question Papers **(S - 1) to (S - 5)**

Unit III

3 Java as Object Oriented Programming Language - Overview

3.1 : Fundamentals of JAVA

Q.1 Justify the meaning of each characteristic of Java in the statement "Java is simple, architecture neutral portable, interpreted and robust and secured programming language".

☞ [SPPU : Dec 17, 22, Marks 6]

Ans. : i) Java is simple :

Java is a simple programming language. Even though you have no programming background you can learn this language comfortably. The programmers who worked on C or C++ can learn this language more efficiently because the syntax of Java resembles with C and C++.

ii) Architecture neutral portable : Platform independence means programs in Java can be executed on variety of platforms.

Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

iii) Interpreted : Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted. First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent

machine code. This machine code is then directly executed to obtain the output.

iv) Robust and Secure : Following are reasons that make Java more secured than other programming languages -

1. Java does not support the concept of pointers directly. This makes it impossible to accidentally reference memory that belongs to other programs or the kernel.
2. The output of Java compiler is not executable code but it is a bytecode. The instructions of bytecode is executed by the Java Virtual Machine (JVM). That means JVM converts the bytecode to machine readable form. JVM understand only the bytecode and therefore any infectious code can not be executed by JVM. No virus can infect bytecode. Hence it is difficult to trap the internet and network based applications by the hackers.
3. In programming languages like C or C++ the memory management is done explicitly by the user. That means user allocates or deallocates the memory. Whereas in Java its automatically done using garbage collection. Thus user can not perform the memory management directly.
4. If an applet is executing in some browser then it is not allowed to access the file system of local machine.

Q.2 Write short notes on Java Virtual Machine(JVM) with diagram.

Ans. : ☞ [SPPU : June-22, Marks 6]

- Java Virtual Machine is a set of software and program components. As the name suggests, it is a virtual computer that resides in the real computer. Java can achieve its **platform** independence feature due to Java Virtual Machine.
- The execution of any Java program is carried out as shown in Fig. Q.2.1.

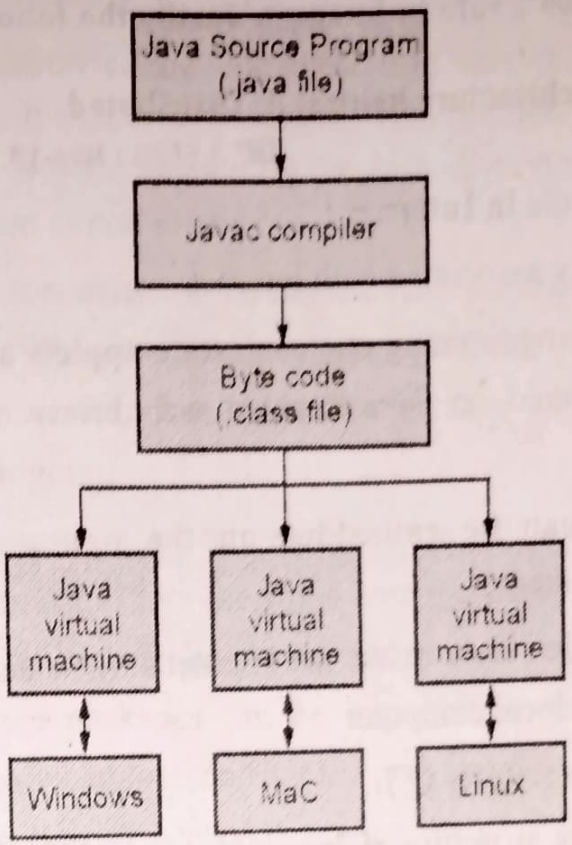


Fig. Q.2.1 Execution of any Java program

- When we want to write any Java program, we write the source code and store it in a file with extension .java.
- This .java file is compiled using **javac** and a class file gets created. This class file is actually a **byte code**. The name **byte code** is given because of the instruction set of Java program.
- The Java Virtual Machine takes **byte code** as an input, reads it, interprets it and then executes it.
- The Java Virtual machine can generate an output corresponding to the underlying platform (operating system).

Q.3 Explain why Java is secured, portable and dynamic ? Which of the concepts in Java ensures these ?

☞ [SPPU : May-18, Marks 6]

Ans. : Refer Q. **.

Q.4 Explain Java's role in Internet. Justify the following features of Java
1. Secure 2. Architecture neutral 3. Distributed

☞ [SPPU : May-19, June-22, Marks 6]

Ans. : Java's role in Internet :

- Java is closely associated with internet.
- Using Java programming one can create **applets** and these applets can be executed on Java enabled web browser(For example - HotJava).
- The applet **can be embedded on the web page** and can be executed on internet.
- The Java applets have made the internet a true extension of storage system of the local computer.

Features of Java : Refer Q.1.

Q.5 Describe the structure of Java program.

Ans. : • The program structure for the Java program is as given in the following figure -

Documentation section
Package statements section
Import statements section
Class definition

Main Method class
{
public static void main(String[] args)
{
//main method definition
}
}

Documentation section : The documentation section provides the information about the source program. This section contains the information which is not compiled by the Java. Everything written in this section is written as comment.

Package section : It consists of the name of the package by using the keyword **package.** When we use the classes from this package in out program then it is necessary to write the package statement in the beginning of the program.

Import statement section : All the required Java API can be imported by the import statement. There are some core packages present in the Java. These packages include the classes and method required for Java programming. These packages can be imported in the program in order to use the classes and methods of the program.

Class definition section : The class definition section contains the definition of the class. This class normally contains the data and the methods manipulating the data.

Main method class : This is called the main method class because it contains the main() function. This class can access the methods defined in other classes.

Q.6 Describe primitive data types. List the primitive data types in Java and their respective storage capacity

☞ **[SPPU : June 22, Marks 6]**

Ans. : Various data types used in Java are byte, short, int, long, char, float, double and boolean.

byte : This is in fact smallest integer type of data type. Its width is of 8 - bits with the range – 128 to 127. The variable can be declared as byte type as

```
byte i,j;
```

short : This data type is also used for defining the signed numerical variables with a width of 16 - bits and having a range from – 32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

int : This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

long : Sometimes when *int* is not sufficient for declaring some data then *long* is used. The range of long is really very long and it is – 9, 223, 372, 036, 854, 775, 808 to 9, 223, 372, 036, 854; 775, 807. The declaration can be

```
long x,y;
```

float : To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is 1.4e – 045 to 3.4e + 038.

double : To represent the real numbers of large range the double data type is used. Its width is 64-bit having the range 4.9e – 324 to 1.8e + 308.

char : This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

Let us see one simple Java program which makes use of various data types.

boolean : Boolean is a simple data type which denotes a value to be either true or false.

Q.7 Write a program in Java using switch-case statement to perform addition, subtraction, multiplication and division of given two numbers and print the result.

☞ [SPPU : Dec.-22, Marks 6]

Ans. :

```
import java.util.Scanner;
class test
{
    public static void main(String[] args)
    {
        char operator;
        Double num1, num2, result;
        Scanner input = new Scanner(System.in);
        System.out.println("Choose an operator: +, -, *, or /");
        operator = input.next().charAt(0);
        System.out.println("Enter first number");
        num1 = input.nextDouble();
        System.out.println("Enter second number");
        num2 = input.nextDouble();
        switch (operator)
        {
          // Addition of two numbers
          case '+':
            result = num1 + num2;
            System.out.println(num1 + " + " + num2 + " = " + result);
          break;
          // subtraction of two numbers
          case '-':
            result = num1 - num2;
            System.out.println(num1 + " - " + num2 + " = " + result);
          break;
          // multiplication of two numbers
          case '*':
            result = num1 * num2;
            System.out.println(num1 + " * " + num2 + " = " + result);
          break;
          // division of two numbers
```

```
        case '/':
          result = num1 / num2;
          System.out.println(num1 + " / " + num2 + " = " + result);
        break;
        default:
          System.out.println("Invalid operator!");
        break;
      }
      input.close();
    }
}
```

Output

```
Choose an operator: +, -, *, or /
+
Enter first number
10
Enter second number
20
10.0 + 20.0 = 30.0
```

Q.8 Write a Java program to display following pattern.

```
        *
      *   *
    *   *   *
  *   *   *   *
*   *   *   *   *
```

Ans. :

```
class TriangleDisplay
{
    public static void main(String[] args)
    {
        int i,j,k;
        for(i=1; i<=5; i++)
        {
          for(j=4; j>=i; j--)
```

```
        {
          System.out.print(" ");
        }
        for(k=1; k<=(2*i-1); k++)
        {
          System.out.print("*");
        }
        System.out.println("");
      }
    }
}
```

Output

```
*
***
*****
*******
*********
```

Q.9 What is the difference between break and continue statement ? Explain it with examples.

Ans. :

Sr. No.	break	continue
1.	The break is used to terminate the execute the loop.	The continue statement is not used to terminate the execution
2.	It breaks iteration.	It skips the iteration.
3.	Break is used in loops and in switch.	Continue is used only in loop.
4.	When this break is executed, the control will come out of the loop.	When continue is executed the control will not come out of the loop, in fact it will jump to next iteration of loop.

5.	**Example :**
	public class breakdemo
	{
	public static void
	main(String[] args)
	{
	for (int i = 1; i <= 20;
	i++)
	{
	if (i % 2 == 0)
	{
	break;
	// skip the even numbers
	}
	System.out.println
	(i + " ");
	//will display only odd
	numbers
	}
	}
	}
	Output
	1
		Example :
		public class continuedemo
		{
		public static void main(String[]
		args)
		{
		for (int i = 1; i <= 20; i++)
		{
		if (i % 2 == 0)
		{
		continue; // skip the
		even numbers
		}
		//will display only odd
		numbers
		System.out.println(i + " ");
		}
		}
		}
		Output
		1
		3
		5
		7
		9
		11
		13
		15
		17

3.2 : Arrays

Q.10 Write a program which receives n integers. Store the integers in an array. Program outputs the number of odd and even numbers present in this array.

☞ [SPPU : Dec. 18, 22, Marks 6]

Ans. :

```
import java.util.*;
public class ArrayDemo
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        int odd_count,even_count;
        odd_count=even_count=0;
        System.out.println("Enter the value of n");
        int n=sc.nextInt();
        int [] A=new int[n];
        System.out.println("Enter the elements");
        for(int i=0;i<n;i++)
          A[i]=sc.nextInt();
        for(int i=0;i<n;i++)
        {
          if(A[i]%2==0)
            even_count++;
          else
            odd_count++;
        }
        System.out.println("The total number of even numbers are:
        "+even_count);
        System.out.println("The total number of odd numbers are:
        "+odd_count);
    }
}
```

Q.11 Write a program in Java to perform the addition of two matrices (multidimensional arrays) and set the diagonal elements of resultant matrix to 0.

☞ [SPPU : Dec.-17,22, Marks 6]

Solution :

```
class Matrix
{
    public static void main(String args[])
    throws java.io.IOException
    {
      int a[][]={
        {10,20,30},
        {40,50,60},
        {70,80,90}
      };
      int b[][]={
        {1,2,3},
        {4,5,6},
        {7,8,9}
      };
      int c[][]=new int [3][3];
      for(int i=0;i<3;i++)
      {
        for(int j=0;j<3;j++)
        {
          c[i][j]=a[i][j]+b[i][j];
        }
      }
      System.out.println("The Addition is...");
      for(int i=0;i<3;i++)
      {
        for(int j=0;j<3;j++)
        {
          System.out.print(" "+c[i][j]);
        }
        System.out.println();
      }
```

```
    for(int i=0;i<3;i++)//setting diagonal elements to 0
    {
      for(int j=0;j<3;j++)
      {
        if(i==j)
        c[i][j]=0;
      }
    }
  }
```

.12 Write a program in Java which reads a matrix of size 3 by 3 nd performs the addition of elements in each row and each olumn. The program prints the each row and column additions.

☞ [SPPU : Nov.-18, Marks 6]

ns. :

```
ublic class Test
{
  public static void main(String[] args)
  {
    int rows, cols, Row_Sum, Col_Sum;

    int a[][] = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
          };

    //Calculates number of rows and columns present in given
      matrix
    rows = a.length;
    cols = a[0].length;

    //Calculates sum of each row of given matrix
    for(int i = 0; i < rows; i++)
    {
      Row_Sum = 0;
      for(int j = 0; j < cols; j++)
```

```
        {
            Row_Sum = Row_Sum + a[i][j];
        }
        System.out.println("Sum of " + (i+1) +" row: " + Row_Sum);
    }
    System.out.println("----------------------------------------------------");
    //Calculates sum of each column of given matrix
    for(int i = 0; i < cols; i++)
    {
        Col_Sum = 0;
        for(int j = 0; j < rows; j++)
        {
            Col_Sum = Col_Sum + a[j][i];
        }
        System.out.println("Sum of " + (i+1) +" column: " +Col_Sum);
    }
  }
}
```

Output

```
Sum of 1 row: 6
Sum of 2 row: 15
Sum of 3 row: 24
----------------------------------------------------
Sum of 1 column: 12
Sum of 2 column: 15
Sum of 3 column: 18
```

3.3 : String Handling

Q.13 What are strings in Java ? Explain following operations of class strings in Java with example –
i) To find length of string ii) To compare two strings iii) Extraction of character from string.

[SPPU : May-19, June 22, Marks 6]

Ans. : Concept string : String is a collection of characters. In Java String defines the object.

i) To find the length : We can find the length of a given string using the method length(). Following program shows the use of **length()** method for strings

```
class Str_lengthDemo
{
public static void main(String[] args)
    {
        char str[]={'P','R','O','G','R','A','M'};
        String S=new String(str);
        System.out.println("The string S is "+S);
        System.out.println("The length of string is "+S.length());
        System.out.print("The string in character array is ");
        for(int i=0;i<S.length();i++)
          System.out.print(str[i]);
    }
}
```

ii) To compare two strings : We use method **equals()** for that purpose. This method is of Boolean type. That is, if two strings are equal then it returns true otherwise it returns false.

```
class StringCompareDemo
{
public static void main(String[] args)
    {
        String str1=new String("INDIA");
        String str2=new String("india");
        if(str1.equals(str2)==true)
          System.out.println("\n The two strings are equal");
        else
          System.out.println("\n The two strings are not equal");
    }
}
```

iii) Extraction of a character from string : String class provides the facility to extract the character from the String object. There is a method called **charAt(index)** with the help of which we can extract the character denoted by some index in the array.

For example :

```
String fruit=new String("mango");
char ch;
ch=fruit.charAt(2);
System.out.println(ch);
```

The output of above code fragment will be n because at the index position 2 of string object fruit the character n is present.

Q.14 Write a program for concatenating two strings.

Ans. :

Java Program [Test.Java]

```
class Test
{
public static void main(String[] args)
{
String fruit=new String("mango");
System.out.println("I like "+fruit +" very much");
}
}
```

Output

```
I like mango very much
```

We can make use of a method **concat()** for concatenating two strings. The above program is slightly modified for the use of concat() -

Java Program

```
class Test
{
public static void main(String[] args)
{
String str=new String("I like ");
String fruit=new String("mango very much");
System.out.println(str.concat(fruit));
}
}
```

Output

I like mango very much

3.4 : Classes and Methods Class Fundamentals

Q.15 How data and methods are organized in an object oriented program ?

Ans. :

Each class is a collection of data and the functions that manipulate the data. The data components of the class are called data fields and the function components of the class are called member functions or methods.

- Thus the state of an object is represented using **data fields** (data members) and behaviour of an object is represented by set of **methods** (member functions). The class template can be represented by following Fig. Q.15.1.

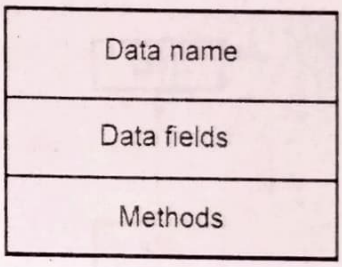


Fig. Q.15.1 Class template

The **Java Class** can written as follows

```
class Customer {

int ID;
int Age;
String Name;

Customer() {
}
Customer(int ID) {
}
```

Data Field

Constructor

```
double withdraw_money() {
...
...
}
```

Method

3.5 : Object

Q.16 What is object ? How to create an object in Java ?

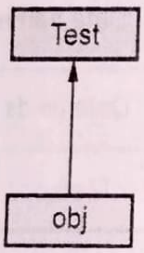
Ans. :

Definition : Objects are nothing but the instances of the class. Hence a block of memory gets allocated for these instance variables.

- For creating objects in Java the operator **new** is used.

```
Test obj ; // Declaration of object obj.
obj = new Test () ; // obj gets instantiated for class Test
```

- We instantiate one object **obj**. It can be represented graphically as -



- Now if we create two objects then it can be

```
Test obj1, obj2;
obj1 = new Test () ;
obj2 = new Test () ;
```

3.6 : Adding Methods to a Class

Q.17 Write a Java program demonstrating the adding a method to a class

Ans. : Following is a simple Java program that illustrates the method inside the class

Java Program

```
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
{
String name;
int roll;
double marks;

public void display(String n,int r,double m)
 {

  System.out.println();
  System.out.println("Name: "+n);
  System.out.println("Roll number: "+r);
  System.out.println("Marks: "+m);

 }
 public static void main(String args[])
 {
  int a,b;
  MyClass obj=new MyClass();
  obj.display("Amar",10,76.65);
 }
}
```

Data members

Members function is defined here

Output

```
Name: Amar
Roll number: 10
Marks: 76.65
```

3.7 : Constructors

Q.18 What is use of constructors ? What are types of constructors in Java ? Give example of each type.

[SPPU : May-18, Marks 7, Dec 22, Nov.-19, Marks 6]

Ans. :

- **Definition** : The constructor is a specialized method for **initializing objects.**
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.

```
Class Test
{
    Test()
    {
      //body of constructor
    }
}
```

Class Name

Constructor

- Whenever an object of its associated class is created, the constructor is invoked automatically.
- The constructor is called constructor because it creates values for data fields of class.

Types of Constructor :

Various types of constructors are -

1. No argument(Default) constructor
2. Parameterized constructor
3. Copy constructor

1. No argument constructor : The default constructor is a constructor for which there are no arguments or parameters passed. For example -

Java Program

```
public class Test
{
    int a;
    Test() //no argument constructor
    {
        a=0;
    }
}
class MainClass {
    public static void main(String args[])
    {
        Test obj=new Test();
    }
}
```

2. Parameterized constructor : The parameterized constructor is a constructor in which the parameters are passed to the constructor function. For example -

Java Program

```
public class Test
{
    int a;
    Test(int x)//parameterized constructor
    {
        a=x;
    }

}
class MainClass {
    public static void main(String args[])
    {
        Test obj=new Test(10);
    }
}
```

3. Copy constructor

Copy constructor is a kind of constructor which can be initialised using the object of the same class. This object is created already. The use of copy constructor is -

1. To initialize one object from another object of same type.
2. The object can be passed as a parameter to the function.
3. The copy of the object can be returned from the function.

Syntax of copy constructor

```
classname (const classname &obj)
{
  // body of constructor
}
```

Java Program

Following example shows how copy constructor can be used to initialize another object of the same class.

```
/*****************************************************************
Program to read complex numbers and to copy it into another using
copy constructor
*****************************************************************/
class Complex
{

  private double real, img;
  public Complex(double r, double i)//parameterized constructor
  {
    this.real = r;
    this.img = i;
  }

  Complex(Complex c)  // copy constructor
  {
    System.out.println("Invoking Copy constructor....");
    real = c.real;
    img = c.img;
```

```
    }
    void display()
    {
     System.out.println(real + " + " + img + "i");
    }
}

public class TestClass
{

    public static void main(String[] args)
    {
      Complex obj1 = new Complex(10,20);
      Complex obj2 = new Complex(obj1);
      Complex obj3 = obj2;
       obj3.display();
    }
}
```

Invoking Copy Constructor

Output

```
Invoking Copy constructor....
10.0 + 20.0i
```

Q.19 What is a constructor ? Show with example the use and overloading of default, parameterized and copy constructor.

☞ [SPPU : Nov.-18, Marks 7]

Ans. : Constructor : Refer Q.18.

Use and overloading of various types of constructor :

```
class Complex
{
    private double real, img;
    public Complex()//default constructor
    {
      real = 0;
      img = 0;
    }
    public Complex(double r, double i)//parameterized constructor
    {
      this.real = r;
```

```
        this.img = i;
    }
    Complex(Complex c) // copy constructor
    {
        System.out.println("Invoking Copy constructor....");
        real = c.real;
        img = c.img;
    }
    void display()
    {
        System.out.println(real + " + " + img + "i");
    }
}
public class TestClass
{
    public static void main(String[] args)
    {
        Complex obj1 = new Complex();
        System.out.println("Initializing real and imaginary values...");
        obj1.display();
        Complex obj2 = new Complex(10,20);
        Complex obj3 = new Complex(obj2);
        Complex obj4 = obj3;
        obj3.display();
    }
}
```

Output

```
Initializing real and imaginary values...
0.0 + 0.0i
Invoking Copy constructor....
10.0 + 20.0i
```

3.8 : this keyword

Q.20 Explain about `this' keyword with an example. ☞ [SPPU : Dec 22, Marks 2]

Ans. :

- The **this** keyword is used by a method to **refer the object which invoked that method.**

- The this keyword is used inside the method definition to refer **the current object**. That means **this** is a reference to the object which is used to invoke it.

Following is a Java program in which the method uses the keyword **this**

Java Program

```
//////////////////////////////////////////////////////////////////////////////
//Program for demonstrating the keyword this
//////////////////////////////////////////////////////////////////////////////
import java.io.*;
import java.lang.*;

public class thisDemo
{
 int a,b;
 public void Sum(int a,int b)
 {
    this.a=a;
    this.b=b;
 }
 public void display()
 {
    int c=a+b;
   System.out.println("a= "+a);
   System.out.println("b= "+b);
   System.out.println("Sum= "+c);
   System.out.println();
 }
 public static void main(String args[])
 {
    thisDemo obj1=new thisDemo();
    thisDemo obj2=new thisDemo();
    obj1.Sum(10,20);
    obj1.display();
    obj2.Sum(40,50);
    obj2.display();
```

```
}
}
```

Output

```
a= 10
b= 20
Sum= 30

a= 40
b= 50
Sum= 90
```

Program explanation

In above program,

1) We have created two objects for the class - **obj1** and **obj2**.
2) The value of a is assigned to be 10 and value of b as 20 for the object **obj1** using the keyword **this**.
3) Similarly the obj2 posses the values of a and b as 40 and 50 respectively. Note that the keyword **this** is useful to refer the current object.

Q.21 Demonstrate through a program in Java how constructors can be used for Initializing the elements of matrix with '0' in the example on addition of 2 matrices.

Ans. : ☞ [SPPU : Dec.-19, Marks 6]

```
import java.io.*;
import java.util.*;

class Matrix
{
  private int A[][] = new int[3][3];
  private int B[][] = new int[3][3];
  private int C[][] = new int[3][3];
  public Matrix() // Constructor that initializes matrix to 0
  {
```

```
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            A[i][j] = 0;
            B[i][j] = 0;
        }
    }
  }
  public void show()
  {
    System.out.println("Resultant Matrix is ...");
    for(int i=0;i<3; i++)
    {
      for(int j=0;j<3;j++)
        System.out.print(C[i][j]+" ");
      System.out.println("");
    }
  }
public void get_matrix()
  {
    System.out.println("Enter matrix A");
    Scanner input = new Scanner(System.in);
    for(int i=0;i<3; i++)
    {
      for(int j=0;j<3;j++)
        A[i][j] = input.nextInt();
    }
    System.out.println("Enter matrix B");
    for(int i=0;i<3; i++)
    {
      for(int j=0;j<3;j++)
        B[i][j] = input.nextInt();
    }
    }
  public void addition()
  {
    for(int i=0;i<3; i++)
```

```
        {
            for(int j=0;j<3;j++)
                C[i][j]=A[i][j]+B[i][j];
        }
    }
}
class Test
{
    public static void main(String args[])
    {
        Matrix M1 = new Matrix();
        M1.get_matrix();
        M1.addition();
        M1.show();
    }
}
```

3.9 : Garbage Collection

Q.22 Explain the garbage collections.

Ans. :

- Garbage collection is a method of automatic memory management.
- It works as follows -

Step 1 : When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called **garbage collector** is called.

Step 2 : This routine then searches the system for the nodes that are no longer accessible from an external pointer. These nodes are then made available for reuse by adding them to available pool. The system can then make use of these free available space for allocating the nodes.

- Garbage collection is usually done in two phases - marking phase and collection phase.

- In **marking phase**, the garbage collector scans the entire system and marks all the nodes that can be accessible using external pointer.
- During **collection phase**, the memory is scanned sequentially and the unmarked nodes are made free.

Marking phase : For marking each node, there is one field called **mark field**. Each node that is accessible using external pointer has the value **TRUE** in marking field. For example -

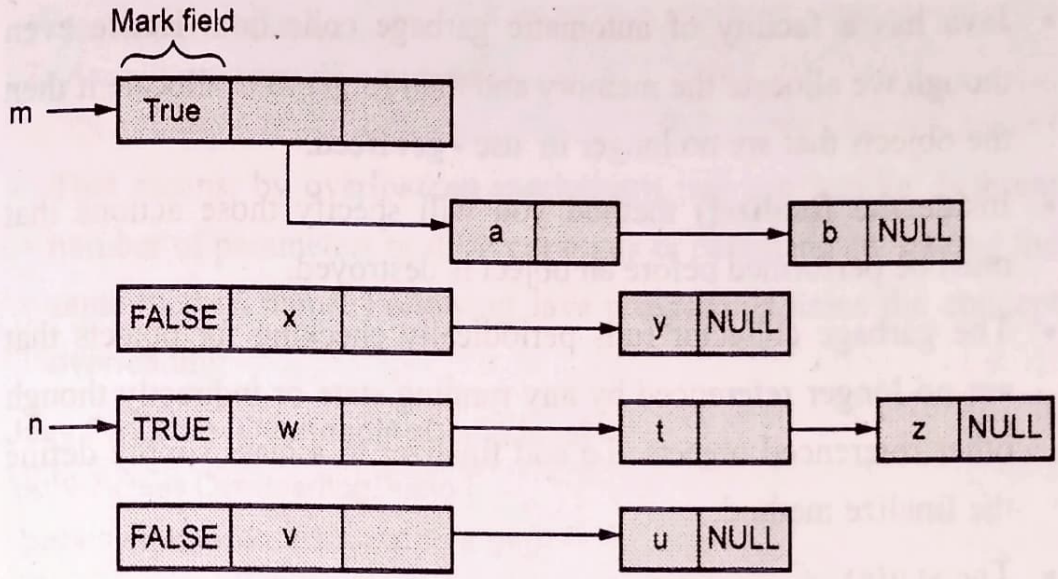


Fig. Q.22.1

Collection phase

- During collection phase, all the nodes that are marked FALSE are collected and made free. This is called **sweeping**. There is another term used in regard to garbage collection called **Thrashing**.
- Consider a scenario that, the garbage collector is called for getting some free space and almost all the nodes are accessible by external pointers. Now garbage collection routine executes and returns a small amount of space. Then again after some time system demands for some free space. Once again garbage collector gets invokes which returns very small amount of free space. This

happens repeatedly and garbage collection routine is executing almost all the time. This process is called **thrashing**. Thrashing must be avoided for better system performance.

3.10 : finalize() method

Q.23 Write short note on - Use of finalize() method.

[SPPU : June-22, Marks 3]

Ans. :

- Java has a facility of automatic garbage collection. Hence even though we allocate the memory and then forget to deallocate it then the objects that are no longer in use - get freed.
- Inside the **finalize()** method you will specify those actions that must be performed before an object is destroyed.
- The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly though other referenced objects. To add finalizer to a class simply define the finalize method.
- The **syntax** is

```
finalize() the code is -
void finalize()
{
finalization code
}
```

- Note that **finalize()** method is called just before the garbage collection. It is not called when an object goes out-of-scope.

3.11 : Overloading Methods

Q.24 What is overloading ? Explain the concept of overloading with suitable Java Program.

Ans. :

Definition : Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

For example :

```
int sum(int a,int b);
double sum(double a,double b);
int sum(int a,int b,int c);
```

- That means, by overloading mechanism, we can handle different number of parameters or different types of parameter by having the same method name. Following Java program explains the concept overloading -

Java Program [OverloadingDemo.java]

```
public class OverloadingDemo {
 public static void main(String args[]) {
 System.out.println("Sum of two integers     ");
 Sum(10,20); <-------------- line A
 System.out.println("Sum of two double numbers     ");
 Sum(10.5,20.4); <-------------- line B
 System.out.println("Sum of three integers    ");
 Sum(10,20,30); <-------------- line C
 }
 public static void Sum(int num1,int num2)
 {
  int ans;
  ans=num1+num2;
  System.out.println(ans);
 }
 public static void Sum(double num1,double num2)
 {
```

```
  double ans;
  ans=num1+num2;
  System.out.println(ans);
 }
 public static void Sum(int num1,int num2,int num3)
 {
  int ans;
  ans=num1+num2+num3;
  System.out.println(ans);
 }
}
```

Output

```
F:\test>javac OverloadingDemo.java
F:\test>java OverloadingDemo
Sum of two integers
30
Sum of two double numbers
30.9
Sum of three integers
60
```

Program explanation

- In above program, we have used three different methods possessing the same name. Note that,

On the line A

- We have invoked a method to which two integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1, int num2)** to fulfil the call. Hence we get the output as 30 which is actually the addition of 10 and 20.

On line B

- We have invoked a method to which two double parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30.9 which is actually the addition of 10.5 and 20.4.

On line C

- We have invoked a method to which the three integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1,int num2,int num3)** to fulfil the call. Hence we get the output as 60 which is actually the addition of 10, 20 and 30.

3.12 : Argument Passing

Q.25 Explain the call by value passing method with suitable example

Ans. : • In the **call by value** method the value of the actual argument is assigned to the formal parameter. If any change is made in the formal parameter in the subroutine definition then that change does not reflect the actual parameters. Following Java program shows the use of parameter passing by value -

Java Program

```
//////////////////////////////////////////////////////////////////////////////////////
//Program implementing the parameter passing by value
//////////////////////////////////////////////////////////////////////////////////////
public class ParameterByVal
{
 void Fun(int a,int b)
 {
  a=a+5;
  b=b+5;
 }
 public static void main(String args[])
  {
   ParameterByVal obj1=new ParameterByVal();
   int a,b;
   a=10;b=20;
   System.out.println("The values of a and b before function call");
```

```
    System.out.println("a= "+a);
    System.out.println("b= "+b);
    obj1.Fun(a,b);
    System.out.println("The values of a and b after function call");
    System.out.println("a= "+a);
    System.out.println("b= "+b);

  }
}
```

Output

```
The values of a and b before function call
a= 10
b= 20
The values of a and b after function call
a= 10
b= 20
```

Program explanation

- The above Java program makes use of the parameter **passing by value.** By this approach of parameter passing we are passing the actual values of variables a and b.
- In the function **Fun** we are changing the values of **a** and **b** by adding 5 to them. But these incremented values will remain in the function definition body only. After returning from this function these values will not get preserved. Hence we get the same values of a and b before and after the function call.

3.13 : Object as Parameter

Q.26 Can we pass object as a parameter in Java ? Justify.

Ans. :

- The object can be passed to a method as an argument.
- Using **dot operator** the object's value can be accessed. Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
    //body of method
}
```

Following is a sample Java program in which the method **area** is defined. The parameter passed to this method is an **object**.

Java Program[ObjDemo.java]

```
public class ObjDemo {
    int height;
    int width;
    ObjDemo(int h,int w)
    {
        height=h;
        width=w;
    }
    void area(ObjDemo o)
    {
        int result=(height+o.height)*(width+o.width);
        System.out.println("The area is "+result);
    }
}
class Demo {
 public static void main(String args[])
{
    ObjDemo obj1=new ObjDemo(2,3);
    ObjDemo obj2=new ObjDemo(10,20);
    obj1.area(obj2);

 }
}
```

height=2 and o.height=10
width=3 and o.width=20
Hence,
result=(2+10)*(3+20)
=12*23
=276

Method with object as parameter

Output

```
F:\test>javac ObjDemo.java
F:\test>java Demo

The area is 276
```

3.14 : Returning Objects

Q.27 Write a Java program that demonstrates returning of object from a method

Ans. : We can return an object from a method. The data type such method is a class type.

Java Program[ObjRetDemo.java]

```
public class ObjRetDemo {
    int a;
    ObjRetDemo(int val)
    {
        a=val;
    }
    ObjRetDemo fun()
    {
        ObjRetDemo temp=new ObjRetDemo(a+5); //created a new
        object
        return temp; //returning the object from this method
    }
}
class ObjRet {
public static void main(String args[])
{
    ObjRetDemo obj2=new ObjRetDemo(20);
    ObjRetDemo obj1;
    obj1=obj2.fun();//obj1 gets the value from object temp
    System.out.println("The returned value is = "+obj1.a);

}
}
```

Data type of this method is name of the class

Output

```
F:\test>javac ObjRetDemo.java
F:\test>java ObjRet
The returned value is = 25
```

3.15 : Access Control

Q.28 Explain the public, private and protected mode in Java with illustrative example

Ans. : • In object oriented programming, a class can protect its member variables or methods from accessing by other entities. That means a class can allow to get the access for particular variable from outside or particular variable can not be accessed at all by the outside entity. This access control mechanism is set by means of **access specifiers.**

- Access specifiers control access to data fields, methods and classes.
- There are three modifiers used in Java -
  - public
  - private
  - default modifier

public allows classes, methods and data fields accessible from any class

private allows classes, methods and data fields accessible only from within the own class.

- If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called package-private or package-access. A package is nothing but the grouping of classes.

For example :

```
package Test;
public class class1
{
    public int a;
    int b;
    private int c;
    public void fun1() {
    }
    void fun2() {
    }
    private void fun3() {
    }
}
public class class2
{
    void My_method() {
    class1 obj=new class1();
    obj.a;//allowed
    obj.b;//allowed
    obj.c;//error:cannot access
    obj.fun1();//allowed
    obj.fun2();//allowed
    obj.fun3();//error:cannot access
    }
}
```

```
package another_Test
public class class3
{
    void My_method() {
    class1 obj=new class1();
    obj.a;//allowed
    obj.b;// error:cannot access
    obj.c;//error:cannot access
    obj.fun1();//allowed
    obj.fun2()//error:cannot access
    obj.fun3()//error:cannot access
    }
}
```

3.16 : Static Members

Q.29 Explain the keyword static with example

☞ [SPPU : Dec.-22, Marks 2]

Ans. :

- The static members can be **static data** member or **static method**.
- The static members are those members which can be accessed **without using object**.

Following program illustrates the use of static members.

Java Program[StaticProg.java]

```
/*****************************************************************
Program to introduce the use of the static method and
static variables
*****************************************************************/
class StaticProg
{
 static int a=10;
 static void fun(int b)
 {
```

```
    System.out.println("b= "+b);
    System.out.println("a= "+a);

  }
}
class AnotherClass
{
  public static void main(String[] args)
  {
    System.out.println("a= "+StaticProg.a);
    StaticProg.fun(20);

  }
}
```

Output

```
a= 10
b= 20
a= 10
```

Q.30 What is the use of static variables and methods in Java ? Give example of static declaration. What are restrictions on methods which are declared static ?

☞ [SPPU : May 17, Marks 7]

Ans. : Static variables and methods : Refer Q.29.

Restriction on methods that are declared static :

- The static methods must can access only static data.
- The static method can call only the static method and can not call a non static method.
- The static method can not refer to this pointer.
- The static method can not refer to super method.

Q.31 Summarize different access controls in Java. Explain the situation if you remove static modifier from the main method.

☞ [SPPU : June-22, Marks 6]

Ans. : Access control : Refer Q.28.

- If remove "static" modifier from the main method then, our program will be compiled without any compilation errors. But at the time of execution, the JVM searches for the main method which is public, static, with a return type and a String array as an argument. If such a method is not found then an **error is generated at the run time**.

3.17 : Nested and Inner Classes

Q.32 Write a Java program that demonstrates the use of nested and inner class

Ans. :

Java Program

```
////////////////////////////////////////////////////////////////////////////////
//Demonstrating the concept of nested and inner class
////////////////////////////////////////////////////////////////////////////////
import java.io.*;
import java.util.*;
class A
{
 int a=10;
 void show()
 {
   B obj_B=new B();
   obj_B.display();
 }
 class B
 {
  void display()
  {
   System.out.println("The value of a is = "+a);
  }
 }
 public static void main(String[] args)
 {
```

```
    A obj_A=new A();
    obj_A.show();
  }
}
```

Output

```
The value of a is = 10
```

Program explanation

In above program, the class B is within the class A. The class A has one function **show()** which invokes the function of class B. The class B defines a function **display()** which invokes the variable **a** of class **A**. That means inner class can access the contents of the outer class.

The access chain will be

Object of class A → Method of class A → Method of class B → Variable of outer class

Fig. Q.32.1 Access chain

3.18 : Command Line Arguments

Q.33 Explain command line argument

Ans. :

- Using command line arguments we can provide the input to the program during the execution.
- For example while executing the program test.java we can provide the command line argument like this -

```
c:\>java test 10 20
```

10 and 20 are the command line arguments

- The command line parameters can be read in an array of strings. For keeping track of total number of command prompt arguments a integer count can be maintained.

Java Program[CommandDemo.java]

```
class CommandDemo
{
 public static void main(String args[])
 {
  int i=0;
  String str;

  System.out.println("\n Following are the strings entered at command
  prompt");
  while(i<args.length)
  {
    str=args[i];
    i=i+1;
    System.out.println(str);
  }
   System.out.println("\n The total number of arguments are..."+i);
 }
}
```

Output

```
D:\>java CommandDemo aaa bbb ccc ddd eee fff

 Following are the strings entered at command prompt
aaa
bbb
ccc
ddd
eee
fff
 The total number of arguments are...6
```

3.19 : Variable Length Arguments

Q.34 Illustrate the use of variable length argument in Java

Ans. : • Java allows the method to use variable length arguments i.e. it allows to create the methods that can accept the arguments of

length zero or multiple. This is called as varargs in Java. The varargs uses three dots after the data type. The syntax is as follows -

Return type name_of_function(data_type ... variable_name)

Following Java program illustrates the use of such variable arguments.

Java Program

```
class Test
{
    public static void main(String[] arg)
    {
        System.out.println("Sum: "+fun(10,20));
        System.out.println("Sum: "+fun(10,20,30));
        System.out.println("Sum: "+fun(10,20,30,40,50));
    }
    public static int fun(int ... val)
    {
        int total=0;
        for(int x:val)
          total+=x;
        return total;
    }
}
```

Output

```
Sum: 30
Sum: 60
Sum: 150
```

END...

Unit IV

4 Inheritance, Packages and Exception Handling using Java

4.1 : Inheritances

Q.1 What is inheritance ? What are advantages of using inheritance ? Show by example the simple inheritance in Java.

☞ [SPPU : May-18, Marks 7, June-22, Marks 6, Dec.-22, Marks 9]

Ans. : Definition of Inheritance : Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.

The inheritance can be achieved by incorporating the definition of one class into another using the keyword **extends**.

Advantages of Inheritance

One of the key benefits of inheritance is to **minimize** the amount of **duplicate code** in an application by sharing common code amongst several subclasses.

1. **Reusability :** The base class code can be used by derived class without any need to rewrite the code.
2. **Extensibility :** The base class logic can be extended in the derived classes.
3. **Data hiding :** Base class can decide to keep some data private so that it cannot be altered by the derived class.
4. **Overriding :** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Implementation of Single Inheritance :

```
class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A  //extending  the base class A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println(" The value of c= "+c);
    }
}
class InheritDemo1
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        obj_B.set_a(10);
        obj_B.set_b(20);
        obj_B.show_a();
```

Note that object of class B is accessing method of class A

```
        obj_B.show_b();
        obj_B.mul();
    }
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 200
```

4.2 : Member Access and Inheritance

Q.2 Explain the protected mode access specifier

Ans. : **Protected** mode is an access specifier which is used in inheritance. The **protected** mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	Class	Subclass	Package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

- For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

4.3 : Super Class References

Q.3 Explain to make super class reference.

Ans. : A subclass can invoke the constructor method defined by the superclass. The syntax for the **super** will be

```
super();
```

or

```
super(parameter_list);
```

To invoke the constructor of superclass, **super()** or **super(parameter_list)** statement should appear in the first line of the subclass constructor.

Java Program

```
class A
{
    int a;
    A(int i)
    {
        this.a=i;
        System.out.println("a is Initialised...");
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A
{
    int b;
    B(int i)
    {
        super(i);
        this.b=i+5;
        System.out.println("b is Initialised...");
    }
    void show_b()
```

It must be on the first line of constructor of the subclass. It calls the constructor of the superclass

```
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}
class SuperConstrDemo
{
    public static void main(String args[])
    {
        B obj_B=new B(20);// Note that the object of subclass is
                                    created only.
        obj_B.show_a();
        obj_B.show_b();
        obj_B.mul();
    }
}
```

Output

```
F:\test>javac SuperConstrDemo.java
F:\test>java SuperConstrDemo
a is Initialised...
b is Initialised...
The value of a= 20
The value of b= 25
The value of c= 500
```

4.4 : Using super

Q.4 State the significance of keyword "Super" in Java. Demonstrate with example for super keyword in Java constructor

☞ [SPPU : Nov.-19, June-22, Marks 6]

Ans. : Super is a keyword used to access the immediate parent class from subclass.

There are three ways by which the keyword **super** is used.

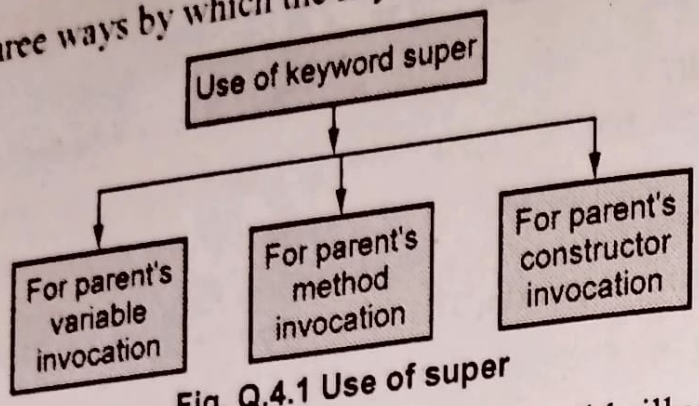


Fig. Q.4.1 Use of super

Let us understand these uses of keyword super with illustrative Java programs.

1. The super() is used to invoke the class method of immediate parent class.

Java Program[B.java]

```
class A
{
    int x=10;
}
class B extends A
{
    int x=20;
    void display()
    {
        System.out.println(super.x);
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
```

Output

```
10
```

Program explanation : In above program,

1) Class A is a immediate parent class of class B. Both the class A and Class B has variables x.

2) In class A, the value of x variable is 10 and in class B the value of variable x is 20.

3) In display function if we would write

```
System.out.println(x);
```

The output will be **20** but if we user **super.x** then the variable **x** of class **A** will be referred. Hence the output is **10**.

2. The super() is used to access the class variable of immediate parent class.

```
class A
{
    void fun()
    {
        System.out.println("Method: Class A");
    }
}
class B extends A
{
    void fun()
    {
        System.out.println("Method: Class B");
    }
    void display()
    {
        super.fun();
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
```

Output

```
Method: Class A
```

Program Explanation : In above program,

1) The derived class can access the immediate parent's class method using **super.fun()**. Hence is the output.

2) You can change **super.fun()** to **fun()**. Then note that in this case, the output will be invocation of subclass method **fun.**

3. The super() is used to invoke the immediate parent class constructor.

```
class A
{
    A()
    {
        System.out.println("Constructor of Class A");
    }
}
class B extends A
{
    B()
    {
        super();
        System.out.println("Constructor of Class B");
    }
    public static void main(String args[])
    {
        B obj=new B();
    }
}
```

Output

```
Constructor of Class A
Constructor of Class B
```

4.5 : Use of Final Keyword

Q.5 Explain the significance of "final" in Java. Demonstrate it with suitable example

Ans. : The final keyword can be applied at three places -

1. Final Variables and Methods

- A variable can be declared as final. If a particular variable is declared as final then it cannot be modified further.
- The final variable is always a constant.

- For example -

```
final int a=10;
```

- The final keyword can also be applied to the method. When final keyword is applied to the method, the method overriding is avoided. That means the methods those are declared with the keyword **final** cannot be overridden.

Consider the following Java program which makes use of the keyword **final** for declaring the method -

```
class Test
{
 final void fun()
 {
  System.out.println("\n Hello, this function declared using final");
}
}
class Test1 extends Test
{
 final void fun()
 {
  System.out.println("\n Hello, this another function");
 }
}
```

Output

```
D:\>javac Test.java
Test.java:10: fun() in Test1 cannot override fun() in Test; overridden
method is final
 final void fun()
        ^
1 error
```

Program Explanation

The above program, on execution shows the error. Because the method **fun** is declared with the keyword **final** and it cannot be overridden in derived class.

2. Final Classes to Stop Inheritance

If we declare particular class as final, no class can be derived from it. Following Java program is an example of final classes.

```
final class Test
{
 void fun()
 {
  System.out.println("\n Hello, this function in base class");
 }
}
class Test1 extends Test
{
 final void fun()
 {
  System.out.println("\n Hello, this another function");
 }
}
```

Output

```
Test.java:8: cannot inherit from final Test
class Test1 extends Test
            ^
1 error
```

4.6 : Multilevel Hierarchy

Q.6 What is inheritance? How multilevel inheritance is achieved in Java? Explain with suitable example.

☞ [SPPU : May-19, Marks 7]

Ans. : Inheritance : Refer Q.1.

Multilevel Inheritance : In the following program, we have created a base class A from which the subclass B is derived. There is a class C which is derived from class B. In the function main we can access any of the field in the class hierarchy by created the object of class C.

Java Program[MultiLvlInherit.java]

```
class A
{
```

```
 int a;
 void set_a(int i)
 {
    a=i;
 }
 void show_a()
 {
    System.out.println("The value of a= "+a);
 }
}
class B extends A
{
 int b;
 void set_b(int i)
 {
    b=i;
 }
 void show_b()
 {
    System.out.println("The value of b= "+b);
 }
}
class C extends B
{
 int c;
 void set_c(int i)
 {
    c=i;
 }
 void show_c()
 {
    System.out.println("The value of c= "+c);
 }
 void mul()
 {
  int ans;
  ans=a*b*c;
  System.out.println(" The value of ans= "+ans);
```

```
 }

}

class MultiLvlInherit
{
 public static void main(String args[])
 {
    A obj_A=new A();
    B obj_B=new B();
    C obj_C=new C();
    obj_C.set_a(10);
    obj_C.set_b(20);
    obj_C.set_c(30);
    obj_C.show_a();
    obj_C.show_b();
    obj_C.show_c();
    obj_C.mul();
 }
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 30
The value of ans= 6000
```

4.7 : Constructor Call Sequence

Q.7 What is constructor chaining? Explain with suitable example

Ans. :

- A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using **super()** as a **first** statement in constructor.
- Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining**.

- Thus the calling of constructor occurs from **top to down**.
- In the following Java program, there are three classes namely: A, B and C.

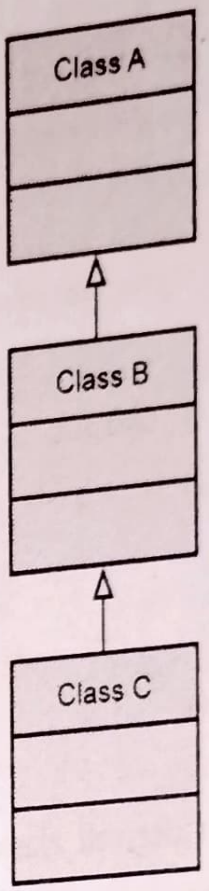


Fig. Q.7.1 Constructor chaining

The constructor chaining can be illustrated by following Java program.

Java Program[C.java]

```
class A
{
 public A()
   {
      System.out.println("1. Statement in class A");
   }
   public B(int i)
   {
   System.out.println(i+". Statement in class B");
   }
```

```
}
public class C extends B
{
    public static void main(String arg[])
    {
     new C();
    }
    public C()
    {
        System.out.println("4. Statement in class C");
    }
}
```

Output

```
F:\test>javac C.java
F:\test>java C
1. Statement in class A
2. Statement in class B
3. Statement in class B
4. Statement in class C
```

Program Explanation

1) When the constructor for derived class **C** is invoked by **new C()** statement then the constructor for its superclass **B** is invoked.
2) In turn, class **B** invokes the constructor for class **A**. This is called constructor chaining.

4.8 : Method Overriding

Q.8 What is method overriding in Java ? What is advantage of using overriding ? Demonstrate method overriding with example

☞ [SPPU : May-17, Marks 6]

Ans. :

Definition : Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

- The method of superclass which gets modified in subclass has the **same name** and **type signature**.
- The overridden method must be called from the subclass.
- Consider following Java Program, in which the method(named as **fun**) in which **a** is assigned with some value is modified in the derived class. When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass. The version of the method defined by the superclass will be hidden.

Example :

```
class A
{
    int a=0;
    void fun(int i)
    {
        this.a=i;
    }
}
class B extends A
{
    int b;
    void fun(int i)
    {

        int c;
        b=20;
        super.fun(i+5);
        System.out.println("value of a:"+a);
        System.out.println("value of b:"+b);
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}
class OverrideDemo
{
```

```
    public static void main(String args[])
    {
        B obj_B=new B();
        obj_B.fun(10);//function re-defined in derived class
    }
}
```

Output

```
F:\test>javac OverrideDemo.java
F:\test>java OverrideDemo
value of a:15
value of b:20
The value of c= 300
```

Advantage of Overriding : The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class code.

Q.9 State the use of the following constructs in Java with example : Method overriding.

☞ [SPPU : Dec.-17, May-19, Marks 2]

Ans : Refer Q.8.

Q.10 What is difference between method overloading and method overrriding?

Ans. :

Method Overloading	Method Overriding
The method overloading occurs at **compile time**.	The method overriding occurs at the run time or **execution time**.
In case of method overloading **different number of parameters** can be passed to the function.	In function overriding the **number of parameters** that are passed to the function are the **same**.
The overloaded functions may have **different return types.**	In method overriding all the methods will have the **same return type**.

Method overloading is performed **within a** class.	Method overriding is normally performed between two classes that have **inheritance relationship.**

4.9 : Dynamic Method Dispatch

Q.11 What is Polymorphism? What are the types of polymorphism?

Ans. :

Definition : Polymorphism is a mechanism which allows to have many forms of the method having the same name.

- That means, a method **A()** may take an instance of an object of one class and may execute its method or the same method **A()** may take another instance of an object and may execute the method belonging to another class.

Example of Polymorphism

Java Program[PolymorphismDemo.java]

```
class A extends Object
{
    public String toString()
    {
        return "A";
    }
}
class B extends A
{
    public String toString()
    {
        return "B";
    }
}
class C extends B
{
    public String toString()
```

```
    {
        return "C";
    }

}
public class PolymorphismDemo
{
    public static void main(String[] args) {
    fun(new C());//invokes the method toString() of class C
    fun(new B());//invokes the method toString() of class B

    fun(new A());//invokes the method toString() of class A

    }
    public static void fun(Object x) {
    System.out.println(x.toString());
    }
}
```

Output

```
F:\test>javac PolymorphismDemo.java

F:\test>java PolymorphismDemo
C
B
A
```

Program Explanation

1) The same function **fun** is used to execute the methods of various class.
2) The selection of the method of the class (i.e. toString method) depends upon the instance of the object which is passed to the function **fun**.

Types of Polymorphism : There are two types of polymorphism –

1. Compile time polymorphism and
2. Runtime polymorphism

- The compile time polymorphism can be achieved by function overloading and runtime polymorphism can be achieved by method overriding.

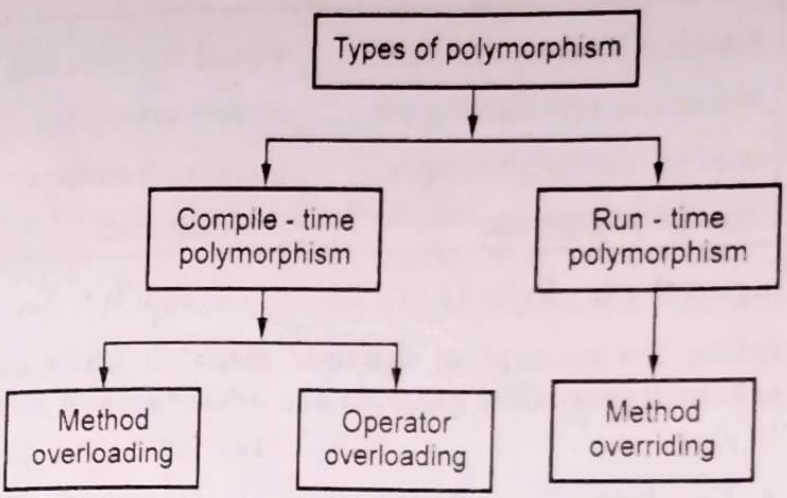


Fig. Q.11.1 Types of polymorphism

Q.12 State the difference in compile time and runtime polymorphism. Show how this is implemented in Java for overriding of methods. ☞[SPPU : Nov.-18, Marks 6]

Ans. : Difference between compile time and runtime polymorphism :

Sr.No.	Compile time polymorphism	Runtime polymorphism
1.	In this type of polymorphism, the call is resolved at compile time.	In this type of polymorphism, the call is resolved at runtime.
2.	This polymorphism is also known as early binding, static binding or overloading	This polymorphism is also known as late binding, dynamic binding or overriding.
3.	Compile-time polymorphism allows us to have more than one method share the same name with different signatures and different return types.	Run-time polymorphism is associated in different classes but allows us to have the same method with the same signature name.

4.	The method is executed at compile time and hence compile time polymorphism provides fast execution.	The method is executed at run time and hence runtime polymorphism provides slow execution.
5.	Function(Method) overloading and operator overloading are used to accomplish compile time polymorphism.	Virtual functions and pointers are used to accomplish runtime polymorphism.

Overriding methods : Refer Q.11.

Q.13 Explain the concept of dynamic dispatch while overriding method in inheritance. Give example and advantages of doing so.

☞ [SPPU : Dec.-17, Marks 5]

Ans. : • The dynamic method dispatch is also called as **runtime polymorphism**.

- During the run time polymorphism, a call to overridden method is resolved at run time.
- The overridden method is called using the reference variable of a super class(or base class). The determination of which method to invoke is done using the object being referred by the reference variable.

Following is a simple Java program that illustrates the Run time polymorphism.

```
class Base
{
 void display()
 {
    System.out.println("\n Base Method Called");
 }
}
class Derived extends Base
{
 void display() //overridden method
```

```
 {
    System.out.println("\n Derived Method Called");
 }
}
public class RunPolyDemo
{
    public static void main(String args[])
 {
  Base obj=new Derived();//obj is reference to base class
                         // which is referred by the derived class
  obj.display(); //method call which is determined at run time
 }
}
```

Output

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

D:\test>java RunPolyDemo

Derived Method Called
```

Q.14 Difference static and dynamic dispatch

☞ [SPPU : May-18, Marks 3]

Ans. :

Sr.No.	Static Dispatch	Dynamic Dispatch
1.	The function call resolution occurs at compile time.	The function call resolution occurs at run time.
2.	It uses type information using which the function call resolution takes place.	It uses actual object of a class using which the function call resolution takes place.
3.	The private, static and final methods are resolved during static dispatch.	Only virtual methods are resolved during dynamic dispatch.
4.	It is not flexible.	It is flexible.
5.	Execution is slower	Faster execution.

Q.15 What is Polymorphism ? Which type of polymorphism is method overriding ? Demonstrate method overriding in Java.

[SPPU : Nov.-19, Marks 7]

Ans. : Refer Q.11.

4.10 : Abstract Classes

Q.16 State the use of the following constructs in Java with example : Abstract class declaration

[SPPU : Dec.-17, May-19, Marks 2]

Ans. :

- In the inheritance hierarchy, the classes become more specific as we move down towards the subclasses and if we move up in this hierarchy classes become more general.
- The superclass is supposed to be the most general or less specific. Sometimes superclass is so general and less specific that it does nothing but lists out only common features of other classes. Then such a superclass is refereed as **abstract class**.

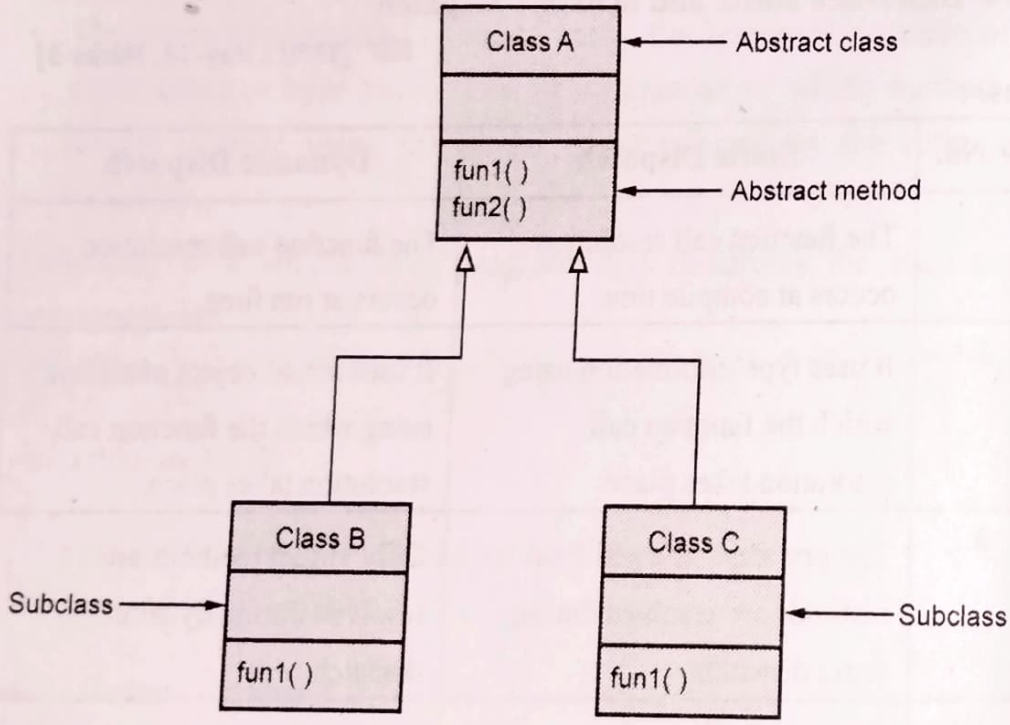


Fig. Q.16.1 Abstract class

- **For example** - In the following Java program we have created three classes - class A is a superclass containing two methods, the class B and class C are inherited from class A. The class A is an abstract class because it contains one abstract method **fun1()**. We have defined this method as abstract because, its definition is overridden in the subclasses **B** and **C**, another function of class **A** that is **fun2()** is a normal function. Refer Fig. Q.16.1.

Java Program[AbstractClsDemo.java]

```
abstract class A
{
 abstract void fun1();
 void fun2()
 {
    System.out.println("A:In fun2");
 }
}
Class B extends A
{
 void fun1()
 {
    System.out.println("B:In fun1");
 }
}
class C extends A
{
 void fun1()
 {
    System.out.println("C:In fun1");
 }

}
public class AbstractClsDemo
{
    public static void main(String[] args)
    {
        B b=new B();
```

This method is so abstract that it has no definition body.
This is an abstract method.

```
        C c=new C();
        b.fun1(); //invoking the overridden method of class B
        b.fun2();
        c.fun1();//invoking the overridden method of class C
        c.fun2();
    }
}
```

Output

```
F:\test>javac AbstractClsDemo.java
F:\test>java AbstractClsDemo
B:In fun1
A:In fun2
C:In fun1
A:In fun2
```

Q.17 Explain abstract classes and polymorphism in Java with appropriate java codes. ☞[SPPU : Dec.-22, Marks 8]

Ans. : Abstract classes : Refer Q.16.

Polymorphism : Refer Q.11.

Q.18 What is the difference between abstract class and concrete class?

Ans. :

Abstract class	Concrete class
The **abstract** keyword is used to define the abstract class.	The keyword **class** is used to define the concrete class.
The abstract class have **partial or no implementation** at all.	The concrete class contains the data members and member functions defined within it.
Abstract class **can not be instantiated.**	Concrete class are usually instantiated in order to access the belonging data members and member functions.

Abstract classes may contain **abstract methods.**	Concrete classes contain concrete method i.e. with code and functionality. Concrete class can not contain abstract method.
Abstract classes **need to be extended** in order to make them useful.	Concrete classes may or may not be extended.
Abstract classes always act as a **parent class.**	Concrete class can be parent or child or neither of the two.

4.11 : Object Class

Q.19 Explain the term - Object class

Ans. :

- In Java there is a special class named **Object**. If no inheritance is specified for the classes then all those classes are subclass of the **Object** class.
- In other words, **Object** is a superclass of all other classes by default. Hence it is specified as -

```
public class A { ...}  is equal to public class A extends Object {...}
```

- The most commonly method of object class being used is **toString()** method.

4.12 : Defining a Package

Q.20 What is the use of package in Java ?

Ans. :

- Package is defined using a keyword **package.**
- The syntax for declaring the package is

```
package name_of_package
```

- This package is defined at the beginning of the program.

- The package statement defines the name space in which the classes are stored. If we omit the package then the default classes are put in the package that has no name.
- Basically Java creates a directory and the name of this directory becomes the package name.
- For example - In your program, if you declare the package as -

```
package My_Package;
```

here we must create the directory name **My_Package** in the current working directory and the required classes must be stored in that directory. Note that the name of the package and the name of the directory must be the same. This name is **case sensitive**.

4.13 : Finding Packages and CLASSPATH

Q.21 How to create and import a package in Java?

Ans. :

Step 1 : Create a folder named **My_Package**.

Step 2 : Create one class which contains two methods. We will store this class in a file named **A.java.** This file will be stored in a folder **My_Package**. The code for this class will be as follows-

Java Program[A.java]

```
package My_Package; //include this package at the beginning
public class A
{
int a;
public void set_val(int n)
{
    a=n;
}
public void display()
{
```

Note that the class and the methods defined must be public.

```
    System.out.println("The value of a is: "+a);
 }
}
```

Note that this class contains two methods namely- **set_val** and **display**. By the set_val method we can assign some value to a variable. The **display** function displays this stored value.

Step 3 : Now we will write another java program named **PackageDemo.java** .This program will use the methods defined in **class** A. This source file is also stored in the subdirectory **My_Package**. The java code for this file is-

Java Program[PackageDemo.java]

```
import My_Package.A;  //The java class A is referenced here by import
statement
class PackageDemo
{
 public static void main(String args[]) throws NoClassDefFoundError
 {
  A obj=new A(); //creating an object of class A
  obj.set_val(10); //Using the object of class A, the methods present
  obj.display();  //in class A are accessed
 }
}
```

Step 4 : Now, open the command prompt and issue the following commands in order to run the package programs

```
D:\>set CLASSPATH=.;D:\;

D:\>cd My_Package

D:\My_Package>javac A.java

D:\My_Package>javac PackageDemo.java
```

Setting the class path

Creating the classes A.class anf package Demo.Class files

```
D:\My_Package>java PackageDemo
The value of a is: 10

D:\My_Package>
```

Running the test program which uses the class **A.Class** stored in another file

Q.22 How CLASSPATH helps to store and retrieve the classes ?

Ans. :

- The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. Thus if the required packages is in the current working directory then it will found. Otherwise you can specify the directory path setting the **CLASSPATH** statement.
- For instance- if the package name **My_Package** is present at prompt D:\> then we can specify

```
set CLASSPATH=.;D:\;
D:\>cd My_Package
D:\My_Package\> Now you can execute the required class files from this location
```

4.14 : Access Protection

Q.23 Explain access protection for class, subclass and package

Ans. : The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	Class	Subclass	Package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

Q.24 What is use of a package in Java ? How CLASSPATH helps to store and retrieve the classes ?
How is access protection provided for packages ?

☞ [SPPU : Nov.-18, Marks 6]

Ans.: Use of package : Refer Q.20.

CLASSPATH : Refer Q.22.

Access Protection : Refer Q.23.

Q.25 Demonstrate the methods of creating packages and importing them in other java programs with example. Write about the access protection in packages.

☞ [SPPU : Dec.-22, Marks 9]

Ans. : Refer Q.21 and Q.23.

4.15 : Importing Packages

Q.26 Write a Java program to implement nested packages.

Ans. : Definition : Nested Package is one package inside another package.

Creating Nested Package

Step 1 : Create a folder named **My_Pkg1**

Step 2 : Create one class which contains two methods. We will store this class in a file named A.java. This file will be stored in a folder Pkg1. The code for this class will be as follows -

Java Program[A.java]

```
package Pkg1; //include this package at the beginning
public class A
{
int a;
```

Note that the class, and the methods defined must be **public**

```
public void set_val(int n)
{
    a=n;
}
public void display()
{
    System.out.println("The value of a is: "+a);
}
}
```

Step 3 : Write a class (in a separate java file) which is to be added in the nested package. Create another directory named **Pkg2** inside the directory **Pkg1**. Then add the class B in the existing package **Pkg2**. I will create a **B.java** file and store the code in it as follows -

Java Program[B.java]

```
package Pkg1.Pkg2; //include nested package at the beginning
public class B
{
 public void show()
 {
    System.out.println("class B is now added in the package");
 }
}
```

Step 3 : Now we will write another java program named **PackageDemo.java**. This program will use the methods defined in class **A**. This source file is also stored in the subdirectory **Pkg1**. The java code for this file is -

Java Program[PackageDemo.java]

```
import Pkg1.A;
import Pkg1.Pkg2.B;  //reference to a B class by import statement
class PackageDemo
{
public static void main(String args[]) throws NoClassDefFoundError
{
```

```
    A Aobj=new A();
    B Bobj=new B(); //creating an object for class B
    Aobj.set_val(10);
    Aobj.display();
    Bobj.show(); //using the object of B class the corresponding
                 function can be accessed.
  }
}
```

Step 4 : Issue the following commands to execute the package with newly added class.

```
C:\Users\admin>d:
D:\>set CLASSPATH=.;D:\;
D:\>cd Pkg1
D:\Pkg1>javac A.java
D:\Pkg1>javac PackageDemo.java
D:\Pkg1>cd Pkg2
D:\Pkg1\Pkg2>javac B.java
D:\Pkg1\Pkg2>cd..
D:\Pkg1>java PackageDemo
The value of a is: 10
class B is now added in the package
```

4.16 : Interfaces (Defining, Implementation, Nesting, Applying)

Q.27 What is an interface in Java ? How is this different than a class ? Give example of interface.

☞ [SPPU : May-17, Marks 6]

Ans. : Interface :

- The interface can be defined using following syntax

```
access_modifier interface name_of_interface
{
   return_type method_name1(parameter1,parameter2,...parametern);
   ...
   return_type method_name1(parameter1,parameter2,...parametern);
```

```
    type static final variable_name=value;
    ...
}
```

- The access_modifier specifies the whether the interface is public or not. If the access specifier is not specified for an interface then that interface will be accessible to all the classes present in that package only. But if the interface is declared as public then it will be accessible to any of the class.
- The methods declared within the interface have no body. It is expected that these methods must be defined within the class definition.

Difference between class and interface :

Sr.No.	Class	Interface
1.	The class is denoted by a keyword class	The interface is denoted by a keyword **interface**
2.	The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.
3.	By creating an instance of a class the class members can be accessed.	You cannot create an instance of an interface.
4.	The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.

5.	The members of a class can be constant or final.	The members of interfaces are always declared as constant(i.e. final).

Example of Interface :

```
interface my_interface //interface defined
{
    void my_method(int val);
}
class A implements my_interface
{
public void my_method(int i)  //defining method declared in interface
{
    System.out.println("\n The value in the class A: "+i);
}
public void another_method() //Defining another method not declared
in interface
{
    System.out.println("\nThis is another method in class A");
}
}
public class InterfaceDemo
{
public static void main(String args[])
{
    my_interface obj=new A(); // object of interface
    //or A obj=new A() is also allowed
        A obj1=new A();
    obj.my_method(100);
    obj1.another_method();
}
}
```

Q.28 State two major differences in class and an interface

☞ [SPPU : Dec.-17, May-18, Marks 4]

Ans. : Refer Q.27.

Q.29 What is meant by Packages and interfaces in Java ? Explain it with suitable example

☞ [SPPU : May-19, Marks 7, June-22, Marks 5]

Ans. : Packages : Refer Q.20 and 21.

Interfaces : Refer Q.27.

4.17 : Variables in Interfaces

Q.30 What is an interface ? What is the difference in class and an Interface ? What is use of declaring variables in an interface in Java ?

☞ [SPPU : Nov.-18, Marks 7]

Ans. : Refer Q.27.

- **Use of Declaring Variables in Interface :** The variables can be assigned with some values within the interface. They are implicitly final and static. Even if you do not specify the variables as final and static they are assumed to be **final** and **static.**

Q.31 Why the members of interface are static and final?

Ans. :

- The members of interface are static and final because –

1) **The reason for being static** - The members of interface belong to interface only and not object.

2) **The reason for being final** - Any implementation can change value of fields if they are not defined as final. Then these members would become part of the implementation. An interface is pure specification without any implementation.

4.18 : Extending Interfaces

Q.32 Describe inheritance. List and explain the different types of Inheritance. Demonstrate how Java supports multiple inheritance.

☞ [SPPU : Nov.-19, Marks 7]

Ans. : Inheritance : Refer Q.1.

Types of inheritance : Refer Q.6.

Multiple inheritance : Multiple inheritance is a mechanism in which the child class inherits the properties from more than one parent classes.

- For example

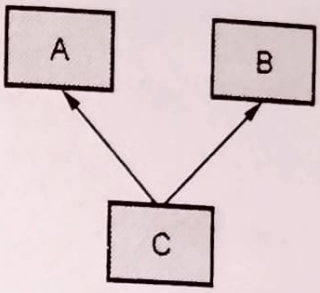


Fig. Q.32.1 Multiple Inheritance

- **Java does not support multiple inheritance** because it creates ambiguity when the properties from both the parent classes are inherited in child class or derived class. But it is supported in case of **interface** because there is no ambiguity as implementation is provided by the implementation class.

Implementation :

```
interface A
{
    void display1();
}
interface B
{
    void display2();
}
class C implements A,B
{
    public void display1()
    {
        System.out.println("From Interface A");
    }
```

```
    public void display2()
    {
        System.out.println("From Interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.display1();
        obj.display2();
    }
}
```

Output

```
From Interface A
From Interface B
```

4.19 : Instanceof Operator

Q.33 Explain the use of instanceof operator in Java with example

Ans. :

- While type casting the objects of the class, the object to be type casted must be an instance of the subclass.
- If some superclass is not an instance of subclass then the run-time exception **ClassCastException** occurs. Hence before doing type casting operation, we must know whether particular object is an instance of some class. This can be done using **instanceof** operator.
- The "Instanceof" is an operator used to test if an object is instance of some class.
- The syntax is

```
name_of_object instanceof class_name
```

Consider following example in which, instanceof operator is used before performing the downcasting.

Java Program[instanceof.java]

```
class LivingBodies extends Object
{
```

```
}
class Human extends LivingBodies
{
}
class Boy extends Human
{

}
public class instanceofDemo
{
    public static void main(String[] args)
    {
     Boy B1=new Boy();
     System.out.println("The value of object B1: "+B1);
     Human H=B1;
     if(H instanceof Boy)
     {
     System.out.println("It is now safe to downcast the object");
     Boy B2=(Boy)H;
     System.out.println("The value of object B2: "+B2);
     }
    }
}
```

Output

```
F:\test>javac instanceofDemo.java

F:\test>java instanceofDemo
The value of object B1: Boy@42e816
It is now safe to downcast the object
The value of object B2: Boy@42e816
```

- Note that **instanceof** is a keyword in which every letter is in lowercase.

4.20 : Fundamental of Exception

Q.34 What is exception handling. Give the benefits of it.

Ans. : Exception handling :

- Exception in Java is an indication of some **unusual event**. Usually it indicates the error.
- In Java, exception is handled using five keywords **try, catch, throw, throws** and **finally.**
- The Java code that you may think may produce exception is placed within the **try block**.

Java Program [ExceptionDemo.Java]

```
class ExceptionDemo
{
public static void main(String args[])
{
 try
 {
  int a,b;
  a=5;
  b=a/0;
 }
catch(ArithmeticException e)
{
System.out.println("Divide by Zero\n");
}
System.out.println("...Executed catch statement");
}
}
```

A try block which includes the lines for which exception must be raised

Exception is caught by catch block. A try-catch pair must exist

Output

```
Divide by Zero

...Executed catch statement
```

Inside a **try** block as soon as the statement: b=a/0 gets executed then an arithmetic exception must be raised, this exception is caught by a **catch** block. Thus there must be a **try-catch pair** and catch block should be immediate follower of try statement. After execution of

catch block the control must come on the next line. These are basically the exceptions thrown by java **runtime systems**.

Benefits : Following are the benefits of exception handling -

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.
4. Due to exception handling mechanism, the errors can be propagated up the method call stack i.e. problems occurring at the lower level can be handled by the higher up methods.

4.21 : Exception Types

Q.35 Define the term exception. State the advantage of exception handling. What are types of exceptions?

☞ [SPPU : Dec.-17, Marks 6]

Ans. : Exception and its advantages : Refer Q.34.

Types of Exceptions :

- There are two type of exceptions in Java

1) **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.

For example : IOException which should be handled using the try-catch block.

2) UnChecked Exception : These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.

For example : ArrayIndexOutOfBounds, NullPointerException, Run Time Exception.

The hierarchy of built in exceptions is as shown below –

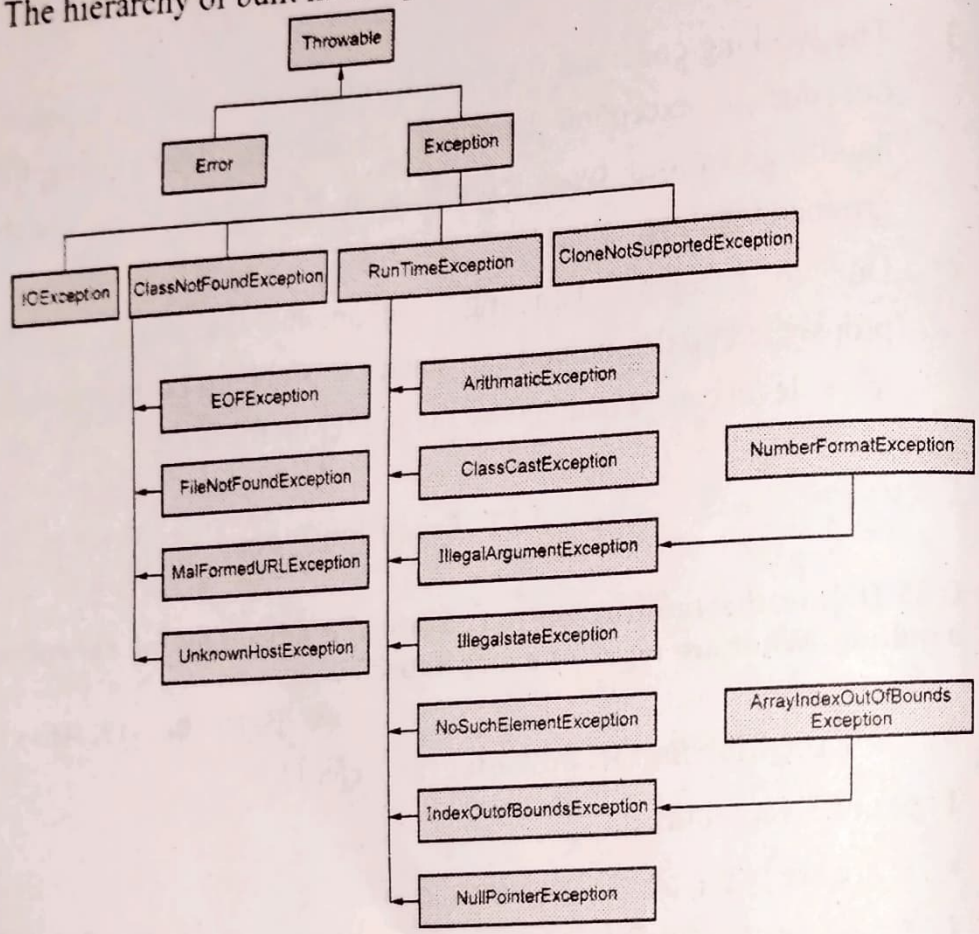


Fig. Q.35.1 Exception hierarchy

4.22 : try, catch, throw, throws, finally

Q.36 What is an exception in Java ? What do you mean by handling an exception ? Give example to show the use of try( ), Catch( ) methods. ☞[SPPU : Nov.-18, Marks 6]

Ans. : Exception : Refer Q.34.

Q.37 Describe Exception. Write any two examples of exception. Explain keywords try, catch, throw, throws and finally related to exception handling.

☞ [SPPU : Dec.-22, Marks 8]

Ans. : Refer Q.34.

Various keywords used in handling the exception are -

try - A block of source code that is to be monitored for the exception.

catch - The catch block handles the specific type of exception along with the try block. Note that for each corresponding try block there exists the catch block.

finally - It specifies the code that must be executed even though exception may or may not occur.

throw - This keyword is used to throw specific exception from the program code.

throws - It specifies the exceptions that can be thrown by a particular method.

Q.38 Write a program in Java to calculate the value of ((x + y) / (x – y)). Program should prevent the condition x – y = 0.

☞ [SPPU : Dec.-17, Marks 6]

Ans. :

```
import java.util.Scanner;
public class ExpressionEval
{
    public static void main(String args[])
    {
        int x,y,N,D;
        float result;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter value of x: ");
        x = input.nextInt();
        System.out.println("Enter value of y: ");
        y = input.nextInt();
```

```
        try
        {
          N=x+y;
          D=x-y;
          result=N/D;
          System.out.println("The Result = "+result);
        }
        catch(ArithmeticException e)
        {
          System.out.println(e);
        }
    }
}
```

4.23 : Uncaught Exceptions

Q.39 What are uncaught exceptions? State the use of try(),catch(),throw() methods

[SPPU : May-17,19, Marks 6]

Ans. : Uncaught Exception : If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.

- Following example illustrates how Java runtime system deals with an **uncaught exception**.
- When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs.

Following Java program illustrates it

Java Program [ExceptionProg.java]

```
class ExceptionProg
{
 static void fun(int a[])
 {
  int c;
```

```
    c=a[0]/a[2];
  }
  public static void main(String args[])
  {
   int a[]={10,5};
   fun(a);
  }
}
```

Output

```
F:\>javac ExceptionProg.java

F:\>java ExceptionProg
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
        at ExceptionProg.fun(ExceptionProg.java:6)
        at ExceptionProg.main(ExceptionProg.java:11)

F:\>
```

Program Explanation :

When Java runtime system detects an attempt to use an array index which is out of bound then it constructs the new exception object and throws the corresponding exception. The default handler displays the description of exception, prints the stack trace from the point at which the exception occurred and then terminates the program.

Use of try, catch and throw :

Refer Q.37.

4.24 : Multiple Catch Clauses

Q.40 How to use multiple catch statements in exception handling ? Explain with suitable example.

Ans. : • It is not possible for the try block to throw a single exception always.

- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.

- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

```
try
    {
        ...
        ...//exception occurs
    }
    catch(Exception_type e)
    {
        ...//exception is handled here
    }
    catch(Exception_type e)
    {
        ...//exception is handled here
    }
    catch(Exception_type e)
    {
        ...//exception is handled here
    }
```

Example

Java Program[MultipleCatchDemo.java]

```
class MultipleCatchDemo
{
  public static void main (String args [ ])
  {
    int a[] = new int [3];
```

```
    try
    {
      for (int i = 1; i <=3; i++)
      {
        a[i] = i *i;
      }

      for (int i = 0; i <3; i++)
      {
        a[i] = i/i;
      }
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println ("Array index is out of bounds");
    }
    catch (ArithmeticException e)
    {
      System.out.println ("Divide by zero error");
    }
  }
}
```

Output

```
Array index is out of bounds
```

4.25 : Nested try Statements

Q.41 Write short note on - Nested try statements

Ans. : • When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.

- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

```
class NestedtryDemo
{
  public static void main (String[] args)
  {
    try
    {
      int a = Integer.parseInt (args [0]);
      int b = Integer.parseInt (args [1]);
      int ans = 0;

      try
      {
        ans = a / b;
        System.out.println("The result is "+ans);
      }
      catch (ArithmeticException e)
      {
        System.out.println("Divide by zero");
      }
    }
    catch (NumberFormatException e)
    {
      System.out.println ("Incorrect type of data");
    }
  }
}
```

Inner try-catch

Outer try-catch

Output

```
D:\>javac NestedtryDemo.java

D:\>java NestedtryDemo 20 10
The result is 2

D:\>java NestedtryDemo 20 a
Incorrect type of data

D:\>java NestedtryDemo 20 0
Divide by zero
```

Outer catch handles the error

Inner catch handles the error

4.26 : Built-in Exceptions

Q.42 What are the types of exceptions ? Explain any three built-in Exceptions with suitable example

☞ [SPPU : May-19, Marks 6]

Ans. : Types of exceptions : Refer Q.35.

Built-in exceptions :

- Various common exception types and causes are enlisted in the following table -

Exception	Description
ArithmeticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.

IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound , this exception will be caused.
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur.
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused.
RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception
ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.

IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

Q.43 State with example the use of the following built in exceptions in Java :

1) IndexOutOfBoundsException( )

2) NullPointerException( )

3) ArrayIndexOutOfBoundsException( )

☞ **[SPPU : Dec.-17, Marks 6]**

Ans. : i) IndexOutOfBoundsException() : This exception is thrown when a program attempts to access a value in an indexable collection in such a way that the value of index is outside the valid range of indices. Following example program shows that the string is of length 5 and we try to access a character at index 10. Due to this the indexoutof bounds is thrown.

Java Program

```
import java.io.*;
class IndexExceptionDemo
{
    public static void main(String [] args)
    {
        String str="India";
        try
        {
          char ch=str.charAt(10);
          System.out.print(ch);
        }
        catch(IndexOutOfBoundsException e)
        {
          System.out.println("Error!!! IndexOutOfBoundsException
                                caught");
```

```
        }
    }
}
```

Output

```
Error!!! IndexOutOfBoundsException caught
```

ii) NullPointerException() :

The NullPointerException occurs when program attempts to use an object reference that has NULL value. Following Java program illustrates this type of exception.

Java Program

```
import java.io.*;
class NullPtrExceptionDemo
{
    public static void main(String [] args)
    {
        String str=null;
        try
        {
          if(str.equals("India"))
            System.out.print(str);
        }
        catch(NullPointerException e)
        {
          System.out.println("Error!!! NullPointer Exception caught");
        }
    }
}
```

Output

```
Error!!! NullPointer Exception caught
```

iii) ArrayIndexOutOfBoundsException() :

When array index is out of bounds then this exception is caused.

```
class ExceptionProg
{
 static void fun(int a[]) throws ArrayIndexOutOfBoundsException
      {
```

```
    int c;
    try
    {
     c=a[0]/a[2];//Exception occurs due to this line
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
    System.out.println("Caught exception: "+e);
    }
}
public static void main(String args[])
{
 int a[]={10,5};
 fun(a);
}
}
```

Then we will get following output

Output

```
F:\>javac ExceptionProg.java
F:\>java ExceptionProg
Caught exception: java.lang.ArrayIndexOutOfBoundsException: 2
F:\>
```

4.27 : Custom Exceptions (Creating Your Own Exception Sub Classes)

Q.44 With illustrative example explain how to create your own exception.

Ans. :

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

  throw new Throwable's subclass

Here the Throwable's subclass is actually a subclass derived from the **Exception** class.

For example -

```
throw new ArithmeticException();
```

Throwable's subclass

Let us see a simple Java program which illustrates this concept.

Java Program[MyExceptDemo.java]

```
import java.lang.Exception;
class MyOwnException extends Exception
{

  MyOwnException(String msg)
   {
     super(msg);
   }
}
class MyExceptDemo
{
   public static void main (String args [])
   {
     int age;
     age=15;
     try
     {
        if(age<21)
       throw new MyOwnException("Your age is very less than the
condition");

     }
     catch (MyOwnException e)
     {
       System.out.println ("This is My Exception block");
       System.out.println (e.getMessage());
     }
     finally
```

```
    {
        System.out.println ("Finally block:End of the program");
    }
  }
}
```

Output

```
This is My Exception block
Your age is very less than the condition
Finally block:End of the program
```

Program Explanation : In above code,

1) the age value is 15 and in the try block - the if condition throws the exception if the value is less than 21.
2) As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block".
3) Then the control is transferred to the **class MyOwnException**(defined at the top of the program). The message is set and it is "Your age is very less than the condition".
4) This message can then printed by the catch block using the **System.out.println** statement by means of **e.message**.
5) At the end the finally block gets executed.

4.28 : Streams

Q.45 What is meant by stream in Java ?

Ans. : • **Stream** is basically a channel on which the data flow from sender to receiver.

- An input object that reads the stream of data from a file is called **input stream** and the output object that writes the stream of data to a file is called **output stream**.

4.29 : Byte Streams and Character Streams

Q.46 What is difference between byte streams and character streams ?

☞ [SPPU : Dec.-17, Marks 3]

Ans. :

Sr.No.	Byte Stream	Character Stream
1.	The byte stream is used for inputting and outputting the bytes.	The character stream is used for inputting and outputting the characters.
2.	There are two super classes used in byte stream and those are - InputStream and OutputStream	There are two super classes used in byte stream and those are - Reader and Writer
3.	A byte is a 8-bit number type that can represent values from – 127 to 127. Ascii values can be represented with exactly 7 bits.	A character is a 16 bit number type that represents Unicode.
4.	It never supports Unicode characters	It supports Unicode characters.

4.30 : Predefined Streams

Q.47 What is meant by stream in Java? How does Java manage Input/Output using streams ?
Differentiate ByteStream, characterStream and Predefined Stream

☞ [SPPU : Nov.-19, Marks 6]

Ans. : Stream : Refer Q.45.

Difference between Byte Stream and character Stream :

Refer Q.46.

Predefined Stream : • For displaying the messages on console we make use of **System.out** statement. There is a class named **System** which is defined in **java.lang** package.

- All Java programs automatically import the java.lang package.
- System contains three predefined stream variables - **in, out** and **err**.
- These are declared public and static within the system.
- The **System.out** defines the standard OutputStream by default. This is **console**.
- The **System.in** refers to standard InputStream which is the **keyboard** by default.
- The **System.err** are the objects for standard input stream and error.

4.31 : Reading Console Input

Q.48 Demonstrate the use of console class to get inputs and show results

☞ [SPPU : Dec.-17, Marks 3]

Ans. :

- In Java input is taken using System.in. For that we wrap **System.in** in a BufferedReader object to create character stream
- Along with it we should also mention the abstract class of BufferedReader class which is InputStreamReader. Hence the syntax is,

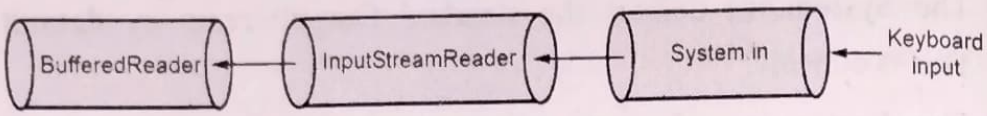
```
BufferedReader object_name =new
BufferedReader(new InputStreamReader(System.in));
```

- For example the object named obj can be created as

```
BufferedReader obj =new BufferedReader(new
InputStreamReader(System.in));
```

- Then, using this object we invoke read() method. For e.g. obj.read(). But this is not the only thing needed for reading the input from console; we have to mention the exception IOException for this purpose. Hence if we are going to use read() method in our program then function main can be written like this -

```
public static void main(String args[]) throws IOException
```



- We can read characters, strings or numeric values from the console.

1. Reading Character

Java program [ReadChar.java]

```
/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadChar
{
 public static void main(String args[])
 throws IOException
 {
  //declaring obj for read() method
  BufferedReader obj=new BufferedReader(new
  InputStreamReader(System.in));
  int count=0;
  char ch;
  System.out.println("\n Enter five characters");
  while(count<5)
  {
```

```
  ch=(char)obj.read();//reading single character
  System.out.println(ch);//outputting it
  count++;//count to keep track of 5 characters
 }
}
}
```

Output

```
Enter five characters
hello
h
e
l
l
o
```

This program allows you take the input from console. As given in above program, **read()** method is used for reading the input from the console. The method **read()** returns the integer value, hence it is typecast to char because we are reading characters from the console.

2. Reading Strings

Java Program [ReadString.java]

```
/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadString
{
public static void main(String args[])
throws IOException
{
BufferedReader obj=new BufferedReader(new
InputStreamReader(System.in));
String s;
s=obj.readLine();//for reading the string
while(!s.equals("end"))
{
System.out.println(s);
```

```
s=obj.readLine();
}
}
}
```

Output

```
hello how are you
hello how are you
end
```

4.32 : Writing Console Output

Q.49 What is character streams and byte streams ? State any two examples of each predefined character and byte stream classes for I/O in Java.

[SPPU : Nov.-18, Marks 7]

Ans. : 1. Byte stream

- The byte stream is used for inputting or outputting the bytes.

(See Fig. Q.49.1 on next page)

- There are two super classes in byte stream and those are **InputStream** and **OutputStream** from which most of the other classes are derived.
- These classes define several important methods such as **read()** and **write()**. The input and output stream classes are as shown in Fig. Q.49.1

2. Character Stream

- The character stream is used for inputting or outputting the characters.
- There are two super classes in character stream and those are **Reader** and **Writer**.
- These classes handle the Unicode character streams. The two important methods defined by the character stream classes are **read()** and **Write()** for reading and writing the characters of data.

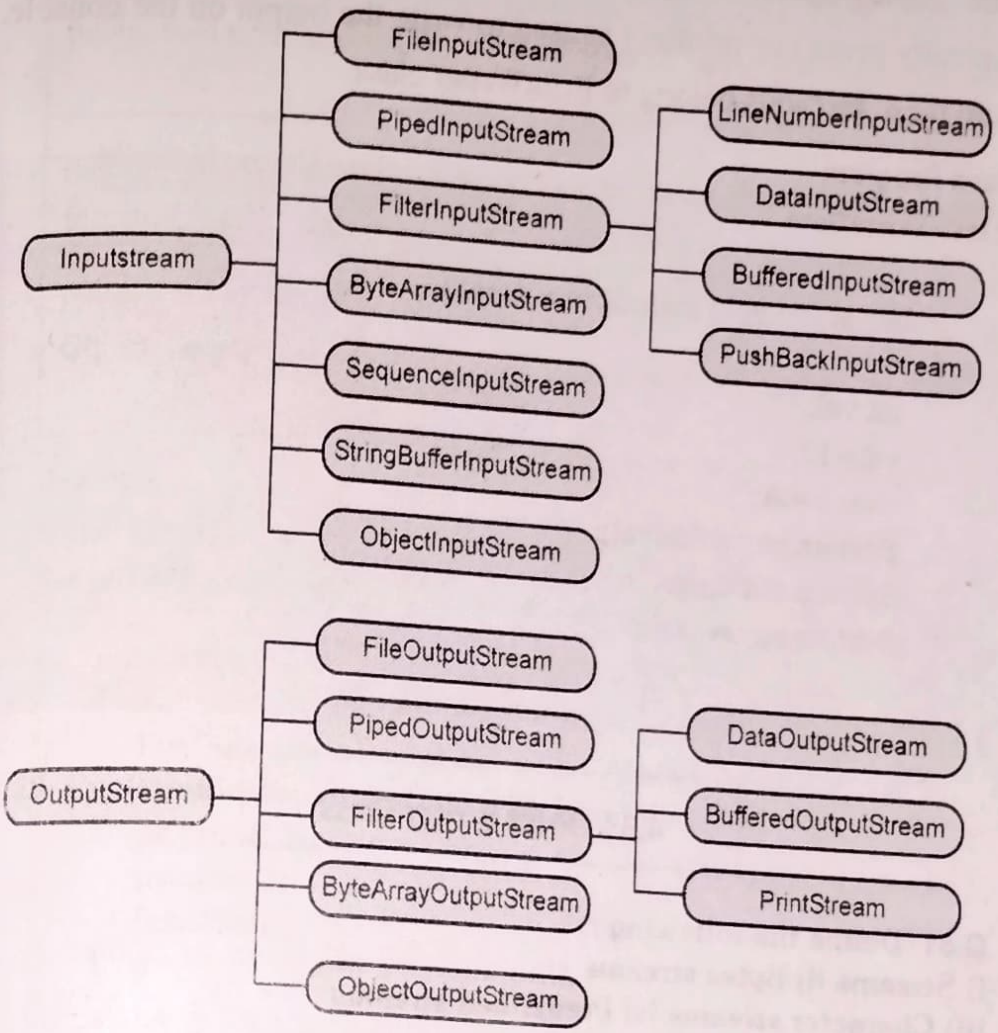


Fig. Q.49.1 Stream classes

Refer Q.50.

Q.50 Explain writing console output with example

Ans. : The simple method used for writing the output on the console is **write()**. Here is the syntax of using write() method -

```
System.out.write(int b)
```

The bytes specified by b has to be written on the console. But typically print() or println() is used to write the output on the console. And these methods belong to PrintWriter class.

Java Program

```
class WriteData
{
    public static void main(String args[])
    {
        int val;
        val=10;
        char c='A';
        System.out.write(val);
        System.out.write('\n');
        System.out.write(c);
    }
}
```

4.33 : Print Writer Class

Q.51 Define the following :
i) Streams ii) Bytes streams
iii) Character streams iv) Predefined streams

Ans. : Refer Q.47 and Q.49.

Q.52 What is the use of PrintStream and PrintWriter classes ? Which methods are supported by these classes ? Give example of each.

☞ **[SPPU : May-17, Marks 6]**

Ans. : PrintStream Class : This class provides methods to write data to another stream. Following are some popularly used methods of PrintStream class.

Methods	Description
public void print(int i)	It prints specific integer value.
public void print(char c)	It prints specific char value.

public void print(String c)	It prints specific string value.
public void println(String c)	It prints specific string value and terminates the line.
public void printf(Locale l, Object format, Object... args)	It writes the formatted string to the current stream
public void format(Object format, Object... args)	It writes the formatted string to the current stream using specified format.

Example -

```
import java.io.*;
class Test
{
    public static void main(String args[])throws Exception
    {
        FileOutputStream fos=new FileOutputStream("input.dat");
        PrintStream ps=new PrintStream(fos);
        ps.print(" Welcome User!!!");
        ps.close();
        fos.close();
    }
}
```

Output

```
D:\>javac Test.java
D:\>java Test
D:\>type input.dat
 Welcome User!!!
```

PrintWriter Class : The System.out is used to write the contents of the console. Similarly the PrintWriter class is also useful in writing the contents to the console.

It is mostly recommended for debugging purpose.

The PrintWriter class is one of the character based classes.

It has several constructors. One of the constructor is

```
PrintWriter(OutputStream outputStream, boolean flushOnNewline)
```

Here, outputStream is an object of type OutputStream and flushOnNewline controls whether Java flushes the output stream every time a println( ) method is called. If flushOnNewline is true, flushing automatically takes place. If false, flushing is not automatic.

This class supports print and println methods.

The writer object for PrintWriter class is created as follows -

PrintWriter pw = new PrintWriter(System.out);

Following Java program shows how to write the contents on the console window

```
import java.io.*;
public class PrintWriterDemo
{
    public static void main(String[] args)
    {
        char c='A';
        int val=100;
        double d = 1234.567;
        // create a new writer
        PrintWriter pw = new PrintWriter(System.out);
        pw.print(c);
        // change the line
        pw.println();
        pw.print(val);
        pw.println();
        pw.print(d);
        // flush the writer
        pw.flush();
    }
}
```

Output

```
A
100
1234.567
```

Q.53 Enlist the methods of PrintWriter class with example

[SPPU : May-19, Marks 7, Dec.-19, Marks 5]

Ans. : Refer Q.52.

Q.54 Explain the Bytestream I/O operations using suitable Java Programs.

Ans. : The FileInputStream class creates an InputStream using which we can read bytes from a file. The two common constructors of FileInputStream are -

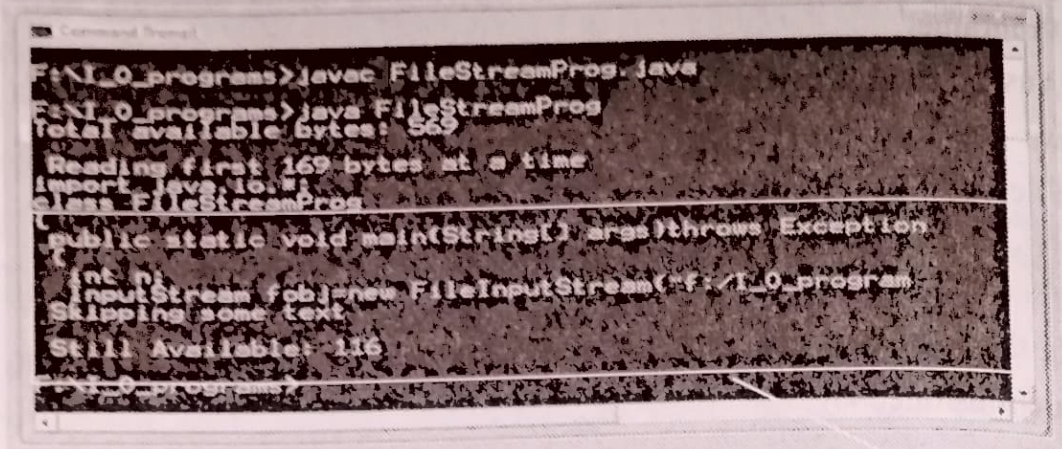
```
FileInputStream(String filename);
FileInputStream(File fileobject);
```

In the following Java program, various methods of **FileInputStream** are illustrated -

Java Program[FileStreamProg.java]

```
import java.io.*;
class FileStreamProg
{
public static void main(String[] args)throws Exception
{
 int n;
 InputStream fobj=new
FileInputStream("f:/I_O_programs/FileStreamProg.java");
 System.out.println("Total available bytes: "+(n=fobj.available()));
 int m=n-400;
 System.out.println("\n Reading first "+m+" bytes at a time");
 for(int i=0;i<m;i++)
 System.out.print((char)fobj.read());
 System.out.println("\n Skipping some text");
 fobj.skip(n/2);
 System.out.println("\n Still Available: "+fobj.available());
 fobj.close();
 }
}
```

Output



Reading this much text from the file

The **FileOutputStream** can be used to write the data to the file using the **OutputStream.** The constructors are-

```
FileOutputStream(String filename)
FileOutputStream(Object fileobject)
FileInputStream(String filename,boolean app);
FileInputStream(String fileobject,boolean app);
```

The *app* denotes that the append mode can be true or false. If the append mode is true then you can append the data in the file otherwise the data can not be appended in the file.

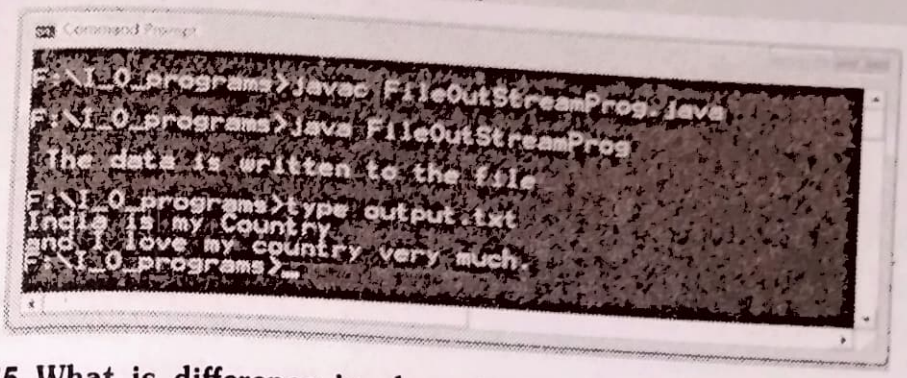
In the following Java program, we are writing some data to the file.

Java Program[FileOutStreamProg.java]

```
import java.io.*;
class FileOutStreamProg
{
 public static void main(String[] args)throws Exception
 {
  String my_text="India is my Country\n"+"and I love my country very
much.";
  byte b[]=my_text.getBytes();
  OutputStream fobj=new
```

```
FileOutputStream("f:/I_O_programs/output.txt");
 for(int i=0;i< b.length;i++)
  fobj.write(b[i]);
 System.out.println("\n The data is written to the file");
 fobj.close();
 }
}
```

Output



Q.55 What is difference in character and byte streams in Java ? Give any two input and any two output classes for character streams.

☞[SPPU : May-18, Marks 6]

Ans. : Difference - Refer Q.46.

Programs using character input classes

A. Output Classes

(1) Using BufferedWriter Class

```
import java.io.*;
public class BufferedWriterProgram {
    public static void main(String[] args) throws Exception {
        FileWriter writer = new FileWriter("output.txt");
        BufferedWriter buffer = new BufferedWriter(writer);
        String str="Hello friends, how are you?";
        buffer.write(str);
        buffer.close();
        System.out.println("Written Data to the File...");
    }
}
```

(2) Using StringWriter Class

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

```
public class StringWriterProgram {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("output.txt");
            StringWriter sw = new StringWriter();
            sw.write("Hello friends, how are you?");
            fw.write(sw.toString());
            sw.close();
            fw.close();
            System.out.println("Written Data to the file...");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

B. Input Classes

(1) Using BufferedReader Class

```
import java.io.*;
public class BufferedReaderProgram {
    public static void main(String[] args) throws Exception {
        FileReader reader = new FileReader("output.txt");
        BufferedReader buffer = new BufferedReader(reader);
        int i;
        System.out.println("\tThe contents of the file are...");
        while((i=buffer.read())!=-1) {
            System.out.print((char)i);
        }
        buffer.close();
        reader.close();
    }
}
```

(2) Using StringReader Class

```
import java.io.*;
public class StringReaderProgram {
    public static void main(String[] args) {
        String str = "Hello friends, how are you?";
```

```
        StringReader reader = new StringReader(str);
        // create a char array to read chars into
        char buff[] = new char[50];
        try {
          // read characters into an array.
          reader.read(buff, 0,str.length());
          System.out.println(buff);
          reader.close();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
    }
}
```

Q.56 What is the use of CharacterArrayReader( ) and CharacterArrayWriter( ) methods in Java ? Write a program which reads string of 10 characters from the user. Program extracts and prints the substring from the given string using above methods.

Ans. : ☞ [SPPU : May-18, Marks 7]

CharArrayReader : The Java CharArrayReader class allows to read the contents of a char array as a character stream.

CharArrayWriter : It makes it possible to write characters via the Writer methods and convert the written characters into a char array.

Java Program :

```
import java.io.*;
import java.util.*;
class Test
{
    public static void main(String arg[])throws IOException
    {
        System.out.println("Enter a string of 10 characters: ");
        Scanner input = new Scanner(System.in);
        String str=input.nextLine();
        if(str.length()>10)
          str=str.substring(0,9);
        char c[]=str.toCharArray();
```

```
        CharArrayReader car = new CharArrayReader(c);
        int i;
        System.out.println("Displaying the string using
CharArrayReader");
        while( (i = car.read()) != -1)
          System.out.print((char)i);
        System.out.println("\nDisplaying the substring");
        CharArrayWriter caw = new CharArrayWriter();
        caw.write(str,2,5);
        System.out.println(caw.toString());
    }
}
```

END...

Unit V

5 Multithreading in Java

Part I : Concurrency and Synchronization

5.1 : Java Thread Model

Q.1 Define thread.

Ans. : Definition of thread : Thread is a tiny program running continuously. It is sometimes called as light-weight process.

Q.2 Interpret the terms multitasking and multiprocessing and multithreading in Java with example.

☞ [SPPU : June-22, Marks 6]

Ans. : Multitasking : When a single resource is used to process multiple tasks then it is called multitasking.

Multiprocessing : When more than one processing unit is used by single computer to perform some task then it is called multiprocessing.

Multithreading : When more than one thread is used to execute the tasks on a single CPU then it is called multithreading.

Java Program for multithreading : Refer Q.5.

Q.3 Differentiate multiprocessing and multi-threading

☞ [Dec.-22, Marks 6]

Ans. :

Sr. No.	Multiprocessing	Multi-threading
1.	In multiprocessing, more CPUs are added to increase the computing power.	In multi-threading, multiple threads are created to increase the computing power.

2.	In this, multiple processes execute concurrently.	In this, multiple threads of a single process execute concurrently.
3.	Multiprocessing can be symmetric and asymmetric.	Multithreading is not classified.
4.	The creation of a process is slow and resource specific.	The creation of multiple thread is economical in resource and time.
5.	In multiprocessing, every process owns a separate address space.	In multithreading, a common address space is shared by all the threads.
6.	Less time is taken for job processing.	A moderate amount of time is taken for job processing.

Q.4 Explain life cycle methods of thread model in Java.

☞ [SPPU : Dec.-22, Marks 6]

Ans. : The Thread model represents the life cycle which specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.

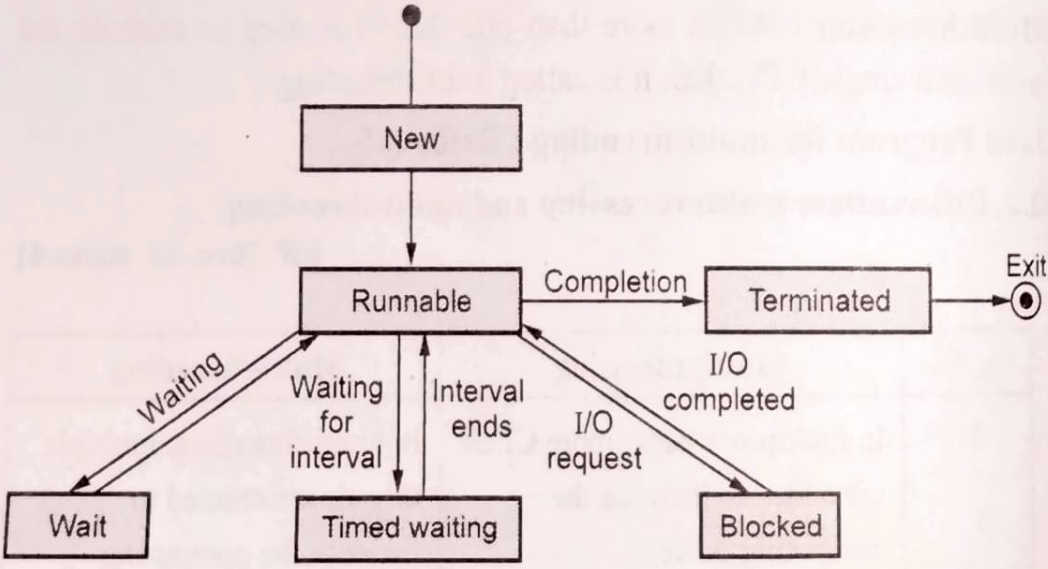


Fig. Q.4.1 Thread model

Let us get introduced with each of these methods one by one -

1) New state

- When a thread starts its life cycle it enters in the new state or a create state.

2) Runnable state

- This is a state in which a thread starts executing.
- Waiting state.
- Sometimes one thread has to undergo in waiting state because another thread starts executing.

3) Timed waiting state

- There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.

4) Blocked state

- When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

5) Terminated state

- After successful completion of the execution the thread in runnable state enters the terminated state.

5.2 : Creating Thread : Implementing Thread using Thread Class and Runnable Interface

Q.5 Explain different methods of creating threads in Java

Ans. : In Java, thread can be created using two approaches -

1) Using **Thread** class 2)Using **runnable** interface.

Example program using thread class

Java Program

```
class MyThread extends Thread
{
  public void run()
  {
    System.out.println("Thread is created!!!");
  }
}
class ThreadProg
{
  public static void main(String args[])
  {
    MyThread t=new MyThread();
    t.start();
  }
}
```

Example program using runnable interface

Java Program

```
class MyThread implements Runnable
{
  public void run()
  {
    System.out.println("Thread is created!");
  }
}
class ThreadProgRunn
{
  public static void main(String args[])
  {
    MyThread obj=new MyThread();
    Thread t=new Thread(obj);
    t.start();
  }
}
```

5.3 : Thread Priorities

Q.6 Elaborate the terms getPriority() and setPriority() methods with example.

Ans. :

☞ [June-22, Marks 6]

- The function **setPriority** is used to set the priority to each thread

```
Thread_Name.setPriority(priority_val);
```

The *priority_val* is a constant value denoting the priority for the thread. It is defined as follows -

  - MAX_PRIORITY = 10
  - MIN_PRIORITY = 1
  - NORM_ PRIORITY = 5

- The function **getPriority** is used to get the priority of the thread.

```
Thread_Name.getPriority();
```

- **Preemption**
  - Preemption is a situation in which when the currently executed thread is suspended temporarily by the highest priority thread.
  - The highest priority thread always preempts the lowest priority thread.

Q.7 Write a Java program illustrating the concept of thread priority.

Ans. :

```
class A extends Thread
{
 public void run()
 {
  System.out.println("Thread #1");
  for(int i=1;i<=5;i++)
  {
   System.out.println("\tA: "+i);
  }
  System.out.println("\n-----End of Thread#1-------");
```

```
  }
}
class B extends Thread
{
 public void run()
 {
   System.out.println("Thread #2");
   for(int k=1;k<=5;k++)
   {
     System.out.println("\tB: "+k);
   }
     System.out.println("\n-----End of Thread#2-------");
  }
}
class Thread_PR_Prog
{
 public static void main(String[] args)
 {
   A obj1=new A();
   B obj2=new B();
   obj1.setPriority(1);
   obj2.setPriority(10);//highest priority
   System.out.println("Starting Thread#1");
   obj1.start();
   System.out.println("Starting Thread#2");
   obj2.start();
  }
 }
```

Output

```
Starting Thread#1
Starting Thread#2
Thread #1
Thread #2
```

Thread 1 and Thread 2 starts

```
B: 1
B: 2
B: 3
B: 4
B: 5
-----End of Thread#2-------
A: 1
A: 2
A: 3
A: 4
A: 5
-----End of Thread#1-------
```

Thread 2 preempts the execution of thread 1

Then Thread 1 continues its execution

5.4 : Synchronization Messaging

Q.8 Explain the term thread synchronization. Explain how to achieve thread synchronization in Java.

☞ [SPPU : Dec.-22, Marks 6]

Ans. : • **Definition :** When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called **synchronization**.

- There are two ways to achieve the synchronization -

1. Using synchronized methods
2. Using synchronized blocks (statements).

1. Using synchronized method

```
class Test
{
synchronized void display(int num)
```

This is a synchronized method

```
{
    System.out.println("\nTable for "+num);
    for(int i=1;i<=10;i++)
    {
        System.out.print(" "+num*i);
    }
    System.out.print("\nEnd of Table");
    try
    {
    Thread.sleep(1000);
    }catch(Exception e){}
}
}
class A extends Thread
{
    Test th1;
    A(Test t)
    {
        th1=t;
    }
    public void run()
    {
        th1.display(2);
    }
}
class B extends Thread
{
    Test th2;
```

```
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}
class MySynThread
{
    public static void main(String args[])
    {
        Test obj=new Test();
        A t1=new A(obj);
        B t2=new B(obj);
        t1.start();
        t2.start();
    }
}
```

Output

```
D:\>javac MySynThread.java

D:\>java MySynThread

Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
 100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

2. **Using synchronized block**

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.

- Synchronized statements must specify the object that provides the intrinsic lock.

Syntax

The syntax for using the synchronized block is -

```
synchronized(object reference )
{
statement;
statement;  //block of code to be synchronized
.
.
.
}
```

Java Program

```
class Test
{
   void display(int num)
   {
     synchronized(this)
     {
      System.out.println("\nTable for "+num);
      for(int i=1;i<=10;i++)
      {
         System.out.print(" "+num*i);
      }
      System.out.print("\nEnd of Table");
      try
       {
          Thread.sleep(1000);
       }catch(Exception e){}
     }
   }
}
class A extends Thread
{
   Test th1;
   A(Test t)
```

This is a synchronized Block

```
  {
    th1=t;
  }
  public void run()
  {
    th1.display(2);
  }
}
class B extends Thread
{
  Test th2;
  B(Test t)
  {
    th2=t;
  }
  public void run()
  {
    th2.display(100);
  }
}
class MySynThreadBlock
{
  public static void main(String args[])
  {
    Test obj=new Test();
    A t1=new A(obj);
    B t2=new B(obj);
    t1.start();
    t2.start();
  }
}
```

Output

```
Command Prompt

D:\>javac MySynthreadBlock.java

D:\>java MySynthreadBlock

Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
 100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

5.5 : Creating Multiple Threads using is Alive() and join()

Q.9 Explain the uses of isAlive() and Join methods in Java thread with example.

☞ [SPPU : June-22, Marks 6]

Ans. : 1. The isAlive() method : This method is used to check if the thread is alive or not. This method returns **true** if thread is still running and not finished otherwise it returns false.

Syntax

```
public final boolean isAlive()
```

Java Program

```
public class ThreadDemo extends Thread
{
    public void run()
    {
        System.out.println("Begin");
        try {
          Thread.sleep(500);
        }
        catch(InterruptedException ie)
```

```
        {
        }
        System.out.println("End");
    }
    public static void main(String[] args)
    {
        ThreadDemo t1=new ThreadDemo();
        ThreadDemo t2=new ThreadDemo();
        t1.start();
        System.out.println("Is Thread1 alive? "+t1.isAlive());
        System.out.println("Is Thread2 alive? " +t2.isAlive());
        t2.start();
        System.out.println("Now Is Thread2 alive?" +t2.isAlive());
    }
}
```

Output

```
C:\Windows\System32\cmd.exe
D:\JavaPrograms>javac ThreadDemo.java

D:\JavaPrograms>java ThreadDemo
Begin
Is Thread1 alive? true
Is Thread2 alive? false
Now Is Thread2 alive? true
Begin
End
End

D:\JavaPrograms>
```

2. The join() Method

This method waits until the thread on which it is called terminates.

Syntax

```
final void join() throws InterruptedException
```

Using **join** method, we tell our thread to wait until the specified thread completes its execution.

Java Program

```
public class ThreadDemo extends Thread
{
    public void run()
    {
        System.out.println("Begin");
        try {
          Thread.sleep(500);
        }
        catch(InterruptedException ie)
        {
        }
          System.out.println("End");
    }
    public static void main(String[] args)
    {
        ThreadDemo t1=new ThreadDemo();
        ThreadDemo t2=new ThreadDemo();
        System.out.println("Starting first thread");
        t1.start();
        try
        {
          t1.join();//waiting for t1 to finish
        }
        catch(InterruptedException e)
        {
        }
        System.out.println("Starting another thread");
        t2.start();
    }
}
```

Part II : Web based Application in Java

5.6 : Use of JavaScript for Creating Web based Applications in Java

Q.10 Enlist the features of JavaScript.

Ans. : Following are some features of JavaScript

1. **Browser support :** For running the JavaScript in the browser there is no need to use some plug-in. Almost all the popular browsers support JavaScripting.
2. **Structure programming syntax :** The Javascript supports much commonly the structured language type syntax. Similar to C-style the programming blocks can be written.
3. It automatically inserts the semicolon at the end of the statement, hence there is no need to write semicolon at the end of the statement in JavaScript.
4. **Dynamic typing :** It supports dynamic typing, that means the data type is bound to the value and not to the variable. For example one can assign integer value to a variable say 'a' and later on can assign some string value to the same variable in JavaScript.
5. **Run time evaluation :** Using the **eval** function the expression can be evaluated at run time.
6. **Support for object :** JavaScript is object oriented scripting language. However handling of objects in JavaScript is somewhat different that the conventional object oriented programming languages. JavaScript has a small number of in-built objects.
7. **Regular expression :** JavaScript supports use of regular expressions using which the text-pattern matching can be done. This feature can be used to validate the data on the web page before submitting it to the server.

Q.11 List the features of JavaScript. Write JavaScript program to display Welcome message.

[SPPU : Dec.-22, Marks 6]

Ans. : Features : Refer Q.10.

JavaScript Program to display Welcome message

```
<!DOCTYPE html >
<html >
    <head>
        <title> My First Javascript </title>
    </head>
    <body>
        <center>
        <script type="text/javascript">
        /*This is the First JavaScript*/
        document.write(" Welcome to First Page of Javascript");
        </script>
        </center>
    </body>
</html>
```

Q.12 Explain the ways by which the JavaScript can be embedded in web document.

Ans : The JavaScript can be directly embedded within the HTML document or it can be stored as external file.

Directly embedded JavaScript

The syntax of directly embedding the JavaScript in the HTML is

```
<script type="text/javascript">
        ...
        ...
        ...
</script>
```

Indirectly Embedding JavaScript

If we want to embed the JavaScript indirectly, that means if the script is written in some another file and we want to embed that script in the HTML document then we must write the script tag as follows -

```
<script type="text/javascript" src="MyPage.js">
    ...
    ...
    ...
</script>
```

Javascript is which is to be embedded is in the file MyPage.js

Q.13 Write JavaScript code for login page form.

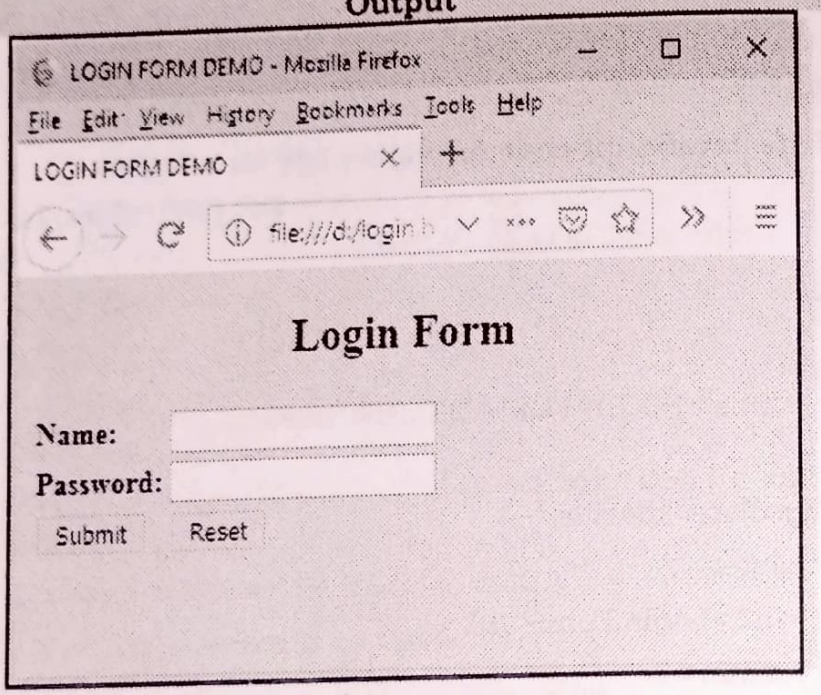
[SPPU : June-22, Marks 6]

Ans. :

```
<html>
<head>
        <title>LOGIN FORM DEMO</title>
</head>
<body bgcolor="khaki">
      <center>
        <h2>Login Form</h2>
      </center>
      <form name="form1">
        <table>
          <tr>
          <td><b>Name:</b></td>
          <td><input type="text" name="userName">
          </tr>
          <tr>
          <td><b>Password:</b></td>
          <td><input type="password"name="pwd"></td>
          </tr>
          <tr>
          <td><input type="submit" name="submit"
```

```
            value="Submit"></td>
        <td><input type="reset" name="reset"
            value="Reset"></td>
        </tr>
      </table>
    </form>
</body>
</html>
```

Output



5.7 : Introduction to Java Script Frameworks - ReactJS, VueJS, AngularJS

Q.14 List the features, advantages and limitations of angular JS.

☞ [SPPU : June-22, Marks 6]

Ans. : Advantages

- Following are the advantages of Angular JS

1. **Built by Google Engineers :** The Angular JS is maintained by dedicated Google Engineers. That means - there is huge community for development and enhancement.

2. **MVC Support** - It is based on model view controller (MVC) architecture which provides separation of model, view and controller components.
3. **Intuitive** - It is more intuitive as it makes use of HTML as a declarative language. Moreover, it is less brittle for reorganizing.
4. **Comprehensive :** AngularJS is a comprehensive solution for rapid front-end development. It does not need any other plugins or frameworks.
5. **Rich Features :** There is a wide range of features supported by Angular JS such as Data binding, dependency injection, enterprise level testing and so on.
6. **Unit Testing :** The Angular JS code is unit testable.
7. **Reusable Code :** Angular JS support for using the reusable components.
8. **Less Code :** It support for less code and more functionality.

Disadvantages

1. As Angular JS is based on JavaScript framework, it is not secure.
2. There are multiple ways to do the same thing with AngularJS. Sometimes, it can be hard for novices to say which way is better for a task.

Features of angular JS

- There are core features Angular JS that are widely used by web developers. These are -

1. **Data binding :** It allows the automatic synchronization between model and view components.
2. **Scope :** There are some objects from model that can be correlated to controller and view.
3. **Controller :** These are basically JavaScript functions that are bound to particular scope.

4. **Directives :** The directives is designed to give HTML new functionality.
5. **Filters :** This feature allows to select subset from array of items.
6. **Routing :** This feature allows to switch between multiple views.
7. **Services :** There is a support for many built in services for Angular JS.
8. **Dependency Injection :** It is a software design pattern that helps the developer to develop and understand the application easily.

Q.15 Write short note on Angular JS, List the advantages and disadvantages ☞ [SPPU : Dec.-22, Marks 6]

Ans. : Refer Q.14.

Q.16 Write short note on ReactJS. Enlist and explain the features of it. ☞ [SPPU : Dec.-22, Marks 6]

Ans. : • ReactJS is an open source, component-based front end JavaScript library maintained by Facebook.

- This library is responsible only for the view layer of the application. That means this JavaScript for building user interfaces.

Features of ReactJS

- **Virtual DOM :** DOM stands for Document Object Model. It also provides a way to update the content, structure and style of the document. Virtual DOM is a representation of original DOM. When user updates something on the web application, DOM gets updated. Updating the DOM is very slow, most of the javascript frameworks update the whole DOM which makes it slower. But actually there is no need to update the whole DOM, instead, these frameworks should update only the part of DOM that is required to update. This is what the virtual DOM does. This is why ReactJS's DOM manipulation is much faster than other frameworks.Due to this property, whenever any change is made in the web application, then those changes are reflected on the web page within no time.

- **Components :** This feature allows the web developer to create custom elements which can be reused in HTML.
- **JSX :** JSX is an extension of JavaScript syntax. JSX can be seen as a combination of javascript and XML. The syntax of JSX is very simple that makes writing components very easy.
- **One way Data Binding :** The ReactJS is designed in such a way that it follows, unidirectional or one way data binding. That means, data is allowed to flow in one direction at a time. This helps in achieving greater control over the application. This makes helps in increasing the efficiency of an application.

Q.17 Write a ReactJS application to display the "welcome user" message.

Ans. :

Step 1 : Create your project folder(Refer steps mentioned in the installation process).

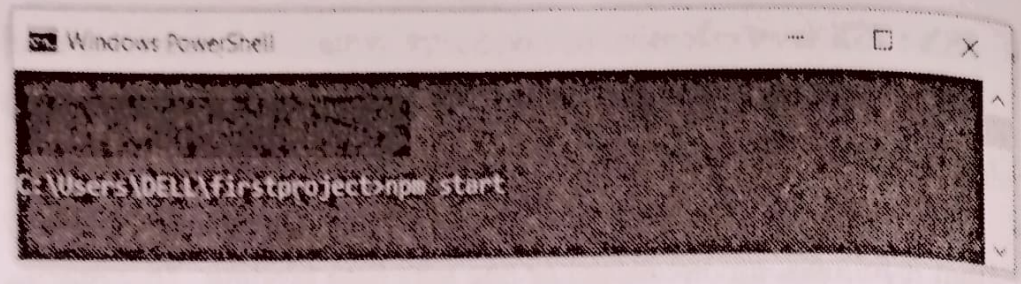
Open the file **index.js** This file is in your project under the **src** folder. Edit is as follows -

```
import React from 'react';
import ReactDOM from 'react-dom';
const mystr = <h1>Welcome User!!!</h1>
ReactDOM.render(mystr, document.getElementById('root'));
```

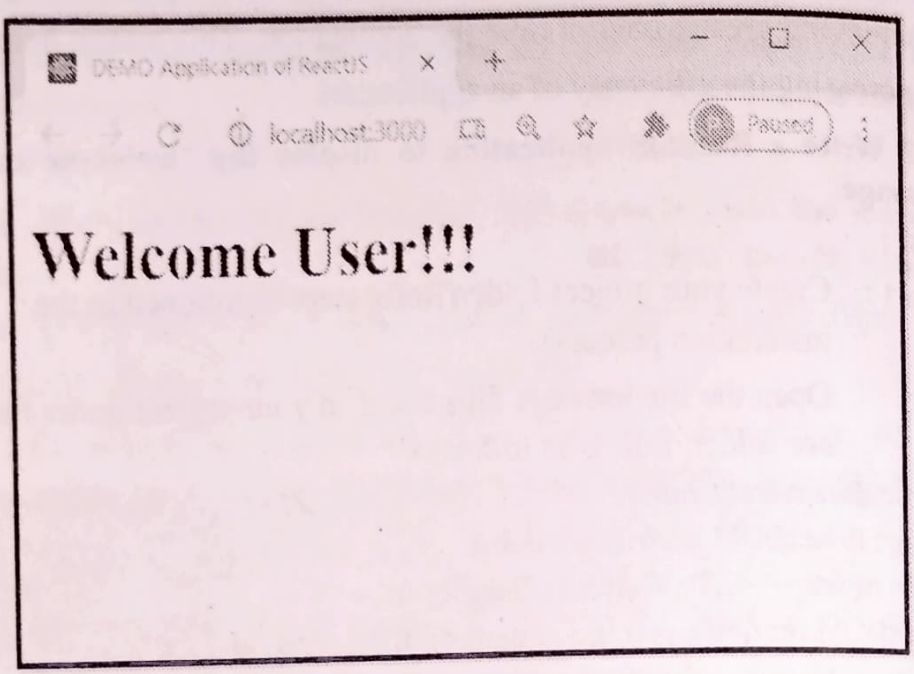
Step 2 : Now open index.html file. This file is present in your folder's public folder. Edit it as follows -

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-
   scale=1" />
  <title>DEMO Application of ReactJS</title>
 </head>
 <body>
  <div id="root"></div>
 </body>
</html>
```

Step 3 : Now open the command prompt window and type following command



The browser window will open and will display the output as follows -



Q.18 Compare react JS and angular JS and Vue JS.

☞ [SPPU : June-22, Marks 6]

Ans. :

Sr. No.	Angular JS	React JS	Vue JS
1.	Angular JS is completely based on Model View Controller(MVC) architecture.	React JS is a library that just offers a view. The developer has to decide how to construct model and controller.	Vue.js is a library that allows you to create interactive web interfaces. It is concerned about view and model of the architecture.

2.	It is a steep learning curve.	It is easier to learn that Angular JS.	There is a small learning curve for Vue.js.
3.	Angular JS uses two way data binding.	React JS uses one way data binding.	Vue.js uses one way data binding.
4.	Angular JS is written in TypeScript.	React JS uses native JavaScript.	Vue.js make use of HTML and JavaScript based template syntax.
5.	If we want to develop large, multifunctional application then Angular JS is used.	If we want to develop lightweight modern apps in short time then we make use of React JS.	For developing Single Page applications with dynamic, high performance that makes use of virtual DOM, we use Vue.js.
6.	It is mainly used by Google, Forbes, Instagram, WhatsApp,	It is mainly used by Facebook, Netflix, Twitter, Walmart.	It is used by Alibaba, Gitlab, Baidu.

Q.19 What is VueJS ? Enlist its features.

Ans. :

- Vue.js is an open source progressive framework for JavaScript used to build web interfaces and one-page applications.
- Vue.js is also used for both Desktop and mobile app as a front end development tool.
- The name Vue is for resembling the word "view".

Features of VueJS

1) Components : This feature allows the web developer to create custom elements which can be reused in HTML.
2) Event Handling : Any event can be handled using VueJS script with the help of **v-on** attribute.
3) Data Binding : This feature helps to manipulate or assign values to HTML attributes or change the style.
4) Animation : Vue JS helps to show animation of the elements on the web page.
5) Computed Properties : This is a feature that can immediately sense the changes made in the UI and make necessary actions. There is no need to perform additional coding for that.
6) Ligheweight and Efficient : The Vue JS is very light weight and efficient JavaScript framework.

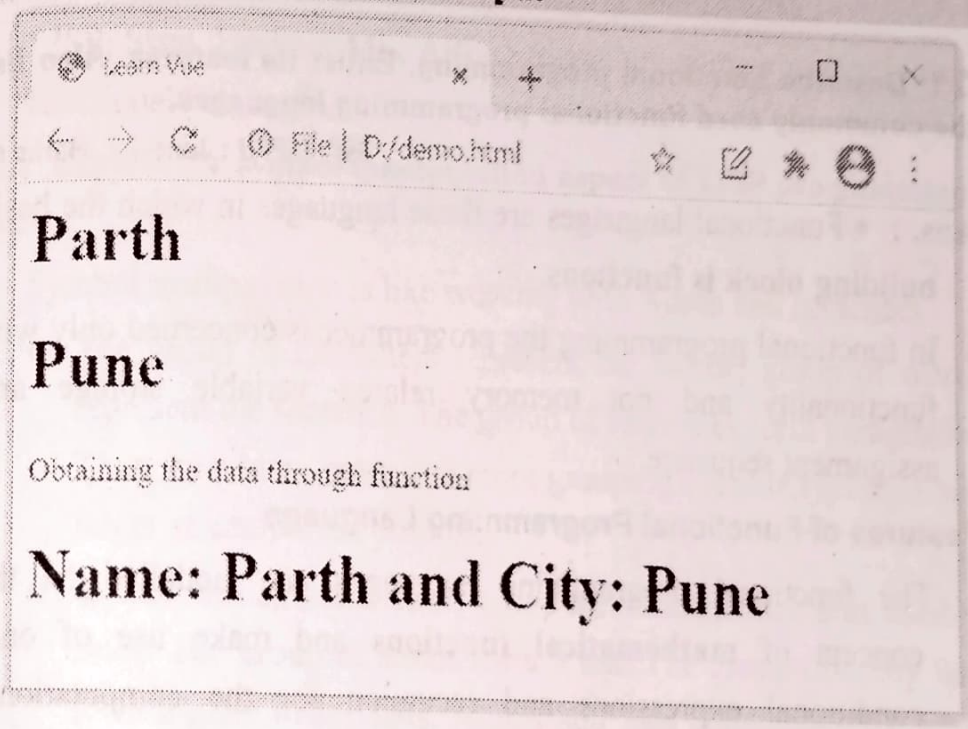
Q.20 Write VueJS script to display the name and city of the user.

Ans. :

```
<html>
<head>
</script>
 <title>
   Learn Vue
 </title>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
    <div id = "myfirstapp">
       <h1>{{name}}</h1>
       <h1>{{city}}</h1>
       <p> Obtaining the data through function </p>
       <h1>{{myfunction()}}</h1>
    </div>
    <script>
       const myvuecomponent = new Vue({
         el : "#myfirstapp",
```

```
        data : {
          name: "Parth",
          city: "Pune"
        },
        methods: {
          myfunction: function() {
          return "Name: "+this.name+" and City: "+this.city;
          }
          }
      })
    </script>
</body>
</html>
```

Output



END...

Unit VI

6 Logical and Functional Programming

Part I : Functional Programming Paradigm

6.1 : Understanding Symbol Manipulation

Q.1 Describe functional programming. Enlist its features. Also list the commonly used functional programming languages.

☞ [SPPU : June-22, Marks 6]

Ans. : • Functional languages are those languages in which the basic **building block is functions**.

- In functional programming the programmer is concerned only with functionality and not memory related variable storage and assignment sequence.

Features of Functional Programming Language

1. The functional programming languages are modelled on the concept of **mathematical functions** and make use of only conditional expressions and recursion for the computational purpose.
2. The programs are constructed **by building functional applications**. That means, the values produced by one or more functions become the parameters to another functions.
3. The **purest form** of functional programming languages use neither variables nor assignment statements, although this is relaxed somewhat in most applied functional languages.

4. The functional languages can be categorized in two types - Pure functional languages and impure functional languages.
   a. The **pure functional language** support only functional paradigm. Example of pure functional language is ML, Haskell.
   b. The **impure functional language** can be used to write imperative style programs(For example C is a imperative style programming language). Example of impure functional programming language is LISP.

Commonly used Functional Programming Languages

- LISP, Sisal, Haskell, ML, APL, Scheme are some commonly used functional languages.

Q.2 Explain the symbol manipulation aspect of LISP programming.

Ans. :

- Symbol manipulation is like working with words and sentences :
  - The group of characters represent the words, group of words represent the sentence. The group of sentences form paragraphs. Thus, by adding more and more grouping eventually, larger and larger structures are possible.
  - In the similar manner, In Lisp, the fundamental unit used is atom. The group of atoms form a **list.** The group of many lists form **higher order lists.** Atoms and lists collectively called as **symbolic expressions**. Hence symbol manipulation is nothing but **list processing.**
- **Symbol Manipulation is needed to make computers Intelligent :**
  - Nearly all the intelligent programs are written in LISP.
  - As Lisp programs have an ability to perform calculus problems, geometric analogy problems, programs diagnosing the blood infections and many more.

- Lisp is a language in which most research is done on knowledge base and expert systems.
- Lisp also provides support for natural language processing, speech and vision recognition systems.

- **LISP is the right symbol-manipulation Language to Learn.**
  - LISP is oriented towards programming at terminal with quick response.
  - LISP can be better used with editing and debugging tools.
  - LISP is basically list processing language, hence it can be very easily used for symbol manipulation.
  - The function and data form of LISP is very much similar. Hence one LISP function can be used to analyze another.
- **LISP programs are easy to learn.**

6.2 : Basic LISP Functions

Q.3 Write sequences of CAR's and CDR's that will pick the atom pear out of the following s-expression :
i) (apple orange pear grapes) ii) ((apple orange) (pear grapes))
iii) (((apple)(orange) (pear) (grapes)))

☞ [SPPU : June-22, Marks 6]

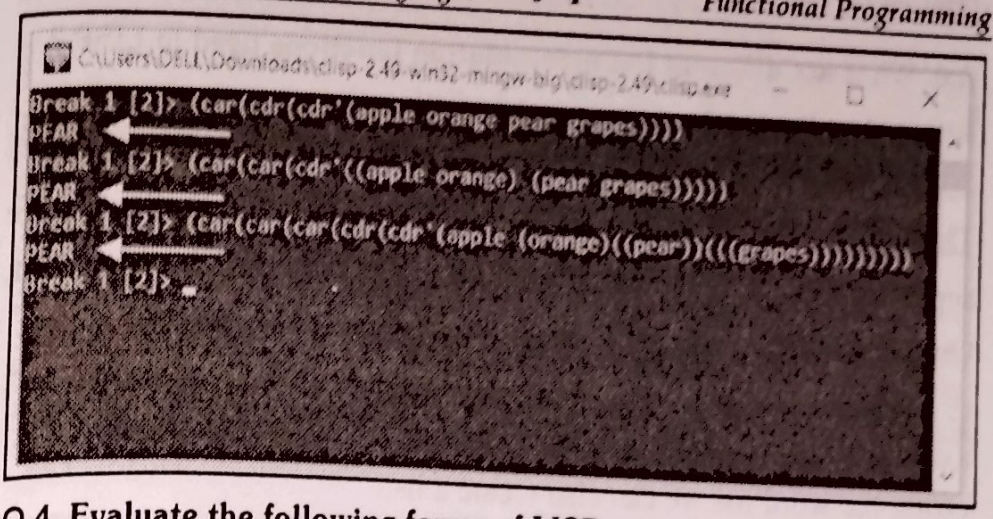
Ans. :

i) (car(cdr(cdr'(apple orange pear grapes))))

ii) (car(car(cdr'((apple orange) (pear grapes)))))

iii) (car(car(car(cdr(cdr'(apple (orange)((pear))(((grapes))))))))

The execution of above statements is as shown by following screenshot



Q.4 Evaluate the following forms of LISP.
i) (car (cdr '(1 2 3 4 5)))
ii) (car (cdr '(a(b c)d e)))
iii) (car (cdr(cdr '(1 2 3 4 5 6 7 8))))

☞ [SPPU : Dec.-22, Marks 5]

Ans. : i) 2 ii) (b c) iii) 3

Q.5 Explain the mathematical functions and eval function used in LISP.

Ans. : Mathematical Functions

- There are various mathematical functions that are illustrated as follows -

```
>(abs -10)
10
>(sqrt 25)
5
>(min 10 20)
10
>(max 10 20)
20
```

Eval Function

- The eval function helps in evaluating an expression. The evaluation function returns the value of the evaluation

```
>(eval (+ 10 20))
30
```

Q.6 What are different list manipulation functions used in LISP ?

Ans. : The list can be represented as follows

```
>(10 20 30)
```

The list is enclosed within the brackets. The elements in the list are separated by space. The list containing no item is called the empty list. On entering the empty list it will return the value NIL.

```
> ( )
NIL
```

Using the word list we can also create a list.

```
> (list 10 20 30)
(10 20 30)
```

There are various functions that can be used for manipulation of lists.

1. Length :

The length of empty list is 0. We can find out the length of the list as follows -

```
> (length '(10 20 30 40 50))
5
```

2. Append :

The append is used to add the element in the list at the end. For example

```
> (append '(10 20 30) '(40 50))
(10 20 30 40 50)
```

We can also create a definition of append function as follows -

```
> (define (append x z)
(cond ((null? x)  z)
(else (cons (car x) (append (cdr x) z))) ))
```

3. Map :

The map is basically applied to a function. Using map we can extend the function from elements to list. Map returns a list of the results of applying the function to elements taken from each list. For example

```
>(map abs '(1 -2 3 -4 -5))
(1 2 3 4 5)
```

```
> (map + '(1 2 3) (4 5 6))
        (5 7 9)
```

Thus map can handle more than one list.

Q.7 Illustrate the use of car, cdr and cons functions used LISP.

Ans. : (1) car, cdr and cons functions

- **The car** returns first element of a non empty list.

For example -

```
> (car '(1 2 3 4))
1
```

- **The cdr** returns rest of the list after the first element is removed. It is pronounced as could-er.

```
> (cdr '(1 2 3 4))
(2 3 4)
```

- We can combine two lists using cons. The cons function takes two arguments and returns a new cons cell containing the two values.

```
> (cons 10 '(20 30 40))
(10 20 30 40)
```

Q.8 Explain the use of first and rest primitives in LISP.

Ans. : • There are two primitives of list selectors in Lisp - **First** and **Rest** First takes a list as an argument and returns the first element of that list. It works like this :

```
>(first '(a e i o u))
a
>(first '((a e) f g))
(a e)
```

- Rest takes a list as an argument and returns the list, minus its first element.

```
>(rest '(a e i o u))
(e i o u).
>(rest '((a e) i o u))
(i o u)
>(rest '((a b) (c d))
((c d))
```

6.3 : Definitions, Predicates, Conditionals and Scoping

Q.9 Explain the following equality predicates using suitable example.
i) Equal ii) EQ iii) EQL iv) =

[SPPU : Dec.-22, Marks 6]

Ans. :

i) **EQUAL :** The equal predicate is true if its arguments are structurally similar (isomorphic) objects. Two objects are equal iff they are eql, strings of eql characters, bit vectors of the same contents, or lists of equal objects. For anything else, eq is used. For example -

```
>(equal (list 1 2 3) (list 1 2 3))
=> T
>(equal "johny" "johny")
=> T
>(equal "hello" "Hello")
=> NIL
```

ii) **EQ :** The (eq x y) is true if and only if x and y are the same identical object. That means x and y are usually eq if and only if they address the same identical memory location. For example -

```
>(eq 'a 'b)
=> NIL

>(eq 'a 'a)
=> T
```

iii) **EQL :** The eql predicate is true if its arguments are eq, or if they are numbers of the same type with the same value, or if they are character objects that represent the same character(the comparison is case sensitive). For example -

```
>eql(10, 10)
=> T
>eql(10, 10.0)
=> NIL
```

iv) =: The = compares only numbers, regardless of type. For example -

```
(= 10 10.0)
=> T
```

Q.10 Explain the following number predicates using suitable example -
i) NUMBERP ii) ZEROP iii) PLUSP iv) EVENP v) ODDP.

☞ [SPPU : June-22, Marks 5]

Ans. : i) NUMBERP : It takes one argument and returns true if the argument is a number, otherwise returns NIL.

```
>(numberp 10)
T
>(numberp 'a)
NIL
```

ii) ZEROP : This predicate returns true if argument is zero. For example -

```
>(zerop 0)
T
```

iii) PLUSP : This predicate returns true if the number is strictly greater than zero and false otherwise. For example -

```
>(plusp 10)
T
>(plusp -10)
NIL
```

iv) EVENP : It takes one numeric argument. If the number is even then it returns T, otherwise it returns NIL. For example -

```
>(evenp 10)
T
>(evenp 11)
NIL
```

v) ODDP : It takes one numeric argument. If the number is odd then it returns T, otherwise it returns NIL

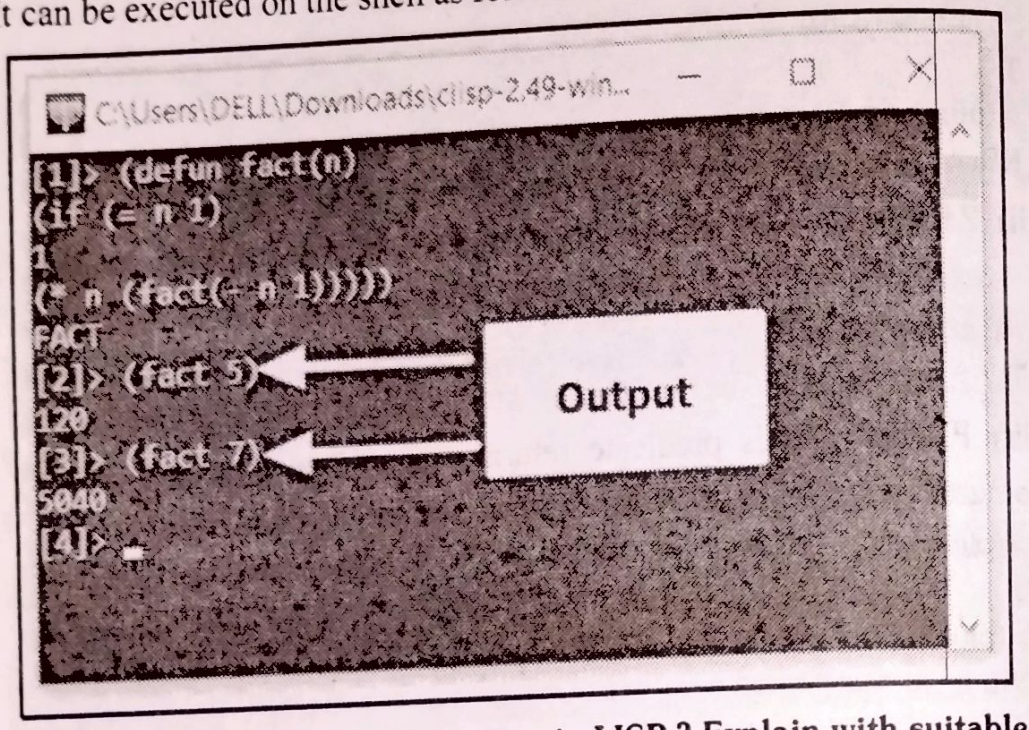
```
>(oddp 10)
NIL
>(oddp 11)
T
```

Q.11 Write a LISP program to find the factorial of n numbers using recursion concept. [SPPU : June-22, Marks 6]

Ans. : • The LISP code for the recursive factorial function is as follows -

```
(defun fact(n)
(if (= n 1)
1
(* n (fact(- n 1)))))
```

It can be executed on the shell as follows -



Q.12 How will you define a function in LISP ? Explain with suitable example.

Ans. : The operator **defun** is used to define the procedure. The syntax of procedure writing in LISP is

```
defun (proc_name(argument1 argument2)
    body of procedure)
```

For example - procedure for addition of two numbers is as follows -

```
> (defun mysum(a b)
(+ a b))
Now we can use this procedure the way we want.
> (mysum 2 3)
5
```

Q.13 Explain various conditional statements used in LISP.

Ans. : • The conditions can be specified using if. The syntax of writing **if** is

```
(if(conditional expression) (true-expression) (false expression))
```

The **true-expression** part will be executed only if the condition specified in **condition expression** is true, otherwise the **false expression** part will be executed. For example

```
>(if(> 10 20) 10)
NIL
```

As 10>20 is false and there is no false expression, NIL will be returned.

We will change the above expression as follows -

```
> (if (> 10 20) 10 20)
20
```

- We can have **nested if statement.** The example is as shown below -

```
> (if(= 10 10) (if(> 5 1) 5 1))
5
```

- If we want to display a series of expression on the if condition then these expressions are represented by a block. For this block **progn** is used. **progn** can take any number of expressions, and evaluates each of its expressions in order. **progn** then returns the value of the last expression.

```
> (if (> 20 10)
(progn
(print "Hello" )
(print "Welcome" ))
(print "Bye" )
)
```

True expression multiple statements are executed for true expression.

False expression.

Output

```
"Hello"
"Welcome"
"Welcome"
```

- The **dotimes** is used for repeatedly executing the statements. The **dotimes** needs a variable. The variable value gets incremented to reach to a value specified by **high-value.** The **dotimes** also requires **optional-return-value** which will be displayed only when the variable reaches to **high-value.**

```
> (dotimes (a 5 "Bye") (print "Hello"))
"Hello"
"Hello"
"Hello"
"Hello"
"Hello"
"Bye"
```

- Similar to **if** the **cond** is used to specify the conditional statements. Following example is an illustrative one.

```
> (cond
((> 20 10) (print "Hi"))
((< 20 10) (print "Hello"))
)
```

Output

```
"Hi"
"Hi"
```

Q.14 Explain the concept of bound and free variables.

Ans. : Concept of Bound and Free variables

- A variable x is **bound** in an expression E if the meaning of E **remains unchanged** by uniform replacement of a variable x by y for every occurrence of x in E.
- For example -

```
(lambda (x) (*x 2))
(lambda (y) (*y 2))
```

- In above expression variable x is bound
- A variable x is **free** in an expression E if the meaning of E **gets changed** by uniform replacement of a variable x by y for every occurrence of x in E.
- For example -

```
(lambda (x) (*x y))
(lambda (y) (*x z))
```

- Here x is still bound but y is a free variable.

6.4 : Recursion and Iteration

Q.15 What is recursive function ? Explain how to use recursive function in LISP with suitable example.

Ans. :

- **Recursive function** is a function which is called by itself for more than once.
- A function is recursive if it calls itself -
  - Boundary condition: not recursive and
  - Recursive condition: must be a smaller problem to converge.
- For example -

```
(defun power (x y)
      (if (= y 0) 1
         (* x (power x (- y 1)))))

>power (3 4)
81
```

- In LISP, the **iterative functions** can be written using do, dotimes, dolists. The syntax of dotimes is

```
(dotimes (<counter> <limit> <result>) <body>)
```

For example

```
(defun power_It (x y)
     (let ((result 1))
```

```
        (dotimes (count y result)
          (setf result (* x result)))))

>power_It (3 4)
81
```

6.5 : Properties A-List, Array and Access Functions

Q.16 Illustrate the use of setf and get in LISP.

Ans. : • Properties are created by using **get** within a **setf** form.

- For example -

```
> (setf (get 'student 'name) '(Ankita))
(ANKITA)
> (setf (get 'student 'rollNo) '(101))
(101)
> (setf (get 'student 'marks) '(98))
(98)
> (write (get 'student 'name))
(ANKITA)(ANKITA)
> (write (get 'student 'rollNo))
(101)(101)
```

Q.17 What is the use of association lists in LISP ? Explain.

Ans. : • An **association list** or **a-list** for short, records a mapping from keys to values. Using **assoc** we can obtain the most recent binding of key to value. For example - following assoc returns the appropriate values for the specified keys.

```
>(assoc 'one '((one 1) (two 2) (one 3)))
(ONE 1)
Another example
(assoc 'age '((name aa) (age 30) (salary 10000)))
(AGE 30)
```

6.6 : Using Lambda Definitions

Q.18 Write a short note on - Lambda definitions in LISP.

Ans. : The lambda function is used to define a function which returns some value. The syntax of this function is

(lambda (*parameters*) *Body*)

For example -

```
>((lambda (x y) (* x y)) 10 20)
200
```

Similarly we can assign a value to a function by the return value of some function.

```
>(define (add n) (lambda (x) (+ x y)))
add
>((add 10) 20)
30
```

Here we have defined the function **add** which returns an object of type function.In order to use the special operator function and apply it to a ***lambda expression.*** It may be abbreviated as #'form. In our example above we used a shorthand notation provided by the macro LAMBDA. We will write it as

```
> (defun add (y)
  #'(lambda (x) (+ x y)))
ADD
```

Now if we try the command

```
>(add 10)
```

It will return the Lambda definition.

6.7 : Printing, Reading and Atom Manipulation

Q.19 How will you perform printing and reading operations in LISP ? Explain.

Ans. : • The print, write are the output functions which are used to display the values or text on the console. Whereas **read** function is for reading the values from the input streams. Following program shows the use of **printing and reading** functions.

- **Example**

```
defun MyFun()
        (print "Enter some number:")
        (setq a(read))
        (print "You have entered...")
        (write a))
```

When above code is executed then we get following output

```
Enter some number: 111
You have entered...111
```

- **Setq Construct :** We can directly set the values to the symbol using **setq** function. For example – following statement assigns value 123 to variable x

```
>(setq x 123)
```

- **Let Construct :** The syntax of Let construct is as follows -

```
(let ((var1 val1) (var2 val2).. (varn valn)), last_val )
```

Where var1, var2, ..varn are variable names and val1, val2, .. valn are the initial values assigned to the respective variables.

When let is executed, each variable is assigned the respective value and lastly the s-expression is evaluated. The value of the last expression evaluated is returned.

```
(let (x = 2, y = 3, z = x + y) print(x + y + z))
```

This will return 10

Part II : Logic Programming Paradigm

6.8 : An Overview of Prolog

Q.20 Describe logical programming. Enlist its features. Also list the commonly used logical programming languages.

☞ [SPPU : June-22, Marks 6]

Ans. : • Logic programming is a programming paradigm in which the set of sentences are written in **logical form.** The logic programs consists of **facts and rules** about some problem domain.

- Logic programming can be viewed as **controlled deduction.**
- An important concept in logic programming is the separation of programs into their logic component and their control component.
- Kowalski illustrates the logic programming by following equation

Algorithm = Logic + Control

where "Logic" represents a logic program in the form of facts and rules. The "Control" represents how the algorithm can be implemented by applying the rules in particular order.

- The concept of logic programming is linked up with a language called prolog. Prolog is acronym of **PRO**gramming in **LOG**ic.

Features of Prolog

1) Prolog is used in **artificial intelligence and expert system** to solve complex problems.
2) It is used on pattern matching algorithm.
3) It is used to develop systems based on **object oriented concepts.**
4) Prolog is used to build **decision support systems** that aid organizations in decision-making. For example - decision systems for medical diagnoses.

Commonly used logical programming languages

- The commonly used logical programming languages are - PROLOG, Answer Set Programming(ASP) and Datalog.

Q.21 Comparison between functional programming and logic programming.

Ans. : [SPPU : Dec.-22, Marks 5]

Sr. No.	Functional Programming	Logical Programming
1.	Functional programming uses functions.	Logical programming uses predicates.
2.	The functions can return a value.	Predicate cannot return a value. The predicate can be true or false.

3.	In functional programming, the variable denotes the concrete value.	In logic programming, a variable does not need to refer to concrete value.
4.	Functional programming is normally used for modelling reasoning.	Logical programming is normally used for modelling knowledge.
5.	Examples – ML, HASKELL are functional programming languages.	Example – PROLOG is a logical programming language.

6.9 : Syntax and Meaning of Prolog Programs

Q.22 Explain the phrases - "Term", "Facts" "Rule", "Goals" used in Prolog with example. [SPPU : Dec.-22, Marks 6]

Ans. : Terms, Goals, Facts and Rules

- A Prolog program consists of a number of clauses. Each clause is either a fact or a rule. After a Prolog program is loaded (or consulted) in a Prolog interpreter, users can submit goals or queries, and the Prolog intepreter will give results (answers) according to the facts and rules.

(1) Terms : • Prolog term is a constant, a variable, or a structure. A constant is either an atom or an integer. Atoms are the symbolic values of Prolog.

- A variable is any string of letters, digits, and underscores that begins with an uppercase letter or an underscore (_). Variables are not bound to types by declarations. The binding of a value, and thus a type, to a variable is called an instantiation.

(2) Goals : • A goal is a statement starting with a predicate and probably followed by its arguments. In a valid goal, the predicate must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the goal must be the

same as that appears in the consulted program. Also, all the arguments (if any) are constants.

- The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database (i.e. the facts and rules in the consulted program). This is similar to proving a hypothesis - the goal being the hypothesis, the facts being the axioms and the rules being the theorems.

(3) Fact : • A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be constants, numbers, variables or lists. Arguments are separated by commas.

- The syntax of fact is

```
pred(arg1, arg2, ... argN).
```

Where

pred is the name of the predicate and arg1, ...argN are the arguments

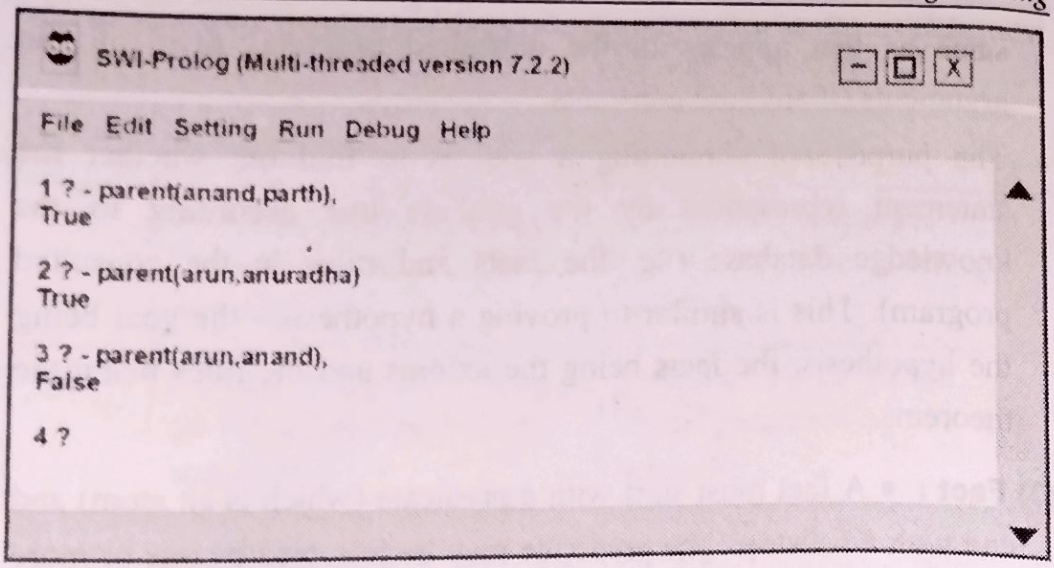
For example -

```
man(anand).
man(arun).

woman(anuradha).
woman(jayashree).

parent(anand,parth). % means anand is parent of parth
Parent(anuradha,parth).
parent(arun,anuradha).
parent(jayashree,anuradha).
```

- The result as either true or false depending upon the facts. It is as follows -



(4) Rule : • A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true. It consists of two parts. A rule is no more than a stored query. Its syntax is

```
head :- body.
```

where

head is a predicate definition (just like a fact),

:- is the neck symbol, sometimes read as "if"

body is one or more goals (a query)

For example -

```
father(F,C):-man(F),parent(F,C).
```

Example using Fact and Rule

```
/* Facts */

man(anand).
man(arun).
woman(anuradha).
woman(jayashree).

parent(anand,parth). % means anand is parent of parth
parent(anuradha,parth).
parent(arun,anuradha).
parent(jayashree,anuradha).
```

```
/* A general rule */
father(F,C):-man(F),parent(F,C).
mother(M,C):-woman(M),parent(M,C).
If we query
-? father(X,parth).
```

The above query can be read who is father of parth? The answer is X=anand is a father of parth.

Q.23 Explain the goal ordering and rule ordering.

Ans. : • Control represents how a language **computes a response to a query.** The control execution is based on two rules -

1. **Goal Order :** That means choose the **leftmost subgoal**
2. **Rule Order :** That means select the **first applicable rule.**

- A **goal** is a statement starting with a **predicate** and probably followed by its **arguments.** In a valid goal, the predicate must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the goal must be the same as that appears in the consulted program. Also, all the arguments (if any) are constants.
- The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database (i.e. the facts and rules in the consulted program). Let us understand how control in prolog works with the help of simple example. Following is a **family.pl** file in which the rules and facts are written.

```
/*Rules*/
gfather(X,Y):- father(X,Z),parent(Z,Y).
parent(X,Y):- father(X,Y).

/*Facts*/

father(dashrath,ram). %means dashrath is father of ram
father(raghu,aja).
```

```
father(aja,dashrath).
father(ram,luv).
father(ram,kush).
```

- The tree for the above facts can be as given below -

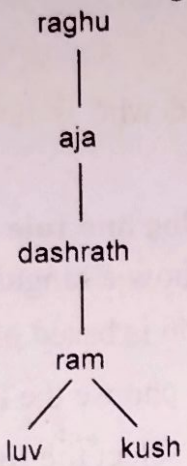


Fig. Q.23.1

The goals can be executed on the prompt window as follows -

```
1 ?- gfather(aja,ram).
true.

2 ?- parent(X,Y).
X = dashrath,
Y = ram .

3 ?- father(X,Y), father(Y,ram).
X = aja,
Y = dashrath
```

Now consider the goal **gfather(aja,ram).**

As we know there are two rules for execution of control -

1. Goal Order : That means choose the leftmost subgoal

2. Rule Order : That means select the first applicable rule.

Here the rules are

```
gfather(X,Y):- father(X,Z),parent(Z,Y).
parent(X,Y):- father(X,Y).
```

Step 1 : For gfather(aja,ram) we apply the first applicable rule(**Rule ordering**)

gfather(X,Y):- father(X,Z),parent(Z,Y)

- Here the leftmost subgoal i.e. father (X,Z) is chosen first.(**Goal ordering**)

Step 2 : The match for father(X,Z) is searched in the fact list with X=aja. We will find the match with father(aja,dashrath). Hence value of **Z=dashrath.**

- Hence the first rule now becomes

gfather(aja,ram) = father(aja,Z),parent(Z,Y). With Z=dashrath and Y=ram

Step 3 : For the subgoal parent(Z,Y) select the rule

parent(X,Y):- father(X,Y).

- With X=Z=dashrath and Y=ram. The parent will return father(X,Y)=father(dashrath,ram) which comes out to be true. Hence we get the answer as **true.**
- While executing the control the prolog choose a rule by,

i) searching sequentially in the program from top to bottom whose head matches with the goal with possible unifier.

ii) Remember the position of the matched rule.

iii) Join the rule body in front of the sequence of sub goals to be solved.

iv) Repeat until either goal is satisfied or is failed.

Q.24 Explain the basic elements of prolog.

Ans. : Refer Q.22.

6.10 : Lists, Operators and Arithmetic

Q.25 How to create a list in prolog ? Give example.

Ans. :

Lists

- A list is an ordered sequence of objects and lists. In Prolog, a list is written as its elements separated by commas and enclosed in brackets. For example :

[ ] ← represents empty list

[a,b,c,[d]].

- In Prolog list elements are enclosed by brackets and separated by commas.

[1,2,3,4]

Q.26 Write short note on - Use of operators in prolog for predicate calculus.

Ans. :

Operator

- Predicate calculus is a fundamental notation for representing and reasoning with logical statements. It extends propositional calculus by introducing the quantifiers, and by allowing predicates and functions of any number of variables.

1) **Propositional symbols :** P, Q, R, S, T, ... denote propositions, or statements about the world that maybe either true or false, such as "Sun is bright" or "India is a country".
2) **Truth symbols :** true, false.
3) **Connectives :** $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\Rightarrow$, =
4) **Propositional calculus sentences :** Every propositional symbol and truth symbol is a sentence.

   For example The negation of a sentence is a sentence. Denoted as $\neg P$

   Any two sentences connected by one of $\{\wedge, \vee, \Rightarrow, =\}$ are a sentence. For instance $P \vee \neg Q \Rightarrow R$.
5) **Conjucts and disjuncts :** In expressions of the form $P \wedge Q$, P and Q are called conjuncts.

   In $P \vee Q$, P and Q are called disjuncts.
6) **Symbols :** The symbols ( ) and [ ] are used to group symbols into sub-expressions and so control their order of evaluation and meaning. $(P \vee \neg Q) = R$ and

   $P \vee (\neg Q = R)$ are different.

6.11 : List Structures

Q.27 Explain the list manipulation in prolog.

[SPPU : Dec.-22, Marks 6]

OR Explain the concept of list structures in prolog with example.

[SPPU : June.-22, Marks 5]

Ans. : • A list is an ordered sequence of objects and lists. In prolog, a list is written as its elements separated by commas and enclosed in brackets. For example :

```
[ ] ← represents empty list
[a,b,c,[d]].
```

- The list contains Head and Tail part. For example - if we type following query

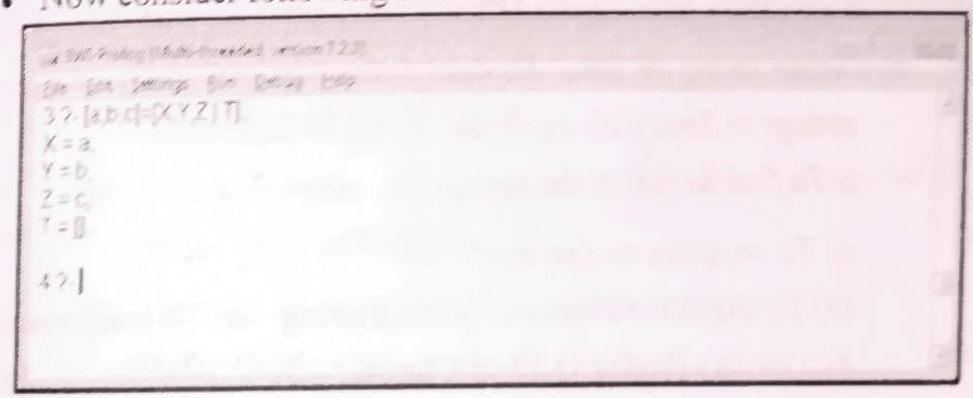
```
?- [H|T] = [10,20,30,40].
```

- We will get

H = 10,

T = [20, 30, 40].

- Now consider following execution of the list



- Now if we execute

```
?- [a,b,c] = [a|[b|[c|[]]]].
```

Then we will get -

true

END...

June-2022 [END SEM] [5869] - 280	Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] [Maximum Marks : 70

Instructions to the candidates :

1) ***Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***
2) ***Neat diagrams must be drawn wherever necessary.***
3) ***Assume suitable data, if necessary.***

Q.1 a) *Describe primitive data types. List the primitive data types in Java and their respective storage capacity* **(Refer Q.6 of Chapter - 3)** [6]

b) *Write short notes on Java Virtual Machine(JVM) with diagram.* **(Refer Q.2 of Chapter - 3)** [6]

c) *State the uses of the final keyword in Java ?* **(Refer Q.2 of Chapter - 3)** [6]

OR

Q.2 a) *Define string in Java. Explain following operations of class strings in Java with example.*

i) To find length of the string

ii) To compare two strings

iii) To extract a character from a string iv) To concatenate two strings **(Refer Q.13 of Chapter - 3)** [6]

b) *Explain Java's role in internet. Justify the following features of Java.*

i) Secure ii) Architectural neutral iii) Distributed. **(Refer Q.4 of Chapter - 3)** [6]

c) *Summarize different access controls in Java. Explain the situation if you remove static modifier from the main method.* **(Refer Q.31 of Chapter - 3)** [6]

Q.3 a) *State the difference between character and byte stream in Java. Give any two input and any two output classes for character streams.* **(Refer Q.55 of Chapter - 4)** [6]

b) *Describe exception. Explain keywords try, catch, throw, throws and finally related to exception handling.* **(Refer Q.37 of Chapter - 4)** [6]

c) *Define package and interfaces in Java ? Explain it with suitable example.* **(Refer Q.29 of Chapter - 4)** [5]

OR

Q.4 a) *Define is inheritance. List the advantages of inheritance. Explain simple inheritance in java with example.* **(Refer Q.1 of Chapter - 4)** [6]

b) *Elaborate the significance of key word "Super" in Java. Demonstrate with example for super keyword in Java constructor.* **(Refer Q.4 of Chapter - 4)** [6]

c) *State the importance of finally blocks. Illustrate the ways finally block differ from finalize() method.* **(Refer Q.23 of Chapter - 3 and Q.37 of Chapter - 4)** [5]

Q.5 a) *Interpret the terms multitasking and multiprocessing and multithreading in Java with example.* **(Refer Q.2 of Chapter - 5)** [6]

b) *List the features, advantages and limitations of angular JS.* **(Refer Q.14 of Chapter - 5)** [6]

c) *Write the JavaScript code to create login page form.* **(Refer Q.13 of Chapter - 5)** [6]

OR

Q.6 a) *Compare React JS and Angular JS and Vue JS.* **(Refer Q.18 of Chapter - 5)** [6]

b) *Elaborate the terms getPriority() and setPriority() methods with example.* **(Refer Q.6 of Chapter - 5)** [6]

c) *Explain the uses of isAlive() and Join() methods in Java thread with example.* **(Refer Q.9 of Chapter - 5)** [6]

Q.7 a) *Describe functional programming. Enlist its features. Also list the commonly used functional programming languages.* **(Refer Q.1 of Chapter - 6)** [6]

b) *Write sequences of CAR's and CDR's that will pick the atom pear out of the following s-expression :*
i) (apple orange pear grapes) ii) ((apple orange) (pear grapes))
iii) (((apple)(orange) (pear) (grapes)))
(Refer Q.3 of Chapter - 6) [6]

c) *Explain the concept of "structures" in prolog with example.* **(Refer Q.27 of Chapter - 6)** [5]

OR

Q.8 a) *Describe logical programming. Enlist its features. Also list the commonly used logical programming languages.* **(Refer Q.20 of Chapter - 6)** [6]

b) *Write a LISP program to find the factorial of n numbers using recursion concept.* **(Refer Q.11 of Chapter - 6)** [6]

c) *Explain the following number predicates using suitable example.*
i) NUMBERP ii) ZEROP iii) PLUSP iv) EVENP
v) ODDP **(Refer Q.10 of Chapter - 6)** [5]

Dec.- 2022 [END SEM] [5925] - 264	Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] [Maximum Marks : 70

Q.1 a) *Justify the meaning of each characteristic of Java in the statement "Java is simple, architecture neutral portable, interpreted and robust and secured programming language".* **(Refer Q.1 of Chapter - 3)** [6]

b) *Write a program in Java to perform the addition of two matrices (multidimensional arrays) and set the diagonal elements of resultant matrix to 0.*
(Refer Q.11 of Chapter - 3) [6]

c) *Write a program in Java using switch-case statement to perform addition, subtraction, multiplication and division of given two numbers and print the result.*
(Refer Q.7 of Chapter - 3) [6]

OR

Q.2 a) *Explain in brief below keywords with example in Java.*
i) Final **(Refer Q.5 of Chapter - 4)**
ii) Static **(Refer Q.29 of Chapter - 3)**
iii) This **(Refer Q.20 of Chapter - 3)** [6]

b) *Define Constructor. List its different types. Demonstrate with suitable example the different types of constructors used in Java.* **(Refer Q.18 of Chapter - 3)** [6]

c) *Write a program which receives n integers. Store the integers in an array. Program outputs the number of odd and even numbers present in this array.*
(Refer Q.10 of Chapter - 3) [6]

Q.3 a) *Define inheritance and list the advantages. Explain the types of inheritance in Java with suitable Java code.*
(Refer Q.1 of Chapter - 4) [9]

b) *Describe exception. Write any two examples of exception. Explain keywords try, catch, throw, throws and finally related to exception handling.* **(Refer Q.37 of Chapter - 4)** [8]

OR

Q.4 a) *Demonstrate the methods of creating packages and importing them in other java programs with example. Write about the access protection in packages.*
(Refer Q.25 of Chapter - 4) [9]

b) *Explain abstract classes and polymorphism in Java with appropriate java codes.* **(Refer Q.17 of Chapter - 4)** [8]

Q.5 a) *Write short notes on React JS. Enlist and explain the feature of it.* **(Refer Q.16 of Chapter - 5)** [6]

b) *Explain life cycle of Thread model in Java.*

(Refer Q.4 of Chapter - 5) [6]

c) *State the term thread synchronization. Explain how to achieve thread synchronization in Java.*

(Refer Q.8 of Chapter - 5) [6]

OR

Q.6 a) *List the features of JavaScript and write a JavaScript program to display Welcome message.*

(Refer Q.11 of Chapter - 5) [6]

b) *Write short note Angular JS. List its advantages and disadvantages.* **(Refer Q.15 of Chapter - 5)** [6]

c) *Differentiate multiprocessing and multi-threading.*

(Refer Q.3 of Chapter - 5) [6]

Q.7 a) *Write a program to find the factorial of a given number using LISP.* **(Refer Q.11 of Chapter - 5)** [6]

b) *Evaluate the following forms of LISP.* [5]

i) (car (cdr '(1 2 3 4 5))) ii) (car (cdr '(a(b c)d e)))

iii) (car (cdr(cdr '(1 2 3 4 5 6 7 8))))

(Refer Q.4 of Chapter - 6)

c) *Explain the basic list manipulation in prolog.*

(Refer Q.27 of Chapter - 6) [6]

OR

Q.8 a) *Explain the following Equality predicates using suitable example. i) EQUAL ii) EQ iii) EQL iv) =*

(Refer Q.9 of Chapter - 6) [6]

b) *Comparisons between functional programming and logic programming.* **(Refer Q.20 of Chapter - 6)** [5]

c) *Explain the phrases - "Term", "Facts", "Rule", Goals" used in Prolog with example.*

(Refer Q.22 of Chapter - 6) [6]

END ...