1. Introduction to Java

History of Java:
- In 1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Project but later its name was changed to Green Project.
- In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project.
- Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.
- James Gosling with his team started developing a new language, which was completely system independent. This language was initially called **OAK**. Since this name was registered by some other company, later it was changed to **Java**.
- James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island'. Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.
- Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released.

Features of Java (Java buzz words):
- **Simple:** Learning and practicing java is easy because of resemblance with c and C++.
- **Object Oriented Programming Language:** Unlike C++, Java is purely OOP.
- **Distributed:** Java is designed for use on network; it has an extensive library which works in agreement with TCP/IP.
- **Secure:** Java is designed for use on Internet. Java enables the construction of virus-free, tamper free systems.
- **Robust (Strong/ Powerful):** Java programs will not crash because of its exception handling and its memory management features.
- **Interpreted:** Java programs are compiled to generate the byte code. This byte code can be downloaded and interpreted by the interpreter. .class file will have byte code instructions and JVM which contains an interpreter will execute the byte code.
- **Portable:** Java does not have implementation dependent aspects and it yields or gives same result on any machine.
- **Architectural Neutral Language**: Java byte code is not machine dependent, it can run on any machine with any processor and with any OS.
- **High Performance:** Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.
- **Multithreaded:** Executing different parts of program simultaneously is called multithreading. This is an essential feature to design server side programs.
- **Dynamic:** We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

Obtaining the Java Environment:
· We can download the JDK (Java Development Kit) including the compiler and runtime engine from Sun at: http://java.sun.com/javase.
· Install JDK after downloading, by default JDK will be installed in
  C:\Program Files\Java\jdk1.5.0_05   (Here jdk1.5.0_05 is JDK's version)

Setting up Java Environment: After installing the JDK, we need to set at least one environment variable in order to able to compile and run Java programs. A PATH environment variable enables the operating system to find the JDK executables when our working directory is not the JDK's binary directory.
· **Setting environment variables from a command prompt:** If we set the variables from a command prompt, they will only hold for that session. To set the PATH from a command prompt:

        set PATH=C:\Program Files\Java\jdk1.5.0_05\bin;%PATH%

· **Setting environment variables as system variables:** If we set the variables as system variables they will hold continuously.

  o Right-click on *My Computer*
  o Choose *Properties*
  o Select the *Advanced* tab
  o Click the *Environment Variables* button at the bottom
  o In system variables tab, select path (system variable) and click on edit button
  o A window with variable name- path and its value will be displayed.
  o Don't disturb the default path value that is appearing and just append (add) to that path at the end:
    ;C:\ProgramFiles\Java\
           jdk1.5.0_05\bin;
  o Finally press OK button.

2
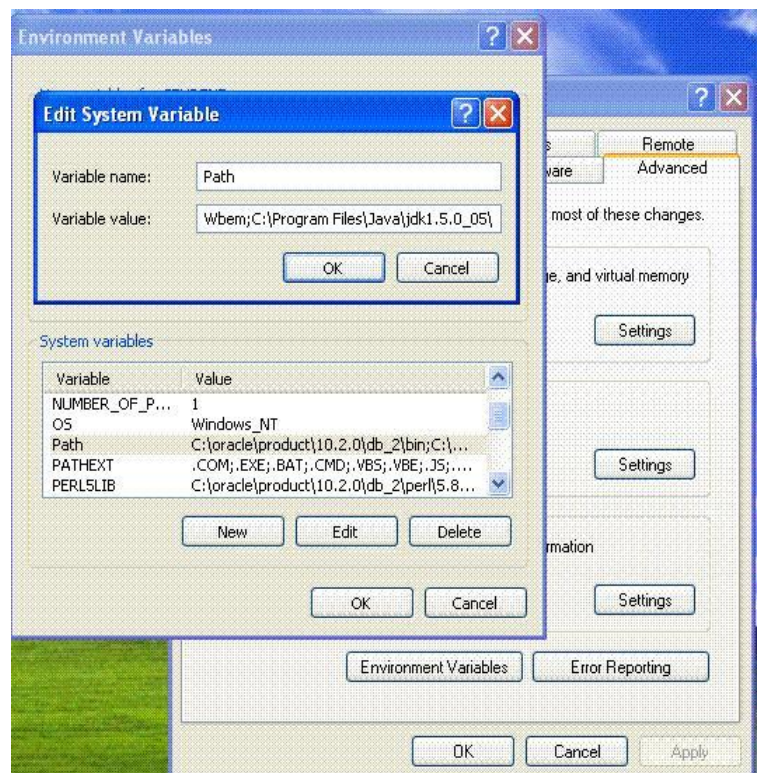
2. Programming Structure

Comments: Comments are description about the aim and features of the program. Comments increase readability of a program. Three types of comments are there in Java:

- **Single line comments:** These comments start with // **e.g.:** // this is comment line
- **Multi line comments:** These comments start with /* and end with */ **e.g.:** /* this is comment line*/
- **Java documentation comments:** These comments start with /** and end with */ These comments are useful to create a HTML file called API (application programming Interface) document. This file contains description of all the features of software.

Structure of the Java Program:

- As all other programming languages, Java also has a structure.
- The first line of the C/C++ program contains include statement. For example, <stdio.h> is the header file that contains functions, like printf (), scanf () etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.
- Similarly in Java first we need to import the required packages. By default java.lang.* is imported. Java has several such packages in its library. A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object. So, it is mandatory to write a class in Java program. We should use class keyword for this purpose and then write class name.
- In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.

```
Sample Program:
//A Simple Java Program
import java.lang.System;
import java.lang.String;
class Sample
{
        public static void main(String args[])
                {
                        System.out.print ("Hello world");
                }
}
```

- Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class. But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- Static methods are the methods, which can be called and executed without creating objects. Since we want to call main () method without using an object, we should declare main ()

method as static. JVM calls main () method using its Classname.main () at the time of running the program.

· JVM is a program written by Java Soft people (Java development team) and main () is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don't declare main () method as public, then it doesn't make itself available to JVM and JVM cannot execute it.

· JVM always looks for main () method with String type array as parameter otherwise JVM cannot recognize the main () method, so we must provide String type array as parameter to main () method.

· A class code starts with a {and ends with a}. A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class. This is our first program, so we had written only one method called main ().

· Our aim of writing this program is just to display a string "Hello world". In Java, print () method is used to display something on the monitor.

· A method should be called by using objectname.methodname (). So, to call print () method, create an object to PrintStream class then call objectname.print () method.

· An alternative is given to create an object to PrintStream Class i.e. System.out. Here, System is the class name and out is a static variable in System class. out is called a field in System class. When we call this field a PrintStream class object will be created internally. So, we can call print() method as: System.out.print ("Hello world");

· println () is also a method belonging to PrintStream class. It throws the cursor to the next line after displaying the result.

· In the above Sample program System and String are the classes present in java.lang package.

Escape Sequence: Java supports all escape sequence which is supported by C/ C++. A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler. When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Creating a Source File:
· Type the program in a text editor (i.e. Notepad, WordPad, Microsoft Word or Edit Plus). We can launch the Notepad editor from the **Start** menu by selecting **Programs > Accessories > Notepad**. In a new document, type the above code (i.e. Sample Program).

· Save the program with filename same as Class_name (i.e. Sample.java) in which main method is written. To do this in Notepad, first choose the **File > Save** menu item. Then, in the **Save** dialog box:

4

o Using the **Save in** combo box, specify the folder (directory) where you'll save your file. In this example, the directory is JQR on the D drive.
o In the **File name** text field, type "Sample.java", including the quotation marks. Then the dialog box should look like this:
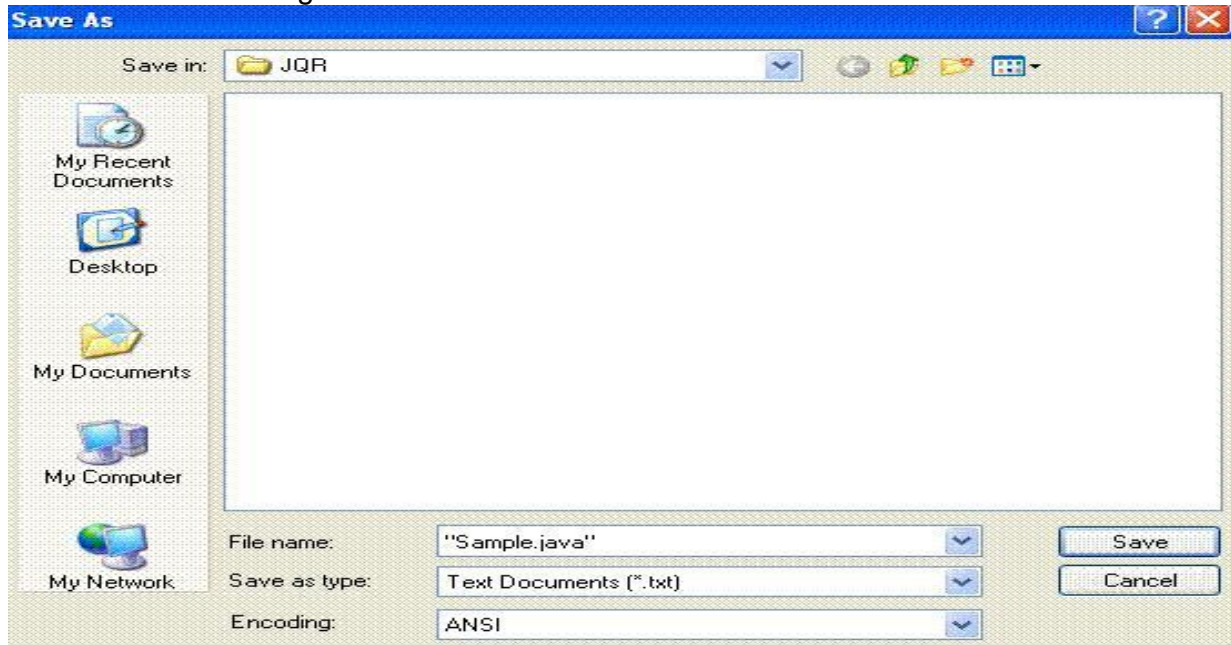
o Now click **Save**, and exit Notepad.

Compiling the Source File into a .class File:

· To Compile the Sample.java program go to DOS prompt. We can do this from the **Start** menu by choosing **Run...** and then entering cmd. The window should look similar to the following figure.

· The prompt shows current directory. To compile Sample.java source file, change current directory to the directory where Sample.java file is located. For example, if source directory is JQR on the D drive, type the following commands at the prompt and press Enter:

Now the prompt should change to D:\JQR>

· At the prompt, type the following command and press Enter.
> javac Sample.java

```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac Sample.java

D:\JQR>
```

· The compiler generates byte code and Sample.class will be created.

Executing the Program (Sample.class):
· To run the program, enter java followed by the class name created at the time of compilation at the command prompt in the same directory as:
               java Sample

```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>java Sample
Hello world
D:\JQR>
```

· The program interpreted and the output is displayed.

The Java Virtual Machine: Java Virtual Machine (JVM) is the heart of entire Java program execution process. First of all, the .java program is converted into a .class file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This .class file is given to the JVM. Following figure shows the architecture of Java Virtual Machine.
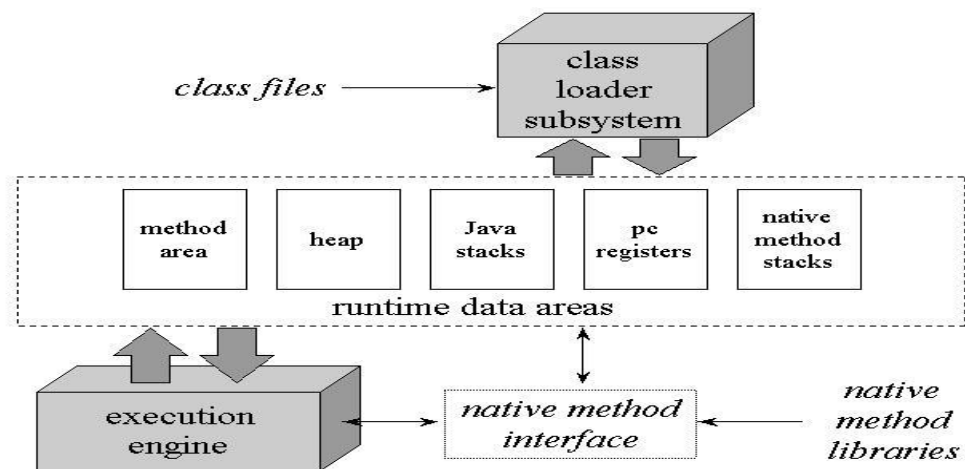

Figure: The internal architecture of the Java virtual machine.

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:
· First of all, it loads the .class file into memory.
· Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.

6

- If the byte instructions are proper, then it allocates necessary memory to execute the program. This memory is divided into 5 parts, called run time data areas, which contain the data and results while running the program. These areas are as follows:
  - o **Method area:** Method area is the memory block, which stores the class code, code of the variables and code of the methods in the Java program. (Method means functions written in a class).
  - o **Heap:** This is the area where objects are created. Whenever JVM loads a class, method and heap areas are immediately created in it.
  - o **Java Stacks:** Method code is stored on Method area. But while running a method, it needs some more memory to store the data and results. This memory is allotted on Java Stacks. So, Java Stacks are memory area where Java methods are executed. While executing methods, a separate frame will be created in the Java Stack, where the method is executed. JVM uses a separate thread (or process) to execute each method.
  - o **PC (Program Counter) registers:** These are the registers (memory areas), which contain memory address of the instructions of the methods. If there are 3 methods, 3 PC registers will be used to track the instruction of the methods.
  - o **Native Method Stacks:** Java methods are executed on Java Stacks. Similarly, native methods (for example C/C++ functions) are executed on Native method stacks. To execute the native methods, generally native method libraries (for example C/C++ header files) are required. These header files are located and connected to JVM by a program, called Native method interface.

Execution Engine contains interpreter and JIT compiler which translates the byte code instructions into machine language which are executed by the microprocessor. Hot spot (loops/iterations) is the area in .class file i.e. executed by JIT compiler. JVM will identify the Hot spots in the .class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter.

3. Naming Conventions, Data Types and Operators

Naming Conventions: Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods etc.

· Package names are written in small letters.
   e.g.: java.io, java.lang, java.awt etc
· Each word of class name and interface name starts with a
   capital **e.g.**: Sample, AddTwoNumbers
· Method names start with small letters then each word start with a
   capital **e.g.**: sum (), sumTwoNumbers (), minValue ()
· Variable names also follow the same above method rule
   e.g.: sum, count, totalCount
· Constants should be written using all capital
   letters **e.g.**: PI, COUNT
· Keywords are reserved words and are written in small
   letters. **e.g.**: int, short, float, public, void

Data Types: The classification of data item is called data type. Java defines eight simple types of data. byte, short, int, long, char, float, double and boolean. These can be put in four groups:

· **Integer Data Types:** These data types store integer numbers

Data Type	Memory size	Range
Byte	1 byte	-128 to 127
Short	2 bytes	-32768to 32767
Int	4 bytes	-2147483648to 2147483647
Long	8 bytes	-9223372036854775808 to 9223372036854775807

   e.g.:   byte rno = 10;
            long x = 150L; L means forcing JVM to allot 8 bytes

· **Float Data Types:** These data types handle floating point numbers

Data Type	Memory size	Range
Float	4 bytes	-3.4e38 to 3.4e38
Double	8 bytes	-1.7e308 to 1.7e308

   e.g.:   float pi = 3.142f;
           double distance = 1.98e8;

· **Character Data Type:** This data type represents a single character. char data type in java uses two bytes of memory also called Unicode system. Unicode is a specification to include alphabets of all international languages into the character set of java.

Data Type	Memory size	Range
Char	2 bytes	0 to 65535

   e.g.:   char ch = 'x';

· **Boolean Data Type:**   can handle truth values either true or false
      e.g.:-   boolean response = true;

Operators: An operator is a symbol that performs an operation. An operator acts on variables called operands.

· **Arithmetic operators:** These operators are used to perform fundamental operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example	Result
+	Addition	3 + 4	7
-	Subtraction	5 - 7	-2
*	Multiplication	5 * 5	25
/	Division (gives quotient)	14 / 7	2
%	Modulus (gives remainder)	20 % 7	6

· **Assignment operator:** This operator (=) is used to store some value into a variable.

Simple Assignment	Compound Assignment
x = x + y	x += y
x = x − y	x -= y
x = x * y	x *= y
x = x y	x /= y

· **Unary operators:** As the name indicates unary operator's act only on one operand.

Operator	Meaning	Example	Explanation
-	Unary minus	j = -k;	k value is negated and stored into j
++	Increment Operator	b++; ++b;	b value will be incremented by 1 (called as post incrementation) b value will be incremented by 1 (called as pre incrementation)
--	Decrement Operator	b--; --b;	b value will be decremented by 1 (called as post decrementation) b value will be decremented by 1 (called as pre decrementation)

· **Relational operators:** These operators are used for comparison purpose.

Operator	Meaning	Example
==	Equal	x == 3
!=	Not equal	x != 3
<	Less than	x < 3
>	Greater than	x > 3
<=	Less than or equal to	x <= 3

· **Logical operators:** Logical operators are used to construct compound conditions. A compound condition is a combination of several simple conditions.

Operator	Meaning	Example	Explanation
&&	and operator	if(a>b && a>c) System.out.print("yes");	If a value is greater than b and c then only yes is displayed
||	or operator	if(a==1 || b==1) System.out.print("yes");	If either a value is 1 or b value is 1 then yes is displayed
!	not operator	if(!(a==0)) System.out.print("yes");	If a value is not equal to zero then only yes is displayed

· **Bitwise operators:** These operators act on individual bits (0 and 1) of the operands. They act only on integer data types, i.e. byte, short, long and int.

Operator	Meaning	Explanation
&	Bitwise AND	Multiplies the individual bits of operands
|	Bitwise OR	Adds the individual bits of operands
^	Bitwise XOR	Performs Exclusive OR operation
<<	Left shift	Shifts the bits of the number towards left a specified number of positions
>>	Right shift	Shifts the bits of the number towards right a specified number of positions and also preserves the sign bit.
>>>	Zero fill right shift	Shifts the bits of the number towards right a specified number of positions and it stores 0 (Zero) in the sign bit.
~	Bitwise complement	Gives the complement form of a given number by changing 0's as 1's and vice versa.

· **Ternary Operator or Conditional Operator (? :):** This operator is called ternary because it acts on 3 variables. The syntax for this operator is:

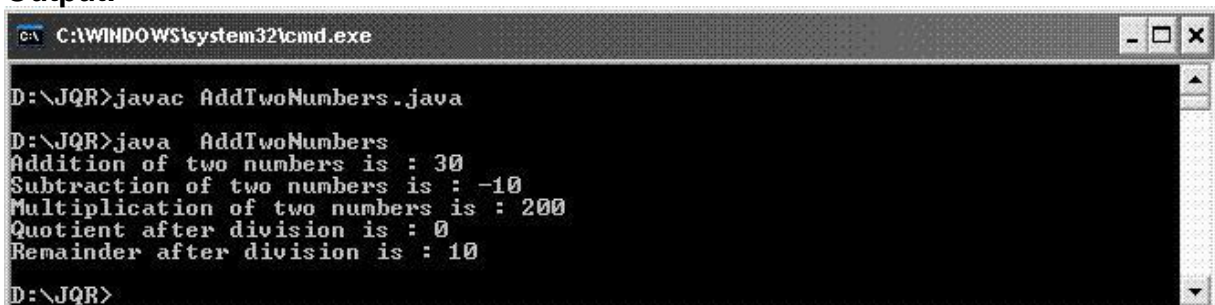Variable = Expression1? Expression2: Expression3;

First Expression1 is evaluated. If it is true, then Expression2 value is stored into variable otherwise Expression3 value is stored into the variable.

e.g.: max = (a>b) ? a: b;

Program 1: Write a program to perform arithmetic operations //Addition of two numbers

```
class AddTwoNumbers
{          public static void mian(String args[])
          { int i=10, j=20;
                System.out.println("Addition of two numbers is : " + (i+j));
                System.out.println("Subtraction of two numbers is : " + (i-j));
                System.out.println("Multiplication of two numbers is : " + (i*j));
                System.out.println("Quotient after division is : " + (i/j) );
                System.out.println("Remainder after division is : " +(i%j) );
          }
}
```

Output:

Program 2: Write a program to perform Bitwise
operations //Bitwise Operations
class Bits
{ public static void main(String args[]) {
        byte x,y;
                x=10;
                y=11;
                System.out.println ("~x="+(~x));
                System.out.println ("x & y="+(x&y));
                System.out.println ("x | y="+(x|y));
                System.out.println ("x ^ y="+(x^y));
                System.out.println ("x<<2="+(x<<2));
                System.out.println ("x>>2="+(x>>2));
                System.out.println ("x>>>2="+(x>>>2));
        }
}
Output:

11

4. Control Statements

Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution. In Java control statements are categorized into selection control statements, iteration control statements and jump control statements.

· **Java's Selection Statements:** Java supports two selection statements: if and switch. These statements allow us to control the flow of program execution based on condition.

  o **if Statement:** if statement performs a task depending on whether a condition is true or false.

  Syntax: if (condition)
                        statement1;
                  else
                        statement2;

  Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block). The condition is any expression that returns a boolean value. The else clause is optional.

  Program 1: Write a program to find biggest of three numbers. //Biggest of three numbers

```
class BiggestNo
{ public static void main(String args[]) {
     int a=5,b=7,c=6;
         if ( a > b && a>c)
             System.out.println    ("a    is
         big"); else if ( b > c)
             System.out.println ("b is big");
         else
             System.out.println ("c is big");
     }
}
```

  Output:


```
D:\JQR>javac BiggestNo.java

D:\JQR>java   BiggestNo
b is big

D:\JQR>_
```

  o **Switch Statement:** When there are several options and we have to choose only one option from the available ones, we can use switch statement.

  Syntax: switch (expression)
                  { case value1: //statement sequence
                                      break;
                      case value2: //statement sequence

12

```
                                        break;
                          ……………..
                    case valueN:  //statement sequence
                                  break;
                    default:      //default statement sequence
             }
```

Here, depending on the value of the expression, a particular corresponding case will be executed.

Program 2: Write a program for using the switch statement to execute a particular task depending on color value.
//To display a color name depending on color
value class ColorDemo
{ public static void main(String args[]) {
        char color = 'r';
             switch (color)
               {   case 'r': System.out.println ("red");   break;
                     case 'g': System.out.println ("green");   break;
                     case 'b': System.out.println ("blue");   break;
                     case 'y': System.out.println ("yellow");   break;
                     case 'w': System.out.println ("white");   break;
                     default: System.out.println ("No Color Selected");
               }
        }
}
```
**Output:**



```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac ColorDemo.java

D:\JQR>java  ColorDemo
red

D:\JQR>
```

· **Java's Iteration Statements:** Java's iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops. A loop repeatedly executes the same set of instructions until a termination condition is met.
   o **while Loop:** while loop repeats a group of statements as long as condition is true. Once the condition is false, the loop is terminated. In while loop, the condition is tested first; if it is true, then only the statements are executed. while loop is called as entry control loop. **Syntax:** while (condition)
```
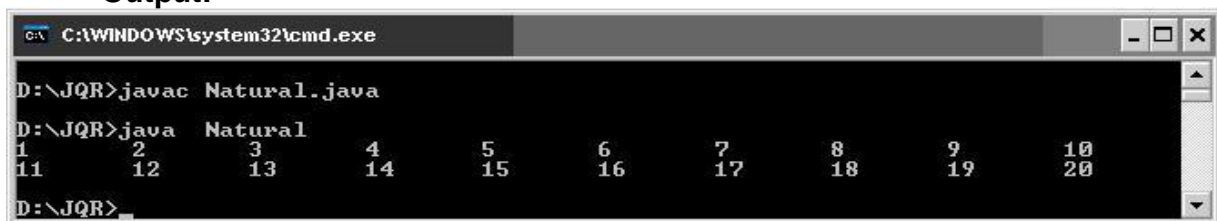                    {
                          statements;
                    }
```

13

Program 3: Write a program to generate numbers from 1
to 20. **//**Program to generate numbers from 1 to 20.

```
class Natural
{ public static void main(String args[]) {
        int i=1;
                while (i <= 20)
                { System.out.print (i + "\t"); i++;

                }
        }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe                                         _ □ ×

D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>
```

- o **do…while Loop:** do…while loop repeats a group of statements as long as
  condition is true. In do...while loop, the statements are executed first and then the
  condition is tested. do…while loop is also called as exit control loop.

  Syntax: do
                 {
                         statements;
                 } while (condition);

Program 4: Write a program to generate numbers from 1
to 20. **//**Program to generate numbers from 1 to 20.

```
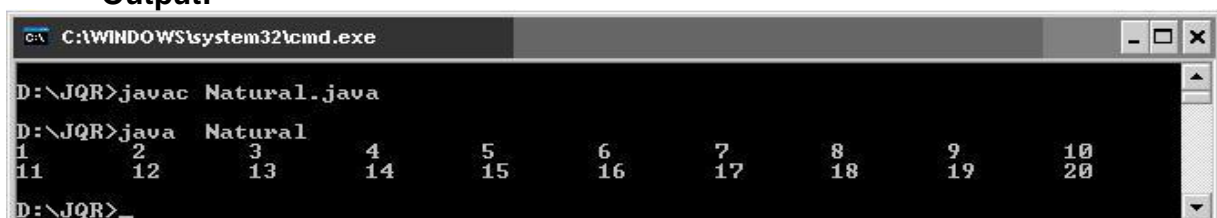class Natural
{ public static void main(String args[]) {
        int i=1;
                do
                { System.out.print (i + "\t"); i++;

                } while (i <= 20);
        }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe                                         _ □ ×

D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>
```

14

- o **for Loop:** The for loop is also same as do…while or while loop, but it is more compact syntactically. The for loop executes a group of statements as long as a condition is true.

  Syntax: for (expression1; expression2; expression3)
  {   statements;
  }

  Here, expression1 is used to initialize the variables, expression2 is used for condition checking and expression3 is used for increment or decrement variable value.

  Program 5: Write a program to generate numbers from 1 to 20. **//**Program to generate numbers from 1 to 20.
  ```
  class Natural
  { public static void main(String args[]) {
          int i;
                  for (i=1; i<=20; i++)
                          System.out.print (i + "\t");
          }
  }
  ```
  Output:

- · **Java's Jump Statements:** Java supports three jump statements: break, continue and return. These statements transfer control to another part of the program.
  - o **break:**
    - ➢ break can be used inside a loop to come out of it.
    - ➢ break can be used inside the switch block to come out of the switch block.
    - ➢ break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.

    Syntax: break; (or)   break label;//here label represents the name of the block.

    Program 6: Write a program to use break as a civilized form of goto. **//**using break as a civilized form of goto
    ```
    class BreakDemo
    { public static void main (String args[]) {
            boolean t = true;
                    first:
                    {
                            second:
                            {
                                    third:
                                    {
    ```
    15

```
                                        System.out.println ("Before the break");
                                        if (t) break second; // break out of second block
                                            System.out.println ("This won't execute");
                            }
                        System.out.println ("This won't execute");
                }
            System.out.println ("This is after second block");
        }
    }
}
```

Output:


```
C:\WINDOWS\system32\cmd.exe                                    - □ ×

D:\JQR>javac BreakDemo.java

D:\JQR>java  BreakDemo
Before the break
This is after second block

D:\JQR>_
```

- o **continue:** This statement is useful to continue the next repetition of a loop/ iteration. When continue is executed, subsequent statements inside the loop are not executed. **Syntax:** continue;

  Program 7: Write a program to generate numbers from 1 to 20.
  //Program to generate numbers from 1
  to 20. class Natural
  { public static void main (String args[]) {
          int i=1;
                  while (true)
                  { System.out.print (i + "\t"); i++;

                      if (i <= 20)
                              continue;
                      else
                              break;
                  }
          }
  }
  ```
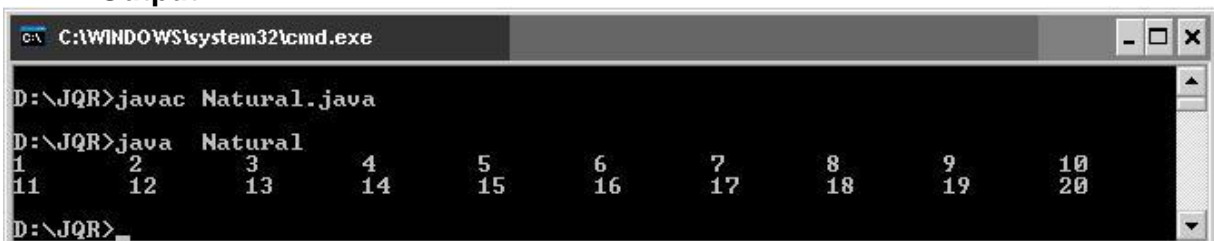
**Output:**



```
C:\WINDOWS\system32\cmd.exe                                    - □ ×

D:\JQR>javac Natural.java

D:\JQR>java  Natural
1        2        3        4        5        6        7        8        9        10
11       12       13       14       15       16       17       18       19       20

D:\JQR>_
```

16

o **return statement:**
  ➤ return statement is useful to terminate a method and come back to the calling method.
  ➤ return statement in main method terminates the application.
  ➤ return statement can be used to return some value from a method to a calling method.

Syntax: return;
                            (or)
                            return value; // value may be of any type

Program 8: Write a program to demonstrate return
statement. //Demonstrate return
class ReturnDemo
{ public static void main(String args[]) {
       boolean t = true;
              System.out.println   ("Before   the
              return"); if (t)
                     return;
              System.out.println ("This won't execute");
       }
}

Output:

Note: goto statement is not available in java, because it leads to confusion and forms infinite loops.

17

5. Accepting Input from Keyboard

A stream represents flow of data from one place to other place. Streams are of two types in java. Input streams which are used to accept or receive data. Output streams are used to display or write data. Streams are represented as classes in java.io package.

· **System.in:** This represents InputStream object, which by default represents standard input device that is keyboard.
· **System.out:** This represents PrintStream object, which by default represents standard output device that is monitor.
· **System.err:** This field also represents PrintStream object, which by default represents monitor. System.out is used to display normal messages and results whereas System.err is used to display error messages.

To accept data from the keyboard:
· Connect the keyboard to an input stream object. Here, we can use InputStreamReader that can read data from the keyboard.

        InputSteamReader obj = new InputStreamReader (System.in);
· Connect InputStreamReader to BufferReader, which is another input type of stream. We are using BufferedReader as it has got methods to read data properly, coming from the stream.

        BufferedReader br = new BufferedReader (obj);

The above two steps can be combined and rewritten in a single statement as:

        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
· Now, we can read the data coming from the keyboard using read () and readLine () methods available in BufferedReader class.
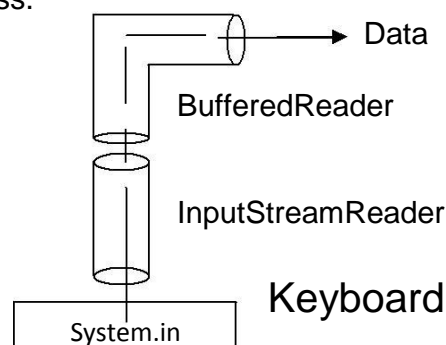

Figure: Reading data from keyboard

Accepting a Single Character from the Keyboard:
· Create a BufferedReader class object (br).
· Then read a single character from the keyboard using read() method as:

        char ch = (char) br.read();
· The read method reads a single character from the keyboard but it returns its ASCII number, which is an integer. Since, this integer number cannot be stored into character type variable ch, we should convert it into char type by writing (char) before the method. int data type is converted into char data type, converting one data type into another data type is called type casting.

Accepting a String from Keyboard:
· Create a BufferedReader class object (br).
· Then read a string from the keyboard using readLine() method as:

      String str = br.readLine ();
· readLine () method accepts a string from keyboard and returns the string into str. In this case, casting is not needed since readLine () is taking a string and returning the same data type.

Accepting an Integer value from Keyboard:
· First, we should accept the integer number from the keyboard as a string, using readLine ()
   as: String str = br.readLine ();
· Now, the number is in str, i.e. in form of a string. This should be converted into an int by using parseInt () method, method of Integer class as:

      int n = Integer.parseInt (str);
   If needed, the above two statements can be combined and
      written as: int n = Integer.parseInt (br.readLine());
· parseInt () is a static method in Integer class, so it can be called using class name as Integer.parseInt ().
· We are not using casting to convert String type into int type. The reason is String is a class and int is a fundamental data type. Converting a class type into a fundamental data type is not possible by using casting. It is possible by using the method Integer.parseInt().

Accepting a Float value from Keyboard:
· We can accept a float value from the keyboard with the help of the following statement:

      float n = Float.parseFloat (br.readLine());
· We are accepting a float value in the form of a string using br.readLine () and then passing the string to Float.parseFloat () to convert it into float. parseFloat () is a static method in Float class.

Accepting a Double value from Keyboard:
· We can accept a double value from the keyboard with the help of the following statement:

      double n = Double.parseDouble (br.readLine());
· We are accepting a double value in the form of a string using br.readLine () and then passing the string to Double.parseDouble () to convert it into double. parseDouble () is a static method in Double class.

Accepting Other Types of Values:
· To accept a byte value: byte n = Byte.parseByte (br.readLine ());
· To accept a short value: short n = Short.parseShort (br.readLine ());
· To accept a long value: long n = Long.parseLong (br.readLine ());
· To accept a boolean value: boolean x = Boolean.parseBoolean (br.readLine ());

If read () / readLine () method could not accept values due to some reason (like insufficient memory or illegal character), then it gives rise to a runtime error which is called by the name IOException, where IO stands for Input/Output and Exception represents runtime error.But we do not know how to handle this exception, in Java we can use throws command to throw the exception without handling it by writing:

throws IOException at the side of the method where read ()/ readLine () is used.

Program 1: Write a program to accept and display student details. // Accepting and displaying student details.

```java
import java.io.*;
class StudentDemo
{ public static void main(String args[]) throws IOException
        { // Create BufferedReader object to accept data
          BufferedReader br =new BufferedReader (new InputStreamReader
                (System.in)); //Accept student details
                System.out.print ("Enter roll number: ");
                int rno = Integer.parseInt (br.readLine());
                System.out.print ("Enter Gender (M/F):
                "); char gender = (char)br.read();
                br.skip (2);
                System.out.print ("Enter Student name:
                "); String name = br.readLine ()
                System.out.println ("Roll No.: " + rno);
                System.out.println ("Gender: " + gender);
                System.out.println ("Name: " + name);
        }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac StudentDemo.java

D:\JQR>java   StudentDemo
Enter roll number: 10
Enter Gender (M/F): M
Enter Student name: Kiran Kumar
Roll No.: 10
Gender: M
Name: Kiran Kumar
D:\JQR>_
```

In the above program after accepting gender of the student, br.skip (2) is used. The reason is that we used read () method to accept the gender value and then readLine () is used to accept the name. When we type M for gender and press enter, then it releases a \n code. So at gender column, we are giving two characters M and \n. But, read () method takes only the first character and rejects the next character, i.e. \n, which is trapped by the next readLine () method and name will accept \n. For this purpose, we can use skip () method of BufferedReader, which helps in skipping a specified number of characters. Suppose we take \n as two characters; now to skip them, we can write br.skip (2);

20

6. Arrays, Strings & StringBuffer

Arrays: An array represents a group of elements of same data type. Arrays are generally categorized into two types:

·   Single Dimensional arrays (or 1 Dimensional arrays)
·   Multi-Dimensional arrays (or 2 Dimensional arrays, 3 Dimensional arrays, …)

Single Dimensional Arrays: A one dimensional array or single dimensional array represents a row or a column of elements. For example, the marks obtained by a student in 5 different subjects can be represented by a 1D array.
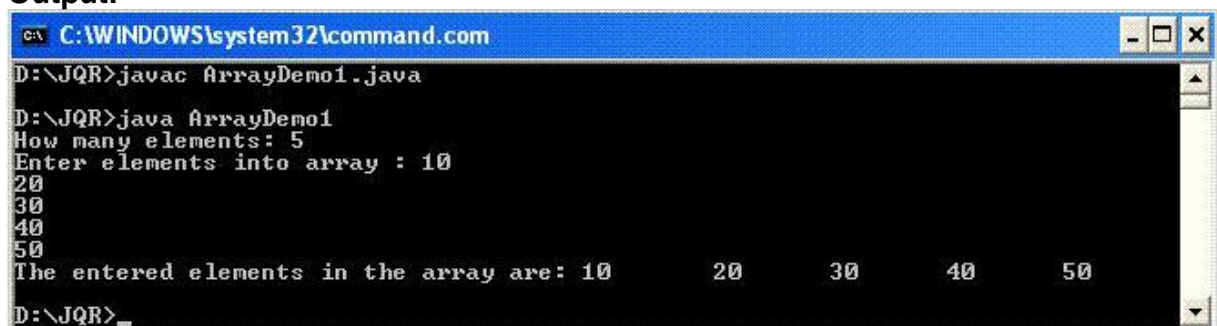
·   We can declare a one dimensional array and directly store elements at the time of its declaration, as: int marks[] = {50, 60, 55, 67, 70};
·   We can create a 1D array by declaring the array first and then allocate memory for it by using new operator, as: int marks[];   //declare marks array
                                    marks = new int[5];   //allot memory for storing 5 elements
    These two statements also can be written as: int marks [] = new int [5];

Program 1: Write a program to accept elements into an array and display the same. // program to accept elements into an array and display the same.

```
import java.io.*;
class ArrayDemo1
{ public static void main (String args[]) throws IOException {
      //Create a BufferedReader class object (br)
            BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
            System.out.println ("How many elements: " );
            int n = Integer.parseInt (br.readLine ());
            //create a 1D array with size n
            int a[] = new int[n];
            System.out.print ("Enter elements into array
            : "); for (int i = 0; i<n;i++)
                  a [i] = Integer.parseInt ( br.readLine ());
            System.out.print ("The entered elements in the array
            are: "); for (int i =0; i < n; i++)
                  System.out.print (a[i] + "\t");
      }
}
```

Output:

Multi-Dimensional Arrays (2D, 3D … arrays): Multi dimensional arrays represent 2D, 3D … arrays. A two dimensional array is a combination of two or more (1D) one dimensional arrays. A three dimensional array is a combination of two or more (2D) two dimensional arrays.

· **Two Dimensional Arrays (2d array):** A two dimensional array represents several rows and columns of data. To represent a two dimensional array, we should use two pairs of square braces [ ] [ ] after the array name. For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array.

 o We can declare a two dimensional array and directly store elements at the time of its declaration, as:

   int marks[] [] = {{50, 60, 55, 67, 70},{62, 65, 70, 70, 81}, {72, 66, 77, 80, 69} }; o We can create a two dimensional array by declaring the array first and then we can allot memory for it by using new operator as:

    int marks[ ] [ ];   //declare marks array
    marks = new int[3][5]; //allot memory for storing 15 elements.

These two statements also can be written as: int marks [ ][ ] = new int[3][5];

Program 2: Write a program to take a 2D array and display its elements in the form of a matrix. //Displaying a 2D array as a matrix

```
class Matrix
{ public static void main(String args[]) {
        //take a 2D array
            int x[ ][ ] = {{1, 2, 3}, {4, 5, 6} };
            // display the array elements
            for (int i = 0 ; i < 2 ; i++)
            { System.out.println (); for (int
                j = 0 ; j < 3 ; j++)
                    System.out.print(x[i][j] + "\t");
            }
        }
}
```

Output:

· **Three Dimensional arrays (3D arrays):** We can consider a three dimensional array as a combination of several two dimensional arrays. To represent a three dimensional array, we should use three pairs of square braces [ ] [ ] after the array name.

 o We can declare a three dimensional array and directly store elements at the time of its declaration, as:

   int arr[ ] [ ] [ ] = {{{50, 51, 52},{60, 61, 62}}, {{70, 71, 72}, {80, 81, 82}}};

o We can create a three dimensional array by declaring the array first and then we can allot memory for it by using new operator as:
int arr[ ] [ ] = new int[2][2][3]; //allot memory for storing 15 elements.

arrayname.length: If we want to know the size of any array, we can use the property 'length' of an array. In case of 2D, 3D length property gives the number of rows of the array.

Strings: A String represents group of characters. Strings are represented as String objects in java.

Creating Strings:
· We can declare a String variable and directly store a String literal using assignment operator.
String str = "Hello";
· We can create String object using new operator with some data.
String s1 = new String ("Java");
· We can create a String by using character array also.
char arr[] = { 'p','r','o','g','r','a','m'};
· We can create a String by passing array name to it, as:
String s2 = new String (arr);
· We can create a String by passing array name and specifying which characters we need:
String s3 = new String (str, 2, 3);
Here starting from 2nd character a total of 3 characters are copied into String s3.

String Class Methods:

Method	Description
String concat (String str)	Concatenates calling String with str. **Note:** + also used to do the same
int length ()	Returns length of a String
char charAt (int index)	Returns the character at specified location (from 0)
int compareTo (String str)	Returns a negative value if calling String is less than str, a positive value if calling String is greater than str or 0 if Strings are equal.
boolean equals (String str)	Returns true if calling String equals str. **Note:** == operator compares the references of the string objects. It does not compare the contents of the objects. equals () method compares the contents. While comparing the strings, equals () method should be used as it yields the correct result.
boolean equalsIgnoreCase (String str)	Same as above but ignores the case
boolean startsWith (String prefix)	Returns true if calling String starts with prefix
boolean endsWith (String suffix)	Returns true if calling String ends with suffix
int indexOf (String str)	Returns first occurrence of str in String.
int lastIndexOf(String str)	Returns last occurrence of str in the String. **Note:** Both the above methods return negative value, if str not

23

	found in calling String. Counting starts from 0.
String replace (char oldchar, char newchar)	returns a new String that is obtained by replacing all characters oldchar in String with newchar.
String substring (int beginIndex)	returns a new String consisting of all characters from beginIndex until the end of the String
String substring (int beginIndex, int endIndex)	returns a new String consisting of all characters from beginIndex until the endIndex.
String toLowerCase ()	converts all characters into lowercase
String toUpperCase ()	converts all characters into uppercase
String trim ()	eliminates all leading and trailing spaces

Program 3: Write a program using some important methods of String class. // program using String class methods

```
class StrOps
{ public static void main(String args [])
    { String str1 = "When it comes to Web programming, Java is #1.";
        String str2 = new String (str1);
        String str3 = "Java strings are
        powerful."; int result, idx; char ch;
        System.out.println ("Length of str1: " + str1.length
         ()); // display str1, one char at a time.
        for(int i=0; i < str1.length(); i++)
                System.out.print (str1.charAt (i));
        System.out.println ();
        if (str1.equals (str2) )
                System.out.println ("str1 equals str2");
        else
                System.out.println ("str1 does not equal
        str2"); if (str1.equals (str3) )
                 System.out.println ("str1 equals str3");
        else
                System.out.println ("str1 does not equal
        str3"); result = str1.compareTo (str3);
        if(result == 0)
                System.out.println ("str1 and str3 are
        equal"); else if(result < 0)
                System.out.println ("str1 is less than str3");
        else
                System.out.println ("str1 is greater than str3");
        str2 = "One Two Three One"; // assign a new string to str2 idx =
        str2.indexOf ("One");
        System.out.println ("Index of first occurrence of One: " +
        idx); idx = str2.lastIndexOf("One");
        System.out.println ("Index of last occurrence of One: " + idx);
    }
```

}
Output:


```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac StrOps.java

D:\JQR>java  StrOps
Length of str1: 46
When it comes to Web programming,  Java is #1.
str1 equals str2
str1 does not equal str3
str1 is greater than str3
Index of first occurrence of One: 0
Index of last occurrence of One: 14

D:\JQR>
```

We can divide objects broadly as mutable and immutable objects. Mutable objects are those objects whose contents can be modified. Immutable objects are those objects, once created can not be modified. String objects are immutable. The methods that directly manipulate data of the object are not available in String class.

StringBuffer: StringBuffer objects are mutable, so they can be modified. The methods that directly manipulate data of the object are available in StringBuffer class.

Creating StringBuffer:
·   We can create a StringBuffer object by using new operator and pass the string to the object,
    as: StringBuffer sb = new StringBuffer ("Kiran");
·   We can create a StringBuffer object by first allotting memory to the StringBuffer object
    using new operator and later storing the String into it as:
                     StringBuffer sb = new StringBuffer (30);
In general a StringBuffer object will be created with a default capacity of 16 characters. Here, StringBuffer object is created as an empty object with a capacity for storing 30 characters. Even if we declare the capacity as 30, it is possible to store more than 30 characters into StringBuffer. To store characters, we can use append () method as:
                     Sb.append ("Kiran");

StringBuffer Class Methods:

Method	Description
StringBuffer append (x)	x may be int, float, double, char, String or StringBuffer. It will be appended to calling StringBuffer
StringBuffer insert (int offset, x)	x may be int, float, double, char, String or StringBuffer. It will be inserted into the StringBuffer at offset.
StringBuffer delete (int start, int end)	Removes characters from start to end
StringBuffer reverse ()	Reverses character sequence in the StringBuffer
String toString ()	Converts StringBuffer into a String
int length ()	Returns length of the StringBuffer

25

Program 4: Write a program using some important methods of StringBuffer class. // program using StringBuffer class methods

```java
import java.io.*;
class Mutable
{ public static void main(String[] args) throws IOException {
        // to accept data from keyboard
        BufferedReader br=new BufferedReader (new InputStreamReader
                (System.in)); System.out.print ("Enter sur name : ");
                String sur=br.readLine ( );
                System.out.print ("Enter mid name :
                "); String mid=br.readLine ( );
                System.out.print ("Enter last name :
                "); String last=br.readLine ( );
                // create String Buffer object
                StringBuffer sb=new StringBuffer ( );
                // append sur, last to sb
                sb.append (sur);
                sb.append (last);
                // insert mid after sur
                int n=sur.length ( );
                sb.insert (n, mid);
                // display full name
                System.out.println ("Full name = "+sb);
                System.out.println ("In reverse ="+sb.reverse ( ));
        }
}
```

Output:

7. Introduction to OOPs

Languages like Pascal, C, FORTRAN, and COBOL are called procedure oriented programming languages. Since in these languages, a programmer uses procedures or functions to perform a task. When the programmer wants to write a program, he will first divide the task into separate sub tasks, each of which is expressed as functions/ procedures. This approach is called procedure oriented approach.

The languages like C++ and Java use classes and object in their programs and are called Object Oriented Programming languages. The main task is divided into several modules and these are represented as classes. Each class can perform some tasks for which several methods are written in a class. This approach is called Object Oriented approach.

Difference between Procedure Oriented Programming and OOP:

Procedure Oriented Programming	Object Oriented Programming
1. Main program is divided into small parts depending on the functions.	1. Main program is divided into small object depending on the problem.
2. The Different parts of the program connect with each other by parameter passing & using operating system.	2. Functions of object linked with object using message passing.
3. Every function contains different data.	3. Data & functions of each individual object act like a single unit.
4. Functions get more importance than data in program.	4. Data gets more importance than functions in program.
5. Most of the functions use global data.	5. Each object controls its own data.
6. Same data may be transfer from one function to another	6. Data does not possible transfer from one object to another.
7. There is no perfect way for data hiding.	7. Data hiding possible in OOP which prevent illegal access of function from outside of it. This is one of the best advantages of OOP also.
8. Functions communicate with other functions maintaining as usual rules.	8. One object link with other using the message passing.
9. More data or functions can not be added with program if necessary. For this purpose full program need to be change.	9. More data or functions can be added with program if necessary. For this purpose full program need not to be change.
10. To add new data in program user should be ensure that function allows it.	10. Message passing ensure the permission of accessing member of an object from other object.
11. Top down process is followed for program design.	11. Bottom up process is followed for program design.
12. Example: Pascal, Fortran	12. Example: C++, Java.

Features of OOP:

· **Class:** In object-oriented programming, a class is a programming language construct that is used as a blueprint to create objects. This blueprint includes attributes and methods that the created objects all share. Usually, a class represents a person, place, or thing - it is an abstraction of a concept within a computer program. Fundamentally, it encapsulates the state

27

and behavior of that which it conceptually represents. It encapsulates state through data placeholders called member variables; it encapsulates behavior through reusable code called methods.

General form of a class:

```
class class_name
{           Properties
        (variables); Actions
        (methods);
}
```

e.g.:   class Student
        { //properties -- variables int
                rollNo;
                String name;
                //methods -- actions
                void display ()
                {
                        System.out.println ("Student Roll Number is: " + rollNo);
                        System.out.println ("Student Name is: " + name);
                }
        }

Note: Variables inside a class are called as instance variables.
        Variables inside a method are called as method variables.

· **Object:** An Object is a real time entity. An object is an instance of a class. Instance means physically happening. An object will have some properties and it can perform some actions. Object contains variables and methods. The objects which exhibit similar properties and actions are grouped under one class. "To give a real world analogy, a house is constructed according to a specification. Here, the specification is a blueprint that represents a class, and the constructed house represents the object".

  o To access the properties and methods of a class, we must declare a variable of that class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.

  o We must acquire an actual, physical copy of the object and assign it to that variable. We can do this using **new** operator. The new operator dynamically allocates memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by new. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

General form of an Object:

        Class_name variable_name;   // declare reference to object
        variable_name = new Class_name ( );   // allocate an object

e.g.:   Student s;   // s is reference variable
        s = new Student ();   // allocate an object to reference variable s

The above two steps can be combined and rewritten in a single
        statement as: Student s = new Student ();

Now we can access the properties and methods of a class by using object with dot operator as:
        s.rollNo, s.name, s.display ()

28

Java Programming

- **Encapsulation:** Wrapping up of data (variables) and methods into single unit is called Encapsulation. Class is an example for encapsulation. Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Encapsulation is the technique of making the fields in a class private and providing access to the fields via methods. If a field is declared private, it cannot be accessed by anyone outside the class.

  e.g.:   class Student
  ```
          {
                  private int rollNo;
                  private String name;
                  //methods -- actions
                  void display ()
                  {
                          System.out.println ("Student Roll Number is: " + rollNo);
                          System.out.println ("Student Name is: " + name);
                  }
          }
  ```

- **Abstraction:** Providing the essential features without its inner details is called abstraction (or) hiding internal implementation is called Abstraction. We can enhance the internal implementation without effecting outside world. Abstraction provides security. A class contains lot of data and the user does not need the entire data. The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data. A bank clerk should see the customer details like account number, name and balance amount in the account. He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank etc. So such data can be abstracted from the clerks view.

  e.g.:   class Bank
  ```
          { private int accno; private
                  String name; private
                  float balance;
                  private float profit;
                  private float loan;
                  void display_to_clerk ()
                  {
                          System.out.println ("Accno = " + accno);
                          System.out.println ("Name = " + name);
                          System.out.println ("Balance = " + balance);
                  }
          }
  ```
  In the preceding class, inspite of several data items, the display_to_clerk () method is able to access and display only the accno, name and balance values. It cannot access profit and loan of the customer. This means the profit and loan data is hidden from the view of the bank clerk.

- **Inheritance:** Acquiring the properties from one class to another class is called inheritance (or) producing new class from already existing class is called inheritance. Reusability of code

Java Programming

is main advantage of inheritance. In Java inheritance is achieved by using extends keyword. The properties with access specifier private cannot be inherited.

e.g.:
```
class Parent
{
    String parentName;
    String familyName;
}
class Child extends Parent
{
        String childName;
        int childAge;
        void printMyName()
        {
                System.out.println ("My name is"+childName+" "+familyName);
        }
}
```
In the above example, the child has inherited its family name from the parent class just by inheriting the class.

· **Polymorphism:** The word polymorphism came from two Greek words 'poly' means 'many' and 'morphos' means 'forms'. Thus, polymorphism represents the ability to assume several different forms. The ability to define more than one function with the same name is called Polymorphism

e.g.: int add (int a, int b) float
add (float a, int b)
float add (int a , float
b) void add (float a)
int add (int a)

· **Message Passing:** Calling a method in a class is called message passing. We can call methods of a class by using object with dot operator as:
object_name.method_name ();
e.g.: s.display (); ob.add (2, 5);   ob.printMyName ();

Program 1: Write a program to display details of student using class and object.
```
//Program to display the details of a student using class and
object class Student
 {      int rollNo;               //properties -- variables
      String name;
      void display ()          //method -- action
      {      System.out.println ("Student Roll Number is: " + rollNo);
              System.out.println ("Student Name is: " + name);
      }
}
class StudentDemo
{ public static void main(String args[])
      {
```

30

```
                    //create an object to Student
                    class Student s = new Student ();
                    //call display () method inside Student class using
                    object s s.display ();
            }
}
```

Output:

When the programmer does not initialize the instance variables, java compiler will write code and initializes the variables with default values.

Data Type	Default Value
Int	0
Float	0.0
Double	0.0
Char	Space
String	null
Class	null
Boolean	false

Initializing Instance Variables:

· **Type 1:** We can initialize instance variables directly in the class using assignment operator. In this type every object is initialized with the same data.

```
            int rollNo = 101;
            String name = "Kiran";
```

Program 2: Let us rewrite the Program 1.

```
//Program to display the details of a student using class and
object class Student
{ int rollNo = 101; String name =
            "Surya"; void display
            ()
            { System.out.println ("Student Roll Number is: " + rollNo);
                    System.out.print ("Student Name is: " + name);
            }
}
class StudentDemo
{ public  static void  main(String  args[]) {
            Student s1 = new Student ();
                    System.out.println ("First Student Details : " );
```

31

```
                    s1.display ();
                    Student s2 = new Student ();
                    System.out.println ("Second Student Details :
                    " ); s2.display ();
            }
    }
```

Output:

```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Surya
Second Student Details :
Student Roll Number is: 101
Student Name is: Surya

D:\JQR>_
```

· **Type 2:** We can initialize one class instance variables in another class using reference variable.

```
            s.rollNo = 101;
            s.name = "Kiran";
```

Program 3: Let us rewrite the Program 1.

```
//Program to display the details of a student using class and
object class Student
{           int rollNo;
            String name;
            void display ()
            {
                    System.out.println ("Student Roll Number is: " + rollNo);
                    System.out.print ("Student Name is: " + name);
            }
}
class StudentDemo
{ public static void main(String args[]) {
            Student s1 = new Student ();
                    System.out.println ("First Student Details :
                    " ); s1.rollNo = 101;
                    s1.name = "Suresh";
                    s1.display ();
                    Student s2 = new Student ();
                    System.out.println ("Second Student Details :
                    " ); s2.rollNo = 102;
                    s2.name = "Ramesh";
                    s2.display ();
            }
}
```

32

Output:


```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×
D:\JQR>javac StudentDemo.java

D:\JQR>java   StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Suresh
Second Student Details :
Student Roll Number is: 102
Student Name is: Ramesh

D:\JQR>
```

In this type of initialization the properties (variables in the class) are not available, if they are declared as private.

Access Specifiers: An access specifier is a key word that represents how to access a member of a class. There are four access specifiers in java.
o   **private:** private members of a class are not available outside the class.
o **public:** public members of a class are available anywhere outside the
class. o **protected:** protected members are available outside the class.
o **default:** if no access specifier is used then default specifier is used by java compiler. Default members are available outside the class.

·   **Type 3:** We can initialize instance variables using a constructor.
Constructor:
o A constructor is similar to a method that initializes the instance variables of
a class. o A constructor name and classname must be same.
o A constructor may have or may not have parameters. Parameters are local variables to receive data.
o A constructor without any parameters is called default
constructor. **e.g.** class Student
```
        { int rollNo; String
              name;
              Student ()
              { rollNo = 101; name
                    = "Kiran";
              }
        }
```
o A constructor with one or more parameters is called parameterized
constructor. **e.g.** class Student
```
        { int rollNo; String
              name;
              Student (int r, String n)
              {   rollNo  =   r;
                    name = n;
              }
        }
```
o A constructor does not return any value, not even void.

33

o A constructor is called and executed at the time of creating an
object. o A constructor is called only once per object.
o Default constructor is used to initialize every object with same data where as
parameterized constructor is used to initialize each object with different data.
o If no constructor is written in a class then java compiler will provide default values.

**Program 4: Write a program to initialize student details using default
constructor and display the same.**
//Program to initialize student details using default constructor and displaying the
same. class Student
{ int rollNo; String
        name;
        Student ()
        { rollNo = 101;
                name = "Suresh";
        }
        void display ()
        { System.out.println ("Student Roll Number is: " + rollNo);
                System.out.println ("Student Name is: " + name);
        }
}
class StudentDemo
{ public static void main(String args[]) {
        Student s1 = new Student ();
                System.out.println ("s1 object contains:
                "); s1.display ();
                Student s2 = new Student ();
                System.out.println ("s2 object contains:
                "); s2.display ();
        }
}
Output:

**Program 5: Write a program to initialize student details using Parameterized
constructor and display the same.**
//Program to initialize student details using parameterized
constructor class Student
{ int rollNo;

34

```
                String name;
                Student (int r, String n)
                {   rollNo   =   r;
                        name = n;
                }
                void display ()
                { System.out.println ("Student Roll Number is: " + rollNo);
                        System.out.println ("Student Name is: " + name);
                }
    }
    class StudentDemo
    { public static void main(String args[])
                { Student s1 = new Student (101, "Suresh");
                        System.out.println ("s1 object contains:
                        " ); s1.display ();
                        Student s2 = new Student (102, "Ramesh");
                        System.out.println ("s2 object contains:
                        " ); s2.display ();
        }
    }
```

Output:

The keyword 'this': There will be situations where a method wants to refer to the object which invoked it. To perform this we use 'this' keyword. There are no restrictions to use 'this' keyword we can use this inside any method for referring the current object. This keyword is always a reference to the object on which the method was invoked. We can use 'this' keyword wherever a reference to an object of the current class type is permitted. 'this' is a key word that refers to present class object. It refers to

· Present class instance variables
· Present class methods.
· Present class constructor.

Program 6: Write a program to use 'this' to refer the current class parameterized constructor and current class instance variable.
```
//this demo
class Person
{ String name;
```

35

```
        Person ( )
        { this ("Ravi Sekhar"); // calling present class parameterized constructor
                this.display ( ); // calling present class method
        }
        Person (String name)
        { this.name = name; // assigning present class variable with parameter "name"
        }
        void display( )
        { System.out.println ("Person Name is = " + name);
        }
}
class ThisDemo
{ public static void main(String args[])
        {
                Person p = new Person ( );
        }
}
```

Output:

Garbage Collection: Generally memory is allocated to objects by using 'new' operator and deleting an allocated memory is uncommon. This deletion of memory is supported by delete operator in C++ but this deletion of allocated memory works automatically in Java. This automatic deletion of already allocated but unused memory is called as garbage collection. This operation of garbage collection is accomplished by a method named "gc ()". This method is used for garbage collection.

The finalize( ) Method: It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize( ) method. To add a finalizer to a class, simply define the finalize( ) method. The Java runtime calls that method whenever it is about to recycle an object of that class. Inside the finalize( ) method specify those actions that must be performed before an object is destroyed. The finalize( ) method has this general form:

```
        protected void finalize( )
        {
                // finalization code here
        }
```

Here, the keyword protected is a specifier that prevents access to finalize ( ) by code defined outside its class. This means that you cannot know when.or even if.finalize ( ) will be executed. For example, if your program ends before garbage collection occurs, finalize ( ) will not execute.

8. Methods & Inner Class

Methods: A method represents a group of statements to perform a task. A method contains two parts: method header (or) method prototype is first part. This part contains method name, method parameters and method returntype.

return_type methodname (param1, param2, ……)

e.g.: double sum (double d1, double d2), void sum (), float power (float x, int n), etc.
The second part contains method body. This part represents the logic to perform the task in the form of a group of statements.

```
{
        Statements;
}
```
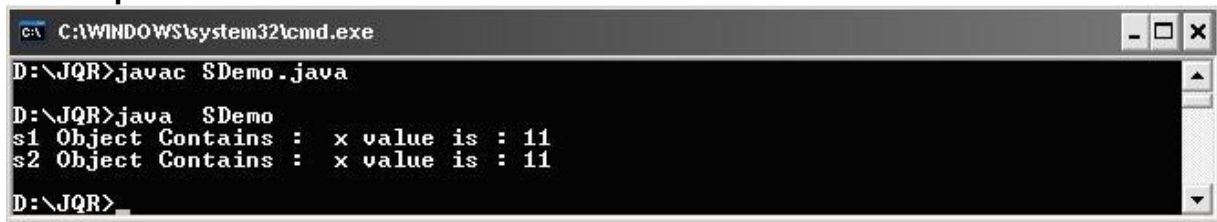
Note: If a method returns a value then return statement should be written in method body. e.g.: return x; return 1; return (x+y-2);

· **Instance Methods:**
o Methods which act upon instance variables of a class are called instance methods. o To call instance methods use objectname.methodname.
o Instance variable is a variable whose separate copy is available in every object.
o Any modifications to instance variable in one object will not affect the instance variable of other objects. These variables are created on heap.

Program 1: Write a program to access instance variable using instance method. //instance method accessing instance variable

```
class Sample
{ int x = 10; void
        display( ) {
        x++;
                System.out.println (" x value is : " + x);
        }
}
class SDemo
{ public  static  void  main(String  args[]) {
        Sample s1 = new Sample( );
                System.out.print ("s1 Object Contains :
                "); s1.display ();
                Sample s2 = new Sample( );
                System.out.print ("s2 Object Contains :
                "); s2.display ();
        }
}
```

Output:


```
D:\JQR>javac SDemo.java

D:\JQR>java  SDemo
s1 Object Contains :   x value is : 11
s2 Object Contains :   x value is : 11

D:\JQR>
```

Note: Instance methods can read and act upon static variables also.
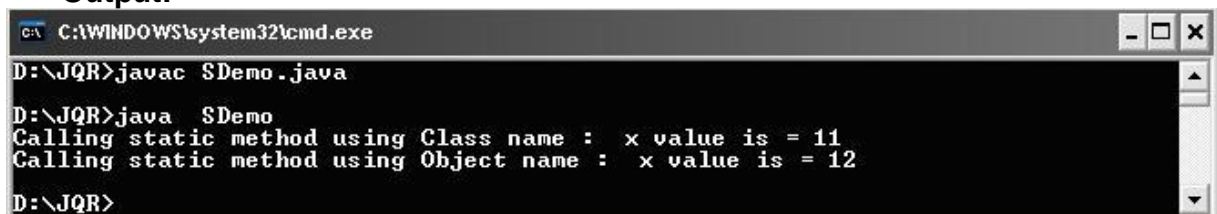
·   **Static Methods:**
  o   Static methods can read and act upon static variables.
  o   Static methods cannot read and act upon instance variables.
  o Static variable is a variable whose single copy is shared by all the
objects. o Static methods are declared using keyword static.
  o Static methods can be called using objectname.methodname (or) classname.methodname.
  o From any object, if static variable is modified it affects all the objects. Static variables are
    stored on method area.

Program 2: Write a program to access static variable using static
method. //static method accessing static variable
class Sample
{ static int x = 10; static void
        display( )
        { x++;
                System.out.println (" x value is = " + x);
        }
}
class SDemo
{ public static void main(String args[])
        { System.out.print ("Calling static method using Class name : ");
                Sample.display ();
                Sample s1 = new Sample ( );
                System.out.print ("Calling static method using Object name
                : "); s1.display ();
        }
    }
Output:


```
D:\JQR>javac SDemo.java

D:\JQR>java  SDemo
Calling static method using Class name :   x value is = 11
Calling static method using Object name :   x value is = 12

D:\JQR>
```

Inner Class: A class with in another class is called Inner class. When the programmer wants to restrict the access of entire code of a class, creates an inner class as a private class. The way to access the inner class is through its outer class only.

- Inner class is a safety mechanism.
- Inner class is hidden in outer class from other classes.
- Only inner class can be private.
- An object to Inner class can be created only in its outer class.
- An object to Inner class cannot be created in any other class.
- Outer class object and Inner class objects are created in separate memory locations.
- Outer class members are available to Inner class object.
- Inner class object will have an additional invisible field called 'this$0' that stores a reference of outer class object.
  - Inner class members are referenced as: this.member;
  - Outer class members are referred as: Outerclass.this.member;

Program 3: Write a program to access private members of a class using inner class. // inner class demo

```java
class Bank
{ private double bal, rate; Bank
      (double b, double r)
      { bal=b; rate =
            r;
      }
      void display ( )
      {  Interest  in=new  Interest  ();
            in.calculateInterest ( );
            System.out.println ("New Balance : " + bal);
      }
      private class Interest
      { void calculateInterest ( )
            { System.out.println ("Balance = "+ bal);
                  double interest=bal* rate/100;
                  System.out.println ("interest =
                  "+interest); bal+=interest;
            }
      }
}
class InnerDemo
{ public static void main (String args[])
      { Bank  account = new Bank (20000, 5);
            account.display ();
      }
}
```

Java Programming

Output:

```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×

D:\JQR>javac InnerDemo.java

D:\JQR>java  InnerDemo
Balance = 20000.0
interest = 1000.0
New Balance : 21000.0

D:\JQR>_
```

9. Inheritance

Inheritance: Creating new class from existing class such that the features of existing class are available to the new class is called inheritance. Already existing class is called super class & produced class is called sub class. Using inheritance while creating sub classes a programmer can reuse the super class code without rewriting it.

Syntax: class subclass_name extends

superclass_name **e.g.:** class Child extends Parent

Program 1: Write a program to create a Person class which contains general details of a person and create a sub class Employ which contains company details of a person. Reuse the general details of the person in its sub class.

```java
// Inheritance Example
class Person
{ String name;
      String   permanentAddress;
      int age;
      void set_PermanentDetails (String name, String permanentAddress,
      int age) { this.name = name;
            this.permanentAddress            =
            permanentAddress; this.age = age;
      }
      void get_PermanentDetails ()
      { System.out.println ("Name : " + name);
            System.out.println   ("Permanent   Address   :   "   +
            permanentAddress); System.out.println ("Age :" + age);
      }
}
class Employ extends Person
{ int id;
      String companyName;
      String companyAddress;
      Employ (int id, String name, String permanentAddress, int age,
                              String companyName, String companyAddress)
      { this.id = id;
            set_PermanentDetails (name, permanentAddress, age);
            this.companyName = companyName;
            this.companyAddress = companyAddress;
      }
      void get_EmployDetails ()
      { System.out.println ("Employ  Id  :  " + id);
            get_PermanentDetails ();
            System.out.println ("Company Name : "+ companyName);
            System.out.println ("Company Address : "+companyAddress);
      }
```

```
}
class InherDemo
{ public static void main (String args [])
        { Employ e1 = new Employ (101, "Suresh Kumar", "18-Madhura Nagar-Tirupati",
                            29, "Centris Software- Chennai", "20-RVS
                Nagar"); e1.get_EmployDetails ();
        }
}
```
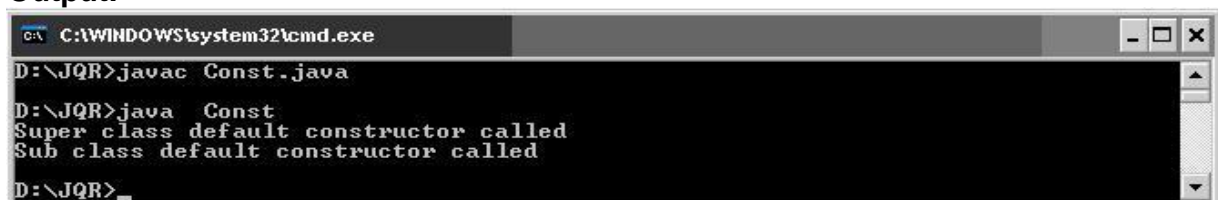
Output:

Program 2: Write a program to illustrate the order of calling of default constructor in super and sub class.

```
// Default constructors in super and sub
class class One
{       One ( )          //super class default constructor
        {
                System.out.println ("Super class default constructor called");
        }
}
class Two extends One
{       Two ( )          //sub class default constructor
        {
                System.out.println ("Sub class default constructor called");
        }
}
class Const
{ public static void main (String args[])
        {       Two t=new Two ( );  //create sub class object
        }
 }
```

Output:

· Super class default constructor is available to sub class by default.
· First super class default constructor is executed then sub class default constructor is executed.
· Super class parameterized constructor is not automatically available to subclass. super is the key word that refers to super class.
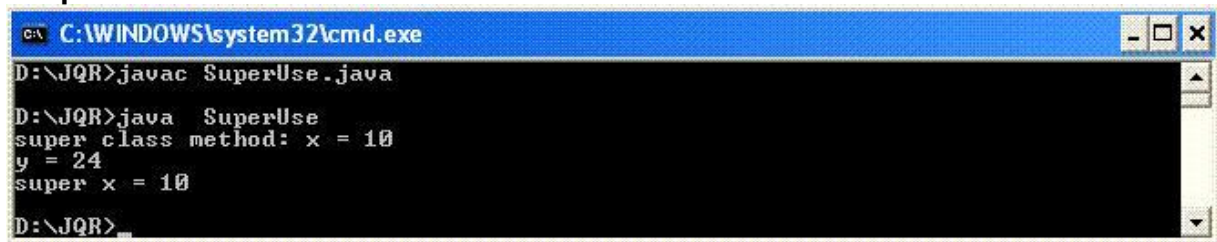
The keyword 'super':
· super can be used to refer super class variables as: super.variable
· super can be used to refer super class methods as: super.method ()
· super can be used to refer super class constructor as: super (values)

Program 3: Write a program to access the super class method, super class parameterized constructor and super class instance variable by using super keyword from sub class.

```java
// super refers to super class- constructors, instance variables and
methods class A
{ int x;
      A (int x)
      {
            this.x = x;
      }
      void show( )
      { System.out.println("super class method: x = "+x);
      }
}
class B extends A
{      int y;
      B (int a,int b)
      {
            super(a);      // (or) x=a;
            y=b;
      }
      void show( )
      {      super.show ();
            System.out.println ("y = "+y);
            System.out.println (" super x = " + super.x);
      }
}
class SuperUse
{
      public static void main(String args[])
      { B  ob = new B (10, 24);
            ob.show ( );
      }
}
```

43

Output:

- Super key word is used in sub class only.
- The statement calling super class constructor should be the first one in sub class constructor.

10. Polymorphism

Polymorphism came from the two Greek words 'poly' means many and morphos means forms. If the same method has ability to take more than one form to perform several tasks then it is called polymorphism. It is of two types: Dynamic polymorphism and Static polymorphism.
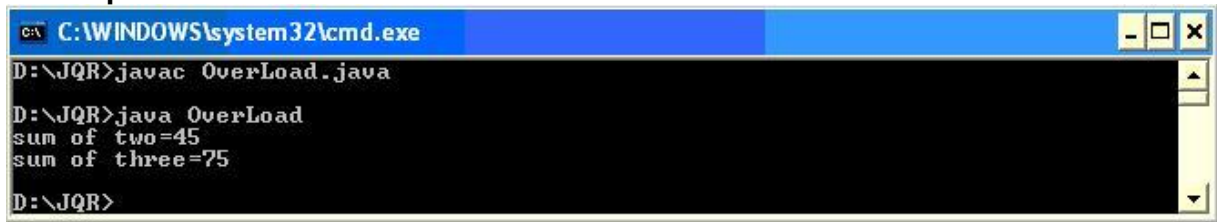
· **Dynamic Polymorphism:** The polymorphism exhibited at run time is called dynamic polymorphism. In this dynamic polymorphism a method call is linked with method body at the time of execution by JVM. Java compiler does not know which method is called at the time of compilation. This is also known as dynamic binding or run time polymorphism. Method overloading and method overriding are examples of Dynamic Polymorphism in Java. o

Method Overloading: Writing two or more methods with the same name, but with a difference in the method signatures is called method over loading. Method signature represents the method name along with the method parameters. In method over loading JVM understands which method is called depending upon the difference in the method signature. The difference may be due to the following:

➢ There is a difference in the no. of parameters. void add (int a,int b)
void add (int a,int b,int c)

➢ There is a difference in the data types of parameters. void add (int a,float b)
void add (double a,double b)

➢ There is a difference in the sequence of parameters. void swap (int a,char b)
void swap (char a,int b)

Program 1: Write a program to create a class which contains two methods with the same name but with different signatures.

```
// overloading of methods --------- Dynamic
polymorphism class Sample
{ void add(int a,int b)
        {
                System.out.println ("sum of two="+ (a+b));
        }
        void add(int a,int b,int c)
        {
                System.out.println ("sum of three="+ (a+b+c));
        }
}
class OverLoad
{ public static void main(String[] args) {
        Sample s=new Sample ( );
                s.add (20, 25);
                s.add (20, 25, 30);
        }
}
```

Output:


```
D:\JQR>javac OverLoad.java

D:\JQR>java OverLoad
sum of two=45
sum of three=75

D:\JQR>
```

· **Method Overriding:** Writing two or more methods in super & sub classes with same name and same signatures is called method overriding. In method overriding JVM executes a method depending on the type of the object.
Program 2: Write a program that contains a super and sub class which contains a method with same name and same method signature, behavior of the method is dynamically decided. //overriding of methods --------------- Dynamic polymorphism

```
class Animal
{   void move()
    {
            System.out.println ("Animals can move");
    }
}
class Dog extends Animal
{   void move()
    {
            System.out.println ("Dogs can walk and run");
    }
}
public class OverRide
{   public static void main(String args[])
    { Animal a = new Animal (); // Animal reference and object
            Animal b = new Dog (); // Animal reference but Dog
            object a.move (); // runs the method in Animal class
            b.move (); //Runs the method in Dog class
    }
}
```

Output:


```
D:\JQR>javac OverRide.java

D:\JQR>java  OverRide
Animals can move
Dogs can walk and run

D:\JQR>
```

Achieving method overloading & method overriding using instance methods is an example of dynamic polymorphism.

· **Static Polymorphism:** The polymorphism exhibited at compile time is called Static polymorphism. Here the compiler knows which method is called at the compilation. This is also called compile time polymorphism or static binding. Achieving method overloading & method overriding using private, static and final methods is an example of Static Polymorphism.

Program 3: Write a program to illustrate static polymorphism. //Static Polymorphism

```
class Animal
{ static void move ()
            { System.out.println ("Animals can move");
            }
}
class Dog extends Animal
{ static void move ()
            { System.out.println ("Dogs can walk and run");
            }
}
public class StaticPoly
{ public static void main(String args[])
            {    Animal.move    ();
                   Dog.move ();
            }
}
```

Output:

The keyword 'final':
· final keyword before a class prevents inheritance. **e.g.:** final class A
        class B extends A //invalid
· final keyword before a method prevents overriding.
· final keyword before a variable makes that variable as a constant. **e.g.:** final double PI = 3.14159; //PI is a constant.

Type Casting: Converting one data type into another data type is called casting. Type cast operator is used to convert one data type into another data type. Data type represents the type of the data stored into a variable. There are two kinds of data types:
· **Primitive Data type:** Primitive data type represents singular values. **e.g.:** byte, short, int, long, float, double, char, boolean.

47

Using casting we can convert a primitive data type into another primitive data type. This is done in two ways, widening and narrowing.

o **Widening:** Converting a lower data type into higher data type is called widening. byte, short, int, long , float, double
   e.g.: char ch = 'a';
      int n = (int) ch;
   e.g.: int n = 12;
      float f = (float) n;
o **Narrowing:** Converting a higher data type into lower data type is called narrowing. **e.g.:** int i = 65;
      char ch = (char)
i; **e.g.:** float f = 12.5;
      int i = (int) f;
· **Referenced Data type:** Referenced data type represents multiple values. **e.g.:** class, String

Using casting we can convert one class type into another class type if they are related by means of inheritance.

o **Generalization:** Moving back from subclass to super class is called generalization or widening or upcasting.
o **Specialization:** Moving from super class to sub class is called specialization or narrowing or downcasting.

Program 4: Write a program to convert one class type into another class type. // conversion of one class type into another class type

```
class One
{ void show1()
            { System.out.println ("One's method");
            }
}
class Two extends One
{ void show2()
            { System.out.println ("Two's method");
            }
}
class Ex3
{
            public static void main(String args[])
            {
    /* If super class reference is used to refer to super class object then only super
    class members are available to programmer. */
                One ob1 = new One
                (); ob1.show1 ();
    /* If sub class reference is used to refer to sub class object then super class
    members as well as sub class members are available to the programmer. */
                Two ob2 = new
                Two(); ob2.show1();
```

48

```
                    ob2.show2();
        /* If super class reference is used to refer to sub class object then super class methods are
        available, sub class methods are not available unless they override super class methods */
                    One ob3 = (One) new Two(); // Generalization
                    ob3.show1();
        /* It is not possible to access any methods if we use subclass object to refer to
        super class as above */
                    Two ob4 = (Two) new
                    One(); ob4.show1();
                    ob4.show2();
            // Specialization
                    One ob5 = (One) new
                    Two(); Two ob6 = (Two)
                    ob5; ob6.show1();
                    ob6.show2();
            }
    }
```

Note: Using casting it is not possible to convert a primitive data type into a referenced data type and vice-versa. For this we are using Wrapper classes.

11. Abstract Class

A method with method body is called concrete method. In general any class will have all concrete methods. A method without method body is called abstract method. A class that contains abstract method is called abstract class. It is possible to implement the abstract methods differently in the subclasses of an abstract class. These different implementations will help the programmer to perform different tasks depending on the need of the sub classes. Moreover, the common members of the abstract class are also shared by the sub classes.

The abstract methods and abstract class should be declared using the keyword abstract. We cannot create objects to abstract class because it is having incomplete code. Whenever an abstract class is created, subclass should be created to it and the abstract methods should be implemented in the subclasses, then we can create objects to the subclasses.

- An abstract class is a class with zero or more abstract methods
- An abstract class contains instance variables & concrete methods in addition to abstract methods.
- It is not possible to create objects to abstract class.
- But we can create a reference of abstract class type.
- All the abstract methods of the abstract class should be implemented in its sub classes.
- If any method is not implemented, then that sub class should be declared as 'abstract'.
- Abstract class reference can be used to refer to the objects of its sub classes.
- Abstract class references cannot refer to the individual methods of sub classes.
- A class cannot be both 'abstract' & 'final'.
  e.g.: final abstract class A // invalid

Program 1: Write an example program for abstract
class. // Using abstract methods and classes.

```
abstract class Figure
{     double    dim1;
      double dim2;
      Figure (double a, double b)
      {     dim1 = a;
            dim2 = b;
      }
      abstract double area ();      // area is now an abstract method
}
class Rectangle extends Figure
{ Rectangle (double a, double b)
      {     super (a, b);
      }
      double area ()        // override area for rectangle
      { System.out.println ("Inside  Area  of  Rectangle.");
            return dim1 * dim2;
      }
}
```

```
class Triangle extends Figure
{ Triangle (double a, double b) {
      super (a, b);
      }
      double area()          // override area for right triangle
      { System.out.println ("Inside  Area  of  Triangle.");
            return dim1 * dim2 / 2;
      }
}
class AbstractAreas
{ public static void main(String args[])
      { // Figure f = new Figure(10, 10); // illegal now
            Rectangle r = new Rectangle(9, 5);
            Triangle t = new Triangle(10, 8);
            System.out.println("Area is " + r.area());
            System.out.println("Area is " + t.area());
      }
}
```

Output:

12. Interface

A programmer uses an abstract class when there are some common features shared by all the objects. A programmer writes an interface when all the features have different implementations for different objects. Interfaces are written when the programmer wants to leave the implementation to third party vendors. An interface is a specification of method prototypes. All the methods in an interface are abstract methods.

· An interface is a specification of method prototypes.
· An interface contains zero or more abstract methods.
· All the methods of interface are public, abstract by default.
· An interface may contain variables which are by default public static final.
· Once an interface is written any third party vendor can implement it.
· All the methods of the interface should be implemented in its implementation classes.
· If any one of the method is not implemented, then that implementation class should be declared as abstract.
· We cannot create an object to an interface.
· We can create a reference variable to an interface.
· An interface cannot implement another interface.
· An interface can extend another interface.
· A class can implement multiple interfaces.

Program 1: Write an example program for
interface interface Shape
```
{ void area (); void
        volume ();
        double pi = 3.14;
}
class Circle implements Shape
{ double r;
        Circle (double radius)
        { r = radius;
        }
        public void area ()
        { System.out.println ("Area of a circle is : " + pi*r*r );
        }
        public void volume ()
        { System.out.println ("Volume of  a circle is : " + 2*pi*r);
        }
}
class Rectangle implements Shape
{ double l,b;
        Rectangle  (double   length,   double
        breadth) { l = length;
                b = breadth;
```

```
        }
        public void area ()
        { System.out.println ("Area of a Rectangle is : " + l*b );
        }
        public void volume ()
        { System.out.println ("Volume of  a Rectangle is : " + 2*(l+b));
        }
}
class InterfaceDemo
{ public static void main (String args[]) {
        Circle ob1 = new Circle (10.2);
                ob1.area ();
                ob1.volume ();
                Rectangle  ob2  =  new  Rectangle  (12.6,
                23.55); ob2.area ();
                ob2.volume ();
        }
}
```
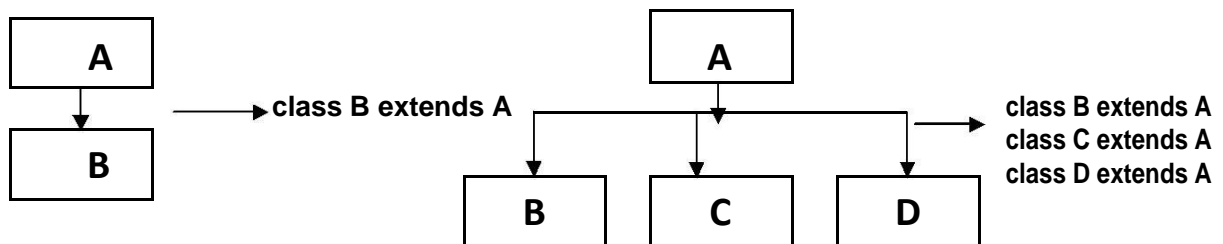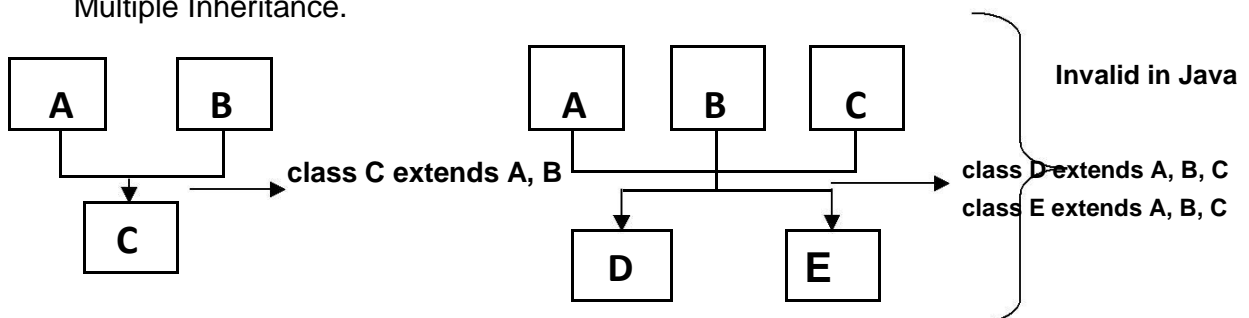
Output:

Types of inheritance:

· **Single Inheritance:** Producing subclass from a single super class is called single inheritance.

· **Multiple Inheritance:** Producing subclass from more than one super class is called Multiple Inheritance.

Java does not support multiple inheritance. But multiple inheritance can be achieved by using interfaces.

Program 2: Write a program to illustrate how to achieve multiple inheritance using multiple interfaces.

```
//interface Demo
interface Father
{ double  PROPERTY  =  10000;
      double HEIGHT = 5.6;
}
interface Mother
{ double  PROPERTY  =  30000;
      double HEIGHT = 5.4;
}
class MyClass implements Father, Mother
{ void show()
      { System.out.println("Total property is :" +(Father.PROPERTY+Mother.PROPERTY));
         System.out.println ("Average height is :" + (Father.HEIGHT + Mother.HEIGHT)/2 );
      }
}
class InterfaceDemo
{ public static void main(String args[])
      { MyClass  ob1  =  new  MyClass();
            ob1.show();
      }
}
```

Output:


```
D:\JQR>javac InterfaceDemo.java

D:\JQR>java  InterfaceDemo
Total property is :40000.0
Average height is :5.5

D:\JQR>
```

13. Packages

A package is a container of classes and interfaces. A package represents a directory that contains related group of classes and interfaces. For example, when we write statemens like:

import java.io.*;

Here we are importing classes of java.io package. Here, java is a directory name and io is another sub directory within it. The '*' represents all the classes and interfaces of that io sub directory. We can create our own packages called user-defined packages or extend the available packages. User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages. Packages provide reusability.

General form for creating a package:

package packagename;

e.g.: package pack;

· The first statement in the program must be package statement while creating a package.
· While creating a package except instance variables, declare all the members and the class itself as public then only the public members are available outside the package to other programs.

Program 1: Write a program to create a package pack with Addition class. //creating a package

```
package pack; public
class Addition
{ private double d1,d2;
      public Addition(double a,double b)
      { d1 = a; d2
            = b;
      }
      public void sum()
      {     System.out.println ("Sum of two given numbers is : " + (d1+d2) );
      }
}
```

Compiling the above program:


```
C:\WINDOWS\system32\cmd.exe                                    _ □ ×
D:\JQR>javac -d . Addition.java
D:\JQR>
```

The –d option tells the Java compiler to create a separate directory and place the .class file in that directory (package). The (.) dot after –d indicates that the package should be created in the current directory. So, out package pack with Addition class is ready.

Program 2: Write a program to use the Addition class of package pack. //Using the package pack

```
import pack.Addition;
class Use
```

55

```
{ public static void main(String args[])
      { Addition  ob1 =  new  Addition(10,20);
            ob1.sum();
      }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe                              _ □ ×

D:\JQR>javac Use.java

D:\JQR>java  Use
Sum of two given numbers is : 30.0

D:\JQR>
```

Program 3: Write a program to add one more class Subtraction to the same
package pack. //Adding one more class to package pack:
package pack;
public class Subtraction
{ private double d1,d2;
      public   Subtraction(double   a,
      double b) { d1 = a;
            d2 = b;
      }
      public void difference()
      {   System.out.println ("Sum of two given numbers is : " + (d1 - d2));
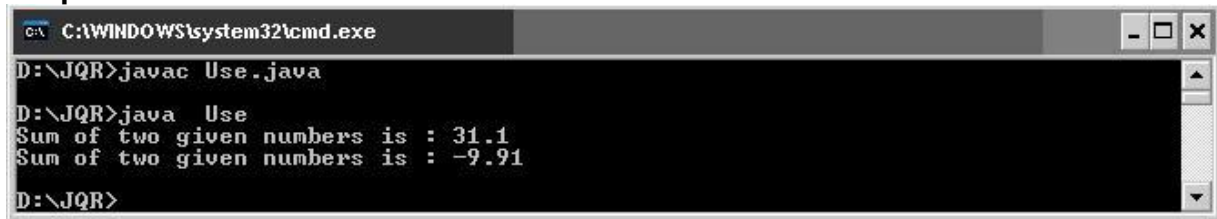   }
}

Compiling the above program:

```
C:\WINDOWS\system32\cmd.exe                              _ □ ×

D:\JQR>javac -d . Subtraction.java

D:\JQR>
```

Program 4: Write a program to access all the classes in the
package pack. //To import all the classes and interfaces in a class
using import pack.*; import pack.*;
class Use
{ public static void main(String args[])
      { Addition ob1 = new Addition(10.5,20.6);
            ob1.sum();
            Subtraction   ob2   =   new
            Subtraction(30.2,40.11); ob2.difference();
      }
}
In this case, please be sure that any of the Addition.java and Subtraction.java programs
will not exist in the current directory. Delete them from the current directory as they cause
confusion for the Java compiler. The compiler looks for byte code in Addition.java and
Subtraction.java files and there it gets no byte code and hence it flags some errors.

Output:

```
C:\WINDOWS\system32\cmd.exe                                    - □ ×

D:\JQR>javac Use.java

D:\JQR>java  Use
Sum of two given numbers is : 31.1
Sum of two given numbers is : -9.91

D:\JQR>
```

If the package pack is available in different directory, in that case the compiler should be given information regarding the package location by mentioning the directory name of the package in the classpath. The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import. If our package exists in e:\sub then we need to set class path as follows:

```
C:\WINDOWS\system32\cmd.exe                                    - □ ×

D:\JQR>set CLASSPATH=e:\sub;.;%CLASSPATH%
```

We are setting the classpath to **e:\sub** directory and current directory (.) and %CLASSPATH% means retain the already available classpath as it is.

Creating Sub package in a package: We can create sub package in a package in
the format: package packagename.subpackagename;
e.g.: package pack1.pack2;
Here, we are creating pack2 subpackage which is created inside pack1 package. To use the classes and interfaces of pack2, we can write import statement as:
import pack1.pack2;

Program 5: Program to show how to create a subpackage in a
package. //Creating a subpackage in a package
package pack1.pack2;
public class Sample
{ public void show ()
        {
                System.out.println ("Hello Java Learners");
        }
}
Compiling the above program:

```
C:\WINDOWS\system32\cmd.exe                                    - □ ×

D:\JQR>javac -d . Sample.java

D:\JQR>
```

Access Specifier: Specifies the scope of the data members, class and methods.
· private members of the class are available with in the class only. The scope of private members of the class is "CLASS SCOPE".

57