**B.Sc (Computer Science)**
**Programming in Java**
**Unit-I**

## 1. Explain the main concepts of Object Oriented Programming.

OOPS stands for Object Oriented Programming System. It is the most popular approach followed by programming languages. Java is one such language which follows OOPS.

OOP treat data as a critical element in the programming development and does not allow it to flow freely around the system. It ties data more closely to the functions. OOP allows decomposition of a problem into a number of entities called objects and build data and functions around these objects. The data of an object can be accessed only by the functions of one object can access the functions of other objects. OOP allows us to decompose a problem into a number of entities called "Objects" and then build data and methods.

Some of the important features of Object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- Follows bottom-up approach in program design.

It supports the following concepts:

**Objects:** Objects are the basic runtime entities in an object-oriented system. They may represent a person, a place, a bank account or any thing that the program can handle.

**Classes:** A class is a collection of objects of similar type. The entire set of data and code of an object can be made a user-defined data type using the concept of a class. A class may be thought of as a data type and an object as a variable of that type. Once a class has been defined, we can create any number of objects belonging to that class.

**Data Abstraction:** Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction such as size, weight and cost and the methods that operate on these attributes. It is a mechanism used to create new data types that include several related operations to be performed on it and attributes to suit the requirements of an application.

**Data Encapsulation:** Encapsulation means "combining". This feature allows combining several related properties (attributes) and functions (methods) into a single unit as its members. The wrapping up of data and methods into a single unit (class) is known as "Encapsulation". The members of that unit are accessible to its instances depending up on their accessibility conditions. The data is not accessible to

the outside world and only those methods, which are wrapped in the class, can access it.

**Inheritance:** This feature is used to derive the properties of an existing object. Inheritance is the process by which objects of one class acquire the properties of objects of another class.

**Polymorphism:** It means "many forms". Different behavior of functions or operators or objects at different situations can be called as "polymorphism". Polymorphism can be exhibited in three different ways: Function Overloading, Operator overloading and Dynamic binding.

## 2. List the benefits of Java/OOP.

**Benefits of OOP:**
Object-Oriented Programming contributes to the solution of many problems associated with the development and quality of software products.

The advantages of OOP are:

- Through inheritance we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch.
- Data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is easy to map objects in the problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture details of a model in implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

## 3. Explain the features of Java.

Java has many advanced features when compared to other programming languages.

- Compiled and Interpreted
- Platform-Independent and Portable
- Object-Oriented
- Robust and Secure
- Distributed
- Familiar, Simple and Small
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

**Compiled and Interpreted:** First, Java compiler translates source code into *bytecode* instructions. Bytecodes are not machine instructions and therefore, in the

second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language.

**Platform-Independent and Portable:** Java programs can be easily moved from one computer system to another, *anywhere* and *anytime.* Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet.

**Object-Oriented:** Java is a true object-oriented language. Almost everything in Java is an *object.* All program code and data reside within objects and classes. Java comes with an extensive set of *classes,* arranged in *packages,* that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

**Robust and Secure:** Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types and errors. It provides automatic garbage-collection. Java also allows exception handling to catch runtime errors.

Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

**Distributed:** Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet. This enables multiple programmers at remote locations to collaborate and work together on a single project.

**Simple, Small and Familiar:** Java is a small and simple language. Familiarity is another important feature of Java. To make the language look familiar to the existing programmers, it was modeled on C and C++ languages. Java uses many constructs of C and C++ and therefore, Java code "looks like a C++" code. In fact, Java is a simplified version of C++.

**Multithreaded and Interactive:** Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs; This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a distant computer.

**High Performance:** Java architecture is designed to reduce overheads during runtime. Further, the incorporation of multithreading enhances the overall execution speed of Java programs.

**Dynamic and Extensible:** Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects. Java programs support functions written in other languages such as C and C++. These functions are known as *native methods.* This facility enables the programmers to use the efficient functions available in these languages.

## 4. Explain main method in Java?

Let us consider the sample Java code:

```
class SampleOne
{
  public static void main (String args[ ])
  {
    System.out.println ("Java is better than C++.");
  }
}
```

### Class Declaration

The first line

```
class SampleOne
```

declares a class, which is an object-oriented construct. As stated earlier, Java is a true object-oriented language and therefore, ***everything*** must be placed inside a class. **class** is a keyword and declares that a new class definition follows. **SampleOne** is a Java *identifier* that specifies the name of the class to be defined.

### Opening Brace

Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace "}", appearing in the last line in the example. This is similar to C++ class construct. (*Note that a class definition in C++ ends with a semicolon.*)

### The Main Line

The third line

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

defines a method named **main**. Conceptually, this is similar to the **main( )** function in C/C++. Every Java application program must include the **main( )** method. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but ***only one*** of them must include a **main** method to initiate the execution.

This line contains a number of keywords, **public, static** and **void**.

**Public:** The keyword **public** is an access specifier that declares the **main** method as unprotected and therefore making it accessible to all other classes. This is similar to the C++ **public** modifier.

**Static:** Next appears the keyword **static**, which declares this method as one that belongs to the entire class and not a part of any objects of the class. The **main** must always be declared as **static** since the interpreter uses this method before any objects are created.

**Void:** The tyep modifier **void** states that the **main** method does not return any value (but simply prints some text to the scree.)

All parameters to a method are declared inside a pair of parentheses. Here, **String args[ ]** declares a parameter named **args**, which contains an array of objects of the class type **String**.

**The Output Line**

The only executable statement in the program is

```
System.out.println ("Java is better than C++.");
```

This is similar to the **printf( )** statement of C or **cout <<** construct of C++. Since Java is a true object oriented language, every method must be part of an object. The **println** method is a member of the **out** object, which is a static data member of **System** class.

This line print the string
Java is better than C++.
to the screen. The method **println** always appends a newline character to the end of string. This means that any subsequent output will start on a new line. Note the semicolon at the end of the statement. *Every Java statement must end with a semicolon.*

## 5. Explain the structure of Java Program.

**Java Program Structure:** A Java program may contain many classes of which only one class defines a main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements. To write a Java program, we first define classes and then put them together. A Java program may contain one or more sections as shown below:

| | |
|---|---|
| Documentation Section | Suggested |
| Package Statement | Optional |
| Import Statements | Optional |
| Interface Statements | Optional |
| Class Definitions | Optional |
| main Method Class<br><br>{<br><br>Main Method Definition<br><br>} | Essential |

**Documentation Section:** The documentation section comprises a set of comment lines giving the name of the program, the author and other details. Comments explain *why* and *what* of classes and *how* of algorithms. This would greatly help in maintaining the program. Java uses a new style of comment /**....*/ known as *documentation comment.* This form of comment is used for generating documentation automatically.

**Package Statement:** The first statement allowed in a Java file is a *package* statement. This statement declares a *package* name and informs the compiler that the classes defined here belong to this package. Example:

*package student;*

The package statement is optional. That is, our classes do not have to be part of package.

**Import Statements:** The next thing after a package statement (but before any class definitions) may be a number **of** *import* statements. This is similar to the *include* statement in C. Example:

*import student.test;*

This statement instructs the interpreter to load the *test* class contained in the package *student***.** Using import statements, we can have access to classes that are part of other named packages.

**Interface Statements:** An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance features in the program.

**Class Definitions:** A Java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.

**Main Method Class:** Since every Java stand-alone program requires a **main** method as its starting point, this class is the essential part of a Java program. A simple Java program may contain only this part. The **main** method creates objects of various classes and establishes communications between them. On reaching the end **of main,** the program gets terminated.

## 6. How to implement a Java Program?

Implementation of a Java application program involves a series of steps. They include:

1. Creating the program
2. Compiling the program
3. Running the program

**Creating the Program:** We can create a program using any text editor like notepad. But, we need to make sure that the program name should be saved with the extension ".java". Advisably, the name of the program should match with the name of the public class or a class that has main() method.

**Example:** assumed that the program name is "Test.java"

```
class Test
{
```

```
public static void main(String as[])
{
System.out.println("Java Program");
}
}
```

**Compiling the program:** Java program involves two steps for its execution: one compilation and the other execution. We use "javac" tool to compile java program. This yields byte code file whose extension is ".class". This is an intermediate file and not directly executable.

**Syntax:** *javac Test.java*

**Running the Program:** Now, java program is interpreted by the tool "java" which as an interpreter. This is different for different machines. This tool generates executable code for the machine in use. Thus one can view the output of the java program.

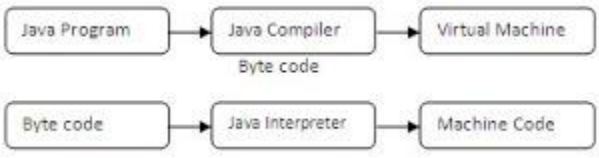
**Syntax:** *java Test*

The complete steps to run a java program are listed below:

1. Type the program in the DOS editor or notepad. Save the file with a .java extension.
2. The file name should be the same (advisably) as the class, which has the main method.
3. To compile the program, using javac compiler, type the following on the command line: *Example:* javac Test.java
4. After compilation, run the program using the Java interpreter.
   *Example:* java Test
5. The program output will be displayed on the command line.

## 7. Write short notes on JVM.

All language compilers translate source code into *machine code* for a specific computer. But, Java compiler produces an intermedia code known as *bytecode* for a machine that does not exist. This machine is called the *Java Virtual Machine* and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer




The virtual machine code is not machine specific. The machine specific code is generated by the Java Interpreter by acting as an intermediary between the virtual machine and the real machine. The Java API acts as the intermediary between the operating system and the Java Framework. The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary

between the virtual machine and the real machine. The interpreter is different for different machines.

## 8. What are Command-Line Arguments? How are they used in Java? (Oct 2011) (Mar 2012)

Command line arguments are the parameters passed to the executable java program in the form of strings. Command line arguments are collected in main function itself. The argument is an array of strings and can collect the list of strings. File name is not included in the list of arguments as in C language. This is very useful to receive quick inputs.

```
class Test
{
public static void main(String args[])
{
System.out.println("Number of arguments="+as.length);
for(int i=0;i<as.length;i++)</as.length;i++)
System.out.println(as[i]);
}
}
```

Execute the above program as follows.

*java Test 1 2 3*

The above command produces the following output:

*Number of arguments=3*

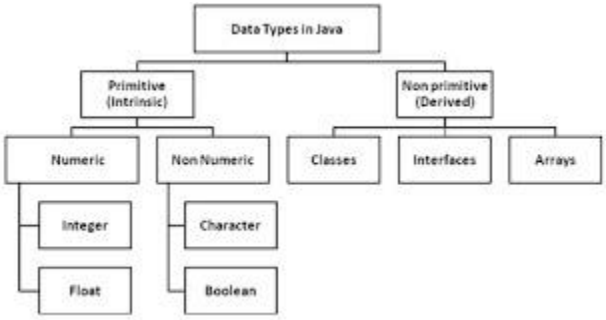
*1*

*2*

*3*

## 9. What are the different data types in Java?

**Data-type:** Data-type specifies what type of value a variable can store. It also specifies its size. Every variable in Java has a data-type. Java language is rich in its data types. The categories of different data-types are shown below. The two main types are Primitive and non-primitive types.




**1. Integer type:** An integer type of variables can hold integer constants. Java supports four types of integer data types. They are *byte, short, int* and *long.* The following table shows size in bytes and range of values each data type can represent. Integer can be converted to long type by typing 'L' after them.

| TYPE | MEMORY (bytes) | VALUES |
|---|---|---|
| byte | 1 | -128 to 127 |
| short | 2 | -32768 to 32767 |
| int | 4 | $-2^{31}$ to $2^{31}-1$ |
| long | 8 | $-2^{63}$ to $2^{63}-1$ |

**2. Floating point type:** This data type is used to represent numbers containing fractional values. They are two types of data-types under this category, namely, float and double. float type number are single precision numbers while double type number are double precision numbers. By default float are double precision and can converted to single precision by adding 'f' at the end. Double precision types are used when greater precision is required in storage of floating point. Floating point data types have special NaN (Not a Number) value for undefined numbers like 0/0 etc.

| TYPE | SIZE | MAXIMUM VALUE | MINIMUM VALUE |
|---|---|---|---|
| float | 4 bytes | 3.4e-038 | 3.4e+038 |
| double | 8 bytes | 1.7e-308 | 1.7e+308 |

**3. Character type:** Java provides a data type called char to hold character values. It is of size 2 bytes and assumes single character. The character value must be enclosed in single quotes.

**4. Boolean type:** It can have only two values, either 'true' or 'false'. Its keyword is 'boolean' and use only 1 bit. All comparison operators return boolean type values. The words 'true' and 'false' cannot be used as identifiers.

## 10. Explain Type Casting.

Type casting is a process to convert one data type to another. It makes the variable compatible temporarily. We often encounter situations where there is a need to store a value of one type into a variable of another type. In such situations, we must cast the value to be stored by preceding it with the type name in parentheses. Casting is often necessary when a method returns a type different than the one we require.

Four integer types can be cast to any other type except boolean. Casting into a smaller type may result in a loss of data. Similarly, the **float and double** can be cast to any other type except boolean. Again, casting to smaller type can result in a loss of data. Casting a floating point value to an integer will result in a loss of the fractional part. If the following order is used for type casting, it guarantees in no loss of information.

*byte → char → short → int → long → float → double*

for example

- *byte* value can be converted to *int* or *double*
- *int* value can be converted to *long* or *float*

The syntax for casting is: *type variable1 = (type) variable2;*

Type casting can be of two types:

**Implicit type Casting (Automatic Conversion):** when constants and variables of different types are mixed in an expression they are converted to the same type. This conversion is done implicitly by the C compiler. The C compiler converts all the operands to the type of the largest operand. For example if an expression involves int and float, int type gets converted to float type.

**Example:**

```
float x=32;
```

The value of the x will be '32.0', because of implicit conversion of value from int to float

**Explicit type Casting:** if a user forcefully changes the data type into other allowable data type it is said to be explicit type casting.

**Example:**

*int to float:*

```
float x;
x=5/2 /* value of x will be 2.0 */
x=(float)5/2; /* value of x will be 2.5 */
```

*float to int:*

```
int x;
```

```
x= 3.2 /* causes error */
x=(int)3.2 /* x will be 3 */
```

## 11. Explain Conditional operator.

The character pair ? : is a ternary operator available in Java. This operator is used to construct conditional expressions of the form:

Exp1? Exp2 : Exp3

Where Exp1, Exp2 and Exp3 are expressions. Exp1 is evaluated first. If it true, then Exp2 is evaluated otherwise Exp3 is evaluated. Only one expression among Exp2 and Exp3 is evaluated.

## 12 . Explain different forms of if –statements.

**Conditional control structure (Branching):**

a) **if:** If structure is also called as conditional statement. In If, statements get executed only when the condition is true. It omits the condition based statements when the condition is false. Braces are not necessary if only one statement is related to the condition.

```
if (condition)  {

statements

}
```

b) **if else:** Statements in if block get executed only when the condition is true and statements in else block get executed only when the condition is false.

```
if (condition)  {

Statements

}

else {

Statements

}
```

**(c). if-else if**:

Statements get executed in loops only when the corresponding conditions are true. Statements in the final else block get executed when all other conditions

are false. The control checks a condition only when all the afore-mentioned conditions are false.

```
if ( x > 0 )

System.out.println( "positive");

else if ( x < 0)

System.out.println("negative ");

else

System.out.println( "zero");
```

**(d). nested if:** Writing if in another if is nested-if. Inner if is processed only when outer if's condition is true. Hence, statements in inner-if get executed only when condition1 and condition2 are true.

## 13. In what way switch statement differs from if statement? (Mar 2011)

**if statement**: if statement is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression.

It allows the computer to evaluate the expression first and then depending on whether the value of the expression is true or false. It transfers the control to a particular statement. This point of program has two paths to follow, one for the true condition and the other for the false condition.

The if statement is implemented in four forms:

1. Simple if statement
2. if…else statement
3. Nested if…else statement
4. else if ladder

**switch statement**: The complexity of a program increases when the number of alternatives increases in if statement. The program becomes difficult to read and follow. It may confuse even the designer.

Java has a built-in multi way decision statement known as switch. The switch statement tests the value of a given variable against a list of case values and when a match is found, a block of statements associated with that case is executed.

The general form of the switch statement is as below:

```
switch(expression)
{
 case value1:  stat1;
               break;

 case value2:  stat2;
               break;
```

```
            .
            .
            .
            .
            .
  case valuen:  statn;
                break;
  default: defaultstat;
}
```

The expression is an integer or character expression. value1,value2,…valuen are know as case labels. There is no need to put braces around these blocks but it is important to note that case labels end with a colon.

When the switch statement is executed, the value of the expression is successively compared against the values value1,value2,…valuen. If a case is found whose value matches with the value of the expression, then the block of the statements that follow the case are executed. The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement. The default is an optional case. When preset, it will be executed if the value of the expression does not match with any of the case values.

**14. Write a program to find the number of and sum of all integers greater than 100 and less than 200 that are divisible bv 7.**

```
class div7
{
        public static void main(String[] args)
        {
                int i=0,sum=0;
                for(int n=101;n<200;n++)
                {
                        if(n%7==0)
                        {
                                sum=sum+n;
                                i=++i;
                                System.out.println(n);
                        }
                }
System.out.println("No of integers divisible by 7 which are >100 and <200 are:" + i);

System.out.println("Sum of integers divisible by 7 which are >100 and <200 are:" + sum);
        }
}
```

**15. Compare while and do...while.**

**While**: The while is an entry-controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it true, the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop.

The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However, it is a good practice to use if the body has only statement.

```
while(condition)
{

Body of the loop;


}
```

**do...while**: The while makes a test condition before the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the do…while statement.

```
do
{

Body of the loop;

}while(condition);
```

On reaching the do statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement. Since the test condition is evaluated at the bottom of the loop, the do…while construct provides an exit-controlled loop and therefore the body of the loop is always executed atleast once.

**16. Explain for statement. (Mar 2010)**

The for loop is entry-controlled loop that provides a more concise loop control structure. The general form of for loop is:

```
for(initialization; test condition;increment/decrement)
{

Body of the loop;

}
```

The execution of for statement is as follows:

1. Initialization of the control variables is done first, using assignment statements such as i=1 and count=0. These variables are known as loop-control variables.
2. The value of the control variable is tested using the test condition. The test condition is a relational expression such as i<10 that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the

loop is terminated and the execution continues with the statement that immediately follows the loop.

3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented/decremented using an assignment statement such as i=i+1 and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

## 17. Differentiate between break and continue.

**break**: an early exit from a loop can be accomplished by using the break statement. When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

```
class PrimeNumber
{
    public static void main (String args[])
      {
        int n=10,i;
        i=2;
       while(i<n)
       {
        if(n%i= =0)
       {
        system.out.println("Number is not prime");
           break;
         }
        i++;
      }
     if(n==i)
       System.out.println("Given no is prime");
    }
}
```

**continue**: Java supports similar statement called the continue statement. However, unlike the break which causes the loop to be terminated, the continue as the name implies causes the loop to be continued with the next iteration after skipping any statements in between.

```
class Demo
       {
              public static void main(String args[])
              {
                     int i;
                   for(i=1;i<=10;i++)
                   {
                  if(i<5)
                 continue;
                 System.out.println(i);
                       }
               }
}
```

In the above program we are redirecting the flow of execution back to the next iteration of the loop till i<5. When i value changes from 1 to 4(i=1,2,3,4) continue

statement will be executed and System.out.println is not executed. Whenever i value will become 5 i values will be displayed.

**18. Write a program using while loop to display reverse the digits of the given number.**

```
import java.io.*;
class reverse
{
	public static void main(String[] args)  throws IOException
	{
		int i,k;
		System.out.println("Enter a number to reverse");
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		String s=br.readLine();
		i=Integer.parseInt(s);
		System.out.print("Reversed Number:");
		while(i>0)
		{
			k=i%10;
			System.out.print(k);
			i=i/10;
		}

	}
}
```

**19. Write a program to compute the sum of the digits of a given number.**

```
import java.io.*;
class sum
{
	public static void main(String[] args)  throws IOException
	{
		int i,k,sum=0;
		System.out.println("Enter a number");
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		String s=br.readLine();
		i=Integer.parseInt(s);
		while(i>0)
		{
			sum=sum+i%10;
			i=i/10;
		}
		System.out.println("Sum of the digits:" + sum);
		}
}
```

**20. Distinguish between a class and an object. (Oct 2012)**

**Class:** A class is a user-defined data type with a template that serves to define its properties. Once the class type has been defined, we can create variables of that type using declarations that are similar to the basic type declaration. These variables are termed as instance of classes.

The basic form a class definition is:

```
class classname [extends superclass]
{
```

```
        [fields declaration]
        [methods declaration]
}
```

The keyword extends indicates that the properties of the superclass are extended to the subclass.

**Object:** An instance of a class is called Object.

Objects in Java are created using **new** operator. The new operator creates an object of the specified class and returns a reference to that object.

```
Rectangle r;
R=new Rectangle();
```

Both statements can be combined into one as below:

```
Rectangle r=new Rectangle();
```

## 21. How is a method defined?

A class with is a Java function. The general form of method is:

```
type methodname(parameter list)
{
        Body of the method
}
```

Method declaration have four basic parts:

- The name of the method
- The type of the value the method returns
- A list of parameters
- The body of the method.

The type specifies the type of the value the method would return. This could be a simple data type such as int, float,char and so on. It could be even void, if the method doesn't return any value. The parameter list is always enclosed in parenthesis. This list contains variable names and types of all values which are given as input. In case where no input data is required, the declaration must retain empty parenthesis. The body actually describes the operations to be performed.

```
class Rectangle
{
  int length;
  int width;

  void getdata(int x,int y)
  {
    length=x;
    width=y;
  }
}
```

## 22. What is a Constructor?

Java supports a special type of method called **Constructor** that enables an object to initialize itself when it is created.

```
class Rectangle
{
  int length;
  int width;

  Rectangle(int x,int y)
  {
   length=x;
   width=y;
  }

}
```

**Special Characters**:

- Constructors have the same name as the class itself.
- They don't have any return type, not even void. This is because they return the instance of the class.

## 23. What is the need for copy Constructor?

**Copy Constructor** is a constructor that takes the object of same class as argument. If all the properties of an object which has to be assigned to another object, this is advisable.

```
Box b1=new Box(5);
Box b2=new Box(b1);
```

## 24. Explain about inheritance in Java.

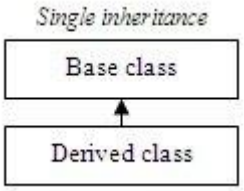
Java classes can be reused in several ways. This is basically done by creating new classes, reusing the properties of existing ones. The mechanism of deriving a new class from old one is called inheritance. The old class is known as **base class** or **super class** or **parent class** and the new one is called the subclass or **derived class** or **child class**.

The inheritance allows subclasses to inherit all the variables and the methods of their parent classes.

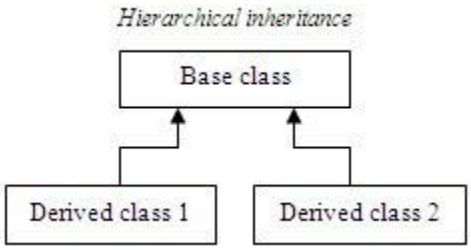
Inheritance may take different forms:

- Single inheritance (only one super class)
- Multiple inheritance (several super class)
- Hierarchical inheritance (one super class, many subclasses)
- Multilevel inheritance (derived from a derived class)

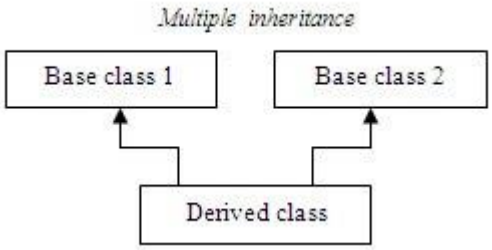
**Single inheritance:** If a class is derived from another class, it can be called as *single* inheritance.



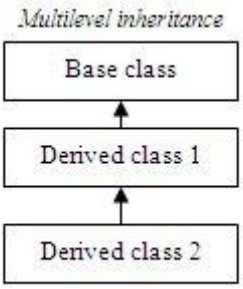

**Hierarchical inheritance:** If two or more classes are derived from a class, it can be called as *hierarchical* inheritance.




**Multiple inheritance:**If a class is derived from two or more classes, it can be called as *multiple* inheritance. Java does not support this type of inheritance.




**Multilevel inheritance: (Oct 2012)** If a class is derived from the class, which is derived from another class; it can be called as *multilevel* inheritance.




**Defining a subclass**: A subclass is defined as follows:

```
class subclass extends superclass
{
        Variable declarations;
        Methods declarations;
}
```

The keyword extends signifies that the properties of the superclass are extended to the subclass. The subclass will now contain its own variables and methods as well those of the superclass. This kind of situation occurs when we want to add some more properties.

## 25. Difference between Overloading and Overriding methods. (Mar 2010)

**Method overloading:-**

Methods are created with the same name but different parameter list and different definitions. This is called **method overloading.**

- Method overloading is used when objects are required to perform similar task but using different. Input parameters.
- When we call a method in an object, java matches up the method name first and then the number of parameters and then type of parameters to decide which one of the definitions to execute. This process is known as polymorphism.
- In the below example, constructor overloading is used.

```
class Room
{
float length;
float breadth;
Room(float x,float y)
{
length=x;
breadth=y;
}
Room(float x)
{
length=breadth=x;
}
Int area()
{
Return length*breadth;
}
```

**Method overriding:-**

A method defined in a super class can be inherited by its subclass and is used by the objects created by its subclass. Method inheritance enables us to define and use methods repeatedly in subclasses without having to define the methods again in the subclass. However, there may be occasions when an object to respond to the same method but have different behavior when that method is called. That means, the super-class method should be override. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a

method in the super-class. When that method is called, the method defined in the subclass is invoked and executed instead of the one in the super-class. This is known as overriding.

```
class A
{
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println("i and j: " + i + " " + j);
}
}

class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
System.out.println("k: " + k);
}
}

class OverrideMethod {
public static void main(String args[]) {
B b = new B(1, 2, 3);
b.show();
A a=new  A(10,20);
a.show();
}
}
```

## 26. When do we declare a method or class as final? (Mar 2011)

All methods and variables can be overridden by default in subclasses. A class that cannot be sub classed is called final class. This is achieved in Java using the keyword final as follows:

```
final class A
{

Body of the class

}
```

Making a method final ensures that the functionality defined in the method will never be altered in any way. Similarly, the value of a final variable can never be changed.

## 27. When do we declare a method or class as abstract?

Java allows the programmer to override a method compulsory. This can be done using the keyword abstract in the method defined.

```
abstract class Shape
{
……..
……..
abstract void draw();
……
……..
}
```

When a class contains atleast one abstract method, it should be declared as abstract. When using the abstract classes, we must satisfy the following conditions:

- We cannot use abstract classes to instantiate objects directly. For example Shape s=new Shape() is illegal because Shape is an abstract class.
- The abstract methods of an abstract class must be defined in its subclass.
- We cannot declare abstract constructors or abstract static methods.

## 28. Define Array.

An array represents a group of elements of same data type. It can store a group of elements.

In java arrays are created on dynamic memory i.e. allocated at runtime by JVM.

Arrays are generally categorized into two parts as follows:

- Single dimensional arrays.
- Multi dimensional arrays.

**Single dimensional arrays:-**

- A list of items can be given one variable name using only one subscript. Such a variable is called single dimensional array.
- Arrays must be declared and created in the computer memory before they are used
- Creation of array involves 3 steps:
  1) Declaring the array
  2) Creating memory locations
  3) Initialization of values into the memory locations.

**Declaring the array:-**

Arrays in java may be declared in two forms:

```
type arrayname[ ];
        (or)
type[ ] arrayname[ ];
```

Ex:-

```
int counter[ ];
int[ ] counter;
```

**Creating memory location to array:-**

After declaration of arrays, memory locations are created by new operator.

arrayname = new type[size];

Ex: counter =new int[5];

These lines create necessary memory locations for the array counter. It is also possible to combine the two steps declaration and creation of memory location as follows:

int number[ ] = new int[5];

**Initialization of arrays:-**

- Putting values into the array is known a "initialization". This is done as follows:

arrayname[subscript]=value;

Ex:-

counter[0]=35;

- Java creates arrays starting with the subscript of 0 and ends with a value one less than the size specified.
- Arrays can also be initialized automatically when they are declared, as shown below:
  Syntax:- type arrayname[ ]={list of values};

  Ex:-

  int number[ ]={5,4,1,8,9};

  In the above example array size is not given, the compiler allocates enough space for all elements specified in the list.

**Multi-dimensional arrays:-**

Multidimensional arrays can represent the data in the form of rows and columns i.e. by taking two or more indexes.

Multidimensional arrays include double dimensional and 3 dimensional arrays.

**2-Dimensional Arrays:** 2-Dimensional Arrays can represent the data in the form of rows and columns i.e. by taking two indexes. Generally, these are used to work with matrices.

In java, a double dimensional array can be declared as follows:

int a[][] = new int[5][5];

**Advantages of arrays:**

- It is capable of storing many elements at a time
- It allows random accessing of elements i.e. any element of the array can be randomly accessed using indexes.

**Disadvantages:**

- Predetermining the size of the array is a must.
- There is a chance of memory wastage or shortage.
- To delete one element in the array, we need to traverse throughout the array.

**Matrix Multiplication(Two-Dimensional Array):**

```
import java.io.*;
class matrixmult
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int m1[][]=new int[3][3];
int m2[][]=new int[3][3];
int m3[][]=new int[3][3];
int i,j,k;
System.out.println("Enter elements of first matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
m1[i][j]=Integer.parseInt(br.readLine());
}
}

System.out.println("Enter elements of second matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
m2[i][j]=Integer.parseInt(br.readLine());
}
}


System.out.println(" elements of first matrix are ....");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m1[i][j]);
}
System.out.println();
}

System.out.println("Elements of second matrix ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m2[i][j]);
}
System.out.println();
}
```

```
/*Matrix multiplication*/
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
m3[i][j]=0;
for(k=0;k<3;k++)
{
//m3[i][j]+=m1[i][k]*m2[k][j];
m3[i][j]=m3[i][j]+m1[i][k]*m2[k][j];
}
}
}


System.out.println("matrix multiplication result is ");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
System.out.print(m3[i][j] +" ");
}
System.out.println();
}
}
}

class VectorTest
{
public static void main(String as[])
{
Vector v1=new Vector(10);
v1.add("10");
v1.add("20");
v1.add("30");
System.out.println("Number of Elements:"+v.size());
System.out.println("Vector Elements:"+v1);
}
}
```

## 29. Define Interfaces.

**Interfaces:** Java provides an approach known as interfaces to support the concept of multiple Inheritance. An interface contains methods and variables. All methods of the interface are public and abstract. Interfaces do not specify any code to implement these methods. Therefore it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

The general form of an interface definition is as follows:

```
interface InterfaceName
{
Variables declaration;
Methods declaration;
```

```
}
```

An interface variable is public, static and final by default. This means all the variables of the interface are constants. Methods declaration will contain only a list of methods without anybody statements.

Interfaces can be implemented in two ways:

1. **Extending interfaces**

Interfaces can also be extended. The sub interface will inherit all the members of the super interface. This is achieved by using the keyword extends.

```
interface interfacename2 extends interfacename1
{
body of name2
}
```

Sub interfaces can not define the methods declared in the super interfaces. So class is
used to implement the derived interface, to define all the methods.

2. **Implementing interfaces:-**

Class must be defined to implement the code for the method of interface.

```
class classname implements interfacename
{
body of classname;
}
```

Here classname **"implements"** the interface interfacename. A class can implements more than one interface, separated by comma.

**Example:**

```
interface Area
{
final static double pi=3.14;
double compute (double x, double y);
}

class Rectangle implements Area
{
public double compute(double x, double y)
{
return(x*y);
}
}

class Circle implements Area
{
public double compute(double x, double y)
{
return(pi*x*x);
}
```

```
class InterfaceDemo
{
public static void main(String args[])
{
Rectangle r=new Rectangle();
System.out.println("Area of rectangle is"+r.compute(10,20));
Circle c=new Circle();
System.out.println("Area of circle is"+c.compute(5.1,0);
}
}
```

**30. What is the difference between an interface & a class?**

| Class | Interface |
|---|---|
| A class must be declared using the keyword **class** | An interface must be declared using the keyword interface |
| A class contains the methods which are defined. | An interface consists of methods which are declared. |
| A class can implement many interfaces | An interface can extend another interface |
| A class can simultaneously extend a class and implement multiple interfaces | An interface is that which can simulate multiple inheritance. |
| A class which implements an interface must implement all of the methods declared in the interface or be an abstract class. | An interface is implicitly abstract. It cannot be directly instantiated except when instantiated by a class . |
| Methods of a class can use any access specifier. | In an interface, all methods are implicitly public |
| Variables of a class can be defined according to the requirement | In an interface, all variables are static and final by default. |