

Unit - I

Fundamentals of object - Oriented programming:

1. Introduction to Java Programming Language:

- Java is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others.
- Java is an object oriented programming language from Sun Microsystems. James Gosling has developed Java language.
- We can use Java to write computer applications that play games, store data or do any of the thousands of other things computer software can do.
- Even though Java is used in many areas, programming community treats Java as a language for internet programming (Web Programming).
- Compared to other programming languages, Java is similar to C Programming language.
- Java was first released publicly in November 1995.

2. What is the 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.
- Java was invented for the development of software for consumer electronic devices like TVs, Toasters, and Refrigerators etc.
- 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.
- 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**.
- The main aim had to make java simple, portable and reliable.
- 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.

Following table shows the year and beginning of Java.

Year	Progress
1990	Sun decided to developed software that could be used for electronic devices. And the project called as Green Project head by James Gosling.
1991	Announcement of a new language named “Oak”
1992	The team verified the application of their new language to manage a list of home appliances using a hand held device.
1993	The World Wide Web appeared on the Internet and transformed the text-based interface to a graphical rich environment.
1994	The team developed a new Web browser called “Hot Java” to locate and run Applets.
1995	Oak was renamed to Java, as it did not survive “legal” registration. Many companies such as Netscape and Microsoft announced their support for Java. Version 1.0 of Java Development kit (JDK) was released for free by sun micro systems.
1996	Java language is now famous for Internet programming as well as a general purpose Object oriented language.
1997	Sun releases Java Development Kit(JDK 1.1)
1998	Sun releases Software Development Kit (SDK 1.2)
1999	Sun releases Java 2 platform Standard Edition (J2SE) and Enterprise Edition J2EE). This comes to three editions. J2SE – Java 2 standard edition J2ME- Java 2 Micro Edition J2MEE – Java 2 Enterprises Edition
2000	J2SE with SDK 1.3 was released.
2002	J2SE with SDK 1.4 was released.
2004	J2SE with JDK 5.0 was released. (Previously numbered 1.5)
2006	Java SE 6 (Previously numbered 1.6)
2011	Java SE 7 (Previously numbered 1.7)
2014	Java SE 8 (Previously numbered 1.8)

LIST OF PROGRAMMING PRINCIPLES

There are six types of programming principles. These are as follows.

- ❖ Unstructured programming (USP)
- ❖ Procedure oriented programming (POP)
- ❖ Modular (or) structured oriented programming (SOP)
- ❖ Object oriented programming (OOP)

Unstructured programming (USP): According to unstructured programming concept, program is developed by organizing data and instructions in sequential order.

Draw backs of unstructured programming.

- Redundant data, Because of redundancy the size of program increases.
- Occupy more space, which reduce speed.
- There is no proper organization of data and operations.

Ex: Assembly languages, Machine Language are USP Programming Languages.

Procedural oriented programming (POP)

Operations provided by a program are divided into small pieces and each piece of a program is called subroutine.

What is subroutine?

A small program within a program is called subroutine. The subroutine can be either procedure or function.

Ex: COBAL, and PASCAL are called procedural oriented languages.

Advantages of procedural oriented programming:

Modularity: It is a concept of developing an application in sub modules i.e. procedure oriented approach.

Reusability: Write once and use many times.

Simplicity: It is easy to understand operations of a program.

Efficiency: By reducing the size of a program efficiency of procedure oriented programs increases.

Characteristics of procedural oriented programming:

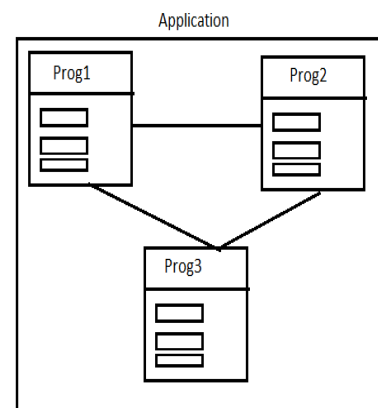
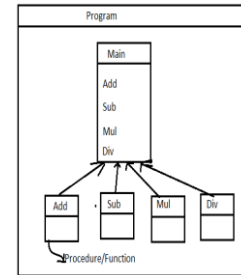
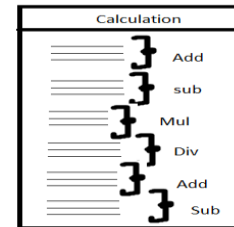
- Large programs are divided into small programs.
- Most of the functions share global data.
- Functions transform data from one another.

Drawbacks of procedural oriented programming:

- Concentrated on development of functions.
- In large program it is very difficult to identify what data is used by which function.
- Debugging application is complex.

Write about structured programming language. Give its advantages.

- ☞ Structured program is a top down approach in which the overall program structure is broken down into separate modules.
- ☞ It allows the code to be efficiently loaded into the memory and to be reused in other programs.
- ☞ Structure oriented program is subset of procedural oriented programming approach.
- ☞ In this approach reusability of subroutines are between the programs.
- ☞ C is a structured oriented (or) procedure oriented programming language.



Advantages:

- ☞ The goal of structured programming is to write correct programs that are easy to understand and modify.
- ☞ Modules enhance the programmer's productivity.
- ☞ With modules, many programmers can work on a single large program, with each working on different module.
- ☞ A structured program can be written in less time than an unstructured program.
- ☞ Modules or procedures written for one program can be reused in both programs as well.
- ☞ A structured program is easy to debug.
- ☞ Individual procedures are easy to change as well as understand.

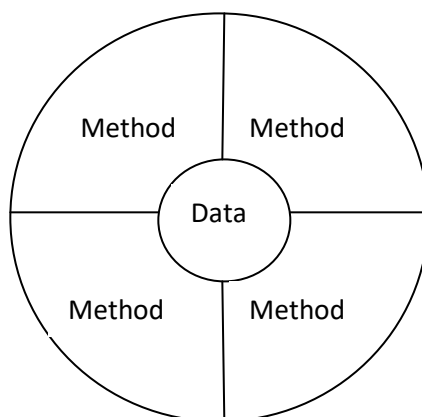
3. What is object oriented programming/Paradigm? What the features are of object oriented paradigm?

Programming paradigm: A **programming paradigm** is a fundamental style of computer programming.

There are four main paradigms:

1. Imperative programming paradigm
2. Functional Programming paradigm
3. Object-oriented programming paradigm
4. Logic programming paradigm

Object oriented programming: Object-oriented programming (**OOP**) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods.



Object = Data + Methods

- ☞ OOPs were invented to overcome the drawbacks of POPs.
- ☞ OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.

- ☞ Object oriented programming allows us to decompose a problem into a number of entities called objects and then builds a data and functions around these entities (objects).
- ☞ Everything is represented in the form of object in OOPs. Objects protects of its data and functions from outside objects.
- ☞ The data of an object can be accessed only by the functions associated with the object.
- ☞ OOP's offers several benefits to both the programming designer and the user.
- ☞ The new technology provides greater programmer productivity, better quality of software and lesser maintenance of cost.
- ☞ A structure once created can be reused this is the fundamental property of OOP's concept.

Features of object oriented programming:

- ☞ 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.
- ☞ Methods that operate on 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 methods.
- ☞ New data and methods can be easily added.
- ☞ Follow bottom-up approach in program design.

4. What is Object oriented programming language and explain the concepts.

OOP's concepts are the concepts or rules which are suppose to be satisfied by a programming language in order to call that programming language, as an object oriented programming language.

The OOP's concepts are

1. Class
2. Object
3. **Encapsulation**
4. **Inheritance**
5. Data abstraction
6. **Polymorphism**
7. Message Passing

OOP's concepts are only three and all others applications of the OOP's concepts. Encapsulation is known as the backbone of the OOP's concepts.

Class:

- A **class** is a group of objects that share common properties and relationships.
- A class is user-defined data type which holds both *data* and *member functions*

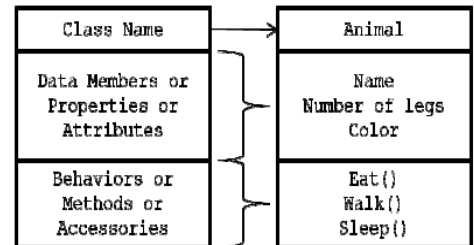
- Class is structure representing the data and its related and corresponding functions.
- For example Mango, Apple and Orange are objects of the class “Fruit”.

Fruit Mango;

Object:

- Object is the basic run time entity in a object oriented system.
- Object may represent a person, a place, a bank account or any item that the program may handle.
- Each object contains both data and code to manipulate the data.
- Objects are the variables of classes.

Class diagram for defining a class:
Example:



Encapsulation:

- The concept of binding the data along with its corresponding and related functionalities is known as encapsulation.
- The concept of encapsulation was invented in order to provide highest security for the data part of an application and make programming simple.

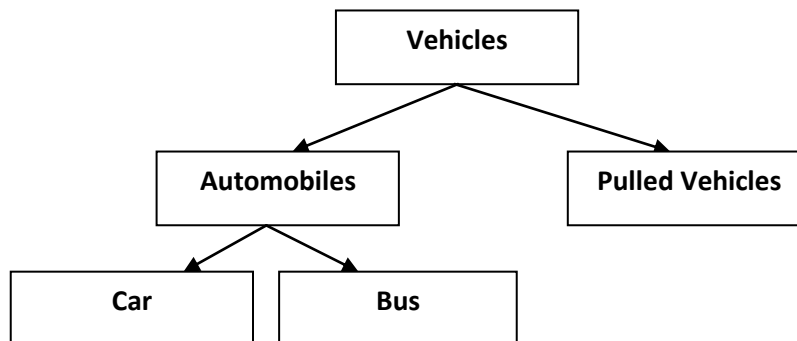
Example: Company – in company, the department access and work with their data (on their own). One department cannot access the data of other department directly

Inheritance:

- The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as base class and new one is called as derived class.

Example: Automobiles and pulled vehicles are sub class of vehicles, vehicles is the base class of automobiles and pulled vehicles.

Car and bus are sub classes of ‘automobiles’. ‘Automobiles’ is the base class of car and bus.



Data abstraction:

- It refers to the act of representing essential features without including the background details or explanation.

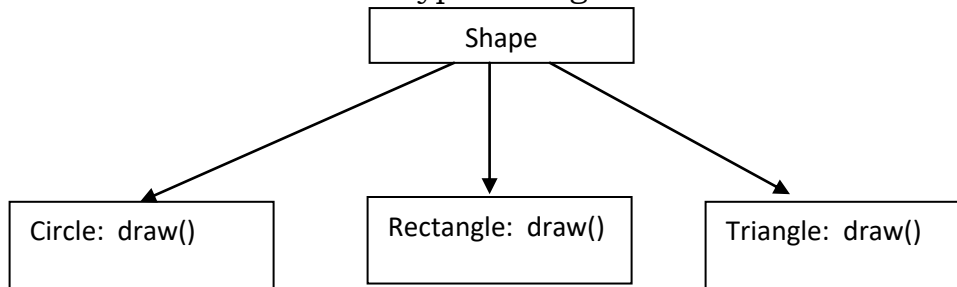
Example: Switch board- you only press certain switch according to your requirement. What is happening inside, how it is happening etc. you need to know, again this is abstraction. You only the essential things to operate on switch board without knowing the background details of the switchboard.

Polymorphism:

- (Poly means —many and morph means —form). Polymorphism means the ability to take more than one form. Polymorphism is broadly used in implementing inheritance.
- Polymorphism is the ability for a message or data to be processed in more than one form.

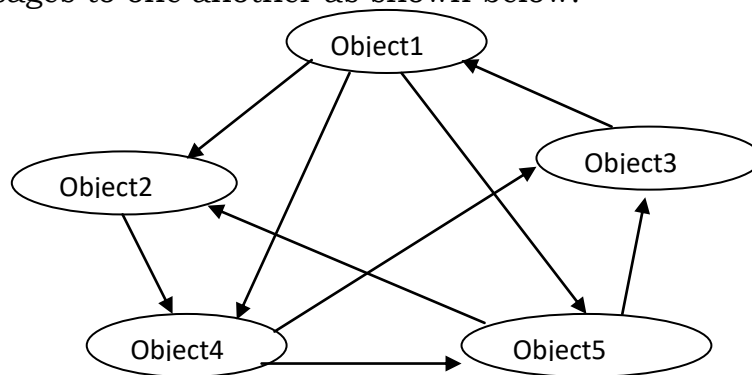
Example: If you give $5+7$, it results into 12, the sum of 5&7. If you give 'A+ BC' it results into 'ABC' the concatenated strings. The same operation '+' is able to distinguish between the two operations depending upon the type it is working on.

The following diagram shows that a single method name draw() can be used to handle different numbers and types of arguments.



Message Passing

An OOP consists of the set of objects that communicate with each other. Objects communicate with one another by sending and receiving information in the same way people pass messages to one another as shown below.



The message passing involves specifying the name of the object, name of the method and information to be sent. For example consider a statement:

Student .marks (htno);

↓ ↓ ↓
 Object Message Information

5. What are the Advantages or benefits of OOP?

The new technology provides greater programmer productivity, better quality of software and lesser maintenance of cost. OOP offers several benefits to both the programming designer and the user.

- 1) Through inheritance we can eliminate redundant code and extend the use of extend the use of existence class.
- 2) It is possible to have multiple instances of an object to co-exist without any interface.
- 3) It is easy to partition the work in a project based on objects.
- 4) User defined data types can be easily created.
- 5) Software complexity can be easily managed.
- 6) Inheritance enforces inventions of new data types.
- 7) Object oriented systems can be easily upgraded from small to large systems.
- 8) It is possible to map objects in problem domain to those objects in the program.
- 9) The principle of data hiding helps the programmer to build secure programs that can be accessed by code in other parts of the program.
- 10) We can build programs from standard working modules this leads saving of development time.

6. Applications of OOP:

OOP is used in the development of most software systems. Today oops is used in the design of Graphical User Interface (GUI) on systems such as windows operating system.

- Artificial Intelligence and expert system.
- Simulation and modeling studies.
- Object oriented data bases systems.
- Object oriented operating systems.
- Real time systems.
- Office Automation system.
- CAD/CAM systems.
- Multimedia applications and Graphical User Interface (GUI).
- Computer based training educations.
- Data recovery systems.
- Distributed systems.
- Audio and Video manipulation tools.
- Image processing tools.

7. What are the Features (Buzz words) of JAVA?

When Java was first created, Sun Microsystems described it with a series of buzzwords. “Java is simple, Object Oriented, robust, secure, portable, high-performance, and architectural, interpreted, multithreaded, distributed, dynamic language”.

Simple: - Java programming language is very easier. It is syntactically similar to C and C++. Complex features of these languages are either simplified or totally eliminated in Java.

Object Oriented:- Unlike C++, Java purely object oriented programming language. Java programs use classes and objects. We call any language as object oriented programming language when it has the five features. 1. Classes and objects 2. Encapsulation 3. Abstraction 4. Inheritance 5. Polymorphism.

Robust (Strong/ Powerful): - Java is a most strong language which provides many securities to make certain reliable code. Java programs will not crash because of its exception handling and its memory management features.

Secure: - Java is designed for use on Internet. Security problems like eyes dropping; tampering, impersonation and virus can be eliminated. Thus java programs are useful in construction of highly secure systems.

Portable: - Programs developed in Java can be compiled in variety of operating environments without modifications to the source code. Memory size of Java variables is the same in environment. This is another contributing factor for the portability of Java application.

High- Performance:- When compared to other interpreted languages, Java offers high performance. Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.

Architectural-neutral:- This feature is in accordance with the sun Microsystems slogan about java programs. “Write once and run anywhere”. This feature is famously known as platform independence. Java byte code is not machine dependent; it can run on any machine with any processor and with any OS.

Interpreted:- Java programs are both compiled and interpreted. JVM interprets the class file into machine code.

Multithreaded: - A thread is an individual process in a program. Each thread (or process) can execute block of statements and perform a separate task. Thus multiple threads can perform multiple tasks. This is an essential feature to design server side programs.

Distributed:- Java has rich libraries for network programming and distributed computing. As java is mainly meant for Internet based programming, it is no surprise that java is distributed. An application is said to be distributed if the business objects are geographically dispersed in the network and communicating one another.

Dynamic: - Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

Overview of Java Language

8. Write about Structure of Java program

A Java program may contain one or more sections as given below.

Document section
Package Statements
Import Statements
Interface Statements
Class Definitions
Main Method Class
<pre>{ Main Method definition }</pre>

1) Document Section:

- ☞ This is an optional section
- ☞ This section consist information about the program
- ☞ It is not understand by compiler
- ☞ This documenttation is provided by using comments.
- ☞ Java supports three methods of documentation.

// Single line comment

/* */ Multiline comment

// * */ Document comment

2. Package Statement:

- ☞ It is an optional section.
 - ☞ This statement declares a **package** name and informs the compiler that the classes defined here are belongs to this package.
- ```
package student;
```

#### **3. Import statement:**

- ☞ Similar to #include statement in C.
- ☞ This statement is used for importing existing packages.

- ☞ It does not import complete class/ package.
  - ☞ It must be exist before class definition
- ```
import student.test;
```

4. **Interface statements:**

- ☞ It is also an optional section.
- ☞ It is like a class but include a group of method declarations.
- ☞ This is used only when we implement the multiple inheritance features in the program.

5. **Class definition section:**

- ☞ class is data type.
- ☞ Classes are the primary and essential elements of a java program.
- ☞ A java program may contain multiple class definitions.

```
class class_name
{
    Attributes;

    Methods;
}
```

6. **Main method class:**

- Main method class is the essential part of a java program. A simple program may contain only this part.
- The **main** method creates objects of various classes and establishes communication between them.

9. Write a Simple Java program and explain in detail.

A simple Java program is as follows

```
class Simple
{
    public static void main(String args[])
    {
        System.out.println("Hello Good Morning , I am Mahesh");
        System.out.println("Java is better than C++");
    }
}
```

Class declaration:

The first line class “Simple” declares a class; everything must be placed inside a class. ‘Simple’ is a java identifier that specifies the name of the class. ‘**class**’ is a keyword and declares a new class definition.

Opening brace:

Every class definition in Java begins with an opening brace and ends a matching closing brace.

The main line:

The third line “public static void main(String args[])” defines a method named **main**. Every Java program must include **main()** method.

- *Public* – The keyword **public** is an access specifier that declares the **main** method is accessible to all other classes.
- *Static* – The keyword **static** declares the method that belongs to entire class. There is no need to create an object to invoke the static method.
- *Void* – The type modifier **void** specifies that the **main** method doesn’t return any value.
- *String args[]* – Declares a parameter **args[]**, is used for command line argument.
- **The output line:** The statement “System.out.println(“Java is better than C++”);” This is similar to the **printf()** statement of ‘C’. The **println** method is a member of **out** object, which is static data member of **System** class. This line prints the string

Helo Good Morning, I am Mahesh

Java is better than C++ on the screen.

The method **println** always appends a new line character to the end of the string. Every Java statement must end with a semicolon.

10. What is token? Write about Java Tokens

Token:

- ☞ A java Program is a collection tokens, comments and white spaces.
- ☞ Smallest unit of a program or an individual unit is called token.
- ☞ Tokens can be keyword, operator, separator, constant and any other identifiers.
- ☞ When we are working with token we can't split the token or we can't break the token.
- ☞ In JAVA we have five types of tokens available.
 - a) Reserved keywords
 - b) Identifiers
 - c) Literals
 - d) Operators
 - e) separators

Keyword:

- ☞ It is a reserved word some meaning is already allocated to this word. And that meaning already knows by compiler.
- ☞ Keywords are an essential part of a language definition.
- ☞ Keywords are combined with operators and separators according to syntax.
- ☞ We cannot use keywords as names for variables, classes, methods, arrays and so on.
- ☞ All keywords must be used in lower case letters.
- ☞ In Java total number of keywords is 50.

Eg: continue for new switch if Boolean

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Identifiers

An identifier is a name given to the program elements such as variables, arrays, classes, objects, methods, labels, packages and interfaces. Identifiers may consist of sequence of letters, numerals, or underscores.

Rules for forming identifier names:

- ☞ Identifiers cannot include any special characters or punctuation marks (like #, , ^, ?, ., etc) except \$ and underscore.
- ☞ Keywords cannot be used as identifiers.
- ☞ Length of identifier is unlimited.
- ☞ In identifier declarations name of the variable must starts with alphabet or underscore only.

Examples: roll_number, marks, name, emp_number, basic_pay, HRA, DA, dept_code

Literals:

Literals are sequence of characters(digits, letters, and other characters) that represent constant values to be stored in variables. Java language specifies 5 types of literals.

- Integer literals Eg: -10, 123
- Floating point literals Eg: -10.55, 0.55
- Character literals Eg: 'x', '5'.
- String literals Eg: "123", "Mahesh".
- Boolean literals Eg: True, false.

Operators:

It is a special kind of symbol which performs a particular task.

Operator: An operator is a symbol that performs an operation.

Operand: An operator acts upon variables which are called operands.

$$a + b$$

Operators are of many types such as Arithmetic operators, Relational operators, Logical operators, Assignment operators, Increment and Decrement operators and Special.

Separators:

By using separators we can separate individual units. Separators are symbols used to indicate where the groups of code is divided and arranged. The separators are

- ✓ Parenthesis ()
- ✓ Braces { }
- ✓ Brackets []
- ✓ Semicolon ;
- ✓ Comma ,
- ✓ Period .

11. Write about Java Statements

A statement is executable combination of tokens ending with a semicolon. Java implements several types of statements.

Statement	Description
1. Empty Statement	These do nothing and are used during program development as a place holder.
2. Labeled Statement	Any statement may begin with a labels in java are used as arguments of jump statements.
3. Expression Statement	Most statements are expression statements. Java have 7 types of expressions statements. Assignment, Pre_increment, Post_increment, Pre_decrement, Post_decrement etc.
4. Selection Statement	This selects one of several control flows. There are 3 types of selection statements in Java. If, if...else, switch.
5. Iteration Statement	This specify how and when looping will take place. There are 3 types of iterative statements. while, do, for
6. Jump Statement	Jump statement pass control to forward or backward in the program. There are 4 types of jump statements. break, continue, return and throw.
7. Synchronization Statement	These are used for handling multithreading.
8. Grouping Statement	Guarding statements are used for safe handling of code from Exceptions. This statements use the keywords try, catch and finally.

12. How to Implement a Java program

JDK is the software development environment. It is used for developing java program and applet. It includes the JRE (Java runtime environment), javac (Java compiler), jar, javap, etc. Implementation of Java program is a series of steps.

- Creating the program.
- Compiling the program.
- Running the program.

Creating the program:

We can create a program using any text editor.

```
eg:  class Test
    {
        public static void main(String args[])
        {
            System.out.println("Hello");
            System.out.println("Welcome to Java");
        }
    }
```

We must save this program in a file called **Test.java**. That is the file name contains class name. This file is called *source file*.

Compiling the program:

To compile the program we must run the Java compiler **javac**, with the name of the source file on the command line.

```
javac Test.java
```

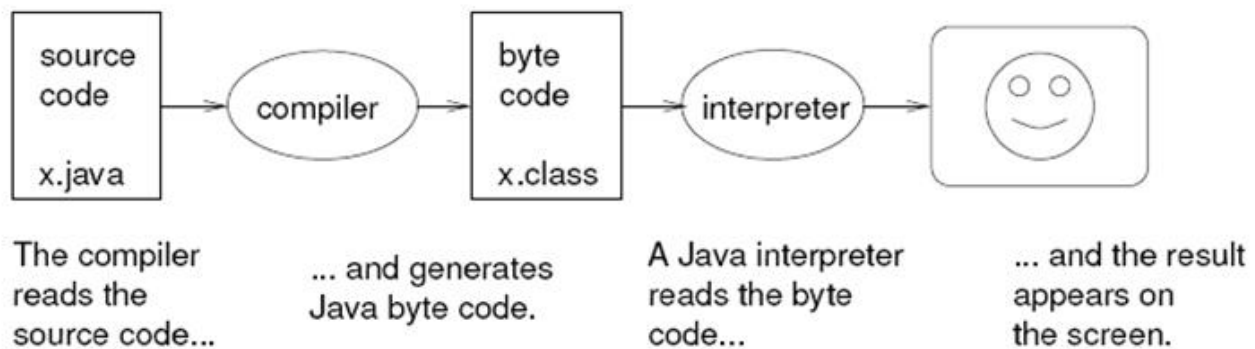
If there is no errors then the compiler creates a file called **Test.class** containing the byte code of the program.

Running the program:

We use the java interpreter to run the program. At the command prompt type

```
java Test
```

Now, the interpreter looks for the main method in the program and begins execution from there. When executed our program displays following

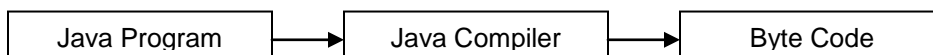


Hello

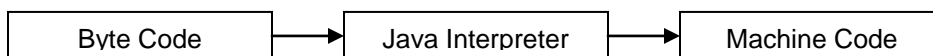
Welcome to Java

13. Write about Java Virtual Machine

- JVM is a program or software which provides runtime environment for byte code.
- JVM is platform dependent program.
- JVM is having common specifications irrespective of O.S. and architecture.
- Java compiler translates the source code into byte code.
- The process of compiling a java program into byte code is also referred as *virtual machine code*.



- The virtual machine code is not machine specific. The machine code is generated by the java interpreter by acting an intermediary between the virtual machine and real machine.



JVM mainly can do the following

- ☞ Locates a file and loads the code.
- ☞ Verifies the code.
- ☞ Converts byte-code into machine code (executed code).
- ☞ Allocates the memory into RAM.
- ☞ Executes the code.

14. Explain in detail Command Line arguments

- Command line arguments are parameters that are supplied to the program at the time of execution.
- Java programs can receive and use the arguments provided in the command line.
- The arguments provided in the command line are passed to the array args[] as its elements.

Eg: java Test Basic Fortran C++ Java

This command line contains 4 arguments. These are assigned to the array args as follows

Basic -> args[0]

Fortran -> args[1]

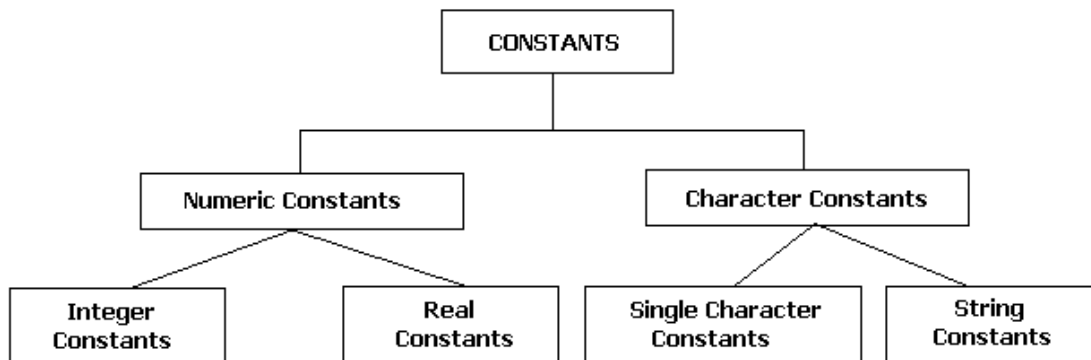
C++ -> args[2]

Java -> args[3]

```
//a program to command line arguments
class Cmdline
{
    public static void main(String args[])
    {
        int count, i=0;
        String item;
        count=args.length;
        System.out.println("No. of arguments="+count);
        while(i<count)
        {
            item=args[i];
            System.out.println("Java is:"+item);
            i=i+1;
        }
    }
}
```

CONSTANTS, VARIABLES & DATA TYPES:**15. What is constant? Explain different type Constants.**

Constants in Java refers to the fixed value, that do not change during execution of a program. It is a fixed never changed during the execution of the program. Constants are used to define fixed values like pi or electron charge.

**Integer Constants:**

An *integer* constant refers to sequence of digits 0-9, preceded by an optional minus signed.

Eg: -32, 100

Real Constants:

The numbers with decimal point is called as *real constants* or *floating point constants*. Real constants are expressed either in *decimal notation* or *exponential notation*.

Eg: 12.5, -0.03

For example, the value 215.65 may be written as 2.1565e2 in exponential notation. Here e2 means multiply by 10^2 .

Single character constants:

It contains a single character enclosed in a pair of single quote mark.

Eg: 'x', '5', '&'.

String Constants:

A string constants is a sequence of character enclosed in a double quote. The characters may be alphabets, digits, special characters and blank space.

Eg: "gminformatics", "123"

Backslash Character constants :

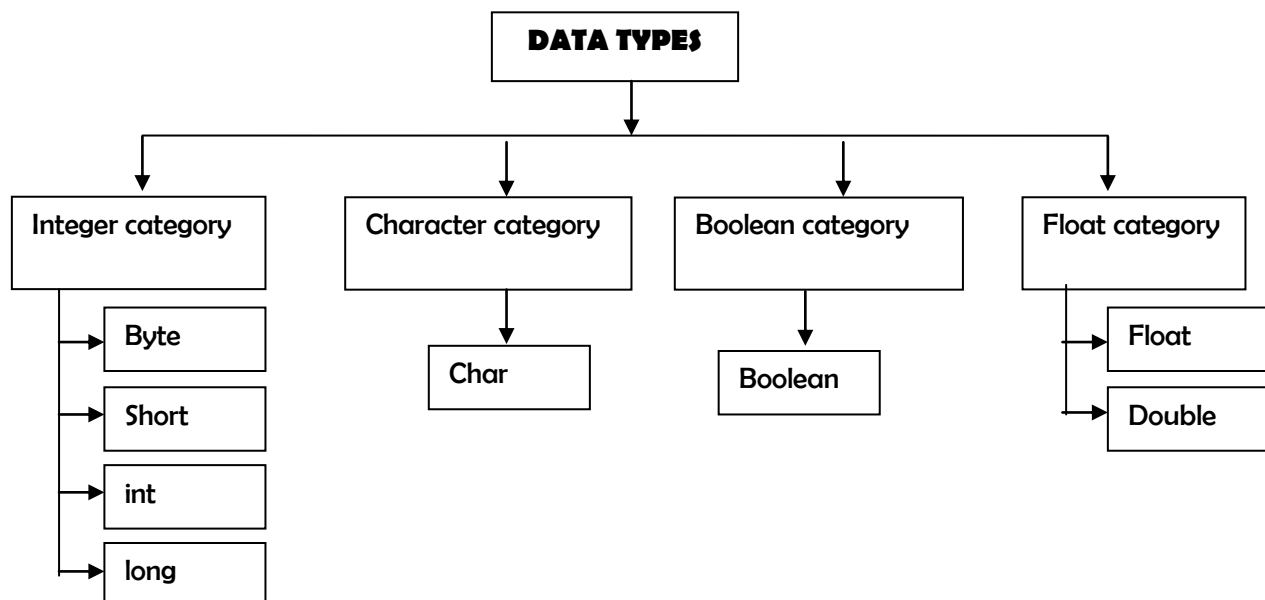
Java supports some special backslash character constants that are used in output methods. For example the symbol '\n' stands for new line character. A list of such backslash character constants are given below. These character combinations are known as "Escape Sequences".

Constant	Meaning
'\b'	Back Space
'\f'	Form Feed
'\n'	New Line
'\r'	Carriage return
'\t'	Horizontal tab
'\"'	Single quotes
'\"'	Double quotes
'\\'	Back Slash

16. What is data type? Explain different types of Data Types.

Data types are the keywords and tell the compiler which type of data (information) is maintained in memory. Data types will decide what type of values need to be hold into variables.

In Java, we have eight data types which are organized in four groups. They are integer category data types, float category data types, character category data types and Boolean category data types.



1. Integer Category data types: These are used to represent integer data type contains four data type which are given in the following table.

S.NO	Data Type	Size (Bytes)	Range
1	Byte	1	0127 to -128
2	Short	2	+32767 to -32768
3	Int	4	+2147483647 to -2147483648
4	Long	8	+ 9.223*10 ¹⁸

2. Float category data types: Float category data types are used for representing the data in the form of scale, precision i.e. these category data types are used for representing float values. This category contains two data types; they are given in the following table.

S.NO	Data type	Size(Byte)	Range	NO. of decimal places
1	Float	4	+2147483647 to -2147483648	8
2	Double	8	+ 9.223*10 ¹⁸	16

3. Character category data types:

A character is an identifier which is enclosed within single quotes.

In java to represent character data, we use a data type called char. This data type takes two bytes since it follows UNICODE character set.

Java is available in 18 international languages and it is following UNICODE character set.

UNICODE character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters.

Data type	Size(Bytes)	Range
Char	2	+32767 to -32767

Boolean category data types:

Boolean category data types are used for representing logical values. i.e. TRUE or FALSE values..To represent logical values we use a keyword called Boolean This data type takes 0 bytes of memory space.

String data type

A string represents group of characters. A string is considered as data type as well as a 'class'.

String name =" Mahesh MCA"

String str = new string(" A string")

17. What is variable and write the rules to follow while selecting a variable?**Variable:**

A variable is defined as a meaningful name given to a data storage location in computer memory. Or

A Variable is a quantity that does change during the program execution.

Rules for forming identifier names:

- The variable must begin with an alphabet.
- Upper and lower case letters are different.
- It should not be a keyword.
- White space is not allowed.
- Variables names can be of any length.
- Name of the memory location is called variable.
- Before using any variable in the program it must be declare first.
- Declaration of variable means need to mention data type, name of the variable followed by semicolon.
- In variable declarations name of the variable must starts with alphabet or underscore only.
- In variable declaration existing keywords operators, separators, constants and any other special characters will be not allowed.

Declaration of variables

The general form is

Syntax: datatype var1, var2, ...varn;

Here, variables are separated by commas and the declaration must end with a semicolon.

Eg: int count;
 float sum, avg;
 double pie;

Giving values to variables

Values can be given to a variable in two ways.

- By using an assignment statement.
- By using an read statement

Assignment Statement:

A simple method of giving values to variables is through the assignment statement.

Syn: variablename=value;

Eg: a=10;

We can also assign a value to variable at the time of declaration.

Syn: datatype variablename=value;

Eg: int a=10;

Scanner classes (For reading inputs dynamically)

- ☞ We can take input from a user and display something on the screen
- ☞ In Java, we input with the help of the **Scanner** class. Java has a number of predefined classes which we can use.
- ☞ This class contains some utility methods to read input like integer, float, double, string, Boolean.
- ☞ Predefined classes are organized in the form of packages.
- ☞ This **Scanner** class belongs to java's "util" package.
- ☞ So to use the Scanner class, we first need to include java.util package in our program.

```
import java.util.Scanner; // This will import just the Scanner class
```

```
import java.util.*; // This will import the entire java.util package
```

- ☞ After importing, we need to write the following statement in our program.

```
Scanner s = new Scanner (System.in);
```

- ☞ Here we are declaring **s** as an object of **Scanner** class.
- ☞ **System.in** as an argument to the constructor. i.e. input will be given to the system.

```
int n;
```

```
n = s.nextInt(); // s is object of Scanner class
```

Statement **n = s.nextInt();** is used to input the value of an integer variable 'n' from the user.

Here, **nextInt()** is a method of the object **s** of the Scanner class.

Method	Inputs
nextInt()	Integer
nextFloat()	Float
nextDouble()	Double
nextLong()	Long
nextShort()	Short
next()	Single word
nextLine()	Line of Strings
nextBoolean()	Boolean

18. How to declare Scope of variable?

Scope is how far a variable can be accessed. The scope of variables is depends on location where a variable is declared. A block is begin with an opening curly brace and ended by a closing curly brace. Java allows variables to be declared with in any block. A block defines a scope.

The scope defined by a method begins with it's opening brace. However, if that method has parameters, they are also included within the method's scope. So we can protect from unauthorized access and modification. Java variables are classified into three kinds.

1. Instance variables.
2. Static Variables (Class variables)
3. Local variables.

Instance variables: (Non static variables)

- ☞ Instance variables are declared inside a class but outside the methods and blocks. Instance variable is also known as non-static variable. It has the unique value for each object (instance).
- ☞ Instance variables must be accessed by object name. i.e object name.var name.
- ☞ Instance variable is created when an object is created with new keyword and destroyed when object is destroyed.

Syntax : DataType v1,v2....vn

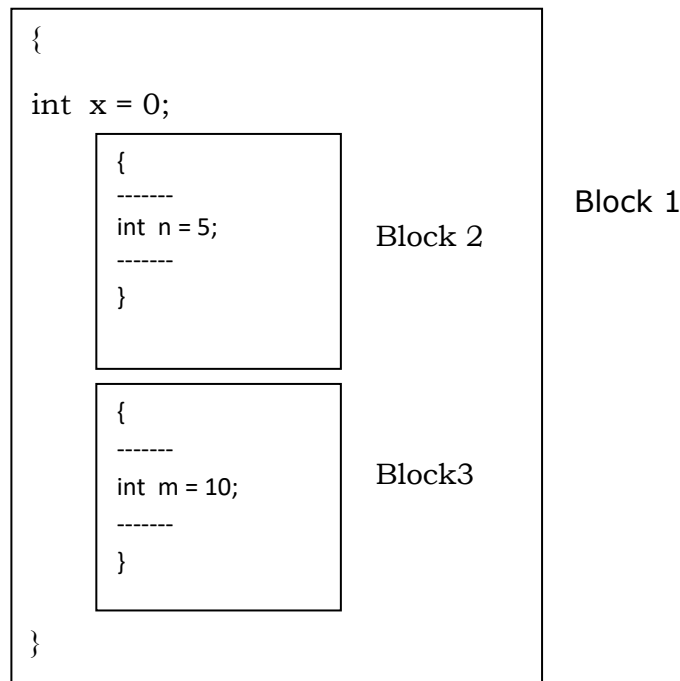
Static variables

- ☞ These variables are also called as class variables.
- ☞ These variables are declared inside the class but outside the methods and blocks but by using '**static**' keyword. These variables also called static variables.
- ☞ Static variable is the class level variable which is declared with static keyword. Static variable tells the compiler that there is only one copy of this variable in existence.
- ☞ If one object change the value if you would like to reflected to all other objects make your variables as static variables.
- ☞ All the objects can have same value same state by using static variables.
- ☞ Static variable is created when program starts and destroyed when the program stop. Static variables must be accessed with respect to class name. i.e. class name. var name

syntax:- static data Type v1,v2....vn

Local variables:

- ☞ Variables declared and used inside methods are called local variables.
- ☞ They are not available for use outside the method definition.
- ☞ Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }.
- ☞ There is no special keyword for declaring local variable.



Each block contains its own set of local variable declarations. In the above figure the variable `x` declared in block 1 is available in all the three blocks. The variable `n` declared in block 2 and is available only in block 2. Similarly `m` is accessible in block3.

```

class VariableType
{
    int a=10;          //instance variable
    static int b=20;    //static variable
    public int m1()
    {
        int c = 30;    //local variable
        System.out.println("c = "+c);
        return 0;
    }
    public static void main(String args[])
    {
        VariableType v = new VariableType ();
        System.out.println("a = "+v.a+" b = "+b);
        v.m1();        //invoke the m1 method
    }
}
  
```

Write about Symbolic Constants

In JAVA to make the identifiers are as constants, we use a keyword called final. Final is a keyword which is playing an important role in three levels. They are at variable level, at method level and at class level.

When we don't want to change the value of the variable, then that variable must be declared as final.

A symbolic constants can be declared with the following format

Syntax for FINAL VARIABLE INITIALIZATION

Final data type v1=val1, v2=val2 ... vn=valn;

For example:

```
Final int a=10;  
    a=a+20; //invalid  
    a=30; //invalid
```

```
FINAL int PASS=40;
```

- Symbolic names take the same form as variable names. But they are written in CAPITALS to make difference from normal variable names.
- After declaration of symbolic constants they shouldn't be assigned any other value in the program.
- They cannot be declared inside a method. They should be used only as class data members in the beginning of the class.

Write about Type Casting

Type casting is the process of converting the value in one data type to another data type. It uses the following syntax.

```
type var1 =( type ) var2;  
Ex:- int m=50;  
    byte b = (byte) m;  
    short b = (short) m;
```

Casting is often necessary when a method returns a type different than one we required. Four integer types can be casted to any type other than Boolean. Casting into smaller type may result in the loss of data. Similarly the float and double can be casted into any type except Boolean. Casting a floating point value to an integer will result in the loss of fractional part.

Automatic conversion (Widening):

The following table shows the casting that results in no loss of information:

From	To
byte	short,int,long,float,double.
short	int,long,float,double.
char	int,long,float,double.
Int	long,float,double.
long	float,double.
float	double
double	

For some types, it is possible to assign a value of one type to a variable of a different type to a cast.java does the conversion of the assigned value automatically this is known as automatic conversion. Automatic type conversion is possible only if the destination type has enough precision to store the source value.

Automatic type conversion is possible only the process of assigning a smaller type to a large is known as promotion or widening.

Eg: byte a=10;

int a=b;

Narrowing:

The process of assigning a large type to a smaller type is known as “narrowing”. This may result in the loss of information. Narrowing is also known as explicit type casting. It is programmers responsibility to cast(convert) large type to small data type.

Ex:- int m=129;

byte b =(byte) m; // here loss of value

//a program to find total and average by using type casting

class Avg

```
{
    public static void main(String args[])
    {
        int a,b,c,tot;
        float avg;
        a=2; b=3; c=6;
        tot=a+b+c;
        avg=(float)tot/3;    //type casting
        System.out.println("tot="+tot);
        System.out.println("avg="+avg);
    }
}
```

Operators and expressions

What is operator explain different type operators:

An operator is a symbol used to perform an operation. If an operator acts on a single variable, then it is called '*Unary Operator*'. If an operator acts on two variables then it is called '*Binary Operator*'. If an operator acts on three variables then it is called '*Ternary Operator*'. The following are the Operators available in Java.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

1. Arithmetic operators.
2. Relational operators.
3. Logical operators.
4. Assignment operators.
5. Increment and Decrement operators.
6. Conditional operator.
7. Bit - Wise operators.
8. Special operators.

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

Operator	Meaning	Example	Result
+	Addition or unary plus	3+4	7
-	Subtraction or unary minus	5-7	-2
*	Multiplication	5*5	25
/	Division (gives quotient)	14/7	2
%	Modulo division(gives remainder)	20%7	6

Relational operators:

Relational operators are used to compare two values or expressions. There are six relational operators.

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
>=	Greater than or equal to	x >= 3

2. Logical operators: Logical operators are used to combine two or more relational expressions.

These operators are useful to construct compound conditions.

To construct compound condition is a combination of more than one simple condition.

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

And (&&)operator: Logical and(&&) returns when both operands are true otherwise it returns false.

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

Or (||)operator : Logical or (||) returns true when either of it's operands are true and returns false if both the operands are false.

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

Not (!) operator: This operator negates the result of a expression. It means if the condition is true then it gives false and if the condition is false then it gives true.

A	!A
T	F
F	T

4. Assignment operator: It is a binary category operator which is used to assign the right side value to left side. When we are working with binary operators it should require and among those two operands.

Syntax: L=R;

When we are working with left side argument must be variable type only, and right argument is optional.

Simple Assignment :

The general form of the simple assignment is

Variable=constant or variable or expression;

Ex : a=b=1;

M=n=g=a+b;

5. Increment and Decrement Operators:

Increment Operator (++):

The increment operator is a unary operator. It is used to increase the value of an operand by 1. The operand must be a variable. The increment operator ‘++’ has different meaning depending on the position it is used. It means this operator is used in two ways. They are pre-increment and post-increment.

Pre-increment: If the ‘++’ operator precedes the operand then it is called “Pre increment”. In this method, the pre-increment operator increments the value of variable first and then the new value is used in the expression.

For example:

c = ++a;	means	a = a + 1;
		c = a;

Post-increment: If the ‘++’ operator succeeds the operand then it is called “Post increment”. In this method, the old value of the variable is used in expression and then it is incremented.

For example: `c = a++;` means `c = a;`
 `a = a + 1;`

Decrement Operator (--):

The decrement operator is a unary operator. It is used to decrease the value of an operand by 1. The operand must be a variable. The decrement operator ‘`-`’ has different meaning depending on the position it is used. It means this operator is used in two ways. They are pre-decrement and post-decrement.

Pre-decrement: If the ‘-’ operator precedes the operand then it is called “Pre decrement”. In this method, the pre-decrement operator decrements the value of variable first and then the new value is used in the expression.

For example:

c = -a;	means	a = a - 1;
		c = a;

Post-decrement: If the ‘-’ operator succeeds the operand then it is called “Post decrement”. In this method, the old value of the variable is used in expression and then it is decremented.

For example: $c = a - -;$ means $c = a;$
 $a = a - 1;$

Initial Values		Expression	Final Values	
A	B		A	B
3	7	a = ++b	8	8
3	7	a = b++	7	8
3	7	a = --b	6	6
3	7	a = b--	7	6

6. Conditional Operator: This operator performs condition based execution. This operator operates on three operands. Hence it is also known as Ternary Operator. This operator simplifies “if...else” control statement. The general form of conditional operator is as follows:

Syntax: **var = (condition) ? Statement1 : Statement2 ;**

Working: The conditional operator first evaluates the given condition. If the condition is true then ‘Statement1’ is evaluated. If the condition is false then ‘Statement2’ is evaluated.

Example:-

```
import java.io.*;
class max
{
    public static void main(String args[]) throws IOException
    {
        int a,b,large;
        DataInputStream d=new DataInputStream(System.in);
        System.out.print("Enter First Number");
        a=Integer.parseInt(d.readLine());
        System.out.print("Enter Second Number");
        b= Integer.parseInt(d.readLine());
        large = (a>b) ? a : b;
        System.out.println("Large value is " + large);
    }
}
```

7. Bitwise Operators: Java has the capability of manipulating the bits (0 or 1) directly stored in the memory. These operators perform bitwise operations on a bit by bit level. However, they must be used only with integer or character type of values. The operators are as follows.

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR (Exclusive OR)
~	1's complement
<<	Left shift
>>	Right shift
>>>	Right shift with Zero fill

Bitwise AND (&) operator: If all the bits are “1” then this operator returns “1” otherwise it returns “0”.

Truth table:

&	1	0
1	1	0
0	0	0

Example: Let x = 15 and y = 19, then z = x & y is as follows

Value of x :	00001111
Value of y :	00010011
Value of z : (x & y)	00000011

Bitwise OR (|) operator: If any or all the bits are “1” then it returns “1” otherwise it returns “0”.

Truth table:

	1	0
1	1	1
0	1	0

Example: Let x = 15 and y = 19, then z = x | y is as follows

Value of x :	00001111
Value of y :	00010011
Value of z : (x y)	00011111

Bitwise XOR (^) operator: If one bit is “1” and other is “0” then it returns “1” otherwise it gives “0”.

Truth table:

^	1	0
1	0	1
0	1	0

Example: Let $x = 15$ and $y = 19$, then $z = x \wedge y$ is as follows

Value of x :	00001111
Value of y :	00010011
Value of z : ($x \wedge y$)	00011100

1's complement (~) operator: This operator is a unary operator that reverses the state of each bit. That means it changes "0" as "1" and "1" as "0".

Example: Let $x = 19$, then $z = \sim x$ is as follows

Value of x :	00010011
Value of z : ($\sim x$)	11101100

Left Shift (<<) operator:

This operator is used to move bit patterns to the left. It is used in the following form

op<<n

Here, 'op' is an integer value and 'n' is number of bit positions. The left shift operator shifts all the bits of operand 'op' to the left by 'n' positions. The left most 'n' bits in the original bit pattern will be lost and right most 'n' bit positions are filled with zeros.

Example: Let us consider 'a' is an unsigned integer whose bit pattern is as follows:

a →	1100110011001111
a<<3 →	0110011001111000
a<<5 →	1001100111100000

The left shift operator is used for multiplication purpose. It means, $x \ll n$ gives the result as $x * 2^n$.

Example : $5 \ll 1$ gives 10, $5 \ll 2$ gives 20, $5 \ll 3$ gives 40

Right Shift (>>) operator:

This operator is used to move bit patterns to the right. It is used in the following form

op>>n

Here, 'op' is an integer value and 'n' is number of bit positions. This operator shifts all the bits of operand 'op' to the right by 'n' positions. The right most 'n' bits in the original bit pattern will be lost and left most 'n' bit positions are filled with zeros.

Example: Let us consider 'a' is an unsigned integer whose bit pattern is as follows:

a →	1100110011001111
a>>3 →	0001100110011001
a>>5 →	0000011001100110

The right shift operators can also be used for division purpose. It means, $x \gg n$ gives the result as $\frac{x}{2^n}$

Example : $15 \gg 1$ gives 7, $15 \gg 2$ gives 3, $15 \gg 3$ gives 1

Right Shift Zero fill (>>>) operator:

When dealing with positive numbers, there is no difference between >>> and >> operators. The difference arises when dealing with negative numbers. The negative numbers have the higher order bit set to 1. The Right shift Zero fill operator shifts zeros into all the upper bits, including the higher order bit. Thus make a negative number into positive number.

op>>>n

8. Special Operators: The following are the special operators used in Java.

- 1) Instanceof Operator
- 2) Dot Operator
- 3) Cast Operator
- 4) new Operator

Instanceof Operator: The instanceof is an object reference operator and return true if the object on the left hand side is an instance of the class given on the right side. This operator determines whether the object belongs to a particular class or not.

Ex: s1 **instanceof** (student);

It gives **true** if the object person belongs to the class student; otherwise it is **false**

Dot Operator: this operator is also called as member selection operator (.). The Dot Operator is used to access the instance variables, methods of class objects, classes, and sub-packages from a package.

Ex: person.age;
 person.salary();

Cast Operator : Cast operator is used to convert one data type into another data type. This operator can be used by writing datatype inside parenthesis.

Ex: double x = 10.54;
 int y = (int) x; // stores 10 into y.

new Operator : new operator is used to create objects to classes. Objects are created on heap memory by JVM, at runtime.

Syntax : **classname obj = new classname();**

Write about expressions:

Definition: An expression is a collection of operands joined together by certain operator that represent a value. There are two types of expressions. They are

1. Arithmetic or Numeric Expression
2. Relational or Boolean Expression

Arithmetic Expression:

When an expression uses arithmetic operators then it is called arithmetic or numeric expression. An arithmetic or numeric expression always represents numeric value.

1. Two successive operators are not permitted. However, the space(s) or parenthesis can be used to separate two successive operators.

Example: $a*-25$ is invalid, $a*(-25)$ is valid

2. Arithmetic operations cannot be implied. It means, no operator is assumed to be present.

Example: $a = bc+d(x+y+z)$ is invalid

$a = b*c+d*(x+y+z)$ is valid.

3. The evaluation of an arithmetic expression follows the precedence order of operators.

Example: $5+2*5$ gives 15 and $(5+2)*5$ gives 35

4. There cannot be imbalance of parenthesis. The number of left parenthesis must be equal to the number of right parenthesis.

5. The arithmetic operation between two integers gives integer result. It is Integer mode expression.

Example: $5/2$ gives 2

6. The arithmetic operation between two float values gives float result. It is real mode expression

Example: $5.0 / 2.0$ gives 2.5

7. The arithmetic operation between an integer and a float value gives float result. It is mixed mode expression.

Example: $5 / 2.0$ gives 2.5

8. The arithmetic operation can be performed between integer and character data types.

Example: $'A'+30$ gives 95

Relational or Boolean Expression:

When an expression uses relational and logical operators then it is called relational expression. A relational expression is also known as condition or logical expression. This expression gives the result either true (1) or false (0).

Examples:

1. $a > b$
2. $\text{salary} \leq 5000$
3. $x == 0$
4. $(a > b \mid a > c)$
5. $((\text{avg} \geq 50) \&\& (\text{avg} \leq 75))$

OPERATOR PRECEDENCE:

In 'java' language, an expression is evaluated based on the precedence order of operators. The following table shows various operators and their precedence.

Evolution of operators in an expression is as follows:

1. An expression will be evaluated based on the priority of operators.

Example: $5 + 3 * 2$

In the above expression, there are two operators (+ and *) in which * has high priority. So the operator * is evaluated first and then + is evaluated. Hence the result is 11.

2. If an expression has equal priority operators then the expression is evaluated based on the associativity.

Example: $5 + 3 - 2$

In the above expression, the operators '+' and '-' have equal priority. Their associativity is left to right. So first "+" is evaluated and then "-" is evaluated. Hence the result is 6.

3. In case, if we want to change priority order then we can put an expression within parenthesis. Example: $(5 + 3) * 2$ gives the result 16

Example:

$$\begin{aligned}
 x &= 3 * 2 / 2 + 3 / 2 + 2 - 2 + 3 * 2 \\
 &= 6 / 2 + 3 / 2 + 2 - 2 + 3 * 2 \\
 &= 3 + 3 / 2 + 2 - 2 + 3 * 2 \\
 &= 3 + 1 + 2 - 2 + 3 * 2 \\
 &= 4 + 2 - 2 + 6 \\
 &= 6 - 2 + 6 \\
 &= 4 + 6 \\
 &= 10
 \end{aligned}$$

Operator	Meaning	Associativity	Rank
.	Member selection	Left to right	1
()	Function Call	Left to right	1
[]	Array Element Reference	Left to right	1
-	Unary Minus	Right to left	2
++	Increment	Right to left	2
--	Decrement	Right to left	2
!	Logical Negation	Right to left	2
~	One's Complement	Right to left	2
(type)	Casting	Right to left	2
*	Multiplication	Left to right	3
/	Division	Left to right	3
%	Modulo Division	Left to right	3
+	Addition	Left to right	4
-	Subtraction	Left to right	4
<<	Left shift	Left to right	5
>>	Right shift	Left to right	5
>>>	Right shift with zero fill	Left to right	5
<	Less than	Right to left	6
<=	Less than or equal to	Right to left	6
>	Greater than	Right to left	6
>=	Greater than or equal to	Right to left	6
Instance Of	Reference compare	Left to right	6
==	Equality	Left to right	7
!=	Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
	Logical OR	Left to right	12
?:	Conditional Operator	Right to left	13
=	Assignment Operator	Right to left	14
Op=	Short hand assignment	Right to left	14

UNIT-II

DECISION MAKING & BRANCHING

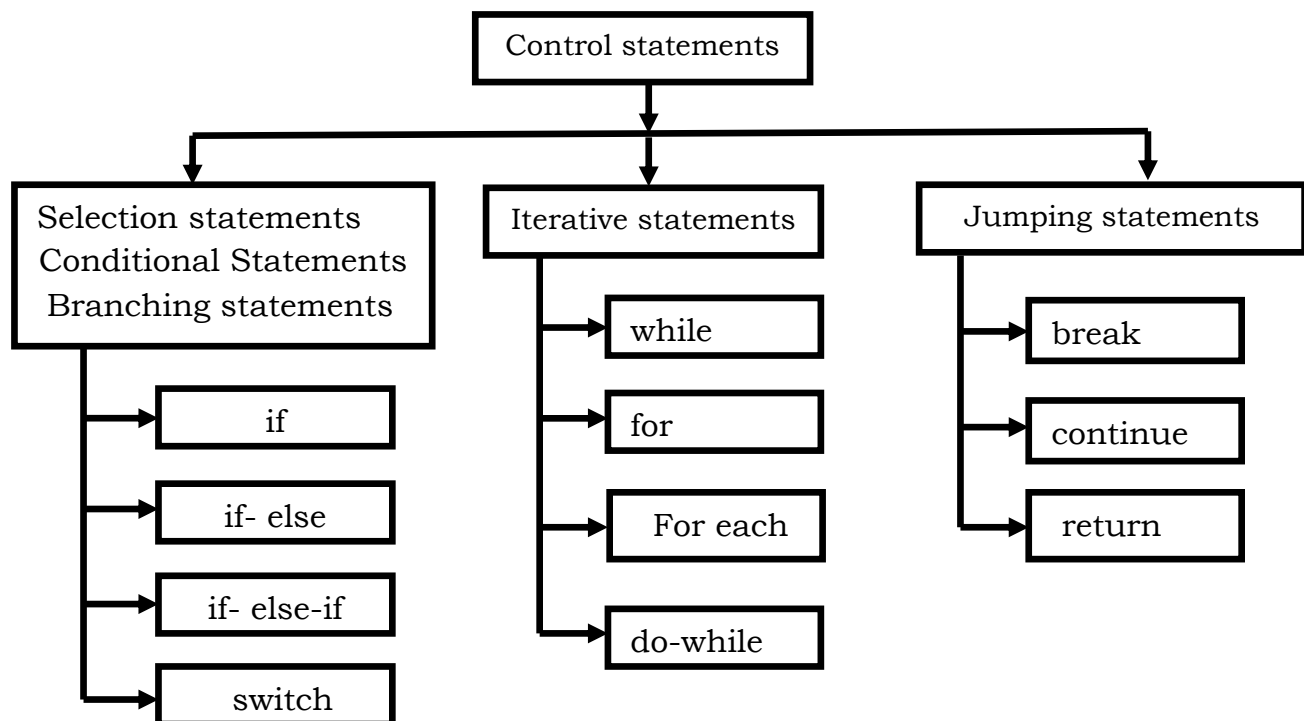
INTRODUCTION TO DECISION CONTROL STATEMENTS:

Control structures: Control statements are used to provide the flow of execution with condition. In java, control structures can be divide three parts:

In JAVA, control statements are classified into three categories as:

- a) Selection (or) Decision Control Statements
- b) Loop (or) Iterative Control Statements
- c) Branch (or) Jump Control Statements

The control structures are used to control the flow of program execution. In C language there are there are three types of flow statements are exist.



Conditional branching or conditional selection statements:

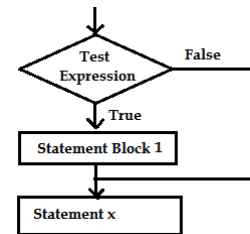
The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not. These decision statements include:

1. If statement
2. if-else statement
3. if-else-if statement
4. switch statement

1. if statement: The simplest form of decision control statement that is generally used in decision making. It executes the specified set of instructions based on the condition. The general syntax of simple 'if statement' follows.

Syntax of if statement:

```
if (condition)
{
    statement1;
    .....
    statementn;
}
```



Working: if the **condition** is true, then the statement block will be executed. Otherwise the statement block will be skipped and the execution will jump to the next statement after the statement block.

The **statement - block** may be a single statement or a group of statements. If there is only a single statement in the statement block, there is necessity of curly braces ({}).

if(expression)

```
{
    statements
}
class SimpleIf
{
    public static void main(String[] args)
    {
        if(5>6)
        {

            System.out.println("Hello !");
            System.out.println("Welcome to JAVA world");
        }
        System.out.println("This is Mahesh!"); }
}
```

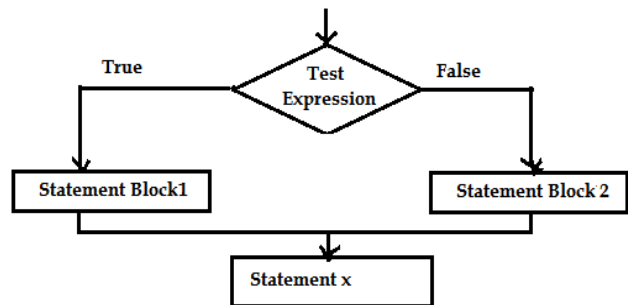
2. **if-else statement:** Using else is always optional.

- ☞ In implementation we having alternate block of condition then go for else part.
- ☞ When we are constructing the 'else' part mandatory to place 'if' part also. Because without if there is no else among those if and else only one block will be executed.

i-e. When if condition false then only else part will be executed.

Syntax:

```
if(condition)
{
    statement1;
    statement2;
}
else
{
    statement3; }
```



Working: if the **Condition** is true then the statement – block 1 will be executed. Otherwise the statement – block2 will be executed.

```
class Exifelse
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
        if(2>3)
        {
            System.out.println("Welcome !");
            System.out.println("Hello!");
        }
        else
        {
            System.out.println("Mahesh!");
            System.out.println("MCA!");
        }
    }
}
```

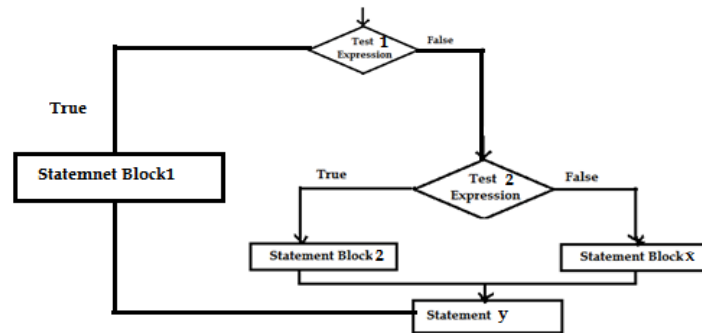

3. if-else-if statement (Nested if else): If it is a concept of placing a condition part within another condition. We can construct 'n' number of nested blocks but any type of editor can support up to 255 blocks.

Syntax:

```

if(condition1)
{
    block 1;
}
else if(condition2)
{
    block2;
}
else
{
    block3;
}

```



Ex: Class Nesting

```

{
public static void main(String args[])
{
    int a=3,b=7,c=4;
    Sytem.out.print("Larget value is : ");
    if( a>b)
    {
        if(a>c)
        System.out.println(a);
        else
        System.out.println(b);
    }
    else
    {
        if (b>c)
        System.out.println(b);
        else
        System.out.println(c);
    }
}
}

```

4) Switch case (case control structures)

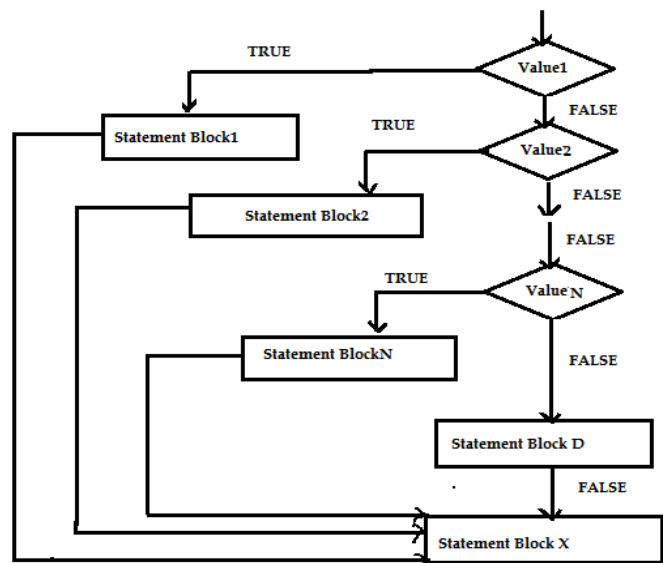
By using series of 'if' statements we can make a selection in a number of alternatives, but it is too complicated to develop. Then go for '**switch statement**' or **case control statement**.

- ☞ Switch is a keyword. By using switch case we can create selection statements with multiple choices.
- ☞ Multiple choices can be created by using case keyword.
- ☞ When we are working with switch it require condition or expression of type integer only
- ☞ When we are working with switch all case constant values will be mapping switch condition return value.
- ☞ At the time of execution if condition return values is match with anyone of the cases then from matching is up to break, everything will be executed, if the break is not exist including default all cases will be executed.
- ☞ A default is a special kind of case which will execute automatically when the matching case is not occurred. Using default is always optional, in implementation when we are not handling all cases of the switch block then recommended go for default.

Syntax to switch case

```
switch(condition)
{
    case const1: Block1;
        break;
    case const2: Block 2
        break;
    case const3: Block 3;
        break;
    default: block n;
}
```

```
class SwitchDemo{
    public static void main(String[] args){
        char grade = 'B';
        switch ( grade )
        {
            case 'A':
                System.out.println( " Excellent ! " );
                break ;
            case 'B':
                System.out.println( " Outstanding ! " );
                break ;
            case 'C':
                System.out.println( " Good ! " );
        }
    }
}
```



```
        break ;
    case 'D':
        System.out.println( " Can do better " );
        break ;
    case 'E':
        System.out.println( " Just passed " );
        break ;
    case 'F':
        System.out.println( " You failed " );
        break ;
    default :
        System.out.println( " Invalid grade " );
    }
} }
```

2. Iteration Statements: Set of instructions give to the compilers to execute set of statements until condition became false that is loop.

The way of repetitions are forms the circle, that's why iteration statements are called loops. i.e. loops means forming a circle. The basic purpose loop is code repetition.

In implementation whenever the repetition are occurred when instance of writing statement go for loops.

In JAVA programming language loops are classified into **three** types.

1. While
2. For
3. Do-while

While loop: This loop repeatedly executes a group of statements as long as condition is true. When we are working with while loop always pre-checking process will occur and it repeats in clock direction.

Syntax to while

```
While(condition)
{
    Statement 1;
    Statement 2;
    .....
    .....
    Statement n;
    inc /dec;
}
```

Working: The 'while' statement executes all the statements in the statement-block as long as the condition returns true (non-zero) value. When the condition returns false (0) then the loop is terminated and the control is transferred to the statement after the statement-block.

Write a program to display 1 to 10 numbers using while loop.

```
import java.io.*;
class Exwhile
{
    public static void main(String args[]) throws Exception
    {
        int i=1;
        System.out.println("list of 1 to 10 numbers");
        while(i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

For loop: - It is an entry controlled loop. When we are working with for loop it contains three parts.

1. Initialization
2. Condition
3. Iteration

Syntax to for loop:

```
for(initialization; condition; iteration)
{
    Statement block;
}
```

Working:

- ☞ The execution process of the for loop always starts from initialization
- ☞ Initialization will be executed only once when we are entering into the loop first time
- ☞ After execution of the initialization block control will pass to conditional block, if the condition is evaluated as true then control will pass to statement block.
- ☞ After execution of the statement block control will pass to iteration block, after execution of the iteration, once again the control will pass to the condition.
- ☞ Always the repetitions will happen between condition statement block and iteration only.
- ☞ When we are working with for loop it repeats in anti clock direction

Write a JAVA program to print 1 to 10 numbers.

```
import java.io.*;
class Exfor
{
    public static void main(String args[]) {
        int i;
        System.out.println("list of 1 to 10 numbers");
        for(i=1;i<=10;i++) {
            System.out.println(i);
        }
    }
}
```

```
}
```

The Enhanced for loop:

The enhanced **for** loop is also called **for each loop**, is an extended language feature introduced with J2SE 5.0. This feature is used to retrieve the array elements efficiently rather than using array indexes.

Syntax:

```
for( type identifier : Array Name )
{
    statement-block;
}
```

In the above example “**type**” represents the data type, **identifier** represents the name of the variable, Array Name indicates an Array.

For each loop

```
import java.util.*;
class ForEachloop
{
    public static void main(String args[] ) {
        int a[]={1,3,5,7,9};
        for(int i:a)
        {
            System.out.println(i+" ");
        }
    }
}
```

Do-while:

In implementation if at least need to be executed the statement block once. Then go for do-while. When we are working with do-while always post checking process will occur I.e. after execution of the statement block only condition will be evaluated.

Syntax to do-while

```
do
{
    Statement 1;
    Statement 2;
    Statement 3;
    Inc/dec;
}
While (condition);
```

Note:- in do-while loop mandatory to place semicolons after while.

Write a program to display 1 to 10 numbers using do-while loop.

```
import java.io.*;
class ExDowhile
{
```

```
public static void main(String args[])
{
    int i=1;
    System.out.println("list of 1 to 10 numbers");
    do
    {
        System.out.println(i);
        i++;
    }
    while(i<=10);
}
```

Break and continue (Jumping or Skipping) statements:

Branch control statements are used to transfer the control from one place to another place. Java language provides three branch control statements. Those are

- i) break statement ii) continue statement iii) return statement

break statements:

- ☞ 'break' is a keyword by using break we can terminate loop body of switch body.
- ☞ Using break is always optional. But it should be exist within the loop body or switch body only.
- ☞ In implementation where we know maximum number of repetitions but certain condition is there we need to stop the repetition process, and then go for break. A break is usually associated with an 'if'.

Syntax: break;

Write a JAVA Program to demonstrate break

```
class BreakExample1
{
    public static void main(String[] args)
    {
        int i=1;
        while(i<=10)
        {
            if(i==6)
                break;
            System.out.println("Mahesh");
            i=i+1;
        }
    }
}
```

The 'continue' statement:

- ☞ Continue is a keyword by using 'continue' we can skip some of statements from loop body.
- ☞ Using continue always optional. But it should be within the loop body only.
- ☞ In implementation where we know maximum number of repetitions but certain conditions are there where we need to skip the statement block of loop body then go for continue.
- ☞ A 'continue' is usually associated with an 'if'.

Syntax: continue;

Write a JAVA Program to demonstrate continue statement.

```
class ContinueExample1
{
    public static void main(String[] args)
    {
        int i;
        for(i=1;i<=10;i++)
        {
            if(i==5)
                continue;
            System.out.println(i);
        }
    }
}
```

Write a JAVA Program to demonstrate continue statement.

```
public class ContinueExample
{
    public static void main(String[] args) {
        int intArray[] = new int[] {1,2,3,4,5};

        System.out.println("All numbers except for 3 are :");
        for(int i=0; i < intArray.length; i++)
        {
            if(intArray[i] == 3)
                continue;
            else
                System.out.println(intArray[i]);
        }
    }
}
```

Return statement:

- ☞ Return statement is used to explicitly return from a method.
- ☞ Return causes program control to transfer back to the caller of the method. The return statement immediately terminates the method in which it is executed.
- ☞ Return statement does not terminate any method only main method will terminate.

Syntax:

return;

Ex: return x value;

return -1;

return (x+y-10);

return arr;

return obj;

Write a java program to demonstrate the effect of return statement.

```
class Return1
{
    public static void main(String args[])
    {
        int num=1;
        System.out.println("Before the return.");
        if(num==1) return; // return to caller
        System.out.println("After return.");
    }
}
```


CLASSES, OBJECTS & METHODS

1. Discuss about class and object

Class:

- A **class** is a group of objects that share common properties and relationships.
- A class is user-defined data type which holds both *data* and *member functions*
- Class is structure representing the data and its related and corresponding functions.
- For example Mango, Apple and Orange are objects of the class “Fruit”.

Fruit Mango;

Class declaration

In java, the data items are called **fields** and functions are called **methods**. Once a class is defined, we can create any number of objects belonging to that class. In java these “variable” are called as *instances* of classes. Classes are defined using the keyword “class”.

Syntax :

```
class <classname >
{
    [fields declaration ; ]
    [methods declaration ;]
}
```

Fields Declaration :

The variables which are declared inside a class are called **fields**. These variables are also called **instance variables** because they are created whenever an object of the class is instantiated.

Ex: class Rectangle

```
{
    int length;
    int width;
}
```

The class Rectangle contains two integer type instance variables. No memory space is reserved for these instance variables.

Methods Declaration

Methods are declared inside the body of the class immediately after the declaration of instance variables. The general form is

Syn: **datatype methodname(parameter-list)**

```
{
    Body of the method;
}
```

Here, Method declarations have four parts.

- The name of the method.
- The type of the value the method returns.
- A list of parameters.
- The body of the method.
- The *method name* is valid identifier.
- If the method does not return any value, the return type will be **void**.
- The *parameter-list* is always enclosed in a parenthesis, contains variable names and type of the values we want to give the method as input. The variables in the list separated by commas.

Eg: class Rectangle

```
{  
  
    int length, width;  
  
    void getdata(int x, int y) //Method declaration  
  
    {  
  
        length=x;  
        width=y;  
    }  
    int area()  
    {  
        int a=length*width;  
        return(a);  
    }  
}
```

Object:

Instance of a class is called an object. Objects are the basic run-time entities in an object Oriented system which can represent a person, a place, a bank account (or) a table of data etc which have state and behavior. Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Creating Objects

Creating an object is also referred as *instantiating* an object. Objects in java are created using the **new** operator.

```
Rectangle r1;  
  
r1=new Rectangle();
```

We can combined both the statements in one as

```
Rectangle r1=new Rectangle();
```

We can declare any number of objects of Rectangle class.

```
Rectangle r1=new Rectangle();
```

```
Rectangle r2=new Rectangle();
```

Accessing members of a class

The class members can be accessed using the **dot(.)** operator associated with object. We cannot access the member variables and methods directly outside the class.

```
objectname.variable=value;
```

```
object.methodname(parameter-list);
```

Here, *objectname* is the name of the object and *variable* is the name of the member variables, *methodname* is the name of the method that we call, and *parameter-list* is separated by commas of actual values that must match in the type and number with parameter list of method declared in the class.

```
r1.length=4;
```

```
r1.width=6;
```

Another way of assigning values to the instance variables by using methods declared inside the class.

```
r1.getdata(4,6);
```

Eg: class Rectarea

```
{  
    public static void main(String args[])  
    {  
        Rectangle r1=new Rectangle();  
        Rectangle r2=new Rectangle();  
        r1.length=4;  
        r1.width=6;  
        r2.getdata(10,5);  
        int area1=r1.length*r1.width;  
        int area2=r2.area();  
        System.out.println("Rectangle R1 area="+area1);  
        System.out.println("Rectangle R2 area="+area2);  
    }  
}
```

What is a constructor? What are its special properties?**Constructors**

- Constructors have the same name as the class itself.
- Constructors do not specify a return type.
- Constructors are invoked automatically, when the objects are created.
- The constructor without arguments is called as default constructor.
- The constructor with arguments is called as parameterized constructor.

Eg: class Rectangle

```
{
    int length, width;
    Rectangle(int x, int y)           //Constructor method
    {
        length=x;
        width=y;
    }
    int area()
    {
        int a=length*width;
        return(a);
    }
}

class Rectarea
{
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle(4,6);
        int area=r1.area();
        System.out.println("area="+area);
    }
}
```

What is method overloading ? Give an example.

- In java, it is possible to create methods that have same name, but different parameters list. This is called *method overloading*.
- Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, it matches the method name first and then the number and type of parameters is to decide which definition to execute. This process is known as *polymorphism*.

Eg: class Room

```
{  
    float l,b;  
    Room(float x, float y)           //Constructor1  
    {  
        l=x;  
        b=y;  
    }  
    Room(float x)                   //Constructor2  
    {  
        l=b=x;  
    }  
    float area()  
    {  
        return(l*b);  
    }  
}
```

Write about Nesting of Methods

A method can be called by using only its name by another method of the same class is known as *nesting of methods*.

Eg: class Big

```
{  
    int a,b;  
    Big(int x, int y)  
    {  
        a=x;  
        b=y;  
    }  
}
```

```
        int largest()
        {
            if(a>b)
                return a;
            else
                return b;
        }
        void display()
        {
            int large=largest();
            System.out.println("Big="+large);
        }
    }
    class Biggest
    {
        public static void main(String args[])
        {
            Big t= new Big(5,7);
            t.display();
        }
    }
```

In the above example class **big** defines one constructor and two methods, namely **largest()** and **display()**. The method **display()** calls the method **largest()** to determine the largest number and display the result.

Defining a Subclass

Syn: **class** subclassname **extends** superclassname

```
{
    variables declaration;
    members declaration;
}
```

The keyword **extends** signifies that the properties of *superclass* extends to the *subclass*. The subclass may have its own variables and methods as well as those of the superclass.

UNIT-III

6) What is inheritance? Write benefits of inheritance

Inheritance: The mechanism of deriving a new class from an old one is called inheritance. In which new class is called subclass or derived class and old class is called super class or base class.

Advantages of INHERITANCE: Benefits of Inheritance:

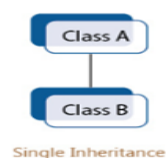
1. Application development time is very less.
2. Redundancy (repetition) of the code is reducing. Hence we can get less memory cost and consistent results.
3. Instrument cost towards the project is reduced.
4. One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses, where equivalent code exists in two related classes. This also tends to result in a better organization of code and smaller, simpler compilation units.
5. Inheritance can also make application code more flexible to change because classes that inherit from a common super class can be used interchangeably.
6. Reusability-> facility to use public methods of base class without rewriting the same
7. Extensibility-> extending the base class logic as per business logic of the derived class
8. Data hiding-> base class can decide to keep some data private so that it cannot be altered by the derived class
9. Overriding-> With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

7) Explain about different types of inheritance with example.

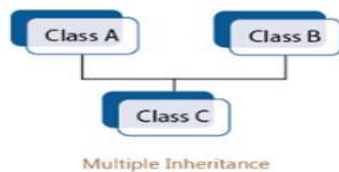
Types of inheritance:

- 1) Single Inheritance
- 2) Multiple Inheritance
- 3) Multi-level Inheritance
- 4) Hierarchical Inheritance
- 5) Hybrid Inheritance/ Multipath Inheritance

Single inheritance: A class is derived from only one super class and that super class contains only one subclass then it is known as single inheritance.

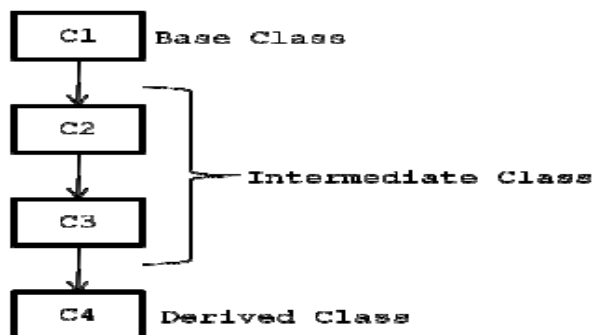


Multiple inheritances: A class is derived from two or more super classes then it is known as multiple inheritances.

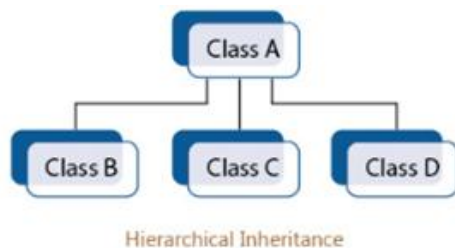


Multilevel

inheritance: - A class is derived from any other derived class is known as multilevel inheritance.

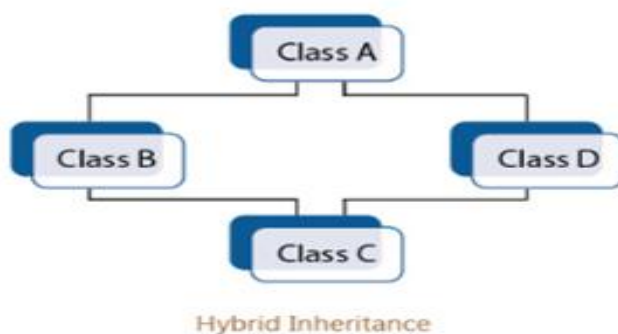


Hierarchical inheritance: A super class contains two or more sub classes then it is known as hierarchical inheritance.



Hybrid inheritance: - Applying of two or more types of inheritances is known as hybrid inheritance.

Hybrid inheritance = combination of any available inheritances types.



Write about Visibility Controls (or) access specifiers?

It is possible to inherit all the members of a super class to its subclass by using the keyword **extends**. However, it may be necessary to restrict the access to certain variables and methods from outside the class.

In java we achieve this by applying visible modifiers to the member variables and methods. The visibility modifiers are also called as access modifiers such as **public**, **private** and **protected**.

Public: If we declare variables and methods as **public**, then these variables and methods are accessible to all the classes outside this class.

Private: The members which are declared in private section can be accessed only from within the class. By default all members are private.

Public: The members which are declared in public section can be accessed from both inside and outside of the class.

Protected: The protected members are also same as private members. But protected members can be also directly accessed in the subclass.

Friendly Access

When no access modifier is specified, then the members and methods defaults *friendly*.

Private Protected:

A member can be declared with two keywords **private** and **protected**. It is accessible only the subclasses in same package and other package. But not accessible other classes in same package and other packages.

Access Modifiers	Public	Friendly	Protected	Private	Private Protected
Access Location					
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	No	Yes
Other class in same package	Yes	Yes	Yes	No	No
Subclass in other package	Yes	No	Yes	No	Yes
Other class in other package	Yes	No	No	No	No

ARRAYS, STRINGS AND VECTORS:

ARRAYS

Introduction to Arrays

Definition: An array is a collection of memory locations which can share same data name and same data type values.”

Syntax:

Data type array name [size];

Properties:

- ☞ An array is a collection of similar data type values in a single variable.
- ☞ In implementation when we require ‘n’ number of similar data type values then go for array.
- ☞ The array variables are created at the time of compilation.
- ☞ When we are working with arrays all elements will allocated in continuous memory location only.
- ☞ When we are working with arrays we can access the elements randomly. Because continuous memory locations are created.
- ☞ When we are working with arrays all elements will share same name with unique identification value called index. Which starts from **zero** and ends with ‘size-1’?
- ☞ When we are working with arrays we need to use array subscript operator i.e. []

What is Array? Explain different types of arrays with Examples?

Definition: An array is a collection of memory locations which can share same data name and same data type values.”

In **JAVA** language, arrays are classified into two types. They are

1. One Dimensional Arrays
2. Two Dimensional Arrays

One dimensional array:

When an array is declared with only one dimension (subscript) then it is called “One dimensional array” or “single dimensional array”.

Declaring one dimensional Array:

The general format of declaring a one dimensional array is as follows.

Syntax: data type array_name[] ;

Format: datatype arrayname[];

Format: datatype[] arrayname;

Eg: int a[];

```
int[] a;
```

Note: We do not enter the size of the arrays in the declaration.

Array initialization:

- After array creation we can put values into the array, this process is known as *initialization*.
- We can initialize the elements of the array in the same way as the ordinary variables when they are declared. The general form of initialization of array is:
- <Data type> array name [size] = {value1, value2, ..., values};
- Where the value1 refers to the first array element, value2 refers to the value of the second array element. ...so on.

Examples of initializing arrays:

```
int num[5]={10,20,30,40,50};
```

```
float cost [4]={0.45, 1.25, 3.75,0.35};
```

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
22	56	89	10	99	43

Syn: arrayname[index]=value;

Eg: a[0]=10;
a[1]=12;
a[2]=14;
a[3]=16;
a[4]=18;

Creating memory locations: After the declaration, we need to create the memory for arrays. Java allows creating arrays using **new** operator.

Syn: arrayname=**new** datatype[size];

Eg: a=new int[5];

Here, the variable **a** refers to an array of 5 integers. It is also possible to combine the declaration and creation of arrays.

Eg: int a[]=**new** int[5];

We can also initialize arrays automatically in the same way as the ordinary variables when they are declared.

Syn: datatype arrayname[]={val1, val2, valN};

Eg: int a[]={10,12,14,16,18};

In the above example the list of values are separated by commas and enclosed within the braces.

/* program for demonstrate array initialization and printing array elements*/

```
class ArrEx1
{
    public static void main(String args[] )
    {
        int arr[]={10,20,30,40,50};
        System.out.println("start array"+arr[0]);
        System.out.println(arr[1]);
        System.out.println(arr[2]);
        System.out.println(arr[3]);
        System.out.println(arr[4]);
    }
}
```

Two dimensional arrays?

Two Dimensional Arrays

Collection of single dimensional array in a single variable is called two dimensional array or array (array).

- ☞ In 2D array elements are arranged in rows and columns.
- ☞ When we are working with 2D array we need to use two subscript operators which indicate **row size** and **column size**.
- ☞ The main memory of 2D array is row and sub memory is columns.
- ☞ In '**2D array**' array name always gives base address of the array i.e. first row base address, arr+1 is next main memory of the array. i.e. second row base address.

	0	1	2
0	[0][0]	[0][1]	[0][2]
1	[1][0]	[1][1]	[1][2]

- We may create a two dimensional array with the statements.

```
int a[][];           //declaration
```

```
a=new int[2][3];     //creation
```

(or)

```
int a[][]=new int[2][3];
```

This creates a table with 2 rows and 3 columns. i.e. it stores 6 integers.

- A two dimensional array may be initialize
`int a[2][3]={0,0,0},{1,1,1};`

Array length

We can obtain the length of the array **a** using **a.length**.

Eg: `int a[]={10,12,14,16,18};`
`int size=a.length;`

Prog1:

```
import java.util.Scanner;
class A4{

    public static void main(String[] args){
        float[][] s = new float[3][2];
        Scanner s1 = new Scanner(System.in);

        for(int i = 0; i < 3; i++)
        {

            System.out.println("Enter marks of student : "+(i+1));
            for(int j = 0; j < 2; j++)
            {
                System.out.println("Subject : "+(j+1));
                s[i][j] = s1.nextFloat();
            }

        }

        for(int i = 0; i < 3; i++)
        {

            System.out.println("Student" + (i+1));

            for(int j = 0; j < 2; j++)
            {
                System.out.println("Subject" + (j+1) + ":" + s[i][j]);
            }

        }
    }
}
```

STRINGS

What is a string? How to use it?

- A string is the sequence of characters enclosed within double quotes. Strings are very useful because most of the data on internet will be in the form of strings only.
- **For example:** name, vehicle number, address, credit card number etc. will come under strings.
- In order to deal with strings we have two classes. They are java.lang.String and java.lang.StringBuffer.
- A java string is not a NULL terminated.

☞ 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[] = { 'g','m','i','n','f','o','r','m','a','t','i','c','s'};
```

Strings may be declared and created as follows:

```
String stringname;
```

```
stringname=new String("string");
```

Eg: 

```
String college;
```

```
college=new String("gminformatics");
```

These two statements may be combined as follows:

Eg: 

```
String college=new String("gminformatics");
```

- Like arrays, it is also possible to get the length of the string using the **length** method of the class **String**.  

```
int m=college.length();
```
- Java strings can be concatenated using '+' operator.  

```
String college="gm"+"informatics";
```

## String methods

| Method Call             | Task Performed                                                              |
|-------------------------|-----------------------------------------------------------------------------|
| s2=s1.toLowerCase       | Converts the string s1 to all lowercase.                                    |
| s2=s1.toUpperCase       | Converts the string s1 to all uppercase.                                    |
| s2=s1.replace('x','y')  | Replace all appearance of x with y.                                         |
| s1.equals(s2)           | Return true, if s1 is equal s2.                                             |
| s1.equalsIgnoreCase(s2) | Returns true, if s1=s2, ignoring the case of characters.                    |
| s1.length()             | Gives the length of s1.                                                     |
| s1.charAt(n)            | Gives the nth character of s1.                                              |
| s1.compareTo(s2)        | Returns negative, if s1<s2. Returns positive, if s1>s2, and zero, if s1=s2. |

**STRINGBUFFER CLASS**

- A **StringBuffer** implements a sequence of characters. A **StringBuffer** is like a **String**, but can be modified in length and content.
- It creates 16 extra memory locations for a string.
- We can insert characters and substrings in the middle of the string (or) append another string to the end.

| Method Call         | Task Performed                   |
|---------------------|----------------------------------|
| s1.setCharAt(n,'x') | Modifies the nth character to x. |

|                              |                                                                                                                                                    |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>s1.capacity()</code>   | Gives the capacity of s1.                                                                                                                          |
| <code>s1.append(s2)</code>   | Appends the string s2 to the end of s1.                                                                                                            |
| <code>s1.reverse()</code>    | Gives the reverse string of s1.                                                                                                                    |
| <code>s1.insert(n,s2)</code> | Inserts the string s2 at the position n in the string s1.                                                                                          |
| <code>s1.setLength(n)</code> | Set the length of the string s1 to n. if <code>n&lt;s1.length()</code> it is truncated. If <code>n&gt;s1.length()</code> , zero's are added to s1. |

### VECTORS

#### What is a vector? Explain the difference between a vector and an array?

- The vector class can be used to create a generic dynamic array known as *vector*. It can hold the elements of different type.
- The **vector** class is contained in **java.util** package.
- Vectors are created like arrays as follows:

**Vector** a=new Vector(); //declaring without size.

**Vector** b=new Vector(3); //declaring with size.

- A vector without size can store any number of items.
- Vectors can have number of advantages over arrays.
  1. It is convenient to use vectors to store elements.
  2. A vector can be used to store a list of elements that may be different items.
  3. We can add and delete elements from the vector list when required.
  4. A major constraint in using vectors is that we cannot directly store a simple data types in a vector, we can store only through objects.
  5. The vector class supports a number of methods that can be used to manipulate the vectors.



| Method Call                               | Task Performed                                           |
|-------------------------------------------|----------------------------------------------------------|
| <code>list.addElement(item)</code>        | Adds the item at the end of the list.                    |
| <code>list.elementAt(n)</code>            | Gives the name of the nth object.                        |
| <code>list.size()</code>                  | Gives the number of objects.                             |
| <code>list.removeElement(item)</code>     | Removes the specified item from the list.                |
| <code>list.removeElement(n)</code>        | Removes the item stored in the nth position of the list. |
| <code>list.removeAllElement()</code>      | Removes all elements in the list.                        |
| <code>list.copyInto(array)</code>         | Copies all items from vector to array.                   |
| <code>list.insertElementAt(item,n)</code> | Inserts the item at nth position.                        |

**What is an interface? How to extending and implementing interfaces?**

- Java supports *interfaces* instead of multiple inheritances.
- A class can *implement* with more than one interface to support multiple inheritance.
- An interface is a named collection of method declaration (without implementations).
- An interface is similarly a class, it contains methods and variables. But the difference is, it have only abstract methods and final variables.
- The interface do not specify any code to implement these methods, it is the responsibility of the class that implement an interface.
- The syntax for defining an interface is similar to that for defining a class.

Syn: **interface** interfacename

```
{

 Variables declaration;

 Methods declaration;

}
```

In the above syntax, **interface** is the keyword and *interfacename* is any valid java identifier.

- Variables are declared in the interface is as follows

Syn: **final** datatype variablename=value;

- Methods are declared in the interface is as follows

Syn: **public returntype methodname**(parameter\_list);

Eg: **interface** item

```
{

 static final int CODE=1001;

 static final string NAMEe="tomy";

 public void display();

}
```

**Extending interfaces**

Like classes, interfaces can also be extended. That is the new sub interface will inherit all the properties of super interface. This is can be achieved by using the keyword **extends**.

Syn: **interface** name2 **extends** name1

```
{

}
```

Eg: interface item

```
{
 static final int CODE=1001;
 static final string NAME="tomy";
 public void display();
}

interface itemcost extends item
{
 public void showcost();
}
```

The interface **itemcost** will inherit both the variables **CODE** and **NAME** into it. We can also extends several interfaces together into a single interface.

interface item

```
{

}
```

interface discount

```
{

}
```

```


 }

 interface itemcost extends item,discount
 {

 }

```

### Implementing interface

Interfaces are used as “superclasses” whose properties are inherited by classes. Therefore it is necessary to create a class that inherits the given interface.

Syn: **class** classname **implements** interfacename

```

{
 body of the class;
}

```

Here, the class *classname* implements the interface *interfacename*. A class can also extends another class while implementing interfaces.

Syn:

**class** classname **extends** superclassname **implements** interface1, interface2, ....

```

{
 body of the class;
}

```

### Accessing interface variables:

Interfaces can be used to declare a set of constants that can be used in different classes. The constants values will be available to any class that implements the interface. The values can be used in any method or any variable declaration.

```
Eg: interface A
 {
 int m=10;
 int n=20;
 }
 class B implements A
 {
 int x=m;

 void demo(int size)
 {

 if(size<n)

 }
 }
```

## UNIT –IV

### What is a thread? How to create a thread?

#### Definition

A thread is said to be a sub process.

- ☞ process is nothing but a program under execution.
- ☞ If a process (application) has multiple sub processes (threads), such application is known as a multithread application.

Mostly used in web applications, Servlets, JSP

Modern operating system can execute several programs simultaneously. This ability is known as multitasking. In system's terminology it is called multithreading.

#### Creating Threads

- Creating threads in Java is simple. Threads contain a method called **run()**, and are implemented in the form of objects.
- The **run()** method makes up the entire body of a thread..
- A **run()** method would appear as follows:

```
public void run()
{

}
```

- The **run()** method should be involved by an object of the thread. This can be achieved by creating the thread and initiating with the help of another thread method called **start()**.
- A new thread can be created as, defining a class that extends **Thread** class and override its **run()** method with the required code.

#### Extending the Thread class

We can make our class runnable as thread by extending the class **java.lang.Thread**. This gives access to all the thread methods directly. It includes the following steps

- 1) *Declaring the class:* The **Thread** class can be extended as follows

```
class A extends Thread
{

}
```

Here, **A** is a new type of Thread.

- 2) *Implementing the run() method:*

```
public void run()
{

 //Thread code
}
```

We have to override this method in order to implement the code to be executed by our thread.

- 3) *Starting the Thread:* To create an object and run the thread class, we must write following statement.

```
A t1=new A();
t1.start(); //accessing run() method
```

Now, the thread is said to be in the running state.

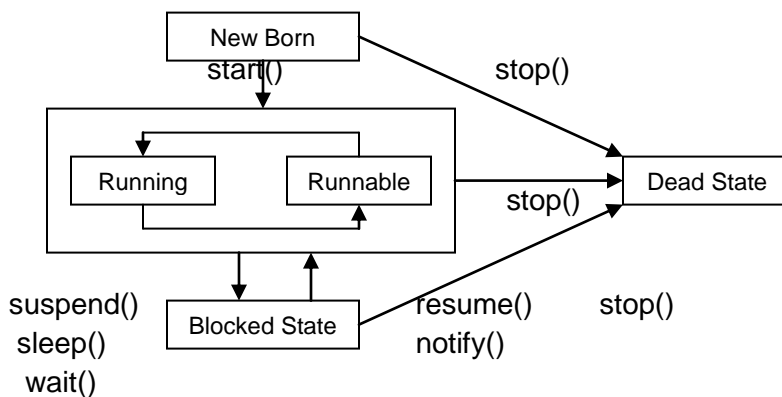
### Explain the life cycle of a thread?

#### The Life Cycle of a Thread

During the lifetime of a thread there are many states. They include.

- 1) Newborn state.
- 2) Runnable state.
- 3) Running state.
- 4) Blocked state.
- 5) Dead state.

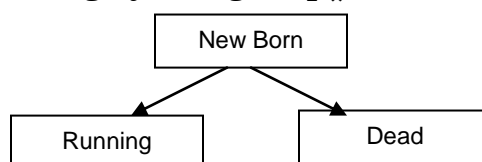
A thread is always is one of the 5 states. It can move from one state to another in the following way.



#### Newborn state:

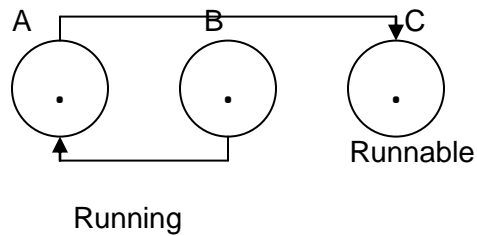
When we create an object of a thread, then the thread is said to be in *newborn* state. At this state, we can do any one of the following.

- 1) For running by using **start()** method.
- 2) For deleting by using **stop()** method.

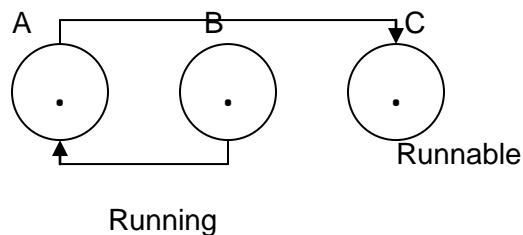


**Running state:**

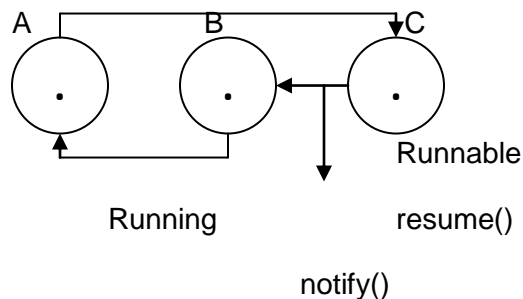
The thread is executing by the processor.

**Runnable state :**

The *runnable* state means that the thread is ready for execution and is waiting for the processor. That is the thread has joined in the queue.

**Blocked state :**

A thread is said to be *blocked* from the running state and runnable state. This happens when the thread is suspended, sleeping or waiting. A blocked thread is considered as “not runnable”, but not dead so it is run once again.

**Dead state:**

Every thread ends its life when it has completed executing its **run()** method. It is a natural dead. However, we can kill a thread by a **stop()** method at any state, so it is a premature dead.



**What is an error handling? Explain different types of error in java?**

A mistake may lead to an error causing a program to produce unexpected results. An error may produce an incorrect output or may terminate the execution of the program.

**Types of Errors:**

Errors may be classified into two types. They are

- Compile-time errors
- Run-time errors

Compile-time Errors:

All syntax errors will be detected and displayed by the java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the **.class** file. Therefore it is necessary to fix the errors for to successfully compile and run the program.

```
class Error1
{
 public static void main(String args[])
 {
 System.out.println("Welcome to Java") //Missing
 }
}
```

The compiler displays the following message on the screen.

```
Error 1.java:5: ; expected
System.out.println("Welcome to Java")
```

Sometimes, a single error may be the source of multiple errors in the compile-time. For example, an undeclared variable in a number of places will cause a series of errors of type “undefined variable”. The most common errors are

- Missing semicolon
- Missing double quotes in strings.
- Use of undeclared variable.
- Misspelling of identifiers and keywords.
- Missing brackets and methods.
- Incompatible types in assignments.
- Use of ‘=’ in the place of ‘==’ operator.

Run-time Errors:

Sometimes, a program may compile successfully creating the **.class** file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate. The most common Run-time errors are

- Dividing an integer by 0(zero).

- Accessing an element that is out of bounds of an array.
- Trying to store a value into an array of an incompatible class.
- Trying to cast an instance of a class to one of its subclass.
- Trying illegally to change the state of a thread.
- Attempting to use a negative size for an array.
- Converting invalid string to a number.

When such errors are encountered, java generates an error message and aborts the program.

```
class Error2
{
 public static void main(String args[])
 {
 int a,b,c;
 a=10;
 b=5;
 c=5;
 int x=a/(b-c);
 System.out.println("x="+x);
 int y=a/(b+c);
 System.out.println("y="+y);
 }
}
```

While executing, it displays following message and terminate the program without executing remaining statements.

```
java.lang.ArithmeticException : / by zero
at Error2.main
```

### What do you mean by exception? How to handle exceptions in java?

An *exception* is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing by zero, which creates an exception object and throws it.

If the exception object is not caught and handled properly, the interpreter to display an error message and terminate the program. If we want the program to continue with the execution of remaining code, then we should introduce the new concept *exception handling*.

When writing programs, we must always lookout the places where an exception could be generated. Some of the exceptions we must watch out for catching.

| Exception Type                  | Cause of Exception                                                                         |
|---------------------------------|--------------------------------------------------------------------------------------------|
| ArithmeticException             | Caused by math errors such as division by zero.                                            |
| ArrayIndexOutOfBoundsException  | Caused by bad array index.                                                                 |
| ArrayStoreException             | Caused when a program tries to store wrong type of data in an array.                       |
| FileNotFoundException           | Caused by an attempt to access a not-existent file.                                        |
| IOException                     | Caused by general input output failures.                                                   |
| NullPointerException            | Caused by referencing a null object.                                                       |
| NumberFormatException           | Caused when a conversion between strings and number fails.                                 |
| OutOfMemoryException            | Caused when there is not enough memory to allocate a new object.                           |
| SecurityException               | Caused when an applet tries to perform an action not allowed by browser security settings. |
| StackOverflowException          | Caused when the system runs out of stacks pace.                                            |
| StringIndexOutOfBoundsException | Caused when a program attempts to access a non-existent character position in a string.    |

## UNIT – V

### What is an applet? What do you mean by remote and local applets?

Applets are small java programs that are basically used in internet programming. They can be transported over the internet from one computer to another and run using appletviewer” (or)any web browsers that supports java .An applet can perform the following tasks.

Arithmetic operations, play sounds, display graphics, create animation, and accept user input and interactive games.

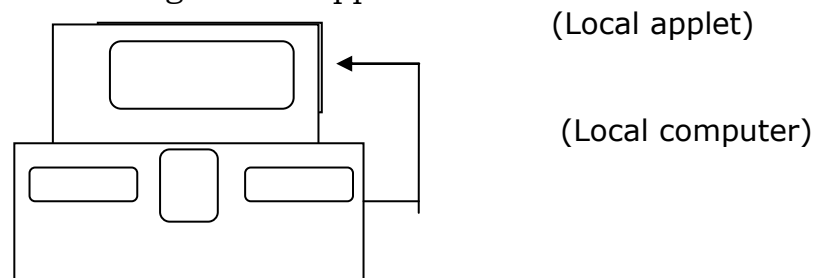
We can embed applets into web pages in two ways. They are

1. Local applets
2. Remote applets

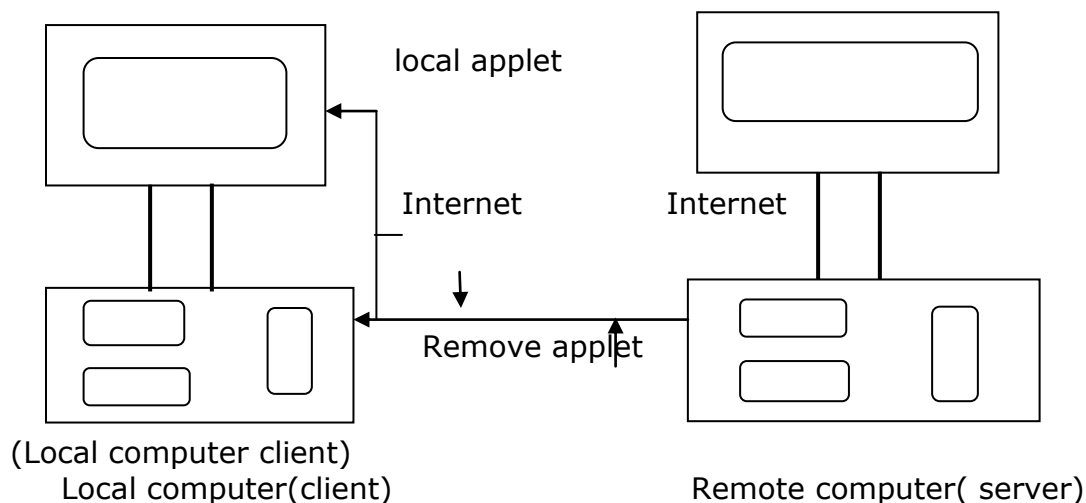
#### **Local applets:-**

An applet developed locally and stored in a local system is known as a “local applet”. We can write our own applets and embedded them into the web pages, local applet does not need to use the internet. It simply searches the directories in the local system and locates and loads the specified applet.

The following shows the loading of local applet.



**Remote applet:** - A remote applet is that which is developed by someone else and stored on a remote computer connected to the internet. If our system is connected to the internet we can download the Remote applet on to our system via at the internet and run it. The following diagram shows the loading a remote applet.



In order to locate and load a remove applet ,we must know the applet address on the web. This address is known as “Uniform Resource Locater”(URL) and must be specified in the applets HTML documents as the value of the CODE BASE ATTRIBUTE

CODE BASE = <http://www.netserve.com/applets>

### **How Applets Differ From Applications:-**

Both the applets and standard alone applications are java programs there are significant differences between them they are:-

1. Applet do not use the main() method for start the execution of the code. Applets when loaded automatically call certain methods of “Applet” Class to start and execute the applet code.
2. Applets are executed from inside a webpage using a special feature known as HTML Tag.
3. Applet cannot read from or write to the files in the local computer.
4. Applets cannot run any program from the local computer.
5. Applets are restricted from using libraries from other languages such as C or C++.
6. Applets can’t communicate with other servers on the network.

### **How to Building an applet code:**

An applet code uses the services of two classes :

-> Applets

->Graphics

The **Applet** class which is contained in the “java.applet” package provides life and behaviour to the applet through its methods such as **init(),start() and paint()**.When an applet is loaded java automatically calls a series of Applet class methods for starting, Running and stopping the applet code. The applet class maintains the life cycle of a applet.

The paint() method of the applet class, is used to displays the result of the applet code on the screen. The output may be text, graphics (or) sound. The paint() method which requires a “**Graphics**” object as an argument is defined as follows.

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

The Graphics class is available in “java.awt.” package.The general form is as shown below.

Ex:

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

public class applet classname extends Applets
{

```

```

public void paint (Graphics g)
{
----- //Applets Operation Code.

}

-----}

```

Ex: Program:-

```

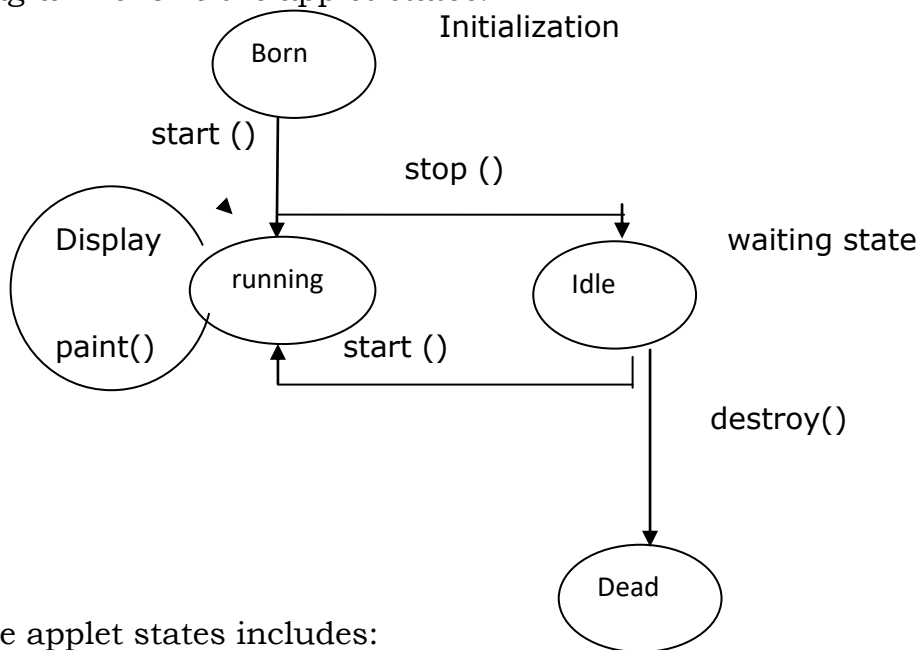
import java.awt.*;
import java.applet.*;
public void HelloJava extends Applet
{
 public void paint (Graphics g)
 {
 g.drawString ("Hello Java",10,100);
 }
}

```

### Explain the applet life cycle?

#### Applet Life Cycle:-

Every java applet inherits a set of default behaviors (methods) from the Applet class. When an applet is loaded, enters into different states. The following diagram shows the applet states.



The applet states includes:

- ☞ Born on initialization state
- ☞ Running
- ☞ Idle state

☞ dead or destroyed state

### **Initialization:-**

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

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

The initialization occurs only once in the applet Life cycle. To provide any of the behaviours mentioned above, we must Override the `init()` method.

```
public void init()
{

----- (Action)
}
```

### **Running state:-**

Applet enters the “Running state” when the system calls the `start()` method of Applet class. This occurs automatically after the applet is initialized. starting can also occur if the already in “stopped” (idle) state.

```
public void start()
{

----- (action)
}
```

### **Idle (or) Stopped state:-**

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page currently running applet. We can also calling enter in to the idle state by calling `stop()` method explicitly. If we use a thread to run the applet, then we must use `stop()` method to terminate the thread. We can achieve this by overriding the `stop()` method.

```
public void stop()
{

----- (action)
}
```

### **Dead state:-**

An applet is said to be “dead” state, when it is removed from memory. This occurs automatically by invoking the `destroy()` method. This occurs automatically when we quit the browser. Dead state also occurs only once in the applet Life cycle. If the applet has created any resources like threads, we may overrides the `destroy()` method to clean up these resources.

```
public void destroy()
{

----- (action)
```

```
}

```

**Display state:-**

Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to perform these task. Almost every applet will have a paint() method. We must override this method if we want anything to be displayed on the screen.

```
public void paint (Graphics g)
{

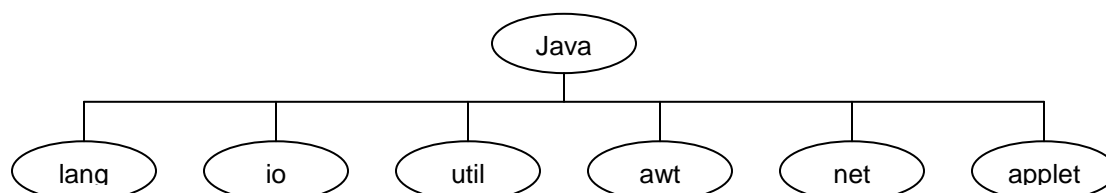
----- (display statements)
}
```

**What is a package? Explain different types of packages?**

A package is a collection of classes and interfaces. In java, all reusable code is put into packages. Java packages are classified into two types. They are *Java API packages* and *User defined packages*.

**Java API packages**

Java API provides a large number of classes grouped into different packages. The following figure shows the packages that are frequently used in the program.



| Method Call | Task Performed                                                                                                                                                                     |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| java.lang   | Language support classes. These classes are automatically imported by the java compiler. They include classes for primitive types, strings, math functions threads and Exceptions. |
| java.io     | Input/Output support classes. They provide facilities for the input and output of data.                                                                                            |



|             |                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------|
| java.util   | Language utility classes such as vectors, hash tables, random numbers, date and time etc.                                    |
| java.awt    | Set of classes for implementing Graphical User Interface (GUI). They include classes for Windows, buttons, lists, menus etc. |
| java.net    | Classes for networking. They include classes for communicating with local computers as well as with internet sources.        |
| java.applet | Classes for creating and implementing applets.                                                                               |

### Using System packages

There are two ways are accessing of classes stored in a package. They are

- The first approach is to use the *fully qualified classname*  
Syn: **import** java.packagename.classname;

For example, if we want to refer the class **String** in the **lang** package, then we use the statement.

Eg: **import** java.lang.String;

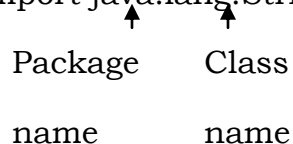
- The second approach is imports all the classes contained in the specified package. We may use the statement  
Syn: **import** java.packagename.\*;

Eg: **import** java.lang.\*;

### Naming conventions

- Packages can be named using the standard java naming rules. Packages begin with lower case letters. This makes easy to make difference between packages and classes. All class names begin with an uppercase letters.

Eg: import java.lang.String;



- Every package name must be different. Duplicate names will cause runtime errors.

**What is a package? How to create and import a package?****Creating user defined packages**

For creating our own package we have follow following steps.

1. Declare the package at the beginning of a file using the form.  
Syn: **package** packagename;
2. Define the class that is to be put in the package and declare it **public**.
3. Create a subdirectory under the directory.
4. Save the file as the 'classname. java'.
5. Compile the file. This creates **.class** file in the subdirectory.

```
package aaa;

public class Pack
{

 Body of the class

}
```

In the above example, the package name is 'aaa' and the class name is 'Pack'. We can compile this file as 'javac -d . Pack.java'

**Using package**

Consider the following package named 'aaa' containing a single class 'Pack'.

```
Eg: package aaa;

 public class Pack
 {

 public int add(int x,int y)
 {

 return(x+y);

 }

 }
```

```
 }
}
```

This file should be save as **Pack.java**. After compiled this program the file **Pack.class** will be stored in the subdirectory 'aaa'. Consider the following program that imports the class 'Pack' from the package 'aaa'.

```
Eg: import aaa.Pack;

 class testpack
 {

 public static void main(String args[])
 {

 Pack a=new Pack();
 int sum=a.add(10,20);
 System.out.println("Sum="+sum);
 }
 }
```

This file should be save as **testpack.java** and then compile as

Syn: javac -classpath . testpack.java

This file should be run as

Syn: java testpack

## What is stream and byte stream?

### Stream

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

**System.out:** standard output stream

**System.in:** standard input stream

**System.err:** standard error stream

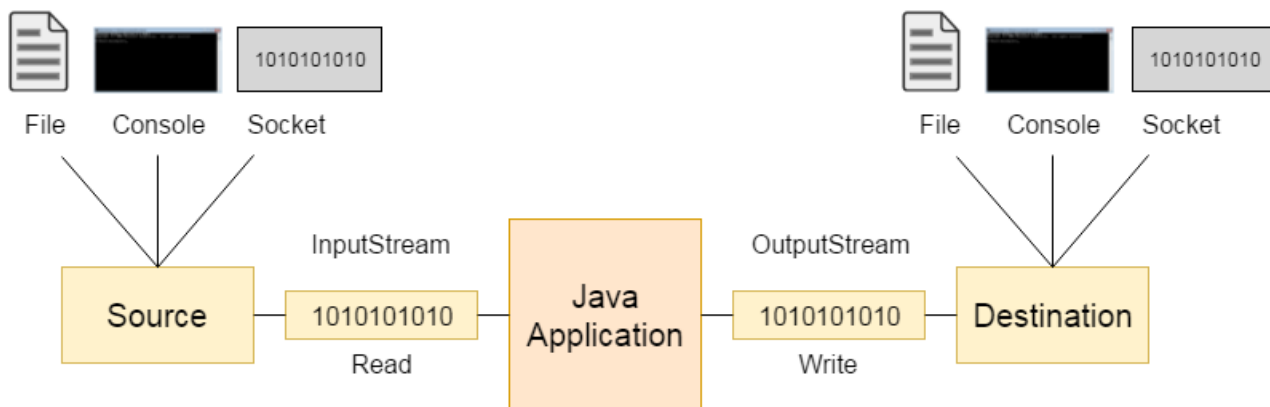
**Byte Stream:** byte streams are used to handle input and output of binary data and hence they are also known as binary streams. `java.io.InputStream` and `java.io.OutputStream` are the top level classes that represent binary or byte streams.

### Input Stream

The `InputStream` class is used for reading the data such as a byte and array of bytes from an input source. An input source can be a file, a string, or memory that may contain the data. An input stream is automatically opened when you create it. You can explicitly close a stream with the `close()` method.

### OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.



**What is Java FileOutputStream Class? Give an example**

If you have to write primitive values into a file, use `FileOutputStream` class. You can write byte-oriented as well as character-oriented data through `FileOutputStream` class. But, for character-oriented data, it is preferred to use `FileWriter` than `FileOutputStream`.

**FileOutputStream class declaration**

```
public class FileOutputStream extends OutputStream
```

**FileOutputStream class methods**

| Method                                                | Description                                                                                                                   |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>protected void finalize()</code>                | It is used to clean up the connection with the file output stream.                                                            |
| <code>void write(byte[] ary)</code>                   | It is used to write <code>ary.length</code> bytes from the byte <a href="#">array</a> to the file output stream.              |
| <code>void write(byte[] ary, int off, int len)</code> | It is used to write <code>len</code> bytes from the byte array starting at offset <code>off</code> to the file output stream. |
| <code>void write(int b)</code>                        | It is used to write the specified byte to the file output stream.                                                             |
| <code>FileChannel getChannel()</code>                 | It is used to return the file channel object associated with the file output stream.                                          |
| <code>FileDescriptor getFD()</code>                   | It is used to return the file descriptor associated with the stream.                                                          |
| <code>void close()</code>                             | It is used to closes the file output stream.                                                                                  |

**Java FileOutputStream Example 1: write byte**

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
 public static void main(String args[]){
 try{
 FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
 fout.write(65);
 fout.close();
 System.out.println("success...");
 }catch(Exception e){System.out.println(e);}
 }
}
```

**Java FileOutputStream example 2: write string**

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
 public static void main(String args[]){
 try{
 FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
 String s="Welcome to javaTpoint.";
 byte b[]=s.getBytes();//converting string into byte array
 fout.write(b);
 fout.close();
 System.out.println("success...");
 }catch(Exception e){System.out.println(e);}
 }
}
```

**What is Java FileInputStream Class? Give an example**

Java FileInputStream class obtains input bytes from a [file](#). It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](#) class.

**Java FileInputStream class declaration**

```
public class FileInputStream extends InputStream
```

**java FileInputStream class methods**

| Method                               | Description                                                                                                  |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------|
| int available()                      | It is used to return the estimated number of bytes that can be read from the input stream.                   |
| int read()                           | It is used to read the byte of data from the input stream.                                                   |
| int read(byte[] b)                   | It is used to read up to b.length bytes of data from the input stream.                                       |
| int read(byte[] b, int off, int len) | It is used to read up to len bytes of data from the input stream.                                            |
| long skip(long x)                    | It is used to skip over and discards x bytes of data from the input stream.                                  |
| FileChannel getChannel()             | It is used to return the unique FileChannel object associated with the file input stream.                    |
| FileDescriptor getFD()               | It is used to return the <a href="#">FileDescriptor</a> object.                                              |
| protected void finalize()            | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close()                         | It is used to closes the <a href="#">stream</a> .                                                            |

### Java FileInputStream example 1: read single character

```

import java.io.FileInputStream;
public class DataStreamExample {
 public static void main(String args[]){
 try{
 FileInputStream fin=new FileInputStream("D:\\testout.txt");
 int i=fin.read();
 System.out.print((char)i);

 fin.close();
 }catch(Exception e){System.out.println(e);}
 }
}

```

**Q) Discuss about Input and Output classes in java.**

1. A Stream is used to accept input from the keyboard at runtime.
2. A Stream represents flow of data from one place to another place.
3. A Stream can carry data from keyboard to memory or from memory to printer.
4. Basically there are two types of streams. They are **input streams** and **output streams**.
5. Input streams are used to receive or read data coming from some other place. Output streams are used to send or write data to some other place.
6. All streams are represented by classes in **java.io (input and output)** package. This package contains a lot of classes. These classes are classified into input streams and output streams.
7. Keyboard is represented by a field, called in the System class. System class is found in **java.lang** package and has three fields. They are

**System.in** : This is InputStream object. It represents standard input device i.e., Keyboard.

**System.out** : This is PrintStream object. It represents standard output device i.e., Monitor.

**System.err** : This field also is PrintStream object, which by default represents monitor.

Example :     System.out.println("Welcome");

                  System.err.println("Welcome");

In java stream classes are classified into two categories. They are

1. **Byte stream classes:** Byte stream classes are defined from two abstract classes: **InputStream** and **OutputStream**. Each of these abstract classes has several concrete subclasses that handle various input and output operations.
  - a. BufferedInputStream
  - b. BufferedOutputStream
  - c. DataInputStream
  - d. DataOutputStream
  - e. FileInputStream
  - f. FileOutputStream
  - g. ByteArrayInputStream
  - h. ByteArrayOutputStream
2. **Character Stream classes:** Character stream classes are defined from two abstract classes: **Reader** and **Writer**. Each of these abstract classes has several concrete subclasses that handle various input and output operations.
  - a. BufferedReader
  - b. BufferedWriter
  - c. CharArrayReader
  - d. CharArrayWriter
  - e. FileReader
  - f. FileWriter



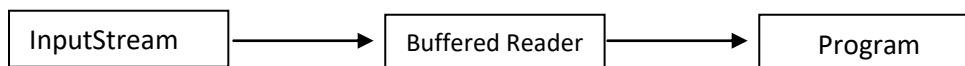
- g. FilterReader
- h. FilterWriter
- i. InputStreamReader
- j. OutputStreamWriter
- k. LineNumberReader

To read the data from the key board, follow the following steps

- 1) Create InputStreamReader object and connect the keyboard(System.in) to it.  
**InputStreamReader obj=new InputStreamReader(System.in);**
- 2) Create BufferedReader object and connect the InputStreamReader to it.  
**BufferedReader br=new BufferedReader(obj);**
- 3) These two steps can be combined and written in a single statement as  
**BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));**
- 4) Now we can read the data coming from the keyboard using read( ) and readLine( ) methods available in BufferdReader class.  
**Ex : String str = br.readLine( );**

#### Q) What is BufferedReader class? Explain.

1. The **java.io.BufferedReader** is the subclass of **java.io.Reader** class.
2. This allows more efficient reading of characters and lines.
3. BufferedReader provides different methods to read data from the keyboard.  
Ex : readLine( )
4. The BuffeeredReader gets data and store that data in a buffer and the program gets data from the buffer.



#### Syntax :

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str=br.readLine();
```

#### Constructors :

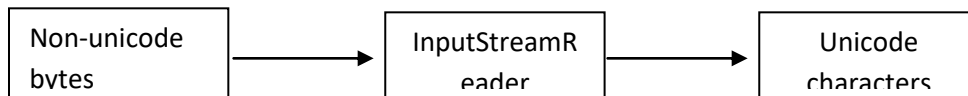
1. **BufferedReader(Reader inputstream)** : it uses a default sized input buffer.
2. **BufferedReader(Reader inputstream, int size)** : it uses a specified sized input buffer.

#### Methods :

1. **close()** Close the stream.
2. **mark(int readAheadLimit)** Mark the present position in the stream.
3. **read()** Read a single character and returns an integer.
4. **readLine()** Read a line of text.
5. **skip(long n)** Skip characters

**Q) What is InputStreamReader class? Explain**

1. The InputStreamReader is a class from java.io package is the subclass of java.io.Reader class.
2. The InputStreamReader class serves as a bridge between byte streams and character streams.
3. It reads bytes from the input stream and translates them into characters according to a specified character encoding.
4. It reads bytes and decodes them into characters using a specified charset.
5. Calling of read() method it gets some bytes of information from input stream.

**Syntax :**

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str=br.read();
```

**Constructors of InputStreamReader Class :**

**InputStreamReader ( InputStream in) :** Create an InputStreamReader that uses the default charset.

**InputStreamReader ( InputStream in, Charset cs) :**

Creates an InputStreamReader that uses the given charset.

**InputStreamReader ( InputStream in, CharsetDecoder dec) :**

Creates an InputStreamReader that uses the given charset decoder.

**InputStreamReader ( InputStream in, String cn) :**

Creates an InputStreamReader that uses the named charset.

**Methods :**

**read( )** : This method is used to read a single character.

**ready( )** : This method tells us whether this stream is ready to be read or not.

**getEncoding( )** : This method is used to know the name of the character encoding being used by this stream.

**close ( )** : This method is used to close the stream.