UNIT-I: Introduction to JAVA

Introduction:

- JAVA is general purpose OOP (Pure OOP, since supports almost all OOP concepts) language developed by "Sun Microsystems" of USA in1991
- 'James Gosling' was the inventor (Creator) of JAVA language.
- Originally or firstly JAVA was named as "Oak" (Oak is name of tree which was found in front of Goslings Office)
- Basically, JAVA was designed for the development of software's for electronic devices like TV's, VCR's, set-top box, Toasters etc.
- Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- The 'C' and 'C++' languages had limitations in terms of reliability and portability therefore they modeled their new language JAVA to overcome the drawbacks of 'C' and 'C++'. Thus, JAVA made really simple, reliable, portable and powerful language.

History OR Evolution of JAVA:

Following table shows some milestones happen in developing of JAVA language:

Year	Development
1990	Sun Microsystem decided to develop special software that could be used to manipulate with electronic devices. A team of Sun Microsystems programmer headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of such idea, the team announced a new programming language called 'Oak' (Oak was the first name for JAVA)
1992	In this year, team of Sun Microsystems actual implements there language in home appliances like Microwave Oven etc. with tiny touch-sensitive screen.
1993	In this year, team of Sun Microsystems came up with new idea to develop web based application that could run on all types of computers connected to Internet. For that, they creates 'applet' (tiny program run on Internet by the browser)
1994	In this year, team of Sun Microsystems developed web browser called "HotJava" to locate and run applet on Internet.
1995	"Oak" was renamed as "JAVA" due to some legal snags (problems). Also, many popular companies like Netscape and Microsoft announced to support for JAVA
1996	Sun Microsystem releases Java Development Kit 1.0 (JDK 1.0) to develop different kinds of software.
1997	Sun Microsystem releases Java Development Kit 1.1 (JDK 1.1)
1998	Sun Microsystem releases JAVA 2 with JDK 1.2 of Software Development Kit (SDK 1.2).
1999	Sun Microsystem releases standard Edition of Java which was called J2SE(Java 2 Standard Edition) and J2EE (Java 2 Enterprise Edition)
2000	J2SE with SDK 1.3 was released
2002	J2SE with SDK 1.4 was released
2004	J2SE with JDK 5 (JDK 1.5) was released

- The latest version of Java is Java 17 or JDK 17 released on September, 14th 2021 and Java is now under administration of Oracle organization.

Features or Characteristics or Advantages of Java:

- **Compiled & Interpreted:**
  - Usually, programming language is either compiled or interpreted. But Java combines both approaches that make Java 'two-stage system'.
  - In case of Java, First Java compiler translates or converts Java source program into 'byte code' (Byte code is not machine instruction code & byte code file having extension '.class')
  - After compilation, Java Interpreter executes this byte code & thus we got our desired output.
    Thus, we can say that Java is Compiled & Interpreted language.
- **Object Oriented:**
  - Java is pure object oriented language that supports for all OOP's concepts.
  - Almost, In Java, everything is an Object. All data and methods are resided (exist in) within an object and classes.
  - The object model in Java is easy to extend because it supports for Inheritance concept.
- **Platform independent and Portable:**
  - Portable: We know that, after compilation of Java source program it produce ".class" file i.e. byte code which is not machine dependent that's why such file is easily moved or transferred from one computer to another computer and hence Java is Portable.
  - Platform independent: After generation of byte code (.class file), this byte code is easily interpreted or executed on different kinds of computers having different platforms (Computers having different Operating system like windows, Linux, Mac OS etc and different processors etc).
- **Simple:**
  - Java is designed in such a way that it would be easy to learn since, most of syntax of java is same as C and C++.
  - If you understand the basic concepts of OOP then it is easy to implement in Java language.
- **Secure:**
  - We know that, most of viruses are attacked on files having extension '.exe', '.doc', '.gif', '.mpg' etc. but after compilation of Java source program it produce ".class" file i.e. byte code and which is virus free. And hence, Java enables us to develop virus-free, tamper -free systems.
- **Architectural-neutral:**
  - Java compiler generates an architecture-neutral class file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Robust:**
  - Java is strict type checking language which checks an error at both time i.e. at compile time and also at run time of program.
  - Due to this ability of checking errors at run time (exception Handling), we can eliminates any risk of crashing the system using Java.
- **Multithreaded:**
  - Multithreaded means handling multiple tasks (jobs) simultaneously (at one time).
  - Java supports for multithreaded programs that means we need not wait for the application to finish its task before beginning another.
  - That is using Java, we can run multiple java applications without waiting to finish another.
- **Distributed:**
  - Java enables us to make such applications that can open and access remotely over the internet or network.
  - That is, multiple programmers at multiple remote locations are capable to work together on single project. That's why Java is distributed.
- **Dynamic and Extensible:**
  - Dynamic: Java is dynamic language which is capable to link new class libraries, methods and objects dynamically.
  - Extensible: Java supports to write functions in C or C++ language such functions are called "native methods" and then we can add or link these methods with Java such that they can be used in many applications.

- **Ability to Deal with Database:**
  - Java supports for JDBC (Java Database Connectivity) to send & retrieve data in tabular format with the database thus with the help of Java we are able to deal with database.
- **Automatic Memory Management:**
  - We know that 'memory' is very important issue while dealing with computer and we have to manage it very efficient manner.
  - Java language supports for 'Garbage Collector' that automatically manages all the memory in efficient manner.

Limitations or Disadvantages of Java:

- **Slow language:**
  As compared to C and C++ languages, Java language compiler took much more time to compile the program & also Java interpreter took much more time to interpret the program that's why Java is slow language.
- **Strict type checking language:**
  Due to strict type checking, Java language checks much run time errors & that's why Java application took much time to execute.
- **Case sensitive language:**
  Due to case sensitive language, we must have to write correct spelling of inbuilt methods, classes, interfaces etc. while doing programming.
- Java does not support for Multiple Inheritance but we can implement it by using 'interface'.

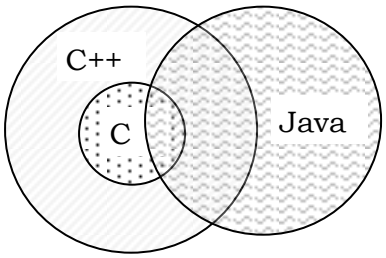
Difference Between C and Java:

'C' Language	Java Language
1) It is not OOP language.	1) It is pure OOP language.
2) It has preprocessor directive statements like #define, #include etc	2) It has no preprocessor directive statements like #define, #include etc.
3) It does not support for data type 'class'	3) It does supports for data type 'class'
4) It has type modifier keywords like auto, extern, register , signed and unsigned.	4) It does not have type modifier keywords like auto, extern, register, signed and unsigned.
5) It supports for 'pointer'	5) It does not supports for 'pointer'
6) It supports for data type 'struct' and 'union'	6) It does not supports for data type 'struct' and 'union'
7) It has 'sizeof' and 'typedef' keywords	7) It has not 'sizeof' and 'typedef' keywords
8) Automatic memory management is not supported.	8) Automatic memory management is supported by 'Garbage Collector'.

Difference Between C++ and Java:

'C++' Language	Java Language
1) It is not pure OOP language.	1) It is pure OOP language.
2) It supports for template classes.	2) It does not support for template classes.
3) It supports for 'Multiple inheritance'	3) It does not supports for Multiple Inheritance but we implement it using 'interface'
4) It supports for global variable.	4) It does not have global variable
5) It supports for 'pointer'	5) It does not supports for 'pointer'
6) It supports for "destructor"	6) It does not supports for "destructor" but it is replaced by finalize( ) method.
7) It has 'goto' statement.	7) It has not 'goto' statement.
8) It has preprocessor directive statements like #define, #include etc.	8) It has no preprocessor directive statements like #define, #include etc.
9) It has three access specifiers viz: public, private and protected	9) It has four access specifiers viz: public, Private, protected and default.
10) It supports for operator overloading.	10) It does not supports for operator overloading
11) Automatic memory management is not supported.	11) Automatic memory management is supported by 'Garbage Collector'.

Following fig. shows overlapping of C, C++ and Java:



From above fig.

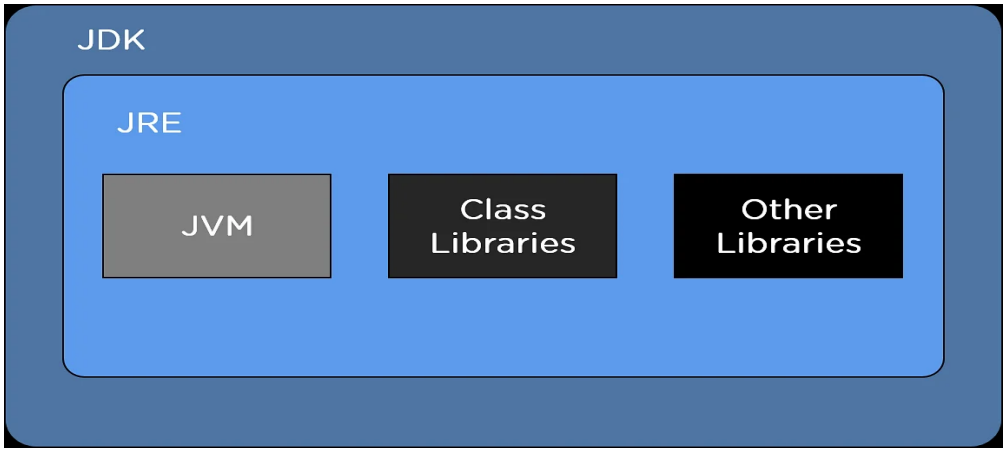
- We know that, 'C++' is superset of 'C' language therefore every 'C' program is easily executed by 'C++' compiler.
- But, Java language is partly combination of 'C' and 'C++' language and it having its own extra features therefore Java can be considered as first cousin of 'C++' and second cousin of 'C'

Java Development Kit (JDK):

- JDK in Java is an essential component necessary to develop programs or software's using JAVA language.
- It is technically an implementation of either Java Standard Edition or Java Enterprise Edition.
- JDK in Java is an abbreviation for Java Development Kit. It is a bundle of software development tools and supporting libraries combined with the Java Runtime Environment (JRE) and Java Virtual Machine (JVM).
- JSL (Java Standard Library) also called as Java API (Application Programming Interface) is the main part of JDK that contains thousands of Packages.
- Further, Packages contains thousands of classes, methods, interfaces etc.

The Architecture of JDK in Java:

- The architecture of JDK in Java includes the following modules as described in the image below.

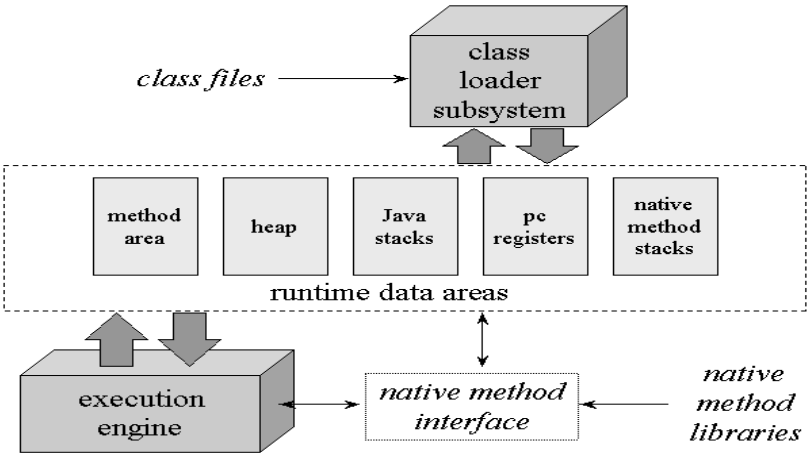


- The two vital software modules of JDK are:

1) JVM (Java Virtual Machine):

- Java Virtual Machine is a software tool responsible for creating a run-time environment for the Java source code to run. The very powerful feature of Java, "Write once and run anywhere," is made possible by JVM.
- The JVM stays right on top of the host operating system and process the Byte Code (machine language), such that it would easily executed by microprocessor.
- Java Virtual Machine plays vital or important role in execution of java program therefore it is heart of java.

Following Figure shows Architecture of JVM:



Note:

- *Java*
  Compiler generates or creates byte code which is machine independent or platform independent therefore it is easily interpreted by any JVM that's why it is called as "write once run anywhere".
- *But, JVM is platform dependent i.e. windows, Linux, Mac OS, Unix etc. operating system having their different-different JVM's.*

Working of JVM:

- First of all, java source file (.java file) is converted into byte code (.class file) by the java compiler and this byte code file is given to the JVM.
- In JVM, there is one module or program called 'Class loader sub system' which performs following functions:
  - First, 'Class loader sub system' loads the '.class' file into memory.
  - Then it verifies whether all byte code instructions are proper or not.
  - If it finds some problem in byte code then it immediately terminates the execution.
  - If byte code is proper then it allocates necessary memory to execute the program.

Also, this memory is divided into 5 parts called 'Runtime data area' & these parts as follows:

1) Method area:
   In this memory area, all class code, variables codes, methods codes etc. are stored.
2) Heap:
   In this memory area, all objects are created and stored.
3) Java Stacks:
   Actually, java methods are stored in 'Method area' but actual execution of such java methods are happen under 'Java stacks' area.
4) PC registers:
   This area contains the memory addresses of instructions of the methods.
5) Native method stacks:
   All native methods (C, C++ functions) are executed under native methods stacks.
   And all native methods are connected with JVM by '*native method interfaces*'

After, allocation of memory into corresponding parts then it comes towards 'Execution Engine'.

- Execution Engine can consists of two things VIZ:
  1) Interpreter 2) JIT (Just In Time) compiler.

- This interpreter and JIT compiler are responsible for converting byte code into machine instruction such that it easily executed by microprocessor.
- After, loading the ".class" file into memory, JVM first identifies which code is to be left to interpreter and which one to JIT compiler so that the performance is better. The blocks of code allocated for JIT compiler are also called 'hotspots'. Thus, the interpreter and JIT compiler will work simultaneously to translate the byte code into machine instructions.

Note that: JIT compiler is a part of JVM which increases execution speed of program.

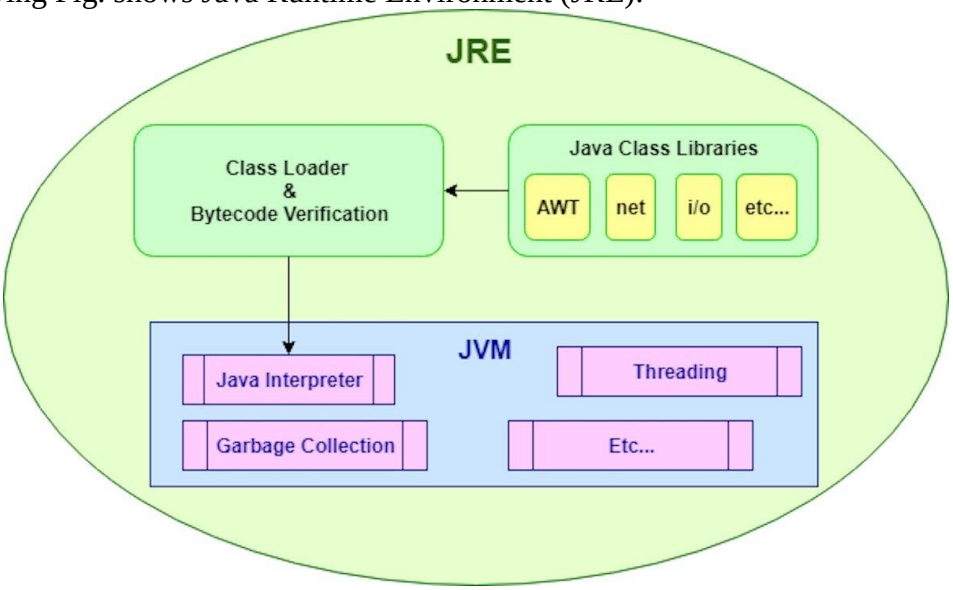
2) JRE (Java Run-time Environment)

- Java Run-time Environment is a software platform where all the Java Source codes are executed.
- JRE is responsible for integrating the software plugins, jar files, and support libraries necessary for the source code to run.
- The Java Runtime Environment, or JRE, is a software layer that runs on top of a computer's operating system software and provides the class libraries and other resources that a specific Java program needs to run.
- A Java™ runtime environment (JRE) is a set of components to create and run a Java application.
- A JRE is part of a Java development kit (JDK).
- A JRE is made up of a Java virtual machine (JVM), Java class libraries, and the Java class loader.
- In short JDKs are used to develop Java software whereas JREs provide programming tools and deployment technologies.

Why use a Java runtime environment?

- In order for software to execute a program, it needs an environment to run in—usually an operating system (OS) like Linux, Unix, Microsoft Windows, or MacOS.
- Because of JRE, java programs are constrained the capabilities of the OS and its resources (such as memory and program files).
- A JRE acts as facilitator or interface between the Java program and the OS that demands resources towards the OS.

Following Fig. shows Java Runtime Environment (JRE):



JDK Components (JDK tools):

Following is the list of tools or components of JDK which are used to develop and run the java programs:

Sr.NO	Tool or Component	Description or Use
1	javac (Java Compiler)	It translates or converts java source program into byte code file & that file understood by java interpreter
2	java (java Interpreter)	It runs java applications by reading corresponding byte code file & gives result.
3	Appletviewer	It runs or views java applets onto the web browser.
4	javap (Java disassembler)	It converts byte code file into program description
5	Javadoc	It creates or produces HTML format documentation of java source file. But it needs public class for documentation.
6	Javah	It creates or produces header files for use of native methods.
7	jdb (Java Debugger)	It helps us to checks errors in java program.

Why Java does not support for pointer?

→

1) We know that 'pointers' are used to hold memory address. And most of viruses are trying to attack on memory that's why Java does not support for pointer and hence Java is secured.

2) Also, pointers are helpful for dynamic memory allocation i.e. it is used for run time memory management, but in Java all memory management is automatically done by 'Garbage collector' that's why not need of pointer.

Structure of JAVA Program:

- We know that single Java program may contains multiple classes but out of these classes, one class should be public and that class contains main( ) method from which JVM interprets the byte code.
- Note that: Java is pure OOP language i.e. all programs must have classes and objects.
- A typical Java Program is divided into several Sections which are shown in following figure:

Documentation Section
Package statement Section
Import statement Section
Interface statement Section
Class definition section

main( ) method Section
{
// main() definition
}

1) Documentation Section:

- This section contains set of comments lines showing details of java source program such as program name, programmer name, date of program, version etc. this help program readability.

In Java, we can give comments by three ways VIZ:-

1) Single line comment:

- If we have to specify general information of program within single line then single line comment is used. Single line comment is given by // notation.
- Also, we can specify this comment anywhere in program.

E.g. // Program Name= Addition of two numbers.

2) Multiline comment:

- If we have to specify general information of program within multiple lines then multi line comment is used. Multiline comment is given by following notation.

```
/* ----------
   ----------
---------- */
```

E.g.

```
/*      Program Name:   Multiplication
        Programmer: James Gosling */
```

3) Third Style comment: (Java documentation Comment)

- This type of comment is specially used for documentation purpose.
- If we specify description using 'Third style comment' then it is shown in HTML files created by using 'Javadoc'
- This comment is used to provide description for every feature in Java source program.

Third Style comment is given by-

```
/**  ----------------------
     ----------------------   */
```

E.g.

```
/**  This class is used for addition   */
public    class      add
{
      /**  This method is used for addition   */
        public    void  addition( )
        {
                // statements
        }
}
```

- In above example, two times documentation comment is used that will show description of class 'add' and description of method 'addition( )' in HTML file. Note that: For generation of HTML documentation of java source program using 'javadoc' component, class and method should be public or protected.

2) <u>Package statement Section:</u>

- This section is used to declare our own package. When we declare own package then it informs to the java compiler to link all classes of our package with java source program.
- Syntax to specify package statement:

```
package      package_name;
```

E.g.

```
package      student;
```

More about package will be discussed in next chapter.

<u>3) import statement Section:</u>

- In this section we can import existing package in our java source program.
- We know that, in case of 'C' language if we have to use printf( ) method then we include 'stdio.h' header file using preprocessor directive '#include'.
- Similarly, if we have to use existing classes or exiting methods of JSL (Java Standard Library) then we have to import that package in our source program using 'import' statement.
- Syntax to import package in program:

```
import          package_name;
```

E.g.

```
import          java.lang.* ;
```

<u>Difference between #include & import:</u>

→

- *When we include header file in program then C/C++ compiler goes to the standard library (it is available at c:\tc\lib) and searches for included header file there. When it finds the header file, <u>it copies entire header file</u> into the program where the #include statement is written. Thus, if our C/C++ program has only 10 lines still C/C++ compiler shows hundreds of line compiled this is due to copy of included header file at #include statement. Therefore our program size increases & hence it causes memory wastage.*
- *When we import package in Java program then JVM checks whether imported package is present in JSL or not. If JVM finds imported package then it executes corresponding method code there and only returns its result to source program therefore size of source program in not increased as happened in C/C++. And hence, memory wastage is solved.*

<u>4) interface statement Section:</u>

- In this section we can define interfaces.
- Interfaces are similar to the classes but all methods of interface are by default 'abstract.
- This is optional section, used while implementing multiple inheritance in java.

5) class definition Section:

- We know that single Java program may contain multiple classes and every class has its own attributes (data members) and methods. Such type of classes can be defined under class definition section.

  Note that:

  - *We know that single Java program may contains multiple classes but out of these classes, one class should be public and that class contains main( ) method from which JVM interprets the byte code.*

6) main( ) method Section:

- We know that in case of C/C++, main( ) function is compulsory from which execution of program starts. Like that java program also have main( ) method from which JVM starts program interpretation. This is compulsory section. Also, main( ) method in Java must be public. If we made main( ) as private or protected then it is not assessable for JVM also.
- main( ) method should be defined under any class of program but that class should be public.

Simple Java Program:

Let's consider following simple java program;

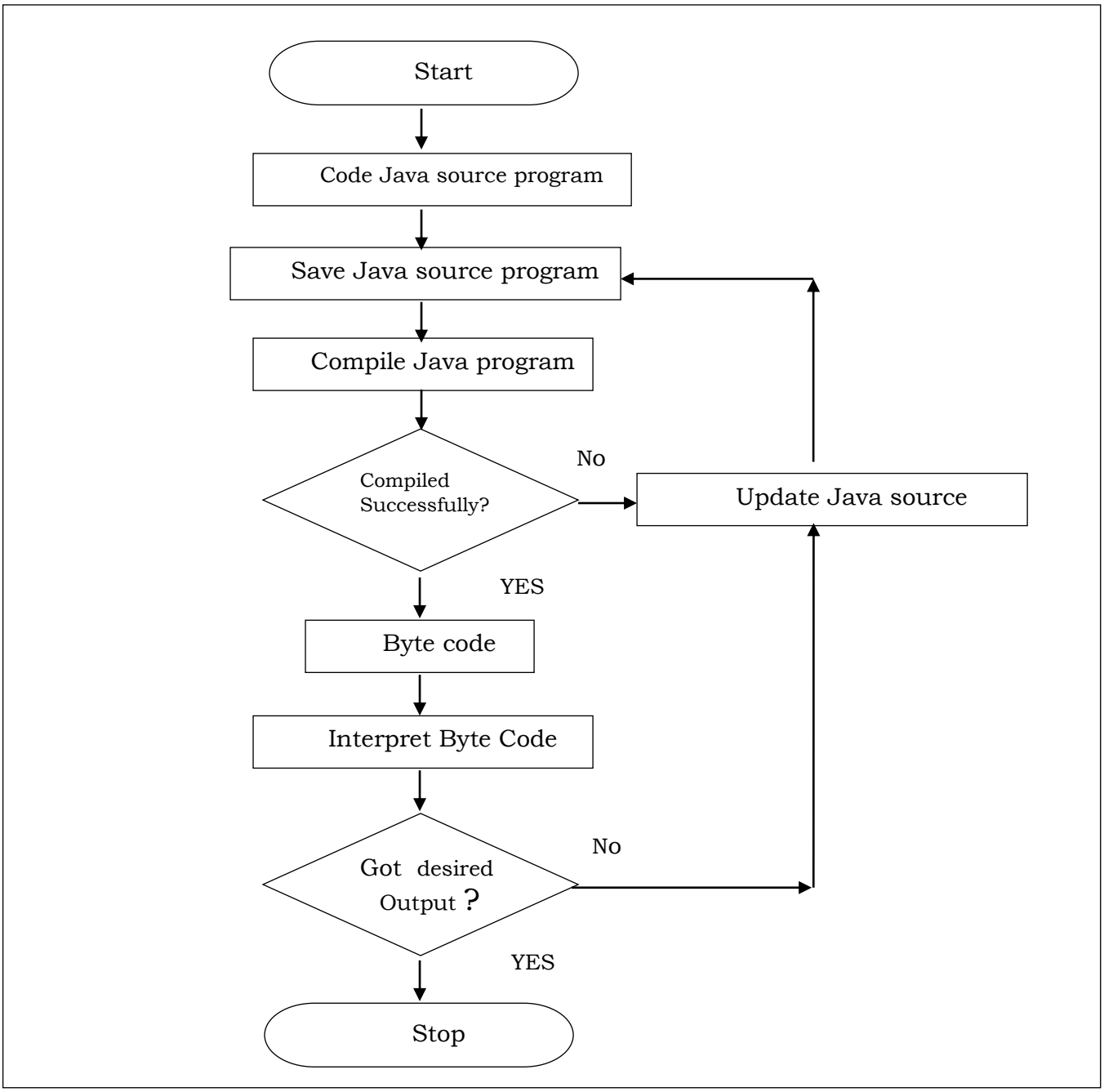
```
import     java.lang.*;
class              first
{
    public   static void   main(String   args[ ])
    {
            System.out.println("Welcome in JAVA programming");
    }
}
```

In above program,

- "java.lang" is a package which is imported using 'import' keyword. This package contains lots of inbuilt classes such as System, String, Integer, Float etc. This is default package i.e. there is no necessary to import it.
- Here, main( ) method is compulsory which is declared as public, static and void
  - It is public because it made available for JVM for interpretation of java program.
  - It is static because it should be called without any object; it is invoked by JVM with class name.
  - It is void because it does not return any value.
- Also, main( ) method accepts array of string as argument which is called as command line argument. The passed values are stored in args[ ] array at individual indices.
- System.out.println( ) statement:
  - "System" is inbuilt wrapper class which was found under 'java.lang' package.
  - "out" is object of 'System' class which is 'static' & hence it is accessed by 'System' class name.
  - "println( )" is a method was found in "System" class used to display output and called by using "out" object.

Steps to execute Java Program:

Following flowchart shows compiling and interpreting Java program;



Syntax to Compile Java Source program:

Java program is compiled with 'java source program' name along with 'javac' component which is given as fallow:

```
javac        JavaSourcePgmName.java ↵
```

E.g. Consider, we have *'good.java'* source program then we can compile it as follow:

```
javac        good.java ↵
```

If *'good.java'* program have one class named 'good' then *'good. Class'* byte code is generated.

Syntax to interpret or Run or Execute the Byte code:

Java program is interpreted or run or execute using byte code (.class file) along with 'java' interpreter which is given as fallow:

```
java        ByteodeFile ↵
```

E.g. Consider, we have *'good.class'* byte code then it is interpreted as follow:

```
java      good ↵
```

Note that: *After compilation of java source program, byte code (.class file) is generated. And then JVM interpret that byte code and we got our result.*

Syntax to pass arguments to main( ) method while interpretation of bytecode :

We can also pass some string type arguments to main ( ) method called 'command line arguments' using following syntax.

```
java      ByteCodeFile      arg1      arg2  - - - -   argN ↵
```

In above syntax;
arg1, arg2, - - - - ,argN are the command line arguments passed to main() method while interpreting.
Note that: All passed arguments are stored in formal parameter (String type array) of main( ) method at individual indices.
E.g. Consider following Program:

```
import    java.lang.*;
class             first
{
    public   static void   main(String   args[ ])
    {

            System.out.println("FirstName= "+ args[0]);
            System.out.println("MiddleName= "+ args[1]);
            System.out.println("LastName= "+ args[2]);

    }
}
OUTPUT:
            javac     first.java
                                  ↵
            java     first          SACHIN     RAMESH     SHINDE
```

In above example; three command line arguments are passed to main( ) method. They are SACHIN RAMESH SHINDE.
All these arguments are stored in 'args' String type array in main( ) method at individual indices as fallow;

args[0]=>	SACHIN
args[1]=>	RAMESH
args[2]=>	SHINDE

Also, we use '+' operator to concatenates two strings with each other.

Naming Conventions in Java:

- *Naming Conventions* specify the rules to be followed by java programmer while writing or coding java source program.
- We know that java program contains the package, classes, interfaces, methods, variables etc. and all these have separate naming conventions they are as follow:

Naming Conventions for Package:

- We know that, Package is one kind of directory that contains the classes and interfaces.
- Package name in java should write in small letters only.

Example:

java.lang
java.awt
javax.swing

Naming Conventions for class or interface:

- We know that, class is model for creating object.
- Class specifies the properties and action for objects.
- An interface is similar to class but it has abstract methods only.
- Class and interface name in java should start with capital letter.

Example:

System
String
Integer
Float etc.

Naming Conventions for methods:

- We know that, methods contain the executable statements or instructions after execution it produce desired result.
- The first word of a method name is in small letters, then from second word onwards, each new word starts with capital letter as:

  Example:

  println();
  readLine();
  getNumberInstance();

Naming Conventions for variables:

- Naming conventions for variable is same as that of methods i.e. *The first word of a variable name is in small letters, then from second word onwards, each new word starts with capital letter as:*

  Example: age
  empName
  empNetSal

Java Tokens:

- "Token is nothing but smallest individual unit of java source program."
- We know that Java is pure OOP language i.e. every program has classes and every classes has some methods and methods contains executable statement and every executable statement contains the tokens i.e. statements are made up of several tokens.
- Following are the several tokens in Java program:

  1) Keywords 2) Data type 3) Identifier 4) Variable
  5) Constant or Literals 6) Operators 7) Special symbols.

Let us see all tokens in details:

1) Keywords (Reserve words):

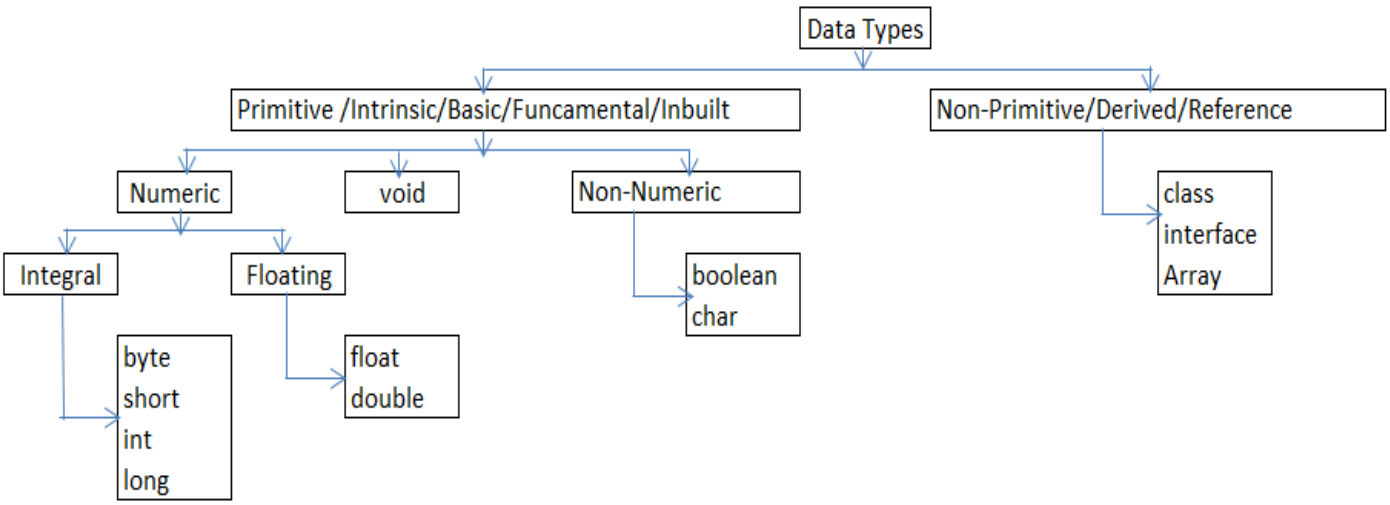
- The words whose meaning is already known by java compiler are called as 'Keywords'.
- These words having fix meaning and we are not able to change that meaning therefore they are also called as 'Reserve Word'.
- Java language contains more than 50 keywords and they are listed as fallow:

Abstract	Continue	For	new	switch
Assert	Default		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
	Float	native	Super	while

2) Data Type:

- Data: "Data is nothing but collection of raw information or unprocessed information that we provide for the computer for processing"
  e.g. numbers, string, alphanumeric etc.
- Data Type:
- Concept: When we give data to the computer for processing at that time compiler does not know which type of input data is.

Generally, Data types are used to tell the compiler which type of input data is.



- *Definition*: "Type of Data is called as Data Type"

Following tree diagram shows data types in Java language:
Let us see all these data types in details:

Primitive Data Types:

There are nine primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

1) byte:

- byte data type is an 8-bit(1 byte) integral data type.
- Its Minimum range value is -128 (i.e. -2^7)
- Its Maximum range value is 127 (inclusive)(i.e. 2^7 -1)
- Its Default value is 0
- byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100; byte b = -50;

2) short:

- short data type is a 16-bit (2 bytes) integral.
- Its Minimum range value is -32,768 (i.e. -2^15)
- Its Maximum value is 32,767 (inclusive) (i.e. 2^15 -1)

- Short data type can also be used to save memory as int data type. A short is 2 times smaller than an int
- Its Default value is 0.
- Example: short s = 10000, r = -20000;

3) int:

- int data type is a 32-bit (4 bytes) signed integral data type.
- Its Minimum range value is - 2,147,483,648.(i.e. -2^31)
- Its Maximum range value is 2,147,483,647(inclusive).(i.e. 2^31 -1)
- int is generally used as <u>the default data type for integral values</u> unless there is a concern about memory.
- Its default value is 0.
- Example: int a = 100000, b = -200000;

4) long:

- long data type is a 64-bit (8 bytes)signed integral data type.
- Its Minimum range value is -9,223,372,036,854,775,808.(i.e. -2^63)
- Its Maximum range value is 9,223,372,036,854,775,807 (inclusive). (i.e. 2^63 -1)
- This type is used when a wider range than *int* is needed.
- Its Default value is 0L.
- Example: long a = 100000L, int b = -200000L;

5) float:

- float data type is a single-precision 32-bit (4 bytes) floating data type.
- Its Minimum range value is $- 3.4e^{38}$ to $-1.4e^{-45}$ for negative value.
- Its Maximum range value is $3.4e^{38}$ to $1.4e^{-45}$ for positive value.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Its default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f;

6) double:

- double data type is a double-precision 64-bit (8 bytes)floating data type.
- Its Minimum range value is $-1.8e^{308}$ to $-4.9e^{324}$ for negative value.
- Its Maximum range value is $1.8e^{308}$ to $4.9e^{324}$ for positive value.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Its Default value is 0.0d.
- Example: double d1 = 123.4;

7) boolean:

- boolean data type represents <u>*one bit*</u> of information.
- There are only two possible values: *true and false*.
- This data type is used for simple flags that track true/false conditions.
- Its default value is *false*.
- Example: boolean one = true;

8) char:

- char data type is a single 16-bit (2 bytes) non-numeric character data type.
- Its Minimum value is '\u0000' (or 0).
- Its Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letter ='A';

9) void:

- void means <u>no value</u>
- This data type is generally used to specify return type of method.
- If return type of method is void then that method does not return any value.

Non-Primitive or Derived or Reference Data Types:

- The data types derived or created with the help of inbuilt of data types is called 'Non-primitive or derived or reference data types"
- Java language has three Non-primitive data types viz. array, class and interface.
- Note- In next units we will discuss above mentioned non-primitive data types.

3) Identifier:

- "Identifier is the name given by the programmer for any variable, package, class, interface, array, object etc."
- There are several rules to declare or define the identifier:
    1) Identifier should not be keyword.
    2) Identifier should not start with digit.
    3) Identifier can be combination of alphabets, digits or **underscore or dollar sign($)**.
    4) Identifier should not contain special symbol except underscore and **dollar sign($)** .
    5) Identifier should not contain any white space character like horizontal tab, vertical tab, new line etc.
    6) Identifier should be meaningful.
    7) Identifier can be of any length.

4) Variable:

- "Variable is the name given to the memory location where the data is stored such quantity is called as Variable"

OR

- "The quantity that changes during program execution is called as Variable"
- Concept: The main concept behind variable is that <u>every variable has an ability to store the data.</u>

<u>Syntax to declare variable:</u>

```
DataType      variableName ;
```

Here;

DataType is any valid data type in 'Java' language.
variableName is an identifier.

Example:
```
int    rollno;
char   x;
```

- There are several rules to declare the variable:

1) Variable should not be keyword.
2) Variable should not start with digit.
3) Variable can be combination of alphabets, digits or underscore **or dollar sign($)**.
4) Variable should not contain special symbol **except underscore and dollar sign**.
5) Variable should not contain any white space character like horizontal tab, vertical tab, new line etc.
6) Variable should be meaningful.
7) Variable can be of any length.
8) Declared *local variable* <u>must</u> be initialized anywhere in block.

<u>Types of Variables in java:</u>

- **Local variables:**
    - The variables which are declared inside methods, constructors or blocks are called <u>local variables</u>.
    - These variables are declared and initialized within the method and they will be destroyed automatically when the method has completed its execution.
- **Instance variables:**
    - Instance variables are variables which are declared within a class but outside any method.
    - These variables are instantiated when the class is loaded.

- Instance variables can be accessed from inside any method, constructor or blocks of that particular class but not accessed within static method *directly*.
- These variables are in the scope of object.

- **Class variables:**
  - Class variables are variables which are declared within a class but outside any method and declared with the static keyword.
  - These types of variables are common to all objects of class i.e. all static data are shared among all objects of class commonly.
  - Note: Such class variables are not in the scope of object.

Following program shows variables in Java-

```
public    class     first
{
                int       a;              //  instance variable
                static  float     b;   // class variable
     public   static void main(String     []arg)
     {
              boolean    flag;     //local variable
      }
}
```

5) Constant (Literals):

- A *literal* represent a fixed value that is stored into variable directly in the program. They are represented directly in the code without any computation.
- Literals can be assigned to any primitive type variable.
  For example:
  - byte p = 68;
  - char a = 'A';

Java has different types of literals VIZ:
1) Integer Literals
2) String Literals
3) Character Literals
4) Float Literals
5) Boolean Literals

Let us see all literals in details:

1) Integer Literals:

- Integer literals represent the fixed integer values like 23, 78, 658, -745 etc.
- The data type byte, int, long, short belongs to decimal number system that uses 10 digits (from 0 to 9) or octal number system that uses 8 digits (from 0 to 7) or hexadecimal number system that uses 16 digits (from 0 to F) to represent any number.

Note that:
Prefix 0 is used to indicate octal and prefix 0x indicates hexadecimal when using these number systems for literals.
For example:
- int decimal = 100;
- int octal = 0144;
- int hexa = 0x64;

2) String Literals:

- String literals are collection of characters which are representing in between a pair of double quotes.

Example:

```
String     x="Hello World";
```

3) Character Literals:

- Character literals are characters which are representing in between a pair of single quotes.
- Character literals are like ‘A’ to ‘Z’, ‘a’ to ‘z’, ‘0’ to ‘9’or Unicode character like ‘\u0042’ or

escape sequence like '\n', '\b' etc.

Example:

```
char    x= ' Z';
```

4) Float Literals:

- Float literals represents fractional values like 2.3, 86.58, 0.0, -74.5 etc.
- These types of literals are used with float and double data types.
- While writing such literals, we can use E or e for scientific notation, F or f for float literal and D or d for double literals (this is default and generally omitted)

Example:

```
float     p = 9.26;
double   q = 1.56e3;
float     m =986.8f;
```

5) Boolean Literals:

- Float literals represents only two values – true or false. It means we can store either 'true' or 'false' into a Boolean type variable

Example:

```
boolean     p =true;
```

6) Operators:

An '*operator*' is a symbol that tells computer to perform specific task.

OR

An 'Operator' is a symbol that operates onto the operand to perform specific task.

Following are the several operators present in Java:

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Increment and decrement operator
5) Assignment operator
6) conditional operator
7) Bitwise operators
8) 'new' operator
9) 'instanceof' operator
10) cast operator

(Note that: All the operators listed above <u>from 1 to 6</u> are same as C/C++ language therefore refer notes of C/C++ language)

Let's see some operators of Java language as follows:

7) Bitwise operators:

- Bitwise operators are used to manipulate data at bit (0 or 1) level.
- These operators act on individual bits of the operands.
- Bitwise operators only act <u>on integral data types such as byte, int, short, long</u>. That is they are <u>not worked on float and double data type</u>.
- When these operators work on data then internally (automatically) data is converted into binary format & then they start their working.
- There are 7 different bitwise operators present in Java as follows:

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (i.e. XOR)
~	Bitwise Complement
<<	Bitwise left Shift
>>	Bitwise Right Shift
>>>	Bitwise Zero fill Right shift

The truth table or working of bitwise operator &, | and ^ is shown in following table:

Op1	Op2	Op1 & Op2	Op1 | Op2	Op1 ^ Op2
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Bitwise AND operator (&):

This operator performs 'AND' operation on individual bits of the numbers. To understand the working of *'&'* operator see following example.

E.g.:

1) 22&5

→

		128	64	32	16	8	4	2	1
22	→	0	0	0	1	0	1	1	0
5	→	0	0	0	0	0	1	0	1
22&5	→	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

In above table, '1' bit is found in '4' column only therefore '22&5' gives result **'4'**

Bitwise OR operator (|):

This operator performs 'OR' operation on individual bits of the numbers. To understand the working of '|' operator see following example.

E.g.:

1) 35|7

→

		128	64	32	16	8	4	2	1
35	→	0	0	1	0	0	0	1	1
7	→	0	0	0	0	0	1	1	1
35|7	→	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**

In above table, '1' bit is found in '1', '2', '4' and '32' columns therefore '35|7' gives result 1+2+4+32= **39**

Bitwise XOR operator (^):

This operator performs 'exclusive OR' operation on individual bits of the numbers. Its symbol is denoted by '^' which is called *cap, carat or circumflex* symbol. To understand the working of '^' operator see following example.

E.g.:1) 47^4

→

		128	64	32	16	8	4	2	1
47	→	0	0	1	0	1	1	1	1
4	→	0	0	0	0	0	1	0	0
47^4	→	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

In above table, '1' bit is found in '1', '2', '8' and '32' columns therefore '47^4' gives result 1+2+8+32=**43**

Bitwise complement operator (~):

This operator gives complement form of the given number. Its symbol is denoted by '~' which is called '*tiled'* symbol. To understand the working of '~' operator see following example.

E.g.:

1) ~47

→ It gives result= -48

2) ~(-26)

→ It gives result= 25

Bitwise left shift operator (<<):

This operator shifts the bits towards *left side* by a specified number of positions. Its symbol is denoted by '<<' which is called *double less than* symbol. To understand the working of '<<' operator see following example.

E.g.:1) 15<<3

→

		128	64	32	16	8	4	2	1
15	→	0	0	0	0	1	1	1	1
	0		1	1	1	1	**0**	**0**	**0**

In above table, '1' bit is found in '8', '16', '32' and '64' columns therefore '15<<3' gives result 8+16+32+64=**120**

Bitwise Right shift operator (>>):

This operator shifts the bits towards *right side* by a specified number of positions. Its symbol is denoted by '>>' which is called *double greater than* symbol. To understand the working of '>>' operator see following example.

E.g.:1) 25>>3

→

		128	64	32	16	8	4	2	1			
25	→	0	0	0	1	1	0	0	1			
				0	0	0	0	1	1			

In above table, '1' bit is found in '1', and '2' columns therefore '25>>3' gives result 1+2=**3**

Bitwise Zero Fill Rightshift operator (>>>):

- This operator also shifts the bits towards *right side* by a specified number of positions. But, it stores '0' in the sign bit. Its symbol is denoted by '>>>' which is called trip*le greater than* symbol. Since, it always fills '0' in the sign bit therefore it is called *zero fill right shift operator*.
- In case of negative numbers, its output will be positive because sign bit is filled with '0'

8) 'new' operator:

- 'new' operator is used to *create object of class*.
- We know that, objects are created on 'heap' memory by JVM dynamically.

Syntax to create object:

```
className      obj=new    className( );
```

Here,

'className' is name of the class.

'obj' is name of created object which is an identifier.

Example: Consider, there is class named 'Employee' then we create its object as follow,

```
Employee      emp = new    Employee( );
```

Here, 'emp' is an object of class 'Employee'

9) 'instanceof' operator:

- 'instanceof' operator is used *to check created object belongs to particular class or not*.
- Also, this operator used *to check created **reference** belongs to particular interface or not*.

Syntax:

```
boolean    var = obj    instanceof    className;
                        OR
boolean    var=  ref    instanceof    interfaceName;
```

Here,

'var' is variable of boolean data type.
'obj' is object of class.
'ref' is reference of interface.
'className' is name of class.
'interfaceName' is name of interface.

Example:

```
boolean    x = emp    instanceof    Employee;
```

Here, 'instanceof' operator checks an object 'emp' is an object of class 'Employee' or not.
If 'emp' is an object is class 'Employee' then 'instanceof' operator return *'true'* otherwise it returns *'false'*

```
// Program that demonstrate use of 'instanceof' operator
class worker
{
}
class   sangola
{
    public static void main(String[] args)
    {
            worker   wk=new   worker();
            boolean  x;
            x=wk    instanceof     worker;
            if(x==true)
                    System.out.println("It is instance of Worker class");
            else
                    System.out.println("It is not instance of Worker class");
    }
}
```

OUTPUT: It is instance of Worker class

10) 'cast' operator:

- cast operator is used to convert one data type into another data type.
- To convert data type of any variable or an expression, just we have to specify conversion data type before variable or expression within simple bracket (braces).

Example:

1)
```
double    x=15.26;
int       y=x;        //Error- because data type of 'x' and 'y' are different.
```

To store value of 'x' into 'y', we have to convert data type of 'x' into data type of 'y' as follow,

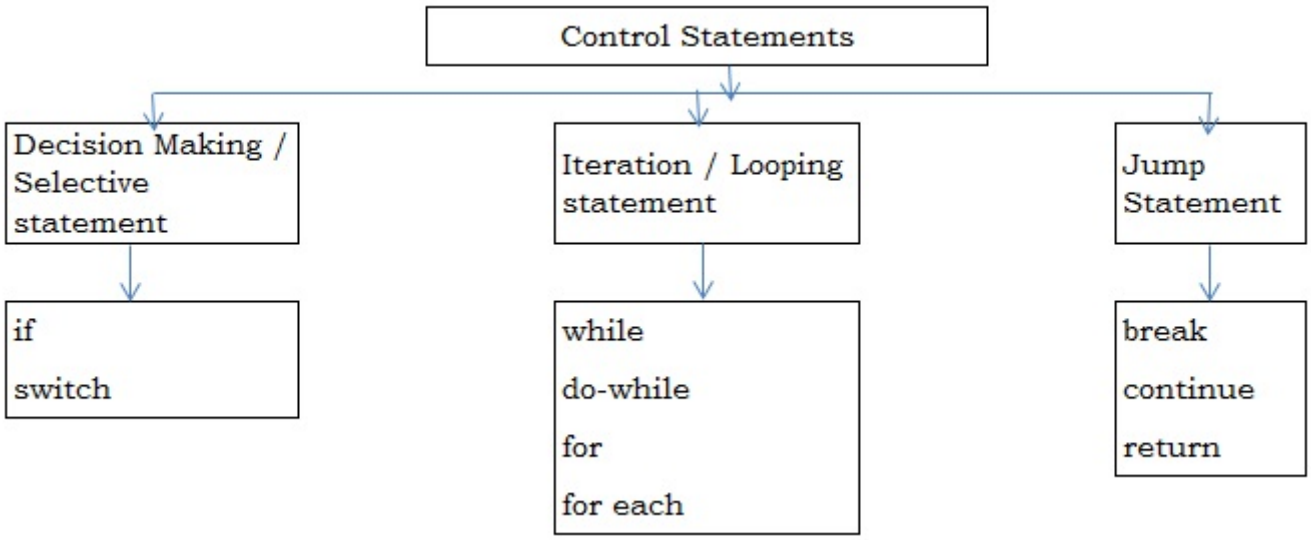
```
int  y= (int) x;    //here, the data type of 'x' is converted into data type of 'y' using (int) cast operator
```

2)
```
int       x=65;
char   y = (char) x;//here, the data type of 'x' is converted into data type of 'y' using (char) cast operator
```

Control Statement in Java:

The statement that controls the flow of execution of program is called as "Control statement" or "Control Structure".

The following tree diagram shows control statements in Java language:



(Note that: All the <u>Control Statement in Java is same as that of C/C++ language</u> therefore refer notes of C/C++ language)

<u>for-each loop:</u>

- This loop is specially designed to handle elements of 'collection'.
- *Collection* represents a group or set of elements or objects.

For example: We can take an 'array' as collection because 'array is set or group of elements

- Also, any class in '<u>java.util</u>' package can be considered as 'collection' <u>because any class in 'java.util' package handles group of objects such as 'stack', 'vector', 'LinkedList' etc</u>.
- The *for-each* loop repeatedly executes a group of statements for each element of the collection.
- The execution of for-each loop depends upon <u>total number of elements or objects present in the collection</u>.

<u>Syntax:</u>

```
for (datatype   var : collection )
{
    Statements;
}
```

Here,

'var' is an identifier which represents each element of collection one by one. Suppose, the collection has 5 elements then this loop will be executed 5 times and 'var' will *store each element of collection one by one*.

'datatype' is any valid data type in Java which is same as collection.

'collection' is any collection such as array, stack, linked list, vector etc.

```
// Program that demonstrate use of for-each loop
class      myloop
{
    public  static void main(String   [ ] args)
    {
            int     arr[ ]={5,6,7,8,9};

            for (int  i : arr )          //  'i' represents each element of 'arr'
            {
              System.out.println(i);
            }
    }
}
```

‘continue’ statement:

- ‘continue’ statement is specially used in looping statement.
- When ‘continue’ statement is executed then *control transferred back to check the condition in loop* and rest of statements are ignored.

```
// Syntax or execution of ‘continue’  statement

While ( condition1 )
{
        if ( condition2 )
        {
          continue;
        }
            -----------
            -----------
}
```

```
// Program that demonstrate use of ‘continue’ statement
class      myloop
{
    public  static void main(String   [  ]st)
    {
        int     i=10;
        while(i>=1)
        {
                if(i>5)
                    {
                            System.out.print(“\t”+i);
                            i--;
                            continue;
                    }
                else
                    {
                              i--;
                    }
        }
    }
}
```

OUTPUT: 10 9 8 7 6

Reading Input using ‘*Scanner*’ class of *‘java.util’* package:

- We can read varieties of inputs from keyboard or from text file using methods of ‘*Scanner*’ class.
- *Scanner* class belongs to ‘java.util’ package.
- When *Scanner* class receives input, it breaks the input into several pieces, called ‘tokens’ and these tokens can be retrieved using object of *Scanner* class.
- Note that: Following methods of *Scanner* class are *non-static* therefore they are called or accessed with the help of *object* of *Scanner* class.

We can create object of Scanner class as follows:

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

Here, ‘obj’ is object of *Scanner* class.
‘System.in’ represents *InputStream*, which is by default represents standard input device i.e. Keyboard.
There are several methods of *Scanner* class used to take different inputs as follows:

Method	Working
next( )	It is used to read single string
nextByte( )	It is used to read single byte type value
nextInt( )	It is used to read single integer type value
nextFloat( )	It is used to read single Float type value
nextLong( )	It is used to read single Long type value
nextDouble( )	It is used to read single Double type value
nextShort( )	It is used to read single short type value

Following program demonstrate the use of different methods of *Scanner* class.

```
import      java.util.Scanner;
class    CriClass
{
            byte     no;
            String   name;
            long     contact;
            int          t_sc;
            short    t_wk;
            float    ball_avg;
            double   bat_avg;
    public static void main(String []args)
    {
            Scanner   sc=new   Scanner(System.in);
            CriClass  obj=new  CriClass();
            System.out.print("Enter Cricketer No= ");
            obj.no=sc.nextByte();
            System.out.print("Enter Cricketer Name= ");
            obj.name=sc.next();
            System.out.print("Enter Cricketer Contact No= ");
            obj.contact=sc.nextLong();
            System.out.print("Enter Cricketer Total Score= ");
            obj.t_sc=sc.nextInt();
            System.out.print("Enter Cricketer Wickets= ");
            obj.t_wk=sc.nextShort();
            System.out.print("Enter Ball AVG= ");
            obj.ball_avg=sc.nextFloat();
            System.out.print("Enter Batting AVG= ");
            obj.bat_avg=sc.nextDouble();
            System.out.println("----------------------------------------");
            System.out.println("CricketerNO="+obj.no);
            System.out.println("CricketerName="+obj.name);
            System.out.println("ContactNo="+obj.contact);
            System.out.println("Total Score="+obj.t_sc);
            System.out.println("Total Wickets="+obj.t_wk);
            System.out.println("Balling AVG="+obj.ball_avg);
            System.out.println("Batting AVG="+obj.bat_avg);
    }
}
```

User defined methods in JAVA:

- Like C or C++ language, JAVA language also have 4 types of methods depending upon parameter acceptance or not and value return or not.
- Following program shows defining 4 types methods in Java language.

```
import      java.util.Scanner;
public    class     FunctionDemo
{
            int     x,y,z;
            Scanner   sc=new Scanner(System.in);
    void      add(int   a,int   b)       //with arg. without return value
    {
            int c;
            c=a+b;
            System.out.println("Addition="+c);
    }
    int     sub(int   a,int   b)      //with arg. with return value
    {
            int c;
            c=a-b;
            return(c);
    }
    int     multi()       //without arg. with return value
    {
            System.out.println("Enter Two Numbers=");
            x=sc.nextInt();
            y=sc.nextInt();
            z=x*y;
            return(z);
    }
    void   division()      //without arg. without return value
    {
            System.out.println("Enter Two Numbers=");
            x=sc.nextInt();
            y=sc.nextInt();
            z=x/y;
            System.out.println("Division="+z);
    }
    public static void main(String []args)
    {
            FunctionDemo    obj=new     FunctionDemo();
            System.out.println("Enter two number");
            obj.x=obj.sc.nextInt();
            obj.y=obj.sc.nextInt();
            obj.add(obj.x,obj.y);
            System.out.println("Enter two number");
            obj.x=obj.sc.nextInt();
            obj.y=obj.sc.nextInt();
            obj.z=obj.sub(obj.x,obj.y);
            System.out.println("Subtraction="+obj.z);
            obj.z=obj.multi();
            System.out.println("Multiplication="+obj.z);
            obj.division();
    }
}
```

Type casting: (Type conversion)

- Converting one data type into another data type is called “Type casting” or “Type-conversion”.
- We can convert the values from one type to another explicitly using the cast operator as follows.
- Syntax for type casting:

  (type_name) expression;
- Here, type_name is any valid data type into which we can convert value of expression.
-

Following program shows type casting that convert char data type into int data type.

```
import    java.util.Scanner;
public   class    typeCast
 {
   public   static   void   main(String []args )
   {
     char       ch;
     int        p;
     Scanner   sc=new  Scanner(System.in);
     System.out.println("Enter any Character:");
     ch=sc.next().charAt(0);  //reading single character
     p = (int) ch;          //type casting
     System.out.println("ASCII value="+p);
    }
}
```

Following program shows type casting that convert int data type into float data type.

```
import    java.util.Scanner;
public   class    typeCast
 {
   public   static   void   main(String []args )
   {
        int        a,b;
        float      p;
        Scanner   sc=new  Scanner(System.in);
  System.out.println("Enter any Two numbers=");
        a=sc.nextInt();
        b=sc.nextInt();
        p = (float) a/b;          //type casting
        System.out.println("Division="+p);
    }
}
```

Theory Assignment No: 01

1) What is Java? Explain its various features.
2) What is Java? Write its evolution. And list out its drawbacks.
3) Write difference between C and Java.
4) Write difference between C++ and Java.
5) Explain different components of JDK with their use.
6) Explain JVM architecture. **OR** Explain working of JVM **OR** How JVM works?
7) Why Java does not supports for Pointers?
8) What is Java Tokens? List out its different tokens.
9) What are the different naming conventions used in Java?
10) Explain for-each loop in Java.
11) What is type casting? How type casting is done in Java?

Practical Assignment No: 1

Note: Implement following programs by using command line Argument in JAVA

1) Write a program to print First name, Middle name and Last name of employee.
2) Write a program which find sum of even numbers and odd numbers from 1 to 20.
3) Write a program which prints first 'n' numbers.
4) Write a program which find sum of first 'n' numbers.
5) Write a program which prints factors of entered number.
6) Write a program which check entered number is Perfect or not.
7) Write a program which find sum of digits (digit sum) of entered number
8) Write a program which check entered number is Armstrong or not.
9) Write a program which reverses the entered number.
10) Write a program which check entered number is Palindrome or not.
11) Write a program which finds face value of entered number.
12) Write a program which check entered number is Prime or not.
13) Write a program which finds factorial of an entered number.
14) Write a program which prints Fibonacci series up to 'n' numbers.
15) Write a program to check entered number is Strong or not.

    *(**Hint:** Strong number is a special number whose sum of the factorial of digits is equal to the original number For Example: 145 is strong number. Since, 1! + 4! + 5! = 145)*

16) Write a program to check entered number is Magic or not.

    *(**Hint:** For example, 325 is a magic number because the sum of its digits (3+2+5) is 10, and again sum up the resultant (1+0), we get a single digit (1) as the result. Hence, the number 325 is a magic number. Some other magic numbers are 1234, 226, 10, 1, 37, 46, 55, 73, etc.)*

Practical Assignment: 02

Note: To accept inputs form keyboard use methods of 'Scanner' class of java.util

1) Write a program to find addition, subtraction, multiplication, division of two numbers.
2) Write a program to find average of five numbers.
3) Write a program to find area of circle.
4) Write a program to find circumference (perimeter) of circle.
5) Write a program to find area of triangle.
6) Write a program which accepts six subject marks and calculates total marks and percentage of student.
7) Write a program to calculate simple interest.
8) Write a program to calculate compound interest.
9) Write a program to swap two integers.
10) Write a program to find distance between two points.
11) Write a program to check entered number is positive or negative.
12) Write a program to check entered number is even or odd.
13) Write a program to check entered year is leap or not.
14) Write a program to find maximum number between three numbers.
15) Write a program to find minimum number between three numbers.

16) Write a program that demonstrate the use of 'instanceof' operator
17) Write a program that demonstrates the use of 'cast' operator.
18) Write a program which calculates total marks and percentage obtained in six subjects and also display grade of student according to following table:

Percentage	Grade
0 to 39.99	Fail
40 to 49.99	Third
50 to 59.99	Second
60 to 69.99	First
70 to 100	Distinction

19) Write a program which calculates income tax corresponding to Following table:

Income	Tax
0 to 150000	0%
150001 to 300000	10%
300001 to 500000	20%
500001 and above	30%

20) Write a program which calculates telephone bill corresponding to following table:

Unit Consumed	Rate/unit in RS.
0 to 200	1.00
201 to 350	1.20
351 to 500	1.50
501 and above	1.75

21) Write a program which take single digit number as input and print corresponding number into word.
22) Write a program menu driven program to find out area of circle, circumference (perimeter) of circle, area of triangle and area of square
23) Write a menu driven program for:
    1: Addition of two numbers.
    2: Subtraction of two numbers.
    3: Multiplication of two numbers.
    4: Division of two numbers.
    5: Modulation of Two numbers.
24) Write a program to print First name, Middle name and Last name of employee.
25) Write a program which find sum of even numbers and odd numbers from 1 to 20.
26) Write a program which prints first 'n' numbers.
27) Write a program which find sum of first 'n' numbers.
28) Write a program which prints factors of entered number.
29) Write a program which check entered number is Perfect or not.
30) Write a program which find sum of digits (digit sum) of entered number
31) Write a program which check entered number is Armstrong or not.
32) Write a program which reverses the entered number.
33) Write a program which check entered number is Palindrome or not.
34) Write a program which finds face value of entered number.
35) Write a program which check entered number is Prime or not.
36) Write a program which finds factorial of an entered number.
37) Write a program which prints Fibonacci series up to 'n' numbers.
38) Write a program to check entered number is Strong or not.

39) Write a program to check entered number is Magic or not.

40) Write a program to find all Armstrong numbers from 1 to 1000

41) Write a program to find all Prime numbers from 1 to 1000

42) Write a program to find all palindrome numbers from 500 to 700

43) Write a program which prints multiplication table

Practical Assignment: 03

Que. Write the program that prints following pattern:

1)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2)

```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
```

3)

```
A B C D C B A
A B C   C B A
A B       B A
A           A
```

4)

```
            A
          A B
        A B C
      A B C D
    A B C D E
```

5)

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
1 2
1 2 3
1 2 3 4
1 2 3 4
1 2 3 4 5
```

6)

```
1
2 3
4 5 6
7 8 9 10
```

7)

```
1
0 1
0 1 0
1 0 1 0
```

8)

```
@
@@
@@@
@@@@
@@@@@
@@@@
@@@
@@
@
```

9)

```
@
@@
@@@
@@@@
@@@@@
@@@@@@
```

10)

```
@
@@
@@@
@@@@
@@@@@
@@@@@@
@@@@@
@@@@
@@@
@@
@
```

11)

```
$ $ $ $ $ $ $ $ $ $
$ $ $ $ $ $ $ $ $
$ $ $ $ $ $ $ $
$ $ $ $ $ $ $
$ $ $ $ $ $
$ $ $ $ $
$ $ $ $
$ $ $
$ $
$
```