Module-1
Chapter-1
An introduction to programming Language

What is a program?
A computer program is a collection of instructions that can be executed by a computer to perform a specific task. A computer program is usually written by a computer programmer in a programming language.

Programming language

A **programming language** is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms. Most programming languages consist of instructions for computers.

Types of programming languages

There are two **types of programming languages**, which can be categorized into the following ways

1. Low level language
   a) Machine language (1GL)
   b) Assembly language (2GL)
2. High level language
   a) Procedural-Oriented language (3GL)
   b) Problem-Oriented language (4GL)
   c) Natural language (5GL)

1. Low level language:

- This language is the most understandable language used by the computer to perform its operations. It can be further categorized as

a) **Machine Language (1GL):** Machine language consists of strings of binary numbers (i.e. 0s and 1s) which the processor directly understands. Machine language has the merits of very fast execution speed and efficient use of primary memory.

Merits

- It is directly understood by the processor. So it has faster execution time since the programs written in this language need not to be translated. It doesn't need larger memory.

Demerits

- It is a very difficult task writing program in machine level language since all the instructions are to be represented by 0s and 1s.
- Use of this machine level language makes programming time consuming.
- It is difficult to find an error and to debug them in machine level language.
- It can be used by experts only.

b) **Assembly Language:** Assembly language is also known as low-level language because to design a program, the programmer requires detailed knowledge of hardware specification. This language uses mnemonics code such as ('ADD' for addition, 'STORE' for keeping data etc') in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.

Merits

- It makes programming easier than machine level language since it uses mnemonics code for programming. Eg: ADD for addition, SUB for subtraction, DIV for division, etc.
- It makes the programming process faster.
- Error can be identified much easily as compared to machine level language.
- It is easier to debug than machine language.

Demerits

- Programs written in this language is not directly understood by the computer. So a translator is required which will translate from assembly language to object code.
- It is the hardware dependent language so programmers are forced to think in terms of computer's architecture rather than the problem being solved.
- Since it is the machine dependent language, programs written in this language are very less or not portable.
- Programmers must know its mnemonics codes to perform any task.

2. High level language:

- Instructions written in this language are closely resembles to human languages or English like words. It uses mathematical notations to perform the task. The high-level language is easier to learn. It requires less time to write and is easier to maintain the errors. The high-level language is converted into machine language by one of the two different languages translator, interpreter or compiler.
- High level language can be further categorized as

a) Procedural-Oriented language (3GL): Procedural Programming is a methodology for modelling the problem being solved by determining the steps and the order of those steps that must be followed in order to reach a desired outcome or specific program state. These languages are designed to express the logic and the procedure of a problem to be solved. It includes languages such as Pascal, COBOL, C, FORTAN, etc.

Merits

- Because of its flexibility, procedural language is able to solve a variety of problems. Programmer does not need to think in term of computer architecture which makes them focused on the problem.
- Programs written in this language are portable.

Demerits

- It is easier but needs higher processor and larger memory. It needs to be translated. So its execution time is more.

b) Problem-Oriented language (4GL): It allows the users to specify what the output should be, without describing all the details of how the data should be manipulated to produce the result. This is one step ahead from 3GL. These are result oriented and include database query language, eg: Visual Basic, C#, PHP, etc.

The objectives of 4GL

- are to increase the speed of developing programs.
- Minimize user's effort to obtain information from computer.

- Reduce errors while writing programs.

Merits

- Programmer need not to think about the procedure of the program. So, programming is much easier.

Demeraits

- It is easier but needs higher processor and larger memory.
- It needs to be translated. So its execution time is more.

c) Natural language (5GL): Natural language are still in developing stage where we could write statements that would look like normal sentences.

Merits

- It is easy to program. Since, the program uses normal sentences, they are easy to understand. The programs designed using 5GL will have artificial intelligence (AI).
- The programs would be much more interactive and interesting.

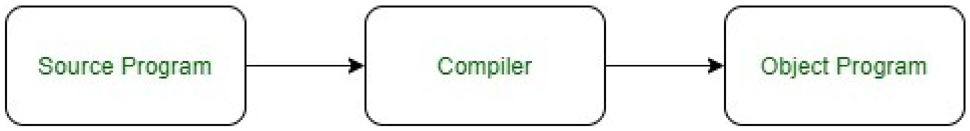
Demerits

- It is slower than previous generation language as it should be completely translated into binary code which is a tedious task. Highly advanced and expensive electronic devices are required to run programs developed in 5GL. Therefore, it is an expensive approach.

Compiler and Interpreter

1. Compiler

- It is a translator which takes input from high Level Language and produces an output in machine level or assembly language.
- Compiler is more intelligent than an assembler. It checks all kinds of limits, ranges, errors etc. But it's program run time is more and occupies a larger part of memory. Its speed is slow because a compiler goes through the entire program and then translates the entire program into machine codes.

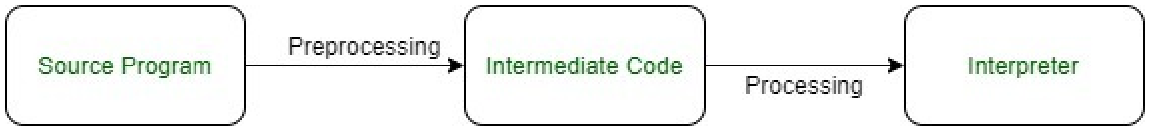


- In the context of java programming, we can say, compiler is a program that translates code written in a high-level programming language (like JavaScript or Java) into low-level code (like Assembly) directly executable by the computer or another program such as a virtual machine.
- For example, the Java compiler converts Java code to Java Bytecode executable by the JVM (Java Virtual Machine). Other examples are V8, the JavaScript engine from Google which converts JavaScript code to machine code or GCC which can convert code written in programming languages like C, C++, Objective-C, Go among others to native machine code.

2. Interpreter

- An interpreter is a program which translates a programming language into a comprehensible language.
- It translates only one statement of the program at a time.

- Interpreters, more often are smaller than compilers.



Difference between Compiler and Interpreter

- Interpreters and compilers are very similar in structure. The main difference is that an interpreter directly executes the instructions in the source programming language while a compiler translates those instructions into efficient machine code.
- An interpreter will typically generate an efficient intermediate representation and immediately evaluate it. Depending on the interpreter, the intermediate representation can be an *AST*, an *annotated AST* or a machine-independent low-level representation such as the *three-address code*.

Compiler	Interpreter
Compiler scans the whole program in one go.	Translates program one statement at a time.
As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
Main advantage of compilers is it’s execution time.	Due to interpreters being slow in executing the object code, it is preferred less.
It converts the the instructions into systematic code.	It doesn’t convert the instructions instead it directly works on source language.
Example: C, C++, C# etc.	Example: Python, Ruby, Perl, SNOBOL, MATLAB etc.

Introduction to Object-oriented programming language

Object oriented thinking and basics need for oops paradigm:

- **Object-oriented programming** (OOP) is a programming **paradigm** based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.
- Since the 1980s the word 'object' has appeared in relation to programming languages, with almost all languages developed since 1990 having object-oriented features. Some languages have even had object-oriented features. It is widely accepted that object-oriented programming is the most important and powerful way of creating software.
- Due to the various problems faced by the procedural languages, the concept of OOP was introduced. Therefore, it has a number of advantages over the procedural programming languages.

Let us discuss some of the advantages of the Object-oriented programming:

1. Real World Entities:
    - In OOP real world entities are used. Classes and objects can be made of the things that are real and exist in the world. Example of real-world entities are pen, chair, student, hospital, etc. these examples can be considered as classes and their attributes will be considered as their object.
    - Let's discuss other example in order to better understand the classes and objects. Suppose, we take a student as class, then objects will be student name, roll number, section, address, mobile number, email, etc.
2. Code Reusability:
    - Code reusability is one of the characteristics of object-oriented programming. The feature that explains this point is inheritance. In inheritance, the class and subclasses or parent and child classes can be derived and its data member and member functions can be used as such. This feature saves time and the user do not need to code again and again, if similar features or functionality is required. So, a lot of time can be saved.
3. Easy Management
    - Code management becomes very easy in the object-oriented programming. As all the code is divided into a number of elements it becomes easy to manage. For instance, the whole program can be termed as a class and even if it contains a number of functions are written or coded in it, their objects can be made. This makes it easy to manage the code, as later you can initialize the object variable and make the function call.
4. Maintenance:
    - Maintenance of code also becomes easy in object-oriented programming. Because of easy management of the code maintenance also becomes easy. If the code is to be used by another programmer, still it will not create any ambiguity or correct guiding of coding is used.
5. Abstraction:
    - In abstraction, only the useful data is visible to the user and not the things which are not required. Abstraction is also called as data hiding. For example, when we login into an email id, we can only type the login Id and password and view it. We do not know or cannot see how the data is being verified and how it is being stored. The user is always not aware about where their id and password are going. In which database it is stored? What are the criteria for checking it? etc. Hence, it is very beneficial practice to use this feature as it provides data security.
6. Polymorphism:
    - It is another feature of the object-oriented programming. Polymorphism simply means that a function has many forms. The '+' operator, for example, it can be used to add two integers whereas used to join two strings. Similarly, even in the programming one function can be written, this function can be used in different forms depending on its arguments. It is called function overloading.

The advantages of OOP are mentioned below:

- OOP provides a clear modular structure for programs.
- It is good for defining abstract data types.

- Implementation details are hidden from other modules and other modules has a clearly defined interface.
- It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages
- encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.
- It implements real life scenario.
- In OOP, programmer not only defines data types but also deals with operations applied for data structures.
- Each object can be easily developed and maintain in program.
- Data hiding and data abstraction increases reliability//y and provide the security.
- It helps to re-usability of code.
- Dynamic binding increases flexibility by permitting the addition of new classes.

Benefits of Oops:

- simulation and modeling.
- User interface design.
- Artificial intelligence and expert systems.
- Neural networks.
- Web designing.
- Component object model(COM).

Summary of OOPs Concepts:

The following are the fundamental OOPs concepts.

- Everything is an Object.
- Computation is performed by the objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action bundled with arguments may be necessary to complete the task.
- Each object has its own memory, which consists of other objects.
- Every object is an instance of a class. A class simply represents a grouping of similar objects such as integers or lists.
- The class is the repository for behavior associated with an object. That is, all objects that are instances of the same class can perform the same actions.
- Classes are organized into a singly rooted tree structure, called the inheritance hierarchy. Memory and behavior associated with instances of a class are automatically available to any class associated with descendent in this tree structure.

Chapter-2
An introduction to Java

What is java?

- Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the Oak name to Java.
- Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX such as sun-Solaris, red-hat, ubuntu etc. Java is a popular third generation programming language, which can do any of the thousands of things that a computer software can do. With the features it offers, java has become the language of choice for internet and intranet applications. The kind of functionality the java offers, has contributed a lot towards the popularity of java.

History of Java Program

- Java was conceived by James Gosling in 1991. Initially it was called as Oak. But it was renamed as Java in 1995. The language evolved from a programming language known as the Oak. Oak was developed in the early 1990s by Sun Microsystems as platform that would support entertainment applications such as video games. Because it implements Object Oriented Principles it is called as Object Oriented Programming Language.
- Originally java started as an elite project (codenamed Green) to find a way of allowing different devices such as TV-top boxes and controllers to use a common language. This language for electronic devices was originally named Oak but failed to find a niche despite its potential. But with the explosion of world, java rose to charts of popularity as it could cater to nearly all platforms. In the following lines, the history of java can be found out
    - 1991 – *James gosling* develops Oak (later named Java) language for programming intelligent consumer electronic devices.
    - 1995 – Java formally announced. Incorporated into Netscape web browser.
    - 1997 –JDK 1.1 launched. Java Servlet API released.
    - 1998–Sun introduces Community source "open licensing". Sun produces a JDK 1.2 for Linux.
    - 2001–JDK1.3 was released, J2EE(Java enterprise edition),J2SE.J2ME appear.
    - 2002–Java support for web services officially released via the java^{TM} web service developer pack.
    - 2005–Java technology –enabled services devices are available.
    - 2006–The NetbeanIDE 5.0 is released.

Applications:

There are 2 basic types of Java applications:

1. Standalone: These run as a normal program on the computer. They may be a simple console application or a windowed application. These programs have the same capabilities of any program on the system. For example, they may read and write files.

Just as for other languages, it is easily to write a Java console program than a windowed program.

2. Applets: These run inside a web browser. They must be windowed and have limited power. They run in a restricted JVM (Java Virtual Machine) called the sandbox from which file I/O and printing are impossible. (There are ways for applets to be given more power.)

Difference between c++ and Java

- C++ uses only compiler, whereas Java uses compiler and interpreter both.
- C++ supports both operator overloading & method overloading whereas Java only supports method overloading.
- C++ supports manual object management with the help of new and delete keywords whereas Java has built-in automatic garbage collection.
- C++ supports structures whereas Java doesn't support structures.
- C++ supports unions while Java doesn't support unions.

Oops concepts in java:

Object oriented program concepts are used to remove some of the existing problems in procedural approach.

- It ties the data as a critical element.
- It doesn't allow to flow freely around the system.
- It protects and provides security use to decompose of a problem into number of entities called objects.
- The data can be accessed only by the functions.
- The earlier languages are C, C++ are not having the capability to simplify the complexity of problem due to involvement of object concept the simplification is to build the tangible project.

Java supports the following object-oriented programming concepts.

- object
- class
- data encapsulation
- data abstraction
- inheritance
- polymorphism
- data binding
- message passing

Generally, main pillars of oops concepts are data encapsulation, data abstraction, inheritance and polymorphism. Rest of the concepts supports to build the object-oriented programming.

1. Object: Object is nothing but a runtime entity which having member variables and Member functions.

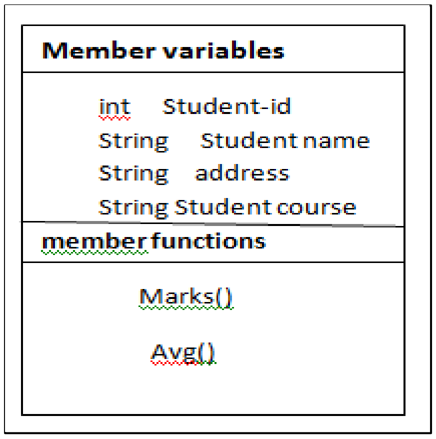
- It can be represented with "new" keyword.
- In object the data associated with function hence it cannot access directly.
- It can access only through functions.
- It can be defined as a vector, person, place and account.

2.Classes: Collection of objects are called classes.It contains member variables and member functions.

- It can be represented with "class" keyword.
- Once we create a class, creates any number of objects (belongs) within the class.

Eg: fruit is a class, it may consider any number of fruit related objects like mango, banana, apple…….

Syntax:
```
class <classname>{                    {
  member variables;
  member functions;
  }
```
Example:



3. Data encapsulation:

- It is a mechanism, to **hide the data properties** in a class called encapsulation or **wrapping-up of data** and functions in a given class is also called encapsulation.
- It is extracted from "capsule" word, how the capsule wraps up hidden parts and it act as a safe guard and provide the security.
- The data is not accessible from outside world. Only wrapped functions can access. It is core part of java.

4. Data abstraction: It is a mechanism, which will hide the implementation details in a given class. Hence it provides security and binding data and methods together in a single unit. It refers with "Abstract Data Type" (ADT)

- Eg: in mobile communication a user can know only usage of lift call and cancel call, don't know the implementation of background coding, networking and routing parts.
- Eg: A driver can drive the car by the functioning of steering, accelerator and gears. But don't know implementation mechanism of engine and circuits.

5.Inheritance: It is an ancient property which derives some common properties from base class to sub class is called inheritance

- It provides the re-usability of code.
- Data shares in the program as well as hide the data
- We can also add additional features to the existing class.

Syntax:
```
class <class name>
{
        instance variables;
        instance methods();
}
Class <subclass name> extends class <basecalss name>
{
Instance variables;
Instance methods();
```

}

6. **Polymorphism**: It is a Greek term Poly means "many", morphs mean "forms".
   - It is an ability to take more than one form is called polymorphism
   - It exhibits different behaviors in different instances. It depends upon type of data used in the operation.
   - Polymorphism classified in two types

   a) Compile time (static):
   - It allocates memory at compile time in program execution.
   - It is also called early binding polymorphism.
   - Example: Method of overloading

   b) Runtime (dynamic):
   - It allocates memory at run time in program execution.
   - It is also called late binding polymorphism.
   - Example: Method of overriding

   Eg: Consider the operation of addition for 2 numbers it will generates the sum of integers.

   A=10, B=20 then, A+B=30

   If the operands/strings, the operation would produce third party of string by applying like concatenation operator called operator overloading.

   Eg: string 1+ string 2= string 3

7. **Data binding**:
   - Binding is a method of calling code by itself.
   - Data binding are of two types: static binding and dynamic binding.
   - Static binding: It occurs during compile time. Private, final, static methods as well as variables involved in static binding.
   - Dynamic binding: It occurs at run time. virtual methods are bonded during run time based upon runtime object.

8. **Message passing**:
   - It is a form of communication used in parallel programming.
   - The object-oriented programming communications are completed by the sending of messages like functions, signals and data packets.

JAVA BUZZ WORDS

The following is the list of Java Buzz words.

- Simple
- Secure
- Portable
- Object Oriented
- Robust
- Multithreaded
- Architecture – neutral
- Interpreted
- High Performance
- Distributed
- Dynamic

1. **Simple**: Java inherits the C/ C++ syntax and many object-oriented features of C++. This makes most programmers to learn java easy.
2. **Secure**: When a java compatible web browser is used, it is safe to down load the java applets. Java achieves this protection by confining a java program to the java execution environment and not allowing it access to other parts of the computer.

3. **Portable:** It supports WORA (Write Once Run Anywhere) properties of Java. The same java program will run, without change the operating system. For programs to be dynamically downloaded to all the various types of platforms connected to the internet, portable executable code is needed. The java compiler generates the portable code **(byte code)** which is executed by the java virtual machine (JVM) on the computer.
4. **Object Oriented:** Java was designed that everything is like an object. The object model is simple and simple and easy to extend.
5. **Robust:** Java programs behave uniformly across different platforms. Java program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of java. Because java is a strictly typed language, it checks the code compile time and also at run time. To better understand how java is robust, consider two main reasons for program failure.
    - Memory management
    - Exception handling
6. **Multithreaded:** Java supports multithreaded programming, which allows us to write program that do many things simultaneously i.e execute multiple processes simultaneously.
7. **Architecture-Neutral:** Java designers provides a JVM which enables you to write a program once and execute it on different machines. It made the program longevity and portable. The goal of the java designers was **"write once run anywhere, anytime, forever."**
8. **Interpreted and High performance:** Java enables the creation of cross platform program by compiling into an intermediate code representation called java byte code. This code can be executed on any machine that implements JVM.
9. **Distributed:** As java handled TCP/IP protocols it is possible to write distributed programs for internet. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.
10. **Dynamic:** Java programs carry with them substantial amount of runtime type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe manner. This is crucial to the robustness of the applet environment, in which small fragments of byte code may be dynamically updated on a running system.

<u>SIMPLE JAVA PROGRAM</u>

- In java, a source file is officially called a compilation unit. It is a text file that contains one or more class definitions.
- By convention, the name of that class should match the name of the file that holds the program.
- Java is **case sensitive programming language.**
- **A simple Java program consists of:**

1. The **comments** are ignored by the compiler by using special symbols like

    // - for single line comment

    /* and end with */ - for multiline comment.

- The class keyword is used to declare a class and follows the class name.

2. class Example

- The entire class definition, including all of its members, will be between the opening curly brace ({) and closing curly brace (})

3. public static void main(String args[])

- All java applications starts execution at main method.
- The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members.
- main() must be declared as public when it is called by code outside of its class when the program is started.
- The keyword **static** allows main()to be called without having instantiate a particular instantiate a particular instance of the class.
- This is necessary since main() is called by the JVM before any objects are made.
- The keyword **void** tells the compiler that main() does not return any value.
- In main(), there is only one parameter that receives String parameter.
- To display the output, **System** is a predefined class that provides access to the system and **out** is the output stream that is connected to the console.
- Output is actually accomplished by the built in function called **println()**

Write a Java program print simple "Hello" using standalone program concept,

```
public class Hello
    {
public static void main( String args[] )
     {
 System.out.println("Hello");
        }
      }
```

A) Creating a Java source file

- Java source files must end in an **.java** extension. The root name must be the same as the name of the one public class in the source file. In the program above, the class is named Hello and thus, the file must be named **Hello.java**.
- Just as for other languages, any text editor can be used to save the text of the program into a text file.

B) Compiling a Java source file

- Sun's JDK includes a Java compiler named javac. To compile the above Java program one would type:

  ```
  javac  Hello.java
  ```
- If successful, this creates a file named Hello.class If not successful, it prints out error messages like other compilers. The output of a Java compiler is not executable code. It is bytecode.
- Bytecode is a set of instructions designed by the Java run time system, which is called the Java Virtual Machine. The JVM is an interpreter for bytecode. This running JVM then executes the Java bytecode which is platform independent, provided that you have a JVM available for it to execute.

C) Running a Java program

- To run a standalone program, Sun's JDK provides a Java interpreter called java. To run the .**class** file created above, type:

  ```
  java Hello
  ```
- Note that .**class** is not specified. The output of running the above program is simply: **Hello**

What is JVM?

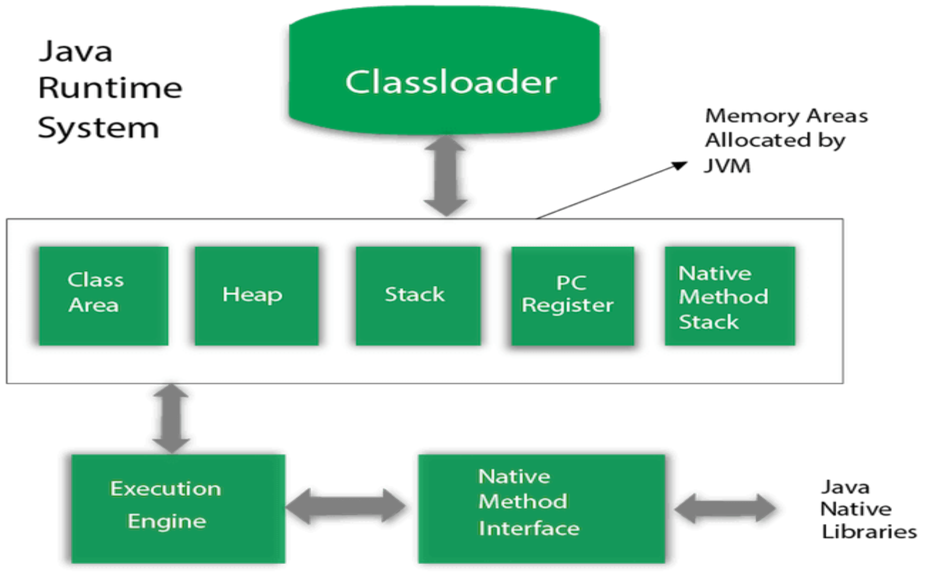
- A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The javac compiler does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).
- Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the machine language for JVM is byte code. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. JVM is the one that makes java platform independent.
- Hence, primary function of JVM is to execute the byte code produced by compiler. Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems. Which means that the byte code generated on Windows can be run on Mac OS and vice versa. That is why we call java as platform independent language.
- Java Virtual Machine converts byte code to the machine-specific code. JVM runs the program by using libraries and files given by Java Runtime Environment. It can execute the java program line by line hence it is also called as interpreter.
- JVM is easily customizable for example, we can allocate minimum and maximum memory to it.
- It is independent from hardware and the operating system. So, you can write a java program once and run anywhere.

Significance of JVM

- JVM provides a platform-independent way of executing Java source code.
- It has numerous libraries, tools, and frameworks.
- Once you run Java program, you can run on any platform and save lots of time.
- JVM comes with JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs more faster as a regular application.

Architecture of JVM

- Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



Let us discuss the working principle

- **Class Loader:** The class loader reads the .class file and save the byte code in the **method area**.
- **Method Area**: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.
- **Heap**: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.
- **Stack**: Stack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.
- **PC Registers**: This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.
- **Native Method stack:** A native method can access the runtime data areas of the virtual machine.
- **Native Method interface**: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.
- **Garbage collection**: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

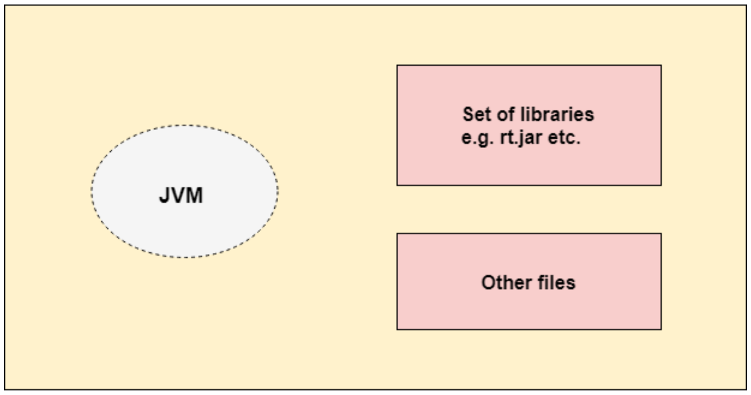
Difference between JDK, JRE, and JVM

JVM:

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.
- JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.
- The JVM performs the following main tasks:
    - Loads code
    - Verifies code
    - Executes code
    - Provides runtime environment

JRE:

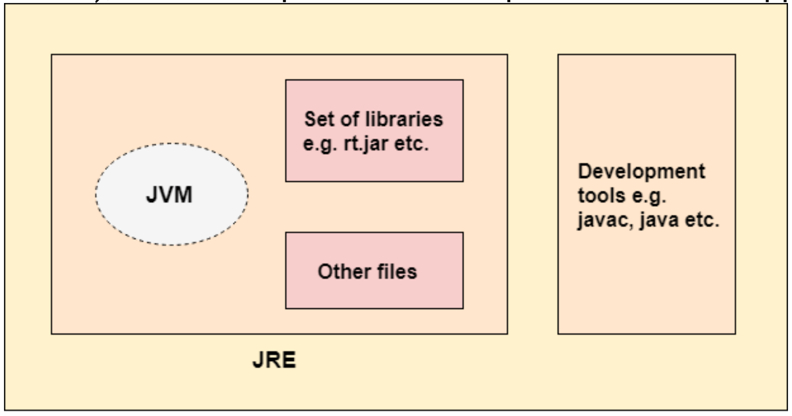
- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
- The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JRE

JDK:

- JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.
- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
    - Standard Edition Java Platform
    - Enterprise Edition Java Platform
    - Micro Edition Java Platform
- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



JDK

Chapter - 3

Java tokens

- In Java, a program is a collection of classes and methods, while methods are a collection of various expressions and statements. Tokens in Java are the small units of code which a Java compiler uses for constructing those statements and expressions. Java supports 5 types of tokens which are:
    - Keywords
    - Identifiers
    - Literals
    - Operators
    - Special Symbols

a) Keywords

- Keywords in Java are predefined or reserved words that have special meaning to the Java compiler. Each keyword is assigned a special task or function and cannot be changed by the user. We cannot use keywords as variables or identifiers as they are a part of Java syntax itself.
- A keyword should always be written in lowercase as Java is a case sensitive language.
- Java supports various keywords, some of them are listed below:

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. throw	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

b) Identifier

- Java Identifiers are the user-defined names of variables, methods, classes, arrays, packages, and interfaces. Once we assign an identifier in the Java program, we can use it to refer the value associated with that identifier in later statements.
- There are some de facto standards which you must follow while naming the identifiers such as:
  1. Identifiers must begin with a letter, dollar sign or underscore.
  2. Apart from the first character, an identifier can have any combination of characters.
  3. Identifiers in Java are case sensitive.
  4. Java Identifiers can be of any length.
  5. Identifier name cannot contain white spaces.
  6. Any identifier name must not begin with a digit but can contain digits within.
  7. **keywords** can't be used as identifiers in Java.

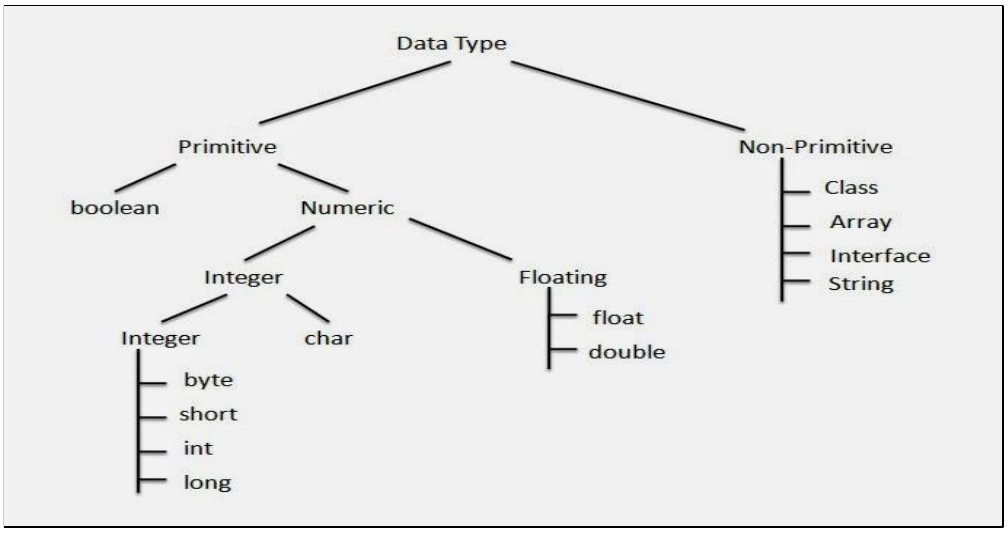
c) Literals

- Literals in Java are similar to normal variables but their values cannot be changed once assigned. In other words, literals are constant variables with fixed values. These are defined by users and can belong to any data type.
- Java supports five types of literals which are as follows:

i. Integer: e.g. 5,15,100 etc
ii. Floating Point: e.g. 5.5, 15.25, 105.538 etc
iii. Character: e.g. 'A', 'b', 'z', 'X' etc.
iv. String: e.g. "ABIT", "CSE", "Cuttack"
v. Boolean: e.g. TRUE and FALSE

Data types in java

- Data types specify the classification of data, it allocates the memory for variable it tells what kind of data values is stored in it.
- Each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types. These are two types
  1. Primitive data type
  2. Non primitive data type



Primitive Datatype:

i) boolean

- boolean data type represents only one bit of information either true or false . Values of type boolean are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.
  Example: // A Java program to demonstrate boolean data type

```
class booldatatype
{
   public static void main(String args[])
   {
      boolean b = true;
      if (b == true)
         System.out.println("welcome dear students");
   }
}
```

ii) byte

- The byte data type is an 8-bit signed two's complement integer. The byte data type is useful for saving memory in large arrays.
- Size: 8-bit (1 byte)
- Value: -128 to 127
  Example : class JavaExample {

```
     public static void main(String[] args) {
            byte num;
            num = 113;
            System.out.println(num);
          }
        }
```

iii) short

- The short data type is a 16-bit signed two's complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.
- Size: 16 bit (2-bytes)
- Value: -32,768 to 32,767 (inclusive)

iv) int

- It is a 32-bit signed two's complement integer.
- Size: 32 bit
- Value: -231 to 231-1

Example:
```
class JavaExample {
     public static void main(String[] args) {
    short num;
   num = 150;
 System.out.println(num);
    }
   }
```

v) long:

- The long data type is a 64-bit two's complement integer.
- Size: 64 bit
- Value: -263 to 263-1.

Example:
```
class Java Example {
    public static void main(String[] args) {
   long num = -12332252626L;
   System.out.println(num);
     }
  }
```

vi) float:

- The float data type is a single-precision 32-bit IEEE 754 floating point. Use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- Size: 32 bits
- Suffix: F/f
- Example: 9.8f

Example:
```
class JavaExample{
 public static void main(String[] args) {
 float num = 19.98f;
 System.out.println(num);
 }
 }
```

vii) double:

- The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice.

Example:
```
class JavaExample{
 public static void main(String[] args) {
 double num = -42937737.9d;
 System.out.println(num);
  }
 }
```

viii) char

- The char data type is a single 16-bit Unicode character. A char is a single character.
- Value: '\u0000' (or 0) to '\uffff' 65535

Example:
```
class JavaExample
 {
public static void main(String[] args) {
char ch = 'Z';
System.out.println(ch);
}
}
```

<u>Non-primitive Data types:</u>

- These data types are derived from primitive data types. They are:

i. **Reference:** reference data types are the more sophisticated members of the data type family. They don't store the value, but store a reference to that value Java keeps the reference, also called address, to that value, not the value itself. These are reference variables.

ii. **Array:** An **array** is a single object that contains multiple values of the same type. There is nothing special about arrays of objects (non-primitives) given a type T, an array of references to T is defined.

Syntax: T[] array= new T[42]; // declares an array of 42 elements

iii. **Class:** In order to create a new non-primitive or reference variable for this class, we have to create a new instance of the Product class. The new keyword is used to create an object

Example:
```
class product
 {
//Body of the code
  }
product P=new product (); //P is the object of class product
```

iv. **Interface:** An interface is like a dashboard or control panel for a class which contains abstract methods and final static variables. It has the buttons, but the function is elsewhere. We won't go into detail on implementing interfaces since the focus is on the interface as a non-primitive, or reference, data type.

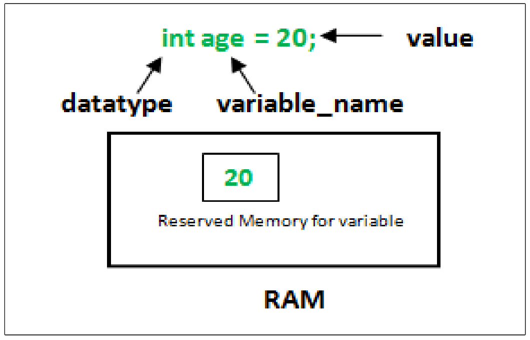
Example:
```
interface products
{
Public void getitemid();
Public void getprice();
}
```

<u>Variables in Java</u>

- A variable is the name given to a memory location. It is the basic unit of storage in a

program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location; all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before they can be used.

Declarations of variables



- **datatype**:Type of data that can be stored in this variable.
- **variable_name**: Name given to the variable.
- **value**: It is the initial value stored in the variable.

Examples:
- float simpleInterest; //Declaring float variable
- int time = 10, speed = 20; //Declaring and Initializing integer variable
- char var = 'h'; // Declaring and Initializing character variable

There are three types of variables in Java:
1. Local Variables
2. Instance Variables
3. Static Variables

1. Local Variables:
- A variable defined within a block or method or constructor is called local variable.
- These variables are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variables only within that block.

Example:
```
public class StudentDetails
 {
  public void StudentAge()
    {  //local variable age
   int age = 0;
  age = age + 5;
  System.out.println("Student age is : " + age);
    }
    public static void main(String args[])
       {
    StudentDetails obj = new StudentDetails();
    obj.StudentAge();
        }
   }
```

2. Instance Variables:

- Instance variables are non-static variables and are declared in a class outside any method, constructor or block.
- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

Example:

```
import java.io.*;
class Marks
{
   //These variables are instance variables.
   //These variables are in a class and are not inside any function
   int engMarks;
   int mathsMarks;
   int phyMarks;
}
class MarksDemo
{
   public static void main(String args[])
   {   //first object
      Marks obj1 = new Marks();
      obj1.engMarks = 50;
      obj1.mathsMarks = 80;
      obj1.phyMarks = 90;
       //second object
      Marks obj2 = new Marks();
      obj2.engMarks = 80;
      obj2.mathsMarks = 60;
      obj2.phyMarks = 85;
       //displaying marks for first object
      System.out.println("Marks for first object:");
      System.out.println(obj1.engMarks);
      System.out.println(obj1.mathsMarks);
      System.out.println(obj1.phyMarks);

      //displaying marks for second object
      System.out.println("Marks for second object:");
      System.out.println(obj2.engMarks);
      System.out.println(obj2.mathsMarks);
      System.out.println(obj2.phyMarks);
   }
```

```
        }
```

3. Static Variables:

- Static variables are also known as Class variables.
- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at start of program execution and destroyed automatically when execution ends.
- To access static variables, we need not to create any object of that class, we can simply access the variable as: class_name.variable_name;

Example:

```
import java.io.*;
class Emp {
    // static variable salary
   public static double salary;
   public static String name = "Harsh";
}
 public class EmpDemo
{
     public static void main(String args[]) {
        //accessing static variable without object
      Emp.salary = 1000;
      System.out.println(Emp.name + "'s average salary:" + Emp.salary);
    }
    }
```

SCOPE AND LIFETIME OF VARIABLES

- Java allows the variables to be declared within any block. A block defines a begun with an opening curly brace and ended by a closing curly brace.
- A block defines a scope. Thus each time you start a new block, you are creating a new scope.
- A scope determines what objects are visible to other parts of the program.
- It also determines the lifetime of those objects.

There are two major scopes in java.

  - class scope: Variable or methods defined within a class is called to be in class scope.
  - Method scope: Variable or methods defined within a function is called to be in method scope.

NOTE:

- Variables declared within the scope are not visible to the code that is defined outside that scope. Scopes can be nested.
- The objects declared in the outer scope will be visible to the code within inner scope.

But the objects declared within inner scope will not be visible outside it.

Example: class Scope
```
{
   public static void main(String args[])
      {
           int x=10;
           if(x==10)
              {
                  int y=20;
                   System.out.println(“x= “+x+”y= “+y);
               }
           System.out.println(“x=”+x);
       }
   }
```

Operators in java

Definition: Operator is a symbol which will operate on one or more than one operand. Hence operator can be of two types:

1. Unary: Operator that works on a single operand.
2. Binary: Operator that works on two operands.

Example: A=10,B=20
```
 C=A+B//Binary operator + adds A and B, result is 30
D=-B // Unary operator (-) negates value of B, result is -20
```

Types of operator:

1.Arithmetic Operators
2.Unary Operators
3.Assignment Operators
4.Relational Operators
5.Logical Operators
6.Ternary Operators
7.Bitwise Operators
8.Shift Operators

1.Arithmetic Operators: Java Arithmetic operators are used for simple math operations, they are:

Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulo (%)

Example
```
public class operators
{
public static void main(String[] args)
{
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
String x = "Thank", y = "You";
// + and - operator
System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));
// + operator if used with strings
// concatenates the given strings.
```

```
        System.out.println("x + y = "+x + y);
        // * and / operator
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));
        // modulo operator gives remainder on dividing first operand with second
        System.out.println("a % b = "+(a % b));
        /*if denominator is 0 in division then Arithmetic exception is thrown. Uncommenting
        below line would throw an exception, will be discussed in Exception Handling concept*/
        // System.out.println(a/c);
        }
        }
```

Output

```
        a+b = 30
        a-b = 10
        x+y = ThankYou
        a*b = 200
        a/b = 2
        a%b = 0
```

2.Unary Operators

Operator Name	Description
Unary Minus (-)	For decreasing the value
Unary Plus (+)	For converting the negative values into positive
Increment operator (++)	Used for increasing of the operand by 1
Post-Increment	The value is incremented first then the result is computed
Pre-Increment	The value is incremented later then the result is computed
Decrement Operator (--)	Used for decreasing the value of operand by 1
Post-Decrement	The value is decremented first then the result is computed
Pre-Decrement	The value is decremented later then the result is computed
Logical not Operator (!)	Used for inverting the values of boolean

Example: // Java program to illustrate unary operators

```
        public class operators
        {
        public static void main(String[] args)
        {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;
        // pre-increment operator
        // a = a+1 and then c = a;
        c = ++a;
        System.out.println("Value of c (++a) = " + c);
```

```
// post increment operator
// c=b then b=b+1
c = b++;
System.out.println("Value of c (b++) = " + c);
// pre-decrement operator
// d=d-1 then c=d
c = --d;
System.out.println("Value of c (--d) = " + c);
// post-decrement operator
// c=e then e=e-1
c = --e;
System.out.println("Value of c (--e) = " + c);
// Logical not operator
System.out.println("Value of !condition =" + !condition);
}
}
```

3.Assignment Operators

Assignment operators are used to assigning values to the left operand. Its types are,

Operator	Description
+=	To add the right and left operator and then assigning the result to the left operator
-=	To subtract the two operands on left and right and then assign the value to the left operand
*=	To multiply the two operands on left and right and then assign the value to the left operand
/=	To divide the two operands on left and right and then assign the value to the left operand
^=	To raise the value of left operand to the power of right operator
%=	To apply modulo operator

Example: // Java program to illustrate assignment operators

```
public class operators
{
public static void main(String[] args)
{
int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;
// simple assignment operator
c = b;
System.out.println("Value of c = " + c);
// This following statement would throw an exception as value of right operand must be
initialised before an assignment, and the program would not compile.
// c = d;
// instead of below statements,shorthand assignment operators can be used to provide
same functionality.
a = a + 1;
b = b - 1;
e = e * 2;
f = f / 2;
System.out.println("a,b,e,f = " + a + ", + b + "," + e + "," + f);
a = a - 1;
b = b + 1;
e = e / 2;
f = f * 2
// shorthand assignment operator
a += 1;
b -= 1;
e *= 2;
f /= 2;
System.out.println("a,b,e,f (using shorthand operators)= " + a + "," + b + "," + e + "," + f);
}
}
```

4.Relational Operators

Java Relational operators are used to check the equality and for comparison.

Operator Name	Description
== (equals to) ex. x==y	True if x equals y, otherwise false
!= (not equal to) ex. x!=y	True if x is not equal to y, otherwise false
< (less than) ex. x<y	True if x is less than y, otherwise false
> (greater than) ex. x > y	True if x is greater than y, otherwise false
>= (greater than or equal to) ex. x>=y	True if x is greater than or equal to y, otherwise false
<= (less than or equal to) ex. x<=y	True if x is less than or equal to y, otherwise false

Example

```
// Java program to illustrate relational operators
public class operators
{
public static void main(String[] args)
{
int a = 20, b = 10;
String x = "Thank", y = "Thank";
int ar[] = { 1, 2, 3 };
int br[] = { 1, 2, 3 };
boolean condition = true;
//various conditional operators
System.out.println("a == b :" + (a == b));
System.out.println("a < b :" + (a < b));
System.out.println("a <= b :" + (a <= b));
System.out.println("a > b :" + (a > b));
System.out.println("a >= b :" + (a >= b));
System.out.println("a != b :" + (a != b));
// Arrays cannot be compared with relational operators because objects store references not
the value
System.out.println("x == y : " + (ar == br));
System.out.println("condition==true :" + (condition == true));
}
```

5.Logical Operators

Operator Name	Description
&& (Logical AND)	Returns the value if both the conditions are true otherwise returns zero.
|| (Logical OR)	Returns the value if even one condition is true

Example: // Java program to illustrate logical operators

```
public class operators
{
public static void main(String[] args)
{
String x = "java";
String y = "class@3";
Scanner s = new Scanner(System.in);
System.out.print("Enter username:");
String uuid = s.next();
System.out.print("Enter password:");
String upwd = s.next();
// Check if user-name and password match or not.
if ((uuid.equals(x) && upwd.equals(y)) ||
(uuid.equals(y) && upwd.equals(x))) {
System.out.println("Welcome user.");
}
else
{
System.out.println("Wrong uid or password");
}
}
}
```

6. Ternary Operators in Java

- Ternary java operators are a shorthand version the 'if else' statements.
- Syntax- **condition? if true : if false**

Example: // Java program to illustrate max of three numbers using ternary operator.

```
public class operators
{
public static void main(String[] args)
{
int a = 20, b = 10, c = 30, result;
//result holds max of three numbers
result = ((a > b) ? (a > c) ? a :
c : (b > c) ? b : c);
System.out.println("Max of three numbers = "+result);
```

```
        }
        }
```

7.Bitwise Operators

Bitwise java operators are used to perform operations on single bitwise values.

Operator Name	Description
Bitwise AND operator (&)	The & operator compares corresponding bits of two operands. If both bits are 1, it gives 1 else 0.
Bitwise OR operator (|)	The | operator compares corresponding bits of two operands. If either of the bits is 1, it gives 1 else 0.
Bitwise XOR operator (^)	The ^ operator compares corresponding bits of two operands. If corresponding bits are different, it gives 1 else 0.
Bitwise Complement operator (~)	The ~ operator inverts the bit pattern. It makes every 0 to 1 and every 1 to 0.

Example: // Java program to illustrate bitwise operators

```
        public class operators
        {
        public static void main(String[] args)
        {
        int a = 0x0005;
        int b = 0x0007;
        // bitwise and Ex. 0101 & 0111=0101
        System.out.println("a&b = " + (a & b));
        // bitwise OR Ex. 0101 | 0111=0111
        System.out.println("a|b = " + (a | b));
        // bitwise xor Ex. 0101 ^ 0111=0010
        System.out.println("a^b = " + (a ^ b));
        // bitwise Complement Ex. ~0101=1010
        System.out.println("~a = " + ~a);
        // can also be combined with assignment operator to provide shorthand assignment
        // a=a&b
        a &= b;
        System.out.println("a= " + a);
        }
        }
```

8.Shift operators:

Operator Name	Description
<< (left shift operator)	Shifts the value to the left which specified by the right operand.
>> (right shift operator)	Shifts the value by zeroes defined by left operand.
>>> (unsigned right shift operator)	It fills the void on left with zeroes which are set by the right operand.

Example: // Java program to illustrate shift operators

```
        public class operators
```

```
{
public static void main(String[] args)
{
int a = 0x0005;
int b = -10;
// left shift operator
// 0000 0101<<2 =0001 0100(20)
// similar to 5*(2^2)
System.out.println("a<<2 = " + (a << 2));
// right shift operator
// 0000 0101 >> 2 =0000 0001(1)
// similar to 5/(2^2)
System.out.println("a>>2 = " + (a >> 2));
// unsigned right shift operator
System.out.println("b>>>2 = "+ (b >>> 2));
}
}
```

Expression:

An expression can be any combination of variables, literals, and operators. They also can be method calls, because methods can send back a value to the object or class that called the method.

Types of Expressions

There are three types of expressions in Java:

1. Those that produce a value, i.e. the result of (1 + 1)
2. Those that assign a variable, for example (v = 10)
3. Those that have no result but might have a "side effect" because an expression can include a wide range of elements such as method invocations or increment operators that modify the state (i.e. memory) of a program.

Examples of Expressions

1. Expressions that Produce a Value

- Expressions that produce a value use a wide range of Java arithmetic, comparison or conditional operators. For example, arithmetic operators include +, *, /, <, >, ++ and %. Some conditional operators are &&, ||, and the comparison operators are <, <= and >. See the Java specification for a complete list.
- These expressions produce a value: 3/2, 5% 3, pi + (10 * 2) and so on.

2. Statements:

- Statements are roughly equivalent to sentences in natural languages. A statement forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;):
  - Assignment expressions
  - Any use of ++ or --
  - Method calls
  - Object creation expressions
- These kinds of statements are called expression statements. Here are some examples of expression statements:
  - Value = 8933.234; //assignment statement

- Value++; //increment statement
- System.out.println(Value); //method call statement
- Integer integerObject = new Integer(4); //object creation statement

- In addition to these kinds of expression statements, there are two other kinds of statements. A declaration statement declares a variable. You've seen many examples of declaration statements.
  - double Value = 8933.234; // declaration statement

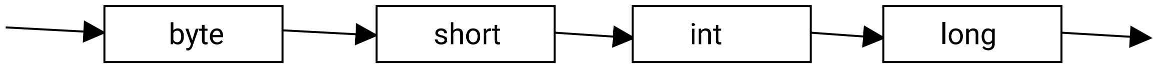
TYPE CONVERSION AND CASTING

Type conversion:

- It is the process of converting one data type into another data type.
- If the two types are compatible, then Java will perform the conversion automatically.

Java supports two kinds’ type conversions

1. Implicit type conversion (wider conversion): It process the conversion of small data type Into big data type, used to increase the size of memory of variable and it cannot loss of data it is also called as wider conversion



Wider conversion

Example:

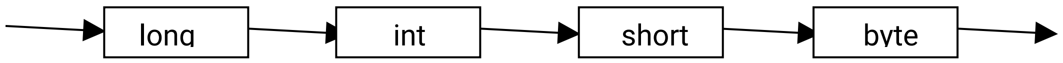
```
class Test
{
   public static void main(String[] args)
   {
      int i = 100;
      // automatic type conversion
      long l = i;
      float f = l;
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
   }
}
```

Output:

```
Int value 100
Long value 100
Float value 100.0
```

NOTE: It is always possible to assign an **int** value to a **long** variable. However, not all types are compatible, and thus, not all type conversions are implicitly allowed. For instance, there is no automatic conversion defined from **double** to **byte**.

2. Explicit type conversion (narrow conversion): It is process the conversion of big data type into small data type, used to reduce the size of memory of variable and it may loss of data it is also called as narrow conversion



Narrow conversion

Example 1: //Java program to illustrate explicit type conversion

```
class Test
{
   public static void main(String[] args)
   {
      double d = 100.04;
      //explicit type casting
      long l = (long)d;
      int i = (int)l;
      System.out.println("Double value "+d);
      //fractional part lost
      System.out.println("Long value "+l);
      System.out.println("Int value "+i);
   }
}
```

Output:

```
Double value 100.04
Long value 100
Int value 100
```

Example 2: //Java program to illustrate Conversion of int and double to byte

```
class Test {
   public static void main(String args[])
   {
      byte b;
      int i = 257;
      double d = 323.142;
      System.out.println("Conversion of int to byte.");
      b = (byte) i; //i%256
      System.out.println("i = " + i + " b = " + b);
      System.out.println("\nConversion of double to byte.");
      b = (byte) d; //d%256
      System.out.println("d = " + d + " b= " + b);
   }
}
```

Output:

```
Conversion of int to byte.
i = 257 b = 1
Conversion of double to byte.
d = 323.142 b = 67
```

Blocks

- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following listing shows two blocks from the MaxVariablesDemo (in a .java source file) program, each containing a single statement:

```
if (Character.isUpperCase(aChar)) {
   System.out.println("The character " + aChar + " is upper case.");
}
else {
   System.out.println("The character " + aChar + " is lower case.");
}
```

Control statements

Depending on the results of computations the flow of control may require to jump one part of the program to another part such jumps are called control statement the control statements are of two types:

1. decision control statements
2. loop control statements
3. jump control statements

1. Decision control statements:

Depending on the result of evalution of an expression a statements are executed.java provide the following decision control statements.

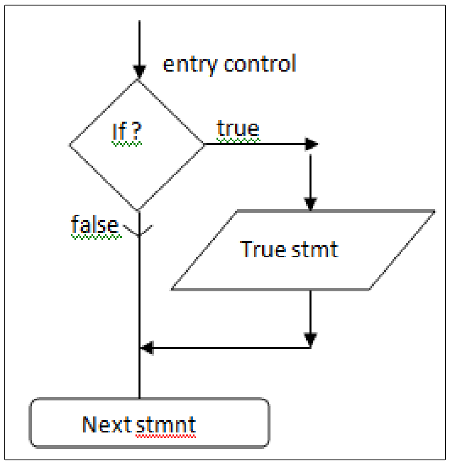
- if/statements
- if/else statements
- nested if/else statements
- if/else if ladder statements
- Switch statement

a) if statement: The if statement conditionally process , when the condition is true ,it selects only one item from one

Syntax:

```
if (condition)
  {
// body of code
   }
```

flowchart:



Example:

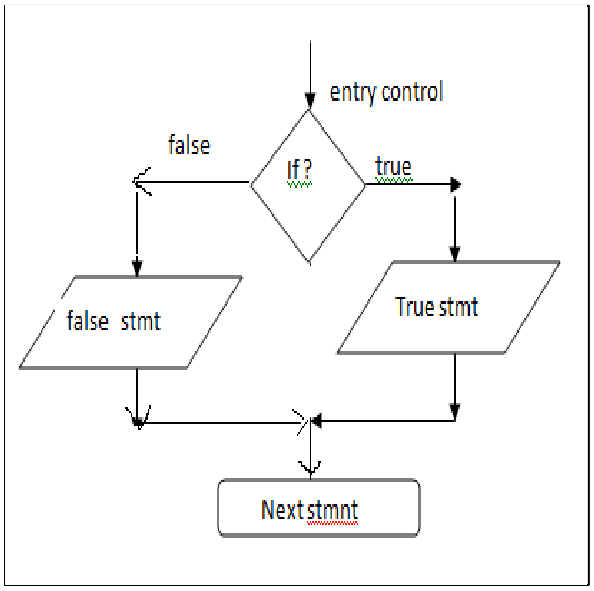
```
import java.io.*;
class  ifstatement
{
public static void main( String args[])
{
 int a=10,b=20;
 if(a>b)
{
System.out.println(“biggest is =:” +  a);
}
}
}
```

b) if/else statements: The if/else statement conditionally process, it selects only one item from two different items when the condition is true.

Syntax: if(condition)

flow chart:

```
         {
    // true statement;
          }
        else
         {
     //false statement;
          }
```



Example:

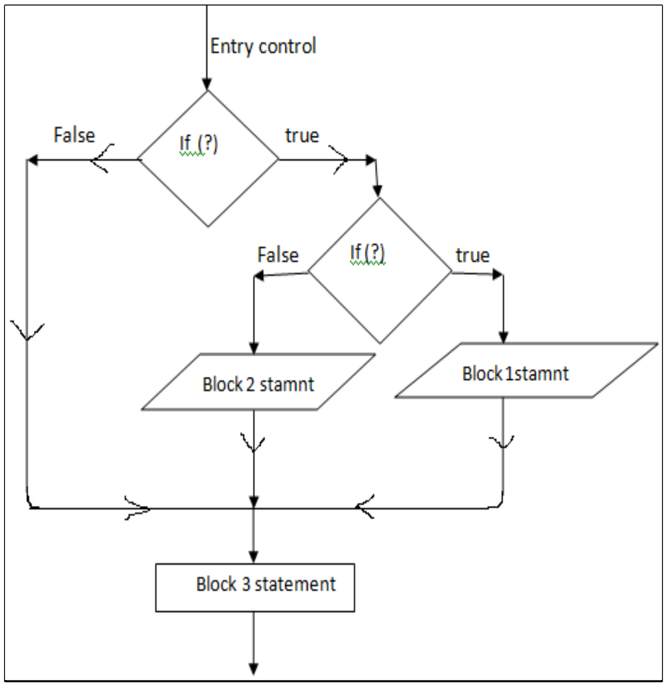
```
import java.io.*;
class  ifelse
{
public static void main( String args[])
{
 int a=10,b=20;
if(a>b){
System.out.println("biggest is =:" +  a);
}
else{
System.out.println("biggest is =:" +  b);
}
}
}
```

c) Nested if/else statements: the if/else statement is placed within another if –else statement it is also conditionally process, it selects only one item from multiple selections of items when the condition is true

Syntax:

```
if (condition)
 {
   else if(condition)
     {
   //block1  statement;
      }
   else
    {
  //block2 statements;
     }
  }
else
{
 //block 3 statemnts;
 }
```

flow chart:



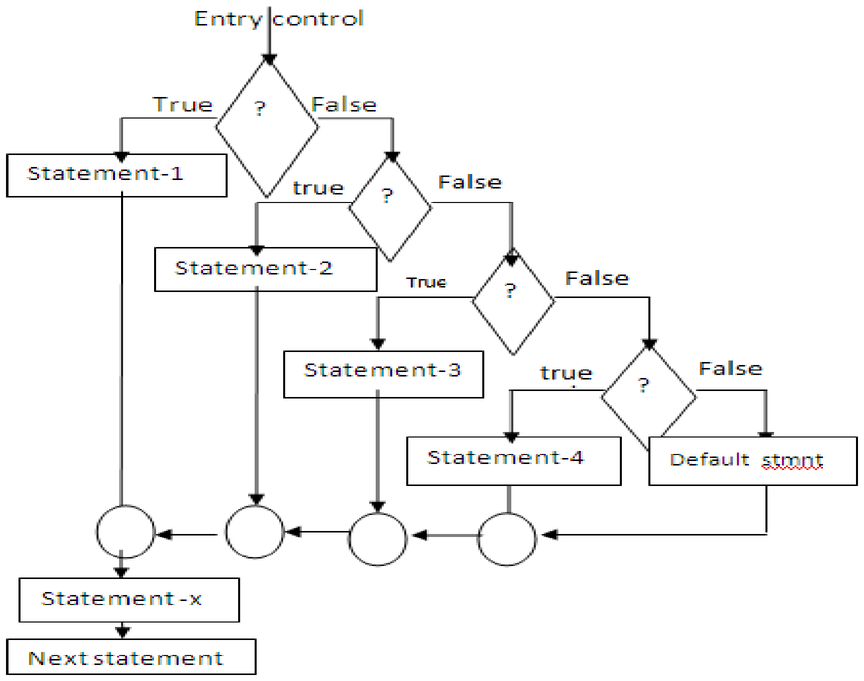
Example:

```
import java.io.*;
import java.util.Scanner;
class nestedifelse
      {
```

```
public static void main(String args[])
        {
         int sal;
         int tot=0;
         Scanner sc=new Scanner(System.in);
         System.out.println("enter the basic salary");
         sal=sc.nextInt();
        if((sal>5000 && sal<10000))
      {
          if((sal>=10000 && sal<20000))
        {
         int hra=4000,ta=200;
         tot=sal+hra+ta;
         System.out.println("total sal is=:" + tot);
         }
         else
         {
         int hra=2500,ta=100;
         tot=sal+hra+ta;
         System.out.println("total sal is=:" + tot);
         }
          } //end of outer if
     else
                  {
        int hra=5000,ta=250;
         tot=sal+hra+ta;
         System.out.println("total sal is=:" + tot);
              }
              }
             }
```

d) The else-if ladder: It consisting the multi path decisions the conditions are evaluated from top to bottom it is also called as entry control loop, when all the conditions becomes the false then the final else containing the default statement will be executed

```
if (condition-1)
   Statement-1;
 else if (condition-2)
    Starement-2;
  else if (condition-3)
    Statement-3;
   else if (condition-n)
    Statement-n;
else
  default statement;
Statement-x;
```



Example: //Java Program to demonstrate the use of If else-if ladder.
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.

```
public class IfElseIfExample {
public static void main(String[] args) {
   int marks=65;
   if(marks<50){
      System.out.println("fail");
   }
   else if(marks>=50 && marks<60){
      System.out.println("D grade");
   }
   else if(marks>=60 && marks<70){
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");
   }else if(marks>=90 && marks<100){
      System.out.println("A+ grade");
   }else{
      System.out.println("Invalid!");
   }
}
}
```

Output: C grade

Example: //Program to check POSITIVE, NEGATIVE or ZERO:

```
public class PositiveNegativeExample {
public static void main(String[] args) {
   int number=-13;
   if(number>0){
   System.out.println("POSITIVE");
   }else if(number<0){
   System.out.println("NEGATIVE");
   }else{
   System.out.println("ZERO");
  }
}
}
```

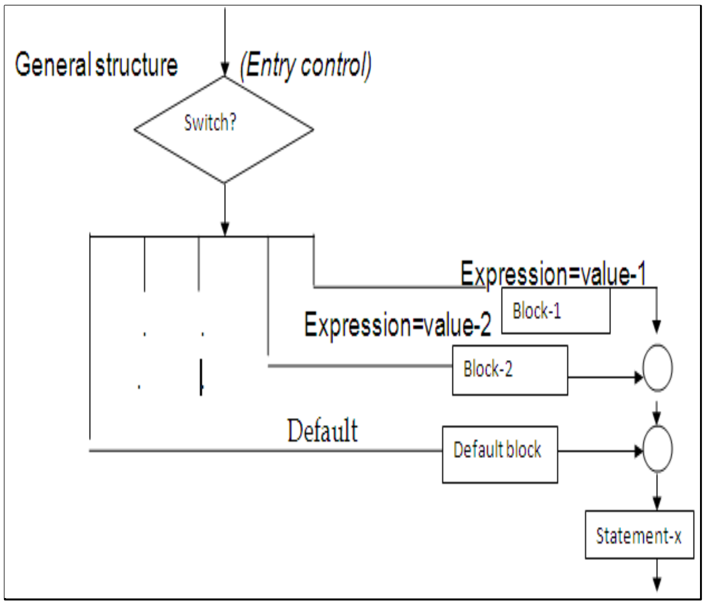
Output: NEGATIVE

d) Switch-Statement: The switch statement will useful for test the multiway decision statements known as switch, it test the value of given variable (or)Expression against a list of case values when a match is found a block of statements associated with that case is executed the general form of switch is as follows

Syntax:

```
Switch (expression)
 {
case value 1: statement-1;
       break
case value 2: statement-2;
       Break
case value 3: statement-3;
        break;
       ……………
                 ……………..
   default:default statement;
   } Statement-x;
```

flowchart:



Example:

```
import java.util.Scanner;
class Main {
  public static void main (String[] args) {
    char operator;
    Double number1, number2, result;

    Scanner scanner = new Scanner(System.in);       // create an object of Scanner class
    System.out.print("Enter operator (either +, -, * or /): ");
    operator = scanner.next(); // ask user to enter operator
    System.out.print("Enter number1 and number2 respectively: ");
    // ask user to enter numbers
    number1 = scanner.nextDouble();
    number2 = scanner.nextDouble();
    switch (operator) {
        // performs addition between numbers
        case '+':
          result = number1 + number2;
          System.out.print(number1 + "+" + number2 + " = " + result);
          break;
        // performs subtraction between numbers
        case '-':
          result = number1 - number2;
          System.out.print(number1 + "-" + number2 + " = " + result);
          break;
        // performs multiplication between numbers
        case '*':
          result = number1 * number2;
          System.out.print(number1 + "*" + number2 + " = " + result);
          break;
        // performs division between numbers
        case '/':
          result = number1 / number2;
          System.out.print(number1 + "/" + number2 + " = " + result);
          break;
        default:
          System.out.println("Invalid operator!");
          break;
      }
    }
}
```

Output:

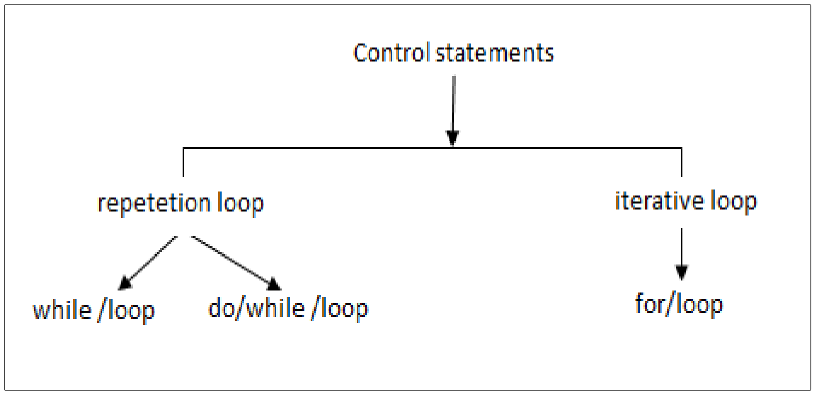
```
Enter operator (either +, -, * or /): *
Enter number1 and number2 respectively: 1.4
-5.3
1.4*-5.3 = -7.419999999999999
```

Rules for switch statement:

- The switch expression must be an integral type.
- Case labels must be constants or constants expressions as well as must unique.
- No two labels can have same value, case labels must end with semicolons.
- The break statement transfers the control out of the switch statement.
- The break statement is an optional and the default label is also an optional.
- There can be at least one default label .it is permitted to nest switch statements.

Loop statements

It allows to run block of statements repeatedly for certain number of times if the condition is true the repetition is continuing when the condition is false the repetition stop and control passed to the next statements.java provides three types of loops control statements



- while loop
- do …. while loop
- for loop

1. While loop:

- It is an entry control loop, executes the statements repeatedly based on the condition
- If the condition is true executes the block of statements reputedly, when the condition becomes false the control immediately pass to the next statements the statements are simple or compound

Syntax:
```
while(condition)
  {
 //block of ststements;
  }
```

Example 1: /* print the series of no 1 to 20*/
```
import java.io.*;
import  java.util.Scanner;
class count
{
public static void main(String args[])
{
 int i=1;
  while(i<21)
  {
    System.out.println(" i value is =:" + i);
     i=i+1;
    }
  }
 }
```

Example 2: /* WAP to print Fibonacci series between 20 */
```
import   java.io.*;
import   java.util.Scanner;
class fibonacci
  {
public static void main(String args[])
   {
 int a=0, b=1, c=0;
 System.out.println(a);
 System.out.println(b);
  while(c<21)
```

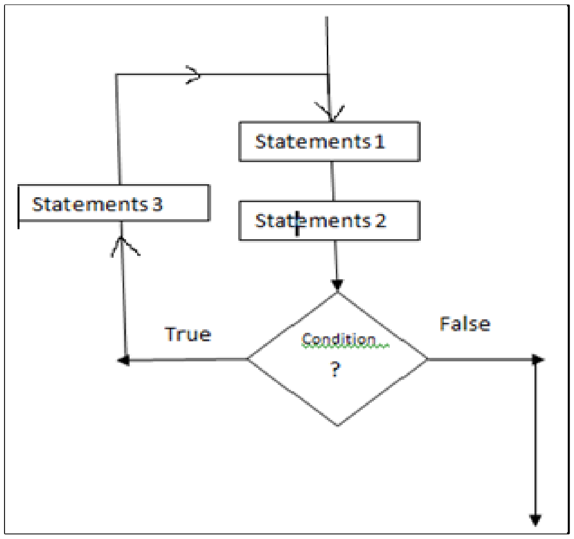
```
                        {
                          c=a+b;
                          System.out.println("\t" + c);
                          a=b;
                           b=c;
                              }
                             }
                          }
```

2. Do /while loop:

- It is an exit control loop that is executes the program first and then check the condition next this process repeatedly done until to satisfy the condition,
- Note: it executes the block of statements at least once

Syntax:
```
do
{
 -------------
 --------------
} while (condition);
```

flowchart:



Example:
```
import java.io.*;
  import java.util.Scanner;
  class dowhile
 {
  public static void main(String args[])
  {
   int i =1;
   do {
         System.out.println(" i value is =:" + i);
         i=i+1;
         } while(i<21);
     }
  }
```

3. For/loop:

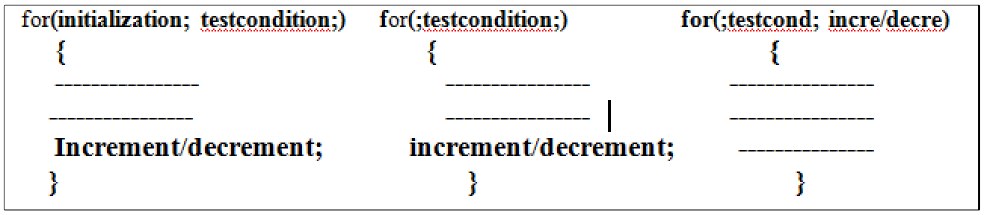
- It is also an entry control loop. It executes the section of code a fixed number of times. It is usually used when we know before entering the loop how many times, we want to execute the code.
- It is a powerful and dynamic loop for processing yields and i.e. it checks the condition first and then executes the program next, it is the counter-based loop provides more concise loop control structure. The general form of the for is as follows

Syntax:
```
for (initialization; test condition; increment/decrement)
{
  Body of the loop;
  }
```

- Initialization: this section will useful for declare the initialization values to the variable
  Ex: i=1, n=1

- Test condition: used for specifying for the conditional expressions
  Ex: num<100; or i==10;
- Increment/decrement: used for increment the variable value or decrement the variable value
  i=10
  Ex: i++=>10, ++i=11 (or) i--=10, --i=9;

It would be declaring different types



```
for(initialization; testcondition;)    for(;testcondition;)          for(;testcond; incre/decre)
   {                                       {                               {
   ----------------                        ----------------            ----------------
  ----------------                         ----------------  |         ----------------
   Increment/decrement;                increment/decrement;             ---------------
  }                                          }                               }
```

Example:

```
import java.io.*;
class forloop
{
public static void main(String args[])
{
 for(int i=0;i<10;i++)
  {
 System.out.println(" i value is =:" + i);
   i=i+1;
     }
  }
 }
```

NOTE:

- In place of condition section, it is also possible to use 2 or more relational expressions.
- Multiple initializations are also possible

Jump statements in java

Java supports the following jump statements
1) Break statement
2) Continue statement
3) Return statement
4) Exit statement

1. Break statement: It immediately terminate the loop in a program and generate the results. It used for passes the control to the inner most enclosing while ,for, do, or switch statement. The following program display the message on the screen for 10 times if use presses any key it discontinues with the help of a break statement

Ex: /* example of break */

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

```
                        for(int i=0;i<10;i++)
                         {
                            if(i==5)
                             break;
                             System.out.print("\t" + i);
                              }
                             }
                            }
```

2. Continue statement: It skip the condition statement variable value or statement when it is the loop does not terminate when a continue statement is encountered but the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.

Ex: /* example of continue */

```
                         import java.io.*;
                         class continuestatement
                        {
                        public static void main(String args[])
                        {
                        for(int i=0;i<10; i++)
                         {
                            if(i==5)
                           continue;
                         System.out.print("\t"  +  i);
                               }
                             }
                           }
```

3. return: The last control statement is return. The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method. At any time in a method the return statement can be used to cause execution to branch back to the caller of the method. Thus, the return statement immediately terminates the method in which it is executed.

4. exit():

- The java.lang.System.exit() method exits current program by terminating running Java virtual machine. This method takes a status code. A non-zero value of status code is generally used to indicate abnormal termination.
- Following is the declaration for java.lang.System.exit() method:
  - public static void exit(int status)
  - exit(0) : Generally used to indicate successful termination.
  - exit(1) or exit(-1) or any other non-zero value: Generally indicates unsuccessful termination.

Note: This method does not return any value.

The following example shows the usage of java.lang.System.exit() method.

Example: // A Java program to demonstrate working of exit()

```
                      import java.util.*;
                      import java.lang.*;
                      class useofexit
```

```
{
   public static void main(String[] args)
   {
      for (int i = 0; i < 10; i++)
      {
         if (i >= 5)
         {
            System.out.println("exit...");
              // Terminate JVM
            System.exit(0);
         }
         else
            System.out.print("\t " +i);
      }
      System.out.println("End of Program");
   }
}
```

Output:

```
1       2       3       4
exit...
```

Reading data from keyboard

There are many ways to read data from the keyboard. For example:

- InputStreamReader and BufferedReader class
- Scanner

a) InputStreamReader class and BufferedReader class

- InputStreamReader class can be used to read data from keyboard.It performs two tasks:
    - o connects to input stream of keyboard
    - o converts the byte-oriented stream into character-oriented stream
- BufferedReader class
    - o BufferedReader class can be used to read data line by line by readLine() method.

Example 1: Reading data from keyboard by InputStreamReader and BufferdReader class

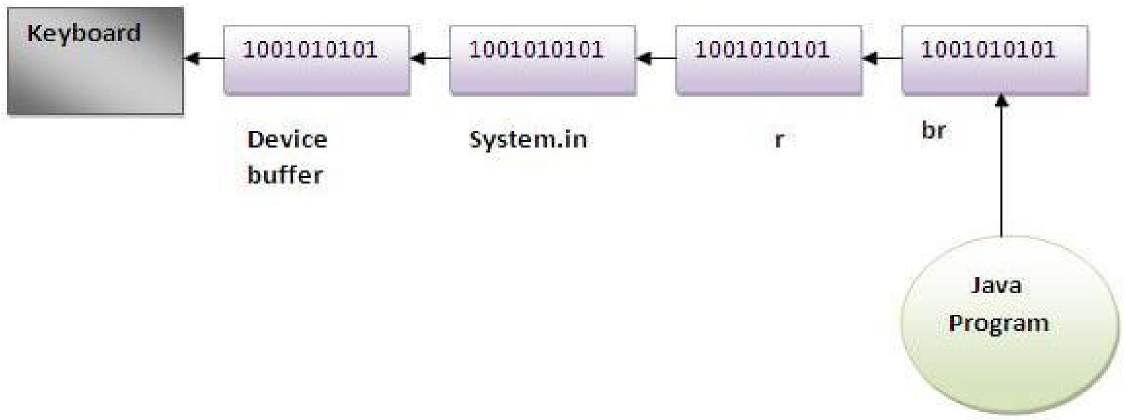
In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```
import java.io.*;
class readfromkeyboard  {
public static void main(String args[])throws Exception{
InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);
System.out.println("Enter your name");
String name=br.readLine();
System.out.println("Welcome "+name);
 }
}
```

Output:

```
Enter your name
Chinmayee
```

```
Welcome Chinmayee
```



Example 2: Reading data from keyboard by InputStreamReader and BufferdReader class until the user writes stop

```
import java.io.*;
class readfromkeyboard{
public static void main(String args[])throws Exception{
  InputStreamReader r=new InputStreamReader(System.in);
 BufferedReader br=new BufferedReader(r);
  String name="";
 while(!name.equals("stop")){
   System.out.println("Enter data: ");
   name=br.readLine();
   System.out.println("data is: "+name);
  }
 br.close();
 r.close();
 }
}
```

Output:

```
Enter data: Chinmayee
data is: Chinmayee
Enter data: 10
data is: 10
Enter data: stop
data is: stop
```

<u>b) Scanner Class in Java</u>

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- It is the easiest way to read input in a Java program.
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.

- The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.
- The Java Scanner class is widely used to parse text for strings and primitive types using a regular
- The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.
- Java Scanner Class Declaration

```
public final class Scanner
    extends Object
    implements Iterator<String>
```

How to get Java Scanner?

- To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

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

- To get the instance of Java Scanner which parses the strings, we need to pass the strings in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner("Hello Javatpoint");
```

Java Scanner Class Methods

The following are the list of Scanner methods:

boolean	nextBoolean()	It scans the next token of the input into a boolean value and returns that value.
byte	nextByte()	It scans the next token of the input as a byte.
double	nextDouble()	It scans the next token of the input as a double.
float	nextFloat()	It scans the next token of the input as a float.
int	nextInt()	It scans the next token of the input as an Int.
String	nextLine()	It is used to get the input string that was skipped of the Scanner object.
long	nextLong()	It scans the next token of the input as a long.

Example 1:

Use of Java Scanner class to read a single input (a string through in.nextLine() method) from the user.

```
import java.util.*;
public class ScannerExample {
public static void main(String args[]){
    Scanner in = new Scanner(System.in);
    System.out.print("Enter your name: ");
    String name = in.nextLine();
    System.out.println("Name is: " + name);
    in.close();
    }
}
```

Output:

```
        Enter your name: Chinmayee Rout
        Name is: Chinmayee Rout
```

Example 2:

//Program to accept multiple data from user

```
    import java.util.Scanner;
    public class ScannerClassExample{
      public static void main(String args[]){
        String s = "Hello, welcome to Java Programming";
        //Create scanner Object and pass string in it
        Scanner scan = new Scanner(s);
        //Check if the scanner has a token
        System.out.println("Boolean Result: " + scan.hasNext());
        //Print the string
        System.out.println("String: " +scan.nextLine());
        scan.close();
        System.out.println("--------Enter Your Details-------- ");
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.next();
        System.out.println("Name: " + name);
        System.out.print("Enter your age: ");
        int i = in.nextInt();
        System.out.println("Age: " + i);
        System.out.print("Enter your CGPA: ");
        double d = in.nextDouble();
        System.out.println("CGPA: " + d);
        in.close();
        }
    }
```

Output:

```
    Boolean Result: true
    String: Hello, welcome to Java Programming
    -------Enter Your Details---------
    Enter your name: ABC
    Name: ABC
    Enter your age: 20
    Age: 20
    Enter your CGPA: 9.5
    CGPA: 9.5
```

Example 3:

//program to separate tokens based on specified delimeter

```
        import java.util.*;
        public class ScannerClassExample {
          public static void main(String args[]){
            String str = "Hello/This is JavaProgramming class/I am Chinmayee.";
            //Create scanner with the specified String Object
            Scanner scanner = new Scanner(str);
            System.out.println("Boolean Result: "+scanner.hasNextBoolean());
            //Change the delimiter of this scanner
            scanner.useDelimiter("/");
            //Printing the tokenized Strings
            System.out.println("---Tokenizes String---");
           while(scanner.hasNext()){
             System.out.println(scanner.next());
           }
            //Display the new delimiter
            System.out.println("Delimiter used: " +scanner.delimiter());
            scanner.close();
            }
        }
```

Output:

```
Boolean Result: false
---Tokenizes String---
Hello
This is JavaProgramming class/
I am Chinmayee.
Delimiter used: /
```

Example 4:

```
// Java program to read some values using Scanner class and print their mean.
import java.util.Scanner;
 public class ScannerDemo
{
    public static void main(String[] args)
    {
        // Declare an object and initialize with predefined standard input object
        Scanner sc = new Scanner(System.in);
        // Initialize sum and count of input elements
        int sum = 0, count = 0;
        // Check if an int value is available
        while (sc.hasNextInt())
        {
            int num = sc.nextInt();          // Read an int value
            sum += num;
            count++;
        }
        int mean = sum / count;
        System.out.println("Mean: " + mean);
    }
}
```

Output:

```
101
223
```

```
238
892
99
500
728
Mean: 397
```

Arrays

Definition: An array is a group of related data items which are stored in homogeneous memory location that shares a common name.

Particular value indicated by writing a number called "index number" arrays are classified into different types they are

- **one dimensional array**
- **two-dimensional array**
- **Multi-dimensional array**

1. One dimensional array:

It is list of homogenous items can be given one variable name using only one subscript. Individual e value of an array is called an element array must be declared and created in the computer memory before they are used creation of an array contains two sections

- declarations of array
- creating memory locations

Syntax 1: type arrayname []; //declaration
type arrayname []=new type [size];

Type→refers datatype

Arrayname →defines the name of array.

Example: int number[];
int number[]=new int[5];

Syntax 2: type arrayname[]={ list of values };

Example: int number[]={34,45,56,67};

Example: // WAP in java to find sum of an array element

```
import java.util.Scanner;
public class Array_Sum
{
   public static void main(String[] args)
   {
      int n, sum = 0;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter no. of elements you want in array:");
      n = s.nextInt();
      int a[] = new int[n];
      System.out.println("Enter all the elements:");
      for(int i = 0; i < n; i++)
      {
         a[i] = s.nextInt();
         sum = sum + a[i];
      }
```

```
        System.out.println("Sum:"+sum);
      }
    }
```

Output:

```
Enter no. of elements you want in array:5
Enter all the elements:
1
2
3
4
5
Sum:15
```

<u>Passing Array to a Method in Java</u>

Program: //Java Program to demonstrate the way of passing an array to method.

```
class Testarray2{
//creating a method which receives an array as a parameter
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];
System.out.println(min);
}
public static void main(String args[]){
int a[]={33,3,4,5};//declaring and initializing an array
min(a);//passing array to method
}
}
```

Output:

```
3
```

<u>Anonymous Array in Java</u>

- Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

Program: //Java Program to demonstrate the way of passing an anonymous array to method.

```
public class TestAnonymousArray{
//creating a method which receives an array as a parameter
static void printArray(int arr[]){
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}
public static void main(String args[]){
printArray(new int[]{11,22,33,44});//passing anonymous array to method
}
}
```

Output:

```
11
```

```
22
33
44
```

<u>Returning Array from the Method</u>

Program: //Java Program to return an array from the method

```
class TestReturnArray{
//creating method which returns an array
static int[] get(){
return new int[]{10,30,50,70,90};
}
public static void main(String args[]){
//calling method which returns an array
int arr[]=get();
//printing the values of an array
for(int i=0;i<arr.length;i++)
System.out.print("\t"+arr[i]);
}
}
```

Output:

```
10      30      50      70      90
```

<u>2. Two dimensional array:</u>

- Two dimensional array is collection of homogenous data items that shares common name using two subscripts.it stores table of data. it the best for representation for matrix
- First subscripts refers the number of rows and second subscripts refers the number of columns.

Syntax: type arrayname[][];//array declaration.
type arrayname[][]=new int[size1][size2];//allocation of memory.

Example:

1. Addition of 2 Matrices in Java

```
//Java Program to demonstrate the addition of two matrices in Java
class arrayaddition{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];
//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
```

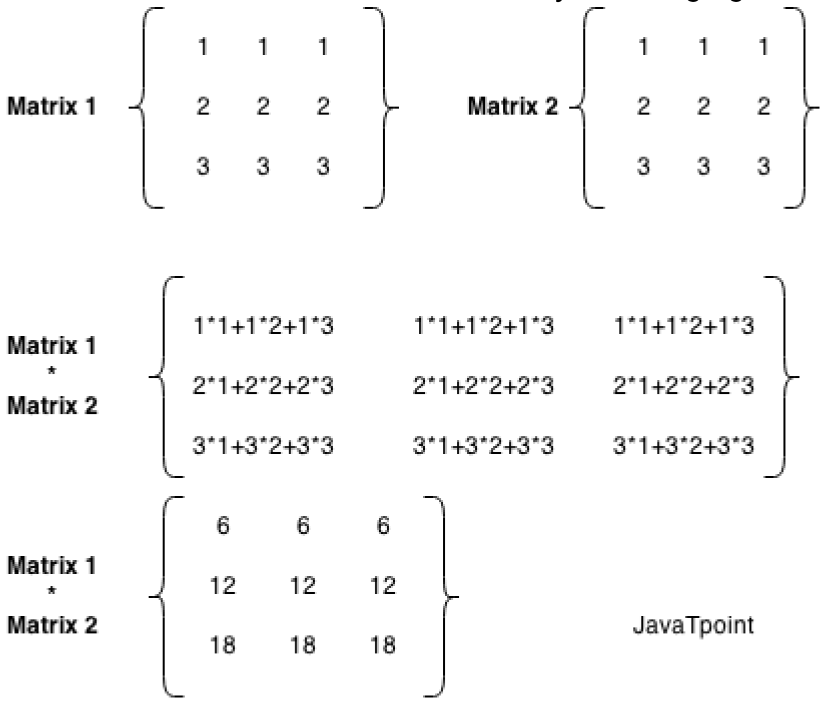
```
        }
        }
        }
```

Output:

```
        2     6     8
        6     8     10
```

2. Multiplication of 2 Matrices in Java

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.



Program: //Java Program to multiply two matrices

```
public class MatrixMultiplication{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};
//creating another matrix to store the multiplication of two matrices
int c[][]=new int[3][3];  //3 rows and 3 columns
//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]= c[i][j] + a[i][k] * b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}
}
```

Output:

```
6       6       6
12      12      12
18      18      18
```

<u>Multi –dimensional Array:</u>

- Multi-dimensional array is collection of homogenous data items that shares common name using multi subscripts

Syntax: type arrayname[][]…[];

type arrayname[][]….[]=new int[size1][size2[size3]….[size n];