MODULE-1 : BASIC CONCEPTS OF JAVA

The History and Evolution of java: History of java, the java Buzz words, The Evolution of java, Lexical issues. **Data types, variables:** Data types, Variables, The Scope and Life time of variables, Operators, Expressions, Control statements, Type conversion and casting, Command Line Arguments.

The History and Evolution of java: The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.
The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". It was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

History of java

1990: Sun Microsystems decided to develop special software to manipulate consumer electronic devices.
1991: Most popular OOPs language Oak was introduced.
1992: Green Project team by sun, demonstrated the application of their new language to control a list Of home appliances.
1993: World Wide Web appeared on the Internet and transformed the text-based internet into a graphical rich environment.
1994: A Web browser called "Hot Java" was development.
1995: Oak was renamed as Java.
1996: Sun releases Java Development Kit 1.0 (JDK 1.0)

JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed. JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created. The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM ARCHITECTURE

It contains class loader, memory area, execution engine etc.

1) Classloader: Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

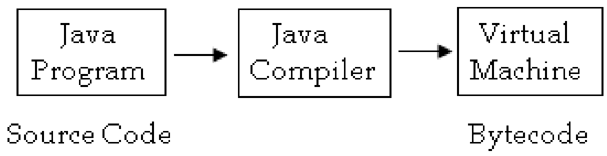
2.Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA_HOME/jre/lib/ext directory.

3. System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

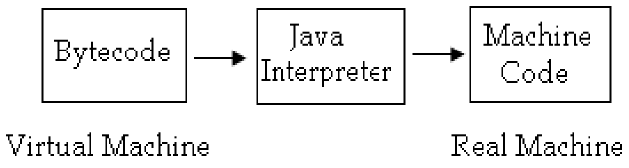
2) Class Area: Class Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.
3) Heap: It is the runtime data area in which objects are allocated.
4) Stack: Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
5) Program Counter Register: PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
6) Native Method Stack: It contains all the native methods used in the application.
7) Execution Engine: It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time (JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

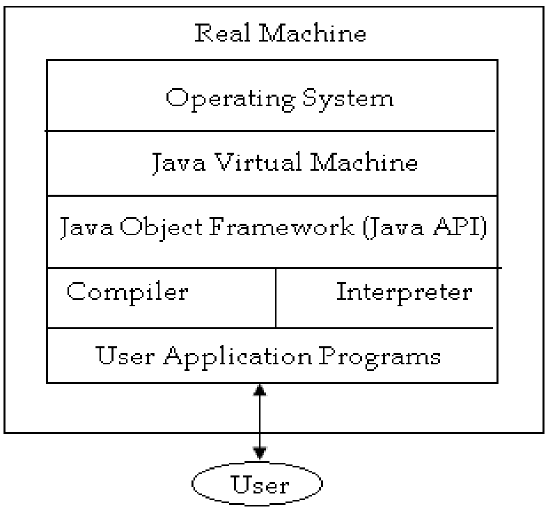
8) Java Native Interface: Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.
Usually compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. But Java compiler produces an intermediate code known as Bytecode for a machine that does not exist. This machine is called 'Java Virtual Machine' (JVM) and it exists inside the computer's memory.



The Bytecode is not machine specific. The machine specific code is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine.



The Java Object Framework (Java API) acts as the intermediary between the user programs and the virtual machine which acts as an intermediary between the operating system and the Java Object Framework.



Layers of interactions for Java programs

FEATURES OF JAVA/JAVA BUZZWORDS:

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.
A list of the most important features of the Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple: Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystems, Java language is a simple programming language because:

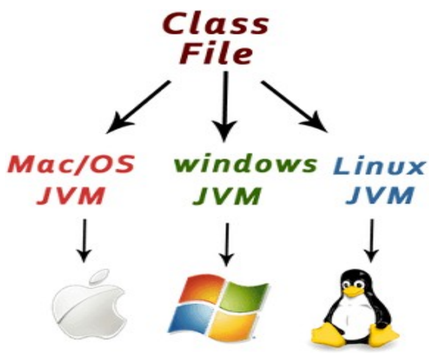
- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented: Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules. Basic concepts of OOPs are:

Object
Class
Inheritance
Polymorphism
Abstraction
Encapsulation

Platform Independent: Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides a software- based platform.
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)



Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Secured: Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**
- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust: The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral: Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable: Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded: A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic: Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

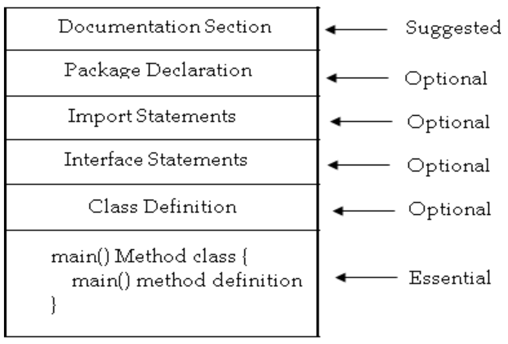
Basic Concepts of Object Oriented Programming/OOP's Principles: Object-Oriented Programming supports the following concepts.

1. Objects
2. Classes
3. Data Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Dynamic Binding
8. Message Communication

Objects:
Objects are primary runt-time entities in an Object-Oriented System. An object can be any thing that exists in the real world. Any programming problem is analyzed in terms of objects. An object is a composition of properties and actions. Properties of an object can be represented with variables and actions are represented with methods.
Ex: Chair, Car, Pen etc.,
Classes:
A Class defines the abstract characteristics of an object, including the characteristics
(Properties) and the behaviors. A Class is a collection of related objects that share common properties and actions. Once a class is defined, we can create any number of objects belonging to that class.
Ex : Person, Vehicle, Furniture, etc.,
Data Abstraction:
Abstraction is simplifying complex reality by modeling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem. The concept of hiding the unnecessary contents from the user is called "Abstraction".
Encapsulation:
The wrapping up of data and methods into a single unit called Class is known as "Encapsulation". The data is not accessible to the other methods and the methods which are wrapped in the class can only access the data items. This insulation of the data from direct access by the program is called "Data Hiding".
Inheritance:
The process of inheriting attributes and behaviors from one class to other is called "Inheritance". Here the objects of one class acquire the properties of another class. We can derive as many classes based on a class. The class which is used as base for deriving a new class is called "Super/Parent/Base Class". The newly developed class is called as "Sub/Child/Derived Class". By using inheritance, we achieve reusability of the code and develop more specialized classes.
Ex:The class Bird is called as Super class and the classes FlyingBird and Non-FlyingBird are called as Sub classes to Bird.
Polymorphism:
Poly means Several and Morphose means Forms. Polymorphism means the ability to take more than one form. If something exists in different forms in different situations it is called as "Polymorphism". Polymorphism allows the programmer to treat derived class members just like their parent class' members.
Dynamic Binding:
Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic Binding means that the code associated with a given procedure call is not known until the time of the call at runtime. Dynamic Binding is associated with the Polymorphism and Inheritance.
Message Communication:
The process by which an object sends data to another object to invoke a method is called as "Message Communication". Message Communication involves the following basic steps:

a. Creating Classes that define objects and their behavior
b. Creating objects from class definition
c. Establishing communication among objects

JAVA PROGRAM STRUCTURE



Java Program Structure

Every programming language follows certain program structure, which is followed by the programmer. A Java program may contain many classes of which only one class defines main( ) method. Classes contain data members and methods that operate on the data members of the class. Each method of a class may contain declarations and executable statements.

Documentation Section:

The documentation section comprises a set of comment lines like giving the name of the program, author and other details. This section is used to explain about the classes and algorithms. Here we use Java Documentation comment type to comment the program. It is an optional section.

Package Declaration:

The first statement in a Java program is the package declaration statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package.

Import Statements:

After package declaration statement, the immediate next line must be the import statement. An import statement instructs the interpreter to load an entire package or some portion into the program. A program can contain multiple import statements. It is an optional section.

Interface Statements:

An interface is like a class, which includes a group of abstract methods. This section is used to declare an interface. It is an optional section.

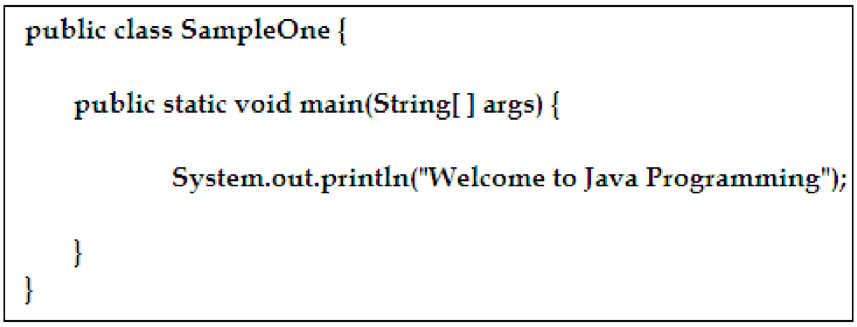
Class Definition:

A Java program can contain multiple classes. These classes are defined in this section with default access modifier. The number of classes here depends on the complexity of the program.

main ( ) method Class:

Since every stand-alone program requires a main() method, this class is the essential part of a Java program. A simple Java program may contain only this part. The main() method creates objects of various classes and establishes communication between them.

Simple Java Program: We begin learning of Java with a simple program.



```
public class SampleOne {

    public static void main(String[ ] args) {

            System.out.println("Welcome to Java Programming");

    }
}
```

File Name : SampleOne.java

public class SampleOne : A source code file can contain any number of classes, but one among them must be matched with the file name. The class which is declared as public will be the source code file name with the extension .java.

public static void main(String[ ] args) : The class which is declared as public must contain a method with the name main(), the execution of the Java program begins at main() method.

main() method does not return any values, it needs to be declared as void as its return type.

main() method must be declared as static method rather than instance method. JVM can execute main() method without creating any objects.

main() method must have access modifier as public to be available to all other code including JVM.

main() method accepts arguments called command line arguments as an array of Strings.

System.out.println() :

System is class, out is a field of System class. System.out returns an object of type PrintStream.
println() is the method of PrintStream class which is used to send some output on to the monitor.

Important Notes :

- There can be only one public class per source code file.
- If there is a public class, then file name must match with that class name.
- If there is any package declaration, then it must be the first statement.
- A source file can have multiple non-public classes
- Files with no public class can have any name.
- Classes can have modifiers before the class declaration.
  public , default, abstract, final, strictfp

LEXICAL ISSUES OF JAVA Lexical Issues During compilation, the characters in Java source code are reduced to a series of tokens. The Java compiler recognizes five kinds of tokens: identifiers, keywords, literals, operators, and miscellaneous separators. Comments and white space such as blanks, tabs, line feeds, and are not tokens, but they often are used to separate tokens.
Java programs are written using the Unicode character set or some character set that is converted to Unicode before being compiled.

Identifiers: Identifiers are programmer-designed tokens. Identifiers are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java follows the following rules for identifiers:

1. They can have alphabets, digits, underscore, dollar characters
2. They should not start with a digit
3. Uppercase and Lowercase letters are distinct
4. They should not contain any white spaces
5. They can be of any length

Naming Conventions for Identifiers:

1. Name of a method or variable starts with lowercase letter
2. When there is more than one word, each word starts with uppercase letter
3. The name of a class or interface starts with uppercase letter and each word is an uppercase letter
4. All Literals use all uppercase letters

Valid Identifiers	Invalid Identifiers
sid123	123sid
std_Name	std Name
unit$price	unit $ price
_sno	
$sno	
StudentData	
Employee	

Literals: A Literal is a fixed value that will not be changed ever during the execution of the program.
Once the programmer defines a value to a literal, it remains same through the entire program. Java supports several types of literals:

1. Integer Literals
2. Real Literals
3. Character Literals
4. String Literals
5. Backslash Character Literals

Integer Literals: An integer Literal is a sequence of numerical digits. There are three types of integer Literals:
Decimal Integers consist of a set of digits 0 to 9. Decimal integers may be either positive or negative.
Ex : 3445 56 -6743 0 - 9359
An Octal Integer Literal consists of any combination of digits from the set 0 to 7 with a leading 0. Octal values have no sign.
Ex: 034 027 0736
Hexadecimal Integer Literals consist a set of digits 0 to 9 and alphabets A to F to represent the values 10 to 15. Each Hexadecimal value begins with 0x.
Ex : 0X2 0X9F5 0XABF6
Real Literals:
The numbers containing fractional parts are called as Real values. To represent fixed floating-point values, we use real Literals. These numbers are represented with a decimal value containing a decimal point. A real Literal may be either positive or negative.
Ex: 3.1428 0.000455 -1.346
A real number may also be expressed in exponential notation.
Ex: 2.1565e2

e2 means multiply the number by 102 then the above value becomes 215.65

Character Literals: A character Literal contains single character enclosed within a pair of single quotation marks.
The character may be an alphabet, digit or any symbol. Ex: 'C'   '8'   '$'
String Literals: A string Literal is a sequence of characters enclosed between a pair of double quotation marks. A string Literal may contain alphabets, digits, any symbols, white spaces.
Ex: "Hello World"   "8932"   "apple123@gmail.com"
Backslash Character Literals:
Java supports backslash character Literals that are used to format the output presented to the user. These characters are also called as escape sequence characters.

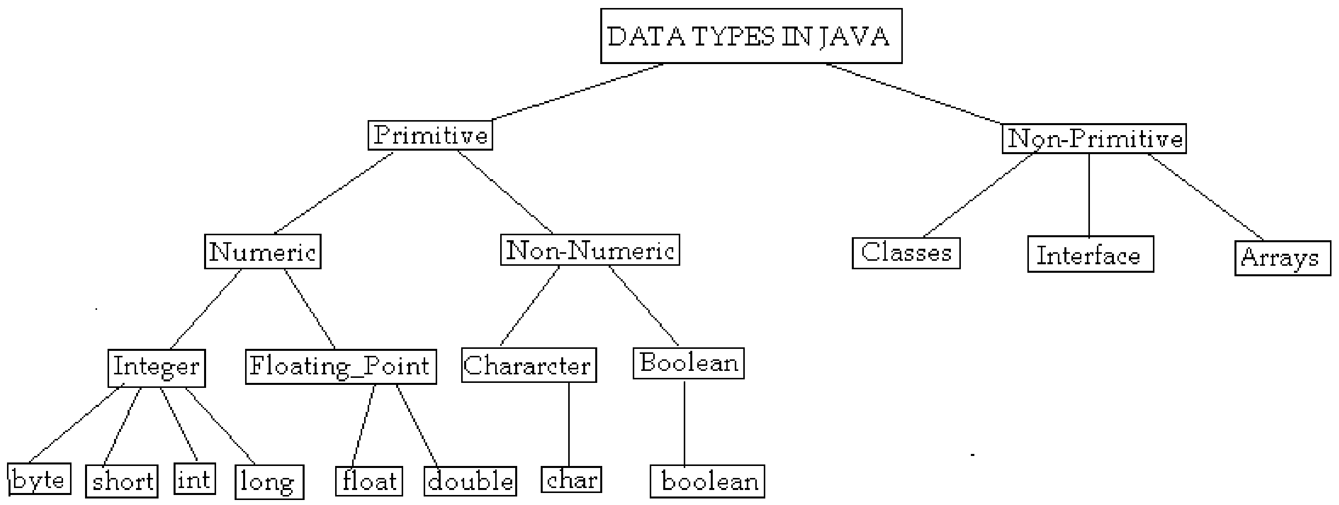
Constant	Meaning
\b	Back Space
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\'	Single Quote
\"	Double Quotes
\\	Back Slash

Keywords: The following identifiers are reserved for use as keywords. They cannot be used in any other way.

abstract	default	goto[a]	null	synchronized
boolean	do	if	package	this
break	double	implements	private	threadsafe
byte	else	import	protected	throw
byvalue[a]	extends	instanceof	public	transient
case	false	int	return	true
catch	final	interface	short	try
char	finally	long	static	void
class	float	native	super	while
const[a]	for	new	switch	
continue				

a. Reserved but currently unused.

Data Type: The type of data, the program is going to store in a variable is called the "Data Type" of a variable. The programmer can choose one data type for a variable depends on his requirement in the program. Data type specifies the size and type of values that can be stored in a variable or constant.
Java supports variety of data types to select the appropriate type for the needs of the application.



Integer Data types:

Integer types are used to declare variables to hold whole numbers. These numbers should be either positive or negative values but they cannot be unsigned. Java supports four types of integer declarations such as byte, short, int, long to support various ranges of values.

Type	Size	Minimum value	Maximum Value
byte	1 Byte	-128	127
short	2 Bytes	-32,768	32,767
int	4 Bytes	-2,147,483,648	2,147,483,647
long	8 Bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Floating Point Types:

Floating point types are used to declare variables to hold real numbers that is numbers containing fractional parts. Java supports two types of floating point declarations such as float and double.

float data type is used to represent single-precision numbers.

double data type is used to represent double-precision numbers. By default, every fractional point value is treated as double value.

Type	Size	Minimum Value	Maximum Value
float	4 Bytes	$3.4e^{-38}$	$3.4e^{+38}$
double	8 Bytes	$1.7e^{-308}$	$1.7e^{+308}$

Character Type:

Character type is used to represent single character values. The datatype char declares a variable of type char, it can hold a character. It occupies 2 Bytes of memory to represent Unicode characters apart from ASCII code characters.

Boolean Type:

Boolean type is used to represent true or false type values. The datatype boolean declares a variable of type boolean, it can hold either true or false. In Java, every conditional expression returns a value of type boolean. These type values are often used to test a condition in flow control statements.

Default Values:

Java primitive types and non-primitive types take default values when we do not assign any value to a variable or reference.

Data Type	Default Value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	white space
boolean	false
Any Class Object	null

Variables:

A Variable is an identifier that is associated with a particular memory location in the computer's memory. A variable is capable of storing data values of type to which it is declared. The value of a variable will be changed frequently during its life in the program.

The declaration of a variable tells the compiler what the variable name is, the datatype of the variable and scope of the variable. Each variable must be declared prior(before) to use it.

Syntax to declare a variable:

<Data Type> <Name of the Variable>;

Ex:

int age; float height; char gender;

Initializing a variable:

Initializing a variable is nothing but assigning a value at the time of variable's declaration. To assign a value to a variable, we use assignment statement (=) along with the variable declaration.

Syntax :

<Data Type> <Name of the variable> = <Initial Value>;

Ex:

int age=19;

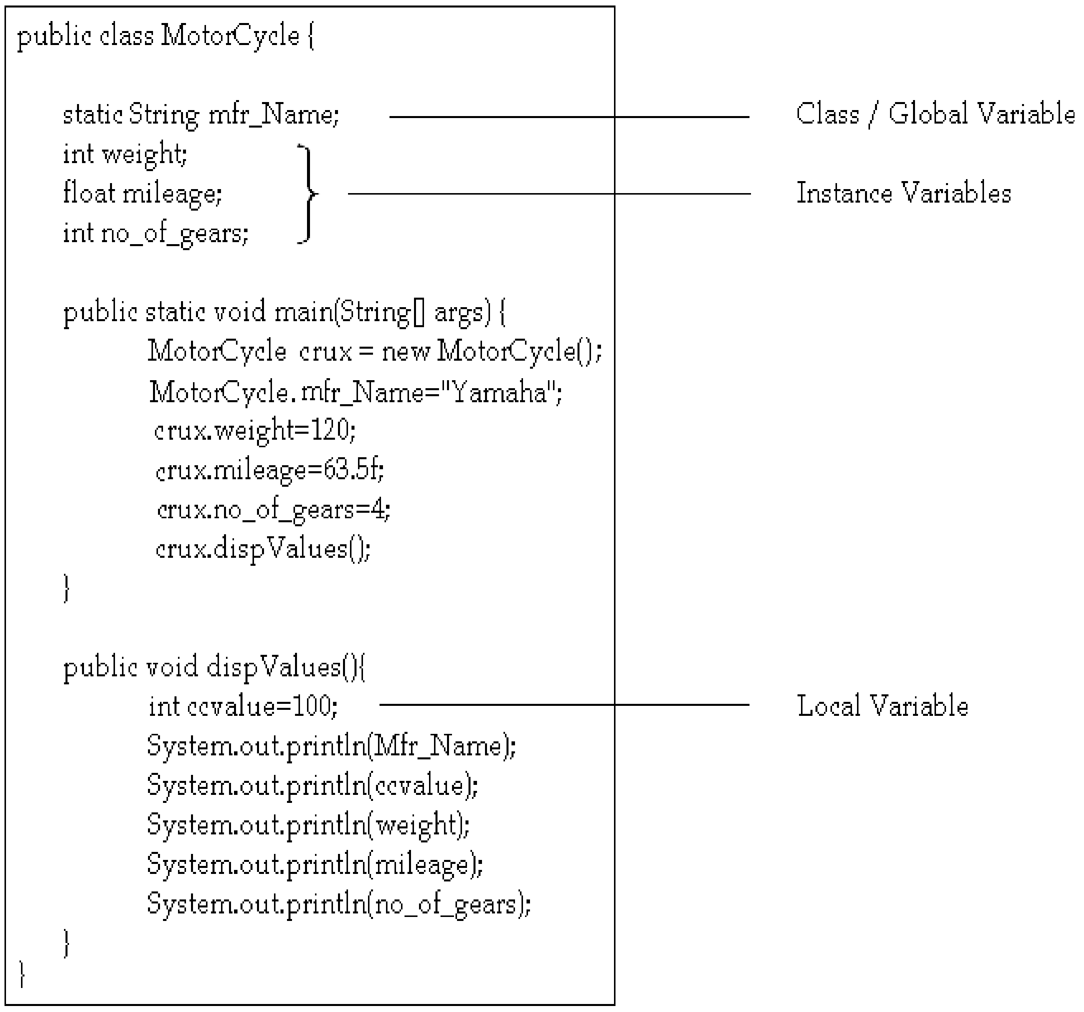
float height=160.75f; char gender='M';

Scope of a Variable:
The area of the program where the variable is accessible is called "Scope" of the variable. The scope of a variable defines the life span of that variable. A variable may have any one of the scopes like block scope, method scope (local scope), class scope (global scope) etc.
Java variables are classified into the following types:

1. Instance Variables
2. Class Variables
3. Local Variables

Instance Variables:
Instance variables are the variables that are declared inside the class. These variables are created when the objects are instantiated and associated with the objects. They take different values for each object.
Class Variables:
Class variables are the variables that are declared inside the class as static variables. These variables are global to entire class thus they are common to entire set of objects created for that class.
Local Variables:
Local variables are the variables that are declared and used inside the methods of a class. These variables are not accessible outside the method in which they are declared. Local variables can also be declared inside blocks which are defined between a pair of opening and closing braces. The variables which are declared inside a block will be accessible only inside that block.

Example :



```
public class MotorCycle {

    static String mfr_Name;
    int weight;
    float mileage;
    int no_of_gears;

    public static void main(String[] args) {
          MotorCycle  crux = new MotorCycle();
          MotorCycle. mfr_Name="Yamaha";
           crux.weight=120;
           crux.mileage=63.5f;
           crux.no_of_gears=4;
           crux.dispValues();
    }

    public void dispValues(){
          int ccvalue=100;
          System.out.println(Mfr_Name);
          System.out.println(ccvalue);
          System.out.println(weight);
          System.out.println(mileage);
          System.out.println(no_of_gears);
    }
}
```

Constants:
Constants in java refer to fixed values that do not change during the execution of a program; java supports several types of constants.

Integer constants: An integer constant refers to a sequence of digits. There are 3 types of integer constants namely, decimal integers, octal integers, and hexadecimal integers.

•Decimal integers consist of a set of digits 0 through 9 preceded by an optional minus sign.

E.g.: 123, -321, 0, and 654321.

•An octal integer constant consists of any combination of digits from set 0 through 7 with a leading 0.

E.g.: 037, 0, 0435, and 0551.

•A sequence of digits preceded by 0x or 0X is considered as hexadecimal integer. They may also include alphabets A through F. A letter A through F represents the numbers 10 through 15.

E.g.: 0X2, 0X9F, 0XBCD.

Real constants: Integer numbers are inadequate to represent that varies continuously, such as distances, heights, widths, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts like 17.587. Such numbers are called real or floating point constants.
E.g.: 0.0086, -0.75, and 435.6.
These numbers are shown in decimal notation, having a whole number followed by a decimal point and fractional part, which is an integer. It is possible that the number may not have digits before the decimal point or digits after the decimal point.
E.g.: 215., .95, -.71 are valid real numbers.

A real number may also be expressed in exponential notation. For e.g. the value 215.65 may be written as 2.1565 e2 in exponential notation. e2 means multiply by 102. The general form is mantissa e exponent.
The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer with an optional + or – sign. The letter e separating the mantissa and the exponent can be written in either upper case or lower case.

E.g.: 0.65e4, 12e-2, and 3.18E3.

White space is not allowed in any numeric constants.
Single character constants: A character constant contains a single character enclosed with in a pair of single quote marks. E.g.: ‘5’, ‘X’, ‘;’, and ‘ ’.
Note that the character constant ‘5’ is not the same as the number 5.
Back slash character constants: Java supports some special back slash character constants that are used in output methods. For e.g., the symbol ‘\n’ stands for new line character. A list of back slash character constants is given below.

Constant	**meaning**
‘\b’	back space
‘\f’	form feed
‘\n’	new line
‘\r’	carriage return
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	back slash

String constants: A string constant is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces. For example
“Hello java”, “1997”, “WELL DONE”, “?...!”, “5+3”

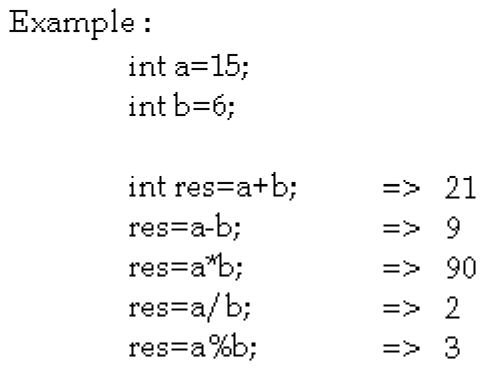
OPERATORS: An operator is a symbol which is used to perform arithmetic and logical operations. Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators

5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator

1. Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

* : Multiplication
/ : Division
% : Modulo
+ : Addition
– : Subtraction



Example :

```
    int a=15;
    int b=6;

    int res=a+b;        =>  21
    res=a-b;            =>  9
    res=a*b;            =>  90
    res=a/b;            =>  2
    res=a%b;            =>  3
```

2. Unary Operators: Unary operators need only one operand. They are used to increment, decrement or negate a value.

– :Unary minus, used for negating the values.

+ :Unary plus, indicates positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is byte, char, or short. This is called unary numeric promotion.

++ :Increment operator, used for incrementing the value by 1. There are two varieties of increment operator.

Post-Increment: Value is first used for computing the result and then incremented.
Pre-Increment: Value is incremented first and then result is computed.

-- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operator.

Post-decrement : Value is first used for computing the result and then decremented.
Pre-Decrement : Value is decremented first and then result is computed.

! : Logical not operator, used for inverting a boolean value.

3. Assignment Operator : '=' Assignment operator is used to assign a value to any variable. It has a right to left associativity, i.e value given on right hand side of operator is assigned to the variable on the left and therefore right hand side value must be declared before using it or should be a constant.

General format of assignment operator is,

variable = value;

In many cases assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement. For example, instead of a = a+5, we can write a += 5.

+=, for adding left operand with right operand and then assigning it to variable on the left.
-=, for subtracting left operand with right operand and then assigning it to variable on the left.
*=, for multiplying left operand with right operand and then assigning it to variable on the left.
/=, for dividing left operand with right operand and then assigning it to variable on the left.
%=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

4. Relational Operators: These operators are used to check for relations like equality, greater than, less than. They return Boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements. General format is variable relation_operator value

Operator	Meaning
<	Less Than
<=	Less Than or Equal to
>	Greater Than
>=	Greater Than or Equal to
==	Equal to
!=	Not Equal to

5. Logical Operators are used to form compound conditions by combining two or more relations. The expression which is formed as combination of two or more relational expressions is called Logical Expression or Compound Relational Expression. In Java all logical operators return boolean values that is either true or false as result.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Truth Tables:
Truth Tables represent the result of a compound statement by combining two or more relational expressions.
Logical AND (&&) : This operator returns true, if both the operands are evaluated to true, otherwise it returns false.

Expr1	Expr2	Expr1&&Expr2
T	T	True
T	F	False
F	T	False
F	F	False

Logical OR (||) :This operator returns false, if both the operands are evaluated to false, otherwise it returns true.

Expr1	Expr2	Expr1||Expr2
T	T	True
T	F	True
F	T	True
F	F	False

Logical NOT (!) :This operator is used to negate a value of type Boolean. It converts a true value into false and false value into true.

Expr	! Expr
T	False
F	True

6. Conditional Operator/Ternary operator: Conditional operator is one of the special operators supported by Java which can be used as an alternative to if..else ladder. Since it acts upon three operands, it is called as Ternary operator. This operator is used to construct conditional expressions of the form:

```
exp1 ? exp2 : exp3
```

In the above form:

exp1 is evaluated first, if it is true then the exp2 is evaluated and becomes the target, if exp1 is false then the exp3 is evaluated and becomes the target.

Ex: // 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 = ((a > b)&& (a > c))? a:(b > c)? b: c); // result holds max of three numbers
System.out.println("Max of three numbers = " + result);
}
}
```

Output:

```
Max of three numbers = 30
```

7. Bitwise Operators: These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.

&, Bitwise AND operator: returns bit by bit AND of input values.

|, Bitwise OR operator: returns bit by bit OR of input values.

^, Bitwise XOR operator: returns bit by bit XOR of input values.

~, Bitwise Complement Operator: This is a unary operator which returns the one’s complement representation of the input value, i.e. with all bits inversed.

8. Shift Operators: These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two. General format-

number shift_op number_of_places_to_shift;

<<, Left shift operator: shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

>>, Signed Right shift operator: shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.

>>>, Unsigned Right shift operator: shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

9. Instance of operator : Instance of operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass or an interface. General format-

object instance of class/subclass/interface

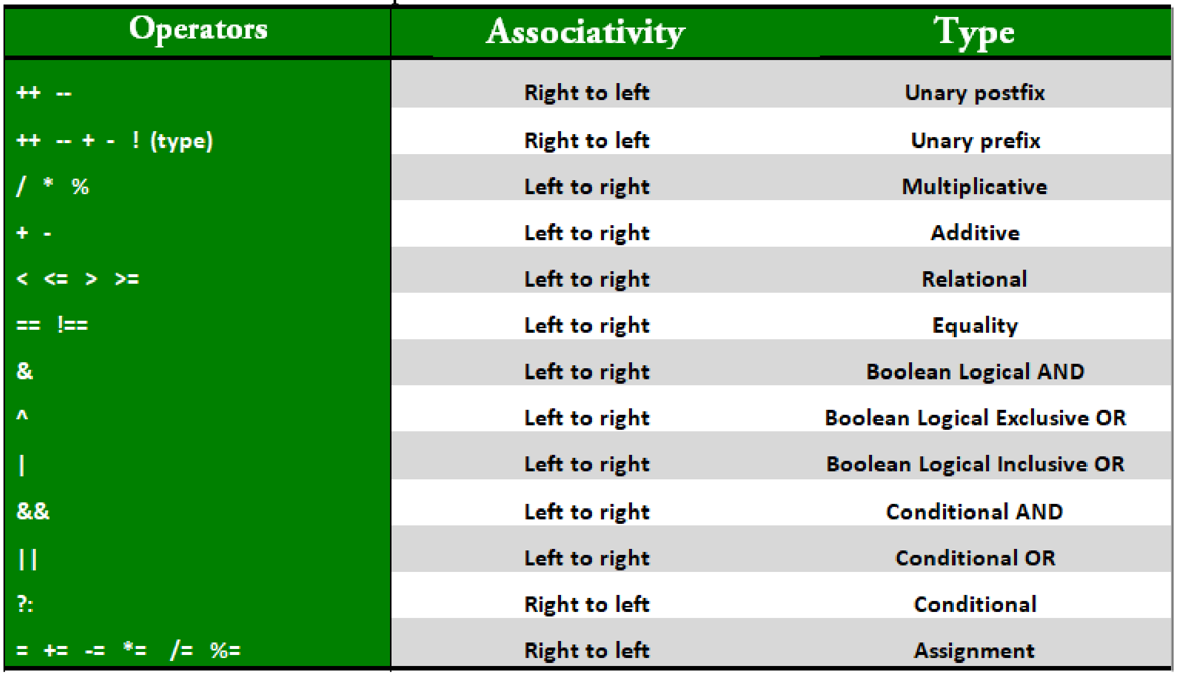
```
// Java program to illustrate instance of operator
class operators
{
public static void main(String[] args)
{
Person obj1 = new Person();
 Person obj2 = new Boy();

// As obj is of type person, it is not an instance of Boy or interface

 System.out.println("obj1 instanceof Person: " + (obj1 instanceof Person));
 System.out.println("obj1 instanceof Boy: " + (obj1 instanceof Boy));
System.out.println("obj1 instanceof MyInterface: " + (obj1 instanceof MyInterface));

// Since obj2 is of type boy, whose parent class is person
// and it implements the interface Myinterface
// it is instance of all of these classes
System.out.println("obj2 instanceof Person: " + (obj2 instanceof Person));
System.out.println("obj2 instanceof Boy: " + (obj2 instanceof Boy));
 System.out.println("obj2 instanceof MyInterface: "+ (obj2 instanceof MyInterface));
}
}

class Person { }
class Boy extends Person implements MyInterface {}
interface MyInterface { }
```

Output:

```
obj1 instanceof Person: true obj1 instanceof Boy: false
obj1 instanceof MyInterface: false obj2 instanceof Person: true
obj2 instanceof Boy: true
obj2 instanceof MyInterface: true
```

Precedence and Associativity of Operators

Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of the equation to consider first as there can be many different valuations for the same equation. The below table depicts the precedence of operators in decreasing order as magnitude with the top representing the highest precedence and bottom shows the lowest precedence.



Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !==	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
|	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
||	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

Precedence and Associativity: There is often a confusion when it comes to hybrid equations that is equations having multiple operators. The problem is which part to solve first. There is a golden rule to follow in these situations. If the operators have different precedence, solve the higher precedence first. If they have same precedence, solve according to associativity, that is either from right to left or from left to right.

```
public class operators {
public static void main(String[] args)
{
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
// precedence rules for arithmetic operators (* = / = %) > (+ = -) prints a+(b/d)
System.out.println("a+b/d = " + (a + b / d));
// if same precedence then associative rules are followed e/f -> b*d -> a+(b*d) -> a+(b*d)-(e/f)
System.out.println("a+b*d-e/f = "+ (a + b * d - e / f));
}
}
```

Output:

```
a+b/d = 20
a+b*d-e/f = 219
```

Be a Compiler: Compiler in our systems uses lex tool to match the greatest match when generating tokens. This creates a bit of a problem if overlooked. For example, consider the statement a=b+++c;, to many of the readers this might seem to create compiler error. But this statement is absolutely correct as the token created by lex are a, =, b, ++, +, c. Therefore this statement has a similar effect of first assigning b+c to a and then incrementing b. Similarly, a=b+++++c; would generate error as tokens generated are a, =, b, ++, ++, +, c. which is actually an error as there is no operand after second unary operand.

```
public class operators {
public static void main(String[] args)
{
int a = 20, b = 10, c = 0;
// a=b+++c is compiled as
// b++ +c
// a=b+c then b=b+1 a = b++ + c;
System.out.println("Value of a(b+c), " + " b(b+1), c = "+ a + ", " + b + ", " + c);
// a=b+++++c is compiled as  b++ ++ +c
// which gives error.
// a=b+++++c;
// System.out.println(b+++++c);
}
}
```

Output:

```
Value of a(b+c), b(b+1), c = 10, 11, 0
```

Using + over (): When using + operator inside system.out.println() make sure to do addition using parenthesis. If we write something before doing addition, then string addition takes place, that is associativity of addition is left to right and hence integers are added to a string first producing a string, and string objects concatenate when using +, therefore it can create unwanted results.

```
public class operators {
public static void main(String[] args)
{
int x = 5, y = 8;
// concatenates x and y as first x is added to "concatenation (x+y) = "
// producing "concatenation (x+y) = 5" and then 8 is further concatenated.
System.out.println("Concatenation (x+y)= " + x + y);
// addition of x and y System.out.println("Addition (x+y) = " + (x + y));
}
}
```

Output:

```
Concatenation (x+y)= 58 Addition (x+y) = 13
```

Expressions:

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language. Java can handle any complex mathematical expressions.
e.g.: a*b+c, x/y+c

Evaluation of expressions: Expressions are evaluated using an assignment statement of the form: Variable = expression;
e.g.: x=a*b-c; y=b/c*a;

Java Type Casting

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

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

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

Automatic conversion(Widening):

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

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

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

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

```
 Eg: byte a=10;
      int a=b; // here no  loss of value
```

byte -> short -> char -> int -> long -> float -> double

<u>**Narrowing :**</u>

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

double -> float -> long -> int -> char -> short -> byte

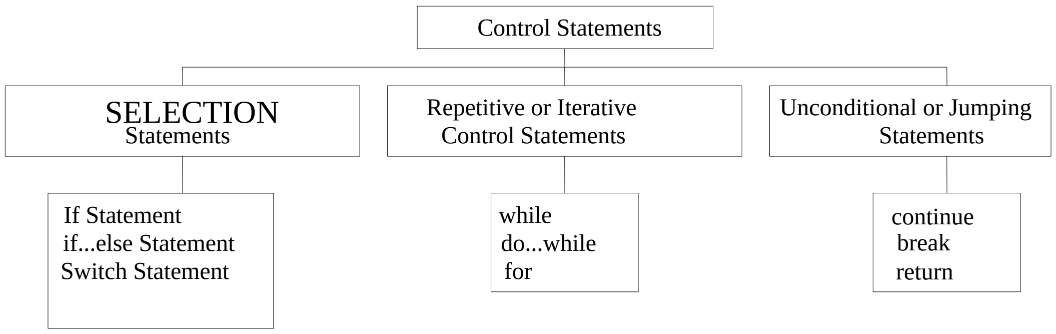
```
   Ex:-   int m=129;
          byte b =(byte) m;   // here loss of value
```

Example:

```
public class Main
{
public static void main(String[] args)
{
int myInt = 9;
double myDouble = myInt; // Automatic casting: int to double
System.out.println(myInt);      // Outputs 9
System.out.println(myDouble); // Outputs 9.0
}
}
```

CONTROL STATEMENTS:

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

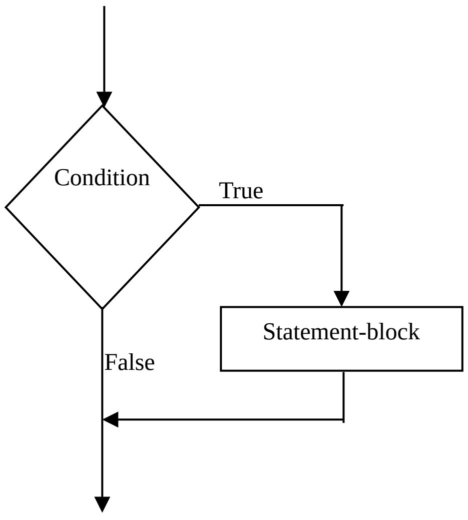


Java's Selection Statements: Java supports two selection statements: if and switch. These statements allow us to control the flow of program execution based on condition.

<u>Decision Making with if statement :</u>The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

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

Simple if statement :
The 'if' statement is a conditional control statement that executes the specified set of instructions based on the condition.
The general form of simple **if** statement is



```
if ( Condition )
{
        Statement – block;
}
```

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

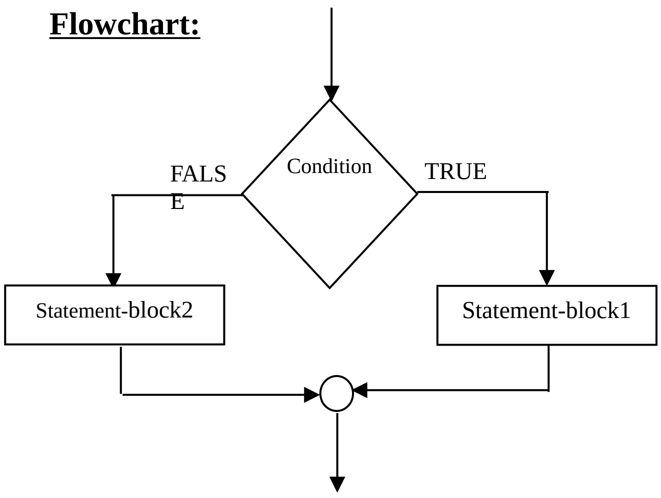
The **statement - block** may be a single statement or a group of statements. If there is only a single statement in the statement block, there is necessity of curly braces ({ }).

If…else statement :

It is also conditional control statement that executes the statements based on the condition. The if..else statement is an extension of simple if statement. The general form is

```
if ( Condition )
      Statement –
block1;
else
      Statement –
block2;
```

<u>Flowchart:</u>



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

Ex :
```
if (n%2==0)
   System.out.println("Even Number");
else
   System.out.println("Odd Number");
```

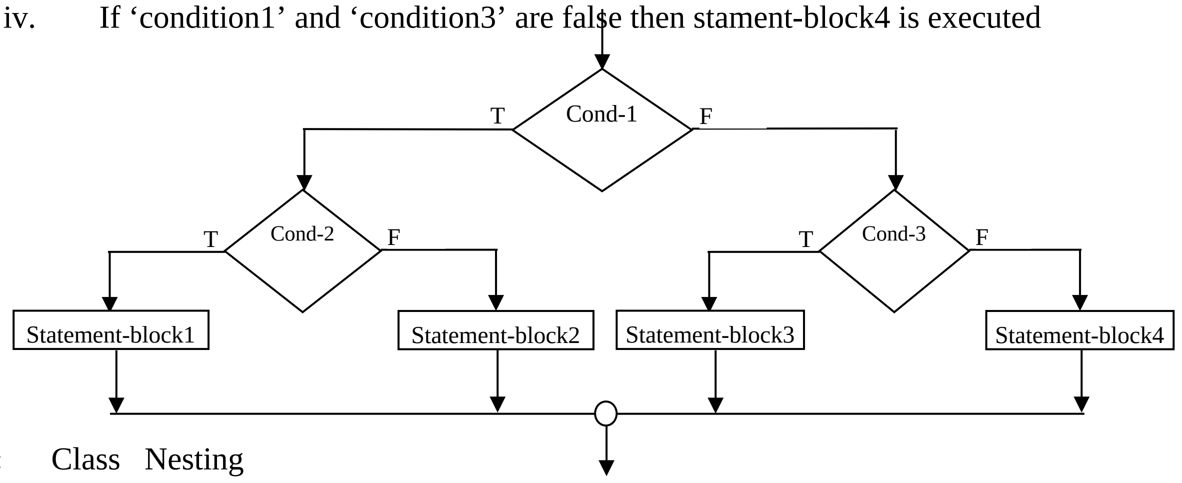
Nested if…else statement :

When the 'if..else' statement is used within another 'if…else' statement then it is called "nested if" statement. The general format is as follows:

Syntax:
```
if (condition1)
{
  if (condition2)
      statement-block1 ;
  else
      statement-block2 ;
 }
else
{
  if (condition3)
      statement-block3 ;
  else
      statement-block4 ;
 }
```

In the above syntax,

i. If 'condition1' and 'condition2' are true then stament-block1 is executed
ii. If 'condition1' is true and 'condition2' is false then stament-block2 is executed
iii. If 'condition1' is false and 'condition3' is true then stament-block3 is executed
iv. If 'condition1' and 'condition3' are false then stament-block4 is executed



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

```
            {
                 if (b>c)
                 System.out.println(b);
                 else
                 System.out.println(c);
               }
          }
          }
```

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

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

Output:

```
D:/DRJSB>javac BiggestNo.java
D:/ DRJSB >java BiggestNo
b is big
```

Switch Statement:

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

Syntax:

```
 switch (expression)
    {
     case value1:  //statement sequence
          break;
      case value2:  //statement sequence
          break;
       ……………..…..
      case valueN:  //statement sequence
          break;
      default:  //default statement sequence
    }
```

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

Program : Write a program for using the switch statement to execute a particular task depending on color value.

```
//To display a color name depending on color value
class ColorDemo
{
 public static void main(String args[])
  {
    char color = ‘r’;
    switch (color)
    {
      case ‘r’: System.out.println (“red”);    break;
      case ‘g’: System.out.println (“green”);  break;
      case ‘b’: System.out.println (“blue”);    break;
      case ‘y’: System.out.println (“yellow”);  break;
      case ‘w’: System.out.println (“white”);  break;
      default: System.out.println (“No Color Selected”);
    }
  }
}
```

Output:

```
D:/ DRJSB >javac ColorDemo.java
D:/ DRJSB >java ColorDemo
red
```

Java’s Iteration Statements: Java’s iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops. **A loop** repeatedly executes the same set of instructions until a termination condition is met.

while Loop: while loop repeats a group of statements as long as condition is true. Once the condition is false, the loop is terminated. In while loop, the condition is tested first; if it is true, then only the statements are executed. while loop is called as entry control loop.

Syntax: while (condition)

```
    {
      statements;
    }
```

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

```
//Program to generate numbers from 1 to 20.
class Natural
{
 public static void main(String args[])
  int i=1;
    while (i <= 20)
    {
     System.out.print (i + “\t”);
      i++;
    }
  }
}
```

Output:

```
D:/ DRJSB >javac Natural.java
D:/ DRJSB >java Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20
```

do-while Loop:

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

Syntax: do

```
    {
      statements;
    } while (condition);
```

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

```
//Program to generate numbers from 1 to 20.
class Natural1
{
 public static void main(String args[])
  {  int i=1;
    do
    {  System.out.print (i + “\t”);
```

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

Output:

```
D:/ DRJSB javac Natural1.java
D:/ DRJSB java Natural1
1        2        3        4        5        6        7        8        9        10
11       12       13       14       15       16       17       18       19       20
```

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

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

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

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

```
//Program to generate numbers from 1 to 20.
class Natural2
{
public static void main(String args[])
  {
 int i;
    for (i=1; i<=20; i++)
      System.out.print (i + “\t”);
  }
}
```

Output:

```
D:/ DRJSB >javac Natural2.java
D:/ DRJSB >java Natural2
1        2        3        4        5        6        7        8        9        10
11       12       13       14       15       16       17       18       19       20
D:/ DRJSB >
```

Java’s Jump Statements: Java supports three jump statements: break, continue and return. These statements transfer control to another part of the program.

break:

- break can be used inside a loop to come out of it.
- break can be used inside the switch block to come out of the switch block.
- break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.

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

Program : Write a program to use break as a civilized form of goto.

```
//using break as a civilized form of goto
    class BreakExample
   {
      public static void main(String[] args) {
          //using for loop
          for(int i=1;i<=10;i++){
              if(i==5){
                  //breaking the loop
                  break;
              }
```

```
            System.out.println(i);
        }
    }
    }
```

OUTPUT :

```
1
2
3
4
```

```
class BreakDemo
{
public static void main (String args[])
  {
 boolean t = true;
    first: {
  second: {
        third:{
            System.out.println (“Before the break”);
            if (t) break second; // break out of second block
              System.out.println (“This won’t execute”);
          }
          System.out.println (“This won’t execute”);
        }
        System.out.println (“This is after second block”);
      }
    }
  }
```

Output:

```
D:/ DRJSB >javac BreakDemo.java
D:/ DRJSB >java BreakDemo
Before the break
This is after second block
D:/ DRJSB >
```

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

Syntax: continue;

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

```
//Program to generate numbers from 1 to 20.
class Natural
{
 public static void main (String args[])
  {
 int i=1;
    while (true)
    {
 System.out.print (i + “\t”);
      i++;
      if (i <= 20 )
        continue;
      else
        break;
    }
  }
}
```

Output:

```
D:/ DRJSB >javac Natural.java
D:/ DRJSB >java Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20
D:/ DRJSB >
```

return statement:

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

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

Program : Write a program to demonstrate return statement.

```
//Demonstrate return
class ReturnDemo
{  public static void main(String args[])
  {  boolean t = true;
    System.out.println ("Before the return");
    if (t)
      return;
    System.out.println ("This won't execute");
  }
}
```

Output:

```
D:/DRJSB>javac RetернDemo.java
D:/ DRJSB >java ReturnDemo
Before the return
```

Command-line argument:

Command line arguments are the parameters that are supplied to the application programs at the time of invoking it for execution. Let us consider the signature of the main( ) method:

public static void main(String args[ ]).

In the above signature of main( ) method,
'args' is declared as an array or strings [known as string object].
Any arguments provided in the command line are passed to the array 'arg' as its elements. We can simply access the array element and use them in them in the program.

For Eg:

D:\chakry> Java Test Basic Fortan c++   java

These comment lines contain 4 arguments. These are assigned to the array 'arg' as follows.
arg[0]="Basic"
arg[1]="Fortan"
arg[2]="c++"
arg[3]="Java"

"Basic"	"Fortan"	"c++ "	"java"
0	1	2	3

The individual elements of the array are accessed by using an index or subscript like arg[i]. The value of 'i' denotes the position of the elements inside the array. In java the array index start from '0' not '1'.

```
/*  A program to display the list of cities */
class  CommandLine
{
```

```
public  static void main(String args[ ])
{
System.out.println(“The cities names are: ”);

for(int i=0;i<args.length;i++)
{
Sy
System.out.println(args[i]);
}
}
}
```

Save: CommandLine.java

Compile : javac CommandLine.java

Interpretation: java CommandLine   Banglore   Goa   Chennai   Hyderabad

EX: The following program displays all of the command-line arguments that it is called with - public class CommandLine

```
{
public static void main(String args[])
{
for(int i = 0; i<args.length; i++)
{
System.out.println("args[" + i + "]: " + args[i]);
}
}
}
```

Try executing this program as shown here -

```
C:\Java\bin>java CommandLine this is a command line 200 -100
```

Output

```
args[0]: this
args[1]: is
 args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```