Course Title: 403 Java Programming Language

Unit 1. Introduction to Java

1.1 Properties of Java

1.2 Comparison of java with C++

1.3 Java Compiler, Java Interpreter

1.4 Identifier, Literals, Operators, Variables, Keywords, Data Types

1.5 Branching: If – Else, Switch

1.6 Looping: While, Do-while, For

1.7 Type Casting

Java Programming- History of Java

➔ The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

➔ For the green team members, it was an advance concept at that time. But it was suited for internet programming. Later, Java technology as incorporated by Netscape.

➔ Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc.

➔ There are given the major points that describes the history of java.

1. **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in **June 1991**. The small team of sun engineers called Green Team
2. Originally designed for small, embedded systems in electronic appliances like set top boxes.
3. Firstly, it was called **"Green talk"** by James Gosling and file extension was .gt.
4. After that, it was called **Oak** and was developed as a part of the Green project.

Why Oak name for java language?

5. Why Oak? Oak is a symbol of strength and choose as a national tree of many countries like U.S.A., France, Germany, Romania etc.
6. **In 1995**, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why they choose java name for java language?

7. The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say. According to James Gosling "Java was one of the top choices along with Silk". Since java was so unique, most of the team members preferred java.
8. Java is an island of Indonesia where first coffee was produced (called java coffee).
9. Notice that Java is just a name not an acronym.
10. **Originally developed by James Gosling at Sun Microsystems** (which is now a subsidiary of **Oracle Corporation**) and **released in 1995** as core component of Sun Microsystems 'Java platform (Java 1.0 [J2SE]).
11. In 1995, Time magazine called Java one of the Ten Best Products of 1995.
12. With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: **J2EE for Enterprise Applications, J2ME for Mobile Applications.**
13. **Sun Microsystems has renamed the new J2 versions as Java SE, Java EE and Java ME, respectively**.
14. Java is guaranteed to be **Write Once, Run Anywhere. (WORA)**
15. Java is an **Object Oriented**. In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

1.1 Properties of Java

➔ 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.**

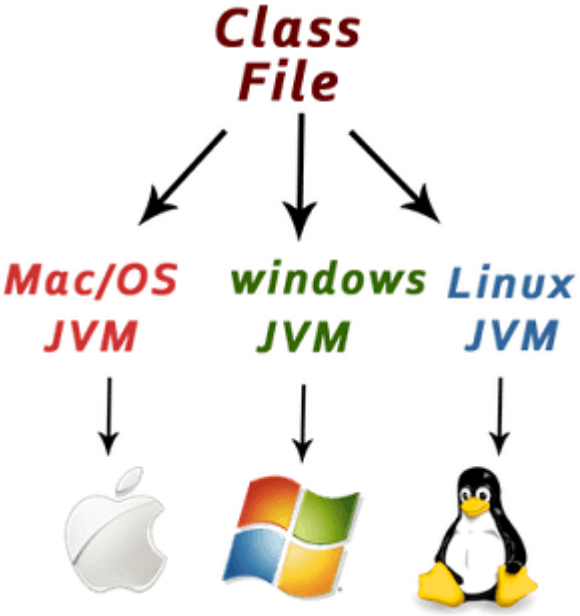
➔ A list of the most important features of the Java language is listed below.

1. **Compiled and Interpreted**

Basically, a computer language is either compiled or interpreted. Java comes together both these approaches thus making Java a two-stage system. Java compiler translates Java code to Bytecode instructions and Java Interpreter generate machine code that can be directly executed by machine that is running the Java program.

2. **Platform Independent and portable**

Java supports the feature portability. Java programs can be easily moved from one computer system to another and anywhere. Changes and upgrades in operating systems, processors and system resources will not force any alteration in Java programs. This is reason why Java has become a trendy language for programming on Internet which interconnects different kind of systems worldwide. Java certifies portability in two ways. First way is, Java compiler generates the bytecode and that can be executed on any machine. Second way is, size of primitive data types are machine independent.



3. **Object- oriented**

Java is truly object-oriented language. In Java, almost everything is an Object. All program code and data exist in objects and classes. Java comes with an extensive set of classes; organize in packages that can be used in program by Inheritance. The object model in Java is trouble-free and easy to enlarge.

4. **Robust and secure**

Java is a most strong language which provides many securities to make certain reliable code. It is design as garbage –collected language, which helps the programmers virtually from all memory management problems. Java also includes the concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system. Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet. The absence of pointers in Java ensures that programs cannot get right of entry to memory location without proper approval.

5. **Distributed**

Java is called as Distributed language for construct applications on networks which can contribute both data and programs. Java applications can open and access remote objects on Internet easily. That means multiple programmers at multiple remote locations to work together on single task.

6. **Simple and small**

Java is very small and simple language. Java does not use pointer and header files, goto statements, etc. It eliminates operator overloading and multiple inheritance.

7. **Multithreaded and Interactive**

Multithreaded means managing multiple tasks simultaneously. Java maintains multithreaded programs. That means we need not wait for the application to complete one task before starting next task. This feature is helpful for graphic applications.

8. **High performance**

Java performance is very extraordinary for an interpreted language, majorly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. The incorporation of multithreading improves the execution speed of program.

9. **Dynamic and Extensible**

Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. Java can also establish the type of class through the query building it possible to either dynamically link or abort the program, depending on the reply. Java program is support functions written in other language such as C and C++, known as native methods.

1.2 Comparison of java with C++

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

➔ Java is a true object-oriented language while C++ is basically C with object-oriented extension. That is what exactly the increment operator ++ indicates. C++ has maintained backward compatibility with C. It is therefore possible to write an old style C program and run it successfully under C++.

➔ Java appears to be similar to C++ when we consider only the "extension" part of C++. However, some object-oriented features of C++ make the C++ code extremely difficult to follow and maintain.

➔ Listed below are some major C++ features that were intentionally omitted from Java or significantly modified.

- Java does not support operator overloading.
- Java does not have template classes as in C++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".
- Java does not support global variables. Every variable *and* method is declared within a *class and* forms part of that class.
- Java does not use pointers.
- Java *has* replaced the destructor function with **a *finalize( )*** function.
- There *are* no header files in Java.

Java also adds some *new* features. While C++ is a superset of C, Java is neither a superset nor a subset of C or C++. Java may be considered as a first cousin of C++ *and* a second cousin of C

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.

Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.

unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Note

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

❖ Java environment

Java environment includes a large number of development tools and hundreds of classes and Methods. The development tools are part of the system known as ***Java Development Kit (JDK)*** and the classes and methods are part of the *Java Standard Library* (JSL), also known as the ***Application Programming Interface (API)***

Java Development Kit

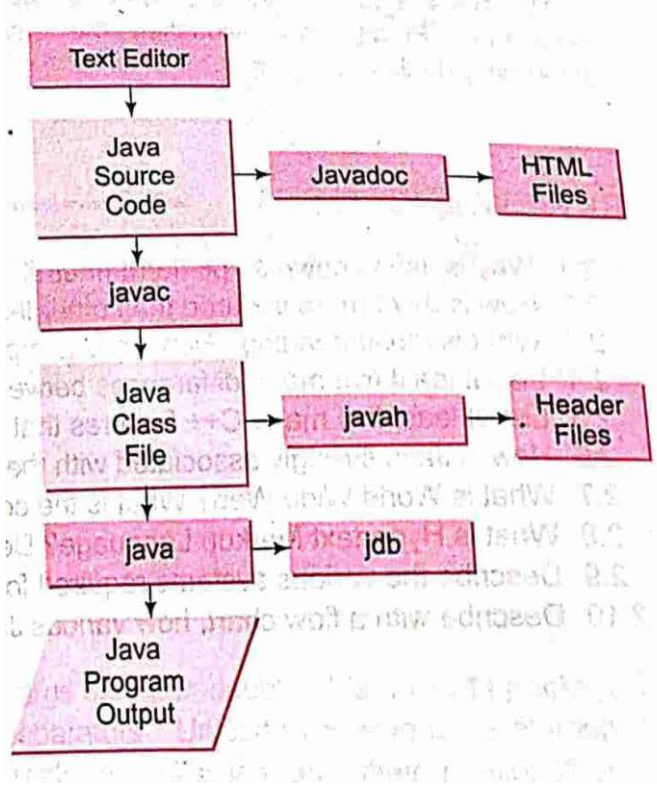
The Java Development Kit comes with a collection of tools that are used for developing and running the Java programs. They include:

- appletviewer (for viewing Java applets)
- javac (Java compiler)
- java (Java interpreter)
- javap (Java disassembler)
- javah (for C header files)
- javadoc (for creating HTML documents)
- jdb (Java debugger)

Java Development Tools

Tool	Description
appletviewer	Enables us to run Java applets (without actually using a Java-compatible browser).
java	Java interpreter, which runs applets and applications by reading and interpreting bytecode files.
javac	The Java compiler, which translates Java sourcecode to bytecode files that the interpreter can understand.
javadoc	Creates HTML-format documentation from Java source code files.
javah	Produces header files for use with native methods.
javap	Java disassembler, which enables us to convert bytecode files into a program description.
jdb	Java debugger, which helps us to find errors in our programs.

Java programming interface



Process of building and running Java application programs

Application Programming Interface
The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages

Most commonly used packages are:

Language Support Package: A collection of classes and methods required for implementing basic features of Java.

Utilities Package: A collection of classes to provide utility functions such as date and time functions.

Input/Output Package: A collection of classes required for input/output manipulation.

Networking Package: A collection of classes for communicating with other computers via Internet_

AWT Package: The Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface.

Applet Package: This includes a set of classes that allows us to create Java applets.

Java Runtime Environment

The Java Runtime Environment (JRE) facilitates the execution of programs developed in Java. It primarily comprises the following:

Java Virtual Machine (JVM): It is a program that interprets the intermediate Java byte code and generates the desired output. It is because of byte code and JVM concepts that programs written in Java are highly portable.
Runtime class libraries: These are a set of core class libraries that are required for the execution of Java programs.
User interface toolkits: AWT and Swing are examples of toolkits that support varied input methods for the users to interact with the application program.
Deployment technologies: JRE comprises the following key deployment technologies
1.Java plug-in: Enables the execution of a Java applet on the browser.
2.Java Web Start: Enables remote-deployment of an application. With Web Start, users can launch an application directly from the Web browser without going through the installation procedure.

JAVA Bytecode

Why understands bytecode?

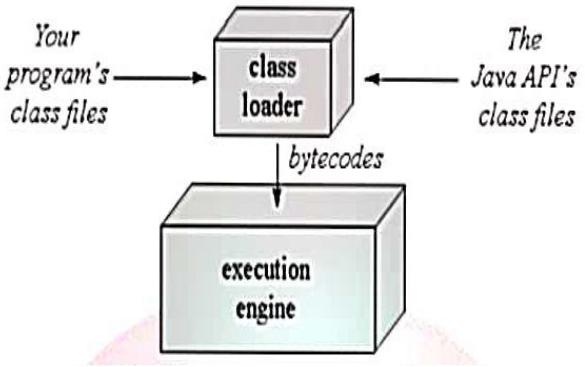
➔ Bytecode is computer object code that is processed by a program, usually referred to as virtual, rather than by the "real" computer machine, the hardware processor. The virtual machine converts each generalized machine instruction into a specific machine instruction or instructions that this computer's processor will understand.

➔ Java code is written in .java file. This code contains one or more Java language attributes like Classes, Methods, Variable, Objects etc. Javac is used to compile this code and to generate .class file. Class file is **also known as "byte code".**

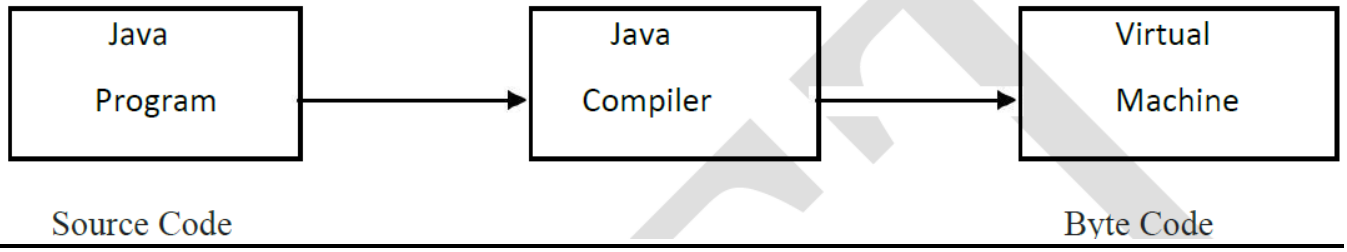
JAVA VIRTUAL MACHINE (JVM)

JVM is the main component of Java architecture and it is **the** part **of** the JRE (Java Runtime Environment). it provides the cross-platform functionality to java. This is a software process that converts the compiled Java byte code to machine code.

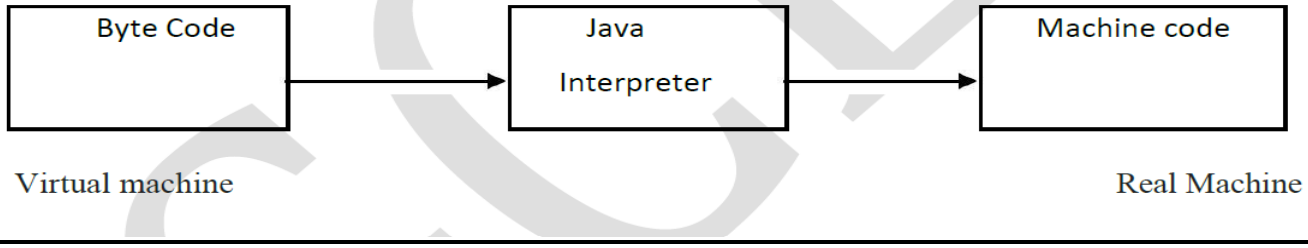
A Java virtual machine's main job is to load class files and execute the bytecodes they contain. Java virtual machine contains a class loader, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program **are** loaded into the virtual machine, The bytecodes are executed in an execution engine.



As we know that all programming language compilers convert the source code to machine code Same job done by Java Compiler to run a Java program, but the difference is that Java compiler convert the source code into Intermediate code is called as bytecode. This machine is called the *Java Virtual machine* and it exits only inside the computer memory.
Following figure shows the process of compilation.



The Virtual machine code is not machine specific. The machine specific code is generated. By Java interpreter by acting as an intermediary between the virtual machine and real machines shown below



1.3 Java Compiler, Java Interpreter

- **javac (Java compiler)**

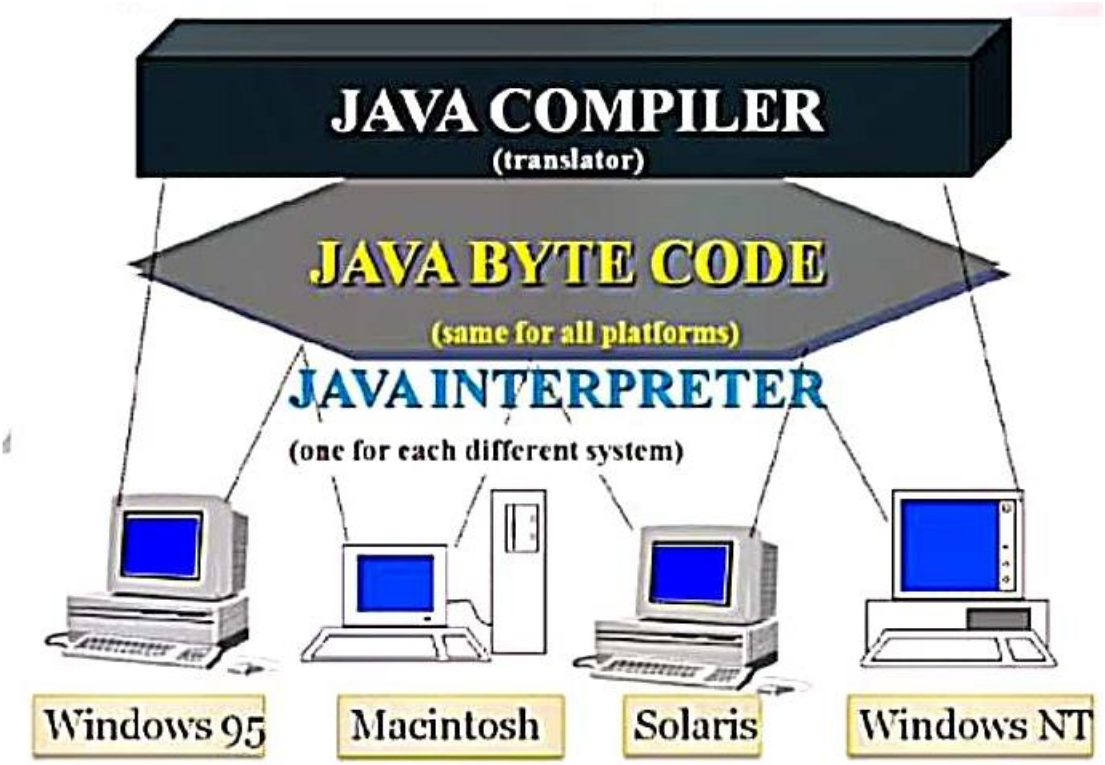
In java, we can use any text editor for writing program and then save that program with .java extension. Java compiler convert the source code or program in bytecode and interpreter convert .java file in .class file.

Syntax:
C:\javac filename.java
If my filename is —abc.java then the syntax will be
C:\javac abc.java

- **java(Java Interpreter)**

As we learn that, we can use any text editor for writing program and then save that program with .java extension. Java compiler convert the source code or program in bytecode and interpreter convert .java file in .class file.

Syntax:
C:\java filename
If my filename is abc.java then the syntax will be
C:\java abc



1.4 Identifier, Literals, Operators, Variables, Keywords, Data Types

JAVA identifier

- Identifier means any legal names of classes, variables, methods , functions etc. when we define names for classes, variables, and methods are called identifiers.
- In Java, followings are rules to define valid identifiers:
  a. All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
  b. After the first character, identifiers can have any combination of characters.
  c. A key word cannot be used as an identifier.
  d. Most importantly, identifiers are case sensitive.

Examples of legal identifiers: age, $charges, _point_1_value
Examples of illegal identifiers: 1a, -ve

Java Literals

Any constant value which can be assigned to the variable is called as literal/constant

1. Integer Literals

- Most Commonly used type in typical program is any whole number value is an integer literal.
- For Example: 1, 2, 3 and 25
- Note: byte, short, long literals value is also type as same Integer literals

2. Floating Point Literals

- Floating point number represents decimal values with fractional components,
- For Example: 6.023E23, 3.1415

3. Boolean Literals

- There are only two logical values that a Boolean value can have true or false

4. Character Literals

- Characters in JAVA are indicates into the Unicode character set. Character literal is represent inside a pair of single quote (‘ ‘)
- For Example: ‘x’ ,’y’

5. String Literals

- String literals in JAVA are specified like they are in most other languages by enclosing a sequence of character between a pair of double quotes (" ")
- For Example: "Hello World" , "SYBCA division 3 and 4"

6. The Null Literal

- The null type has one value, the null reference, represented by the literal null, which is formed from ASCII characters. A null literal is always of the null type.
- For Example: null

Operators in Java

It is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

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

1. **– :Unary minus**, used for negating the values.
2. **+ :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.
3. **++ :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.
4. **— : 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.
5. **! : Logical not operator**, used for inverting a boolean value.

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

1. *** :** Multiplication
2. **/ :** Division
3. **% :** Modulo
4. **+ :** Addition
5. **– :** Subtraction

iii. **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.

iv. **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

Some of the relational operators are-

1. **==, Equal to :** returns true if left hand side is equal to right hand side.
2. **!=, Not Equal to :** returns true if left hand side is not equal to right hand side.
3. **<, less than :** returns true if left hand side is less than right hand side.
4. **<=, less than or equal to :** returns true if left hand side is less than or equal to right hand side.
5. **>, Greater than :** returns true if left hand side is greater than right hand side.
6. **>=, Greater than or equal to:** returns true if left hand side is greater than or equal to right hand side.

v. **Logical Operators :** These operators are used to perform "logical AND" and "logical OR" operation, i.e. the function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e. it has a short-circuiting effect. Used extensively to test for several conditions for making a decision.
Conditional operators are-

1. **&&, Logical AND :** returns true when both conditions are true.
2. **||, Logical OR :** returns true if at least one condition is true.

vi. **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary. General format is-

condition ? if true : if false

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

vii. **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.

viii. **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.

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

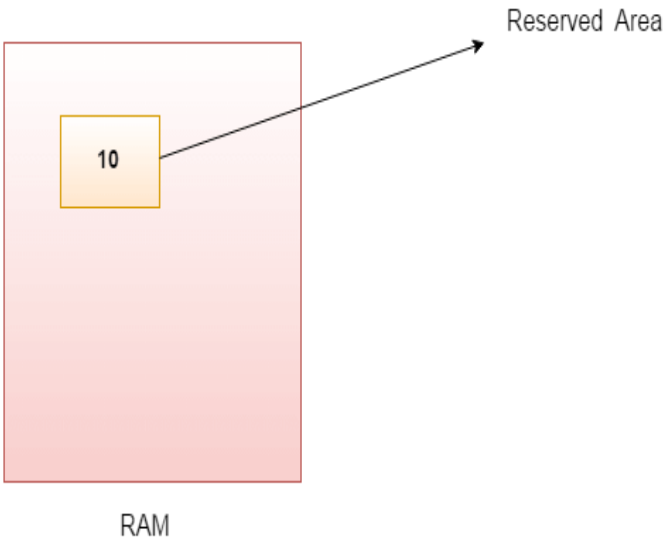
Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

Variable

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.




```
int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
public class A
{
  static int m=100;//static variable
  void method()
  {
    int n=90;//local variable
  }
  public static void main(String args[])
  {
    int data=50;//instance variable
  } }//end of class
```

Data Types in Java

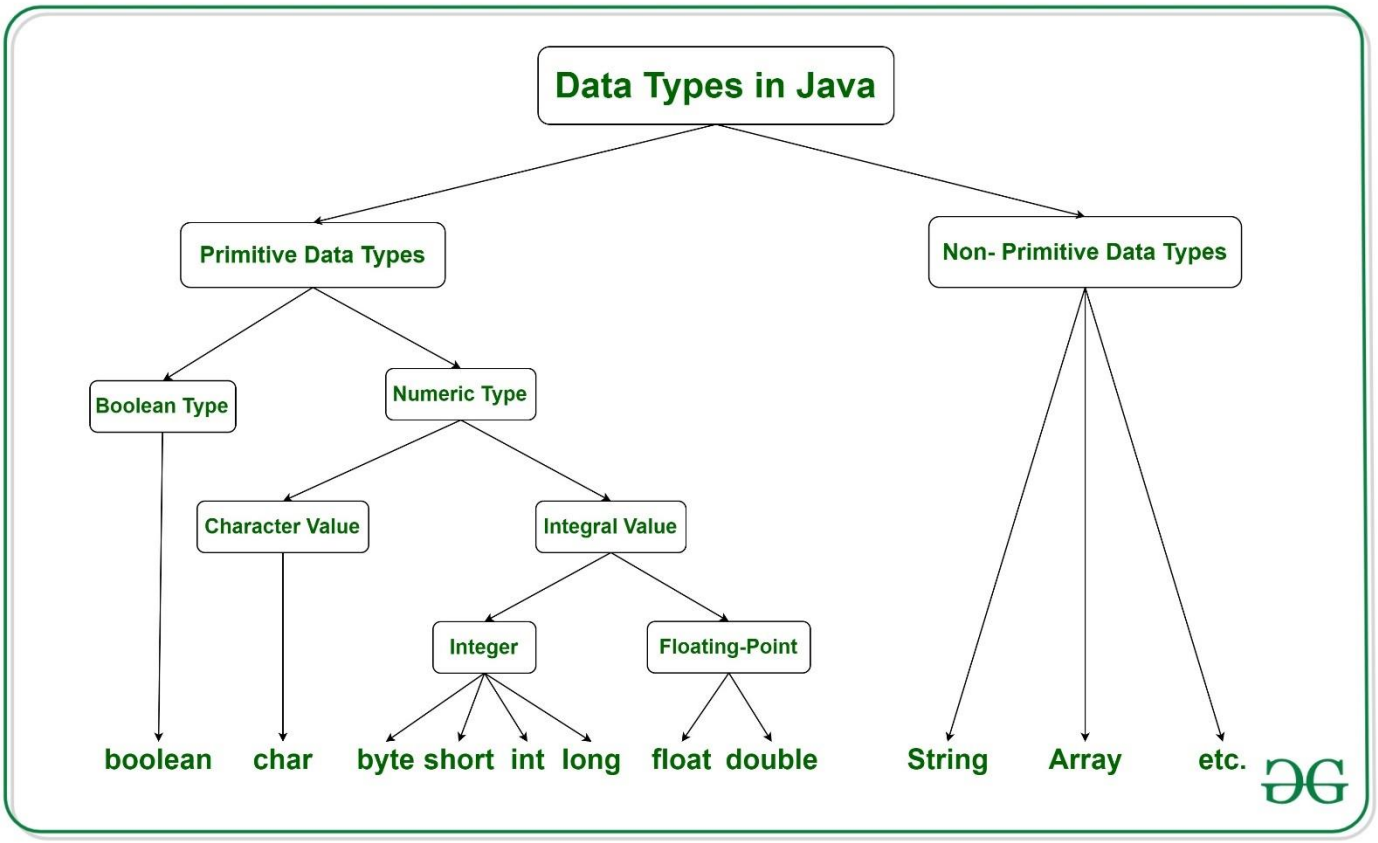
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

NOTE : java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Java Keywords

- Keywords are reserved words which are used by lava programming language for its own feature and functionality. These keywords cannot be used as an Identifier'
- The keywords const and goto are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs.

Keywords In Java

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

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

1.5 Branching: If – Else, Switch (Decision Making statements)

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

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

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

Example:

```
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Output:

```
Age is greater than 18
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

Example:

```
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

Output:

```
odd number
```

Using Ternary Operator

We can also use ternary operator (? :) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned.

Example:

```
public class IfElseTernaryExample {
public static void main(String[] args) {
   int number=13;
   //Using ternary operator
   String output=(number%2==0)?"even number":"odd number";
   System.out.println(output);
}
}
```

Output:

```
odd number
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Example:

```
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
```

Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

Example:

```
public class JavaNestedIfExample {
public static void main(String[] args) {
   //Creating two variables for age and weight
   int age=20;
   int weight=80;
   //applying condition on age and weight
   if(age>=18){
      if(weight>50){
         System.out.println("You are eligible to donate blood");
      }
   }
}}
```

Output:

```
You are eligible to donate blood
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

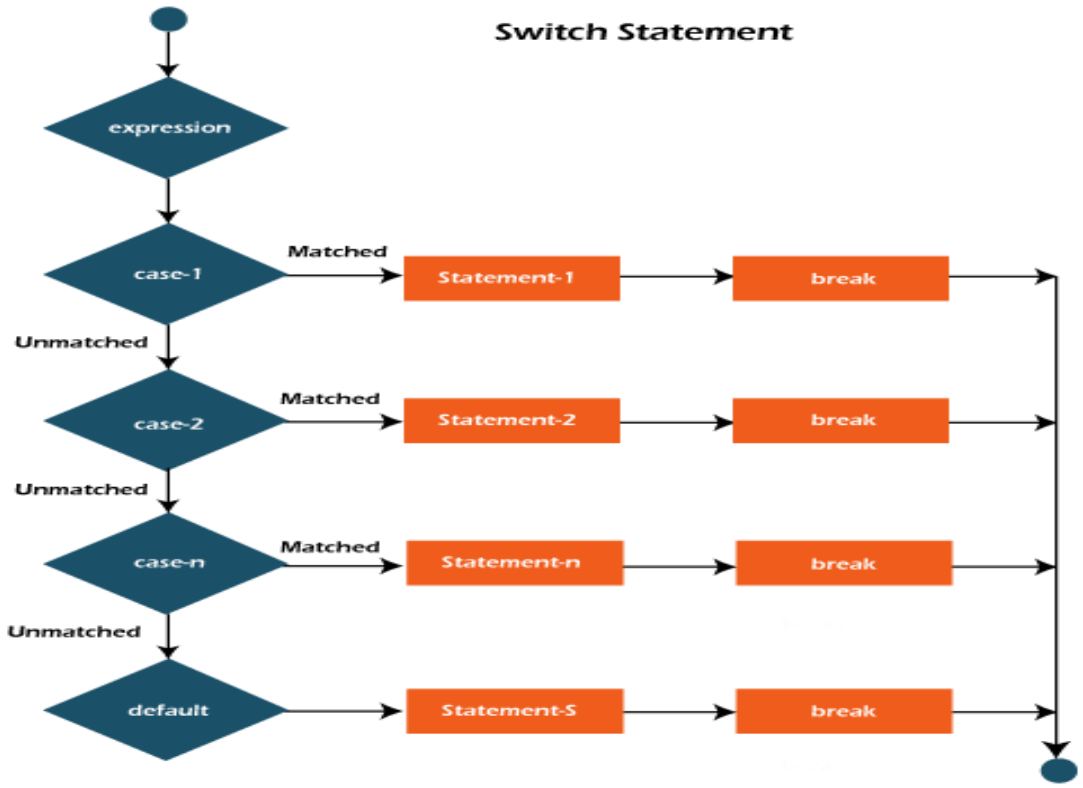
Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

```
switch (expression){
   case value1:
    statement1;
    break;
   .
   .
   .
   case valueN:
    statementN;
    break;
   default:
    default statement;
}
```

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.



Example:

```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Output:

```
20
```

Java Nested Switch Statement

We can use switch statement inside other switch statement in Java. It is known as nested switch statement.

Example:

```
//Java Program to demonstrate the use of Java Nested Switch
public class NestedSwitchExample {
   public static void main(String args[])
    {
    //C - CSE, E - ECE, M - Mechanical
     char branch = 'C';
     int collegeYear = 4;
     switch( collegeYear )
     {
        case 1:
           System.out.println("English, Maths, Science");
           break;
        case 2:
           switch( branch )
           {
              case 'C':
                 System.out.println("Operating System, Java, Data Structure");
                 break;
              case 'E':
                 System.out.println("Micro processors, Logic switching theory");
                 break;
              case 'M':
                 System.out.println("Drawing, Manufacturing Machines");
                 break;
           }
           break;
        case 3:
           switch( branch )
           {
              case 'C':
                 System.out.println("Computer Organization, MultiMedia");
                 break;
```

```
                case 'E':
                    System.out.println("Fundamentals of Logic Design, Microelectronics");
                    break;
                case 'M':
                    System.out.println("Internal Combustion Engines, Mechanical Vibration");

                    break;
            }
            break;
        case 4:
            switch( branch )
            {
                case 'C':
                    System.out.println("Data Communication and Networks, MultiMedia");
                    break;
                case 'E':
                    System.out.println("Embedded System, Image Processing");
                    break;
                case 'M':
                    System.out.println("Production Technology, Thermal Engineering");
                    break;
            }
            break;
        }
    }
}
```

Output:

```
Data Communication and Networks, MultiMedia
```

1.6 Looping: While, Do-while, For (Loop statements)

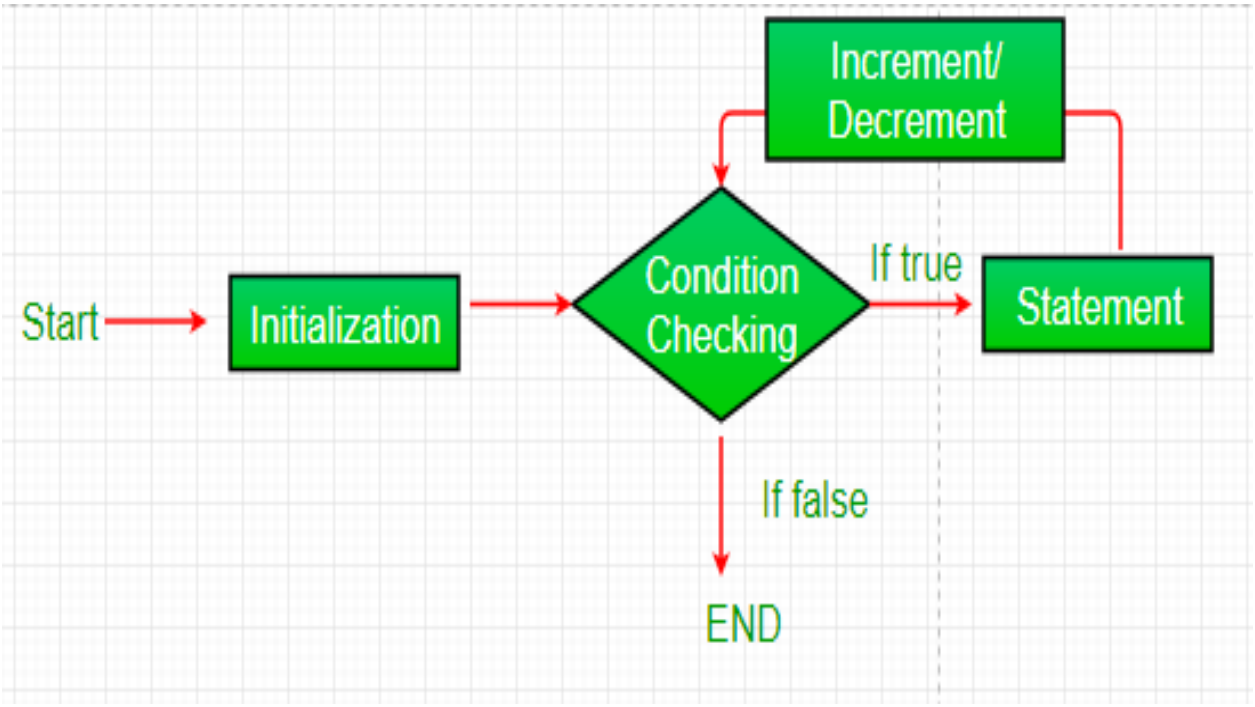
Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. **for loop**: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition;
                        increment/decrement)
{
   statement(s)
}
```

Flowchart:



1. **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
3. **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
4. **Increment/ Decrement:** It is used for updating the variable for next iteration.
5. **Loop termination:**When the condition becomes false, the loop terminates marking the end of its life cycle.

Example

//Java Program to demonstrate the example of for loop which prints table of 1

```
public class ForExample {
public static void main(String[] args) {
   //Code of Java for loop
   for(int i=1;i<=10;i++){
      System.out.println(i);
   }
}
}
```

Output:

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

Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
      System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

Output:

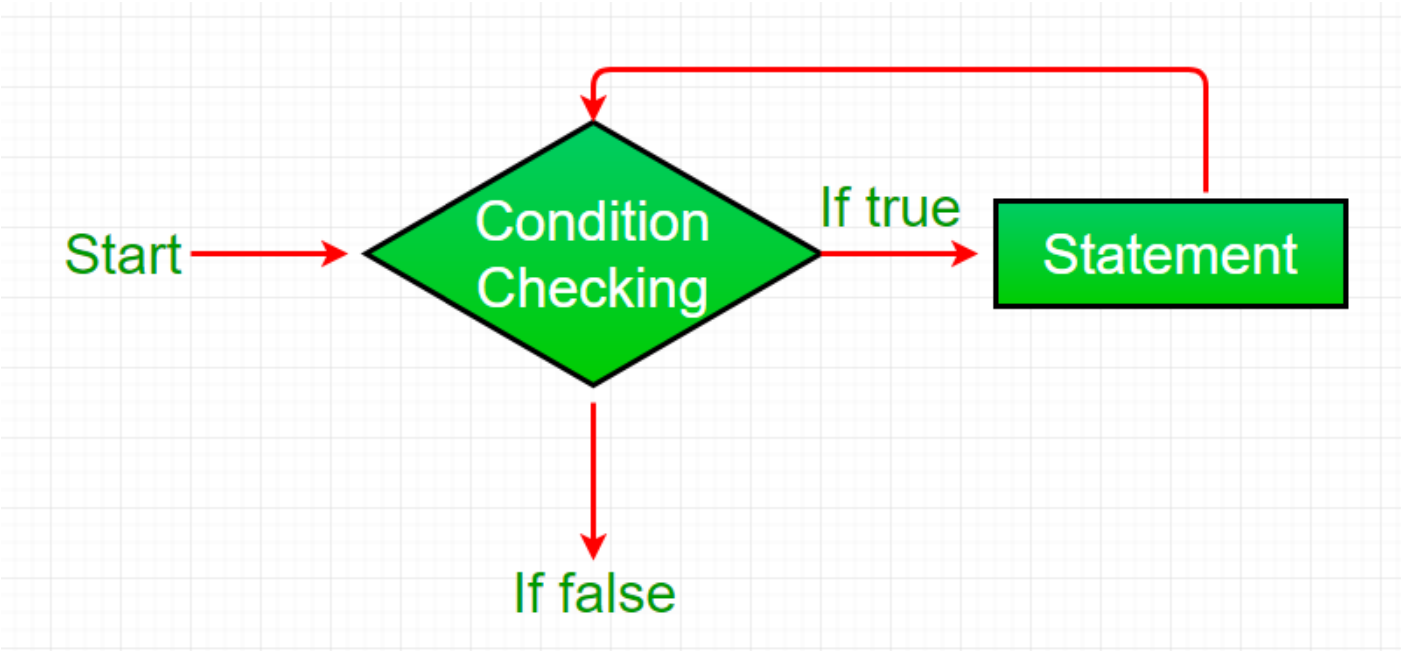
```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

2. **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
  loop statements...
}
```

Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   }
}
}
```

Output:

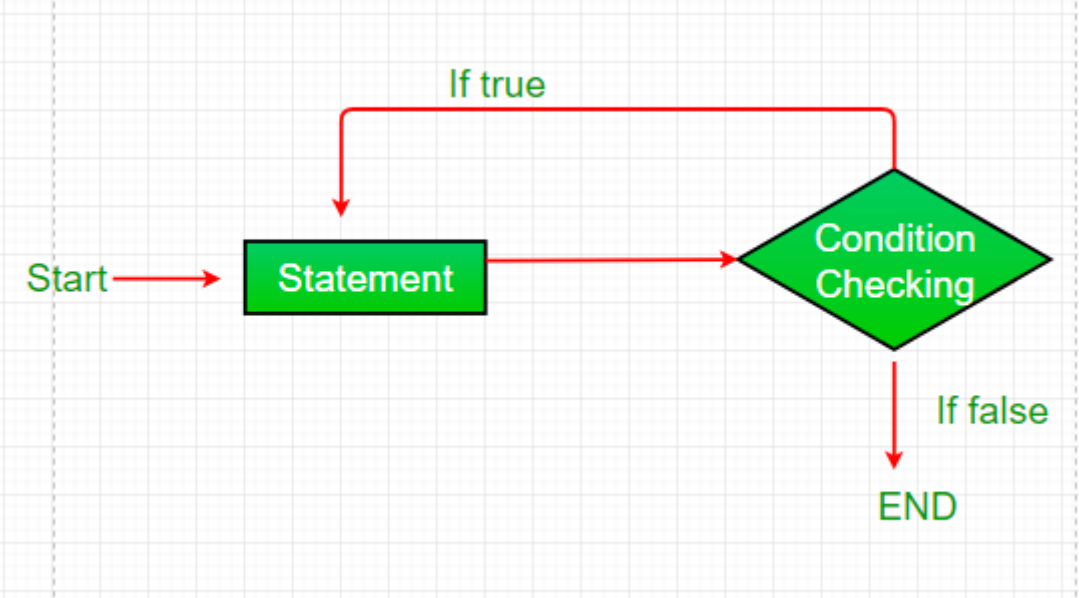
```
2
3
4
5
6
7
8
9
10
```

3. **do while:** do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax:

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

Flowchart:



1. do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
2. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
3. When the condition becomes false, the loop terminates which marks the end of its life cycle.
4. It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

Output:

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

Java for-each Loop

The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on the basis of elements and not the index. It returns element one by one in the defined variable.

Syntax:

```
for(data_type variable : array_name)
{
    //code to be executed
}
```

Example:

//Java For-each loop example which prints the elements of the array

```
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr){
        System.out.println(i);
    }
} }
```

Output:

```
12
23
44
56
78
```

Infinite loop: One of the most common mistakes while implementing any sort of looping is that that it may not ever exit, that is the loop runs for infinite time. This happens when the condition fails for some reason.

`for(;;){`
`//code to be executed`
`}`
	`while(true){`
	`//code to be executed`
	`}`
		`do{`
		`//code to be executed`
		`}while(true);`

Jump statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

```
//Java Program to demonstrate the use of break statement inside the for loop.
public 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
```

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

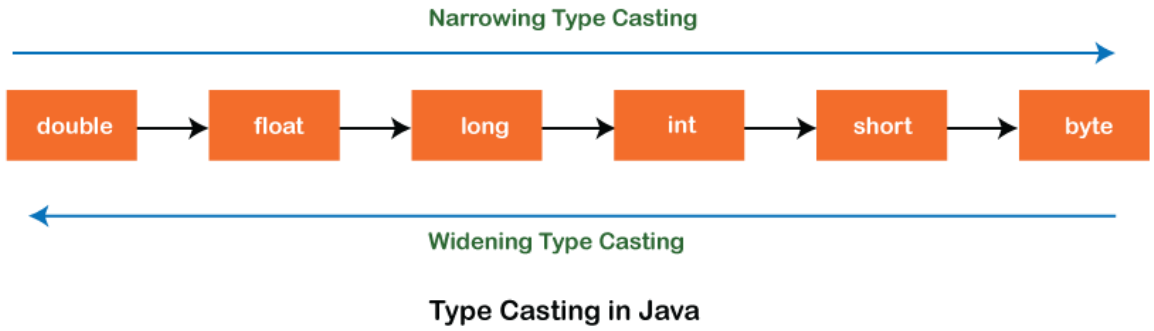
```
/Java Program to demonstrate the use of continue statement inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Output:

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

1.7 Type Casting

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.



Type Casting in Java

1. Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

1. **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

Example:

```
int myInt = 9;
double myDouble = myInt; // Automatic casting: int to double
System.out.println(myInt);      // Outputs 9
System.out.println(myDouble);   // Outputs 9.0
```

2. Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Example:

```
double myDouble = 9.78d;
int myInt = (int) myDouble; // Manual casting: double to int
System.out.println(myDouble);   // Outputs 9.78
System.out.println(myInt);      // Outputs 9
```