Java Programming

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Example of simple java program

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

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

Desktop Applications such as acrobat reader, media player, antivirus, etc.

Web Applications such as irctc.co.in, javatpoint.com, etc.

Enterprise Applications such as banking applications.

Mobile

Embedded System

Smart Card

Robotics

Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone

application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)

4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

Features 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 given below.

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, 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

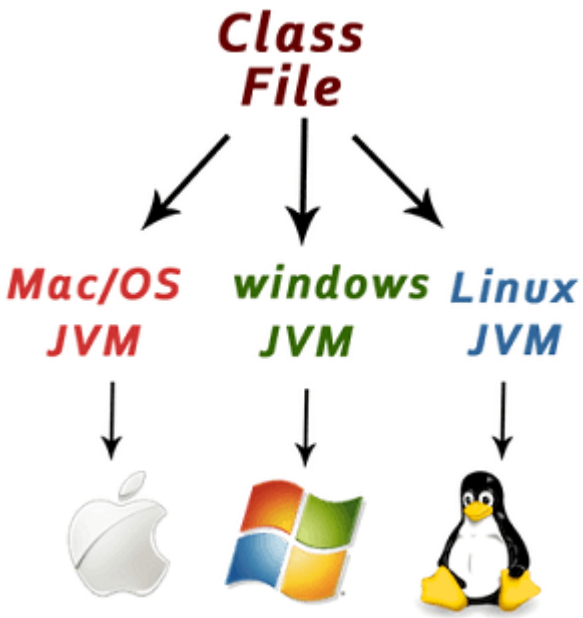
Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. 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).

C++ vs Java

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:

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	Java was designed and created as an interpreter for printing systems but later extended as a support network

	extension of the C programming language.	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.

First Java Program | Hello World Example

1. Software Requirements
2. Creating Hello Java Example
3. Resolving javac is not recognized

In this section, we will learn how to write the simple program of Java. We can write a simple hello Java program easily after installing the JDK.

To create a simple Java program, you need to create a class that contains the main method. Let's understand the requirement first.

The requirement for Java Hello World Example

For executing any Java program, the following software or application must be properly installed.

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the Java program
- Compile and run the Java program

Creating Hello World Example

Let's create the hello java program:

```
1. class Simple{
2.     public static void main(String args[]){
3.       System.out.println("Hello Java");
4.     }
5. }
```

Test it Now

Save the above file as Simple.java.

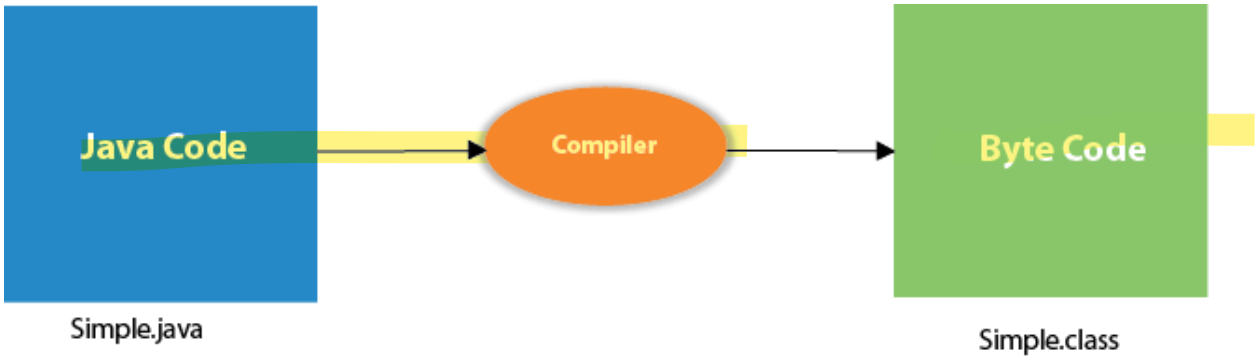
To compile:	javac Simple.java
To execute:	java Simple

Output:

```
Hello Java
```

Compilation Flow:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.



Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

JVM (Java Virtual Machine

1. Java Virtual Machine

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

Variables

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.

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.

```
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

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.

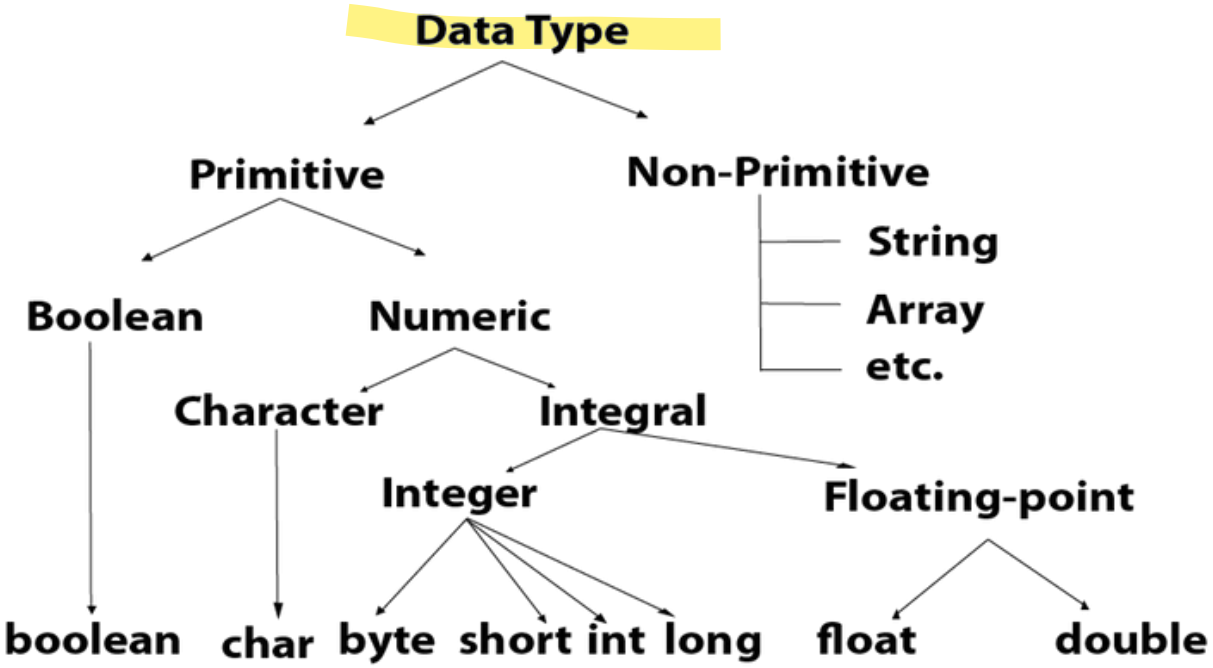
Java Primitive Data Types

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

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type

- float data type
- double data type



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

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

```
1. Boolean one = false
```

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

```
1. byte a = 10, byte b = -20
```

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

```
1. short s = 10000, short r = -5000
```

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

```
1. int a = 100000, int b = -200000
```

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

```
1. long a = 100000L, long b = -200000L
```

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

```
1. float f1 = 234.5f
```

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

```
1. double d1 = 12.3
```

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

```
1. char letterA = 'A'
```

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most languages.

Why java uses Unicode System?

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for chinese, and so on.

Operators in Java

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

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

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence

Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression

- inverting the value of a boolean

Java Unary Operator Example: ++ and --

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int x=10;
4. System.out.println(x++);//10 (11)
5. System.out.println(++x);//12
6. System.out.println(x--);//12 (11)
7. System.out.println(--x);//10
8. }}
```

Output:

```
10
12
12
10
```

Control Structure

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

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

Let's understand the if-statements one by one.

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.

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
```

Consider the following example in which we have used the **if** statement in the java code.

Student.java

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. int x = 10;
4. int y = 12;
5. if(x+y > 20) {
6. System.out.println("x + y is greater than 20");
7. }
8. }
9. }
```

Output:

```
x + y is greater than 20
```

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:

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
4. else{
5. statement 2; //executes when condition is false
6. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. int x = 10;
4. int y = 12;
5. if(x+y < 10) {
6. System.out.println("x + y is less than 10");
7. } else {
8. System.out.println("x + y is greater than 20");
```

```
9.  }
10. }
11. }
```

Output:

```
x + y is greater than 20
```

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.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. }
4. else if(condition 2) {
5. statement 2; //executes when condition 2 is true
6. }
7. else {
8. statement 2; //executes when all the conditions are false
9. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. String city = "Delhi";
4. if(city == "Meerut") {
5. System.out.println("city is meerut");
6. }else if (city == "Noida") {
7. System.out.println("city is noida");
8. }else if(city == "Agra") {
9. System.out.println("city is agra");
10. }else {
```

```
11. System.out.println(city);
12. }
13. }
14. }
```

Output:

```
Delhi
```

4. 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.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. if(condition 2) {
4. statement 2; //executes when condition 2 is true
5. }
6. else{
7. statement 2; //executes when condition 2 is false
8. }
9. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. String address = "Delhi, India";
4.
5. if(address.endsWith("India")) {
6. if(address.contains("Meerut")) {
7. System.out.println("Your city is Meerut");
8. }else if(address.contains("Noida")) {
9. System.out.println("Your city is Noida");
10. }else {
```

```
11. System.out.println(address.split(",")[0]);
12. }
13. }else {
14. System.out.println("You are not living in India");
15. }
16. }
17. }
```

Output:

```
Delhi
```

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.

```
1. switch (expression){
2.     case value1:
3.      statement1;
4.      break;
5.     .
6.     .
7.     .
```

```
8.      case valueN:
9.       statementN;
10.     break;
11.    default:
12.     default statement;
13. }
```

Consider the following example to understand the flow of the switch statement.

Student.java

```
1.  public class Student implements Cloneable {
2.  public static void main(String[] args) {
3.  int num = 2;
4.  switch (num){
5.  case 0:
6.  System.out.println("number is 0");
7.  break;
8.  case 1:
9.  System.out.println("number is 1");
10. break;
11. default:
12. System.out.println(num);
13. }
14. }
15. }
```

Output:

```
2
```

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.

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

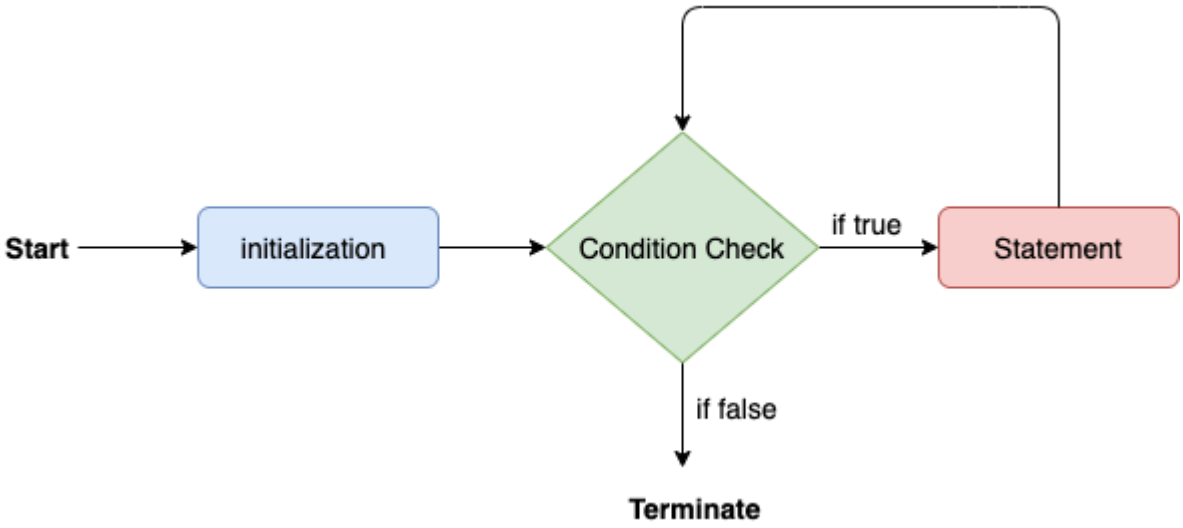
Let's understand the loop statements one by one.

Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
1. for(initialization, condition, increment/decrement) {
2. //block of statements
3. }
```

The flow chart for the for-loop is given below.



Consider the following example to understand the proper functioning of the for loop in java.

Calculation.java

```
1. public class Calculattion {
2. public static void main(String[] args) {
```

```
3. // TODO Auto-generated method stub
4. int sum = 0;
5. for(int j = 1; j<=10; j++) {
6. sum = sum + j;
7. }
8. System.out.println("The sum of first 10 natural numbers is " + sum);
9. }
10.}
```

Output:

```
The sum of first 10 natural numbers is 55
```

Java for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
1. for(data_type var : array_name/collection_name){
2. //statements
3. }
```

Consider the following example to understand the functioning of the for-each loop in Java.

Calculation.java

```
1. public class Calculation {
2. public static void main(String[] args) {
3. // TODO Auto-generated method stub
4. String[] names = {"Java","C","C++","Python","JavaScript"};
5. System.out.println("Printing the content of the array names:\n");
6. for(String name:names) {
7. System.out.println(name);
8. }
9. }
10.}
```

Output:

```
Printing the content of the array names:

Java
C
C++
Python
JavaScript
```

Java while loop

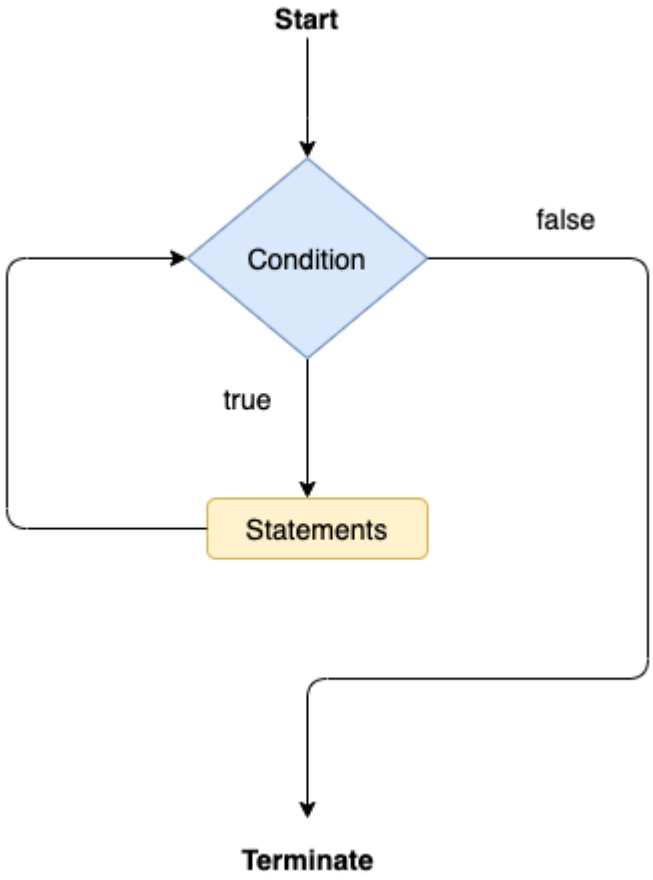
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
1. while(condition){
2. //looping statements
3. }
```

The flow chart for the while loop is given in the following image.



Consider the following example.

Calculation .java

```
1. public class Calculation {
2. public static void main(String[] args) {
3. // TODO Auto-generated method stub
4. int i = 0;
5. System.out.println("Printing the list of first 10 even numbers \n");
6. while(i<=10) {
7. System.out.println(i);
8. i = i + 2;
9. }
10.}
11.}
```

Output:

```
Printing the list of first 10 even numbers

0
```

```
2
4
6
8
10
```

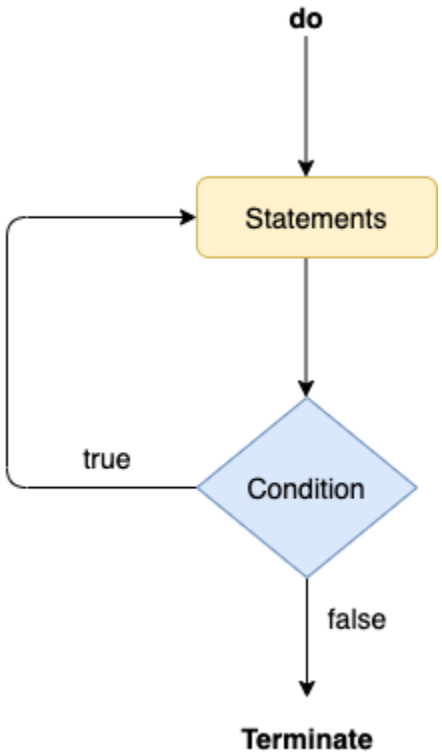
Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
1. do
2. {
3. //statements
4. } while (condition);
```

The flow chart of the do-while loop is given in the following image.



Consider the following example to understand the functioning of the do-while loop in Java.

Calculation.java

```
1. public class Calculation {
2. public static void main(String[] args) {
3. // TODO Auto-generated method stub
4. int i = 0;
5. System.out.println("Printing the list of first 10 even numbers \n");
6. do {
7. System.out.println(i);
8. i = i + 2;
9. }while(i<=10);
10.}
11.}
```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

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.

The break statement example with for loop

Consider the following example in which we have used the break statement with the for loop.

BreakExample.java

```
1.  public class BreakExample {
2.
3.  public static void main(String[] args) {
4.  // TODO Auto-generated method stub
5.  for(int i = 0; i<= 10; i++) {
6.  System.out.println(i);
7.  if(i==6) {
8.  break;
9.  }
10. }
11. }
12. }
```

Output:

```
0
1
2
3
4
5
6
```

break statement example with labeled for loop

Calculation.java

```
1.  public class Calculation {
2.
3.  public static void main(String[] args) {
4.  // TODO Auto-generated method stub
5.  a:
6.  for(int i = 0; i<= 10; i++) {
7.  b:
8.  for(int j = 0; j<=15;j++) {
9.  c:
10. for (int k = 0; k<=20; k++) {
11. System.out.println(k);
12. if(k==5) {
13. break a;
14. }
```

```
15. }
16. }
17.
18. }
19. }
20.
21.
22. }
```

Output:

```
0
1
2
3
4
5
```

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.

Consider the following example to understand the functioning of the continue statement in Java.

```
1.  public class ContinueExample {
2.
3.  public static void main(String[] args) {
4.  // TODO Auto-generated method stub
5.
6.  for(int i = 0; i<= 2; i++) {
7.
8.  for (int j = i; j<=5; j++) {
9.
10. if(j == 4) {
11. continue;
12. }
13. System.out.println(j);
14. }
15. }
```

16. }
17.
18. }

Output:

```
0
1
2
3
5
1
2
3
5
2
3
5
```

Method Overloading in java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

```
3. class Adder{
4. static int add(int a,int b){return a+b;}
5. static int add(int a,int b,int c){return a+b+c;}
```

```
}
class TestOverloading1{
public static void main(String[] args){
Adder A1=new Adder();
System.out.println(A1.add(11,11));
System.out.println(A1.add(11,11,11));
}}
```

Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
Adder A1=new Adder();
System.out.println(A1.add(11,11));//ambiguity
}}
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading4{
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
}
```

Array in Java

Normally, an array is a collection of similar type of elements which has contiguous memory location.

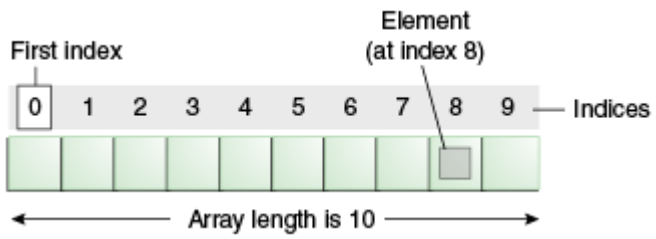
Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, **array is an object of a dynamically generated class.** Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

```
1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];
```

Instantiation of an Array in Java

```
1. arrayRefVar=new datatype[size];
```

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. class Testarray{
4. public static void main(String args[]){
5. int a[]=new int[5];//declaration and instantiation
6. a[0]=10;//initialization
7. a[1]=20;
8. a[2]=70;
9. a[3]=40;
10. a[4]=50;
11. //traversing array
12. for(int i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
14. }}
```

Test it Now

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1. int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization  of Java array in a single line
3. class Testarray1{
4. public static void main(String  args[]){
5. int a[]={33,3,4,5};//declaration, instantiation and initialization
6. //printing  array
7. for(int i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9. }}
```

Test it Now

Output:

```
33
3
4
5
```

For-each Loop for Java Array

We can also print the Java array using **for-each loop**. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
1. for(data_type variable:array){
2. //body of the loop
3. }
```

```
1. //Java Program to print the array elements using for-each loop
2. class Testarray1{
3. public static void main(String args[]){
```

```
4.  int arr[]={33,3,4,5};
5.  //printing array using for-each loop
6.  for(int i:arr)
7.  System.out.println(i);
8.  }}
```

Output:

```
33
3
4
5
```

Java Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

```
1.  public class JavaMathExample1
2.  {
3.      public static void main(String[] args)
4.      {
5.          double x = 28;
6.          double y = 4;
7.
8.          // return the maximum of two numbers
9.          System.out.println("Maximum number of x and y is: " +Math.max(x, y));
10.
11.         // return the square root of y
12.         System.out.println("Square root of y is: " + Math.sqrt(y));
13.
14.         //returns 28 power of 4 i.e. 28*28*28*28
15.         System.out.println("Power of x and y is: " + Math.pow(x, y));
16.
17.         // return the logarithm of given value
18.         System.out.println("Logarithm of x is: " + Math.log(x));
19.         System.out.println("Logarithm of y is: " + Math.log(y));
```

```
20.
21.         // return the logarithm of given value when base is 10
22.         System.out.println("log10 of x is: " + Math.log10(x));
23.         System.out.println("log10 of y is: " + Math.log10(y));
24.
25.         // return the log of x + 1
26.         System.out.println("log1p of x is: " +Math.log1p(x));
27.
28.         // return a power of 2
29.         System.out.println("exp of a is: " +Math.exp(x));
30.
31.         // return (a power of 2)-1
32.         System.out.println("expm1 of a is: " +Math.expm1(x));
33.     }
34. }
```