# Introduction to Java

Java was invented by James Gosling and his team and released in 1995 for public use. It was invented as a result of Project Green for Consumer Electronics.

Note Guys: This PDF has analytics included. So I would get to know whether you have read any particular line or not...Yeh Jaadu ka Kamaal hai...

## Features of Java.

**This language brings with it immense popularity due to its Buzzwords or Features.**

1. **It's Damn Simple** : For a person who has been coding in C and C++, the **syntax of Java is 70% similar**. So, if you are seeing these notes and you belong to C or C++ Coder Family, then Java is just a cup of coffee for you.

2. **Object Oriented** : All Java Programs are well structured around Objects. **Objects are nothing but real world entities, i.e. things that exist in real world. Each object has its own properties and behavior.** You are also an object, this means that you have some properties like your *skin-tone, hair-length, etc.* And talking about your behaviors, like on Sundays, you be like "Netflix and Chill".

3. **Robust** : *Java checks for any kind of errors at compile time and exceptions at runtime*. Java has a great inbuilt exception handling system which detects almost every Runtime Exception and saves the program from crashing like in C and C++.

4.  **Secure** : *Java programs run on a sandbox namely JVM.* JVM provides a **restricted environment** to run programs and restrict their access to Computer RAM and Hard Disks.

5.  **Scalable** : Java Applications are scalable around multiple systems and can be used to make a wide scaled project.

6.  **Multithreaded** : Java programs can run concurrently and simultaneously on multiple threads. This makes a Java Program multithreaded. The ability of a program to run on multiple threads is called Multithreading. Java provides a huge package for multithreading features and concurrency.

7.  **Write once, run anywhere** : **Java Programs are both compiled and interpreted.** The Java Compiler compiles and converts the source code to byte code. **This bytecode is a highly optimized set of instructions which is passed to Byte Loader Subsystem in the Java Virtual Machine to be interpreted and converted to machine code.**

These features bring Java to become one of the most popular language in the market now.

💡 Interview Question.
1. Why pointers were eliminated from Java?
2. What adds more to the Robustness of Java?

## Getting started with Installation of JDK 16.0.2 and setting up Environmental Variables.

So as to work with Java on your Local System, you need to have some kind of toolkit which would aid you in writing programs.

Step 1 : Open Browser and type : -

🛠️ https://www.oracle.com/java/technologies/downloads/#java16-windows

Step 2 : Click on the *jdk-16.0.2_windows-x64_bin.exe* and install it.

Step 3 : After download, run the .exe application. Press "YES" on a pop-up and keep clicking "Next" to install it successfully.

**Do not change the installation location of JDK**

Now that we have successfully installed JDK 16.0.2, head over to "Program Files/Java" and check if jdk-16.0.2 is present or not. If it is not present, then it is sure that you missed out a process or overlooked something.

## Post-process after JDK Installation.

Step 4 : Click on This PC🖥 and open it.

Step 5 : Right Click anywhere on the window and click *Properties* .

Step 6 : Look up for *Advanced System Settings* and then click on Environmental variables.
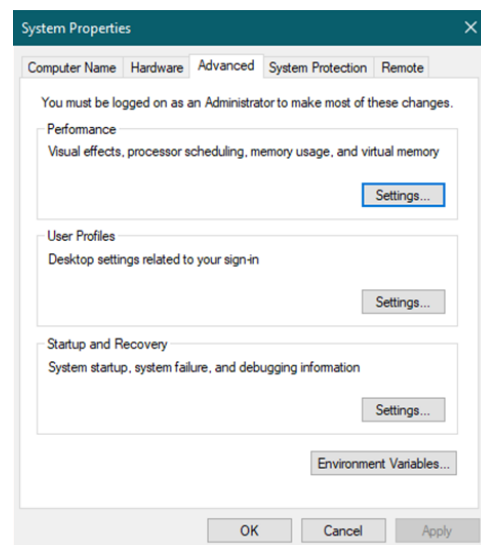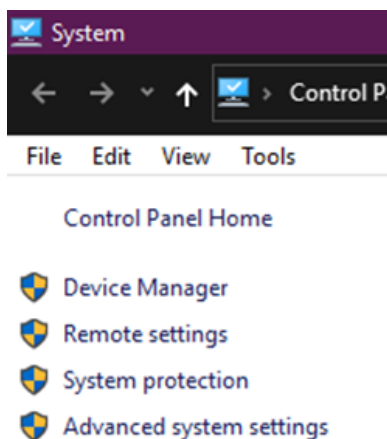
Step 7 : In the Dialog Box that appears, click "NEW" in the "User Variables for ..." and write the name "JAVA_HOME" as the Variable Name and mention the path of the Variable as the Folder of *bin* inside the *jdk-16.0.2* folder present in Program Files.
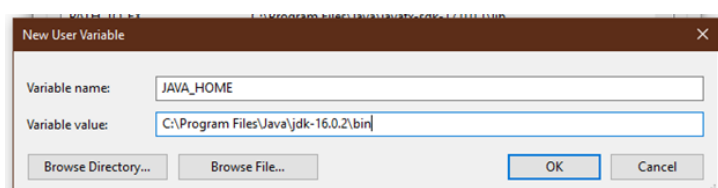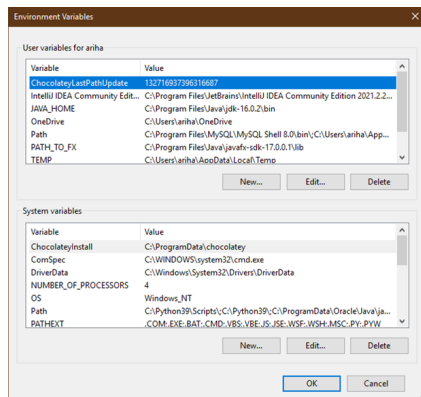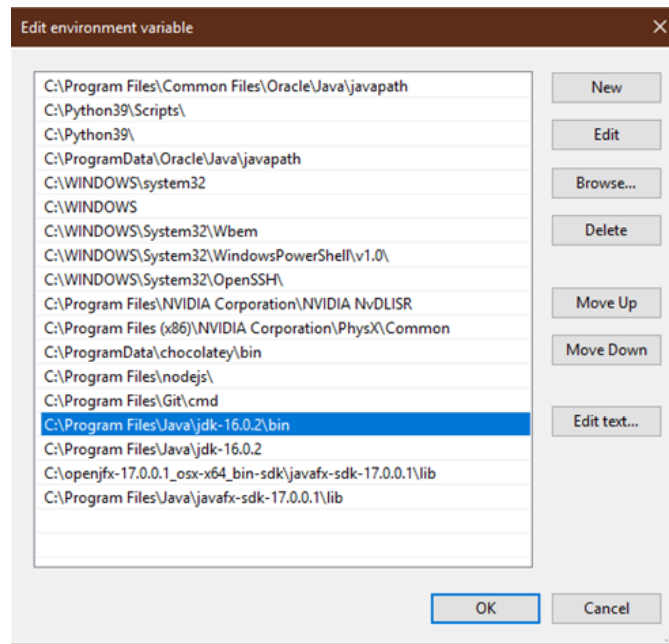
Step 8 : Under the **System Variables** section, select *Path* and click on it. Then press the **Edit** option.
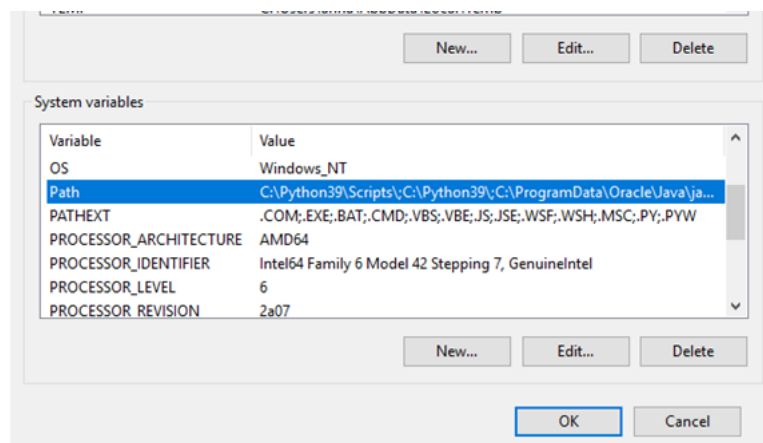


Step 9 : Press "NEW" in the dialog box that appears and copy and paste the same folder path of jdk-16.0.2/bin there.

Step 10 : Done. Finish. Khatam. Bye Bye.

Hey wait, let's check if JDK has been there in your computer or not.

## Post Setup of Environmental Variables.

Step 11 : Open CMD anywhere on your Computer.

Step 12 : Write this command.

```
javac --version
```

It should output *javac 16.0.2*. Then only, your Java 16 is working on your PC.

Step 13 : Write this command too.

```
java -version
```
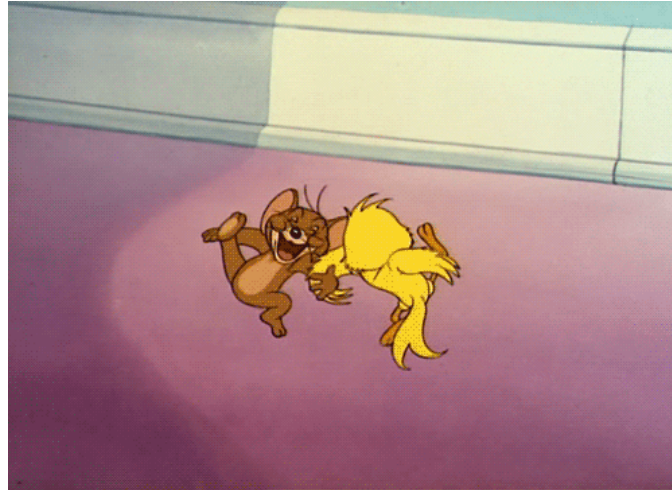
This should give you the answer as : -

> java version "16.0.2" 2021-07-20
> Java(TM) SE Runtime Environment (build 16.0.2+7-67)
> Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)

Yay 🎉🎉🎉 ...Finally you did it.



After finishing up setting the IDE, let's write our very first program in Java.

But prior to that, we must know 2 things.

## Comments

A good programmer's practice is to always describe and give details about his/her program. Imagine if you went to github to contribute and the README.md file was empty, how would you have contributed then, when you don't even know what the repository. Similarly, it is important for us to specify the purpose and use cases of our program. These information is not required by the Java Virtual Machine. Comments are simply ignored by the Compilation Unit of Java.

Comments are just for human understanding, they are not required by the Java Compiler and Java Virtual Machine.

JVM be like : - (Le Programmer)

Programmers be like : -



This is done by 3 ways: -

(i) Single Line Comments , (ii) Multi Line Comments and (iii) Documentation Comments.

1. Single Line Comments : Let's say that a single line is enough to briefly describe the purpose of your program. Then, its better to use Single Line Comments. It basically elaborates a single code statement in your program.

```
// This is a Single Line Comment.
```

**Single Line comments are denoted by the beginning of 2 forward slashes(*//*).**

2. Multi Line Comments : Sometimes, a single line is not enough to describe the purpose of the program you write. Let's say that you want to have the Problem Statement as the comment in your program. You cannot just keep on indenting the code and assigning a single line. Hence, if you require to elaborate your program, we use Multi Line Comments.

```
/* Hi All,
   this is a multiline comment
   this feels good if you have a lot of information to give to your client about your code
*/
```

Multi Line Comments are denoted by "/*" in the beginning, and it ends with "*/".

3. Documentation Comments : *These comments are written so that they are visible when we generate the documentation of our Program*. This might sound weird. You might think that why a program needs to have Documentation even. Its simple. Just like oracle has all the documentation for its whole API, its important to write documentation if you are working on a large scale project in a company so that whenever a newbie joins, he/she can relate the project company is working on and can contribute to it.

```
/* * Begins Documentation
   * First Line : Author Name
   * Second Line : Version Info
   * Third ... nth Line : Documentation Begins
*/
```

A documentation comment begins with "/*" and ends with "*/". Each line you write, begins with an asterisk mark (*).

Interview Question
What is an API Document?

Now, we are good to write our very first program in Java. Let's do it.

```
// This is a Hello World Program.
public class HelloWorld {
```

```
    public static void main(String[] args) {
        System.out.println("Hello World! This is me");
    }
 }
```

Whoa! that's damn simple program. Now its time to run our program so that we can see the output.

There are basically 3 steps of a program : -

1. **Write Good Comments and a sweet programs.**

2. **Compile and Debug them successfully.**

3. **And Run the Program to get the output.**

📝Write —> 🐛Debug —> ▶️Run

Our Program seems absolutely cute and pretty🥰. Now, Let's compile our program.

So as to compile a Java program, we first open the Command Prompt in that File Location, where we had saved our Program.

Once it opens, write this command: -

🛠️    javac HelloWorld.java

**javac** is a combination of 2 things : **java** and **c**. Java means Java and **c** means Compiler.

When combined together, this means Java Compiler. As mentioned earlier, Java programs are first compiled into bytecode and then, they are run by Java Virtual Machine.

Step 1 : Compile your program to check for any Syntax Violations by using the **javac** command.

Step 2 : Pass the Compiled Program to the Java Virtual Machine so as to run it successfully by using the command : -

> 🔧 java HelloWorld

The file "HelloWorld" has an extension of ".class" . This *class* file is nothing but the platform independent bytecode. This *.class* file is the reason for Java's Platform Independent Nature.

Now, what's the output?

```
Hello World! This is me
```

Now, let's derivate our program line by line.

```
public class HelloWorld {

}
```

*A **class** represents a binding unit.* **Everything in a Basic Java Program should be written within a <span style="color:red">class</span>.** *A class serves as a single unit through which we can access all the members inside it.*

Best example of class serving as a single unit of accessibility can be seen during the compilation of the program.

Since Java is Object-oriented, everything is done within a class. It is required to have at least one class. Writing a class is pretty simple. Here is the syntax : -

```
class Class_Name {
    //Statements goes here.
}
```

We use the <span style="color:red">class</span> keyword to define a class. "Class_Name" denotes the name of the class. As in the first program, the name of the class is *HelloWorld*.

But wait, what are { } ? Well, these things are called Blocks.

**Blocks defines a scope of accessibility in Java within which other members can access each other.**

We use this block so that *all the members of the class can access each other.* Since, *Java is object oriented and everything must be in the scope of a class, hence we use {} to denote the scope of the class.*

Wait a second... You might be thinking why are we using **public** in front of the class. JVM is also a program that is written by people who made Java. So as to ensure that Java Virtual Machine can access your program from anywhere, we use the **public** keyword. For now, the use of *public* keyword is enough. When we advance to higher concepts, we would understand its broader aspect.

Then comes a method...The mighty *main()* method.

```
public static void main(String[] args) {
    // Method code goes here.
}
```

main( ) method acts as an Entry to your Program. This is the method which JVM searches for when you run your Java Program. Just like your house has only one main door of entry, A Java Program also has only 1 place of Entry. This place is *main()* method.

This method has **public** access. This basically means that it can be accessed anywhere throughout the class and is also accessible outside the file from the object of the class.

So as to print something on the Screen in your computer, we use a *println()* method which is present inside the **PrintStream** class. You can remember the print syntax in somewhat the way given below: -

*System.out* —> Access your system and choose to output something.
*println( )* —> It does a task of Printing something to your output window.

The word "println" is made of two parts : -

1. print —> Print something

2. ln —> Line Break.

In combination, it forms *println* which means that **it would print something to the output window and would change the line of cursor and set it to the next line**.

But wait, what's this strange *static* stuff???



Confusing hai kitna yaar....Alternative bata.

Its easy to understand. **static** keyword is used in a Java Program to denote that a particular member is accessible through its class name itself in other files. Let's say that you have another class where you want to call **HelloWorld's** *main method*. Let's code it at once.

So you must have rightly thought, we are going to have another empty java file and would write the code.

```
// Program to show accessibility of main() method in another files.
public class File2 {
  public static void main(String[] args) {
      HelloWorld.main(args);  // This simply denotes that we can access main() method fro
  }
}
```

🔧 HelloWorld.main(args)

This states the Java Virtual Machine that **Krish has a class *HelloWorld* where there is a *main( )* method which accepts a *String type array* as its argument.**

Simply, we pass *args* as the parameter of *HelloWorld's main() method* because *args* itself is a String type Array in *File2* class. This is one of the reason why we use **static** keyword in the method prototype( prototype means the method header which

comprises of keywords like public, private, protected, static, void, etc. with the method name and the formal parameter list ) of *main() method*.

When you try compiling the **File2** using the commands *javac* and *java*, JVM searches for the main( ) method with public access and defined with the keyword static and void having a String array type parameter.

The formal parameter list is the list of combination of datatypes and variables. main( ) method has a String type array in its formal parameter list. This *args* is accessible everywhere in the main( ) method. The start of parentheses ( ) just immediately after the method name denotes the formal parameter list.

Now, the use of *void* is what? It specifies that a method does not return anything. Basically, a method is a separate piece of code which may or may not accept any argument and may or may not return a value, but it performs a specific task. In the above code example, main( ) method has a String type array as its parameter but does not return anything. We know that main() method should not return anything but who would inform this info to JVM. Here, *void* is used as the return type of a method which does not return anything.

> Why is it necessary to write the same method header or prototype in a Java Program.?
>
> **It is because JVM looks for a main() method with a String type array as its parameter and is defined with atleast *public* and *static* keywords having its return type as *void*.**

## More Information about Basic Knowledge in Introduction to Java.

Now that we have decomposed our First Program, let's get deeper into concepts. The very first thing we are going to understand is some basic knowledge of variables, datatypes and stuff.

**Now, let's say that I have a Girlfriend(unfortunately, this isn't true). Now if I ask what is the gender of my Girl-Friend, then surely the answer, no doubt, is Girl**. So, *Girl* is a type. Similarly, when you want to store some values in Java, you need

to specify the type of Data that you are going to store in a variable. This proves the robustness of Java in yet another way. This is the reason why you need to specify the type of Data before storing it. Hence Java is strongly typed language.

But now, what is a Variable?

Well, variables are just like containers which store a value, which when we want, can be changed when the program runs.

Let's see now, how we can store some value in a variable and print it.

```
int firstVariable = 10;
System.out.println(firstVariable);
```

As you might have rightly pointed out, *int* is a datatype. Datatypes are the types of Data a Variable can store. So, what's the container which would store the value **10**. Its the *firstVariable*.

**Datatype : int**

**Variable Name : firstVariable**

**Value stored in the variable : 10**

But how to remember *int* and other datatypes?? It comes by practice. Howsoever, there are only 8 primitive types only, and these 8 types are enough to represent any basic level type.

Basically, *int* means that a value that you are going to store in the variable *firstVariable* can range between 2 * 10^-31 to 2 * 10^31 - 1. ***int* is an abbreviation of *Integer*.**

Syntax of using a variable: -

```
datatype variableName = value ;
```

**A semicolon(;) is necessary after all executable statements. If applied after a executable statement, it means that the statement would terminate and the Compiler would then check for the next line.**


## Let's Begin The Menti: -