Introduction to JAVA

Meaning and definition of JAVA

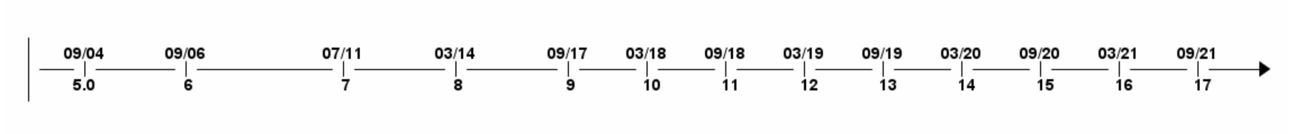
Java is a general-purpose, multi-platform, object-oriented, and network-centric language. It is among the most used programming language. Java is also used as a computing platform.

It is considered as one of the fast, secure, and reliable programming languages preferred by most organizations to build their projects.

History of JAVA

- The JAVA language was initially called OAK.
- Originally, it was developed for handling portable devices and set-top boxes. Oak was a massive failure.
- In 1995, Sun changed the name to “Java” and modified the language to take advantage of the burgeoning www (World Wide Web) development business.
- Later, in 2009, Oracle Corporation acquired Sun Microsystems and took ownership of three key Sun software assets: Java, MySQL, and Solaris.

The first publicly available version of Java (Java 1.0) was released in 1995 and over the time new enhanced version of the Java have been released. The current stable version of the Java is Java-17 which was recently released in September 2021.



In order to know about the JAVA in more detail we have to be familiar with the JAVA platform which direct us for overall development and operations of the application developed using this language.

Features of JAVA

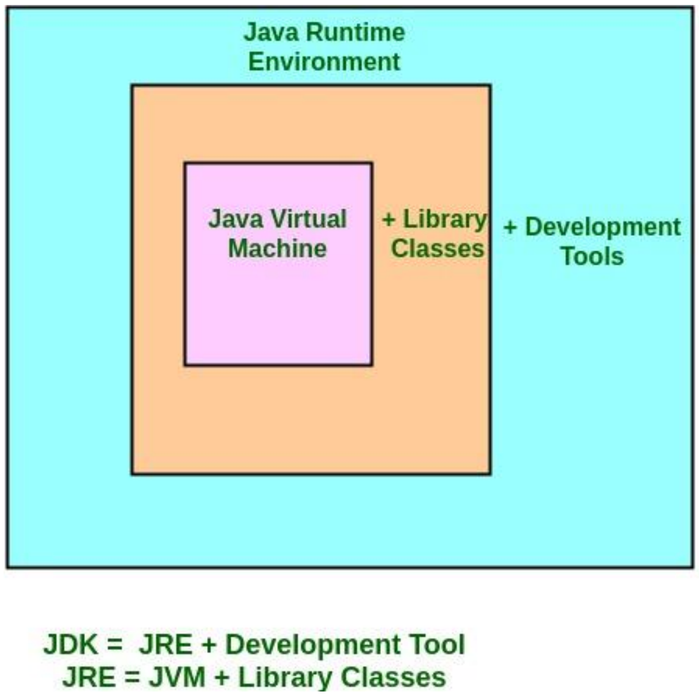
- It is one of the easy-to-use programming languages to learn.
- Write code once and run it on almost any computing platform.
- Java is platform independent. Some programs developed in one machine can be executed in another machine.
- It is designed for building object-oriented applications.
- It is a multithreaded language with automatic memory management.
- It is created for the distributed environment of the Internet.
- Facilitates distributed computing as its network centric.

JAVA Platform

Java Platform is a collection of programs that help programmers to develop and run Java programming applications efficiently. It includes an execution engine, a compiler, and a set of libraries in it. It is independent of any particular OS which makes Java programming language a platform-independent language.

The platform consists of following components;

- Java language
- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Java Compiler
- Java Virtual Machine (JVM)



Now let us look into each of components in detail as;

Java language

Java is a programming language that the Java platform uses. Java is an object-oriented programming language whose syntax is derived from C and OOPS features are derived from C++. It has its syntax, rules, format, and programming paradigm.

Java Compiler

This is a compiler for Java programming language and its function is to generate Java class files from the Java source code. Java class file contains a platform-independent Java byte code.

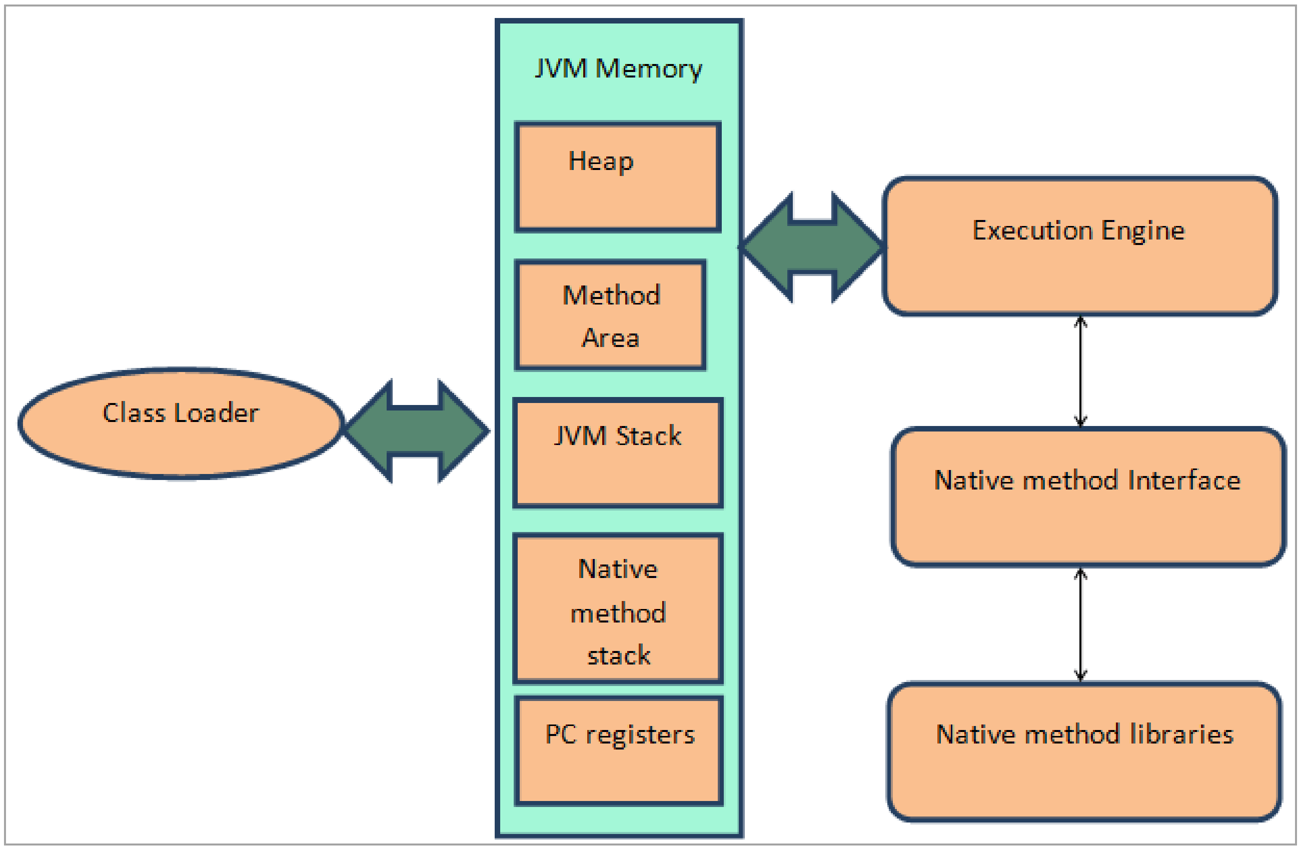
After generating class files, JVM loads these class files and either interprets the byte code or compiles it to machine code using **Just-in-time (JIT)** compiler.

Java Virtual Machine

JVM is the center of Java programming language and Java platform.

The JVM converts the byte code into machine-specific code (also known as object code in other programmer languages).

JVM provides the functionality of garbage collection, memory management, security, etc.



JVM is platform-independent, and we can customize its functionality using a Virtual interface it provides which is not machine-dependent and is also independent of the operating system.

This platform-independence of JVM allows us to create Java programs on one machine and execute them on another machine **(WORA – Write – Once – Run – Anywhere)**.

Note: JIT or Just-in-time compiler is a part of the Java Virtual Machine (JVM). It interprets a part of the Byte Code that has similar functionality at the same time. In other words, it converts the byte code into native machine code at the same programming level. This is the reason for which Java is compiled as well as an interpreted language.

Java Runtime Environment

JRE is the runtime environment that is required to execute Java programs and applications. JRE consists of Java Virtual Machine (JVM) and binaries and other classes to successfully execute Java programs.

JRE is a subset of JDK and doesn't contain any development tools such as Java compiler, debugger, etc. Hence if your Java applications are developed elsewhere, and you need to only execute them on your machine, then you can install JRE separately. You do need to install JDK for this.

You need a JRE installed on your machine as it's the minimum requirement to run Java programs on your machine.

The JRE includes the following components.

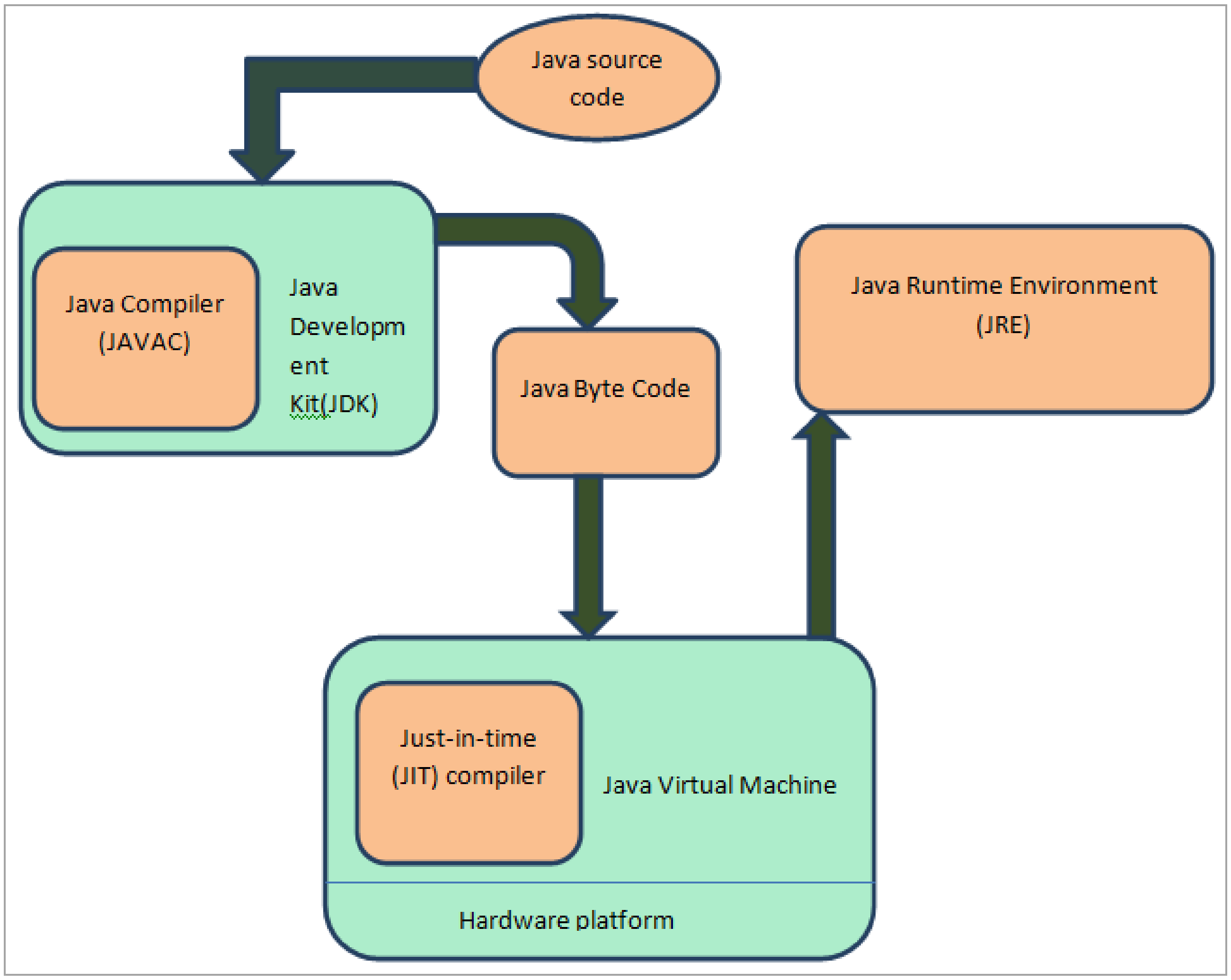
- Code libraries, property settings, and resource files: These include files like charsets.jar, rt.jar, etc.
- DLL files: Used by Java hotspot client virtual machine and server virtual machine.
- Java extension files: For Example, files related to locale specification like localedata.jar
- Files required for security management. For Example, java.policy, java.security
- Applet support classes.
- True Type font files: Usually required by the platform.

To execute any application/program written in Java, you need JRE installed on your system. JRE is platform dependent. This means you need to download and install JRE that's compatible with your O.S. and architecture.

Java Development Kit

This is the core component of any Java environment. JDK contains JRE (Java Runtime Environment) along with Java compiler, Java debugger, and other core classes. JDK is used for Java development as it provides the entire executable and binaries as well as tools required to compile, debug a Java program.

JDK is a platform-specific software and thus we will have separate JDK installers for each Operating system.



JDK contains the following components:

- **jConsole**: This is a Java monitoring and management Console.
- **jar**: This is the archiver. This tool is used to package related class libraries into a single Jar file as well as to manage Jar files.
- **jarSigner**: This tool is used for jar signing and verifying.
- **javap:** This is a tool for class file disassembler.
- **javaws**: Java web start launcher for JNLP applications.
- **jhat**: Java heap analysis tool.
- **jrunscript**: Java command-line script shell.
- **jstack**: Utility used to print stack traces for Java threads.
- **Javadoc**: This automatically generates documentation from the source code comments.
- **appletviewer**: Used for applet execution and debugging without a web browser.
- **apt**: Annotation processing tool.
- **extCheck**: Utility used to check jar file conflicts.

- **keytool**: Using this utility you can manipulate Keystore.
- **policytool**: This is a policy creation and management tool.
- **xjc**: This is a part of the XML binding (JAXB) API that accepts XML schema and generates Java classes.
- **And** many more!!!!

Thus you can see that the components of JDK are the ones that are required by us from a development point of view.

As shown in the diagram, the Java source code is converted to byte codes by Java compiler which is a part of JDK. Then this byte code is passed on to JVM and from there it goes to JRE where the code is executed, and program's output is seen.

Now after learning about both JRE and JDK you might think that one is the replacement for the other. But it is like that, JRE and JDK are totally different from one other.

No	JRE	JDK
1	JRE stands for Java Runtime Environment.	JDK stands for Java Development Kit.
2	Mostly used for the execution of Java programs.	JDK is used by developers for developing Java programs.
3	Contains Java Virtual machine (JVM) bundled inside it.	Doesn't have JVM.
4	JRE doesn't have Java compiler so cannot compile programs.	JDK has javac compiler and is responsible for compiling programs.
5	Contains java class library, the java command, and other infrastructure.	JDK contains tools like Javadoc and archiver that are used to develop Java applications.
6	JRE can be installed as a standalone program.	JDK is a separate installer and comes bundled with JRE.
7	Takes the compiled/interpreted Java program as input and generates output.	Compiled Java source program and generates a class file which is then given to JVM.

Till now we discussed about Java Development Kit (JDK) that comes bundled with Java Runtime Environment (JRE) and Java compiler.

JRE, in turn, contains Java Virtual Machine (JVM) which is responsible for converting byte code generated by Java compiler into machine-specific code.

All these including Java language are components of a parent entity called Java platform which is an environment that helps us to run a Java application. Apart from we also discussed the detailed architecture and working of JVM as well as the execution of a JAVA program.

Now we will look into the practical implementation of what we learnt and discussed till now.

Setting up the Java Platform and working with the first Java file

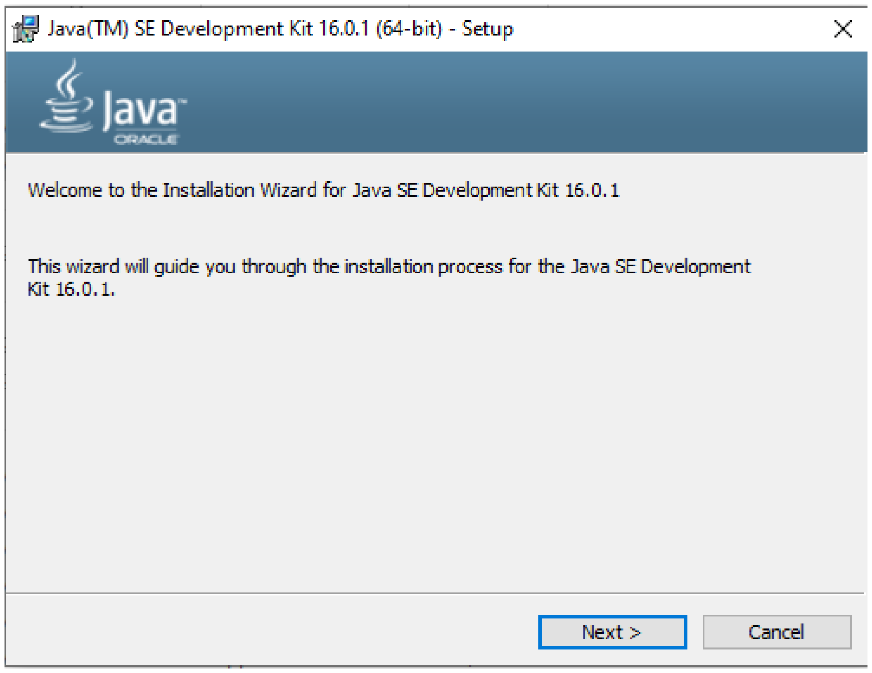
Setting up JDK

As we have already studied earlier that to begin working with the Java we have to first set up the Java Platform. For that we will be looking into the setting up the JDK.

To download the JDK of your choice go through the link below;

https://www.oracle.com/java/technologies/downloads/

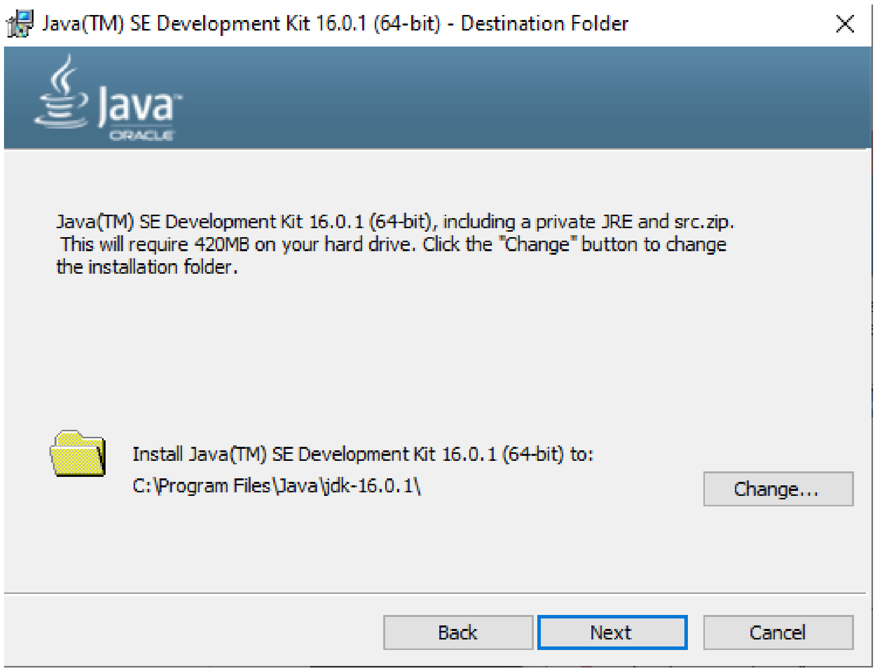
After downloading open up the executable file and follow the steps as;



Java(TM) SE Development Kit 16.0.1 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 16.0.1

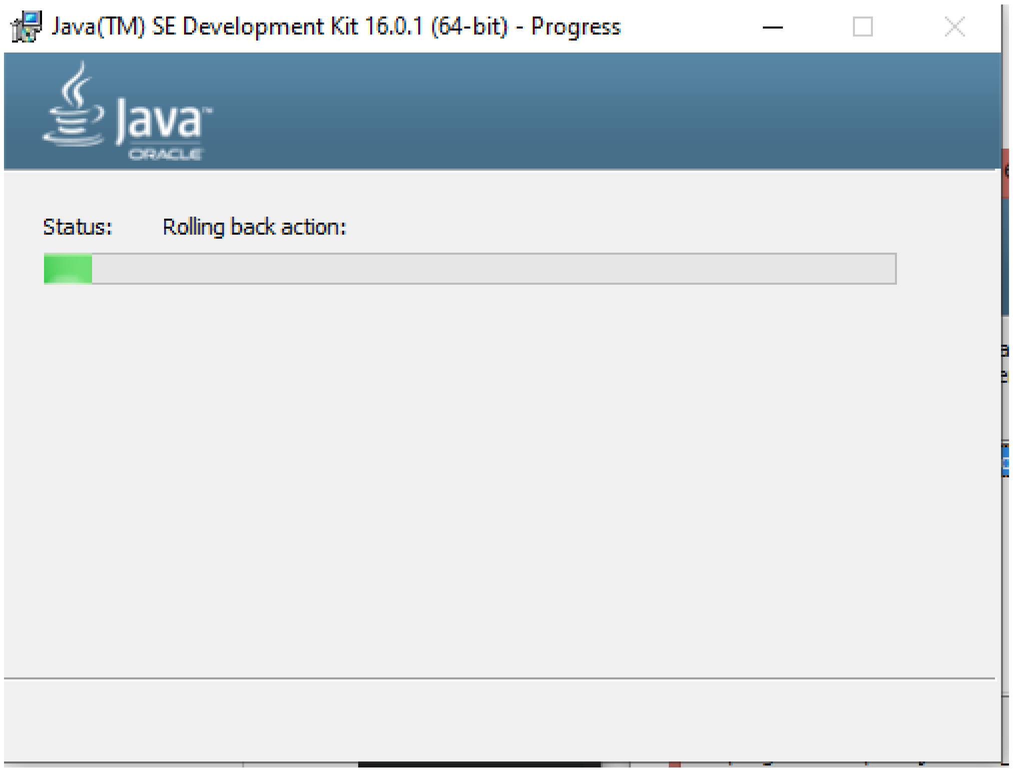
This wizard will guide you through the installation process for the Java SE Development Kit 16.0.1.



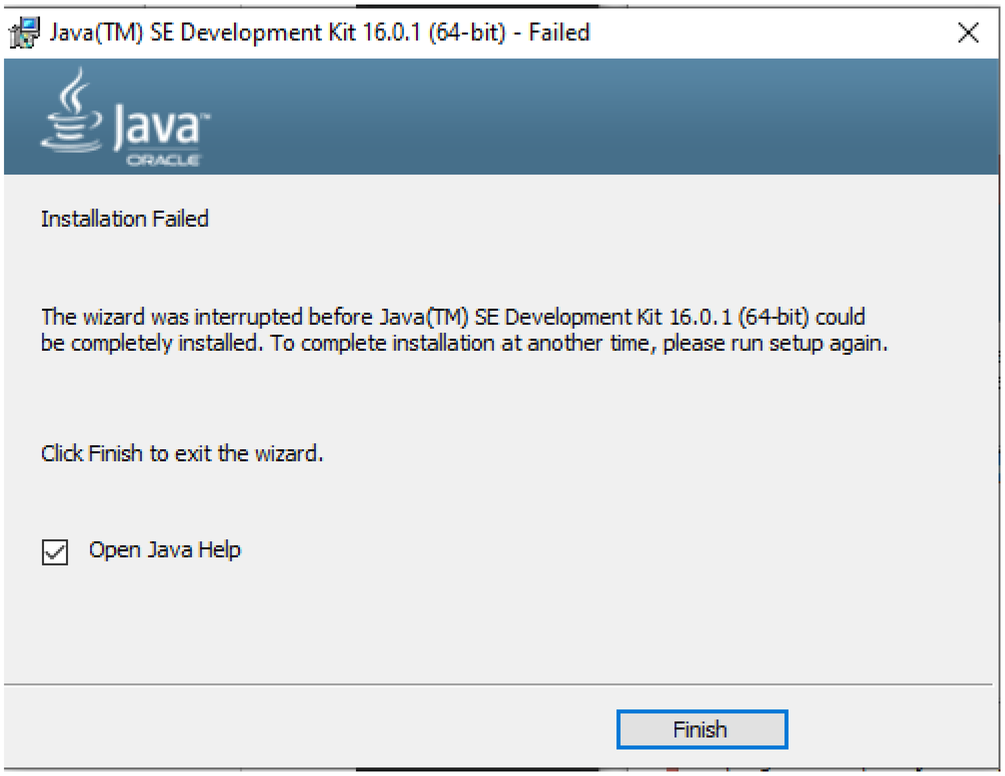
Java(TM) SE Development Kit 16.0.1 (64-bit) - Destination Folder

Java(TM) SE Development Kit 16.0.1 (64-bit), including a private JRE and src.zip. This will require 420MB on your hard drive. Click the "Change" button to change the installation folder.

Install Java(TM) SE Development Kit 16.0.1 (64-bit) to:
C:\Program Files\Java\jdk-16.0.1\


Java(TM) SE Development Kit 16.0.1 (64-bit) - Progress

Status: Rolling back action:


Java(TM) SE Development Kit 16.0.1 (64-bit) - Failed

Installation Failed

The wizard was interrupted before Java(TM) SE Development Kit 16.0.1 (64-bit) could be completely installed. To complete installation at another time, please run setup again.

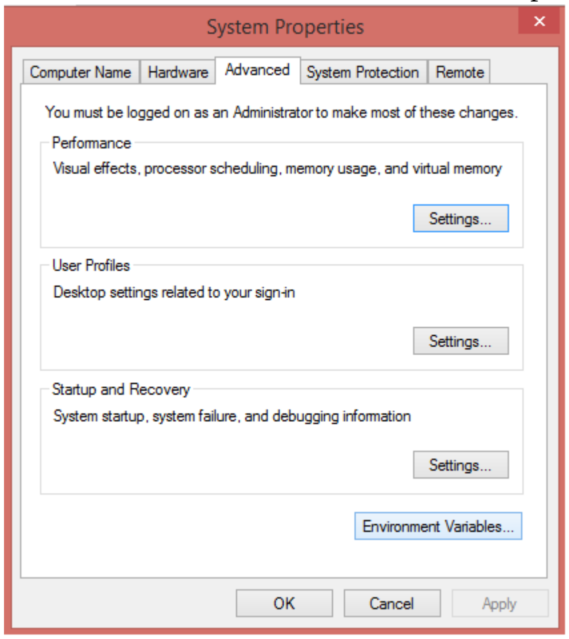
Click Finish to exit the wizard.

☑ Open Java Help

Here, we have successfully installed JDK.

Now we have to verify whether the path to the JDK has been added in the environment variables. If not then we have to add it to the environment variables via the following steps as;

1. Search for the “environment variables” in your computer by clicking the “Start” icon. After that go the “Edit the system environment variables” tab and follow the steps as;



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

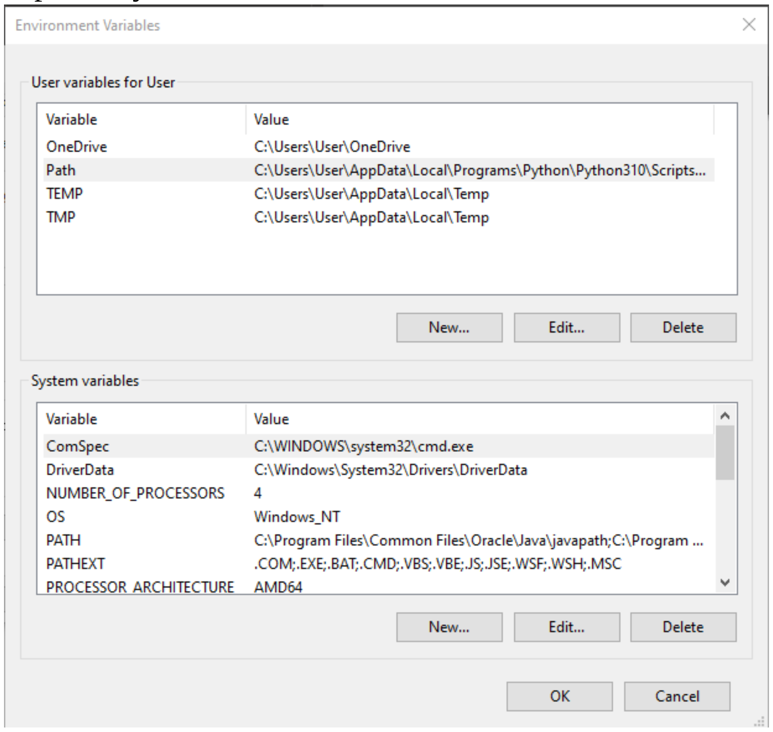
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

2. Click on the “Environment Variables” and go the user variables and select the “Path” and click edit button to add new path of your installed JDK.



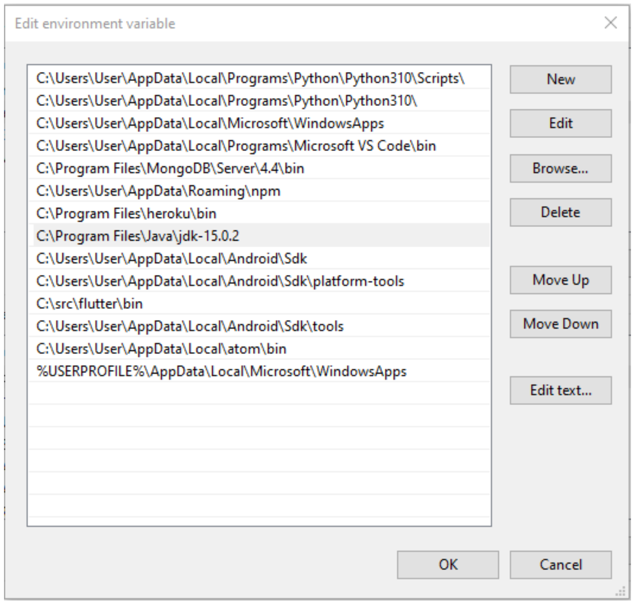
User variables for User

Variable	Value
OneDrive	C:\Users\User\OneDrive
Path	C:\Users\User\AppData\Local\Programs\Python\Python310\Scripts...
TEMP	C:\Users\User\AppData\Local\Temp
TMP	C:\Users\User\AppData\Local\Temp

System variables

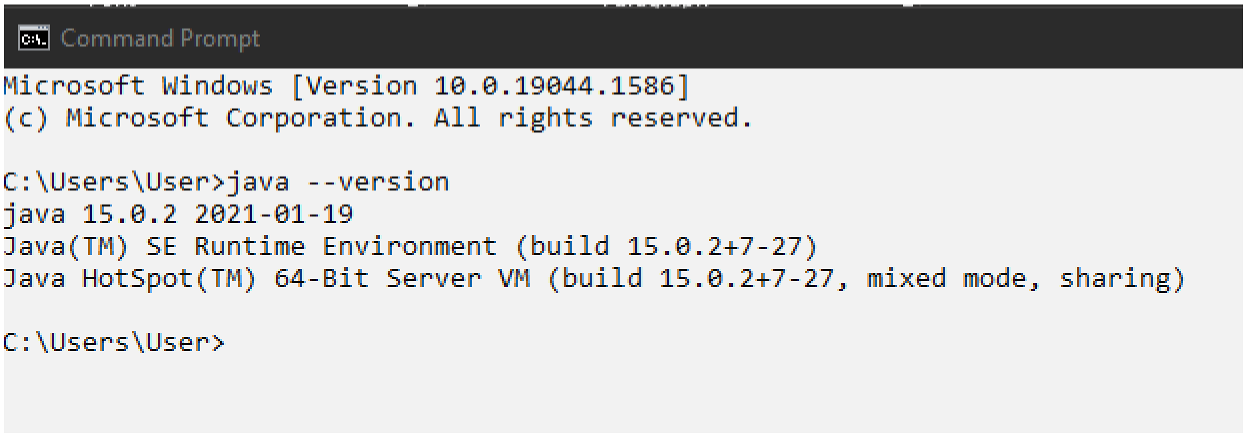
Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
PATH	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program ...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

3. Here, to add the path of the JDK go the file explorer and search for the following path and add it as;



```
C:\Users\User\AppData\Local\Programs\Python\Python310\Scripts\
C:\Users\User\AppData\Local\Programs\Python\Python310\
C:\Users\User\AppData\Local\Microsoft\WindowsApps
C:\Users\User\AppData\Local\Programs\Microsoft VS Code\bin
C:\Program Files\MongoDB\Server\4.4\bin
C:\Users\User\AppData\Roaming\npm
C:\Program Files\heroku\bin
C:\Program Files\Java\jdk-15.0.2
C:\Users\User\AppData\Local\Android\Sdk
C:\Users\User\AppData\Local\Android\Sdk\platform-tools
C:\src\flutter\bin
C:\Users\User\AppData\Local\Android\Sdk\tools
C:\Users\User\AppData\Local\atom\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
```

The version of the installed JDK might be different. Look on to it.
Now to verify the installation of the JDK open up the command prompt and type "java –version" command to see the version of the installed JDK.



```
Microsoft Windows [Version 10.0.19044.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>java --version
java 15.0.2 2021-01-19
Java(TM) SE Runtime Environment (build 15.0.2+7-27)
Java HotSpot(TM) 64-Bit Server VM (build 15.0.2+7-27, mixed mode, sharing)

C:\Users\User>
```

Thus the JDK has been installed in our machine. Now we can easily work with the Java programs and initiate the development.

Setting up IDE

Now, we need to configure IDEs like NetBeans, Eclipse, IntelliJ IDEA, Visual Studio Code etc. in order to run and develop Java Program.

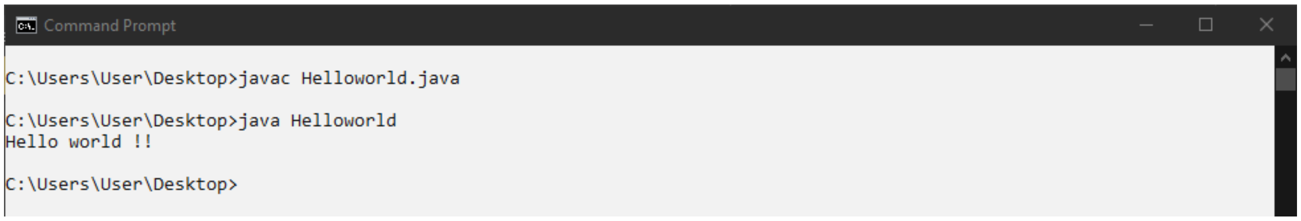
Writing the first Java program

Now let us create a new file in the notepad as “Helloworld.java” and save it in the desktop or the root directory of your machine.

After this add following lines of code in the file;

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

Then save the file and open it in the terminal or command prompt. Now write the following commands as;



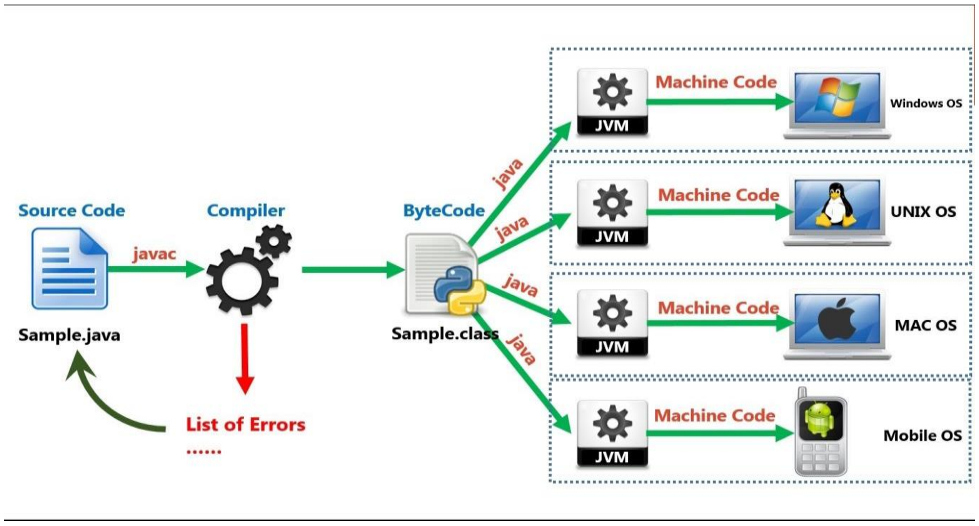
```
C:\Users\User\Desktop>javac Helloworld.java

C:\Users\User\Desktop>java Helloworld
Hello world !!

C:\Users\User\Desktop>
```

- To compile the example:
  - **javac Helloworld.java**
    - Notice the .java file extension is needed.
    - This will result in a file named *Helloworld.class* being created.
- To run the example:
  - **java Helloworld**
    - Notice there is no file extension here.
    - The *java* command assumes the extension is .class.

Behind the scenes



Parameters used in First Java Program

Let's see what’s 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. The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.
- **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 look into it in the future classes.

Task to do:

Create a new project in the IntelliJ IDEA and write the Helloworld.java program.