ENGINEERING TUTOR

Visit Our Website: www.engineeringtutor.in

Email: info.engineeringtutor@gmail.com

Address:- B-499 Main Raod Besid Sbi ATM,Jai Hind Chowk,Royal Plaza, Smriti Nagar,Bhilai

CALL: 9644304435, 7987092522, 07883553367

Training Features

* 2 Hrs Training in a Days (1 hr Theory+ 1 hr Practical)
* Real Time Project
* Complete Study Material
* Weekly Tests assignment
* per Days Special Interview Classes
* Sunday holiday as per requirement

Training Details:

* **Course Duration:** 21 days

* **Course Hours:** 2 Hours/ Day / Batch

* **Name of Course Coordinator:** Satish Kumar Dhiwar

* **Name of Tutor:** Satish Kumar Dhiwar

* **Tutor Contact No:** 9644304435, 7987092522

* **Proposed Fee: Rs. 12000/- Per Subject (Online Registration and Certificate)**

* **Coordinator/Tutor Email:** info.engineeringtutor@gmail.com

* **No. Of students should be registered in the course:** 30 Students At least.

Topic	Timing	Trainer
Java Vocational Training		Mr. Satish Kumar Dhiwar

Java is the most popular programming language & is the language of choice for Android programming. This course is taught in practical GOAL oriented way. It is recommended you practice the code assignments given after each core java tutorial.

What is Java?

Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies.

It is a computing platform for application development.

Java is fast, secure and reliable therefore it is widely used for developing Java applications in laptops, data centers, game consoles, scientific supercomputers, cell phones, etc

What is Java Platform?

Java Platform is a collection of programs that helps programmers to efficiently develop and run Java applications.

It includes an **execution engine**, a **compiler** and a set of **libraries** in it.

It is a set of computer software and specifications.

James Gosling developed the Java platform at Sun Microsystems and it was later acquired by the Oracle Corporation.

This video introduces the **Java platform**, and explains why Java is a platform as well as a programming language.

What is PC?

A computer is an electronic device capable of performing computations, and we all know that it is composed of a monitor, keyboard, mouse, and memory to store information. But the most important component of the computer is a PROCESSOR. Which does all thinking of computer, but the question is how the computer does this thinking? How does it understand text, images, videos, etc.?

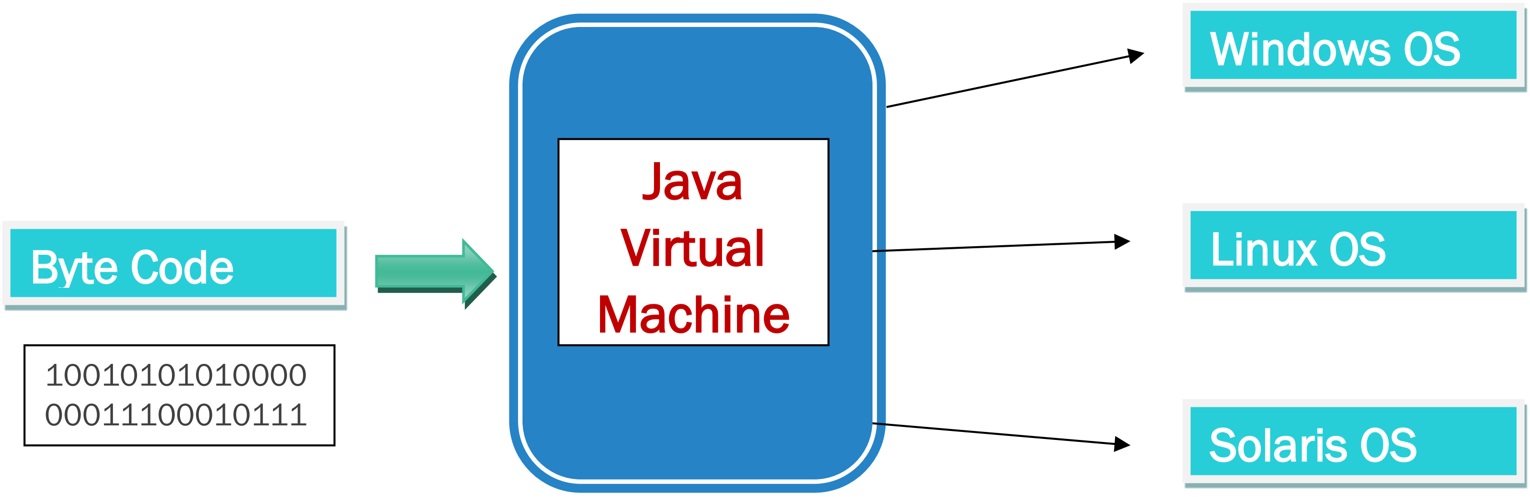
What is Assembly Language?

The computer is an electronic Machine, and it can only understand electronic signals or binary signals. For example, the 5-volt electronic signal may represent binary number 1 while 0 volts may represent binary number 0. So your PC is continuously bombarded with these signals.

> "**Assembly Language** **is the most elementary form of software development languages."**

How Java Virtual Machine works?

By using **Java Virtual Machine**, this problem can be solved. But how it works on different processors and O.S. Let's understand this process step by step.



Step 1) The code to display addition of two numbers is System.out.println (1+2), and saved as .java file.

Step 2) Using the java compiler the code is converted into an intermediate code called the **byte code.** The output is a **.class file.**

Step 3) this code is not understood by any platform, but only a virtual platform called the **Java Virtual Machine.**

Step 4) This Virtual Machine resides in the RAM of your operating system. When the Virtual Machine is fed with this byte code, it identifies the platform it is working on and converts the byte code into the native machine code.

In fact, while working on your PC or browsing the web whenever you see either of these icons be assured the java virtual machine is loaded into your RAM.

But what makes java lucrative is that code once compiled can run not only on all PC platforms but also mobiles or other electronic gadgets supporting java.

Hence,

"Java is a programming language as well as a Platform"

How is Java Platform Independent?

Like C compiler, Java compiler does not produce native executable code for a particular machine. Instead, Java produces a unique format called bytecode.

It executes according to the rules laid out in the virtual machine specification.

Bytecode is understandable to any JVM installed on any OS.

In short, the java source code can run on all operating systems.

How is Java Platform Independent?

Like C compiler, Java compiler does not produce native executable code for a particular machine. Instead, Java produces a unique format called byte code.

It executes according to the rules laid out in the virtual machine specification.

Bytecode is understandable to any JVM installed on any OS.

In short, the java source code can run on all operating systems.

What is JVM?

Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications.

It converts Java byte code into machines language. **JVM is a part of Java Run Environment (JRE)**.

In other programming languages, the compiler produces machine code for a particular system.

However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

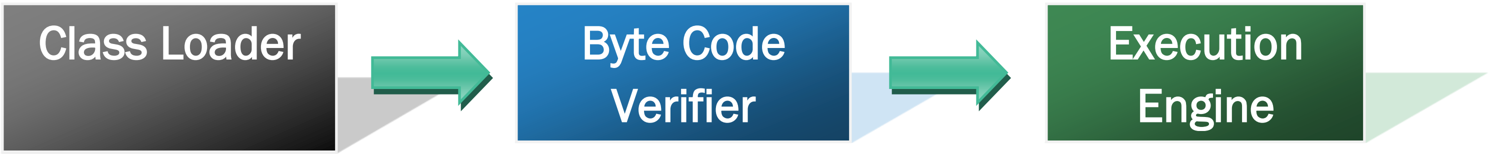
Here is How JVM works

First, Java code is complied into bytecode.

This byte code gets interpreted on different machines

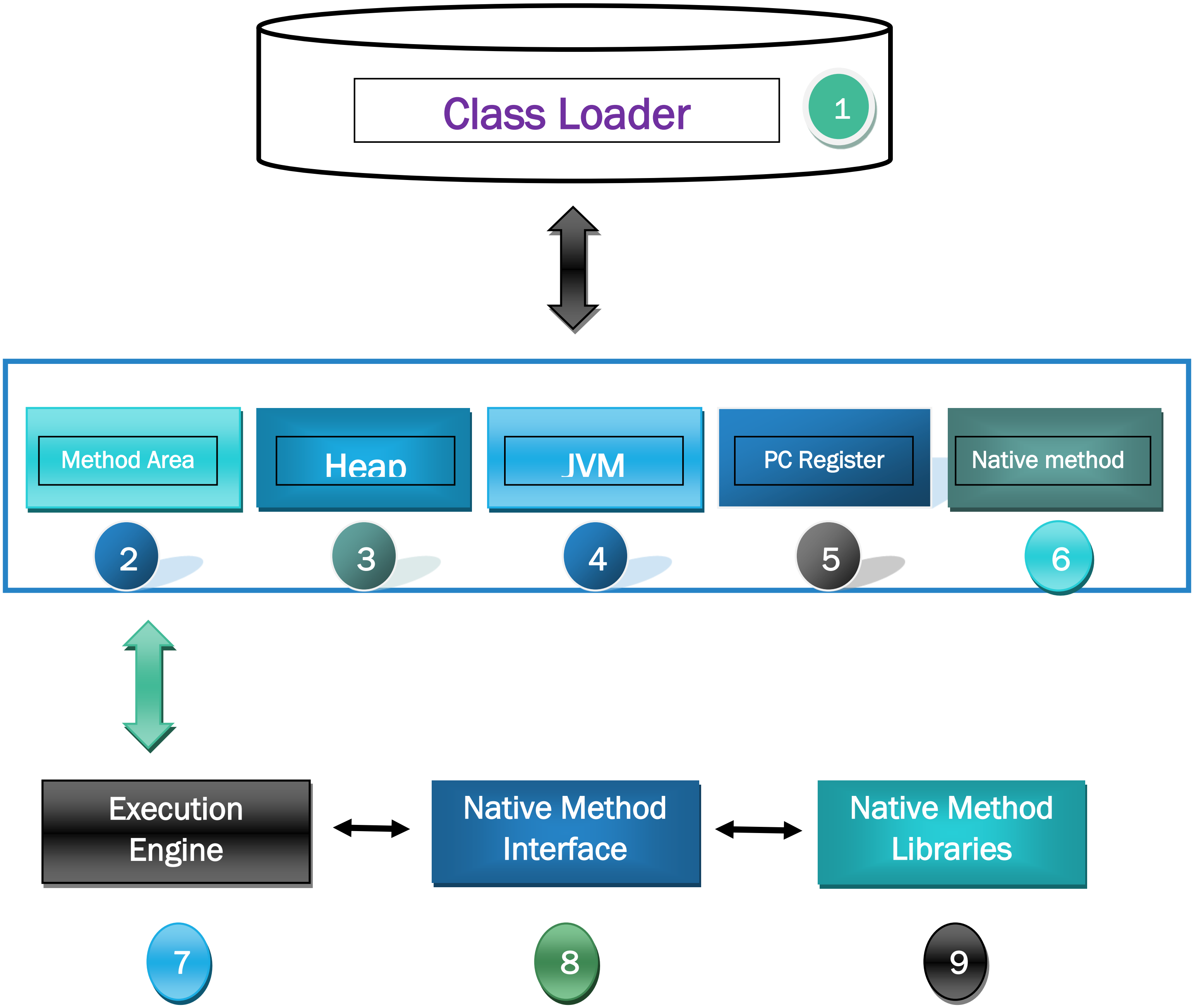
Between host system and Java source, Bytecode is an intermediary language.

JVM is responsible for allocating memory space.



Jvm Architecture:

Let's understand the Architecture of JVM. It contains class loader, memory area, execution engine etc.



1) ClassLoader

The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

2) Method Area

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

3) Heap

All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

4) JVM language Stacks

Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

5) PC Registers

PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

6) Native Method Stacks

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

7) Execution Engine

It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

8) Native Method interface

The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

9) Native Method Libraries

Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

Software Code Compilation & Execution process

In order to write and execute a software program, you need the following

1) Editor – To type your program into, a notepad could be used for this

2) Compiler – To convert your high language program into native machine code

3) Linker – To combine different program files reference in your main program together.

4) Loader – To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.

5) Execution – Actual execution of the code which is handled by your OS & processor.

With this background, refer the following video & learn the working and architecture of the Java Virtual Machine.

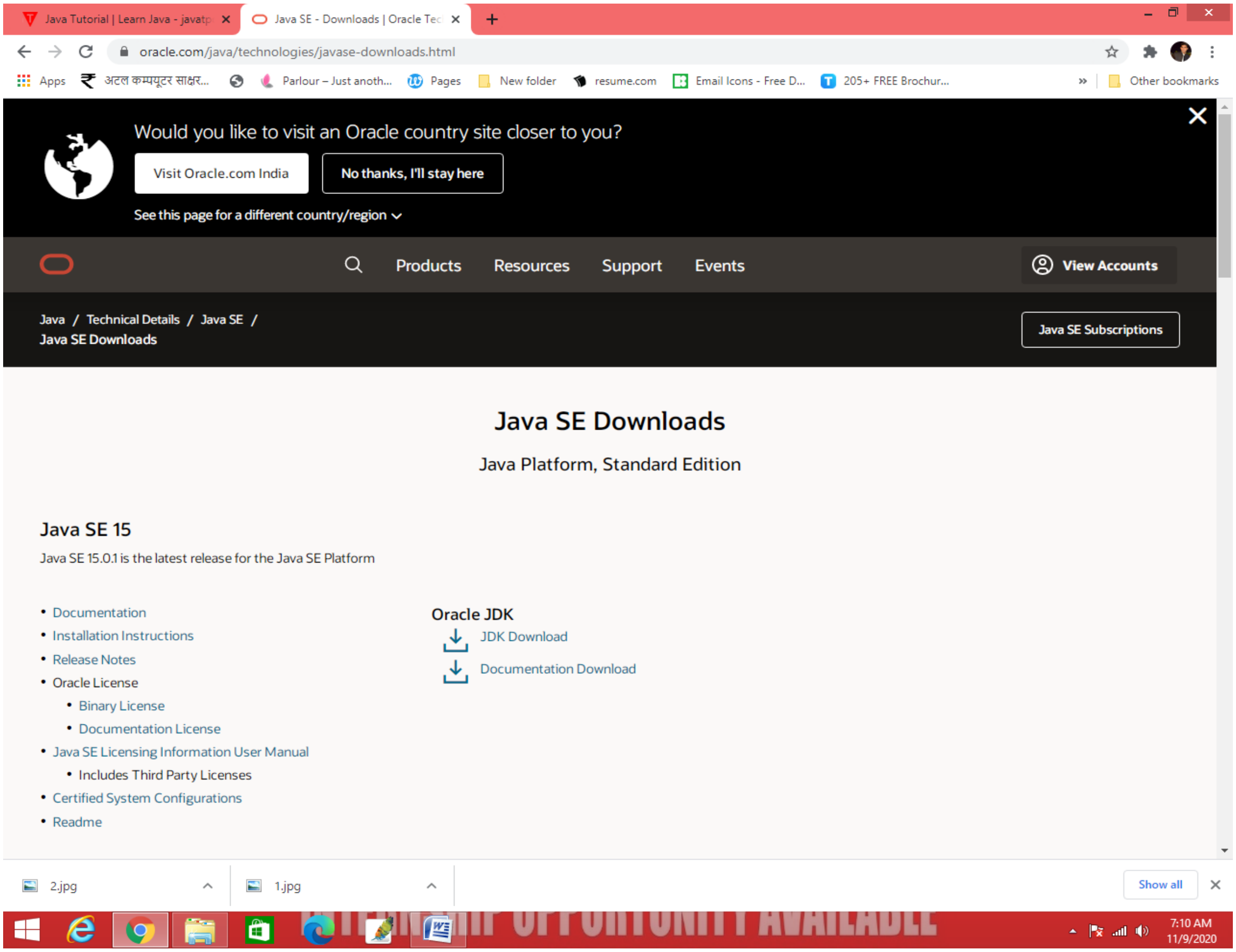
Link :=> https://www.guru99.com/java-virtual-machine-jvm.html

How to Download & Install Java JDK 8 in Windows

This Java Development Kit(JDK) allows you to code and run Java programs. It's possible that you install multiple JDK versions on the same PC. But Its recommended that you install only latest version.

Following are steps to install Java in Windows

Step 1) Go to link. Click on Download JDK. For java latest version



Would you like to visit an Oracle country site closer to you?

Java SE Downloads

Java Platform, Standard Edition

Java SE 15

Java SE 15.0.1 is the latest release for the Java SE Platform

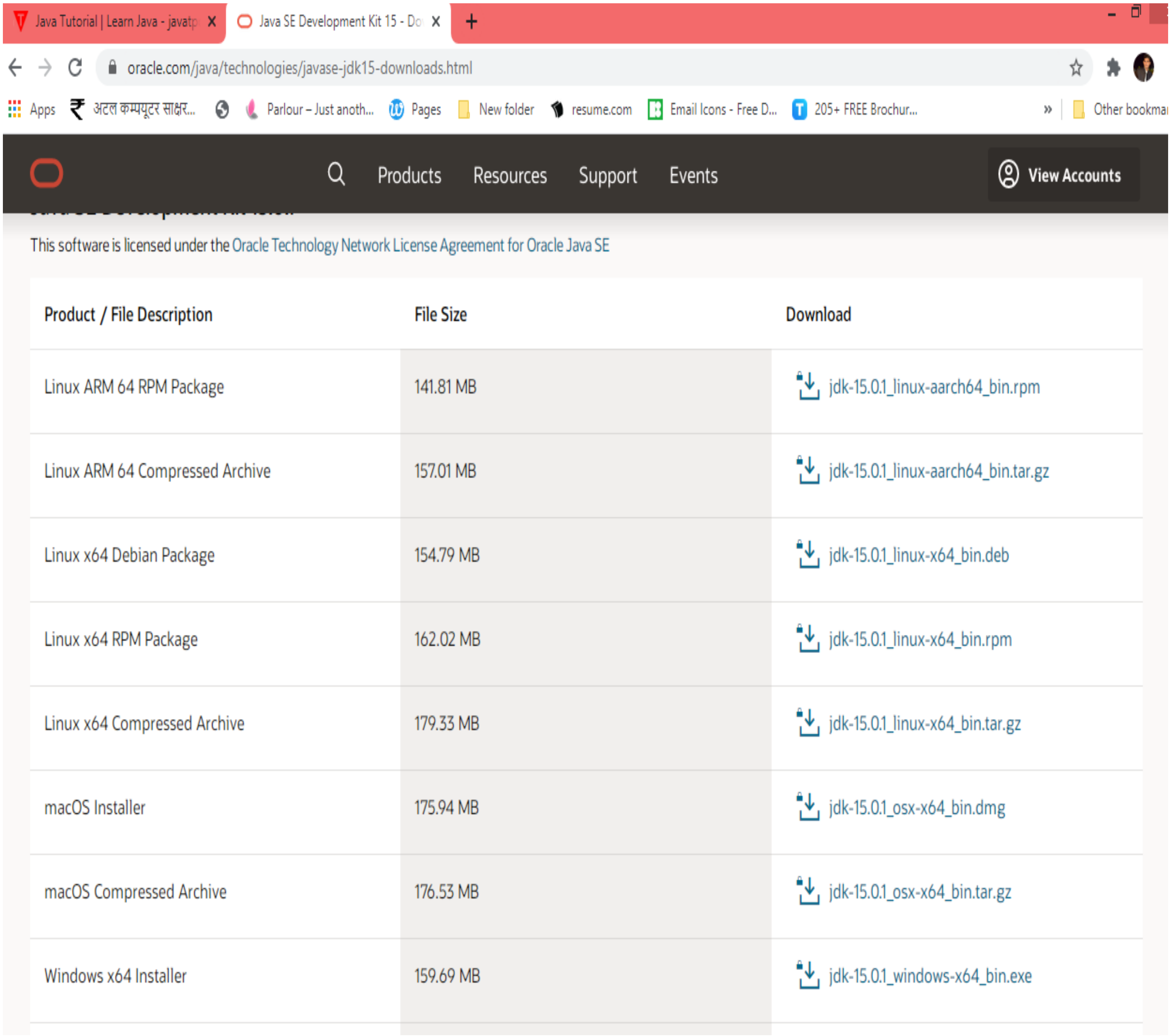
- Documentation
- Installation Instructions
- Release Notes
- Oracle License
  - Binary License
  - Documentation License
- Java SE Licensing Information User Manual
  - Includes Third Party Licenses
- Certified System Configurations
- Readme

Oracle JDK

- JDK Download
- Documentation Download

Step 2) Next steps you need to follow,

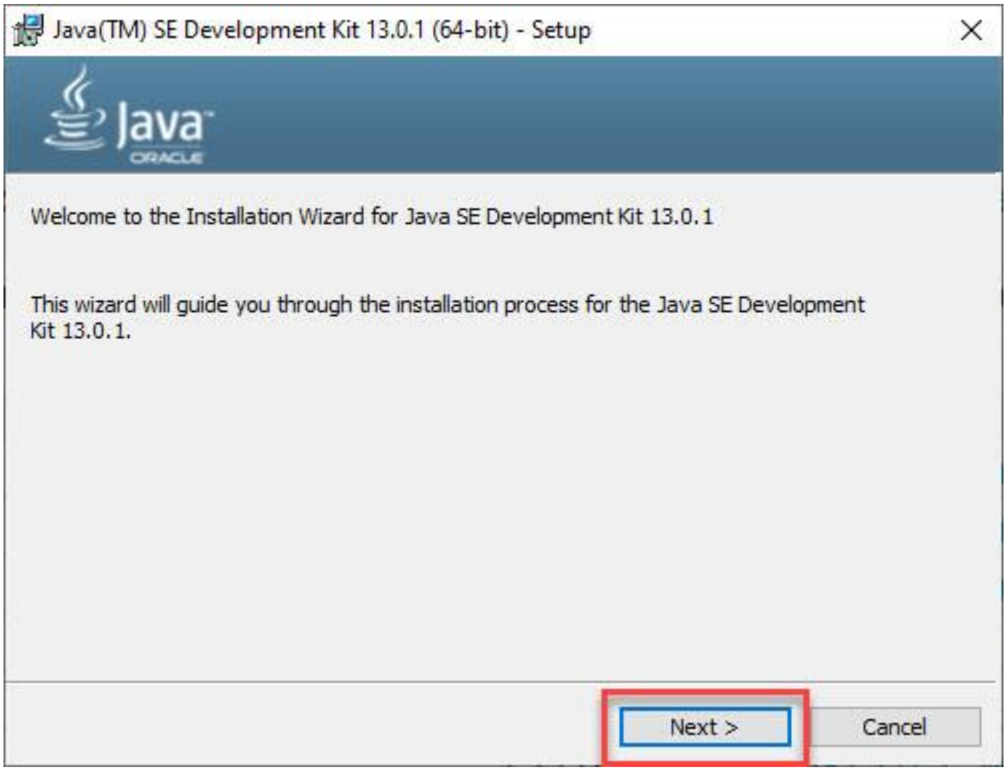
1. Accept License Agreement
2. Download latest Java JDK for your version (32 or 64 bit) of java for Windows



This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux ARM 64 RPM Package	141.81 MB	jdk-15.0.1_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	157.01 MB	jdk-15.0.1_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	154.79 MB	jdk-15.0.1_linux-x64_bin.deb
Linux x64 RPM Package	162.02 MB	jdk-15.0.1_linux-x64_bin.rpm
Linux x64 Compressed Archive	179.33 MB	jdk-15.0.1_linux-x64_bin.tar.gz
macOS Installer	175.94 MB	jdk-15.0.1_osx-x64_bin.dmg
macOS Compressed Archive	176.53 MB	jdk-15.0.1_osx-x64_bin.tar.gz
Windows x64 Installer	159.69 MB	jdk-15.0.1_windows-x64_bin.exe

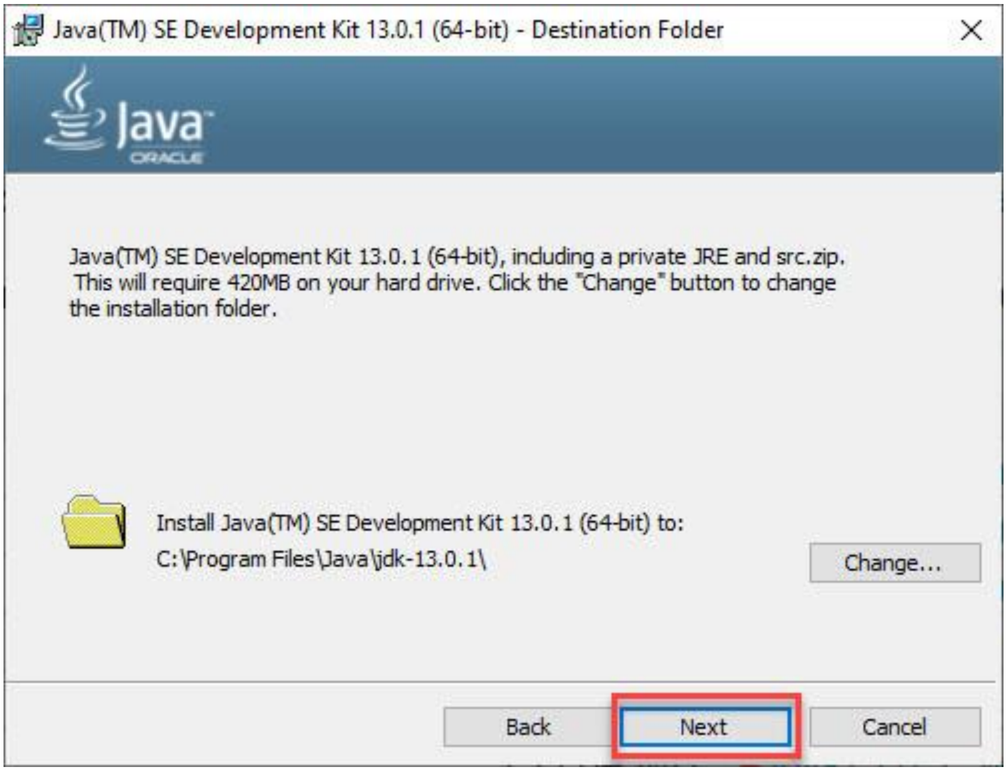
Step 3) Once the download is complete, run the exe for install JDK. Click Next



Welcome to the Installation Wizard for Java SE Development Kit 13.0.1

This wizard will guide you through the installation process for the Java SE Development Kit 13.0.1.

Step 4) Select the PATH for Java installation and click next.



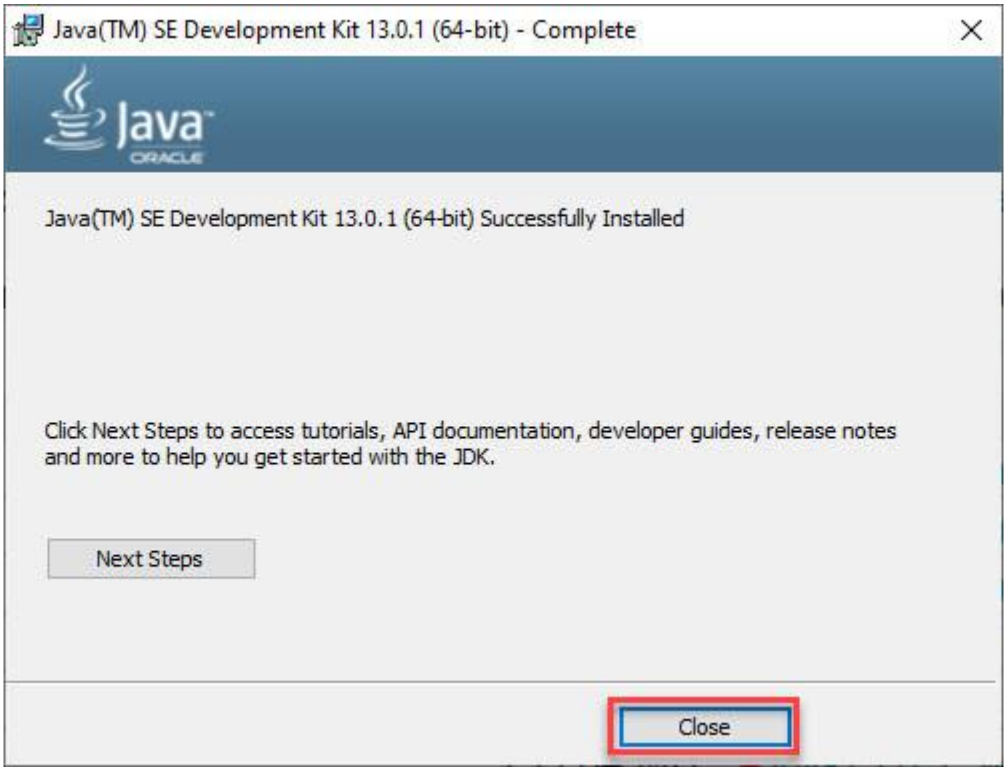
Java(TM) SE Development Kit 13.0.1 (64-bit) - Destination Folder
Java
ORACLE

Java(TM) SE Development Kit 13.0.1 (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 13.0.1 (64-bit) to:
C:\Program Files\Java\jdk-13.0.1\

Change...
Back
Next
Cancel

Step 5) Once installation is complete click Close



Java(TM) SE Development Kit 13.0.1 (64-bit) - Complete

Java(TM) SE Development Kit 13.0.1 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

How to set Environment Variables in Java: Path and Class path

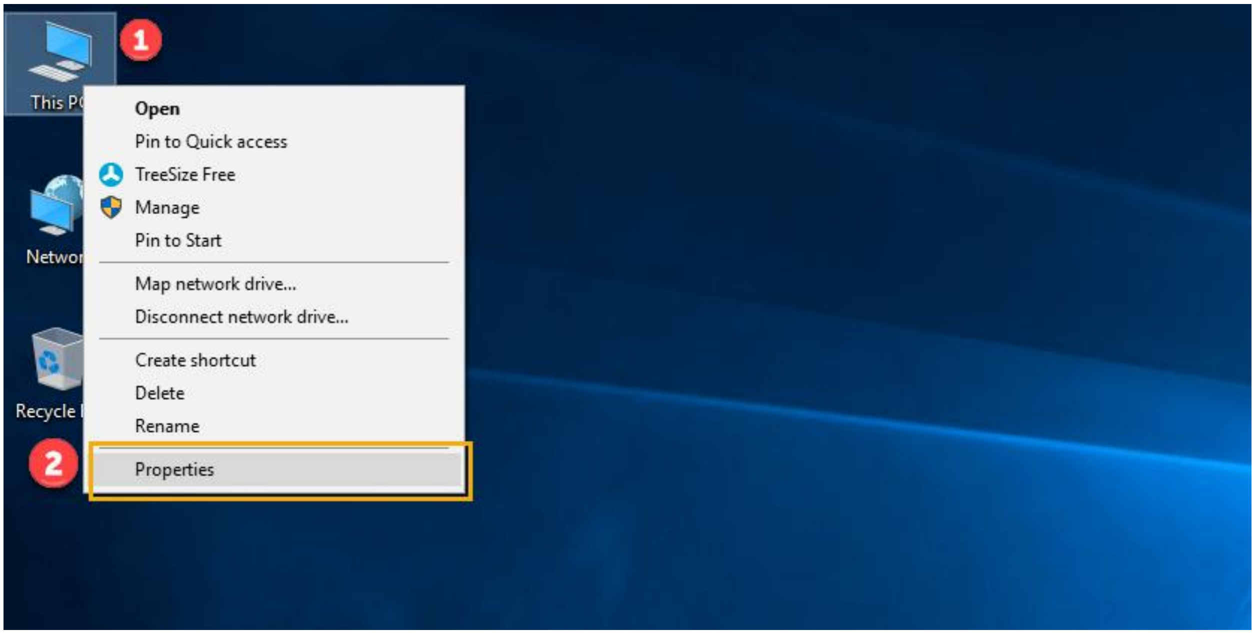
The PATH variable gives the location of executables like javac, java etc. It is possible to run a program without specifying the PATH but you will need to give full path of executable like

C:\Program Files\Java\jdk-15.0.1\bin\javac A.java instead of simple **javac A.java**

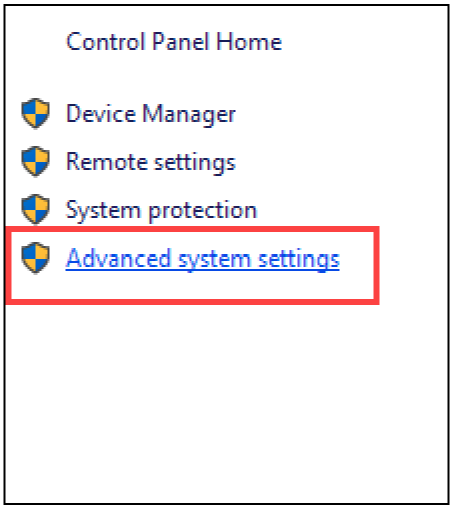
The CLASSPATH variable gives location of the Library Files.

Let's look into the steps to set the PATH and CLASSPATH

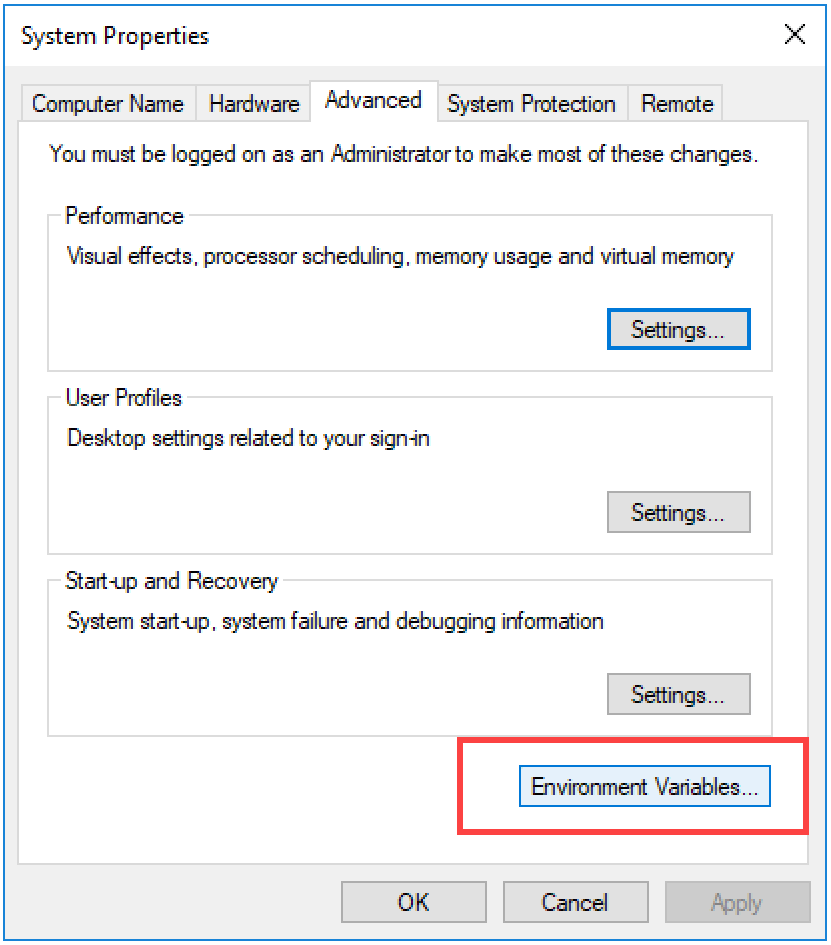
Step 1) Right Click on the My Computer and Select the properties



Step 2) Click on advanced system settings



Step 3) Click on Environment Variables



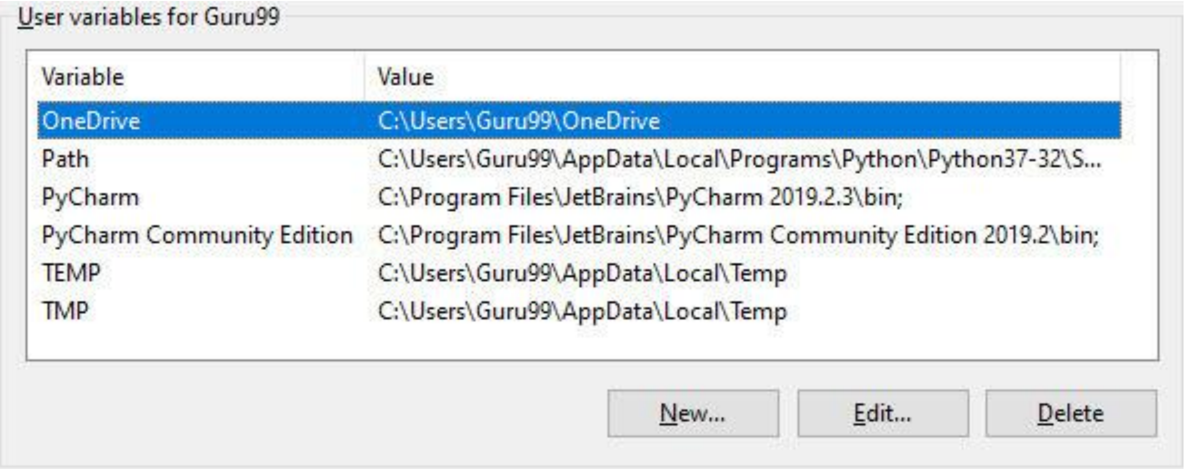
You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage and virtual memory

Desktop settings related to your sign-in

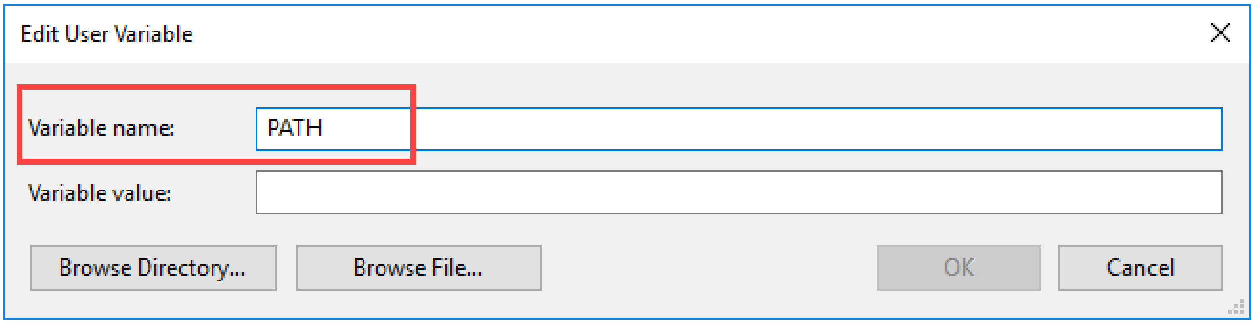
System start-up, system failure and debugging information

Step 4) Click on new Button of User variables



Variable	Value
OneDrive	C:\Users\Guru99\OneDrive
Path	C:\Users\Guru99\AppData\Local\Programs\Python\Python37-32\S...
PyCharm	C:\Program Files\JetBrains\PyCharm 2019.2.3\bin;
PyCharm Community Edition	C:\Program Files\JetBrains\PyCharm Community Edition 2019.2\bin;
TEMP	C:\Users\Guru99\AppData\Local\Temp
TMP	C:\Users\Guru99\AppData\Local\Temp

Step 5) Type PATH in the Variable name.

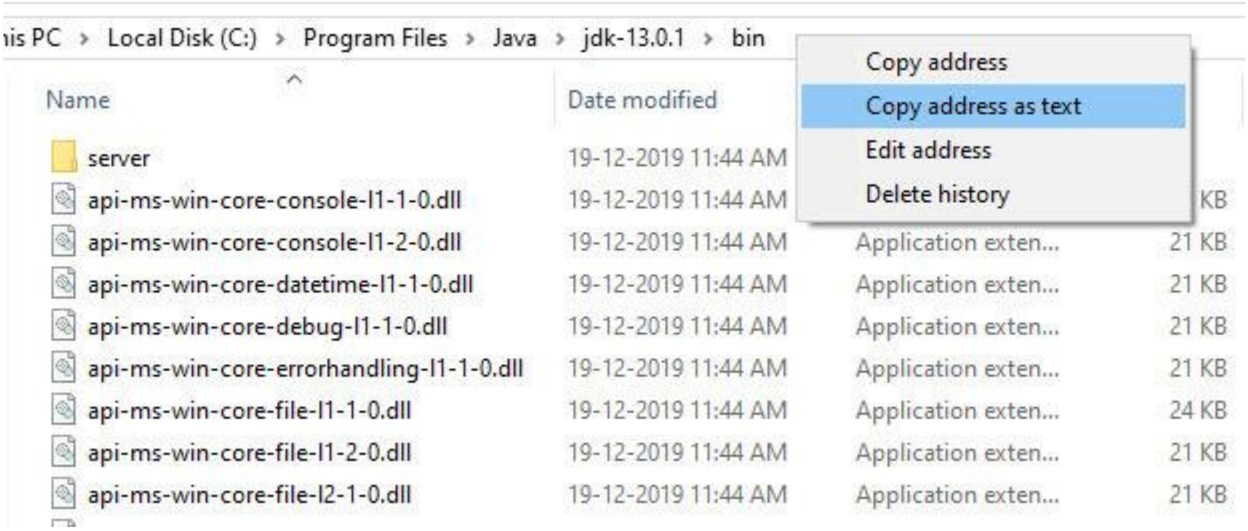


Edit User Variable

Variable name: PATH

Variable value:

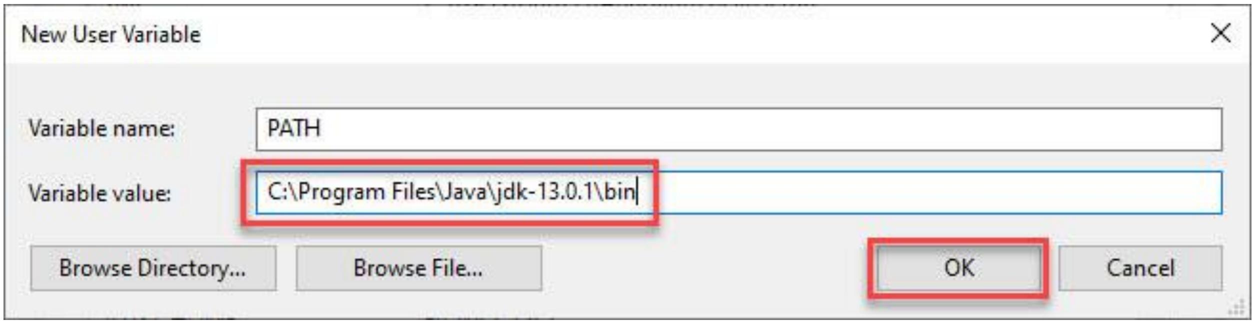
Step 6) Copy the path of bin folder which is installed in JDK folder.



is PC › Local Disk (C:) › Program Files › Java › jdk-13.0.1 › bin

Name	Date modified		
server	19-12-2019 11:44 AM		
api-ms-win-core-console-l1-1-0.dll	19-12-2019 11:44 AM		KB
api-ms-win-core-console-l1-2-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB
api-ms-win-core-datetime-l1-1-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB
api-ms-win-core-debug-l1-1-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB
api-ms-win-core-errorhandling-l1-1-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB
api-ms-win-core-file-l1-1-0.dll	19-12-2019 11:44 AM	Application exten...	24 KB
api-ms-win-core-file-l1-2-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB
api-ms-win-core-file-l2-1-0.dll	19-12-2019 11:44 AM	Application exten...	21 KB

Step 7) Paste Path of bin folder in Variable value and click on OK Button.



New User Variable

Variable name: PATH

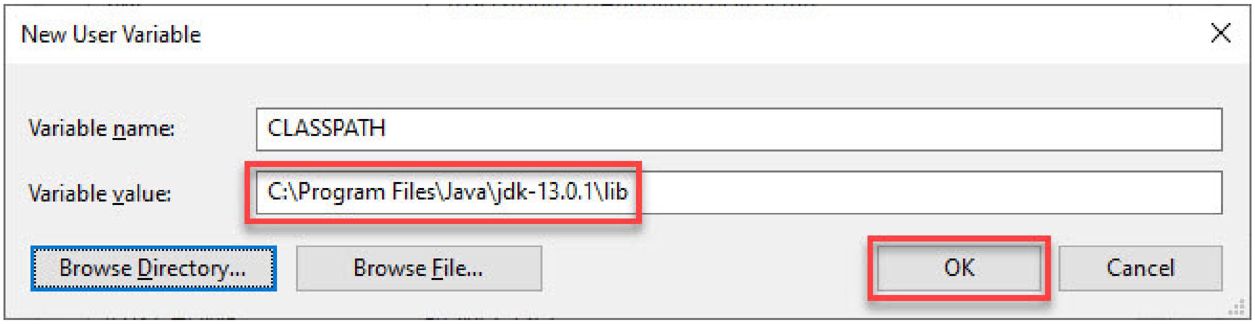
Variable value: C:\Program Files\Java\jdk-13.0.1\bin

Note: In case you already have a PATH variable created in your PC, edit the PATH variable to

```
PATH = <JDK installation directory>\bin;%PATH%;
```

Here, %PATH% appends the existing path variable to our new value

Step 8) you can follow a similar process to set CLASSPATH



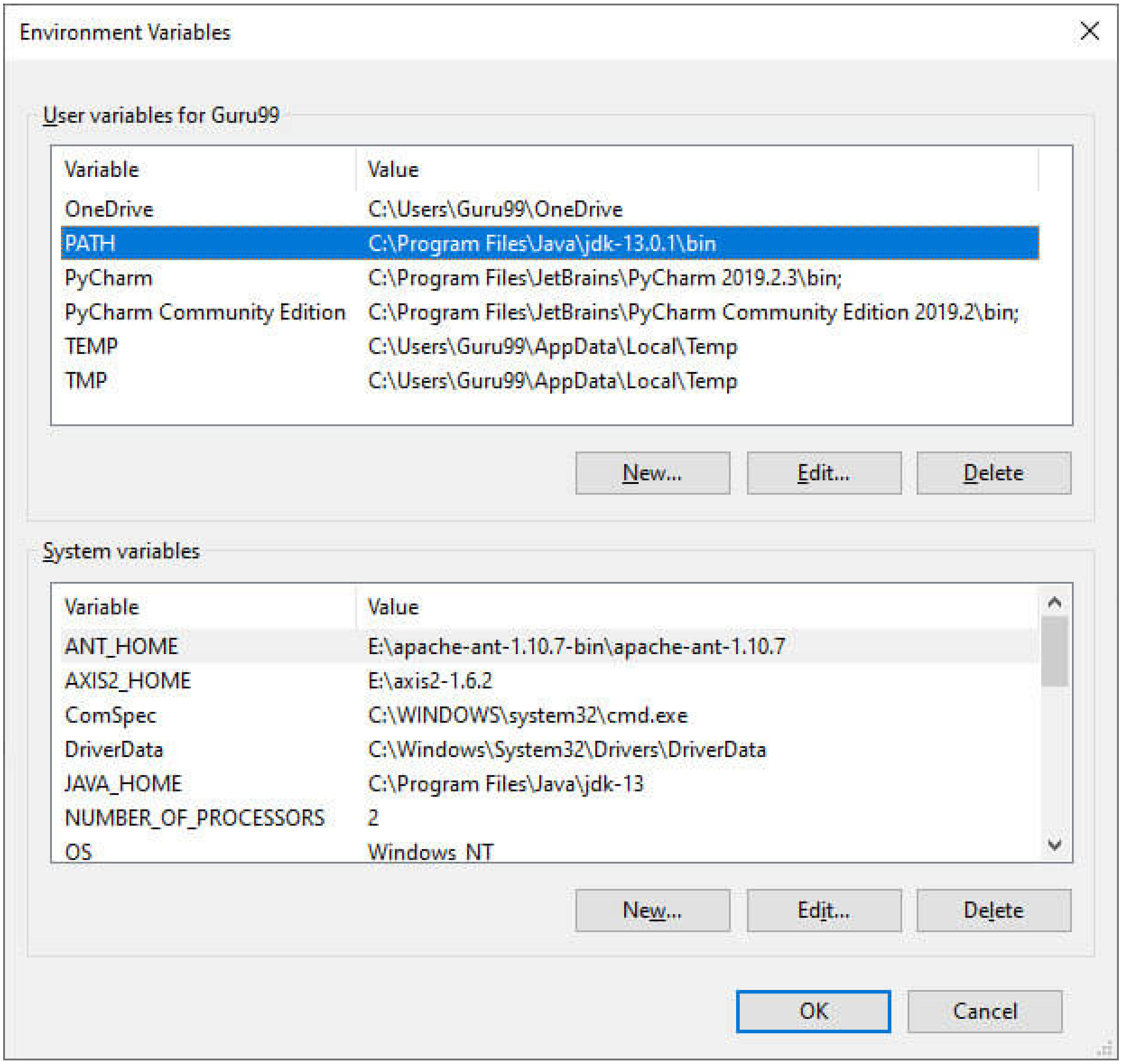
Variable name: CLASSPATH

Variable value: C:\Program Files\Java\jdk-13.0.1\lib

Note: In case you java installation does not work after installation, change classpath to

```
CLASSPATH = <JDK installation directory>\lib\tools.jar;
```

Step 9) Click on OK button



User variables for Guru99

Variable	Value
OneDrive	C:\Users\Guru99\OneDrive
PATH	C:\Program Files\Java\jdk-13.0.1\bin
PyCharm	C:\Program Files\JetBrains\PyCharm 2019.2.3\bin;
PyCharm Community Edition	C:\Program Files\JetBrains\PyCharm Community Edition 2019.2\bin;
TEMP	C:\Users\Guru99\AppData\Local\Temp
TMP	C:\Users\Guru99\AppData\Local\Temp

System variables

Variable	Value
ANT_HOME	E:\apache-ant-1.10.7-bin\apache-ant-1.10.7
AXIS2_HOME	E:\axis2-1.6.2
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\Java\jdk-13
NUMBER_OF_PROCESSORS	2
OS	Windows_NT

Step 10) Go to command prompt and type javac commands.

How to Check if Java is Installed or Not:

First, **let's open a command window or terminal and enter:**

`> java –version` **for check which version is installed**

If Java is installed and the PATH is configured correctly, our output will be similar to:

```
java version "15.0.1"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java Hotspot(TM) Client VM (build 25.31-b07, mixed mode, sharing)
```

Otherwise, we'll see an error message like the one below and we need to check elsewhere:

```
'java' is not recognized as an internal or external command,
Operable program or batch file.
```

The exact messages we see will vary depending on the operating system used and the Java version installed.

Going to a command line and typing *java -version* can tell us for sure if Java is installed. However, if we see an error message, Java might still be installed – we'll just need to investigate further.

Many discussions about using *java -version* mention the JAVA_HOME environment variable.

This is misleading because JAVA_HOME won't affect our *java -version* results.

Additionally, JAVA_HOME should point to a JDK and other applications that use JDK features, such as Maven, use it.

For more information, please check our articles JAVA_HOME should point to a JDK and how to set JAVA_HOME.

So, let's look at alternative ways to find Java, in case the command-line failed us

Application

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

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. 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 enterprise application. It has advantages of the 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 which is mainly used to develop web and enterprise applications. It is built on the 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 which is mainly used to develop mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a light-weight user interface API.

History of Java:

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic".

Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

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) Initially designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

Why Java named "Oak"?

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the 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 Java Programming named "Java"?

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 than other names.

8) Java is an island of Indonesia where the first coffee was produced (**called java coffee**). It is a kind of espresso bean. Java name was chosen by James Gosling while having coffee near his office.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996). After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds the new features in Java.

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 11.0.8 (20th Mar 2018)
13. Java SE 14.0.2 (20th Mar 2018)
14. Java SE 15 latest October 2020

First Java Program | Hello World Example

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, you need to

- 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:

```
public class Simple{

public static void main(String args[]){

System.out.println("Hello Engineering Tutor");
}}
```

Save this file as **Simple.java**

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

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 which 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 to create 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** is used for command line argument. We will learn it later.

```
}
```

- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

How many ways can we write a Java program

There are many ways to write a Java program. The modifications that can be done in a Java program are given below:

1) By changing the sequence of the modifiers, method prototype is not changed in Java.

Let's see the simple code of the main method.

```
static public void main(String args[])
```

2) The subscript notation in Java array can be used after type, before the variable or after the variable.

Let's see the different codes to write the main method.

```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
```

3) You can provide var-args support to the main method by passing 3 ellipses (dots)

Let's see the simple code of using var-args in the main method. We will learn about var-args later in Java New Features chapter.

```
public static void main(String... args)
```

4) Having a semicolon at the end of class is optional in Java.

Let's see the simple code.

```
class A{

static public void main(String... args){

System.out.println("hello Satish");

}
};
```

Valid java main method signature

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])
4. **public static void** main(String... args)
5. **static public void** main(String[] args)
6. **public static final void** main(String[] args)
7. **final public static void** main(String[] args)
8. **final strictfp public static void** main(String[] args)

Invalid java main method signature

1. **public void** main(String[] args)
2. **static void** main(String[] args)
3. **public void static** main(String[] args)
4. **abstract public static void** main(String[] args)

Resolving an error " javac is not recognized as an internal or external command "?

If there occurs a problem like displayed in the below figure, you need to set path. Since DOS doesn't know javac or java, we need to set path.

The path is not required in such a case if you save your program inside the JDK/bin directory. However, it is an excellent approach to set the path.

Java Variables Detailed Introduction:

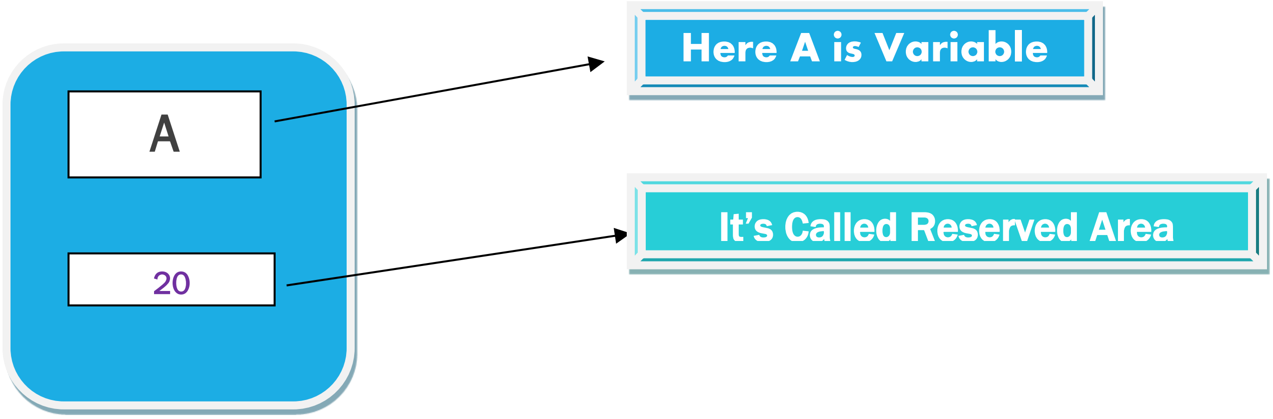
A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

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

There are two types of data types in Java: primitive and non-primitive.

Variable:

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



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

Types of Variables:

There are three types of variables in Java:

- local variable
- instance variable
- static variable

Types of Variables:

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 instance variable. It is not declared as static.

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

3) Static variable

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

Example to understand the types of variables in java

```
class A
{                        // Starting Of Class
int data=50;             //instance variable
static int m=100;       //static variable
void method()
{
int n=90;               //local variable
}
}                       //end of class
```

Java Variable Example: Add Two Numbers

```
class Simple
{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Output: 20

Java Variable Example: Widening:

```
class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output: 10, 10.0

Java Variable Example: Narrowing (Typecasting)

```
class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;        //Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}}
```

Output: 10.5, 10

Java Variable Example: Overflow

```
class Simple{
public static void main(String[] args){
//Overflow
int a=130;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}}
```

Output: **130, -126**

Java Variable Example: Adding Lower Type:

```
class Simple{
public static void main(String[] args){
byte a=10;
byte b=10;
//byte c=a+b;
//Compile Time Error: because a+b=20 will be int
byte c=(byte)(a+b);
System.out.println(c);
}}
```

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

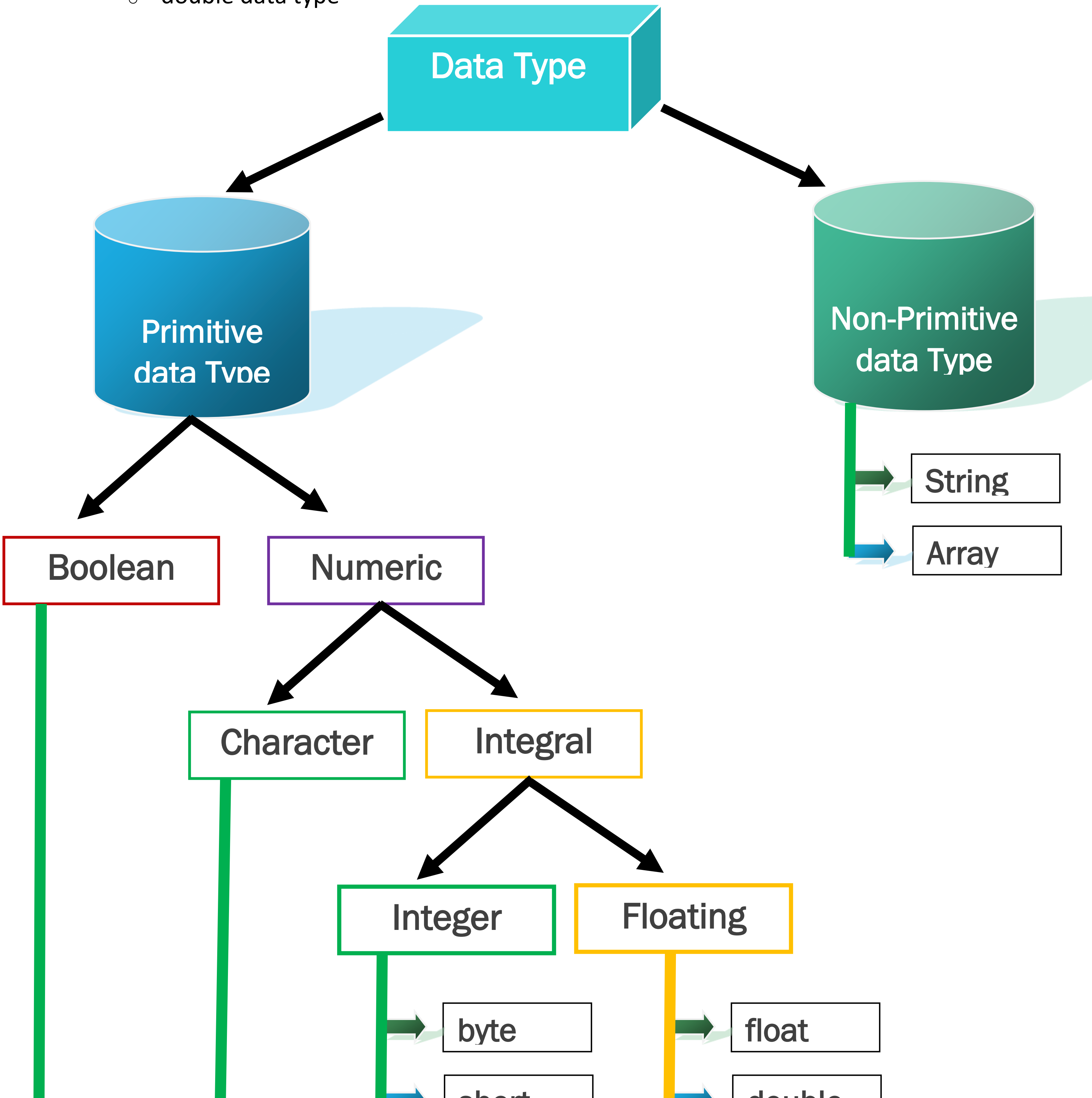
Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

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



There are Data types and Default Size:

Data Type	Default Size
boolean	1 Bit
char	2 Byte
byte	1 Byte
short	2 Byte
int	4 Byte
long	8 Byte
float	4 Byte
double	8 Byte

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. To get detail explanation about Unicode visit next page.

Unicode System

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

Problem

This caused two problems:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length.Some common chara
   encoded as single bytes, other require two or more byte.

Operators in Java

Operator in Java is a symbol which 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	< > <= >= instance of
	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 --

```
class OperatorExample
{
public static void main(String args[])
{
int x=10;
System.out.println(x++);       //10 (11)   First write Than Increment
System.out.println(++x);       //12
System.out.println(x--);       //12 (11)
System.out.println(--x);       //10
}}
```

Output:

```
10,
12,
12,
10.
```

Java Unary Operator Example 2: ++ and –

```
class OperatorExample
{
public static void main(String args[])
{
int a=10;
int b=10;
System.out.println(a++ + ++a);      //10+12=22
System.out.println(b++ + b++);     //10+11=21
}
}
```

Note Practice Required

Java Unary Operator Example: ~ and !

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example:

```
class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

System.out.println(a+b);    //15

System.out.println(a-b);    //5

System.out.println(a*b);    //50

System.out.println(a/b);    //2

System.out.println(a%b);    //0

}}
```

Output: 15, 5, 50, 2, 0

Java Left Shift Operator:

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example:

```
class OperatorExample{

public static void main(String args[]){

System.out.println(10<<2);      //10*2^2=10*4=40

System.out.println(10<<3);     //10*2^3=10*8=80

System.out.println(20<<2);     //20*2^2=20*4=80

System.out.println(15<<4);    //15*2^4=15*16=240

}}
```

Java Right Shift Operator:

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Java Right Shift Operator Example:

```
class OperatorExample{

public static void main(String args[]){
```

```
System.out.println(10>>2);       //10/2^2=10/4=2

System.out.println(20>>2);      //20/2^2=20/4=5

System.out.println(20>>3);     //20/2^3=20/8=2

}}
```

Java Shift Operator Example: >> vs. >>>

```
class OperatorExample{

public static void main(String args[]){

    //For positive number, >> and >>> works same

    System.out.println(20>>2);

    System.out.println(20>>>2);

    //For negative number, >>> changes parity bit (MSB) to 0

    System.out.println(-20>>2);

    System.out.println(-20>>>2);

}}
```

Java AND Operator Example: Logical && and Bitwise & -:

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);     //false && true = false
System.out.println(a<b&a<c);      //false & true = false
}}
```

Java AND Operator Example: Logical && vs Bitwise &

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);  //false && true = false
System.out.println(a);  //10 because second condition is not checked
System.out.println(a<b&a++<c);  //false && true = false
System.out.println(a);      //11 because second condition is checked
}}
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true.

It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true  //|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Java Ternary Operator:

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. it is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Another Example:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Java Assignment Operator:

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example:

```
class OperatorExample{
```

```
public static void main(String args[]){

int a=10;

int b=20;

a+=4;//a=a+4 (a=10+4)

b-=4;//b=b-4 (b=20-4)

System.out.println(a);

System.out.println(b);

}}
```

Java Assignment Operator Example:

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;        //10+3
System.out.println(a);
a-=4;         //13-4
System.out.println(a);
a*=2;         //9*2
System.out.println(a);
a/=2;        //18/2
System.out.println(a);
}}
```

Java Assignment Operator Example: Adding short:

```
class OperatorExample{

public static void main(String args[]){

short a=10;

short b=10;

//a+=b;//a=a+b internally so fine

a=a+b;//Compile time error because 10+10=20 now int

System.out.println(a);

   }}  // Output: Compile Time Error.
```

After type cast:

```
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
a=(short)(a+b);          //20 which is int now converted to short
System.out.println(a);
}}
```

Java Keywords

Java keywords are also known as **reserved words**. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name.

List of Java Keywords

A list of Java keywords or reserved words is given below:

1. **abstract:** Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
4. **byte:** Java byte keyword is used to declare a variable that can hold an 8-bit data values.
5. **case:** Java case keyword is used to with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

7. **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class**: Java class keyword is used to declare a class.
9. **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default**: Java default keyword is used to specify the default block of code in a switch statement.
11. **do**: Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
12. **double**: Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
13. **else**: Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final**: Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
17. **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
18. **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if**: Java if keyword tests the condition. It executes the if block if condition is true.
21. **implements**: Java implements keyword is used to implement an interface.

22. **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof**: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int**: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long**: Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new**: Java new keyword is used to create new objects.
29. **null**: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package**: Java package keyword is used to declare a Java package that includes the classes.
31. **private**: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. **protected**: Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
33. **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return**: Java return keyword is used to return from a method when its execution is complete.
35. **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.

36. **static**: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.

37. **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.

38. **super**: Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.

39. **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

40. **synchronized**: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

41. **this**: Java this keyword can be used to refer the current object in a method or constructor.

42. **throw**: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.

43. **throws**: The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.

44. **transient**: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

45. **try**: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

46. **void:** Java void keyword is used to specify that a method does not have a return value.

47. **volatile**: Java volatile keyword is used to indicate that a variable may change asynchronously.

48. **while**: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop

2. Conditional Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in Java.

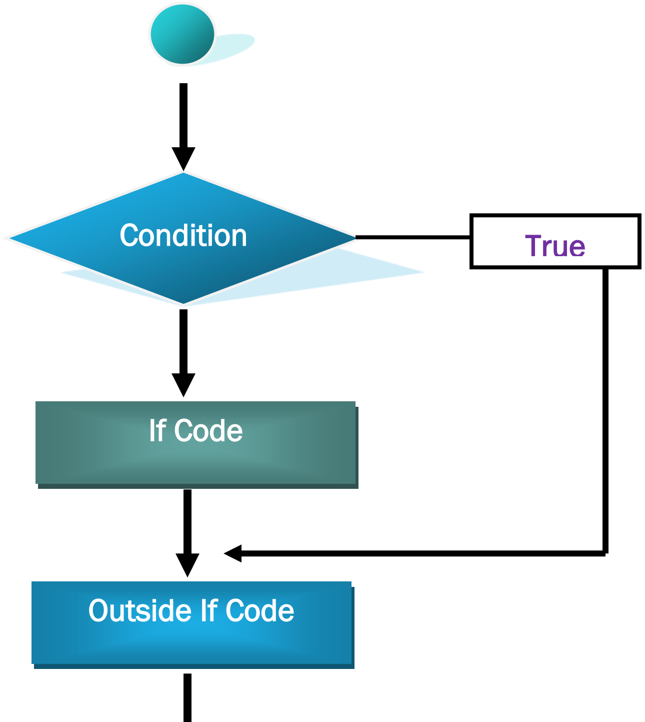
- **if** statement
- **if-else** statement
- **if-else-if** ladder
- **nested if** statement

Java if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if (condition){
//code to be executed
}
```



```
//Java Program to demonstate the use of if statement.

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");

    }

}

}
```