Java OOP (Object-Oriented Programming)

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers *write once, run anywhere* meaning that compiled Java code can run on all platforms that support Java without the need to recompile.

Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub particularly for client–server web applications, with a reported 9 million developers.

Java was originally developed by James Gosling at Sun Microsystems. It was released in May 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GPL-2.0-only license. Oracle offers its own HotSpot Java Virtual Machine, however the official reference implementation is the OpenJDK JVM which is free open-source software and used by most developers and is the default JVM for almost all Linux distributions.

Why we use Java?

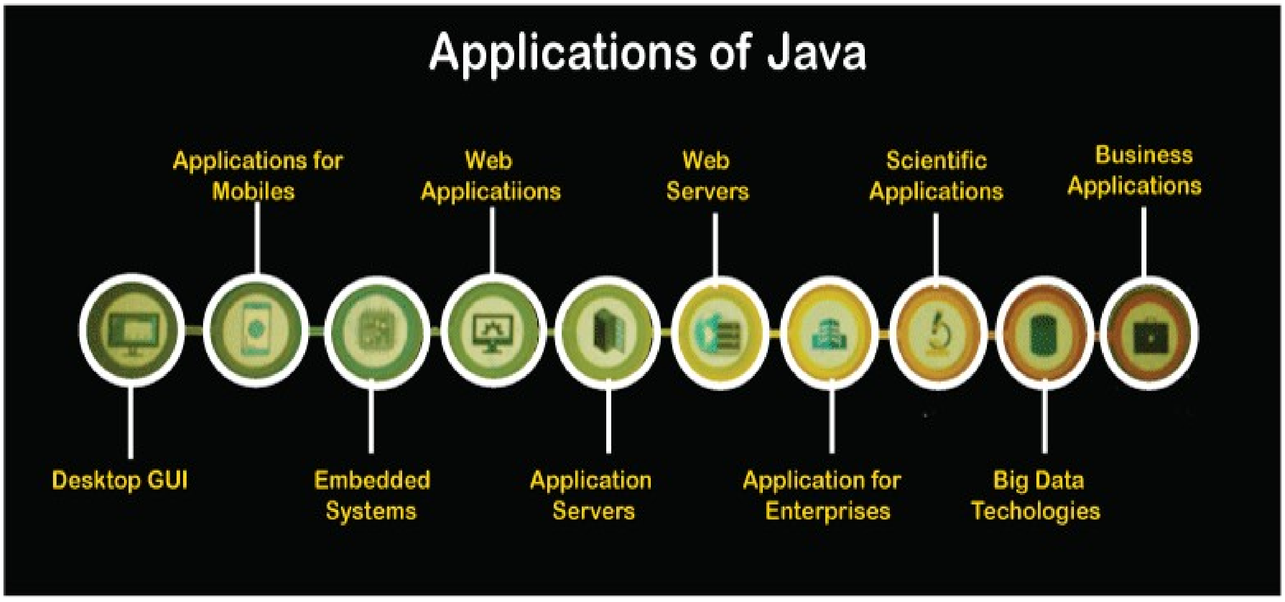
In comparison to other programming languages, Java stands alone for its security and functionality. Java isolates itself from other programming languages because of functionality and security and it is relevant too. There are some other reasons to use Java are as follows:

- **Scalability:** Scalability adds capacity to our system. It improves the capacity of the system by adding the system resources without affecting the deployment architecture. We can achieve scalability by increasing the resources such as RAM and CPU in a single system. It is important because it handles the workload, increases the system performance, and maximizes productivity.

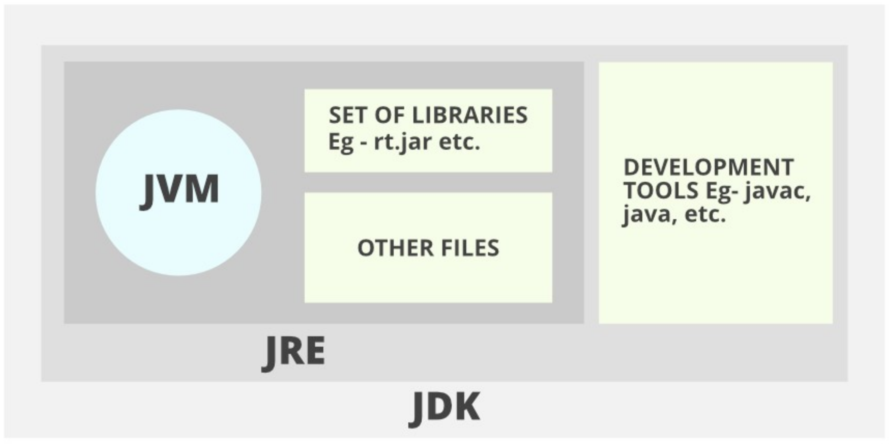
- **Cross-Platform:** Cross-platform means, a compiled Java program can be run on all the platforms. Remember that the system must have JVM. After compiling a Java program, the Java code gets converted into the bytecode which is platform-independent. This bytecode is understood by the JVM. We can run this bytecode on any platform.
- **Memory-Management:** Java provides its own mechanism for managing the memory is known as garbage collection. We need not to care about memory and do not required to implement it to manage the memory. It automatically deletes the objects when they no longer used by the application. It improves the speed of the application.
- **Multi-threading:** Thread is a light-weight subprocess. Multi-threading in Java allows concurrent execution of two or more threads simultaneously. It maximizes the utilization of the CPU.

What is Java Used For?

Java is the most popular, widely used object-oriented programming language. The security feature of Java makes it popular and widely used. It used by many Java enthusiasts for different purposes. By **using Java**, we can develop a variety of applications such as enterprise applications, network applications, desktop applications, web applications, games, android app, and many more.



Below are the environment settings for both Linux and Windows. JVM, JRE, and JDK three are all platform-dependent because the configuration of each Operating System is different. But, Java is platform-independent. Few things must be clear before setting up the environment which can better be perceived from the below image provided as follows:


JVM
SET OF LIBRARIES Eg - rt.jar etc.
OTHER FILES
DEVELOPMENT TOOLS Eg- javac, java, etc.
JRE
JDK

- **JDK**(Java Development Kit): JDK is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.

- **JRE**(Java Runtime Environment): JRE contains the parts of the Java libraries required to run Java programs and is intended for end-users. JRE can be viewed as a subset of JDK.

- **JVM:** JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides a runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms.

Install Java in Linux Operating System

In Linux, there are several ways to install java. But we will refer to the simplest and easy way to install java using a terminal. For Linux, we will install

Lab@2022

Step 1: Go to **Application -> Accessories -> Terminal**. Or press (Alt +Ctrl+T)

Step 2: Type command as below as follows:

sudo apt update

```
sudo apt  install openjdk-8-jdk
```

Install Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

1. From your shell, install Git using apt-get:

   ```
   $ sudo apt-get update
   
   $ sudo apt-get install git
   ```

2. Verify the installation was successful by typing git --version:

   ```
   $ git –version
   ```

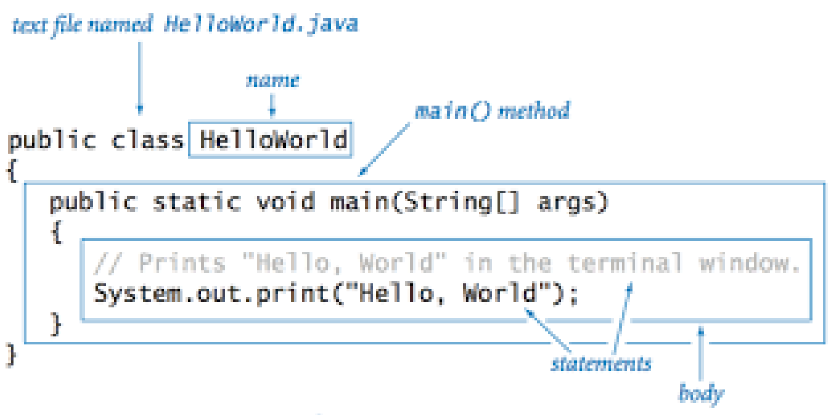
3. Configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

   ```
   $ git config --global user.name "your Account Name"
   
   $ git config --global user.email "Email@Yahoo.com"
   
   $ git clone https://github.com/gitrepository/projectname.git
   ```

```
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
```

What is java Program

Java program is a collection of objects, and these objects communicate through method calls to each other to work together.



```
public class HelloWorld
{
   public static void main(String[] args)
   {
      // Prints "Hello, World" in the terminal window.
      System.out.print("Hello, World");
   }
}
```

The below-given program is the most simple program of Java printing “Hello World” to the screen. Let us try to understand every bit of code step by step.

```
// This is a simple Java program.
// FileName : "HelloWorld.java".

class HelloWorld
{
// Your program begins with a call to main().
// Prints "Hello, World" to the terminal window.
   public static void  main(String args[])
{
System.out.println("Hello, World");
}
}
```

Output

```
Hello, World
```

Parts Of java Program

1. Class definition

This line uses the keyword **class** to declare that a new class is being defined.

```
class HelloWorld {
    //
    //Statements
}
```

2. main method:

In the Java programming language, every application must contain a main method. The main function(method) is the entry point of your Java application, and it’s mandatory in a Java program. whose signature in Java is:

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

3. Output method:

```
System.out.println("Hello, World");
```

This line outputs the string “Hello, World” followed by a new line on the screen. Output is accomplished by the built-in println(*)* method. The **System** is a predefined class that provides access to the system, and **out** is the variable of type output stream connected to the console.

4.Comments

They can either be multiline or single-line comments.

```
// This is a simple Java program.
// Call this file "HelloWorld.java".
```

multiline comments, they must begin from /* and end with */.

Compiling the program

- After successfully setting up the environment, we can open a terminal in both Windows/Unix and go to the directory where the file – HelloWorld.java is present.

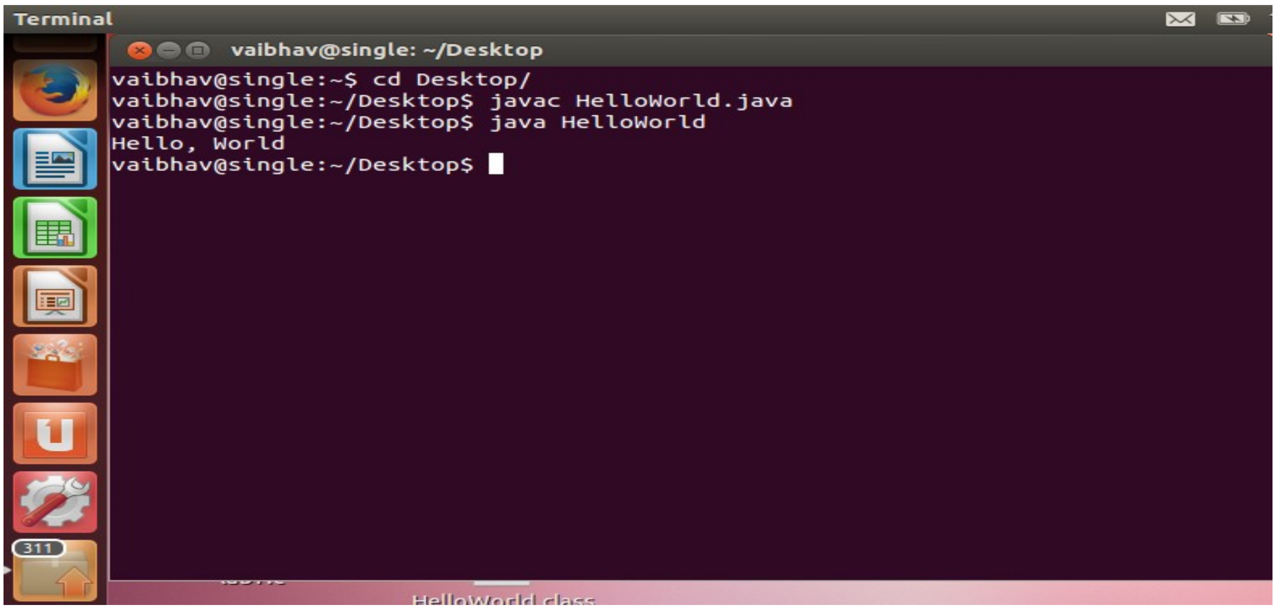
- Now, to compile the HelloWorld program, execute the compiler – javac, to specify the name of the **source** file on the command line, as shown:

```
javac HelloWorld.java
```

- The compiler creates a HelloWorld.class (in the current working directory) that contains the bytecode version of the program. Now, to execute our program, **JVM**(Java Virtual Machine) needs to be called using java, specifying the name of the **class** file on the command line, as shown:

```
java HelloWorld
```

- This will print “Hello World” to the terminal screen.



```
vaibhav@single:~$ cd Desktop/
vaibhav@single:~/Desktop$ javac HelloWorld.java
vaibhav@single:~/Desktop$ java HelloWorld
Hello, World
vaibhav@single:~/Desktop$ 
```

. Source File Name

The name of a source file should exactly match the public class name with the extension of **.java**. The name of the file can be a different name if it does not have any public class. Assume you have a public class h**elloworld**. Try Compilation of the code.

3. Case Sensitivity

Java is a case-sensitive language, which means that the identifiers ***AB, Ab, aB***, and ***ab*** are different in Java.

```
System.out.println("Alice"); // valid syntax
system.out.println("Alice"); // invalid syntax
```

4. Class Names

i. The first letter of the class should be in Uppercase (lowercase is allowed, but discouraged).

ii. If several words are used to form the name of the class, each inner word's first letter should be in Uppercase. Underscores are allowed, but not recommended. Also allowed are numbers and currency symbols, although the latter are also discouraged because the are used for a special purpose (for inner and anonymous classes).

```
class MyJavaProgram    // valid syntax
class 1Program         // invalid syntax
class My1Program       // valid syntax
class $Program         // valid syntax, but discouraged
class My$Program       // valid syntax, but discouraged (inner class Program
inside the class My)
class myJavaProgram    // valid syntax, but discouraged
```

5. public static void main(String [] args)

The method main() is the main entry point into a Java program; this is where the processing starts. Also allowed is the signature **public static void main(String… args)**.

6. Method Names

i. All the method names should start with a lowercase letter.

ii. If several words are used to form the name of the method, then each first letter of the inner word should be in Uppercase. Underscores are allowed, but not recommended. Also allowed are digits and currency symbols.

```
public void employeeRecords() // valid syntax
public void EmployeeRecords() // valid syntax, but discouraged
```

Java Identifiers

In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label. For example :

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 20;
    }
}
```

In the above java code, we have 5 identifiers namely :

- **Test**: class name.
- **main**: method name.
- **String**: predefined class name.
- **args**: variable name.
- **a**: variable name.

Rules for defining Java Identifiers

There are certain rules for defining a valid java identifier. These rules must be followed, otherwise we get compile-time error. These rules are also valid for other languages like C,C++.

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '**$**'(dollar sign) and '**_**' (underscore).For example "geek@" is not a valid java identifier as it contain '@' special character.

- Identifiers should **not** start with digits(**[0-9]**). For example “123geeks” is a not a valid java identifier.
- Java identifiers are **case-sensitive**.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- **Reserved Words** can’t be used as an identifier. For example “int while = 20;” is an invalid statement as while is a reserved word. There are **53** reserved words in Java.

Examples of valid identifiers :

```
MyVariable
MYVARIABLE
myvariable
x
i
x1
i1
_myvariable
$myvariable
sum_of_array
geeks123
```

Examples of invalid identifiers :

```
My Variable  // contains a space
123geeks   // Begins with a digit
a+c // plus sign is not an alphanumeric character
variable-2 // hyphen is not an alphanumeric character
sum_&_difference // ampersand is not an alphanumeric character
```

Reserved Words

Any programming language reserves some words to represent functionalities defined by that language. These words are called reserved words.

List of Java reserved words

abstract	do	if	private	this
assert	double	implements	protected	throw
boolean	else	import	public	throws
break	enum	instanceof	return	transient
byte	extends	int	short	true
case	false	interface	static	try
catch	final	long	strictfp	void
char	finally	native	super	volatile
class	float	new	switch	while
const	for	null	synchronized	continue
default	goto	package		

Variables in Java

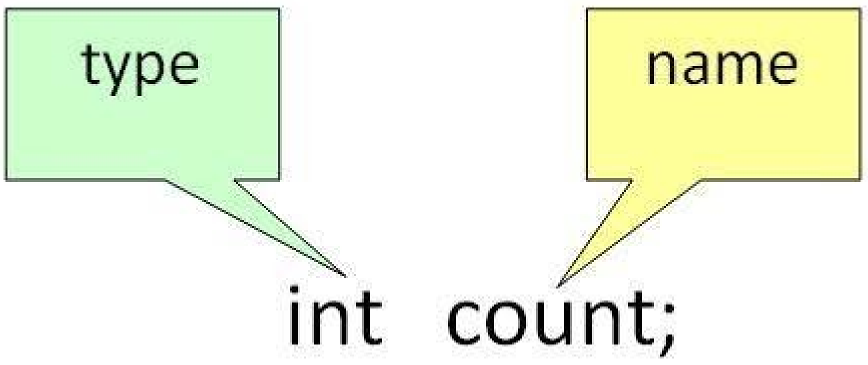
Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.

A variable is a name given to a memory location. It is the basic unit of storage in a program

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.

How to declare variables?

We can declare variables in Java as pictorially depicted below as a visual aid.



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

1. datatype: Type of data that can be stored in this variable.

2. data_name: Name given to the variable.

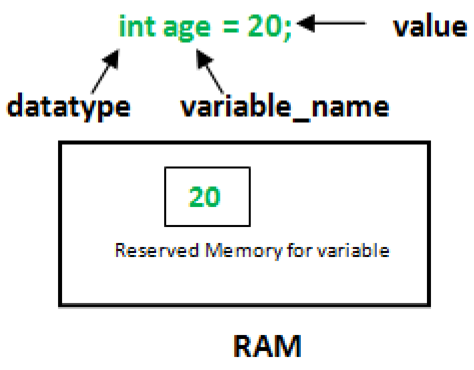
In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

How to initialize variables?

It can be perceived with the help of 3 components that are as follows:

- **datatype**: Type of data that can be stored in this variable.
- **variable_name**: Name given to the variable.
- **value**: It is the initial value stored in the variable.



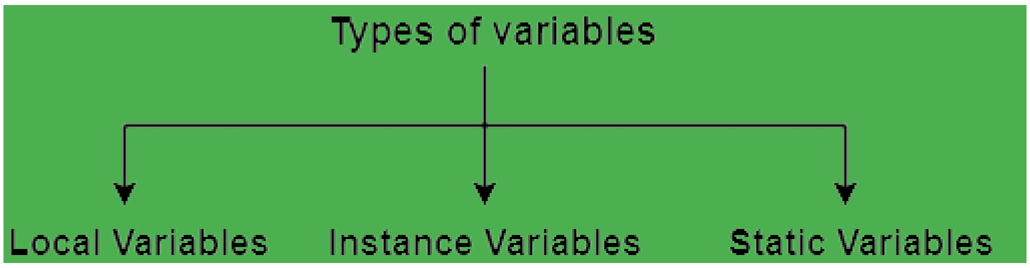
Illustrations:

```
float simpleInterest;
// Declaring float variable

int time = 10, speed = 20;
// Declaring and initializing integer variable

char var = 'h';
// Declaring and initializing character variable
```

Types of Variables in Java

Now let us discuss different types of variables which are listed as follows:

1. Local Variables
2. Instance Variables
3. Static Variables



1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

```
/*package whatever //do not write package name here */
// Contributed by Shubham Jain
import java.io.*;

class GFG {
public static void main(String[] args)
{
int var = 10; // Declared a Local Variable
// This variable is local to this main method only
System.out.println("Local Variable: " + var);
}
}
```

Output

```
Local Variable: 10
```

2. Instance Variables

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of an instance variable is not mandatory. Its default value is 0.
- Instance variables can be accessed only by creating objects.

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

        public String geek; // Declared Instance Variable

        public GFG()
        { // Default Constructor

                this.geek = "Ali ahmed"; // initializing Instance Variable
        }
//Main Method
        public static void main(String[] args)
        {

                // Object Creation
                GFG name = new GFG();
                // Displaying O/P
                System.out.println("his  name is: " + name.geek);
        }
}
```

Output

```
his name is: Ali Ahmed
```

3. Static Variables

Static variables are also known as class variables.

- These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of a static variable is not mandatory. Its default value is 0.
- If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won’t halt the program. The compiler will replace the object name with the class name automatically.
- If we access a static variable without the class name, the compiler will automatically append the class name.

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

public static String geek = "Ali Ahmed";                    //Declared static variable

        public static void main (String[] args) {

        //geek variable can be accessed without object creation
        //Displaying O/P
        //GFG.geek --> using the static variable
                System.out.println("his Name is : "+GFG.geek);
        }
}
```

Output

```
his Name is : Ali Ahmed
```