**JAVA**

**JAVA** was developed by James Gosling at **Sun Microsystems** Inc in the year **1995** and later acquired by Oracle Corporation.

[Java](https://www.geeksforgeeks.org/java/) is a class-based, object-oriented programming language.

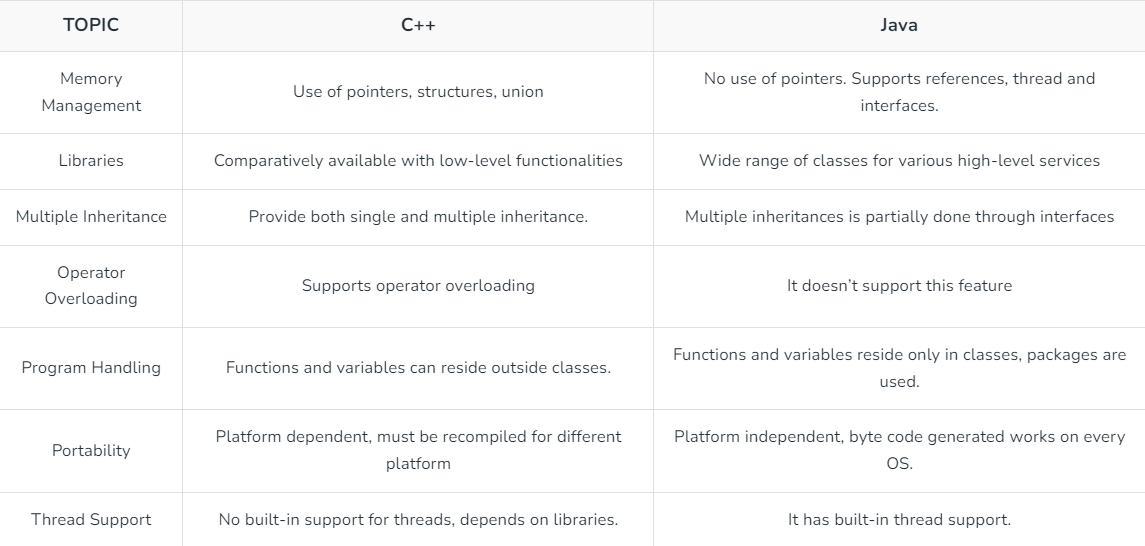
**Steps for running of java program :**

- Writing a program is done by a java programmer like you and me.

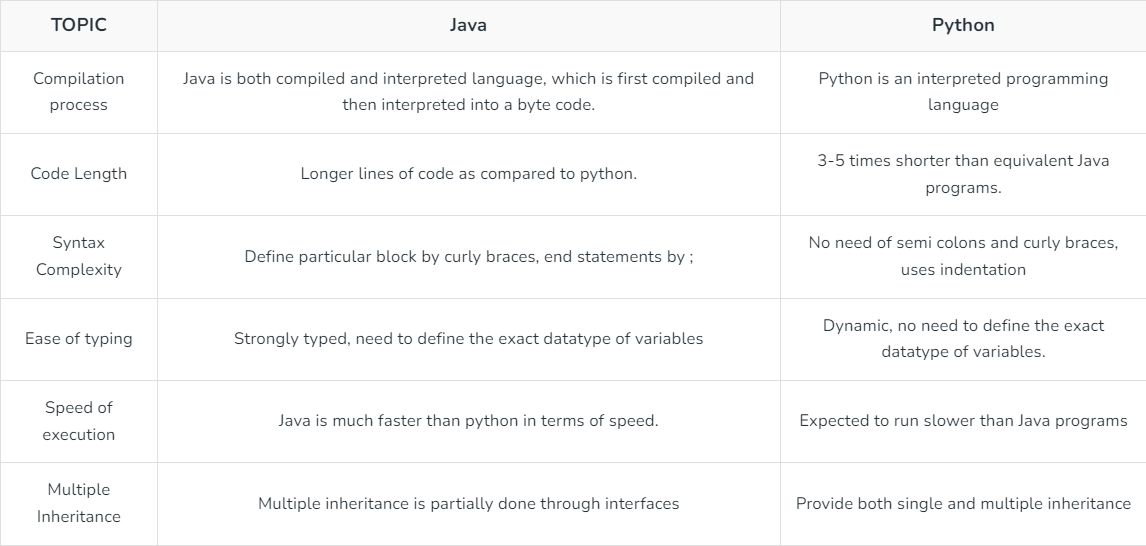
- The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.

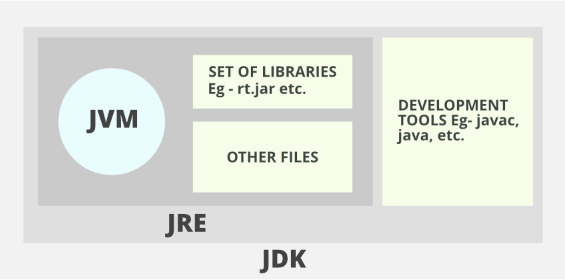
- In the Runningphase of a program,**JVM** executes the bytecode generated by the compiler.

**C++ vs Java**



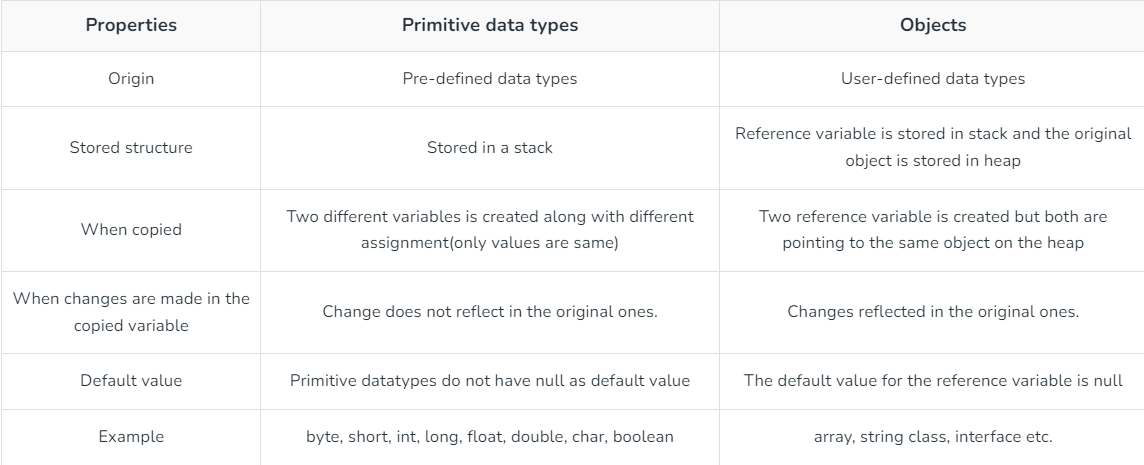
**Java vs Python**





* **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.

**Difference between primitive and object data types:-**



**Java Identifiers**

Java Identifiers can be a class name, method name, variable name, or label.

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

In this code, there are 5 identifiers : Test, main, String, args, a.

Valid Identifier : contains alphabets, number, $ and underscore. Can’t start with digits, case-sensitive. Keywords can’t be used as identifier.

**Arithmetic Operators**

**\* :**Multiplication

**/ :**Division

**% :**Modulo

**+ :**Addition

**– :**Subtraction

**Unary Operators**

**– :** **Unary minus**

**+ :** **Unary plus**

**++ :** **Increment operator**

**Post-Increment**

**Pre-Increment**

**– –  : Decrement operator**

**Post-decrement**

**Pre-Decrement**

**! : Logical not operator**

**Assignment Operators**

= += -= \*= /= %= >>= <<= >>>=

**Relational Operators**

== != >= <= > <

**Logical Operators**

&& || !

**Ternary Operator**

Condition ? if true : if false

**Bitwise Operators**

& | ^ ~

**Shift Operators**

**<<, Left shift operator:**shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of 2.

**>>, Signed Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect to dividing the number with some power of 2.

Example: (-5) >> 1 results in -3 because -5 in binary is 1111 1111 1111 1011, and shifting right by 1 position results in 1111 1111 1111 1101, which represents -3 in two's complement form.

**>>>, Unsigned Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

Example: (-5) >>> 1 results in 2147483645 because -5 in binary is 1111 1111 1111 1011, and shifting right by 1 position results in 0111 1111 1111 1101, which represents 2147483645 in two's complement form.

**instanceof Operator**

The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface. General format-

object **instance of** class/subclass/interface

**E.g**

Person obj1 = **new** Person();

System.out.println("obj1 instanceof Person: " + (obj1 **instanceof** Person));

// Output : true

**Using + over ():**

### When using the + operator inside system.out.println() make sure to do addition using parenthesis. If we write something before doing addition, then string addition takes place, that is, associativity of addition is left to right, and hence integers are added to a string first producing a string, and string objects concatenate when using +. Therefore it can create unwanted results.

E.g

public class operators {

public static void main(String[] args)

{

int x = 5, y = 8;

System.out.println("Concatenation (x+y)= " + x + y);

System.out.println("Addition (x+y) = " + (x + y));

}

}

Output:

Concatenation (x+y)= 58

Addition (x+y) = 13

## Types of Variables in Java

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

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

**Local variables** : Defined inside any block or method or constructor. Scope exists only within the block.

**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.

-  Its default value is dependent on the data type of variable. For *String* it is *null,*for*float*itis*0.0f,*for*int*it is*0,*for Wrapper classes like *Integer* it is *null, etc.*

- They can be accessed only by creating objects.

e.g:

**class** GFG {

    // Declared Instance Variable

**public** String geek;

**public** **int** i;

**public** Integer I;

**public** GFG()

    {

        // Default Constructor

        // initializing Instance Variable

**this**.geek = "Shubham Jain";

    }

    // Main Method

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

    {

        // Object Creation

        GFG name = **new** GFG();

        // Displaying O/P

        System.out.println("Geek name is: " + name.geek);

        System.out.println("Default value for int is "

                           + name.i);

        // toString() called internally

        System.out.println("Default value for Integer is "

                           + name.I);

    }

**Static Variables -**

- static variables are declared using the static keyword within a class outside of any method, constructor, or block.

- 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.

- Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for *Wrapper classes* like *Integer* it is *null,* etc.

- 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. But for accessing the static variable of a different class, we must mention the class name as 2 different classes might have a static variable with the same name.

- Static variables cannot be declared locally inside an instance method.

**E.g**

**class** GFG {

**public** **static** String geek = "Jyotiranjan Kunda";

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

    {

        System.out.println("Geek Name is : " + GFG.geek);

        // static int c=0;

        // This will throw an error as static variables cannot be declared locally.

    }

}

- We can declare class variables anywhere in class, but outside methods.

- Access specified of member variables doesn’t affect scope of them within a class.

- Member variables can be accessed outside a class with following rules

**Modifier Package Subclass World**

public Yes Yes Yes

protected Yes Yes No

Default (no

modifier) Yes No No

private No No No

**Wrapper Classes in Java**

A Wrapper class in Java is a class whose object wraps or contains primitive data types.

char -> Character

int -> Integer

bool -> Boolean

…. and so on

### ****Some important features and function of wrapper class****

### ****1. Autoboxing****

The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.

**char** ch = 'a';

// Autoboxing- primitive to Character object conversion

Character a = ch;

// Normally it should had been done like this : which is called boxing, since java does it automatically, so it is auto-boxing

Character a = new Character(ch);

### ****2. Unboxing****

It is just the reverse process of autoboxing.

Character ch = 'a';

// unboxing - Character object to primitive conversion

**char** a = ch;

// Normally, it should have been done like this :

**char** a = ch.charValue();

// To get any primitive value from its wrapper class, use xxxValue() method, where xxx refers to the primitive data type. It is an non static method, so used with object name.

**3. String to Integer or any other data type**

String str = “123”;

int num = Integer.parseInt(str);

// Now, num becomes an int value.

// Use the function parseXxx function, where Xxx represents the primitive data type

**4.From any data type to string**

Use the String.valueOf() method

E.g: int i = 10;

String s = String.valueOf(i);