**\*Topic 1 : Java Features.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Java came in 1995 ==> 2020 ==> 25 Years.

High level Programming language.

\*The power of Java is coming from the following 12 Key Features :

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1. Simple.**

**2. Platform Independent. --> Important.**

**3. Architecture neutral.**

**4. Portable.**

**5. Security.**

**6. Object Oriented.**

**7. MultiThreaded.**

**8. Robust.**

**9. Distributed.**

**10. Compiled and Interpreted Language.**

**11. High Performance. --> Important.**

**12. Dynamic. --> Important.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1 . SIMPLE :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Nursery level programming language to learn.

All the complex features of languages like C and C++ are replaced like Pointers.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2 . PLATFORM INDEPENDENT :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Write Once Run Anywhere. --> (WORA).

Write java code --> demo.java.

Convert to bytecode --> demo.class. by --> javac demo.java

Send that code to the respective JVM of the OS.

Run the code using --> java demo.

Hence no matter which platform, there is only one java code.

For example consider that I compile a file an Windows OS and then i ran that compiled .class file on the

Linux OS , it will run without any issue because of the JVM of Linux OS.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3 . Architecture Neutral :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

This talks about specific machine.

It basically means that the java code or the .class file does not have any dependency on the version of that particular machine.

For example consider that I have a Windows 32 Bit OS and I want to upgrade it to 64 Bit OS , then I can do it without having to make any changes in my Java Setup.

OR

For example consider that I have a i5 Processor on my OS and I want to upgrade it to i7 , then I can do it without having to make any changes in my Java Setup.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4 . Portable :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

This may sound similar to Platform Independentness , but if we want to migrate a Java Program from say

Windows OS to Linux OS , I don't need to perform any changes in the Java Setup.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**5 . Security :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Step 1 : Write a java code and save it as .java file.

Step 2 : Compile it using java compiler to convert it into .class file.

Step 3 : JVM downloads the .class file which is basically bytecode.

Step 4 : JVM have a component Bytecode Verifier which checks if the .class file contains any kind of virus or malware and reports it using VerifyError. It includes 3 level Sandbox Checking.

Step 5A: After it checks and the .class file is fine it is loaded into the memory of JVM by class loader.

Step 5B: After it checks and the .class file contains some kind of virus or unauthorized activities , it won't let the .class file load into the memory of JVM and thus it won't run.

Step 6 : It is then converted to Native code which is basically machine code and runs it.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**6 . Object Oriented :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

In java almost everything is considered as Object.

Even for conversion of Primitives to Objects we have Wrapper class.

Inheritance. --> Parent components to Child.

Polymorphism. --> Overloading and Overriding.

Encapsulation. --> Getter and Setter methods for private variables.

etc...

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**7 . MultiThreaded :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Imagine if there is a work of 100 hours and 1 person can dedicate 10 hours max.

In languages like C and C++ there is no way of recruiting more than 1 person.

But in java we can hire unlimited number of people for the job to reduce the time.

This features is called Multi Threading. While other languages follow Sequential Execution , Java follows

Parallel Execution.

Thread Class ,

Runnable Interface ,

Callable Interface

are available to support Multi Threading in Java.

By default the flow of Java code is sequential , we can use the concept of Multi Threading to make it

parallel execution.

S1 : JVM will convert sequential execution to parallel execution automatically. [False]

S2 : JVM won't do automatic conversion , we have to write code for parallel execution. [True ]

Thus we can't say that Java have high performance due to Multi Threading.

Just that we have support for Multi Threading in Java.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Note :

When does static methods like sqrt() from Math Class loads ?

-->

Math.sqrt();

Whenever we are using such static method or static variable

Math.class file will be loaded inside JVM memory.

Inside that class if there are any static blocks , they will be executed automatically.

After all this , then sqrt() will be executed.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**8 . Robust :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

The chance of failing of Java program is very less.

1. Type related problems of variables ----> Very Less.

2. Garbage Collector ----> Always free memory is available.

3. Exception Handling ----> try , catch , finally , throws , throw.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**9 . Distributed :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

If the application is distributed across multiple JVMs.

RMI,EJB etc.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**10 . Compiled and Interpreted Language :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

javac Test.java (Compiler) ---> Compilation.

java Test (Interpreter) ---> Execution.

The main advantage is to get all the errors at the time of compilation itself.

If we get all the error at runtime then there will a lot of load on the interpreter and we won't be able to solve those errors.

Java Interpreter is included in JVM.

Compiler ----> All Syntax related issues.

Interpreter ----> Interpret and execute.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**11 . High Performane :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Java performance is less when compared with C and C++ , where as it is very high when compared with other interpreted languages like Pearl and Python.

HOTSPOT JVM ---> 1.2 V.

From 1.3 V bydefault available.

JVM with JIT compiler ---> HOTSPOT JVM.

Many times there is a method which we call again and again.

Imagine if we call a method m1() which contains 1 crore lines of code around 10 times.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

class Test

{

public void m1()

{

//1 crore lines of code.

}

public static void main(String...args)

{

m1();

m1();

m1();

m1();

m1();

m1();

m1();

m1();

m1();

m1();

}

}

Now here there is a count set by the JVM , count of the number of times a method is called.

And there is a threshold value of that count that differs from JVM to JVM , for example 4 or 5.

If that threshold value is reached , JIT compiler stores the interpreted version of that method and

directly uses it the next time that method is called saving a lot of time in future.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**12 . Dynamic :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Java is dynamic in nature that means , a class file in java will be loaded only when required.

class Test

{

public static void main(String...args)

{

Student s = new Student();

Customer c = new Customer();

}

}

Here if we compile this file , Test.class , Student.class and Customer.class file will be compiled.

Then if we run using command Test.java then only at that instance Test.class file will be loaded.

Not Student.class and Customer.class file. Then as the interpreter will start executing the file , it

will reach the line Student s = new Student() and at that instance only the Student.class file will be

loaded into the JVM memory and same will apply for the Customer.class file.

ALL IN ALL WE CAN SAY THAT WHENEVER WE ARE USING A PARTICULAR CLASS THEN ONLY THE CORRESPONDING CLASS

FILE WILL BE LOADED.

For example , if we import java.util.\*;

All the class files inside util package won't be loaded.

But if we call ArrayList al = new ArrayList();

Then the ArrayList.class file will be loaded.

HENCE JAVA IS DYNAMIC AS IT LOADS .CLASS FILES AT RUNTIME RATHER THAN AT COMPILE TIME.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**