History of Java Programming Language :-

Ek software company thi - Sun microsystem. Iske ek brilliant engineer the James Gosling or unki ek choti si team thi.

Unka mission tha : "Ek aisi language banani jo chhoti electronic devices (Jaise TV, remote, etc)pe bhi kaam kar ske , bina kisi dikat ke, har machine pe chale !"

Tab Unhone ek project ka naam diya Project Green

Project Green ka goal tha : "Ek aisi language banana jo portable , secure , robust aur platform-independent ho."

Gosling ne pahle jis programming language ka prototype banaya uska naam rakha Oak.

"Oak" why ?? kyuki unke office ke window ke Samne bara sa Oak tree tha.

Lekin baad mein pata chala ki "Oak" naam already kisi aur company ne use kiya hua tha. Naam copyright me phas gaya!

Phir team me pad gai - ab naya naam kya ho?

Ek din team brainstorming kar rahi thi ek caffee me.

Wahan sab coffee pee rahe the - aur woh caffee thi : Java Coffee - ek popular coffee jo Indonesia ke Java island se aati thi.

Unhe laga: "Java naam catchy hai, cool hai, aur programmers ko coffee toh waise bhi pasand hai!"

Aur bas Java naam final ho gaya.

1995 me Sun Microsystem ne Java 1.0 ko officially launch kiya. Aur salogan diya:

"Write Once, Run Anywhere"

Is feature ne Java ko Super-hit bana diya.

Aur phir uske bad:

- Java pe web apps, mobile apps (Android), enterprises software, sab kuch banana assan ho gaya.
- Java ne phr platform independence, automatic memory management (Garbage Collection) aur security ke features diye.

2010 me Oracle ne Sun Microsystem ko kharid liya. Tabse Java malik Oracle ban gaya.

How Java Code Run ???

Step 1 : Java Code → Byte code

- Java compiler (javac) se tumhare code ko .class file (bytecode) me convert karta hai.
- Byte code = platform-independent (JVM ke liye).

Step 2 : JVM Starts Running Bytecode.

- JVM interpreter pehle bytecode ko line-by-line run karta hai.
- Jab koi method frequently used hota hai, JVM decide karta hai: "Ab isko JIT "(Just-in-Time) compiler jo ki JVM ka hi part hai," se compile karwa lete hain.

Step 3 : JIT Compiler

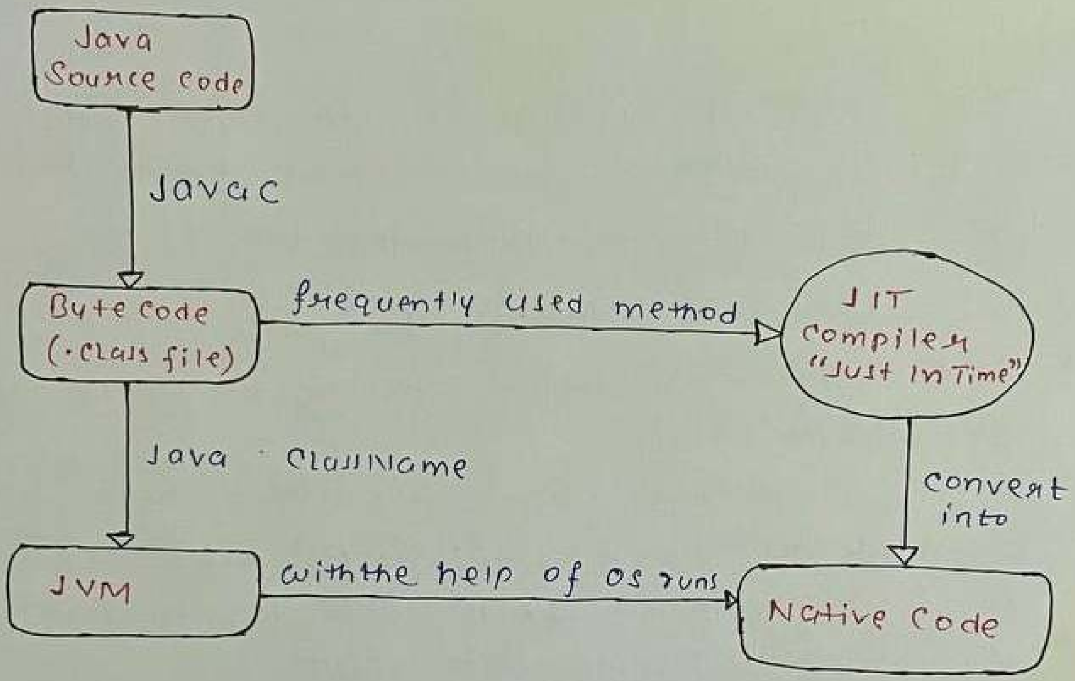
- JIT compiler byte code ko convert karta hai native machine code me (CPU architecture specific).
- Ye native code JVM ke internal memory (cache) me store hota hai.

Step 4 : JVM Execution Decision

- JVM check karta hai, "kya is method ka native version available hai?"
- Agar haan : bytecode ko skip karo.
- Native code ko call karo.

Step 5 : Native Code Execution

- JVM OS/system ke help se us native code ko address CPU ko deta hai.
- CPU us native code ko run karta hai (as binary instructions).



Java Source code
Javac
Byte code (.class file)
frequently used method
JIT compiler "Just In Time"
Java ClassName
convert into
JVM
with the help of OS runs
Native Code

All about JAVA (JVM , JRE and JDK)

JVM (Java Virtual Machine)

Definition: JVM ek virtual machine hai jo Java bytecode ko chalata hai. Har platform ke liye JVM ka alag version hota hai (Windows, Linux, Mac).

- .class file (bytecode) ko padhta hai.
- Bytecode ko machine code me convert karta hai.
- Java program ka execution karta hain.

Internal:

- Class Loader : .class files ko load karta hai.
- Bytecode Verifier : code secure hai ya nahi, check karta hai.
- Interpreter : Byte code ko line-by-line machine code me convert karta hai.
- Garbage Collector : Unused Memory ko saaf karta hai.

JRE (Java Runtime Enviornment)

Definition: JRE JVM ka ek package hai jisme JVM + liberaries + supporting files hoti hai, taki Java Program run ho sake.

components:

- JVM
- Java Class libraries (e.g. java.lang , java.util)
- Other Supporting files (config files, property files)

Use:

- Java application ko run karne ke liye JRE chahiye.
- Java code likhne ke liye JRE se kaam nahi chalega - uske liye JDK chahiye.

JDK (Java Development kit)

Definition: JDK ek full package hai jisme java code likhne, compile karne, aur run karne ka saare tools hote hote hai.

Components:

- JRE (JVM + libraries)
- Javac (Java Compiler) -> .java -> .class
- Tools like debugger, documentation generator, etc.

Use:

- Java application banane ke liye JDK chahiye.
- JDK me hi javac, java, aur javadoc jaise commands available hote hai.

mere Dimage me ek sawal aya!!!
JIT bhi ek compiler hai aur JDK bhi code ko compile karta hai, too kya dono Same hai ya alag ???

Difference Between JDK and JIT compiler.

JDK compiler	JIT compiler.
• Ye ek Java source code compiler hai.	• Ye ek runtime compiler hai.
• Part of JDK.	• Part of JVM.
working:	working:
• ye .java file ko compile karta hai. • Output deta hai .class file (bytecode). • javac command ko yhi execute karta hai.	• Jab JVM bytecode (.class) file ko run karta hai, to wo code ko line-by-line interp-ret karta hai. • Lekin agar koi part of code bar-bar run ho raha ho, to JIT compiler us part ko machine code me convert kar deta hai - taki wo fast chale.
	Purpose: To make java Prog-ram fast at runtime.

JAVA is platform dependent or platform independent ???

*** Java is platform-independent at the source level and byte code level, but platform-dependent at the JVM level.

Step-by-step Flow :

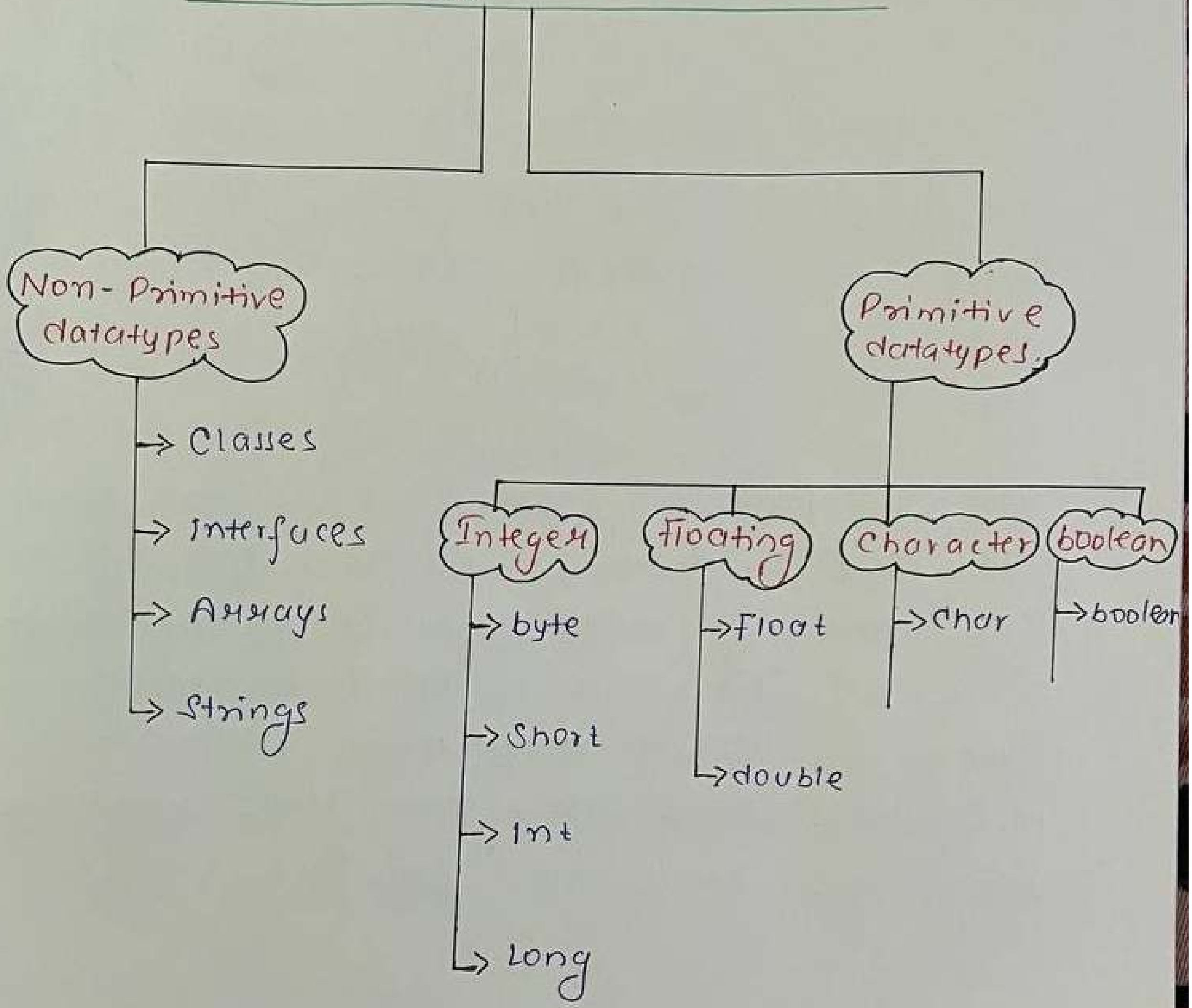
- Java code (.java) likhte ho.
- javac se compile karte ho -> .Class file (bytecode) banta hai.
- Bytecode ko JVM samajhta hai.
- JVM us bytecode ko machine code me convert karta hai (using JIT).
- Machine code ko Os + CPU execute karta hai.

Ab samjho bytecode yani .class file jo hai wo koi bhi computer pe chal sakta hai chahe koi bhi OS ho (windows, mac, linux) bas usme JVM hona chahiye, isiliye Java ko platform independent bolte hai.

Lekin JVM khud ek software hai, jo har operating system ke liye alag alag hota hai isliye JVM ko platform dependent bolte hai.

conclusion, Java ka bytecode platform independent hota hai lekin jo usko samajhta hai JVM wo platform dependent hota hai.

Datatypes in Java



Primitive data-types : Primitive datatypes Java ke basic built in datatypes hote hai jaise int, float, boolean etc, jo ek simple value store karte hai aur inke pass koi methods nahi hote hai.

Non-Primitive datatypes : Non-primitive datatypes Object-based hote hai jaise String, Array or custom classes. Ye reference store krte hai, inko object heap me store hota hai, ye null bhi ho skte hai aur ye methods ke sath aate hai.

" Java me primitive data types Stact memory me directly store hote hai, kyuki ye lightweight aur fixed size ka hota hai.
Non-primitive data types jaise String, Array, Class unka reference stack me store hota hai aur actual object heap memory me store hota hai."

" Java me primitive data types ka type aur size compile-time pe fix hota hai, lekin unki memory allocation stack me run-time pe hota hai. Non-primitive data types ka object heap mein run-time pe create hota hai, aur unka reference stack me store hota hai."