Java : features

- fast, Secure, General Purpose
- Easy to understand
- Platform independent

- architectural neutral (datatype size same for all)
- distributed, dynamic, high perform

Program which are written in java, after compilation are converted into bytecode which is a part of java platform irrespective of m/c on which program run.
& this makes java highly portable as its bytecode can be run on any m/c by an interpreter called JVM.

- Java is an OOPL
- Java is Multithreaded language.

Applications:

- Desktop appl^n (stand-alone appl^n) ex. antivirus, VLC media player
- Enterprise { distributed in nature, banking, load balancing, clustering, high level security, EJB (Enterprise Java Beans), Scientific appl^n.
- Web based (client Server architecture), Servlet, JSP, Struds, Spring, Hybernet, JSF, etc.
- Mobile appl^n, Gaming appl^n - Micro editions of java, Java ME

JDK : Java Development Kit

- It includes mainly two things

1. Development tools (which is used to provide an environment to develop our java program)
2. JRE (to execute java program)

JRE : Java Runtime Environment

it is an installation package that provides an environment to run java program.

JVM : Java Virtual Machine

Whatever java program we run using JRE or JDK goes into JVM & JVM is responsible for executing java program line by line & hence it is also known as interpreter.

Components of JRE -

1. Deployment technologies (deployment, Java web start)
2. UI toolkit (Abstract Window toolkit), swing)
3. Integration libraries - JDBC
4. lang & util based libraries
5. JVM, JavaHotSpot client. Server VM

JIT Compiler

- Just In Time
- JIT is a part of JVM
- It compiles bytecodes to m/c code at run time
- Very frequently used methods are compiled once & kept into memory
- It performs some optimization on code

JVM vs JIT

while both JVM & JIT are part of Java platform but the key diff^{n} bet^{n} them is that JVM is an interpreter while JIT is a compiler.

JVM is mandatory to run a java program as it convert bytecode to m/c code

JIT comes next to improve performance by replacing java byte code with m/c code.

class Loader

Java classloader is a part of JRE that dynamically loads Java classes into JVM.

Java runtime system does not need to know about files and file system bc2 of classloaders.

Java classes are not loaded into memory all at once, but when req by app^{n} at this point, Javaclassloader is called by JRE & these classLoader load classes into memory dynamically.

① Bootstrap class loader : m/c code that starts oper^{n} when JVM calls it. It loads core java classes from rt.jar file

② Extension class loader

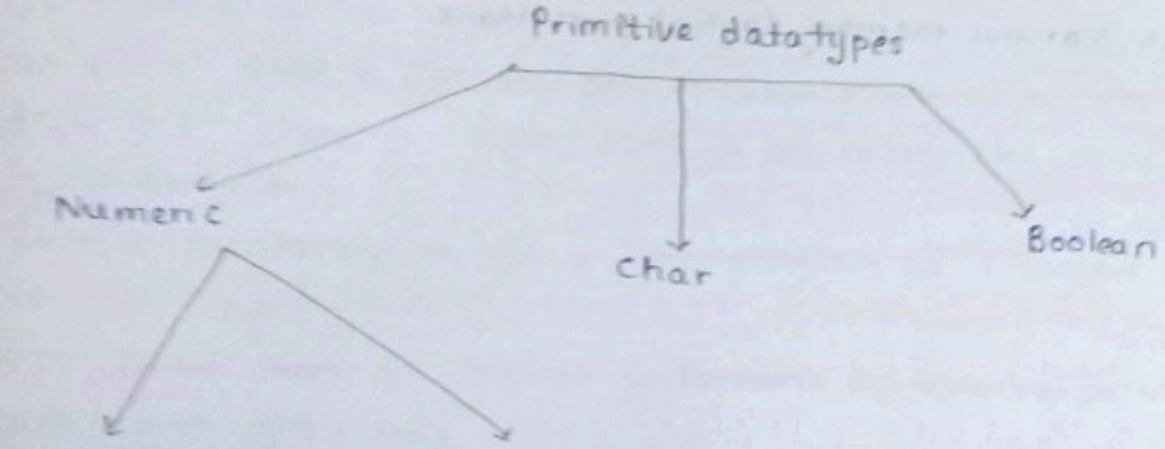
③ System class loader

• What gives java its 'Write Once & run anywhere" nature

→ bcz of bytecode. java compiler converts java programs into class file i.e. bytecode which is the intermediate language bet" source code & mlc code
This bytecode is not platform specific & can be executed on any computer.

- History of Java. who invented Java
  - developed by James Gosling at Sun Microsystems (1991)
  - Previous name - Oak (tree - symbol of strength & unity) but trademark of another company.

- Various datatype in Java



- diff^ bet^

  System.out.print : print content without switching to next line

  System.out.println : print content & switch to next line.

  System.err.print : used to output error

• What is bytecode? How is it diff^{t} from m/c code?

→ Byte code is an intermediate code bet^{n} the source code & m/c code. It is a low-level code that is the result of the compilation of a source code which is written in a high level language.

Bytecode is a non-runnable code after it is translated by an interpreter into m/c code then it is understandable by the m/c. It is compiled to run on JVM, any system having JVM can run it irrespective of their OS. That's why Java is platform-independent. Bytecode is referred to as a Portable code.

→ M/c code is set of instructions that is directly m/c understandable & it is processed by CPU.
M/c code is in binary format which is completely diff^{t} from byte code & source code.

• diff^{n} bet^{n} Jar file & runnable jar file?

JAR file is a Java appl^{n} which requires a command line to run. & runnable JAR file can be directly executed by clicking it.

[JAR file is a package file - used to aggregate many java class files & resources - built on zip format - .jar]

[Runnable JAR - use to run java classes without having to know class names & type them in cmd].

- diff^n bet^n Runnable jar & exe file ?

main diff^n bet^n runnable jar & exe file is the platform on which they can be executed.
runnable jar file can be executed on any platform that has a JVM installed, including windows, Mac, Linux whereas exe file can be executed on windows OS

- How is C platform dependent language.

In C, m/c code is diff^t for diff^t processor architecture, & thus could not run natively on incompatible platforms
(C produces m/c code which is platform dependent)

- diff^n bet^n path & class path ?

path - environment variable that is used to find & locate binary files like "java" & "javac" & to locate needed executables from cmd line / Terminal window
(used by OS to find exe files)

classpath - env. var. is used by Java compiler to find path of classes.

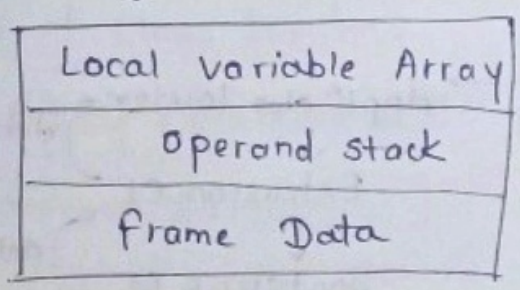
Heap Area

- will be created at time of JVM startup.
- obj on corresponding instance var. will be stored in Heap Area
- Arrays
- can be accessed by multiple threads. → data stored in Heap Memory is not thread safe.
- Need not be continuous.

```
Runtime r = Runtime.getRuntime();
r.maxMemory(), r.initialMemory(), r.freeMemory()
```

Stack Memory

Each n Every method call perform by thread will be stored in stack including local var. also.
after completing method corresponding entry from stack will be removed.
Each entry – stack frame / activation record.



Stack Frome

Catch block (Exception)

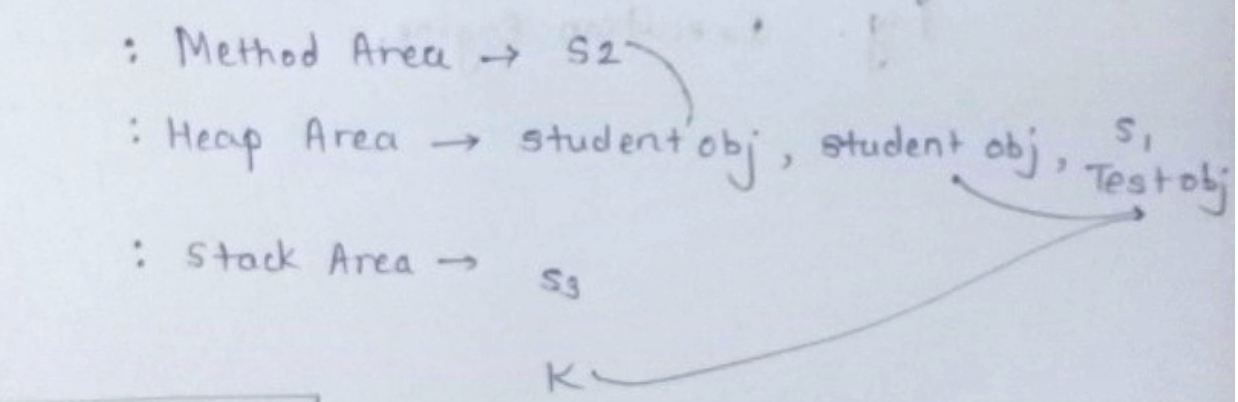
PC Registers

For every thread separat PC reg will be created at time of thread created.
Contains addr of current exe. instⁿ.

Native Method stacks

- Hashcode method

```
class Test {
    Student s1 = new student();
    Static student s2 = new student();
    P.s.v.m(s.arg[]) {
        Test t = new Test();
        Student s3 = new student();
    }
}
```



Execution Engine

- mainly contains 2 components : 1. Interpreter
  2. JIT compiler
     ↳ purpose : improve performance, maintains count of every methods

whenever JVM come across any method call 1st that method will be interpreted normally by interpreter & JIT compiler ↑ments corresponding count var