***History of Java:-***

1. *Java was developed by* ***James Gosling*** *at* ***Sun Microsystems in 1996.***
2. *It is commonly used to develop web applications, apps, enterprise softwares,etc*
3. *Java name was derived from* ***Coffee Bean****.*
4. *It was initially called as* ***‘Oak’*** *later renamed as* ***“Java”.***
5. *It is popular, high level, object oriented programming language known for its versatility and portability across all platforms.*
6. *Currently JAVA is owned by* ***Oracle Systems.***

***Features Of Java with Real-Life Examples:-***

1. ***OOP’s Concepts:-***
   1. *Java follows OOP’s paradigm which means* ***everything is treated as Object*** *and software is designed using Classes and Objects.*

***Real-Life Examples:-***  *Consider a library management systems. Each book is an object, with attributes like title, author, . The library system manages these objects and performs operations like lending & returning a book.*

1. ***Platform Independence :****-*
   1. *Java code is compiled into bytecode that runs on any platform with JVM, making JAVA programs portable across different OS like Windows, Linux, macOS.*

***Real-Life Examples:-*** *If we develop a mobile app in Java for android, that app can run on any android phone ,regardless of manufacturer OS or hardware configurations.*

1. ***Automatic Garbage Management (Garbage Collection):-***
   1. *Java does automatically handles memory management by removing unused objects through garbage collection. This helps to prevent memory leaks.*

***Real-Life Examples:-*** *In Banking application when users complete transactions the system automatically cleans up memory used by completed transaction objects, ensuring the system continues to perform well as more transactions are processed.*

1. ***Multithreading:-*** 
   1. *Java supports multithreading , which allows multiple threads to run concurrently, enabling CPU better utilization and smoother performance.*

***Real-Life Examples:-*** *In Online gaming application, multiple players actions can be handled by different threads, allowing the game to process all players movements simultaneously without lagging.*

1. ***Robust and Secure :-***
   1. *Java offers secure and strong memory management and exception handling, as it provides some features as data abstraction as well.*

***Real-Life Examples:-*** *In online banking application java’s security features ensure that users personal and transaction data remains safe by preventing unauthorized access to code’s sensitive information.*

1. ***High performance:-***
   1. *While java is interpreted via(JVM), it also supports Just-In-Time(JIT) compilation, which improves performance by compiling bytecode to native machine code during runtime.*

***Real-Life Examples:-***

1. *Stock Trading System.*
2. ***Exception Handling:-*** 
   1. *Java has a robust exception handling mechanism that ensures a program can handle errors gracefully without crashing.*

***Real-Life Examples****:- In ticket booking system, if user tries to book seat that is already taken, then JAVA’s exception handling will catch that error and notify the user without terminating entire application.*

1. ***Rich Standard Library:-*** 
   1. *Java provides a rich set of libraries (API’s) that include the utilities for file handling, networking, database, connectivity, GUI and much more.*

***Real-Life examples:-*** *In e-commerce applications we can use libraries to handle user authentication, database transactions and secured payments without need to write manual code for the applications.*

***Java program code has 5 main blocks:-***

1. *Main Method*
2. *Methods*
3. *Default Constructors*
4. *Static blocks*
5. *Non-Static Blocks*

***Variables in Java:-***

1. *Variables are used to store data values that can be referenced and manipulated throughout your program.*
2. *Each variable has a type (which determines what kind of data it can hold) and a name (used to reference it in the code).*

***There are three main types of variables:-***

* 1. ***Local Variable****:- It is written inside class and declared inside methods, constructors, or blocks, and have method or block scope. Scope of local Variable is limited to that specific method, constructor, block.*
  2. ***Global Variable / Instance Variable****:- Declared inside a class but outside methods, constructors, blocks they are specific to each object (instance) of the class.*
  3. ***Class Variables (Static Variables):*** *Declared with the static keyword, shared by all objects of the class. These variables are common for entire class.*

***Notes:-***

* *Java is a case-sensitive Language*
* *Class Name, interface Name, etc always written in ‘Pascal’ standard.*
* *Variables & Methods always written in ‘camelCase’ standard.*
* *Same name variables are not allowed in same method.*
* *A class can have same named variable but one should be local and other should global, in this case if the variable is called inside method it will always give preference to its own variable and if the variable is called outside method then the global variable will get called.*
* *Command to compile multiple files in folder-> javac \*.java*
* *Command to open existing java classes created and displays all methods and fields present in that class -> javap ClassName*
* *Command to open specific method of a class -> javap -c ClassName methodName*
* *In Eclipse* ***.java files stored in -> src folder*** *and* ***.CLASS files stored in -> bin folder src folder has all source code files and bin folder has all compiled binary code class files.***

|  |  |  |
| --- | --- | --- |
| ***Comparison Index*** | ***C++*** | ***Java*** |
| ***Platform-independent*** | ***C++ is platform-dependent.*** | ***Java is platform-independent.*** |
| ***Mainly used for*** | ***C++ is mainly used for system programming.*** | ***Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.*** |
| ***Design Goal*** | ***C++ was designed for systems and applications programming. It was an extension of the***[***C programming language***](https://www.javatpoint.com/c-programming-language-tutorial)***.*** | ***Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.*** |
| ***Multiple inheritance*** | ***C++ supports multiple inheritance.*** | ***Java doesn't support multiple inheritance through class. It can be achieved by using***[***interfaces in java***](https://www.javatpoint.com/interface-in-java)***.*** |
| ***Operator Overloading*** | ***C++ supports***[***operator overloading***](https://www.javatpoint.com/cpp-overloading)***.*** | ***Java doesn't support operator overloading.*** |
| ***Pointers*** | ***C++ supports***[***pointers***](https://www.javatpoint.com/cpp-pointers)***. You can write a pointer program in C++.*** | ***Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.*** |
| ***Compiler and Interpreter*** | ***C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.*** | ***Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.*** |
| ***Thread Support*** | ***C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.*** | ***Java has built-in***[***thread***](https://www.javatpoint.com/multithreading-in-java)***support.*** |
| ***Documentation comment*** | ***C++ doesn't support documentation comments.*** | ***Java supports documentation comment (/\*\* ... \*/) to create documentation for java source code.*** |
| ***Inheritance Tree*** | ***C++ always creates a new inheritance tree.*** | ***Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the***[***inheritance***](https://www.javatpoint.com/inheritance-in-java)***tree in java.*** |

***Q1) Can we have save java file with different name?***

* ***Yes we can have save java file with ay name only if there is no public class present in that file, if public class is there then we need to save file with that name only.***

***Package & Import :-***

1. *Package is nothing but a folder or directory in java which is similar to folder in our Operating systems.*
2. *Package is declared using keyword* ***package***
3. ***import*** *keyword is used to use data of other class package in current working class.*
4. ***com****. Is mostly used package name where com refers to a Company.*
5. ***package com.ntrs, com.infosys, com.cognizant***
6. *Two or more classes within same package doesn’t need import keyword.*
7. *If we want to use classes from another package then only we need to use import keyword.*
8. *If file has multiple import (more than 5) of same file then IDE automatically changes to* ***\****
9. *Package arranges number of classes, interfaces and sub-packages of same type in particular group/ folder/directory.*
10. *It is a type of file folder containing same type of files.*
11. *If we want to share codes then we just need to share binary code package to other people and then they can run it without having source code.*

***How to create package using Notepad :-***

***javac -d . fileName.java***

***java filename.className***

***eg:-***

***1.Compile the Welcome.java file:***

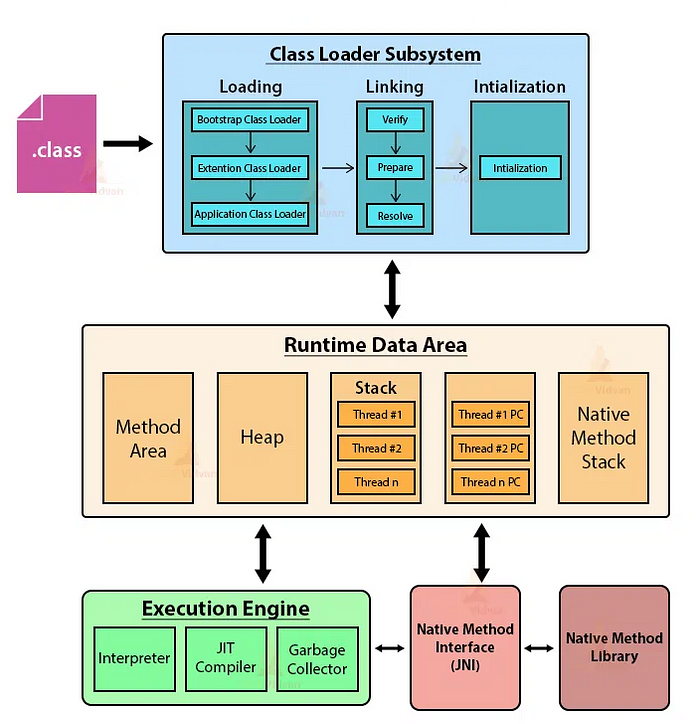
***Command: javac Welcome.java***

***2. This command creates a Welcome.class file. To place the class file in the appropriate package directory, use:***

***Command: javac -d . Welcome.java***

***3. This command will create a new folder called FirstPackage. To run the class, use:***

***Command: java FirstPackage.Welcome***



***JVM Architecture:-***

* *It is a software simultaion of machine which can perform operations like a physical machine.*
* *Two types of Virtual Machines:-*
  + *Hardware based VM 🡪 VMWare*
  + *Application based VM 🡪These VM act as runtime engines to run a particular programming language applications.*
    - *JVM(Java virtual machine acts as runtime engine to run Java based application)*
* *JVM is part of JRE and it is* ***responsible to load and run java .class files***

***Class Loaders Subsystem:-***

1. *Class loaders of system are responsible for following threee activities:-*
2. ***Loading****: Loading refers to* ***reading .class*** *and* ***store corresponding binary data in Method area****.For each .class file JVM will store corresponding information in Method area.*
   * + *Fully qualified name of class and immediate parent class.*
     + *Methods, variables, constructors, modifiers, constatnt pool information.*

*After loading .class file* ***JVM creates an object for that class in Heap memory*** *of type java.lang.Class. class Class object can be used by programmer to get Class-level information like method,variables,constructors information.*

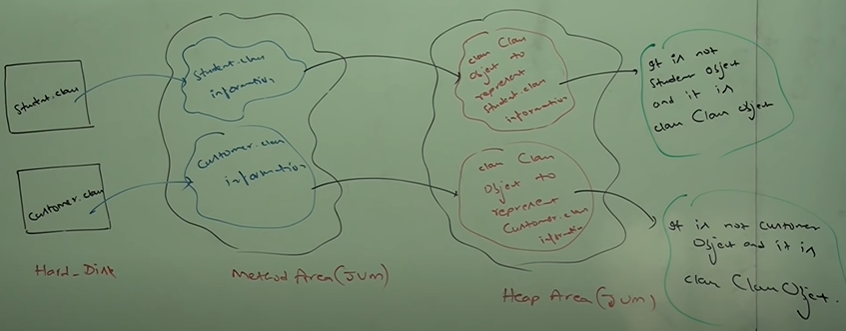
***Class c = Class.forName(“Student”); 🡪To get class-levelinformation***

*Method[] m=c.getDeclaredMethods(); //Method class from* ***java.lang.reflect;***

*for( Method m1 : m){*

*System.out.println(m1.getName()); }*

*For every loaded type only one class object will be created , eventhough we are using Class multiple times in our program.*



1. ***Linking:*** *It consists 3 activities:-*

***1)Verify:-*** *It is the process ensuring binary representation of class is struturally correct or not i.e JVM will check whether .class file is generated by valid compiler or not and .class is properly formatted or not. Internally bytecode verifier is responsible for this activity. Bytecode verifier is part of class Loader subsystem . If verification fails then RuntimeException saying* ***java.lang.verifyError***

***2) Prepare:*** *In this phase JVM will allocate memory for class level static variables and assign default values. Here only default values will be assigned according to data type.*

***Note:-***  *In initilization phase original values will be assigned to static variables.*

***3) Resolve:*** *It is the process of replacing symbolic names in our program with original memory references from Method area.*

*class* ***Test*** *{*

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

***String*** *str=new String(“Yash”);*

***Student*** *s1= new Student(); }*

*For above class class loader loads Test.class, String.class, Student.class and Object.class.*

***The names of these classes are stored in constant pool of Test class. In resolution phase these names are replaced with original memory -level references from Method area.***

1. ***Initialization :-*** *In this all static variables are assigned with original values and static blocks will be executed from Parent -> Child (Top->Bottom)*

***Note:-***

* ***While loading , linking, Initilaization if any error occurs then we will get Runtime exception saying LinkageError.***

***Types of ClassLoaders:-***

*There are 3 types of Class Loaders*

1. ***Bootstrap Class Loader: (****jdk/jre/lib/rt.jar)*

* *It is responsible to load core Java API classes i.e classes present in rt.jar..*
* *The location is called Bootstrap classpath. It is written in native languages.( C, C++)*

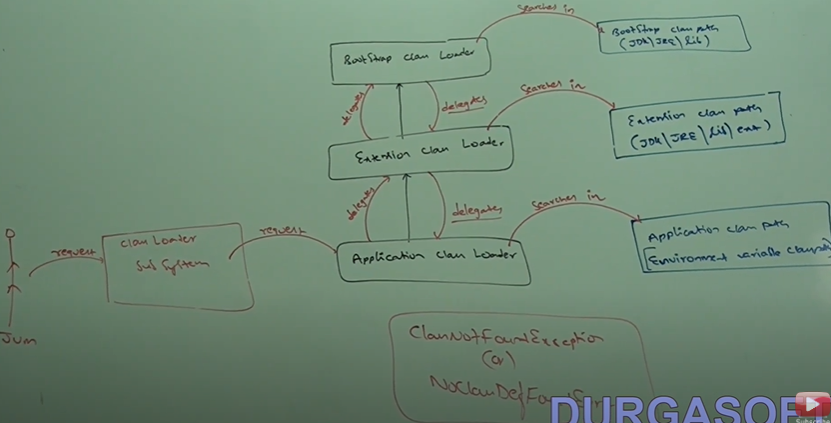
1. ***Extension Class Loader: (****jdk/jre/lib/ext/\*.jar)*

* *It is child class of Bootstrap class loader.*
* *It is responsible to load classes from extension classpath*
* *It is implemented in java and corresponding .class file is->sun.misc.Launcher$ExtClassLoader.class ( $ represents Inner static class)*

1. ***Application Class Loader***

* *It child class of extension class loader*
* *It is responsible to load classes from application classpath*
* *It internally uses environment variable classpath*
* *It is implemented in java and corresponding .class file is->sun.misc.Launcher$AppClassLoader.class ( $ represents Inner static class)*

***Internal Working of Class Loader:***



1. *Class loader follows delegation-hierarchy principle.*
2. *Whenever JVM come across a particular class, first it will check whther the .class is already loaded or not. If already loaded in method area then JVM will consider that loaded class.*
3. *If it is not loaded then JVM request class loader sub system to load particular class.Then class loaded subsystem handover the request to application class loader.*
4. *Application class loader delegates the request to extension class loader which in turn delegates request to Bootstrap class loader.*
5. *Then bootstrap class loader will fetch in Bootstrap classpath. If it is available then corresponding .class will be loaded by Bootstrap class loader.*
6. *If not availabe then bootstrap loader delegates the request to extension class loader.*
7. *Extension class loader will search in extension class path , if available then will be loaded and if not then delegates the request to Application loader.*
8. *Application class loader will search in application class path, if available then will be loaded and if not then will throw* ***RE:NoClassDefFoundError or ClassNotFoundException.***

*Note:-*

* *We can use* ***.getClassLoader()*** *to fetch the mentioned class present in which ClassLoader*
* *String.class.getClassLoader() 🡪 null (BecauseBootstrap is not Java object)*
* *Student.class.getClassLoader() 🡪 sun.misc.Launcher$AppClassLoader.class*
* *Customer.class. getClassLoader() 🡪 sun.misc.Launcher$ExtClassLoader.class*

***How to create customised ClassLoaders:-***

* *We will create one Class first lets say (CustomClassLoader) which extends ClassLoader*

*and then we will override .load() method of ClassLoader class and provide functioncality for that .load().*

***JVM Memory management:-***

*Whenever jvm loads and runs a java program it needs memory to store several things like bytcode, objects, variables ,etc. Total jvm memory organised in 5 categories:-*

1. ***Method Area***
2. ***Heap Area***
3. ***Stack Area***
4. ***PC Registers***
5. ***Native methods stacks***
6. ***Method Area:-***

* *For every JVM one Method Area will be available.*
* *Method area will be created at the time of JVM startup.*
* ***Inside method area , class-level binary data including static variables will be stored.***
* ***Constant pools of a class will be stored inside method area.***
* *Method area can be accessed by multiple threads simultaneously, so it is not thread-safe.*

1. ***Heap Area:-***

* *For every JVM one Heap area is available.*
* *It will be created at the time of JVM startup.*
* ***Objects and instanceVariables will be stored in Heap area.***
* ***Every array in java is object only , hence arrays also will be stored in Heap Area. It can accessed by multiple threads so data stored in Heap area is not Thread-safe.***
* *Basically whatever we create using new keyword will be stored in Heap Area only.*
* *Heap memory is finite memory but on our requirement we can set Min and Max Heap Size.*

***Program to display Mmeory statistics of JVM:-***

*Runtime r=Runtime.getRuntime(); //java.lang //Singleton class*

*r.****maxMemory****();*

*r.****totalMemory****();*

*r.****freeMemory****();*

*----------------------------Java command prompt----------------------------------------*

*java -Xmx512m \_className\_ 🡪 Sets Max memory to 512 MB*

*java -Xms64m \_className\_ 🡪 Sets Min memory to 64 MB*

1. ***Stack Memory Area:-***

* *For every thread JVM will create a separate stack at the time of Thread creation.* ***Each and every method call performed by that thread will be stored in Stack. Including localVariables also.***
* *After completing the method the corresponding entry from stack will be removed.*
* *After completing all methods calls the Stack will become empty and that empty stack will be destroyed by JVM. Just before terminating the thread.*
* *Main() ->m1()-> m2() Stack trace*
* *This data is thread safe as the corresponding thread only can access the above Stack trace.*

1. ***PC (Program Counter) Registers:-***
   * *For every thread a separate PC Register will be created at time of thread creation.*
   * ***PC Registers contains address of current executing instruction.***
   * ***Once instruction execution completes automatically PC Register will be incremented to hold address of next instruction.***
2. ***Native Method Stack:-***

* *For every thread JVM will create separate native method stack.*
* ***All native method calls invoked by Thread will be stored in corresponding native method stack.*** *Like hashcode(), wait()*

***For Every JVM execution:-***

1. *1 Heap Area (instance variables/ Globally created objects)*
2. *1 Method Area (static variables)*

***For Every Thread:-***

1. *1 Stack Area / Thread (Local variables)*
2. *1 PC Register / Thread*
3. *1 Native Method Stack / Thread*

***Execution Engines:-***

* *This is central component of JVM. It is responsible to execute java .class files*
* *It mainly contains 2 components :-*
  + *Interpreter*
  + *JIT Compiler*
  + *Garbage Collector(GC)*

1. ***Interpreter:-***

* *It is responsibleto read bytecode and interpret into machine code(binary code) and execute that line-by-line.*
* *The problem with interpreter is , it interprets everytime even same method invoked multiple times,* ***that reduces performance of system****.*
* *To overcome this problem* ***sun people introduced JIT Compiler in 1.1 version***

1. ***JIT Compiler:-***

* *The main purpose of JIT Compiler is to improve performance.*
* *Internally JIT Compiler maintains separate count for every method.*
* *Whenever JVM come across any method call first the method will be interpreted normally by interpreter and then JIT Compiler implements corresponding count variable.This process will continue for every method.*
* *Once if any method count reaches a threshhold value, then JIT compiler identifies that method is repeatedly used method. Such methods are called hotspots.*
* *Immediately JIT compiler compiles that method and generates corresponding native code.*
* *So next time JVM comes across that method call , so then JVM uses that native code directly instead of interpreting again and again.*
* *So that performance of system will be improved. Threshhold count varies from JVM* ***to JVM.***
* *Jit compilation is applicable only for repeatedly required methods and not for every method.*

***Note:***

*Internally* ***profiler (part of JVM)*** *is responsible to identify hotspots( Repeatedly used methods)*