Java-Day-1[Java Core and Basics]:

Annotations

Annotation	Full-Form
JVM	Java Virtual machine
JRE	Java Runtime Environment
JIT	Just In Time Compiler
OOP	Object Oriented Programming
JDK	Java Development Kit

1) What is Java?
    1) Java is a high-level, Object-oriented, platform independent, robust and secure language
    2) Developed in 1995 by Sun Microsystems
2) Java Plaforms:
    1) Java Standard Edition:
        1) It is used for developing desktop and console based applications
    2) Java Enterprise Edition:
        1) It is used when we are developing server side applications
    3) Java Micro-Edition:
        1) It enables to build Java application for micro-devices such as mobile phones
3) Features;
    1) Object-Oriented:
        1) Everything in java is an object
        2) OOPs is a methodology that simplifies s/w development
            1) Object
            2) Class
            3) Inheritance
            4) Polymorphism
            5) Abstraction
            6) Encapsulation
    2) Simple:
        1) Java is very easy to learn, syntax is simple, clean and Easy to understand
    3) Secured:
        1) No explicit pointer
        2) Uses its own runtime environment runs inside JVM(Java Virtual Machine)
    4) Platform Independent:

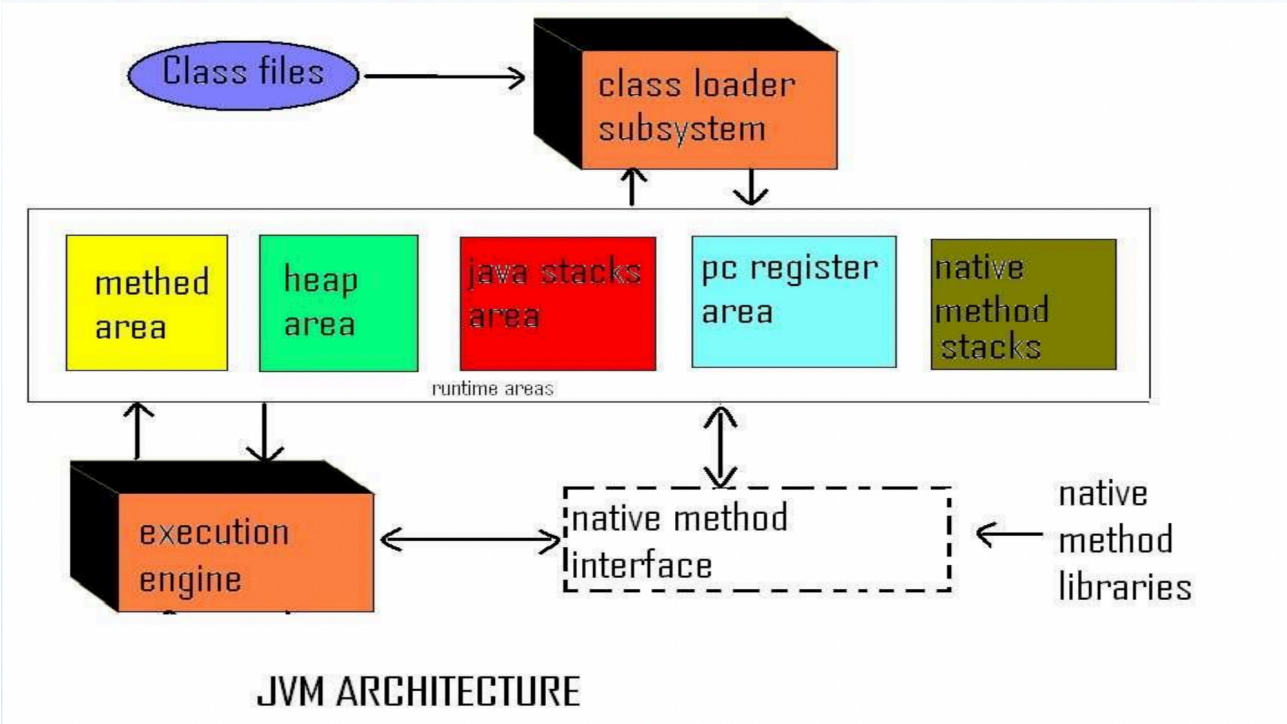
1) Java unlike others is not compiled into platform specific machines, it gets compiled to a .class(byte code -> combinations of 0s and 1s and is platform independent) file that can be shared across and run anywhere

5) Robust:
   1) Strong memory Management
   2) No Pointers-> better security-> we cannot directly access the address space
6) Portable:
   1) .class file/bytecode can be shared and run anywhere
7) Architecture neutral:
   1) There are no implementation dependent features eg: size of primitive types is fixed
8) Dynamic:
   1) Classes get loaded in demand I.e. as and when required
   2) It supports features or functions of its native language C, C++
9) Interpreted:
   1) It is compiled language
   2) Instead of compiling directly to a executable machine code it gets compiled to JVM bytecode(.class file)
   3) This JVM byte code is the compiled/interpreted
10) High Performance:
   1) Java is faster
   2) Because -> java byte code is very close to native machine code
11) Multi-Threaded:
   1) Important for multi-media and web-applications
12) Distributed:
   1) Allows us to create distributed applications
   2) EJB and RMI are used

4) Java Architecture:
   1) JDK -> Developing Tools {Compiler->javac, application-launcher-> java.exe}, JRE{ -> imports, JVM}
   2) JDK v/s JRE

JDK	JRE
Bundle of software that is used to develop java based application	Implementation of virtual machine and it actually executes the java programs
It is needed for developing java applications	Plug-in needed to run java application
It needs more disk space, it contains JRE along with various development tools	Is smaller than JDK, so it needs less space

3) JVM
    1) Working



2) Tasks:
    1) Loads the code:
        1) Loading:
            1) The class loader reads the .class file, generates the corresponding binary data and save it in method area
    2) Verifies the code:
        1) Linking:
            1) It ensures the correctness of .class file, i.e. it checks whether this file is properly formatted and generated by a valid compiler or not
    3) Executes the code:
        1) The byte code which is assigned to the Runtime Data area, will be executed by the execution engine
        2) The execution engine will read the byte code and execute it piece by piece
3) Stack memory:
    1) Contains methods, local variables, reference variables
    2) Stack memory size is smaller than heap memory size in Java

3) Follows LIFO(Last-In-First-Out)
4) Heap memory:
1) Objects(global-access and can be accessed from anywhere)
2) May store reference variables
3) Instance variables are created inside the heap
4) Garbage collection runs on heap memory:
1) Garbage Collector:
1) Collects and removes unreferenced objects
2) System.gc() -> it does not guarantees the execution
3) Thread -> Daemon Thread -> runs continuously in the background
4) Main Objective -> free heap memory -> by destroying unreachable objects
5) 4 possible Ways:
1) By nullifying the reference variables
2) By reassigning the reference variables
3) By creating objects inside method
4) Island of isolation
6) Finalisation:
1) Garbage collector calls the **finalize()** method to perform cleanup activities
2) Once **finalize()** method completes its execution the garbage collector destroys the object
3) We can override **finalize()** method and we can make it work according to our needs

5) Java Basics:
1) Commenting:
1) Single Line -> //
2) Multiple Line -> /**/
3) Java Docs -> Documentation -> right-click on method name -> source-> generate element components
OR
alt-shift-J

6) Package:
1) Is a collection of related classes
2) It helps in organising the classes into a folder structure and makes it easy for us to locate and use them
3) Syntax:
1) package <<package-name>>;
4) Default package -> Lang
5) To call a class from a package:
1) **import** com.example.basic.comments.Comments;

7) Variables:
1) Instance Variables:
1) The variables declared inside the class but outside the method

2) Class/static variables:
    1) Declared as static inside the class
3) Local variables :
    1) Declared inside methods/constructors
4) Reference Variables:
    1) Refers to a class and declared during class instantiation
    2) Can be used to access instance variables and methods

8) Coding Standards:
    1) Identifier naming and capitalisation:
        1) Use descriptive names for variables, constants and methods
        2) Use single letter identifiers if you want to use in loops
        3) Variable names should always start with small letters
        4) Multi-word identifiers are internally capitalised: camel-casing
            1) Eg: sampleAge
        5) Do not use hyphens(-) or underscores(_) in naming variables that are not final
            1) Eg:
                1) private int sampleAge -> correct
                2) private int sample_age -> incorrect
        6) When we are naming final variables then all the letters in the variable name should be capital and separated by underscores(_)
            1) Eg:
                1) final int DAYS_IN_YEAR=365;
9) Data Types:
    1) Primitive:
        1) Integer{int}, Long{long}, Double{double}, Float{float}, Character{char}, Boolean{boolean}
        2) Integer, Long, Double -> Wrapper classes -> wrap the primitive data-types eg: Integer wraps in int
    2) Non-primitive:
        1) Classes, String, Interfaces, Arrays, List etc.
10) Access Specifiers:
    1) Private: within the same class
    2) Default: within same package
    3) Protected: Within same package and child class
    4) Public: Anywhere
11) Control Statements:
    1) Conditional:
        1) If, else if, else
        2) Switch case
    2) Break
    3) Continue
    4) Looping
        1) For
        2) While

3) Do-While: it will run at least once because the control is passed first through the do condition and the the main condition is checked in the while loop

NOTES:

1) **Interpreter:**
    1) **Interprets the byte code faster but executes slowly?**
        1) **Because when one method is called multiple times, every time a new interpretation is required**
    2) **Solution?**
        1) **JIT compiler:**
            1) **The execution engine will be using the help of interpreter in converting to byte code, but when it finds repeated code it used JIT compiler**
            2) **JIT compiler compiles the entire byte code and changes it into native code**
            3) **The above created native code will be used directly for repeated methods call**