Friday, 2 October 2020

Java - Overview

Java

- Object Oriented Programming Language.
- Simple : No struct, pointer(explicitly), operator overloading; concise
- Object Oriented : emphasis on data(object) and interface(method)
- Interpreted And Portable : execute byte code on any machine on which interpreted is ported
- Secure And Distributed : Designed with distributed environment of internet and has extensive library of routines for TCP/IP protocols, etc.
- Robust : Strictly typed and performing runtime checks
- Dynamic : Library freely add new method/instance variables without effects on client
- Automatic Memory Management : Garbage Collector
- High Performance : Runtime byte code to machine code translation for particular CPU on which application is running.
- Multi Threading : Single program has different thread executing independently at same time.

Hello World

```java
/* This is a simple Java program.
   FileName : "HelloWorld.java". */
class HelloWorld
{
    // Your program begins with a call to main().
    // Prints "Hello, World" to the terminal
window.
    public static void main(String args[])
    {
        System.out.println("Hello, World");
    }
}
```

Friday, 2 October 2020

- class : keyword to declare new class is being defined
- public : Access Modifier ; JVM can access method from anywhere
- static : To be called without object
- void : Method do not return anything
- main() : name configured in JVM
- String arg[] : to provide command line arguments
- S.o.p.In(System.out.printIn) : To print hello world

Naming Conventions

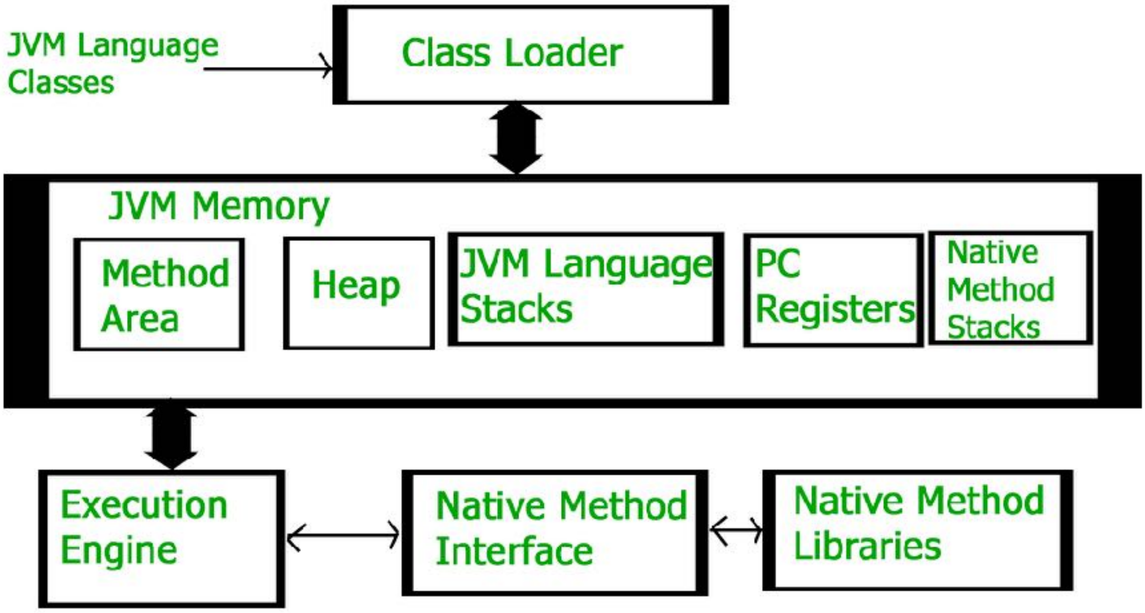
- Camel Case Programming
- class : Noun and 1st letter Capital ; ex = MountainBike
- Method : Verb and 1st Letter Small ; ex = changeGear()
- Variable : Can start with _ or $; temporary could be i, j ,k
- Constant Variable : Upper case and separated by _ ; ex = POSITIVE_INFINITY
- Package : all Lower Case and should be one of top domain names like com, edu, net ; ex = com.sun.eng

---

JVM

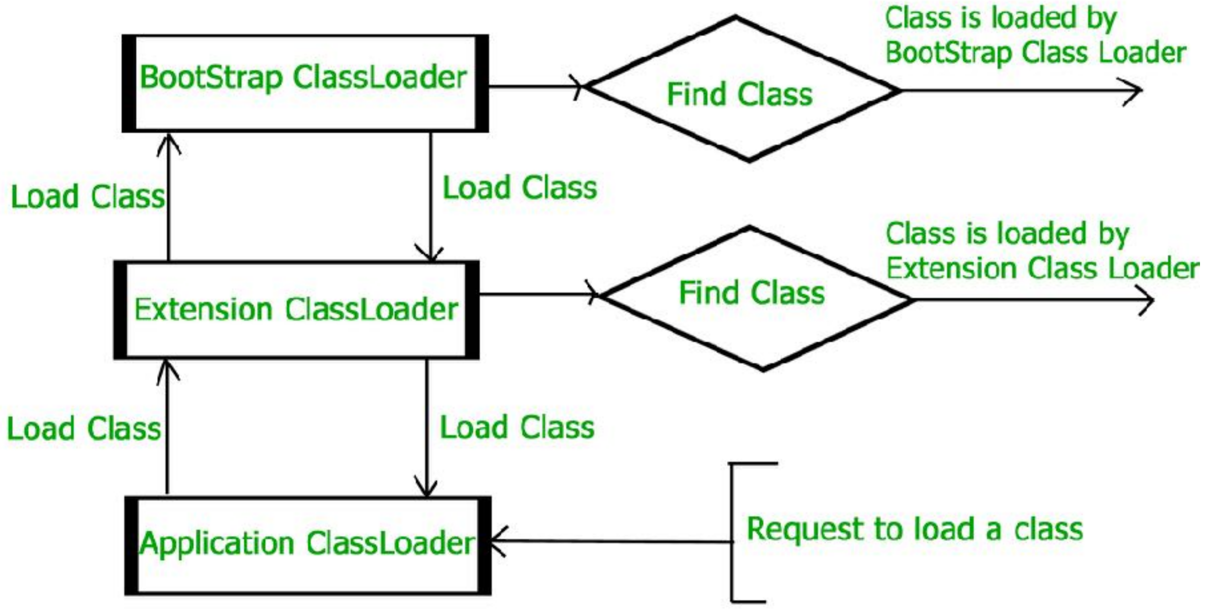
- Run Time Engine that allows java program (src code) compiled into byte code to run on any computer that has native JVM.
- Allows Java to Be WORA(Write Once Run Anywhere)
- .class file goes into various steps that describe the JVM. Refer diagram :

Friday, 2 October 2020



- **<u>Class Loader</u> :**
- 3 activities :
  - *<u>Loading</u>* =>
    - Reads .class file generates corresponding binary data and save in method area(fully qualified class name, method, variables)
    - After loading , JVM created object of Type Class (predefined in java.lang) in heap
  - *<u>Linking</u>* =>
    - Verification of correct format generated by valid compiler ;
    - Preparation of memory allocation for class variables and initialising the memory to default values
    - Resolution : replace symbolic reference with direct reference
  - *<u>Initialization</u>* =>
    - Static variables are assigned with values defined in code or static block.
    - Executed top bottom and Parent-Class hierarchy approach.
- 3 Class Loaders:
  - *<u>BootStrap</u>* => loads core Java Api from JAVA_HOME/jre/lib

- *Extension* =>child of Bootstrap ; loads from JAVA_HOME/ jre /lib/ext (extension dir)
- System/Applictaion => child of extension class loader; It is responsible to load classes from application class path.



- **JVM Memory :**
- Method area :
  - Contains all class level information : name, method , immediate parent class, static variables
  - One method area per JVM
  - Shared resource
- Heap :
  - Information of all object is stored
  - One heap per Jam
  - Shared resource
- Stack Area :
  - For every thread , JVM creates one runtime stack, which is stored here.

  - Each block of this stack is called Activation record/ stack frame which stores method calls
  - Local variable stored in corresponding stack frame and on thread termination JVM destroys the frame.
  - Not a shared resource

- PC Registers :
  - Stores address of current execution instruction of thread
  - Each thread has separate PC Register
- Native Method Stack:
  - Per thread separate Native stack and stores native information
- **Execution Engine:**
- Executes byte code line by line and uses information from present in various memory areas for execution of instruction
- Classified in 3 parts
  - Interpreter : Interprets the byte code line by line and then executes. If one method is called multiple times, every time interpretation is needed.
  - Just In Time Complier : Complies byte code to native code for repeated method calls to improve efficiency
  - Garbage Collector : Destroys unreferenced objects
- **Java Native Interface (JNI) :**
- Interface which interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution.

JVM Stack Frame Structure

- 3 parts :
  - *Local Array Variable* :
    - Organised as zero based array of words
    - Contains all parameters and local variables of method
    - Each entry in array is 4 bytes(byte, short, char converted into int)

Friday, 2 October 2020

- *Operand Stack* :
  - JVM performs two operation on stack : Push and Pop
  - Used for storing intermediated calculation result
  - Organised as array but not accessed via index by rather by instruction like push and pop
- *Frame Data* :
  - Stores symbolic reference, reference to execution table that provides corresponding catch block in case of exceptions

---

JVM Shutdown Hook

- Special construct or arbitrary block of code called when JVM is shutting down(ex : kill request from OS or out of memory where System.exit(0) don't work)
- For clean up operations

```
class ThreadChild extends Thread {

    public void run() {
          /* Logic for shut down hook */
        System.out.println("In clean up code");
        System.out.println("In shutdown hook");
    }
}

class Demo {

    public static void main(String[] args) {

        Runtime current = Runtime.getRuntime();
        current.addShutdownHook(new ThreadChild());

        for(int i = 1; i <= 10; i++)
            System.out.println("2 X " + i + " = " + 2 * i);
    }
}
```

- May not be executed in some case : SIGKILL, Runtime.Halt() or crash due to internal error.
- Once started, shutdown hook can be forcely stopped before completion
  - Os waits for process to terminate for specific amount of time once SIGTERM is given.
  - If does not terminate within this time limit, forcibly terminates it by issuing SIGTERM.

- Can have more than one shutdown hooks but execution order not guaranteed(can even be concurrent)
- Cannot register/unregister shutdown hood within shutdown hook
- Need shutdown hook security permission during runtime, if using Java Security Manager
- Once shutdown sequence starts , can only be stopped using Runtime.halt()

Java Class File

- Contains bytecode; .class file extension; compiled from .java file; executed by JVM
- If .java has more than one class, multiple class file created.
- Elements of class file :
    - *Magic Number* : First 4 bytes of class file; Predefined value to identify if generated from valid compiler
    - *Major and minor Version* : M.m format; Lower version can be executed on higher versu=ion compiler and not vice versa; ex = jdk 1.7 means 51.0
    - *Constant_pool_count* : Number of constants present in constant pool(symbolic reference)
    - *constant_pool[]* : Information about these constants
    - *access_flags* : formation about modifier declared to class file
    - *this_class* : represents fully qualified name of class
    - *supper_class* : represents immediate parent/ super class of current class(could be object class)
    - *interface_count* : Number of interface implemented by current class
    - *interface[]* : returns interfaces information implemented by current class file.
    - *fields_count* : number of fields(static variable) present in current class file
    - *fields[]* : It represent fields (static variable) information present in current class file.
    - *method_count* : represents number of methods present in current class file.
    - *methods[]* : returns information about all methods present in current class file.

- *attributes_count* : returns the number of attributes (instance variables) present in current class file.
- *attributes[]* : provides information about all attributes present in current class file.

JDK JRE

- JDK (Java Development Kit) :
  - Contains development tools (env to develop programs) and JRE(to execute programs).
  - Includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc)
- JRE (Java Runtime Environment) :
  - Only to execute and not run the program. Specification where working of JVM is specified
  - Consist of JVM, core classes and supporting files.

Main Method

- Object not created by JVM as needed to be accessed everywhere
- Before JDK 7 was not mandatory and could write complete code in static block and execute without main

File name and Class name

- Need to be same if class declared Public else can work even if different
- Best practice keep same but approach can be used for debugging

```
/*** File name: Trial.java ***/
class ForGeeks {
    public static void main(String[] args)
    {
        System.out.println("For Geeks class");
    }
}

class GeeksTest {
    public static void main(String[] args)
    {
        System.out.println("Geeks Test class");
    }
}
```

  - When above file is complied ForGeeks and GeeksTest class files created.
  - Each of them can be tested individually as separate main methods
    - java ForGeeks o/p is For Geeks Class
    - java GeeksTest o/p is Geeks Test Class
- Class name and variable names can be of Predefined class; should not be keywords however.
  - In case of of using String as class name, if predefined class path not specified gives run time error as Main method not found in class

```
public class String
{
    public static void main (java.lang.String[] args)
    {
        System.out.println(“Need to specify path of predefined class");
    }
}
```

JDBC(Java Data Base Connectivity) DRIVERS

- JDBC : API which defines how client may access tabular data(relational database)
- JDBC Drivers : Client-side adapters that convert request from java programs into protocols that DBMS can understand.
- 4 types:
  - *Type 1 driver or JDBC-ODBC(open database connectivity) bridge driver*
    - Converts JDBC method calls to ODBC function calls
    - Universal driver since can connect to any database
    - Not secure as common driver to interact with different database
    - Not portable since not written in java
    - Needs to be installed I individual client machine
    - The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.
  - *Type 2 driver or Native API driver*
    - Converts JDBC method calls to native calls of database API
    - Uses client side libraries of database
    - Secure : need local api to interact with different database

- Not portable
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

- *Type 3 driver or Network Protocol driver*
  - Uses middleware(Applciation server) to convert JDBC call to directly / indirectly vendor specific database protocols
  - No need of individual client side installation, no client side library
  - only network support on client machine
  - Portable but costly since it requires database-specific coding to be done in the middle tier.
  - If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- *Type 4 driver or Thin Driver or Native Protocol driver*
  - Interacts directly with database
  - Does not require Native library, middle ware server, client or server side installation.(Hence Thin)
  - Portable
  - If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is type-4.

Micro Service Architecture

- Small loosely based coupled distributed service
- Small modules : easy to develop code and maintain
- Easier Process Adaption
- Independent Scaling : independently via X-axis scaling (cloning with more CPU or memory) and Z-axis scaling (sharding(distributed partitioning), based upon their needs.
- Unaffected : Large applications remain largely unaffected by the failure of a single module
- DURS: Independently deployed, updated, replaced and scaled.
- Restrictions:

Friday, 2 October 2020

- Configuring hundreds of components
- Debugging service failure across different components in absence of centralized logging and dashboards
- Automation of every single component build, deploy monitor
- Testing : more effort as dependent services need to be up and running

- <u>Frameworks :</u> Spring Boot , Spark, Reslet, Drop Wizard.