INSIDE JVM

AMEYA JOSHI

1

WHAT IS A JVM?

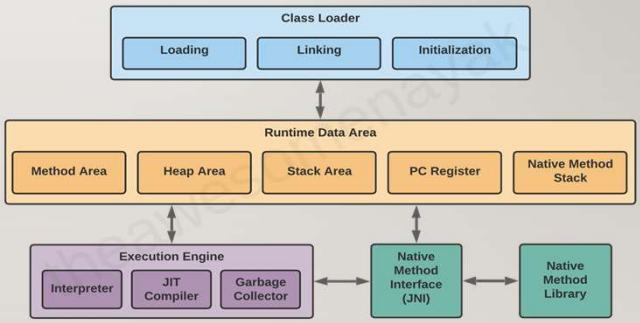
- JVM (Java Virtual Machine) is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.

2

JVM ARCHITECTURE

- The JVM consists of three distinct components:

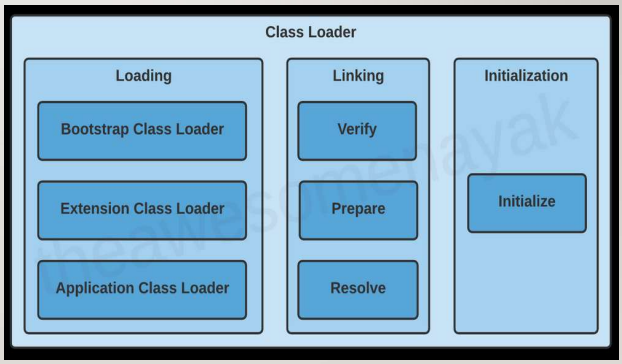
1. Class Loader
2. Runtime Memory/Data Area
3. Execution Engine



3

CLASS LOADER

- The class loader loads the classes into the main memory.
- There are three phases in the class loading process:
  - Loading
  - Linking
  - Initialization



4

LOADING

- Loading involves taking the binary representation (bytecode) of a class or interface with a particular name, and generating the original class or interface from that.
- There are three built-in class loaders available in Java:
  - **Bootstrap Class Loader**
  - **Extension Class Loader**
  - **Application Class Loader**
- The JVM uses the **ClassLoader.loadClass()** method for loading the class into memory.

5

LINKING

- Linking a class or interface involves combining the different elements and dependencies of the program together.
- Linking includes the following steps:
  - **Verification**
  - **Preparation**
  - **Resolution**

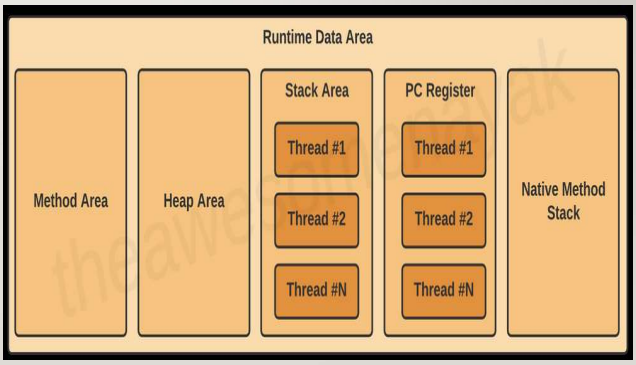
6

INITIALIZATION

- Initialization involves executing the initialization method of the class or interface (known as <clinit>).
- This can include calling the class's constructor, executing the static block, and assigning values to all the static variables.
- This is the final stage of class loading.

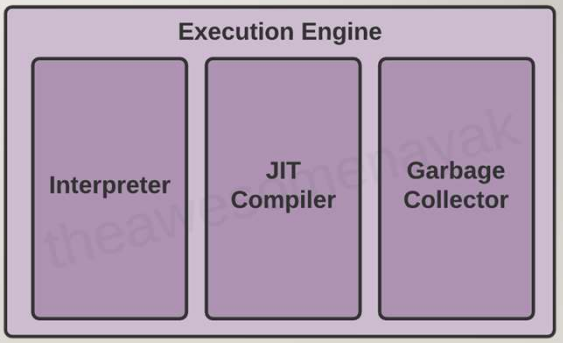
RUNTIME DATA AREA

- There are five components inside the runtime data area:
  - **Method Area**
  - **Heap Area**
  - **Stack Area**
    - **Local Variables**
    - **Operand Stack**
    - **Frame Data**
  - **Program Counter (PC) Registers**
  - **Native Method Stacks**



EXECUTION ENGINE

- The Execution Engine executs the code present in each class.
- Execution engine has 3 components:
  - **Interpreter**
  - **JIT Compiler**
  - **Garbage Collector**



9

GARBAGE COLLECTOR

- The Garbage Collector (GC) collects and removes unreferenced objects from the heap area.
- It is the process of reclaiming the runtime unused memory automatically by destroying them.
- It involves two phases:
  - **Mark -** in this step, the GC identifies the unused objects in memory
  - **Sweep -** in this step, the GC removes the objects identified during the previous phase
- Compact remaining objects in memory

10

DE-REFERENCING OBJECTS

- The main objective of Garbage Collection is to free heap memory by destroying the objects that don't contain a reference.
- There are various ways in which the references to an object can be released:
  - **By making a reference null**
  - **By assigning a reference to another**
  - **By using an anonymous object**

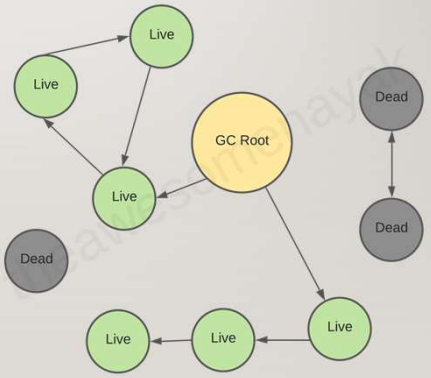
11

GARBAGE COLLECTION ROOTS

- Garbage collectors work on the concept of *Garbage Collection Roots* (GC Roots) to identify live and dead objects.
- Examples of such Garbage Collection roots are:
  - Classes loaded by system class loader (not custom class loaders)
  - Live threads
  - Local variables and parameters of the currently executing methods
  - Local variables and parameters of JNI methods
  - Global JNI reference
  - Objects used as a monitor for synchronization
  - Objects held from garbage collection by JVM for its purposes

12

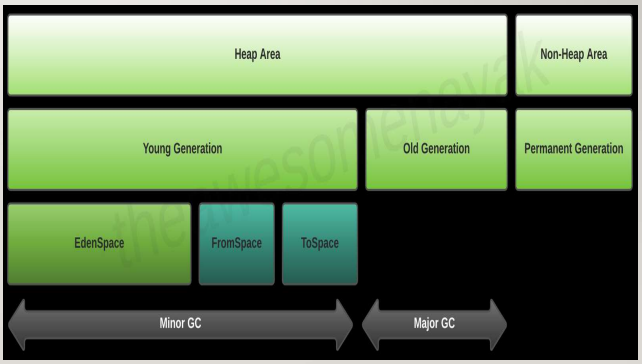
- The garbage collector traverses the whole object graph in memory, starting from those Garbage Collection Roots and following references from the roots to other objects.



13

GENERATIONAL GARBAGE COLLECTION

- Java Garbage Collectors implement a *generational garbage collection strategy* that categorizes objects by age.
- **Young Generation**
  - Eden space
  - Survivor spaces (FromSpace and ToSpace)
- **Old Generation**
- **Permanent Generation**



14

GARBAGE COLLECTOR TYPES

- JVM contains 3 different types of garbage collectors:
  - **Serial GC –** JVM argument => -XX:+UseSerialGC
  - **Parallel GC –** JVM argument => -XX:+UseParallelGC
  - **Garbage First (G1) GC –** JVM argument => -XX:+UseG1GC

15

THREAD DUMPS

- **A thread dump is a snapshot of the state of all the threads of a Java process**.
- The state of each thread is presented with a stack trace, showing the content of a thread's stack.
- A thread dump is useful for diagnosing problems as it displays the thread's activity.
- **Thread dumps are written in plain text, so we can save their contents to a file and look at them later in a text editor**.

16

JDK UTILITIES – THREAD DUMPS

- The JDK provides several utilities that can capture the thread dump of a Java application.
- **All of the utilities are located under the *bin* folder inside the JDK home directory**.
- **Jstack**
  - jstack is a command-line JDK utility we can use to capture a thread dump.
  - jstack [-F] [-l] [-m] <pid>
  - jstack 17264 > /tmp/threaddump.txt
- **Jcmd**
  - jcmd 17264 Thread.print

17

HEAP AND NON-HEAP MEMORY

- **Heap memory**
  - This is the runtime data area from which the Java VM allocates memory for all class instances and arrays.
  - The heap may be of a fixed or variable size.
  - The garbage collector is an automatic memory management system that reclaims heap memory for objects.
- **Non-heap memory**
  - includes a method area shared among all threads and memory required for the internal processing or optimization for the Java VM.

18

HEAP DUMPS

- **A heap dump is a snapshot of all the objects that are in memory in the JVM at a certain moment**.
- They are very useful to troubleshoot memory-leak problems and optimize memory usage in Java applications.
- **Heap dumps are usually stored in binary format hprof files.**
- We can open and analyze these files using tools like JVisualVM. Also, for Eclipse users it's very common to use MAT.

19

JDK TOOLS – HEAP DUMPS

- The JDK comes with several tools to capture heap dumps in different ways.
- **All these tools are located under the *bin* folder inside the JDK home directory**.
- **jmap**
  - jmap is a tool to print statistics about memory in a running JVM.
  - jmap -dump:[live],format=b,file=<file-path> <pid>
  - jmap -dump:live,format=b,file=/tmp/dump.hprof 12587

  (we can get the pid with jps command)

20

JDK TOOLS – HEAP DUMPS

- **Jcmd**
  - jcmd is a very complete tool that works by sending command requests to the JVM. We have to use it in the same machine where the Java process is running.
  - jcmd <pid> GC.heap_dump <file-path>
  - jcmd 12587 GC.heap_dump /tmp/dump.hprof
- **Capture a Heap dump automatically**
  - `java -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=<file-or-dir-path>`

21

JVM OPTIONS - CONTROLLING HEAP SIZE

- Java heap size can be controlled using the below vm options
  - Use -Xmx to specify the maximum heap size
  - Use -Xms to specify the initial Java heap size
  - Use -Xss to set the Java thread stack size
  - Use –Xmn to specify initial Java heap size for Eden generation
- Java `-Xms<heap size>[unit]`
  - Here, ***unit*** denotes the unit in which the memory (indicated by ***heap size***) is to be initialized. Units can be marked as ***'g'*** for GB, ***'m'*** for MB and ***'k'*** for KB.
  - `Java -Xms2G -Xmx5G`

22

JVM OPTIONS - CONTROLLING GARBAGE COLLECTION

- The following options can be used to set GC type
  - `-XX:+UseSerialGC`
  - `-XX:+UseParallelGC`
  - `-XX:+UseG1GC`

23

JVM OPTIONS – OUT OF MEMORY ERROR

- Jvm parameters to dump heap memory to file
  - `-XX:+HeapDumpOnOutOfMemoryError`
  - `-XX:HeapDumpPath=./java_pid<pid>.hprof`
  - `-XX:OnOutOfMemoryError="< cmd args >;< cmd args >"`
  - `-XX:+UseGCOverheadLimit`

24

VERBOSE GC

- **Verbose garbage collection logging is often required when tuning and debugging many issues**, particularly memory problems.
- In fact, some would argue that in order to strictly monitor our application health, we should always monitor the JVM's Garbage Collection performance.
- **For each GC happening, the GC log provides exact data about its results and duration.**
- **It helps optimize GC frequency and collection times by specifying the best heap sizes, other JVM options, and alternate GC algorithms.**
- `-XX:+UseSerialGC -Xms1024m -Xmx1024m -verbose:gc`

25

JCONSOLE

- The JConsole graphical user interface is a monitoring tool that complies to the Java Management Extensions (JMX) specification.
- JConsole uses the extensive instrumentation of the Java Virtual Machine (Java VM) to provide information about the performance and resource consumption of applications running on the Java platform.
- The jconsole executable can be found in JDK_HOME/bin, where JDK_HOME is the directory in which the Java Development Kit (JDK) is installed
- You can use JConsole to monitor both local applications, namely those running on the same system as JConsole, as well as remote applications, namely those running on other systems

26

VISUAL VM

- VisualVM provides detailed information about Java applications while they are running on the Java Virutal Machine (JVM).
- VisualVM's graphical user interface enables you to quickly and easily see information about multiple Java applications.
- VisualVM perfectly fits all the requirements of application developers, system administrators, quality engineers and end users.