CoreJava Concepts

**Understanding JVM Architecture:**

Java files are compiled and convered into class files called byte code.

Java Runtime Environment composed of the Java API and JVM.

JVM will load java class through one of its class loader and execute it along with Java API.

NOTE: JVM is platform dependent.

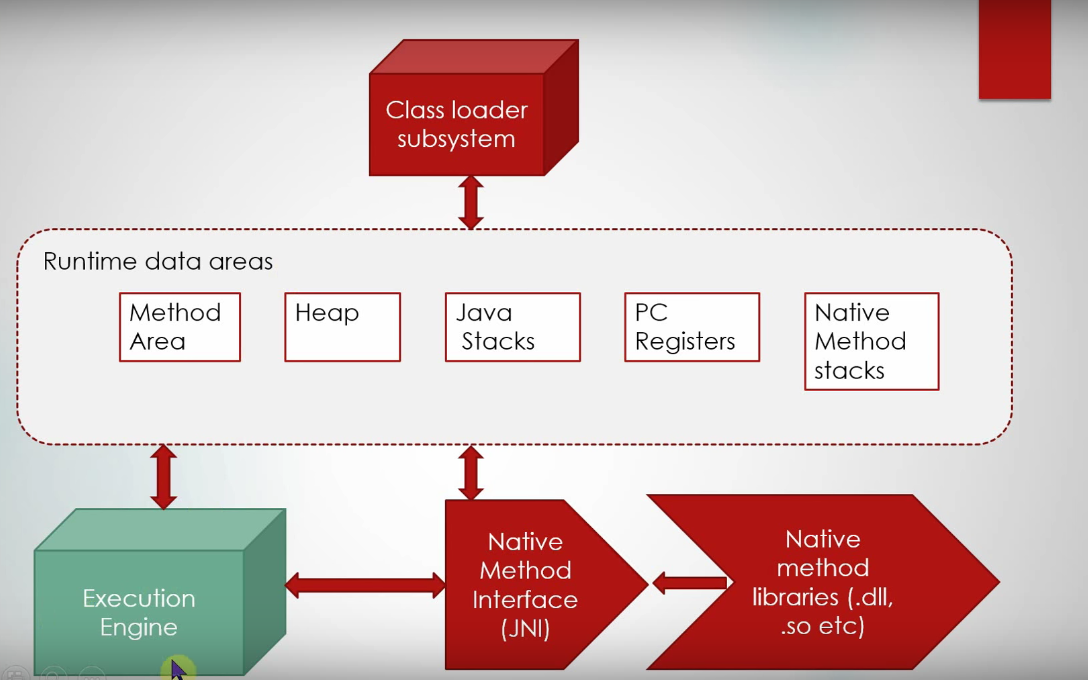
Reference: <https://www.youtube.com/watch?v=ZBJ0u9MaKtM>

JVM Architecture has following areas.

Class loader subsystem,

Runtime data areas

Execution engine

****

Class Loader Subsystem: Class loader subsystem has three main phases.

* Load
* Link
* Initialize

Load Phase involves three different types of class loaders

1. Bootstrap Class loader (**rt.jar**)
2. Extension Class Loader (**jre/lib/ext**)
3. Application ClassLoader (**Classpath** or **–cp**)

Link Phase has three different phases

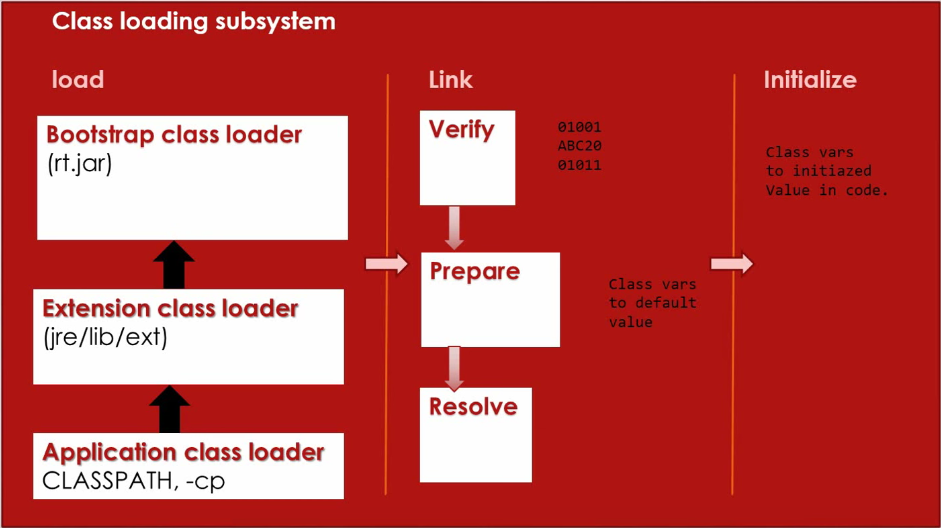
* Verify
* Prepare
* Resolve

**Verify**: Magic number verification etc.

**Prepare**: Memory for the class variables happends (static variables). Default values assigned in this phase.

**Resolve**: Symbolic references are resolved. Class references etc. if the class is not found ClassDefNotFound exception is thrown.

**Initialize Phase:** static blocks are initialized. Actual values are assigned in this Phase in the memory location.



**Runtime Data Areas**

**Method area:** Method area contains class metadata. Byte code etc. it is also called **Permgenspace**. **–XX: MaxPermSpace.**  In java 8 it is called **MetaSpace**. Moved permgen into separate memory in native operatiing system memory. Metaspace doesn’t have limit it can grow in system memory. Using some Parameter you can limit.

**-XX: MetaspaceSize=<size>**

**-XX: MaxMetaspaceSize=<size>**

**Heap:** Heap is an important memory area. Objects, instance variables and arrays are created inside the heap.

Using -**Xms**={¼th Physical memory} and –**Xmx** we can customize Heap memory.

**PC Registers:** Program Counter Register. It is pointer to the next instruction for Thread.

**Java Stack:** Stack Frame is specific to method. Used to store local variables. If the recursive algorithm is infinite **java.lang.StackOverflowError**

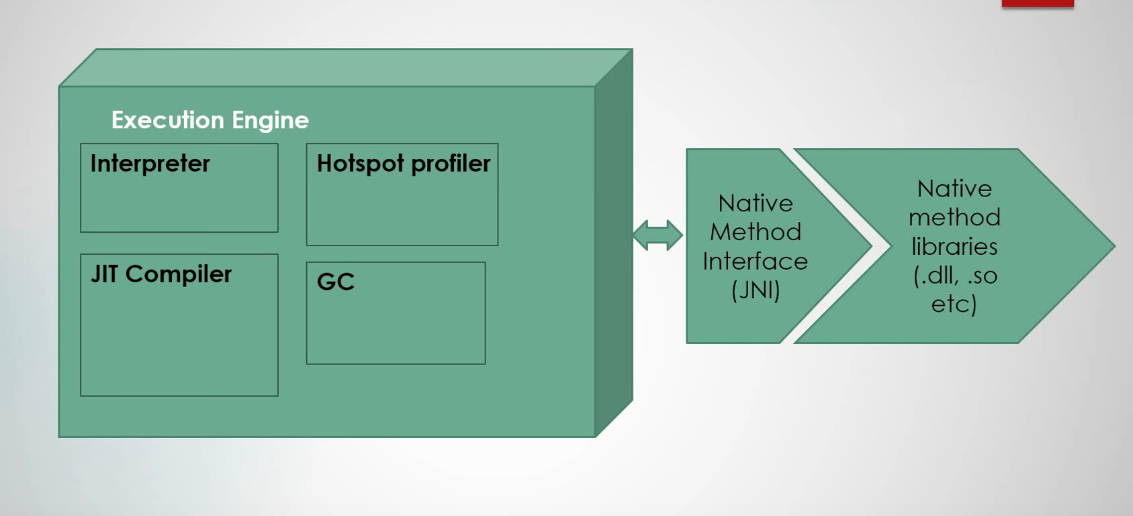
**Native Method Stack:** Native method stack.

**Execution Engine: It contains four main areas**

1. **Interpreter:** interprets bytecode into machine level code
2. **JIT compiler:** repeatedly interpreted instructions directly compilter on the fly

The just-in-time compiler comes with the virtual machine and is used optionally. It compiles the bytecode into platform-specific executable code that is immediately executed. Sun Microsystems suggests that it's usually faster to select the JIT compiler option, especially if the method executable is repeatedly reused.

1. **Hotspot profiler:** it helps JIT compiler
2. **GC:** Garbage collection.



**Q) ClassNotFoundException**:  ClassNotFoundException occurs when class loader could not find the required class in class path. So, basically you should check your class path and add the class in the classpath.  
  
**NoClassDefFoundError**: This is more difficult to debug and find the reason. This is thrown when at compile time the required classes are present, but at run time the classes are changed or removed or class's static initializes threw exceptions. It means the class which is getting loaded is present in classpath, but one of the classes which are required by this class, are either removed or failed to load by compiler .So you should see the classes which are dependent on this class.

**Q) What all memory areas are allocated by JVM?**  
Heap (method area/ permgen space, runtime consant pool), Stack, Program Counter Register and Native Method Stack

**Q). What are the three steps of GC?**

**Mark:** it Tranvers the entire object grap and mark or unmark each and every node in that graph.

**Sweep:** cleans up the objects that are not reachable.

**Compacting:** rearrange objects (contiguous mean side by side)

<https://www.youtube.com/watch?v=UnaNQgzw4zY>

**Q) What is MaxTenuringThreashold?**

This is value of each object in the young generation Heap memory. It is the number of cycles it is sruvivde before going into old generation.

Young Generation having **Eaden**, **Survivor1** and **Survivor2** spaces. Each minor GC is called on Eaden space and moved objects to S1 or S2 space until this threashold value is reached. After that it moves into Old generation.

**Q) What is latency and throughput?**

* **Latency:** How quickly application responds for the piece of data.
* **Throughput**: number of inputs allows within a unit of time.

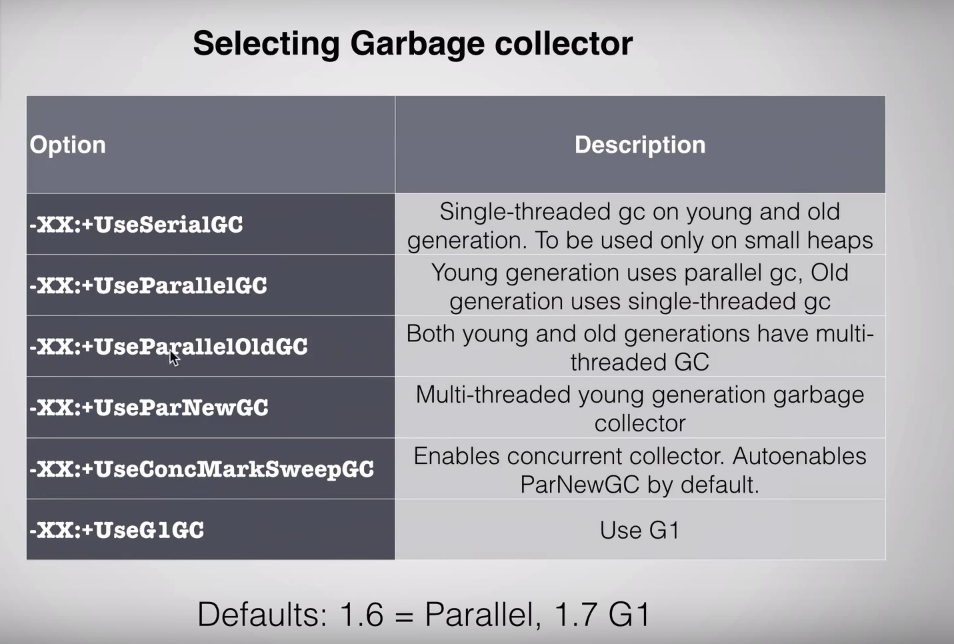
**Q) What are different Garbage collectors in java?**

**Serial Garbage Collector:**

**Parallel Garbage Collector:**

**CMS: Concurrent Mark Sweep:**

**G1: Garbage first**



**Q). What is the difference between dynamic Polymorphism vs Static Polymorphism?**

**Dynamic polymorphism** is run time polymorphism that is invokes overrided method using parent class reference.

**Static Polymorphism** is compile time polymorphism, which means invoking overloaded method /static method.

**Q). What is aggregation how it is useful?**

Aggregation is one type of association and it is directional association. Which is Has-A relationship. Example Address entity is aggreation with Student, College, Staff.  **In UML it is represented as open diamond arrow.**

Hecne 🡺 Student **Has-A** Address

🡺 College **Has-A** Address

🡺 Staff **Has-A** Address

**Q). What is Associattion?**

Association establishes relation between two classes through their objects.

Ex: Car.name is using by Driver.name

Here Car and Driver are two classes made a relationship.

**In UML it is represented as Arrow or strait line.**

**Q) What is composition?**

Compostion is resticted form of Aggregation that is strictly dependent on each other.

**In UML it is represented as closed diamond arrow.**

**Q) What is Coupling and Cohesion?**

**Coupling:** By definition coupling is the degree to which one class has knowledge of another or in other words one class has a dependency upon another.

**Cohesion:** Class is designed to specific purpose instead of multiple sets of functions. High Cohesion

**Q) S.O.L.I.D Principles?**

**S- Single Responsibility Principle:** One class should have one and only one responsibility. Class should have only one reason to change.

**O- Open Close Principle:** Software components should be open for extension, but closed for modification

**L-Liskov Substituion Principle:** Derived types must be completely substitutable for their base types

**I- Interface Segregration Principle:** Clients should not be forced to implement unnecessary methods that they will not use

**D-Dependency Inversion Principle:** Depend on abstractions, not on concretions.

**Q) What is the purpose of Volatile keyword?**

In a multi-theading environment it will always check the value of the variable in the main memory instead of CPU cache. So that half initialized values you never get.

**Q) What is Encapsulation?**

Hiding the implementation details from outside by making the fields as private and writing getter and setter methods.

**Q) What are the annotations and what are the built in annotations in core Java?**

Annotations are meta-data for the classes and methods.

There are three built-in annotations in java

(@Deprecated, @Override, @SuppressWarning)

**Q) How to create custom annotations?**

Annotations are created using **@interface.** All annotations extends **java.lang.annotation.Annotation** interface**.**

**@Documented** annotation indicates that element using this annotation should be documented by JavaDoc. When you generate javaDoc it is included in that.

**@Target (ElementType.*METHOD*):** it specified where we could use the annotation. In the below code, we have defined METHOD which means that below annotation can only be used on methods.

**@Inherited:** The @inherited annotation singals that a custom annotation used in a class should be inherited by all of its sub classes.

**@Retention (RetentionPolicy.*RUNTIME)*:** it indicates that how long annotation with the annotated types are to be retained.

**Q) What is the difference between String literal and String Object?**

String literals are stores in the String constant Pool and String objects stored in the Heap.

String s1=”Hello”;

String s2= new String (“Hello”). Intern ();

Now s1==s2 🡺 true

**Q) How does String intern method works?**

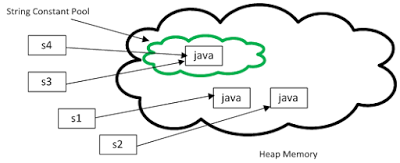
**String.intern()** method can be used to to deal with String duplication problem in Java. By carefully using the intern() method you can save a lot of memories consumed by duplicate String instances.

A string is duplicate if it contains the same content as another string but occupied different memory location e.g. str1 != str2 but str1.equals(str2) is true. Since String object consumes a large amount of heap memory in average Java application, it makes sense to use intern() method to reduce duplication and take advantage of [String pool feature](http://javarevisited.blogspot.com/2013/07/java-string-tutorial-and-examples-beginners-programming.html) provided by Java. You can use intern() method to intern a String object and store them into String pool for further reuse.

For example, when you create a [String literal](http://java67.blogspot.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html) e.g. "abc", it's automatically stored inside String pool, but when you create a new String object e.g. new String("abc"), even though it's same String, a new object at a different memory location is created. This is a duplicate String. By calling the **intern()** method on this object, you can instruct JVM to put this String in the pool and whenever someone else creates "abc", this object will be returned instead of creating a new object. This way, you can save a lot of memory in Java, depending upon how many Strings are duplicated in your program. You can also read [Java Performance by Charlie hunt](http://www.amazon.com/dp/0137142528/?tag=javamysqlanta-20" \t "_blank) and [Java Performance, The Definitive Guide](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20" \t "_blank) By Scott Oaks to learn more about troubleshooting Java Performance issue and optimizing the performance of Java applications.

**Some important things about String.intern() method**

Here are some of the important points about the intern() method from java.lang.String class which are worth remembering:  
  
1) String.intern() method is there in String class from JDK 1.1. It returns a canonical representation of String object. When the intern method is invoked, if the String pool already contains that String object such that [equals()](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html) return true, it will return the String object from the pool, otherwise it will add that object into the pool of unique String.  
  
  
2) After calling intern() method on s1 and s2, **s1.intern() == s2.intern()**, if s1.equals(s2) because both will be pointing same String constant in pool.  
  
  
3) Prior to Java 6, uncontrolled usage of String.intern() method can cause [java.lang.OutOfMemory: PermGen space](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html)because String pool was physically located on PermGen area of Java heap, which is quite small in many JVM (32M to 96M) and fixed. From Java 7 onward, the intern()  method has become even more useful because String pool is relocated to main [heap space of JVM](http://javarevisited.blogspot.com/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html). This will help to further reduce String duplication by using String.intern() method.

[](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html)

4) The intern() method is a non-static method and should be called using a String literal or String object in Java.

<http://javarevisited.blogspot.in/2015/12/when-to-use-intern-method-of-string-in-java.html>

**Q) What is String deduplication?**

Java 8 update 20 also introduced a new feature called *String deduplication*, which can reduce memory footprint caused by duplicate String without writing a single line of code, but, unfortunately, it's only available for G1 garbage collector and you cannot use it if you are using **ConcurrentMarkSweep** garbage collector.

**Q) Why String is immutable?**

Since Strings are managed in String constant pool to save the memory.

1. If it is mutable, chacing in one String literal affect the other String lieratal, which are pointing to the same.
2. **Thread safe:** if multiple threads are changing the value of string that will affect the other liferal references.
3. String literal caches hashcode to avoid repeatedly calculating the hashCode. Especially useful in the case of HashBased Algorithms.

**Q) Which API classes are useful to find the matches?**

**Pattern pattern= java.util.regex.Pattern.compile(“StringToSearch”);**

**Matcher matcher= Pattern.macher(input);**

**while( matcher.find()){**

**counter++;**

**}**

**Q) What happens when you run below code?**

String output=input.substring (input.length ());

It will return empty string instead of throwing exception.

**Q) StringBuilder which design patterns it uses?**

**StringBuilder uses Builder Design Pattern**

**New StringBuilder(“Hello”).append(“world”).append(true);**

All the (+) will replaced by final code with Stringbuilder. You can check this using below command.

**$/>javap –c <YOUR\_CLASS\_FILE>**

**Q) Difference between concat () and append () methods?**

The concat () method is on String object and append () method is on StringBuilder.

**Q) Why generally passwords are stored in the char array instead of String?**

Strings are managed inside the String constat pool they are not garbage collected. If someone got that memory dump they can easily read the password from that. To avoid this we are generally use char array to store password. Arrays are stored inside the Heap can be garbage collected if it is not reacheable.

**Q) How the HashMap put and get methods works?**

It calcuates the hashcode of the KEY which is in String format

Since Hashcode is very large number and therefore it calcuates the index using below logic

**Int index= hashCode & (length-1)**

Once the index is calculated which is within the range of hashMap size. So it will place that object in that index.

If there an element already exists in that location it will prepare a linkedList in that location and add new element there.

Get method works as mentioned below

1. First it will calcuate hashcode of the key
2. Calculates the index value of that key
3. Using this index it will look into that place for the element.
4. If the element found check both hashcodes
5. If both hashcodes are same it also checks equals method contract
6. If that returns true, then then it will return the value from that entry.

**Q) What is the Performance improvement for HashMap in java 8?**

If the more number of keys having the same hashcode (theashould above>8) linked list is replaced with the Balanced Tree. So that time complexity of the map element changed **O (n)** to **O (log n)**