Garbage Collection

**Introduction** :

* Very important to understand GC in JVM
* Concept of Live objects and Dead objects.
* Concept of Stack and Heaps
* References and variables live in Stack and Objects and Instance variables live in Heap
* Java 7 - Parallel GC
* Java 8 - Parallel GC
* Java 9 - G1 GC

**GC Steps** :

* Mark, Delete/Sweep and Collect
* There is always a root class in Java from where the application starts , eg a class having main method, eg public class Test{

psvm(…){

}

}

Test iw the root class/node

1. The objects are laid out in memory in form of a graph /tree(?) and goes through each and every node and marks them as either **Reachable or non Reachable**.
2. Deletes Unreachable objects and reclaims unreachable memory
3. Then it compacts/rearrange memory in contagious blocks i.e the nodes are reclaimed and addd to the free memory tree/list/graph

**Generational Collectors :**

* Heap is divided into 2 tree/graph/List , Young and Old generation
* Young generation space is further divided into Eden space – space for new Objects creation, empty space. Survivor Space – If **Eden Spce** becomes full, **The GC runs and moves the Reachable Objects from Eden Space to Survivor space.**
* Imagine an Object its first created in **Eden Space** , then after a few cycle runs of GC it moves to the **Survivor space** and after a lot of cycle runs if the object is still referenced it gets moved to **Old Generation**
* The minor GC runs across the **Eden space**  whereas the Major GC sweeps **across all the GC Spacec**
* GC run is called “Stop the World ” GC because when the GC runs it causes the application to stop temporily
* Minor GC runs continuously across the Young generation and keeps moving from Survivor 1 to survivor 2 and so on. It keeps on incrementing the GC sweep number and when the surviving object reaches the saturation number the Objects are promoted to the Old Generation by the Major GC

**Issues till now:**

**The Garbage Collector may halt the JVM when performing Mark, Sweep and Recollect.**

**Performance Improvement:**

* The GC times depends o Responsiveness/Latency and throughput.

**Types of Garbage Collectors :**

**Serial GC :**

* Comes with Java, runs in a single thread, application is running , will be paused, **runs GC etc**

**Concurrent GC:**

* Runs concurrent to application.
* Stops the world for Mark in young and old generation but runs Sweep and recollect along with application running
* Should be used when there is more memory
* high number of CPUs
* Application should be highly Responsive
* Also known as Low Latency Collector

**Parallel GC:**

* Uses Multiple cores of CPU to perform GC and runs with multiple threads for GC
* Kicks in when old Gen is 90% full example.
* Lesser Memory and less CPU, high Throughput required

**G1 Garbage first Collector:**

* Implemented in Java 7
* Devides Heap Into small region of memory.
* Whichsoever region has more garbage that area quickly gets collected
* Its COnsurrent and Parallel collector combination

**How do we use a particular GC?:**

* **Small heap use Serial GC +xX:+UseSerialGC etc**

**How do we implement these in Java code?**

* **protected void finalize() throws Throwable** is called when the GC happens.
* There is no guarantee when it will be called.
* DON’T USE it
* finalize method is always called once for an object

**Some flags to tune our GC parameters:**

* We can tune the heap and the GC will be autotned
* -XmsValue is the minimum amount of heap allocated to the JVM
* -XmxValue is the maximum amount of memory allocated to heap
* -XX:newration(value) how do we want to devide old and new gen. Eg 2 means 1/3rs will be used by Old gen and 2/3rd by the young gen
* -XX:PermGen for static vars and objects
* This will autromatically tune the GC

**GC Logging if we think GC is the culprit :**

* -Verbose:gc
* -xx:+printGC Details
* -XLOggc:gc.log
* There are applications like GC Log analyser from IBM which tells about GC in graphical format and suggests tune in features
* Visual VM(**JvisualVm**), it connects the application to the GC
* **Use JHAT Java Heap Analyser Tool lets us look at the Heap**
* **Terracota Big Memory – wil manage objects outside the heap**