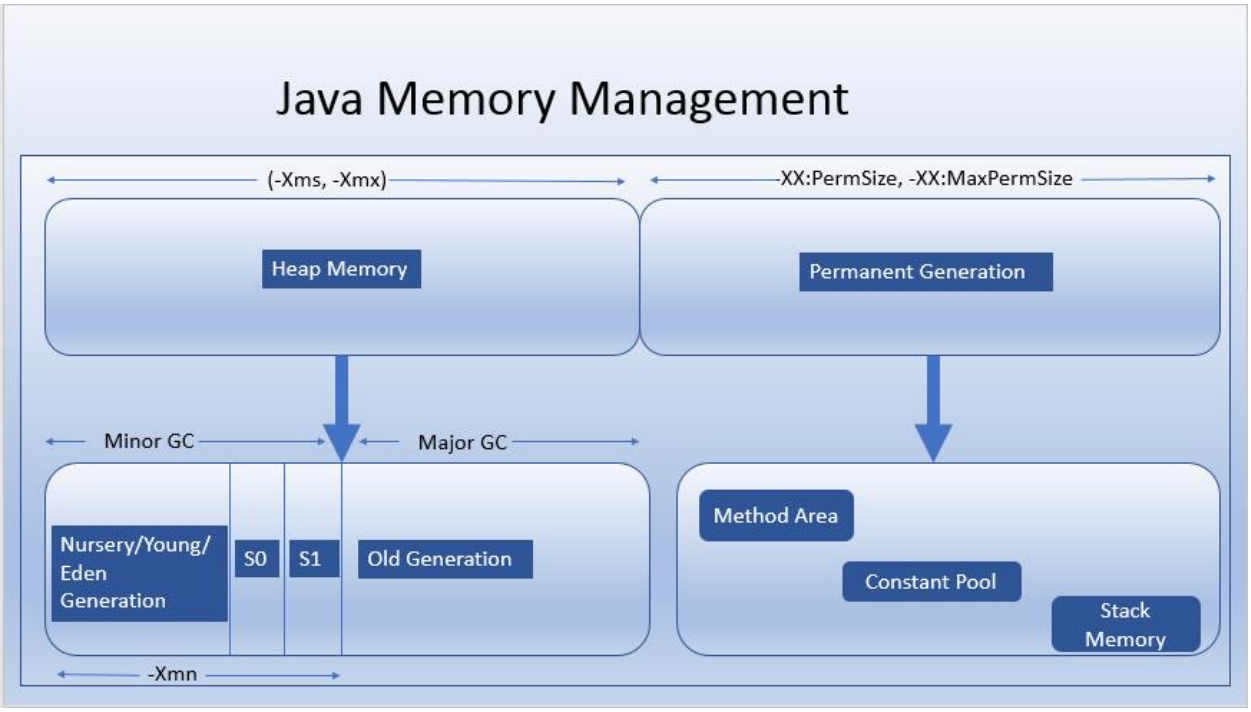
**Java Memory Management**




- Object Allocation
- Garbage collection is Stop the World Event (Major & Minor GC)
- Garbage Collection Process (Mark -Sweep – Compact)

**Common Heap Related Switches:**

| Switch | Description |
|---|---|
| -Xms | Sets the initial heap size for when the JVM starts. |
| -Xmx | Sets the maximum heap size. |
| -Xmn | Sets the size of the Young Generation. |
| -XX:PermSize | Sets the starting size of the Permanent Generation. |
| -XX:MaxPermSize | Sets the maximum size of the Permanent Generation |

**Performance Turing:**

- Response Time:Responsiveness refers to how quickly an application or system responds with a requested piece of data
- Throughput: Throughput focuses on maximizing the amount of work by an application in a specific period of time.

**Below are the different Types of Garbage collector algorithms:**

**Serial GC:**The serial collector is the default for client style machines in Java SE 5 and 6. With the serial collector, both minor and major garbage collections are done serially (using a single virtual CPU). In addition, it uses a **mark-compactcollection** method. This method moves older memory to the beginning of the heap so that new memory allocations are made into a single continuous chunk of memory at the end of the heap. This compacting of memory makes it faster to allocate new chunks of memory to the heap.

To enable the Serial Collector use option:-XX:+UseSerialGC

**Syntax:**java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseSerialGC -jar <jar name>

**The Parallel GC/Parallel Old GC :** The parallel garbage collector uses multiple threads to perform the young genertion garbage collection. By default on a host with N CPUs, the parallel garbage collector uses N garbage collector threads in the collection. The number of garbage collector threads can be controlled with command-line options:-XX:ParallelGCThreads=<desired number>

**Syntax:**

java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseParallelOldGC -jar <jar Name>

java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseParallelGC -jar <jar name>

**The Concurrent Mark Sweep (CMS) Collector:** The Concurrent Mark Sweep (CMS) collector (also referred to as the concurrent low pause collector) collects the Old generation. It attempts to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads. Normally the concurrent low pause collector does not copy or compact the live objects. A garbage collection is done without moving the live objects. If fragmentation becomes a problem, allocate a larger heap.

> To enable the CMS Collector use:-XX:+UseConcMarkSweepGC
> and to set the number of threads use:-XX:ParallelCMSThreads=<n>
>
> Syntax: java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseConcMarkSweepGC -XX:ParallelCMSThreads=2 -jar <jar name>

**The G1 Garbage Collector:** The Garbage First or G1 garbage collector is available in Java 7 and is designed to be the long term replacement for the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector that has quite a different layout from the other garbage collectors described previously.

To enable the G1 Collector use:
-XX:+UseG1GC

Syntax: java -Xmx12m -Xms3m -XX:+UseG1GC -jar <jar name>

Tools to Monitor The Garbage collection process

- **Jvisualvm Tool**
- **JConsole Tool**
- **Jmc tool**

**Is it good practice to call Garbage Collector Explicitly?**

No, it is definitely not good practice. But still if you have requirement then you call the garbage collector explicitly by invoking "System.gc() or Runtime.getRuntime().gc()" method.

Note that this is not guaranteed to call the garbage collector - it only gives a hint to the system that it might be a good idea to do garbage collection

**What is different difference between System.gc() vs finalize() method?**

**System.gc()** - kindly asks the system to perform a garbage collection

**finalize()** - can be overridden to specify what to do when a given object is garbage collected (actually what to do just before it is garbage collected)

Classical use of finalizer is to de-allocate system resources when an object is garbage collected, e.g. release file handles, temporary files, etc.

**What is OutOfMemoryException? When it will occur and how to solve?**

- OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError
- Below is different type of OutOfMemoryExceptions we observe while working with Java
  - java.lang.OutOfMemoryError: Java heap space
  - java.lang.OutOfMemoryError: PermGen space

To Solve the OutOfMemoryException, first setup is to increase the space based on the type of error (Java Heap/PermGen)

cmd> Java -Xms1024m -Xmx1024m //to increase the JVM heap size

cmd> java -XX:Permsize=128M -XX:MaxPermsize=256M → To increase the PermGen Size

If you still persist the problem, you need to diagnose by going through the application code to find any memory leaks happens in code and fix the same.

Below are the java tools used to identify the Memory Consumption on the application.

- Jconsole.exe
- Jcm.exe
- Jvisualvm.exe