**Eden Space:** When an instance is created, it is first stored in the eden space in young generation of heap memory area.

**Survivor Space (S0 and S1):** As part of the minor garbage collection cycle, objects that are live (which is still referenced) are moved to survivor space S0 from eden space. Similarly the garbage collector scans S0 and moves the live instances to S1.

Instances that are not live (dereferenced) are marked for garbage collection. **Depending on the garbage collector** (there are four types of garbage collectors available) chosen either the marked instances will be removed from memory on the go or the eviction process will be done in a separate process.

**Old Generation:** Old or tenured generation is the second logical part of the heap memory. When the garbage collector does the minor GC cycle, instances that are still live in the S1 survivor space will be promoted to the old generation. Objects that are dereferenced in the S1 space is marked for eviction.

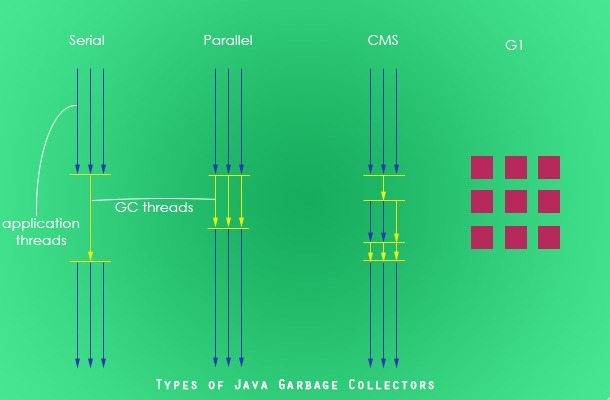
**Major GC:** Old generation is the last phase in the instance life cycle with respect to the Java garbage collection process. Major GC is the garbage collection process that scans the old generation part of the heap memory. If instances are dereferenced, then they are marked for eviction and if not they just continue to stay in the old generation.

Java has **four types of garbage collectors**,

1. [Serial Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#serial-garbage-collector) –
   1. works by holding all the application threads
   2. designed for the single-threaded environments.
   3. It uses just a single thread for garbage collection.
   4. may not be suitable for a server environment.
2. [Parallel Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#parallel-garbage-collector)
   1. It is the default garbage collector of the JVM.
   2. This uses multiple threads for garbage collection.
   3. Similar to serial garbage collector this also freezes all the application threads while performing garbage collection
3. [CMS (Concurrent Mark Sweep) Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#cms-garbage-collector)
   1. Uses multiple threads to scan the heap memory to mark instances for eviction and then sweep the marked instances.
   2. CMS collector uses more CPU to ensure better application throughput.
   3. CMS garbage collector holds all the application threads in the following two scenarios only.
      1. While marking the referenced objects in the tenured generation space.
      2. If there is a change in heap memory in parallel while doing the garbage collection.
4. [G1 Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#g1-garbage-collector)
   1. 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
   2. The G1 collector is a server-style garbage collector, targeted for multi-processor machines with large memories.
      1. The heap is partitioned into a set of equal-sized heap regions, each a contiguous range of virtual memory.
      2. G1 performs a concurrent global marking phase to determine the liveness of objects throughout the heap.
      3. After the mark phase completes, G1 knows which regions are mostly empty. It collects in these regions first, which usually yields a large amount of free space.
      4. **This is why this method of garbage collection is called Garbage-First**.
      5. **Compaction in G1 (DeFragmentation)**: G1 offers more **predictable garbage collection pauses** than the CMS collector, and allows users to specify desired pause targets. G1 copies objects from one or more regions of the heap to a single region on the heap, and in the process both compacts and frees up memory.

-XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70

* + 1. GC is best when there is heap size of around 6GB or larger, and stable and predictable pause time below 0.5 seconds.

Types of Collectors
• The G1 collector does not use generations
– Heap divided into ~2000 regions
– Objects are moving bet...

### Type of Garbage Collector to run

|  |  |
| --- | --- |
| **Option** | **Description** |
| -XX:+UseSerialGC | Serial Garbage Collector |
| -XX:+UseParallelGC | Parallel Garbage Collector |
| -XX:+UseConcMarkSweepGC | CMS Garbage Collector |
| -XX:ParallelCMSThreads= | CMS Collector – number of threads to use |
| -XX:+UseG1GC | G1 Gargbage Collector |

### GC Optimization Options

|  |  |
| --- | --- |
| **Option** | **Description** |
| -Xms | Initial heap memory size |
| -Xmx | Maximum heap memory size |
| -Xmn | Size of Young Generation |
| -XX:PermSize | Initial Permanent Generation size |
| -XX:MaxPermSize | Maximum Permanent Generation size |

There are two main types of garbage collection: tracing and [reference counting](http://en.wikipedia.org/wiki/Reference_counting).

Reference counting schemes record the number of references to a given object and collect the object when the reference count becomes zero.