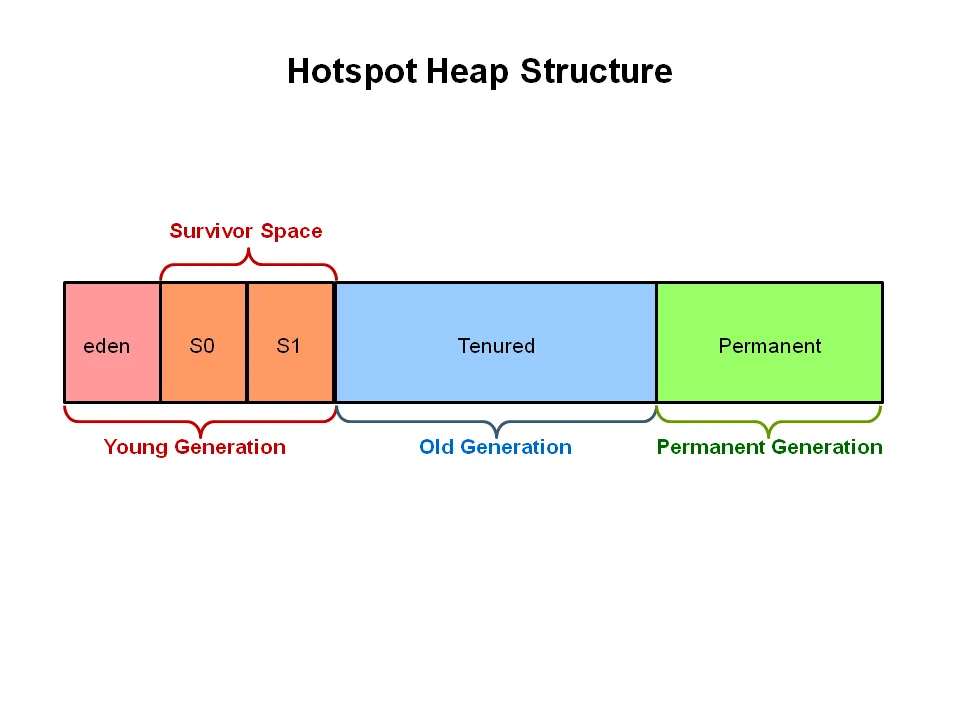
*Source - https://docs.oracle.com/en/java/javase/17/gctuning/introduction-garbage-collection-tuning.html#GUID-8A443184-7E07-4B71-9777-4F12947C8184*

*Garbage collection in Java is the automated process of deleting code that's no longer needed or used. This automatically frees up memory space and ideally makes coding Java apps easier for developers. Java applications are compiled into bytecode that may be executed by a JVM.*

**

*Java 8 Heap Structure*

*What Is Garbage Collection Tuning*

*Garbage Collection GC tuning is the process of adjusting the startup parameters of your JVM-based application to match the desired results. It can be as simple as adjusting the heap size – the -Xmx and -Xms parameters or it can be as complicated as tuning all the advanced parameters to adjust the different heap regions.*

*Why Is Garbage Collection Tuning Important*

*Cleaning our applications’ JVM process heap memory is not free. There are resources that need to be designated for the garbage collector so it can do its work. You can imagine that instead of handling the business logic of our application the CPU can be busy handling the removal of unused data from the heap.*

*This is why it’s crucial for the garbage collector to work as efficiently as possible. The GC process can be heavy. What’s dangerous, however, is a complete stop of the application threads for a very long period of time – like seconds or in extreme cases even minutes. To avoid that we need to ensure that the garbage collector that is running for our JVM applications is well configured.*

*When to Do Garbage Collection Tuning*

*Garbage collection should be one of the last operations you do. Unless you are absolutely sure that the problem lies in the garbage collection, don’t start with changing JVM options. If your JVM memory utilization looks good and your garbage collector works without causing trouble, you shouldn’t spend time turning your garbage collection. Your Java monitoring tool will provide you information regarding your JVM metrics.*

*Starting GC Tuning –*

* *Assigned objects in the Eden generation are moved to Survivor space.*
* *Assigned objects in the Survivor space are moved to Tenured generation if the counter is high enough or the counter is increased.*
* *Assigned objects in the Tenured generation are ignored and will not be collected.*

*Heap Size - What should you consider when setting up the heap for your application? heavily I/O dependent and can share the operating system file system cache. If your application processes a lot of data or does a lot of parsing, larger heaps may be needed. So how do we set the heap size? By setting its minimum and maximum size. The minimum size is set using the -Xms JVM parameter and the maximum size is set using the -Xmx parameter. to set the heap size of 2GB we would add -Xms2g -Xmx2g to our application startup parameters.*

*Serial Garbage Collector*

*The Serial Garbage Collector is the simplest, single-threaded garbage collector. You can turn on the Serial garbage collector by adding the -XX:+UseSerialGC flag to your JVM application startup parameters.*

*Parallel Garbage Collector*

*The Parallel garbage uses multiple threads to perform garbage collection on your application heap. You can turn on the Parallel garbage collector by adding the -XX:+UseParallelGC flag to your JVM application startup parameters. To disable it entirely, use the -XX:-UseParallelGC flag. The number of threads that the garbage collector can use is set by using the -XX:ParallelGCThreads flag added to our application startup parameters.* *if we would like 4 threads to do the garbage collection*

*-XX:ParallelGCThreads=4*

*The second option that can be used is -XX:MaxGCPauseMillis. It specifies the maximum pause time goal between two consecutive garbage collection events. It is defined in milliseconds. For example, with a flag -XX:MaxGCPauseMillis=100 we tell the Parallel garbage collector that we would like to have the maximum pause of 100 milliseconds between garbage collections.*

*Concurrent Mark Sweep Garbage Collector*

*The Concurrent Mark Sweep garbage collector, a mostly concurrent implementation that shares the threads used for garbage collection with the application. You can turn it on by adding the -XX:+UseConcMarkSweepGC flag to your JVM application startup parameters.*

*Tuning the Concurrent Mark Sweep Garbage Collector - CMS garbage collector is generational which means that you can expect two types of events to happen – minor and major collections. During normal work, most of the garbage collection is done without stopping application threads. CMS only stops the threads for a very short period of time at the beginning and the middle of the collection during the major collection. Minor collections are done in a very similar way to how the Parallel garbage collector works – all application threads are stopped during GC.*

*G1 Garbage Collector*

*G1 garbage collector, the default garbage collector in the newest Java versions targeted for latency-sensitive applications. You can turn it on by adding the -XX:+G1GC flag to your JVM application startup parameters.*

*Tuning G1 Garbage Collector*

*There are also two things worth mentioning. The G1 garbage collector tries to perform longer operations in parallel without stopping the application threads. The G1 garbage collector cleans memory mostly in evacuation fashion. Live objects from one memory area are copied to a new area and compacted along the way. After the process is done, the memory area from which the object was copied is again available for object allocation.*

*G1GC goes between two phases. The first phase is called young-only and focuses on the young generation space. During that phase, the objects are moved gradually from the young generation to the old generation space. The second phase is called space reclamation and is incrementally reclaiming the space in the old generation while also taking care of the young generation at the same time. Let’s look closer at those phases as there are some properties we can tune there.*

*The young-only phase starts with a few young-generation collections that promote objects to the tenured generation.*

*Z Garbage Collector*

*Still experimental, very scalable and low latency implementation. If you would like to experiment with that Z garbage collector you must use JDK 11 or newer and add the -XX:+UseZGC flag to your application startup parameters along with the -XX:+UnlockExperimentalVMOptions flag as the Z garbage collector.*