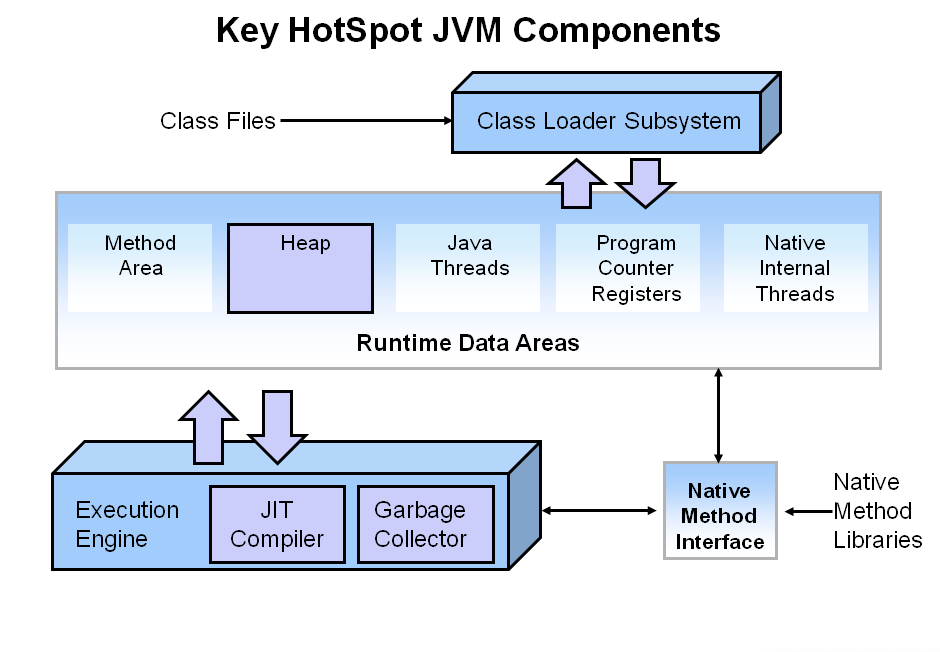
##### **JVM components**



The heap is where your object data is stored. This area is then managed by the garbage collector selected at startup. Most tuning options relate to sizing the heap and choosing the most appropriate garbage collector for your situation.

Garbage Collection

Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

Java GC uses Mark and Sweep strategy to clean the heap.

* Mark – it is where the garbage collector identifies which pieces of memory are in use and which are not
* Sweep – this step removes objects identified during the “mark” phase

JVM has five types of GC implementations:

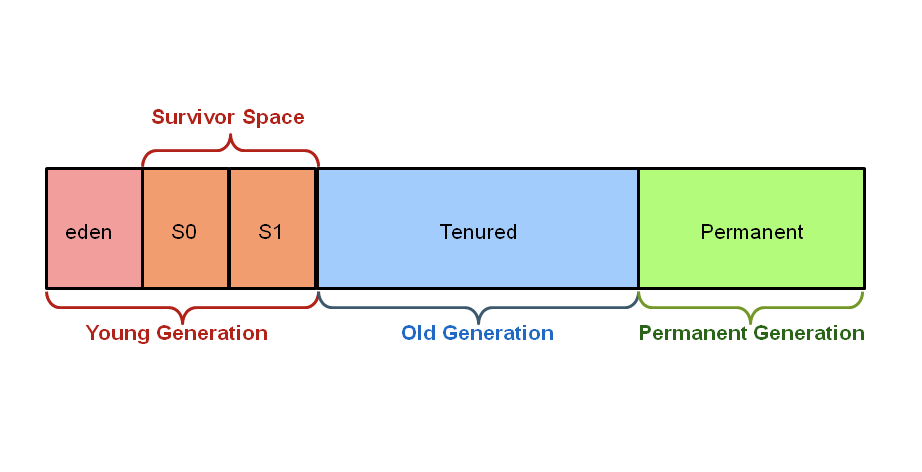
* Serial Garbage Collector
* Parallel Garbage Collector
* CMS Garbage Collector
* G1 Garbage Collector
* Z Garbage Collector

Each of them has its own approaches differ from each other in terms of performance, threading, concurrency. Each version of java also choose different default GC.

##### **JVM Generations**

As stated earlier, having to mark and compact all the objects in a JVM is inefficient. As more and more objects are allocated, the list of objects grows and grows leading to longer and longer garbage collection time. However, empirical analysis of applications has shown that most objects are short lived

The heap is broken up into smaller parts or generations. The heap parts are: Young Generation, Old or Tenured Generation, and Permanent Generation( before java 8).



Note: From Java 8, permanent generation is removed.

The ****Young Generation**** is where all new objects are allocated and aged. When the young generation fills up, this causes a **minor garbage collection**. Minor collections can be optimized assuming a high object mortality rate. A young generation full of dead objects is collected very quickly. Some surviving objects are aged and eventually move to the old generation.

The ****Old Generation**** is used to store long surviving objects. Typically, a threshold is set for young generation object and when that age is met, the object gets moved to the old generation. Eventually the old generation needs to be collected. This event is called a **major garbage collection**.

 The ****Permanent generation**** contains metadata required by the JVM to describe the classes and methods used in the application. The permanent generation is populated by the JVM at runtime based on classes in use by the application. In addition, Java SE library classes and methods may be stored here. From Java 8, Permanent generation is replaced with MetaSpace

Metaspace is native memory grows automatically by default.

[Click here for more detail about PermGen vs Metaspace](https://www.baeldung.com/java-permgen-metaspace" \o "PermGen vs Metaspace" \t "https://lms.antra.com/main/course/_blank)

[Official GC tuning document](https://docs.oracle.com/javase/9/gctuning/toc.htm" \o "Official GC tuning document" \t "https://lms.antra.com/main/course/_blank)