

Garbage Collector- Daemon thread- having less priorty , always executes on back side and provide facility to current thread.

Student s =new Student();

Heap – object create for Student class( rollno, name)🡪 reference – 1098

s- 1098

what is s?

reference object which stores address of 1 object from heap

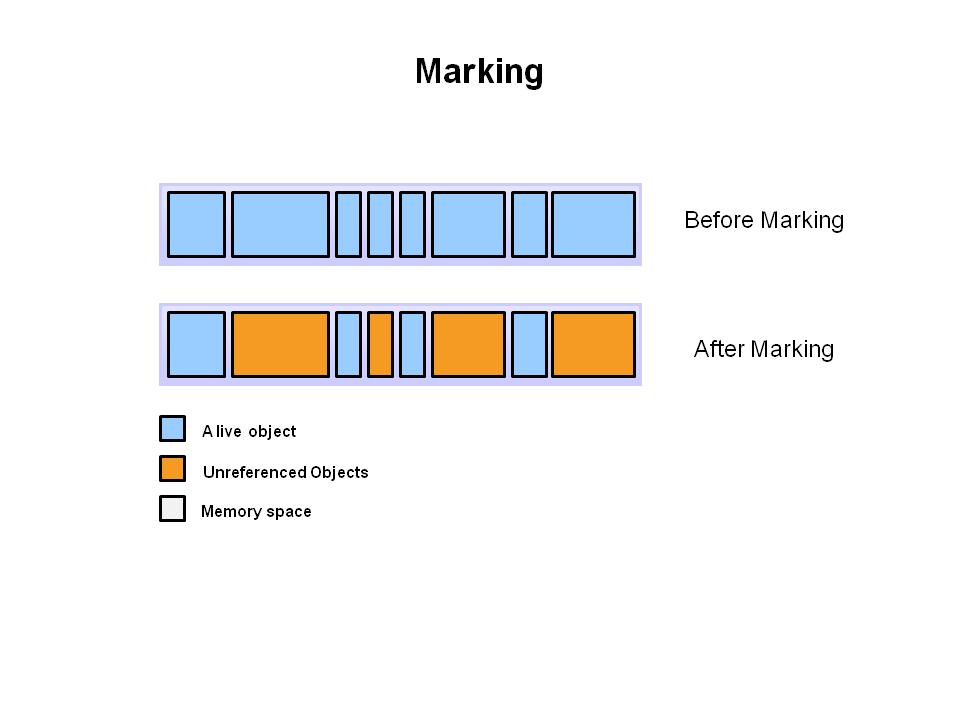
s object will store in jvm stack

s=null🡪 gc can release memory of 1098

**Process of Garbage Collecting**

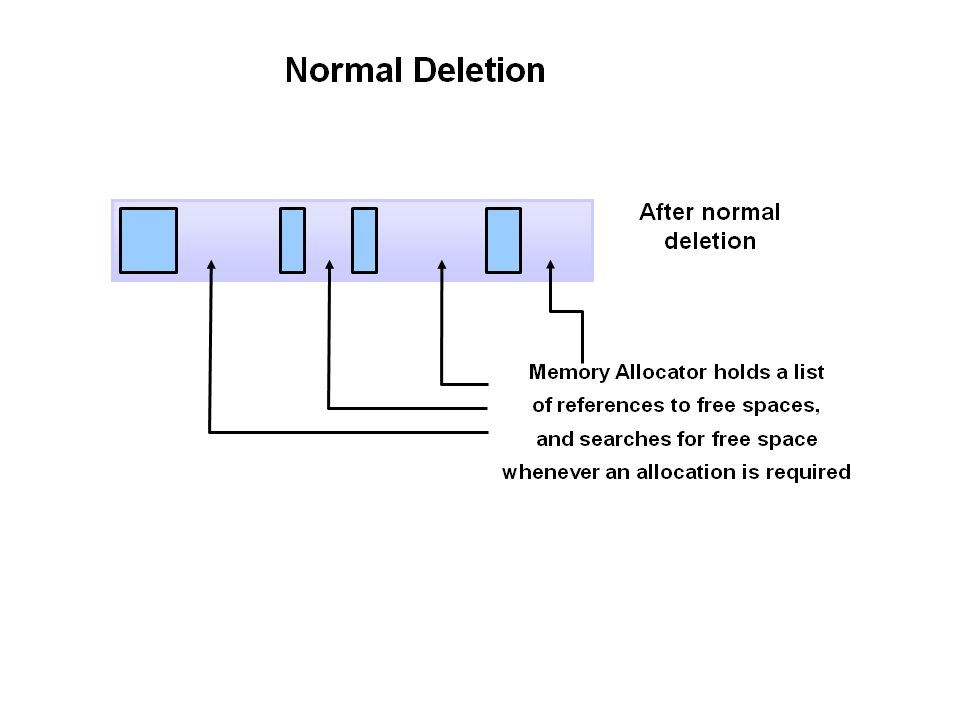
**Step 1: Marking**

The first step in the process is called marking. This is where the garbage collector identifies which pieces of memory are in use and which are not.



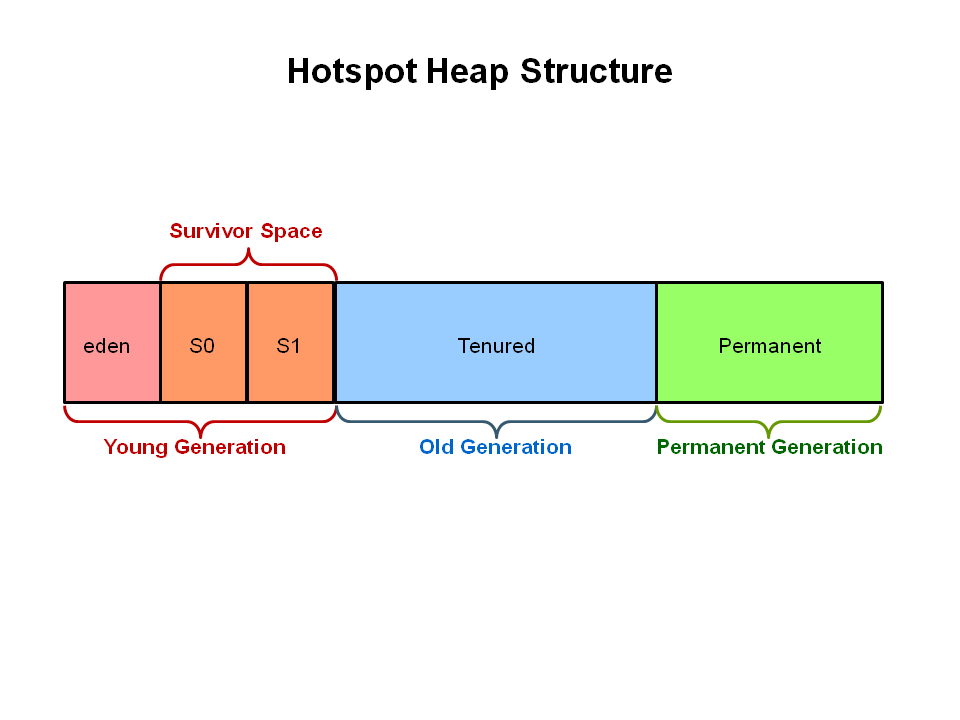
#### Step 2: Normal Deletion

Normal deletion removes unreferenced objects leaving referenced objects and pointers to free space.



**Jvm generation**

 the heap is broken up into smaller parts or generations. The heap parts are: Young Generation, Old or Tenured Generation, and Permanent Generation



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.

## **GC Implementations**

JVM has five types of GC implementations:

* Serial Garbage Collector

t basically works with a single thread. As a result, **this GC implementation freezes all application threads when it runs**. Hence, it is not a good idea to use it in multi-threaded applications like server environments.

* Parallel Garbage Collector

this **uses multiple threads for managing heap space**.

we can specify maximum garbage collection threads and pause time, throughput, and footprint (heap size).

* CMS Garbage Collector

**Concurrent Mark Sweep (CMS) implementation uses multiple garbage collector threads for garbage collection.** It's designed for applications that prefer shorter garbage collection pauses, and that can afford to share processor resources with the garbage collector while the application is running.

System.gc();

G1 Garbage Collector

* **G1 (Garbage First)** Garbage Collector is designed for applications running on multi-processor machines with large memory space.

G1 collector partitions the heap into a set of equal-sized heap regions, each a contiguous range of virtual memory. When performing garbage collections, G1 shows a concurrent global marking phase (i.e. phase 1 known as Marking) to determine the liveness of objects throughout the heap.

After the mark phase is completed, G1 knows which regions are mostly empty. It collects in these areas first, which usually yields a significant amount of free space (i.e. phase 2 known as Sweeping). It is why this method of garbage collection is called Garbage-First.

* Z Garbage Collector (ZGC)—JDK 11

ZGC performs all expensive work concurrently, without stopping the execution of application threads for more than 10 ms,