How Garbage Collection Works in Java?

n **Java, garbage collection** is the process of managing memory, automatically. It finds the unused objects (that are no longer used by the program) and delete or remove them to free up the memory. The **garbage collection** mechanism uses several GC algorithms. The most popular algorithm that is used is **Mark and Sweep**.

## **Garbage Collector Overview**

When a program executes in [Java](https://www.javatpoint.com/java-tutorial), it uses memory in different ways. The heap is a part of memory where objects live. It's the only part of memory that involved in the garbage collection process. It is also known as **garbage collectible heap**. All the garbage collection makes sure that the heap has as much free space as possible. The function of the garbage collector is to find and delete the objects that cannot be reached.

### **Important Points About Garbage Collector**

* It is controlled by a thread known as **Garbage Collector**.
* Java provides two methods **System.gc()** and **Runtime.gc()** that sends request to the JVM for garbage collection. Remember, it is not necessary that garbage collection will happen.
* Java programmer are free from memory management. We cannot force the garbage collector to collect the garbage, it depends on the JVM.
* If the Heap Memory is full, the JVM will not allow to create a new object and shows an error **java.lang.OutOfMemoryError**.
* When garbage collector removes object from the memory, first, the garbage collector thread calls the finalize() method of that object and then remove.

## **Object Allocation**

The small objects are stored in **Thread Local Area** (TLA) which is a free chunk of the heap. TLA does not synchronize with other threads. When TLA becomes full, it requests for new TLA.

On the other hand, large objects that do not fit inside the TLA directly allocated into the heap.

### **When an Object becomes eligible for Garbage Collection?**

An object become eligible if it is not used by any program or thread or any static references or its references is null.

What does Java Garbage Collector?

JVM controls the garbage collector. JVM decides when to perform the garbage collection. You can also request to the JVM to run the garbage collector. But there is no guarantee under any conditions that the JVM will comply. JVM runs the garbage collector if it senses that memory is running low.

**when an object becomes eligible for garbage collection?"**

an object is eligible for garbage collection when no live thread can access it. The garbage collector considers that object as eligible for deletion. If a program has a reference variable that refers to an object, that reference variable available to live thread, this object is called **reachable**.

## **Types of Garbage Collection**

## **Garbage Collection**

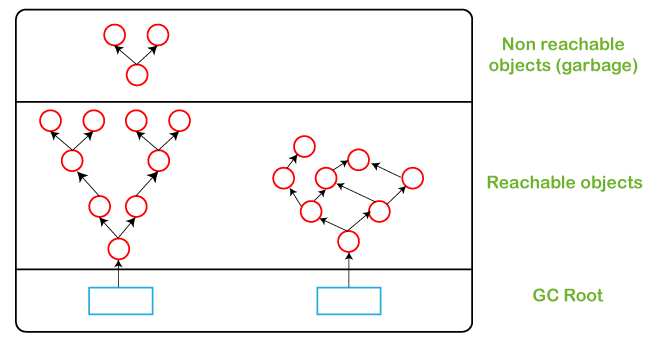
There are five types of garbage collection are as follows:

* **Serial GC:** It uses the mark and sweeps approach for young and old generations, which is minor and major GC.
* **Parallel GC:** It is similar to serial GC except that, it spawns N (the number of CPU cores in the system) threads for young generation garbage collection.
* **Parallel Old GC:** It is similar to parallel GC, except that it uses multiple threads for both generations.
* **Concurrent Mark Sweep (CMS) Collector:** It does the garbage collection for the old generation. You can limit the number of threads in CMS collector using XX:ParalleCMSThreads=JVM option. It is also known as Concurrent Low Pause Collector.
* **G1 Garbage Collector:** It introduced in Java 7. Its objective is to replace the CMS collector. It is a parallel, concurrent, and CMS collector. There is no young and old generation space. It divides the heap into several equal-sized heaps. It first collects the regions with lesser live data

## **Mark and Sweep Algorithm**

JRockit JVM uses the mark and sweep algorithm for performing the garbage collection. It contains two phases, the mark phase, and the sweep phase.

**Mark Phase:** Objects that are accessible from the threads, native handles, and other GC root sources are marked as live. Every object tree has more than one root object. GC root is always reachable. So, any object that has a garbage collection root at its root. It identifies and marks all objects that are in use, and the remaining can be considered garbage.



**Sweep Phase:** In this phase, the heap is traversed to find the gap between the live objects. These gaps are recorded in the free list and are available for new object allocation.

There are two improved versions of mark and sweep:

* **Concurrent Mark and Sweep**
* **Parallel Mark and Sweep**

### **Concurrent Mark and Sweep**

It allows the threads to continue running during a large portion of the garbage collection. There are following types of marking:

* **Initial marking:** It identifies the root set of live objects. It is done while threads are paused.
* **Concurrent marking:** In this marking, the reference from the root set are followed. It finds and marks the rest of the live objects in a heap. It is done while the thread is running.
* **Precleaning marking:** It identifies the changes made by concurrent marking. Other live objects marked and found. It is done while the threads are running.
* **Final marking:** It identifies the changes made by precleaning marking. Other live objects marked and found. It is done while threads are paused.

### **Parallel Mark and Sweep**

It uses all available [CPU](https://www.javatpoint.com/cpu-full-form) in the system for performing the garbage collection as fast as possible. It is also called the parallel garbage collector. Threads do not execute when the parallel garbage collection executes.

**Pros of Mark and Sweep**

* It is a recurring process.
* It is an infinite loop.
* No additional overheads allowed during the execution of an algorithm.

**Cons of Mark and Sweep**

* It stops the normal program execution while the garbage collection algorithm runs.
* It runs multiple times on a program.