# Internal Working of Garbage Collector in Java

## 1. What is Garbage Collection (GC)?

Garbage Collection automatically reclaims memory occupied by objects that are no longer in use. It helps in preventing memory leaks and optimizing memory usage.

## 2. How GC Works?

GC identifies and removes unreachable objects. Objects are considered unreachable if they have no references pointing to them.

## 3. Phases of Garbage Collection

✅ Mark Phase  
- Identifies live objects that are still in use.  
- Marks all reachable objects from GC roots.

✅ Sweep Phase  
- Removes unreferenced (unreachable) objects.  
- Frees the memory occupied by the unreferenced objects.

✅ Compact Phase (Optional)  
- Rearranges remaining objects to reduce fragmentation.  
- Improves cache locality and reduces overhead.

## 4. Types of Garbage Collectors in Java

⚡ Serial GC (-XX:+UseSerialGC)  
- Suitable for small applications with single-threaded environments.

⚡ Parallel GC (-XX:+UseParallelGC)  
- Multiple threads perform GC.  
- Suitable for multi-core processors.

⚡ CMS (Concurrent Mark Sweep) GC (-XX:+UseConcMarkSweepGC)  
- Performs GC with minimal pause time.  
- Preferred for applications requiring low-latency.

⚡ G1 GC (Garbage First) (-XX:+UseG1GC)  
- Divides heap into regions and prioritizes garbage collection.  
- Suitable for large heap sizes and real-time applications.

## 5. Heap Memory Structure

- Young Generation: Contains new objects.  
 - Eden Space: New objects are allocated.  
 - Survivor Space: Holds objects that survive multiple GCs.

- Old Generation (Tenured): Stores long-lived objects.

- Permanent Generation (MetaSpace in Java 8+): Stores metadata, class structures, etc.

## 6. Triggers for GC

- Low memory conditions.  
- Explicitly calling System.gc() (not recommended).  
- JVM’s internal algorithms.

## 7. Finalization and GC

finalize() is called before reclaiming memory, but it is unpredictable and not guaranteed.

## 8. Tuning GC Performance

Use JVM options to optimize GC:

-XX:+UseG1GC  
-XX:MaxHeapSize=512m  
-XX:NewRatio=2  
-XX:SurvivorRatio=8

## Important Points

- GC cannot be explicitly controlled.  
- GC pauses may affect application performance.  
- Avoid excessive object creation to reduce GC overhead.

👍 Best Practices:  
- Minimize object creation.  
- Use memory-efficient data structures.  
- Monitor and tune GC performance.