Garbage Collection

1. Make an Object Eligible for Garbage Collection

In Java, an object becomes **eligible for GC** when **no live reference** points to it.

Ways to make objects eligible:

a) Assign the reference variable to null

```
Example e = new Example();
e = null;     // object becomes eligible for GC
```

b) Reassign the reference variable

```
Example e1 = new Example();
Example e2 = new Example();
e1 = e2;      // first object becomes eligible for GC
```

c) Objects created inside methods

They become eligible when method execution completes.

```
void test() {
   Example e = new Example(); // eligible for GC after method ends
}
```

---

2. Requesting JVM to Run Garbage Collector

Java provides two ways to **request** GC:

a) System.gc()

```
System.gc();
```

b) Runtime.getRuntime().gc()

```
Runtime.getRuntime().gc();
```

Important:
These methods *request* GC, but **JVM may not run it immediately**. It's *not guaranteed*.

---

3. How and When to Use Finalization

What is finalization?

Before destroying an object, JVM may call finalize() method (deprecated in Java 9, removed in Java 18).

```
@Override
protected void finalize() throws Throwable {
   System.out.println("finalize called");
}
```

When does it run?

- Called *once* before the object is destroyed by GC.
- Not guaranteed to run immediately.
- Not guaranteed to run at all.

When to use?

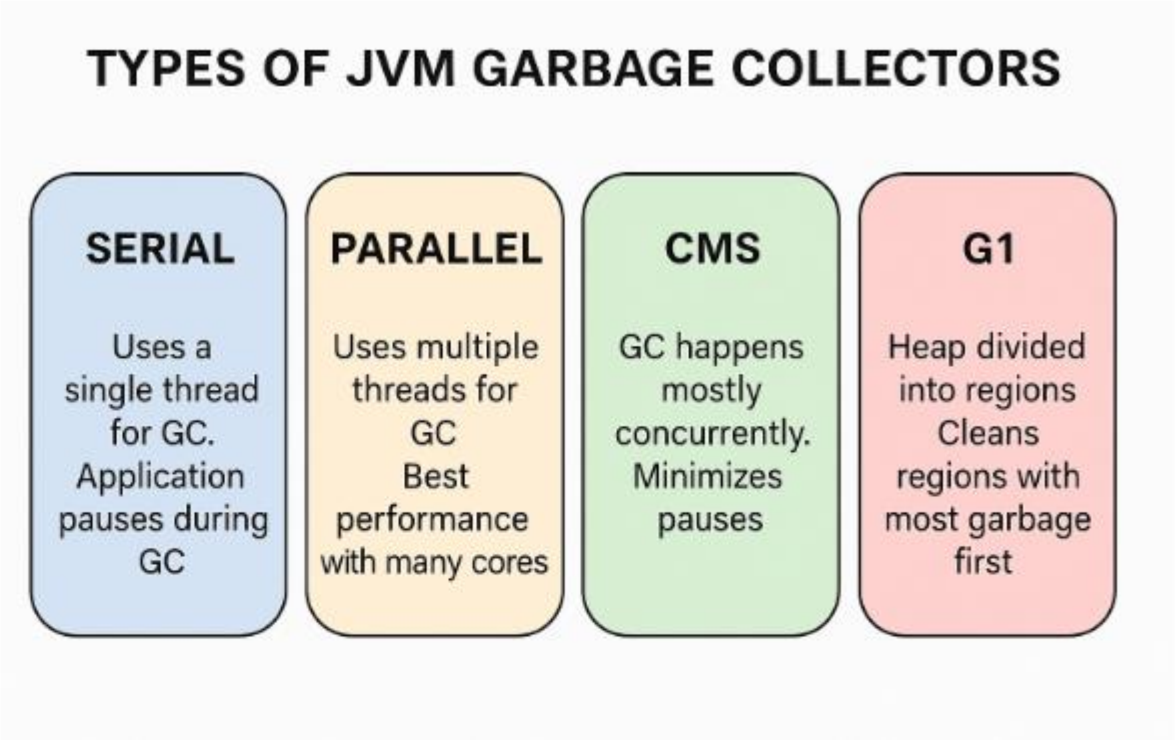
Modern Java: DO NOT USE finalize()
It is unreliable and slow.

Alternative?

Use:

- **try-with-resources**
- **Cleaner API**
- **Explicit close() methods**

4. Types of JVM Garbage Collectors (GCs)



TYPES OF JVM GARBAGE COLLECTORS

SERIAL

Uses a single thread for GC. Application pauses during GC

PARALLEL

Uses multiple threads for GC Best performance with many cores

CMS

GC happens mostly concurrently. Minimizes pauses

G1

Heap divided into regions Cleans regions with most garbage first

1. Serial Garbage Collector

How it works:

- Uses a **single thread** for GC.
- Application pauses during GC.

Good for:

- Small applications
- Single-core machines

Set by:

-XX:+UseSerialGC

---

2. Parallel Garbage Collector (Throughput GC)

How it works:

- Uses **multiple threads** for GC.
- Best performance when many cores available.

Good for:

- High-throughput applications
- Multi-core CPUs

Set by:

-XX:+UseParallelGC

---

3. CMS (Concurrent Mark Sweep) Collector

How it works:

- GC happens **mostly concurrently** with application.
- Minimizes pauses.

Good for:

- Low-latency applications

Set by:

-XX:+UseConcMarkSweepGC

Deprecated since Java 9.

---

4. G1 (Garbage First) Collector

How it works:

- Heap divided into regions.
- Cleans regions with most garbage first.
- Predictable pause times.

Good for:

- Large heaps (4GB+)
- Modern servers

Default in Java 9+.

Set manually:

-XX:+UseG1GC

---

5. ZGC (Z Garbage Collector)

How it works:

- **Ultra-low pause time** (<10ms)
- Works concurrently with application

Good for:

- Very large heaps (multi-GB)
- Real-time systems

-XX:+UseZGC

---

6. Shenandoah GC

How it works:

- Similar to ZGC (region-based)
- Low pause time (10ms or less)

Good for:

- Large heap, low latency needs

-XX:+UseShenandoahGC

Summary Table

GC Type	Threads	Pause Time	Best For
Serial	Single	High	Small apps, single-core
Parallel	Multi	Medium	High throughput
CMS	Multi	Low	Low-latency (deprecated)
G1	Multi	Predictable low	Default, large heaps
ZGC	Multi	Ultra-low	Huge heaps
Shenandoah	Multi	Ultra-low	Low-latency, large heaps