1Z0-808 Exam Topic Reviewer

TopicId: 1018
Topic: Garbage Collection and Object Lifecycle

August 5, 2025

Java's Automatic Memory Management

Alright team, let's discuss one of Java's most celebrated features: automatic garbage collection. In languages like C++, you are responsible for manually allocating and deallocating memory. Forgetting to free memory leads to memory leaks; freeing it too early leads to crashes. Java saves us from this headache with its Garbage Collector (GC). The GC is a low-priority background process in the JVM that automatically identifies and reclaims memory from objects that are no longer in use. For the 1Z0-808 exam, you don't need to know the complex algorithms behind the GC. Your job is to know the answer to one critical question: **When does an object become eligible for garbage collection?**

1 Object Reachability: The Key to Eligibility

The entire concept boils down to *reachability*. The JVM considers certain variables as "GC Roots"—these are special variables that are assumed to be reachable, such as local variables on the current thread's stack and static variables.

- **Reachable Object:** An object is considered "alive" if the GC can trace a path of references to it, starting from any GC Root.
- **Unreachable Object:** An object becomes **eligible for garbage collection** when it is no longer reachable. That is, there are no more reference paths from any GC Root to the object.

How an Object Becomes Unreachable

The exam will test your ability to spot when an object's last reference is lost. Here are the common scenarios:

(a) **Nullifying a Reference:** Explicitly setting a reference variable to `null` breaks the link to the object.

```
String s = new String("Hello");
// At this point, the "Hello" object is reachable via 's'.
s = null;
// Now, if 's' was the only reference, the object is eligible for GC.
```

(b) **Re-assigning a Reference:** Pointing a reference variable to a different object abandons the original object.

```
Person p1 = new Person("Alice");
Person p2 = new Person("Bob");
p1 = p2; // The object "Alice" is now eligible for GC (if p1 was the only re
```

(c) **Object Goes Out of Scope:** When a method finishes executing, all of its local variables are destroyed. Any objects that were only referenced by those local variables become eligible.

```
public void process() {
    Report r = new Report(); // The Report object is created.
    r.generate();
} // When process() ends, 'r' is destroyed. The Report object is now eligibl
```

(d) **Island of Isolation:** This is a classic exam trap. Two or more objects can reference each other, but if they are not reachable from any GC Root, the entire group (or "island") is eligible for collection.

```
class Island { Island i; }

Island i1 = new Island();
Island i2 = new Island();
i1.i = i2; // i1 points to i2
i2.i = i1; // i2 points to i1

i1 = null;
i2 = null;
// Even though the two objects still reference each other, they form an
// unreachable island and are both eligible for GC.
```

2 Interacting with the GC

You have very limited control over the GC, and the exam loves to test these limits.

`System.gc()`

This static method can be used to *suggest* that the JVM run the garbage collector.

Critical Exam Point: Calling `System.gc()` provides **no guarantee whatsoever**. The JVM may choose to ignore the request entirely. The GC may or may not run, and if it does, there's no telling when. Any exam question that implies a guaranteed, immediate action from `System.gc()` is a trap.

The `finalize()` Method

The `Object` class provides a method: `protected void finalize() throws Throwable`. The GC calls this method on an object just before it reclaims its memory.

Key Behaviors for the Exam:

- **No Guarantees:** Just like `System.gc()`, there is no guarantee that `finalize()` will ever be called for a given object. The program might exit before the object is collected.
- **Called At Most Once:** The JVM will call `finalize()` at most one time for any given object.

- **Resurrection:** An object can "save itself" from collection within its `finalize()` method by making itself reachable again (e.g., assigning `this` to a static variable). If it does this, the GC will not collect it in that cycle. If the object later becomes unreachable again, it will be collected *without* `finalize()` being called a second time.
- **Modern Practice:** The `finalize()` method is considered a poor mechanism for resource cleanup. Modern Java strongly prefers using `try-with-resources` statements for managing resources like files and network connections.

3 Key Takeaways for the 1Z0-808 Exam

- An object is eligible for GC when it becomes **unreachable**.
- Be able to spot the four main ways an object becomes unreachable.
- `System.gc()` is only a **suggestion**.
- `finalize()` is **not guaranteed to run** and runs **at most once**.