**Garbage Collection**

**Introduction:**

* Garbage collector always runs in the background (demon thread) to destroy the useless objects which means programmer is not responsible to destroy them.
* By this chance of failing java program with memory problems is very low.

**Ways to Make an Object eligible for GC:**

* It is recommended that Programmer has to make an object eligible for GC if it is no longer required.
* If object doesn’t have any reference, then it is eligible for GC.
* There are 4 methods to make an object eligible for GC

1. **Nullifying the Reference Variable:**
   * If object no longer required, then assign null to all its reference variables.

**A close up of text on a whiteboard

Description generated with high confidence**

1. **Re-assigning the Reference Variable:**
   * If an object no longer required, then re-assign its reference variable to some other variable then old object by default eligible for garbage collection.

**A close up of text on a whiteboard

Description generated with high confidence**

1. **Objects inside method:**
   * The objects created inside method are by default eligible for GC once method completes, because local variables are destroyed when method completes.

**A close up of text on a whiteboard

Description generated with high confidence**

**A close up of text on a whiteboard

Description generated with very high confidence**

1. **Island of Isolation:**
   * Even though there are internal references, if there are no external references then are objects are eligible for GC called as Island of Isolation.

**A close up of text on a whiteboard

Description generated with very high confidence**

**Methods to run JVM to run GC:**

* We can request JVM to run GC using some methods, weather JVM accepts it or not is depends upon the JVM. Mostly 99% times JVM accepts the request.
* Ways for requesting JVM to run GC.
  1. By using System Class:
     + It contains a static method “gc” for this purpose (**System.gc())**
  2. By using Runtime Class (in lang package and singleton class)
     + Java application can communicate with JVM using this object.
     + to create object for singleton class
       - Runtimr r = Runtime.getRuntime();
     + r.totalMemory();
       - it returns total number of bytes of memory present in Heap
     + r.freeMemory();
       - it returns total number of bytes of free memory present in Heap
     + r.gc();
       - Requesting to run GC.

**A screen shot of a computer

Description generated with high confidence**

* gc method present in system class is static method whereas gc method in Runtime class is instance method

A close up of text on a whiteboard

Description generated with high confidence

* It is recommended to use Runtime.gc because internally static method system calls runtime object.

A close up of text on a whiteboard

Description generated with high confidence

**Finalization:**

* GC calls finalize method to clean up activities just before destroying an object.
* Finalize method is defined in object class.
* Protected void finalize() throws Throwable is the signature of the method, we can override this method in our class to define our own clean up activities.

**A screenshot of a cell phone

Description generated with very high confidence**

Here there are two threads, one is main thread and other is demon thread which runs in background when we called garbage collector. As GC calls finalize method and main thread executes rest of code we cannot expect which one comes first on the console.

The out of this program is just “end of main”.

Note: GC may or may not respond every time.

**Case2:**

* We can call finalize method explicitly in program, but object won’t be destroyed.

A screenshot of text

Description generated with very high confidence

* Init, service, destroy methods are considered as life cycle methods of servlets, just before destroying servlets web container calls destroy method. We can also call destroy method from init a service method, it acts like normal method call and servlet won’t be destroyed.
* **Even though an object is eligible for GC multiple times. JVM calls finalize method only once.**

**A screenshot of a computer screen

Description generated with very high confidence**

Here object eligible for garbage collection for 2 times, but finalize method is called only once.

We can’t expect exact behavior of garbage collector, it is varied from Jvm to Jvm. For these statements there are no correct answers,

1. When exactly JVM runs GC.
2. In which order GC identifies eligible objects.
3. In which order destroys eligible objects.
4. Weather GC destroys all eligible objects not.
5. What is the algorithm? (most follow **Mark and Sweep**)

Whenever program runs with low memory JVM runs GC.

**Memory Leaks:** The objects which are not using in our program and which are not eligible for GC. Such types of useless objects are called memory leaks. In our program if such memory leaks are present the program is terminated with OutOfMemoryError.

There are some third-party memory management tools which finds memory leaks in our program.

* HP OVO
* HP J Meter
* JProbe
* Patrol
* IBM Tivoli