Java Garbage Collection

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

In C language using free() function and delete() in C++ constructor and destructor are used for for creation of new objects and destroying an object.

But, in java it is performed automatically.

* Java provides better memory management, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.
* Garbage collector is best example of [Daemon thread](https://www.geeksforgeeks.org/daemon-thread-java/) as it is always running in background.
* Main objective of Garbage Collector is to free heap memory by destroying unreachable / no longer used objects.

**Advantage of Garbage Collection**

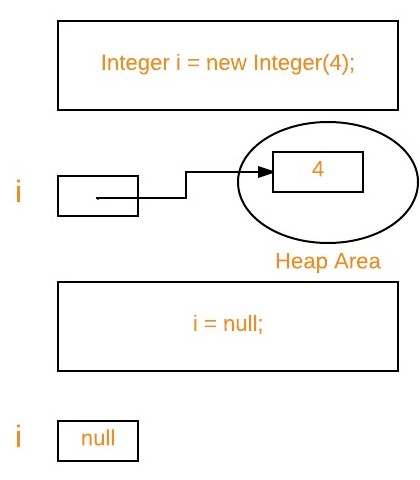
* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Integer i = new Integer(4);

// the new Integer object is reachable via the reference in 'i'

**i = null;**

// the Integer object is no longer reachable.



1. **Eligibility for garbage collection :** An object is said to be eligible for GC(garbage collection) iff it is unreachable. In above image, after

*i = null;*

ex: integer object 4 in heap area is eligible for garbage collection.

**Making an object eligible for GC**

Even though the programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.

* There are generally four different ways to make an object eligible for garbage collection.
  + 1. Nullifying the reference variable
    2. Re-assigning the reference variable
    3. Object created inside method
    4. [Island of Isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/)

All above ways with examples are discussed, [How to make object eligible for garbage collection](https://www.geeksforgeeks.org/how-to-make-object-eligible-for-garbage-collection/)

**Ways for requesting**[**JVM**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)**to run Garbage Collector**

* Once we made object eligible for garbage collection, it may not destroy immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. But when JVM runs Garbage Collector, we can not expect.
* We can also request JVM to run Garbage Collector. There are two ways to do it :
  + 1. Using System.gc() method : System class contain static method gc() for requesting JVM to run Garbage Collector.
    2. Using Runtime.getRuntime().gc() method : [Runtime class](https://www.geeksforgeeks.org/java-lang-runtime-class-in-java/) allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

|  |
| --- |
| // Java program to demonstrate requesting  // JVM to run Garbage Collector  public class Test  {      public static void main(String[] args) throws InterruptedException      {          Test t1 = new Test();          Test t2 = new Test();            // Nullifying the reference variable          t1 = null;            // requesting JVM for running Garbage Collector          System.gc();             // Nullifying the reference variable          t2 = null;            // requesting JVM for running Garbage Collector          Runtime.getRuntime().gc();        }        @Override      // finalize method is called on object once      // before garbage collecting it      protected void finalize() throws Throwable      {          System.out.println("Garbage collector called");          System.out.println("Object garbage collected : " + this);      }  } |

* + 1. Output:
    2. Garbage collector called
    3. Object garbage collected : Test@46d08f12
    4. Garbage collector called
    5. Object garbage collected : Test@481779b8
* Note :
  + 1. There is no guarantee that any one of above two methods will definitely run Garbage Collector.
    2. The call System.gc() is effectively equivalent to the call : Runtime.getRuntime().gc()

**Finalization**

* Just before destroying an object, Garbage Collector calls finalize() method on the object to perform cleanup activities. Once finalize() method completes, Garbage Collector destroys that object.
* finalize() method is present in [Object class](https://www.geeksforgeeks.org/object-class-in-java/) with following prototype.
* protected void finalize() throws Throwable

Based on our requirement, we can override *finalize()* method for perform our cleanup activities like closing connection from database.

**Note :**

* 1. The *finalize()*method called by Garbage Collector not [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). Although Garbage Collector is one of the module of JVM.
  2. [Object class](https://www.geeksforgeeks.org/object-class-in-java/) *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
  3. The *finalize()*method is never invoked more than once for any given object.
  4. If an uncaught exception is thrown by the *finalize()* method, the exception is ignored and finalization of that object terminates.