Garbage Collector in JAVA

- In C/C++, programmer is responsible for both creation and destruction of objects.
- But in Java, 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** as it is always running in background.
- Daemon Thread: Daemon thread is a **low priority** thread that runs in background to perform tasks such as garbage collection.
- Main objective of Garbage Collector is to free heap memory by **destroying unreachable** / **unreferenced objects**.
- Unreachable Objects : An object is said to be unreachable iff it **doesn't contain any reference** to it.
  Example:

  1. By nulling the reference
     Employee e = new Employee(); // Employee object created in Heap Memory.
     E = null; // Now above created Employee object has no reference (is unreachable).

  2. By assigning reference to other
     Employee e1 = new Employee(); Employee e2 = new Employee();
     e1 = e2; // Now reference to first object has been lost.

  3. By anonymous object
     new Employee();

  4. Object Created Inside a method

  5. Island of Isolation: It is a group of objects that **reference each other** but none of these objects are referenced by any active object in application. It can be understood as a Connected Component disconnected from active objects. Even a single unreferenced object can be called Island of Isolation.

- Eligibility for garbage collection : An object is said to be eligible for GC(garbage collection) iff it is **unreachable**.
- Ways to make an object eligible for GC : These are same as 5 ways already mentioned above.

Ways for requesting JVM to run Garbage Collector :

- Once we made object eligible for garbage collection, it **may not** be **destroyed immediately** by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed and it cannot be predicted.

- We can only **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** 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();

        t1 = null; // Nullifying the reference variable

        System.gc(); // requesting JVM for running Garbage Collector

        t2 = null; // requesting JVM for running Garbage Collector

        Runtime.getRuntime().gc();   // requesting JVM for running Garbage Collector
    }
    // finalize method is called on object once  before garbage collecting it
    protected void finalize() throws Throwable  {
        System.out.println("Garbage collector called");
    }
}
```

Output:
Garbage Collector called
Garbage Collector called

Note:
- There is **no guarantee** that any one of above two methods will definitely run Garbage Collector.
- 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. After that Garbage Collector destroys that object.

- ***finalize()*** method is present in Object Class 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 :

- The *finalize() is* method **called by Garbage Collector** not JVM. Although Garbage Collector is one of the module of JVM.

- **Object Class *finalize()* method has empty implementation**, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.

- The *finalize()* method is **never invoked more than once** for any given object.

- If an **uncaught exception** is thrown by the *finalize()* method, the **exception** is **ignored** and finalization of that object terminates.

Further Study: Output Questions on Garbage Collector https://www.geeksforgeeks.org/output-of-java-programs-set-10-garbage-collection/