Garbage collection

After an object has carried out its purpose, it is often not needed by the program any longer. All objects consume at least some system resources, and it is important to free these resources **when an object is no longer needed**. Whenever an object no longer has any references to it, **the Java runtime system will consider the object unused, and therefore available for garbage collection**.

Automatic garbage collection is a great simplification for the programmer. Unlike in other object-oriented languages such as C++, the Java programmer is not responsible for explicitly freeing resources used by an object. The garbage collector handles all this. However**, having garbage collection is not totally a free ride**. First, **you never can predict when the garbage collector will be invoked, and thus won't know just when an object has its resources freed**. Java provides a special method for each class called finalize(). The finalizer is called when the garbage collector frees the resources of the object. In theory, **this should allow the object to free other resources** it might have used, such as file handles. In practice, this is not something to rely on, and finalize() is in fact relatively worthless for all but a few specialized cases that most programmers will never see. If you have an object that uses system resources like file handles or network connections or whatever, you should be sure that the object frees them when you know you are done with the object, and before you release the last reference to it, and not count on finalize. There is another problem with garbage collection**. When the garbage collector does run, it can more or less freeze a program for a few moments while garbage collection takes place**. As processors get faster, this issue becomes less important, but it is still possible to see a Java program momentarily pause for no apparent reason. This is the garbage collector running.

Garbage Collection in Java

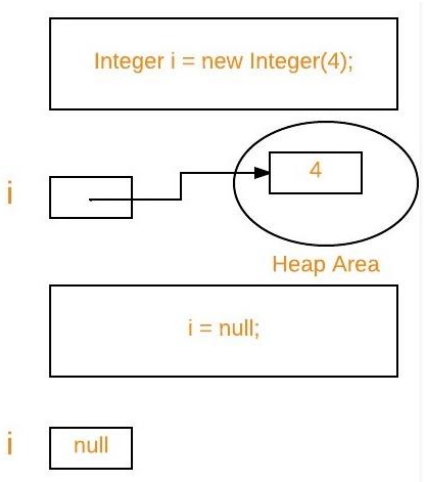
Introduction

- In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing **OutOfMemoryErrors**.
- But in Java, the programmers 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.
- Main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**.

Important terms :

1. **Unreachable objects :** An object is said to be unreachable iff it doesn't contain any reference to it. Also note that objects which are part of island of isolation are also unreachable.

```
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.
```



2. **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;* integer object 4 in heap area is eligible for garbage collection.

Ways to make an object eligible for GC

- Even though 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

  All above ways with examples are discussed in separate article : How to make object eligible for garbage collection.

Ways for requesting JVM to run Garbage Collector

- Once we made object eligible for garbage collection, it may not destroy immediately by garbage collector. Whenever JVM runs Garbage Collector program, then only object will be destroyed. But when JVM runs Garbage Collector, we cannot 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 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);
    }
}
```

- Output:

```
Garbage collector called
Object garbage collected : Test@46d08f12
Garbage collector called
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 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. Although Garbage Collector is one of the module of JVM.
2. 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.
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.

Island of Isolation in Java

In java, object destruction is taken care by the Garbage Collector module and the objects which do not have any references to them are eligible for garbage collection. Garbage Collector is capable to identify this type of objects.

Island of Isolation:

- Object 1 references Object 2 and Object 2 references Object 1. Neither Object 1 nor Object 2 is referenced by any other object. That's an island of isolation.
- Basically, an island of isolation is a group of objects that reference each other but they are not referenced by any active object in the application. Strictly speaking, even a single unreferenced object is an island of isolation too.

```
public class Test {
    Test i;
    public static void main(String[] args){
        Test t1 = new Test();
        Test t2 = new Test();
              // Object of t1 gets a copy of t2
              t1.i = t2;
              // Object of t2 gets a copy of t1
              t2.i = t1;
              // Till now no object eligible
              // for garbage collection
              t1 = null;
              //now two objects are eligible for
              // garbage collection
              t2 = null;
              // calling garbage collector
              System.gc();
    }
    @Override
    protected void finalize() throws Throwable
    {
        System.out.println("Finalize method called");
    }
}
```

Output:

```
Finalize method called
Finalize method called
```

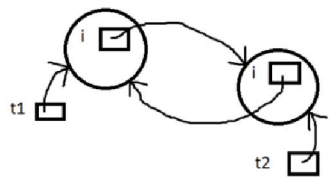
Explanation :

Before destructing an object, Garbage Collector calls finalize method at most one time on that object.

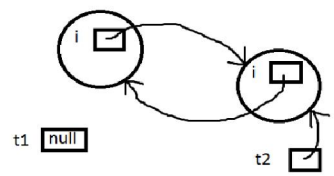
The reason finalize method called two times in above example because two objects are eligible for garbage collection.This is because we don't have any external references to t1 and t2 objects after executing t2=null.

All we have is only internal references(which is in instance variable i of class Test) to them of each other. There is no way we can call instance variable of both objects. So, none of the objects can be called again.

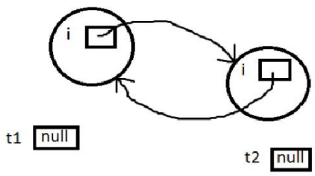
Till t2.i = t1 : Both the objects have external references t1 and t2.



t1 = null : Both the objects can be reached via t2.i and t2 respectively.



t2 = null: No way to reach any of the objects.



Now, both the objects are eligible for garbage collection as **there is no way we can call them**. This is popularly known as **Island of Isolation.**