Garbage Collection

Garbage collection in Java is the process of automatically freeing heap memory by deleting unused objects that are no longer accessible in the program.

In other simple words, the process of automatic reclamation of runtime unused memory is known as garbage collection.

The program that performs garbage collection is called a **garbage collector** or simply a collector in java.

It is a part of the Java platform and is one of the major features of the Java Programming language.

Java
garbage collector runs in the background in a low-priority thread and automatically cleans up heap memory by destroying unused objects.

However,
before destroying unused objects, it makes sure that the running program in its current state will never use them again.

This way, it ensures that the program has no reference variable that does not refer to any object.

Dead object or Garbage in Java

An object that cannot be used in the future by the running program is known as **garbage in java**. It is also known as dead object or unused object.

For example, an object exists in the heap memory, and it can be accessed only through a variable that holds references to that object.

```
Hello h1 = new Hello();
Hello h2 = new Hello();
h1 = h2;
```

Here, we have assigned one reference variable h1 to another reference variable h2. After the assignment statement h1 = h2, h1 refers to the same object referenced by h2 because the reference variable h2 is copied to variable h1. Due to which the reference to the previous object is gone.

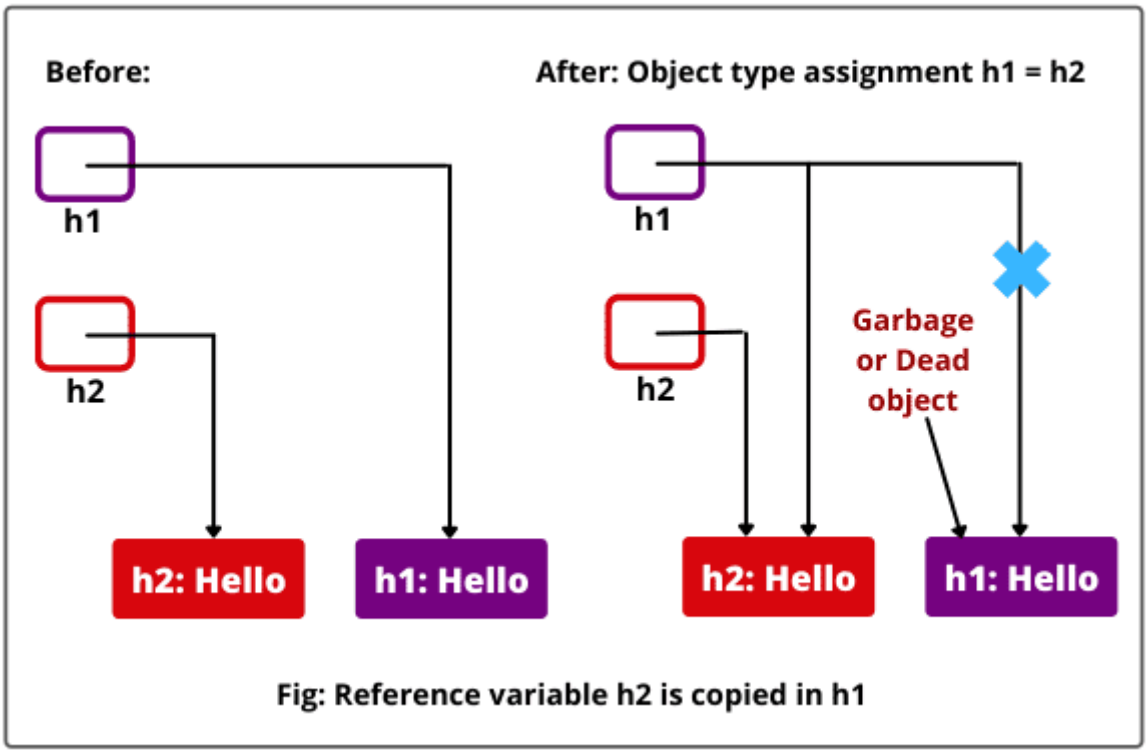

Fig: Reference variable h2 is copied in h1

```
package garbageCollectorProgram;
public class InvokingGC {
public static void main(String[] args)
{
  long m1, m2, m3;
// Call getRuntime() method to get a runtime object.
     Runtime rt = Runtime.getRuntime();
    for(int i = 0; i < 3; i++)
    {
 // Get free memory.
  m1 = rt.freeMemory();
 // Call createObjects() method to create some objects.
  createObjects(500);
 // Get free memory
  m2 = rt.freeMemory();
 // Invoke garbage collection.
  System.gc();
 // Get free memory
  m3 = rt.freeMemory();
  System.out.println("m1 = " + m1 + ", m2 = " + m2 + ", m3 = " +m3 + "\nMemory freed b
y gc() =  " + (m3 - m2));
  System.out.println("-------------------------");
    }
  }
public static void createObjects(int count)
{
 for(int i = 0; i < count; i++)
 {
// Do not store references of new objects, so they are immediately eligible for garbag
e collection.
```

```
     new Object();
  }
 }
}
```

```
package garbageCollectorProgram;
public class GCTest
{
String objRef;
public GCTest(String objRef)
{
    this.objRef = objRef;
}
public static void main(String[] args)
{
GCTest t1 = new GCTest("t1");
GCTest t2 = new GCTest("t2");
   t1 = t2;
// Invoke garbage collection.
      System.gc();
  }
@Override
/* Overriding finalize method to check whether object previously referenced by objRef
 is garbage collected or not. */
protected void finalize() throws Throwable
{
     System.out.println("Object previously referenced by "+ this.objRef + " is success
fully garbage collected.");
 }
}
```

```
package garbageCollectorProgram;
public class GCTest
{
String objRef;
public GCTest(String objRef)
{
   this.objRef = objRef;
}
public static void main(String[] args)
{
GCTest t1 = new GCTest("t1");
     t1 = null;
// Invoke garbage collection.
     System.gc();
  }
@Override
/* Overriding finalize method to check whether the object referenced by objRef is garb
age collected or not. */
protected void finalize() throws Throwable
{
    System.out.println("Object referenced by "+ this.objRef + " is successfully garbag
e collected.");
```

```
 }
}
```

```
package garbageCollectorProgram;
public class GCTest
{
String objRef;
public GCTest(String objRef)
{
   this.objRef = objRef;
}
public static void main(String[] args)
{
     new GCTest("t1"); // Anonymous object.
// Invoking garbage collection.
     System.gc();
  }
@Override
/* Overriding finalize method to check whether anonymous object is garbage collected o
r not. */
protected void finalize() throws Throwable
{
    System.out.println(this.objRef+" is successfully garbage collected.");
 }
}
```

Object Finalization in Java | Finalize, Example

Finalization in Java
 is an action that is automatically performed on an object before the memory occupied by the object is freed up by the garbage collector.

The
 block of code that contains the action to be performed is called finalizer in java. In Java, the finalizer is just opposite to the constructor.

A constructor performs initialization for an object, whereas the finalizer method performs finalization for the object. *Garbage collection* automatically cleans up the memory occupied by unused objects.

But
 when objects hold other kinds of resources such as opening files, closing files, network connection, etc, garbage collection does not free
 these resources for you.

```
package garbageCollectorProgram;
public class Finalizer
{
// Declaring an instance variable id that is used to identify the object.
      private int id;
// Constructor which takes the id as parameter.
    public Finalizer(int id)
    {
    this.id = id;
     }
// This is the finalizer for objects. JVM will call this method, before the object is
 garbage collected.
    public void finalize()
    {
// Here, we will display a message indicating which object is being garbage collected.
// Display message when id is a multiple of 50.
     if (id % 50 == 0)
     {
    System.out.println ("finalize() method called for " + id ) ;
     }
 }
public static void main(String[] args)
{
// Create 1000 objects of the Finalizer class
    for(int i = 1; i <= 1000; i++)
    {
// Do not store reference to the new object
  new Finalizer(i);
   }
// Invoking the garbage collector.
   System.gc();
 }
}
```