



Garbage Collection in Java



Introduction:

1. In old languages like C++ programmer is responsible for both creation and destruction of objects.
2. Usually programmer is taking very much care while creating object and neglect destruction of useless objects.
3. Due to his negligence at certain point of time for creation of new object sufficient memory may not be available and entire application may be crashed due to memory problems.
4. But in java programmer is responsible only for creation of new object and his not responsible for destruction of objects.



Introduction:

5. Sun people provided one assistant which is always running in the background for destruction at useless objects.
6. Due to this assistant the chance of failing java program is very rare because of memory problems.
7. This assistant is nothing but garbage collector. Hence the main objective of GC is to destroy useless objects.




Garbage collection

1. **Garbage collection** in Java is the process of automatically freeing heap memory by deleting unused objects that are no longer accessible in the program.
2. In other simple words, the process of automatic reclamation of runtime unused memory is known as garbage collection.
3. The program that performs garbage collection is called a garbage collector or simply a collector in java.
4. It is a part of the Java platform and is one of the major features of the Java Programming language.



Garbage collection

5. Java garbage collector runs in the background in a low-priority thread and automatically cleans up heap memory by destroying unused objects.
 6. However, before destroying unused objects, it makes sure that the running program in its current state will never use them again.
 7. 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.

What should be done with a reference variable that is not pointing to any object? Such situations can happen in the program.

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

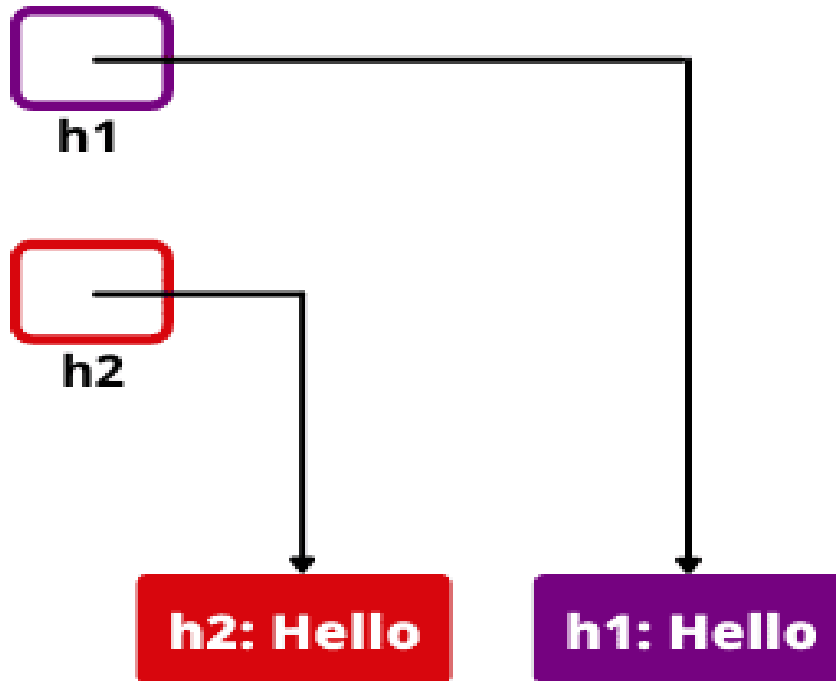
```
Hello h2 = new Hello();
```

```
h1 = h2;
```

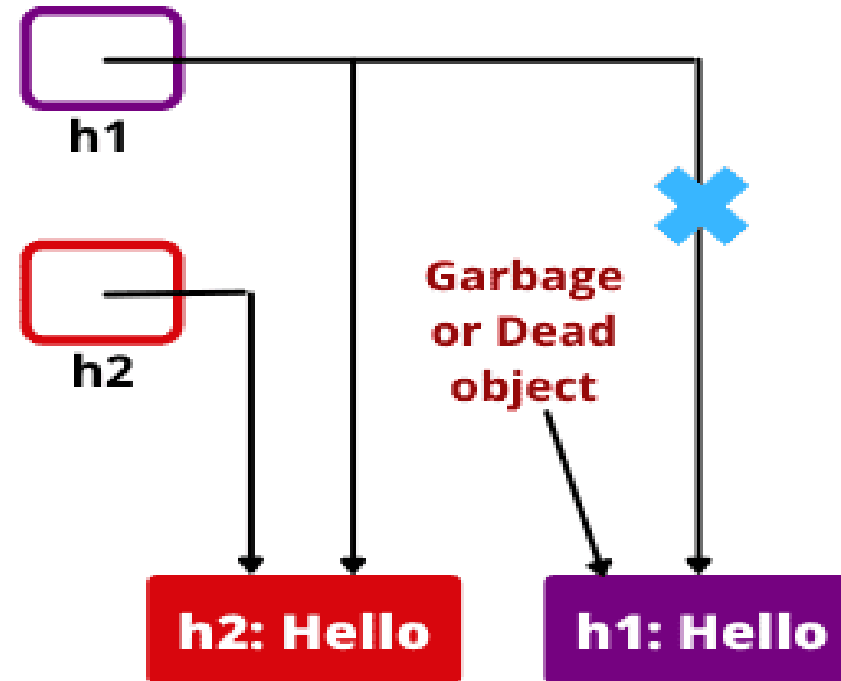
Dead Object or Garbage in Java

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

Before:



After: Object type assignment `h1 = h2`



Dead Object or Garbage in Java

1. Since garbage occupies memory space, therefore, the Java runtime system (JVM) detects garbage and automatically reclaims the memory space it occupies.
2. This process is called **garbage** collection in java. The garbage collector has the responsibility to keep track of which objects are “**garbage**.”
3. In **C/C++** programming languages, the programmer has the responsibility for both creation and destroying of objects by using methods provided by C and C++ languages.

Dead Object or Garbage in Java

4. Generally, programmer ignores or forgets destroying of unused objects. Due to this carelessness, after a certain time, sufficient memory space is not available for creating new objects causes **OutOfMemoryErrors**.
5. But in Java language, the programmer has no responsibility to keep the track of all those unused objects which are no longer in use. The garbage collector has the responsibility to destroy these unused objects to clean up the memory space.
6. The best example of garbage collector is Daemon thread that is always running in the background.
7. An object that can be used in the future by the running program is called live object in java or a reachable object.

Ways for Invoking Garbage Collector (GC)

1. When the unused object becomes eligible for garbage collection, garbage collector does not destroy them immediately.
2. **JVM** runs garbage collector whenever it runs low in memory.
3. It tries its best to clean up the memory of all unused objects before it throws a **java.lang.OutOfMemoryError** error.
4. Therefore, we can only request **JVM** to run garbage collector. But it has free to ignore the request.

Ways for Invoking Garbage Collector (GC)

There are two methods for requesting JVM to run garbage collector. They are as follows:

1. **`Runtime.getRuntime().gc()`**
2. **`System.gc()`**

Ways for Invoking Garbage Collector (GC)

1. **Runtime.getRuntime().gc()** method: Runtime class permits the program to interface with the JVM in which the program is running. By using its **gc()** method, we can request JVM to run Garbage Collector.

// Get the Runtime object.

```
Runtime rt = Runtime.getRuntime();
```

```
rt.gc();    // Call the garbage collector.
```

// Or Combine the above two statements into one

```
Runtime.getRuntime().gc();
```

Ways for Invoking Garbage Collector (GC)

2. Using `System.gc()` method:

System class contains a convenience method named `gc()` for requesting JVM to run Garbage Collector. It is a static method that is equivalent to executing the **`Runtime.getRuntime().gc()`** statement.

We can also use the following code to run the garbage collector:

```
// Invoke the garbage collector
```

```
System.gc();
```

Which of the following are valid ways for requesting jvm to run GC ?

1. `System.gc();`
2. `Runtime.gc();`
3. `(new Runtime).gc();`
4. `Runtime.getRuntime().gc();`

1. **Note:** `gc()` method present in **System** class is **static**, where as it is instance method in **Runtime** class.
2. **Note:** Over **Runtime** class `gc()` method , **System** class `gc()` method is recommended to use.
3. **Note:** in java it is not possible to find size of an **object** and **address** of an object.



Object Finalization in Java

Object Finalization in Java

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.

Object Finalization in Java

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.

In this case, we need to write a finalizer method for an object that performs such tasks as closing files, deleting temporary files, terminating network connection, etc.

Finalizer Method in Java

A **finalizer** is an instance method that is provided by **Object** class, which is the top of the Java platform's class hierarchy and a superclass of all classes in java.

To add a finalizer to a class, we simply declare a **finalize()** method. Inside the **finalize()** method, we will write the finalization code for those actions that must be performed before an object is destroyed by the garbage collector.

The syntax to declare **finalize()** method of **Object** class is as follows:

```
protected void finalize() throws Throwable
{
    // finalization code here.
}
```

Important Points about Finalizer

1. The main objective of finalization is to call **finalize()** method before the garbage collection frees up the memory occupied by an object.
2. The **finalize()** method of Object class can be called by garbage collector only once before object is garbage collected.
3. An object's finalize() method cannot be called while it is reachable.
4. Since all Java classes extend the **Object** class, therefore, the finalize() method can be called on all Java objects.
5. Any class can override and implement its own version of the **finalize()** method to perform finalization necessary for objects of that type.
6. The **finalize()** method does not invoke when an object goes out of scope.

Different Cases:

Case 1:

Just before destroying any object GC calls `finalize()` method on the object which is eligible for GC then the corresponding class `finalize()` method will be executed.

For Example if String object is eligible for GC then String class `finalize()` method is executed but not class `finalize()` method.

Case 2:

We can call **`finalize()`** method explicitly then it will be executed just like a normal method call and object won't be destroyed. But before destroying any object GC always calls **`finalize()`** method.

Different Cases:

Case 3:

1. **finalize()** method can be call either by the programmer or by the GC .
2. If the programmer calls explicitly **finalize()** method and while executing the **finalize()** method if an exception raised and uncaught then the program will be terminated abnormally.
3. If **GC** calls **finalize()** method and while executing the **finalize()** method if an exception raised and uncaught then JVM simply ignores that exception and the program will be terminated normally.



Different Cases:

Case 4:

On any object GC calls finalize() method only once.

Note:

The behavior of the GC is vendor dependent and varied from **JVM** to **JVM** hence we can't expect exact answer for the following.

1. What is the algorithm followed by **GC** ?
2. Exactly at what time **JVM** runs **GC** ?
3. In which order **GC** identifies the eligible objects ?
4. In which order **GC** destroys the object etc ?
5. Whether **GC** destroys all eligible objects or not ?

When ever the program runs with low memory then the **JVM** runs **GC**, but we can't expect exactly at what time.

Most of the **GC's** followed **mark & sweep** algorithm , but it doesn't mean every **GC** follows the same algorithm.

Which of the following is true?

1. While executing finalize() method JVM ignores every exception.
2. While executing finalize() method JVM ignores only uncaught exception.

1. **invalid**

2. **valid**

Memory leaks:

1. An object which is not using in our application and it is not eligible for GC such type of objects are called "**memory leaks**".
2. In the case of memory leaks **GC** also can't do anything the application will be crashed due to memory problems.
3. In our program if memory leaks present then certain point we will get **OutOfMemoryException**. Hence if an object is no longer required then it's highly recommended to make that object eligible for GC.
4. By using monitoring tools we can identify **memory leaks**.

These are monitoring tools (or memory management tools)

Example:

1. HPJ meter
2. HP ovo
3. IBM Tivoli
4. J Probe
5. Patrol and etc