**How can you force garbage collection?**

You cannot force Garbage Collection, but could request it by calling System.gc(). Though you manually call this command from your program, the JVM does not guarantee that GC will be started immediately.

**How can you minimize the need of garbage collection and make the memory use more effective?**

Use object pooling and weak object references. We need to ensure that all objects that are no longer required in the program are cleared off using finalize() blocks in your code.

**Explain garbage collection ?**

Garbage collection is an important part of Java's security strategy. Garbage collection is also called automatic memory management as JVM automatically removes the unused variables/objects from the memory. The name "garbage collection" implies that objects that are no longer needed or used by the program are "garbage" and can be thrown away to create free space for the programs that are currently running. A more accurate and up-to-date term might be "memory recycling." When an object is no longer referenced by the program, the heap space it occupies must be recycled so that the space is available for subsequent new objects. The garbage collector must determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects. This way, unused memory space is reclaimed and made available for the program to use.

**Does garbage collection guarantee that a program will not run out of memory?**

Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection as well. Hence, the Garbage Collection mechanism is a "On Best Effort Basis" system where the GC tries to clean up memory as much as possible to ensure that the system does not run out of memory but it does not guarantee the same.

**Can an object's finalize() method be invoked while it is reachable?**

An object's finalize() method cannot be invoked by the garbage collector while the object is still reachable. However, an object's finalize() method may be invoked by other objects.

**What is the purpose of finalization?**

The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected. We usually nullify the object references of large objects like collections, maps etc in the finalize block.

**How many times may an object's finalize() method be invoked by the garbage collector?**

An object's finalize() method may only be invoked once by the garbage collector.

**Can an object be garbage collected while it is still reachable?**

A reachable object cannot be garbage collected. Only unreachable objects may be garbage collected. The system will not do it and you cannot force it either.

**If an object is garbage collected, can it become reachable again?**

Once an object is garbage collected, it ceases to exist. i.e., it is dead or deleted. It can no longer become reachable again.

**What is the purpose of garbage collection?**

The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources may be reclaimed and reused.

**Can an unreachable object become reachable again?**

An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects.

**When is an object subject to garbage collection?**

An object is subject to garbage collection when it becomes unreachable to the program in which it is used. i.e., no other object or code in the system is going to access this current object

**Does System.gc() and Runtime.gc() guarantee garbage collection**

No.

**Do we need memory management code in our application?**

No. Memory Management is a complicated activity and that is exactly why the creators of the Java language did it themselves so that programmers like us do not have to go through the pain of handling memory management.

**What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?**

This Error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector. Note: Its an Error (extends java.lang.Error) not Exception.

Two important types of OutOfMemoryError are often encountered

1. java.lang.OutOfMemoryError: Java heap space - The quick solution is to add these flags to JVM command line when Java runtime is started as follows:

-Xms1024m -Xmx1024m

2. java.lang.OutOfMemoryError: PermGen space - The solution is to add these flags to JVM command line when Java runtime is started as follows:

-XX:+CMSClassUnloadingEnabled-XX:+CMSPermGenSweepingEnabled

Increasing the Start/Max Heap size or changing Garbage Collection options may not always be a long term solution for your Out Of Memory Error problem. Best approach is to understand the memory needs of your program and ensure it uses memory wisely and does not have leaks.

**What is Garbage Collection in Java?**

Garbage Collection is an automatic memory management feature.

The process of destroying unreferenced objects is called Garbage Collection.

Once object is unreferenced it is considered as unused object, hence JVM automatically destroys that object.

In java developers responsibility is only to creating objects and unreferencing those objects after usage.

**How JVM can destroy unreferenced object?**

JVM internally uses a daemon thread called "garbage collector" to destroy all unreferenced objects.

A daemon thread is a service thread. Garbage Collector thread is called daemon thread because it provides services to JVM to destroy unreferenced objects.

This thread is low priority thread. Since it is a low priority thread we can not guarantee this execution.

**So can you guarantee objects destruction?**

No, we can not guarantee objects destruction even though it is unreferenced, because we can not guarantee garbage collector execution.

So, we can confirm whether object is eligible for garbage collection or not.

**Can we force garbage collector?**

No, we can not force garbage collector to destroy objects , but we can request it.

**How can we request JVM to start garbage collection process?**

We have a method called gc() in system class as static method and also in Runtime class as non static method to request JVM to start garbage collector execution.

System.gc();

Runtime.getRuntime().gc();

**What is the algorithm JVM internally uses for destroying objects?**

"mark and swap" is the algorithm JVM internally uses.

**Which part of the memory is involved in Garbage Collection?**

Heap.

**What is responsibility of Garbage Collector?**

Garbage Collector frees the memory occupied by the unreachable objects during the java program by deleting these unreachable objects.

It ensures that the available memory will be used efficiently, but does not guarantee that there will be sufficient memory for the program to run.

**When does an object become eligible for garbage collection?**

An object becomes eligible for garbage collection when no live thread can access it.

**What is purpose of overriding finalize() method?**

The finalize() method should be overridden for an object to include the clean up code or to dispose of the system resources that should to be done before the object is garbage collected.

**How many times does the garbage collector calls the finalize() method for an object?**

Only once.

**What happens if an uncaught exception is thrown from during the execution of finalize() method of an object?**

The exception will be ignored and the garbage collection (finalization) of that object terminates

**What are the different ways to call garbage collector?**

System.gc();

Runtime.getRuntime().gc();

**How to enable /disable call of finalize() method of exit of application?**

Runtime.getRuntime().runFinalizersOnExit(boolean value). passing the boolean value true and false will enable or disable the finalize() call.