Types of Java Garbage Collectors

**Mark and Sweep Algorithm**

Any garbage collection algorithm must perform 2 basic operations. One, it should be able to detect all the unreachable objects and secondly, it must reclaim the heap space used by the garbage objects and make the space available again to the program.  
The above operations are performed by Mark and Sweep Algorithm in two phases:  
1) Mark phase  
2) Sweep phase

**Advantages of Mark and Sweep Algorithm**

* It handles the case with cyclic references, even in case of a cycle, this algorithm never ends up in an infinite loop.
* There are no additional overheads incurred during the execution of the algorithm.

**Disadvantages of Mark and Sweep Algorithm**

* The main disadvantage of the mark-and-sweep approach is the fact that that normal program execution is suspended while the garbage collection algorithm runs.
* Other disadvantage is that, after the Mark and Sweep Algorithm is run several times on a program, reachable objects end up being separated by many, small unused memory regions.

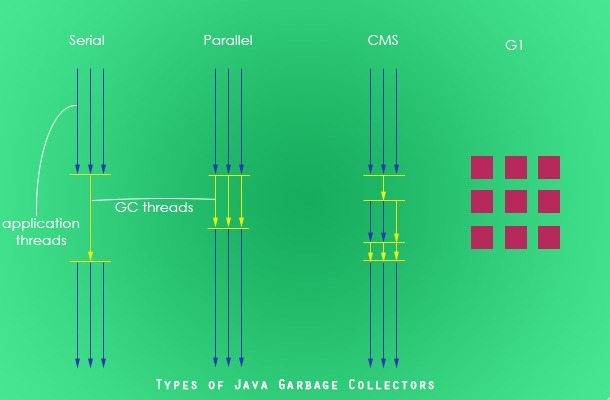
In this tutorial we will go through the various type of Java garbage collectors available. Garbage collection is an automatic process in Java which relieves the programmer of object memory allocation and de-allocation chores. This is the third part in the garbage collection tutorial series. In the previous part 2 we saw about [how garbage collection works in Java](http://javapapers.com/java/how-java-garbage-collection-works/), it is an interesting read and I recommend you to go through it. In the part 1 [introduction to Java garbage collection](http://javapapers.com/java/java-garbage-collection-introduction/), we saw about the JVM architecture, heap memory model and surrounding Java terminologies.



Java has **four types of garbage collectors**,

1. [Serial Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#serial-garbage-collector)
2. [Parallel Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#parallel-garbage-collector) /// default garbage collector of the JVM.
3. [CMS Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#cms-garbage-collector) Concurrent Mark Sweep
4. [G1 Garbage Collector](http://javapapers.com/java/types-of-java-garbage-collectors/#g1-garbage-collector)

Each of these four types has its own advantages and disadvantages. Most importantly, we the programmers can choose the type of garbage collector to be used by the JVM. We can choose them by passing the choice as JVM argument. Each of these types differ largely and can provide completely different application performance. It is critical to understand each of these types of garbage collectors and use it rightly based on the application.



1. Serial Garbage Collector

Serial garbage collector works by holding all the application threads. It is designed for the single-threaded environments. It uses just a single thread for garbage collection. The way it works by freezing all the application threads while doing garbage collection may not be suitable for a server environment. It is best suited for simple command-line programs.

Turn on the -XX:+UseSerialGC JVM argument to use the serial garbage collector.

2. Parallel Garbage Collector

Parallel garbage collector is also called as throughput collector. It is the default garbage collector of the JVM. Unlike serial garbage collector, this uses multiple threads for garbage collection. Similar to serial garbage collector this also freezes all the application threads while performing garbage collection.

3. CMS Garbage Collector

Concurrent Mark Sweep (CMS) garbage collector uses multiple threads to scan the heap memory to mark instances for eviction and then sweep the marked instances. CMS garbage collector holds all the application threads in the following two scenarios only,

1. while marking the referenced objects in the tenured generation space.
2. if there is a change in heap memory in parallel while doing the garbage collection.

In comparison with parallel garbage collector, CMS collector uses more CPU to ensure better application throughput. If we can allocate more CPU for better performance then CMS garbage collector is the preferred choice over the parallel collector.

Turn on the XX:+USeParNewGC JVM argument to use the CMS garbage collector.

4. G1 Garbage Collector

G1 garbage collector is used for large heap memory areas. It separates the heap memory into regions and does collection within them in parallel. G1 also does compacts the free heap space on the go just after reclaiming the memory. But CMS garbage collector compacts the memory on stop the world (STW) situations. G1 collector prioritizes the region based on most garbage first.

Turn on the –XX:+UseG1GC JVM argument to use the G1 garbage collector.

**Java 8 Improvement**

Turn on the -XX:+UseStringDeduplication JVM argument while using G1 garbage collector. This optimizes the heap memory by removing duplicate String values to a single char[] array. This option is introduced in [Java 8](http://javapapers.com/java/java-8-features/) u 20.

Given all the above four types of Java garbage collectors, which one to use depends on the application scenario, hardware available and the throughput requirements.

Garbage Collection JVM Options

Following are the key JVM options that are related to Java garbage collection.

**Type of Garbage Collector to run**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -XX:+UseSerialGC | Serial Garbage Collector |
| -XX:+UseParallelGC | Parallel Garbage Collector |
| -XX:+UseConcMarkSweepGC | CMS Garbage Collector |
| -XX:ParallelCMSThreads= | CMS Collector – number of threads to use |
| -XX:+UseG1GC | G1 Gargbage Collector |

**GC Optimization Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -Xms | Initial heap memory size |
| -Xmx | Maximum heap memory size |
| -Xmn | Size of Young Generation |
| -XX:PermSize | Initial Permanent Generation size |
| -XX:MaxPermSize | Maximum Permanent Generation size |

**Example Usage of JVM GC Options**

java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseSerialGC -jar java-application.jar

In the next part of this Java garbage collection tutorial series, we will see about how to monitor and analyze the garbage collection with an example Java application.

**Java Heap Space vs Stack – Memory Allocation in Java**

**Java Heap Space**

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space.

Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

**Java Stack Memory**

Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method.

Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.

As soon as method ends, the block becomes unused and become available for next method.  
Stack memory size is very less compared to Heap memory.

**Heap and Stack Memory in Java Program**

Let’s understand the Heap and Stack memory usage with a simple program.

package com.journaldev.test;

public class Memory {

public static void main(String[] args) { // Line 1

int i=1; // Line 2

Object obj = new Object(); // Line 3

Memory mem = new Memory(); // Line 4

mem.foo(obj); // Line 5

} // Line 9

private void foo(Object param) { // Line 6

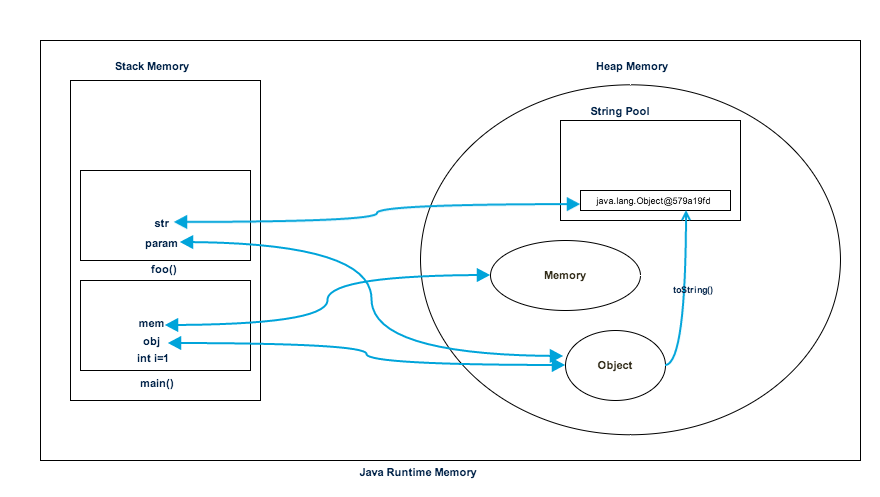
String str = param.toString(); //// Line 7

System.out.println(str);

} // Line 8

}

Below image shows the Stack and Heap memory with reference to above program and how they are being used to store primitive, Objects and reference variables.



**When are class garbage collected, when are classes unloaded.**  
The only way that a Class can be unloaded is if the Classloader used is garbage collected.  **How to ensure that instance is never garbage collected.**  
A singleton kind of pattern.   
There's a static reference to a singleton, so it won't be eligible for garbage collection until the   
classloader is eligible for garbage collection.