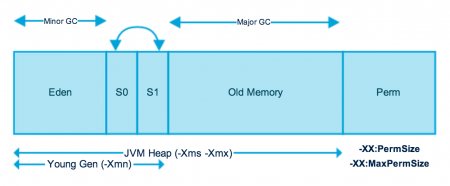
**Java (JVM) Memory Model – Memory Management in Java**

JUNE 11, 2016 BY [PANKAJ](http://www.journaldev.com/author/pankaj) [81 COMMENTS](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comments)

Understanding **JVM Memory Model**, **Java Memory Management** are very important if you want to understand the working of **Java Garbage Collection**. Today we will look into memory management in java, different parts of JVM memory and how to monitor and perform garbage collection tuning.

**Java (JVM) Memory Model**

[](http://cdn.journaldev.com/wp-content/uploads/2014/05/Java-Memory-Model.png)

As you can see in the above image, JVM memory is divided into separate parts. At broad level, JVM Heap memory is physically divided into two parts – **Young Generation** and **Old Generation**.

**Memory Management in Java – Young Generation**

Young generation is the place where all the new objects are created. When young generation is filled, garbage collection is performed. This garbage collection is called **Minor GC**. Young Generation is divided into three parts – **Eden Memory** and two **Survivor Memory** spaces.

Important Points about Young Generation Spaces:

* Most of the newly created objects are located in the Eden memory space.
* When Eden space is filled with objects, Minor GC is performed and all the survivor objects are moved to one of the survivor spaces.
* Minor GC also checks the survivor objects and move them to the other survivor space. So at a time, one of the survivor space is always empty.
* Objects that are survived after many cycles of GC, are moved to the Old generation memory space. Usually it’s done by setting a threshold for the age of the young generation objects before they become eligible to promote to Old generation.

**Memory Management in Java – Old Generation**

Old Generation memory contains the objects that are long lived and survived after many rounds of Minor GC. Usually garbage collection is performed in Old Generation memory when it’s full. Old Generation Garbage Collection is called **Major GC** and usually takes longer time.

**Stop the World Event**

All the Garbage Collections are “Stop the World” events because all application threads are stopped until the operation completes.

Since Young generation keeps short-lived objects, Minor GC is very fast and the application doesn’t get affected by this.

However Major GC takes longer time because it checks all the live objects. Major GC should be minimized because it will make your application unresponsive for the garbage collection duration. So if you have a responsive application and there are a lot of Major Garbage Collection happening, you will notice timeout errors.

The duration taken by garbage collector depends on the strategy used for garbage collection. That’s why it’s necessary to monitor and tune the garbage collector to avoid timeouts in the highly responsive applications.

**Java Memory Model – Permanent Generation**

Permanent Generation or “Perm Gen” contains the application metadata required by the JVM to describe the classes and methods used in the application. Note that Perm Gen is not part of Java Heap memory.

Perm Gen is populated by JVM at runtime based on the classes used by the application. Perm Gen also contains Java SE library classes and methods. Perm Gen objects are garbage collected in a full garbage collection.

**Java Memory Model – Method Area**

Method Area is part of space in the Perm Gen and used to store class structure (runtime constants and static variables) and code for methods and constructors.

**Java Memory Model – Memory Pool**

Memory Pools are created by JVM memory managers to create a pool of immutable objects, if implementation supports it. String Pool is a good example of this kind of memory pool. Memory Pool can belong to Heap or Perm Gen, depending on the JVM memory manager implementation.

**Java Memory Model – Runtime Constant Pool**

Runtime constant pool is per-class runtime representation of constant pool in a class. It contains class runtime constants and static methods. Runtime constant pool is the part of method area.

**Java Memory Model – 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. You should read [Difference between Stack and Heap Memory](http://www.journaldev.com/4098/java-heap-space-vs-stack-memory).

**Memory Management in Java – Java Heap Memory Switches**

Java provides a lot of memory switches that we can use to set the memory sizes and their ratios. Some of the commonly used memory switches are:

|  |  |
| --- | --- |
| VM SWITCH | VM SWITCH DESCRIPTION |
| -Xms | For setting the initial heap size when JVM starts |
| -Xmx | For setting the maximum heap size. |
| -Xmn | For setting the size of the Young Generation, rest of the space goes for Old Generation. |
| -XX:PermGen | For setting the initial size of the Permanent Generation memory |
| -XX:MaxPermGen | For setting the maximum size of Perm Gen |
| -XX:SurvivorRatio | For providing ratio of Eden space and Survivor Space, for example if Young Generation size is 10m and VM switch is -XX:SurvivorRatio=2 then 5m will be reserved for Eden Space and 2.5m each for both the Survivor spaces. The default value is 8. |
| -XX:NewRatio | For providing ratio of old/new generation sizes. The default value is 2. |

Most of the times, above options are sufficient, but if you want to check out other options too then please check [JVM Options Official Page](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html).

**Memory Management in Java – Java Garbage Collection**

Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space to be allocated to objects created in the future processing. One of the best feature of java programming language is the **automatic garbage collection**, unlike other programming languages such as C where memory allocation and deallocation is a manual process.

**Garbage Collector** is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program. All these unreferenced objects are deleted and space is reclaimed for allocation to other objects.

One of the basic way of garbage collection involves three steps:

1. **Marking**: This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
2. **Normal Deletion**: Garbage Collector removes the unused objects and reclaim the free space to be allocated to other objects.
3. **Deletion with Compacting**: For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.

There are two problems with simple mark and delete approach.

1. First one is that it’s not efficient because most of the newly created objects will become unused
2. Secondly objects that are in-use for multiple garbage collection cycle are most likely to be in-use for future cycles too.

The above shortcomings with the simple approach is the reason that **Java Garbage Collection is Generational** and we have **Young Generation** and **Old Generation** spaces in the heap memory. I have already explained above how objects are scanned and moved from one generational space to another based on the Minor GC and Major GC.

**Memory Management in Java – Java Garbage Collection Types**

There are five types of garbage collection types that we can use in our applications. We just need to use JVM switch to enable the garbage collection strategy for the application. Let’s look at each of them one by one.

1. **Serial GC (-XX:+UseSerialGC)**: Serial GC uses the simple **mark-sweep-compact** approach for young and old generations garbage collection i.e Minor and Major GC.

Serial GC is useful in client-machines such as our simple stand alone applications and machines with smaller CPU. It is good for small applications with low memory footprint.

1. **Parallel GC (-XX:+UseParallelGC)**: Parallel GC is same as Serial GC except that is spawns N threads for young generation garbage collection where N is the number of CPU cores in the system. We can control the number of threads using -XX:ParallelGCThreads=n JVM option.

Parallel Garbage Collector is also called throughput collector because it uses multiple CPUs to speed up the GC performance. Parallel GC uses single thread for Old Generation garbage collection.

1. **Parallel Old GC (-XX:+UseParallelOldGC)**: This is same as Parallel GC except that it uses multiple threads for both Young Generation and Old Generation garbage collection.
2. **Concurrent Mark Sweep (CMS) Collector (-XX:+UseConcMarkSweepGC)**: CMS Collector is also referred as concurrent low pause collector. It does the garbage collection for Old generation. CMS collector tries to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads.

CMS collector on young generation uses the same algorithm as that of the parallel collector. This garbage collector is suitable for responsive applications where we can’t afford longer pause times. We can limit the number of threads in CMS collector using -XX:ParallelCMSThreads=nJVM option.

1. **G1 Garbage Collector (-XX:+UseG1GC)**: The Garbage First or G1 garbage collector is available from Java 7 and it’s long term goal is to replace the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector.

Garbage First Collector doesn’t work like other collectors and there is no concept of Young and Old generation space. It divides the heap space into multiple equal-sized heap regions. When a garbage collection is invoked, it first collects the region with lesser live data, hence “Garbage First”. You can find more details about it at [Garbage-First Collector Oracle Documentation](http://docs.oracle.com/javase/7/docs/technotes/guides/vm/G1.html).

**Memory Management in Java – Java Garbage Collection Monitoring**

We can use Java command line as well as UI tools for monitoring garbage collection activities of an application. For my example, I am using one of the demo application provided by Java SE downloads.

If you want to use the same application, go to [Java SE Downloads](http://www.oracle.com/technetwork/java/javase/downloads/index.html) page and download **JDK 7 and JavaFX Demos and Samples**. The sample application I am using is **Java2Demo.jar** and it’s present in jdk1.7.0\_55/demo/jfc/Java2Ddirectory. However this is an optional step and you can run the GC monitoring commands for any java application.

Command used by me to start the demo application is:

pankaj@Pankaj:~/Downloads/jdk1.7.0\_55/demo/jfc/Java2D$ java -Xmx120m -Xms30m -Xmn10m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseSerialGC -jar Java2Demo.jar

**jsat**

We can use jstat command line tool to monitor the JVM memory and garbage collection activities. It ships with standard JDK, so you don’t need to do anything else to get it.

For executing jstat you need to know the process id of the application, you can get it easily using ps -eaf | grep java command.

pankaj@Pankaj:~$ ps -eaf | grep Java2Demo.jar

501 9582 11579 0 9:48PM ttys000 0:21.66 /usr/bin/java -Xmx120m -Xms30m -Xmn10m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseG1GC -jar Java2Demo.jar

501 14073 14045 0 9:48PM ttys002 0:00.00 grep Java2Demo.jar

So the process id for my java application is 9582. Now we can run **jstat**command as shown below.

pankaj@Pankaj:~$ jstat -gc 9582 1000

S0C S1C S0U S1U EC EU OC OU PC PU YGC YGCT FGC FGCT GCT

1024.0 1024.0 0.0 0.0 8192.0 7933.3 42108.0 23401.3 20480.0 19990.9 157 0.274 40 1.381 1.654

1024.0 1024.0 0.0 0.0 8192.0 8026.5 42108.0 23401.3 20480.0 19990.9 157 0.274 40 1.381 1.654

1024.0 1024.0 0.0 0.0 8192.0 8030.0 42108.0 23401.3 20480.0 19990.9 157 0.274 40 1.381 1.654

1024.0 1024.0 0.0 0.0 8192.0 8122.2 42108.0 23401.3 20480.0 19990.9 157 0.274 40 1.381 1.654

1024.0 1024.0 0.0 0.0 8192.0 8171.2 42108.0 23401.3 20480.0 19990.9 157 0.274 40 1.381 1.654

1024.0 1024.0 48.7 0.0 8192.0 106.7 42108.0 23401.3 20480.0 19990.9 158 0.275 40 1.381 1.656

1024.0 1024.0 48.7 0.0 8192.0 145.8 42108.0 23401.3 20480.0 19990.9 158 0.275 40 1.381 1.656

The last argument for jstat is the time interval between each output, so it will print memory and garbage collection data every 1 second.

Let’s go through each of the columns one by one.

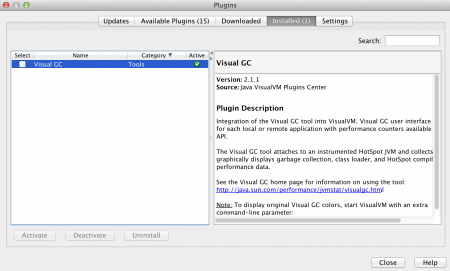
* **S0C and S1C**: This column shows the current size of the Survivor0 and Survivor1 areas in KB.
* **S0U and S1U**: This column shows the current usage of the Survivor0 and Survivor1 areas in KB. Notice that one of the survivor areas are empty all the time.
* **EC and EU**: These columns show the current size and usage of Eden space in KB. Note that EU size is increasing and as soon as it crosses the EC, Minor GC is called and EU size is decreased.
* **OC and OU**: These columns show the current size and current usage of Old generation in KB.
* **PC and PU**: These columns show the current size and current usage of Perm Gen in KB.
* **YGC and YGCT**: YGC column displays the number of GC event occurred in young generation. YGCT column displays the accumulated time for GC operations for Young generation. Notice that both of them are increased in the same row where EU value is dropped because of minor GC.
* **FGC and FGCT**: FGC column displays the number of Full GC event occurred. FGCT column displays the accumulated time for Full GC operations. Notice that Full GC time is too high when compared to young generation GC timings.
* **GCT**: This column displays the total accumulated time for GC operations. Notice that it’s sum of YGCT and FGCT column values.

The advantage of **jstat** is that it can be executed in remote servers too where we don’t have GUI. Notice that sum of S0C, S1C and EC is 10m as specified through -Xmn10m JVM option.

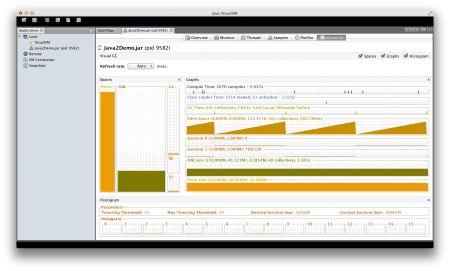
**Java VisualVM with Visual GC**

If you want to see memory and GC operations in GUI, then you can use jvisualvm tool. Java VisualVM is also part of JDK, so you don’t need to download it separately.

Just run jvisualvm command in the terminal to launch the Java VisualVM application. Once launched, you need to install **Visual GC** plugin from Tools -< Plugins option, as shown in below image.

[](http://cdn.journaldev.com/wp-content/uploads/2014/05/VisualVM-Visual-GC-Plugin.png)

After installing **Visual GC**, just open the application from the left side column and head over to **Visual GC** section. You will get an image of JVM memory and garbage collection details as shown in below image.

[](http://cdn.journaldev.com/wp-content/uploads/2014/05/Serial-GC-VisualGC.png)

**Java Garbage Collection Tuning**

**Java Garbage Collection Tuning** should be the last option you should use for increasing the throughput of your application and only when you see drop in performance because of longer GC timings causing application timeout.

If you see java.lang.OutOfMemoryError: PermGen space errors in logs, then try to monitor and increase the Perm Gen memory space using -XX:PermGen and -XX:MaxPermGen JVM options. You might also try using -XX:+CMSClassUnloadingEnabled and check how it’s performing with CMS Garbage collector.

If you are see a lot of Full GC operations, then you should try increasing Old generation memory space.

Overall garbage collection tuning takes a lot of effort and time and there is no hard and fast rule for that. You would need to try different options and compare them to find out the best one suitable for your application.

That’s all for Java Memory Model, Memory Management in Java and Garbage Collection, I hope it helps you in understanding JVM memory and garbage collection process.

FILED UNDER: [JAVA](http://www.journaldev.com/dev/java)

**About Pankaj**

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on [**Google Plus**](https://plus.google.com/118104420597648001532/posts?rel=author), **[Facebook](https://www.facebook.com/journaldev)** or [**Twitter**](https://twitter.com/JournalDev). I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **[Youtube](https://www.youtube.com/user/JournalDev)**.

[« Java 8 Date – LocalDate, LocalDateTime, Instant](http://www.journaldev.com/2800/java-8-date-localdate-localdatetime-instant)

[Hibernate Tutorial For Beginners »](http://www.journaldev.com/2882/hibernate-tutorial-for-beginners)

**Comments**

1. [**Muhammad Rashid**](http://www.netpace.com/)**says**

[AUGUST 7, 2017 AT 11:32 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-39011)

Very Good explanation. However, I noted one point where we executed the following program:

pankaj@Pankaj:~/Downloads/jdk1.7.0\_55/demo/jfc/Java2D$ java -Xmx120m -Xms30m -Xmn10m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseSerialGC -jar Java2Demo.jar

We used the UseSerialGC as Garbage Collector Algorithm. Whereas when we executed the following command:

pankaj@Pankaj:~$ ps -eaf | grep Java2Demo.jar  
501 9582 11579 0 9:48PM ttys000 0:21.66 /usr/bin/java -Xmx120m -Xms30m -Xmn10m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseG1GC -jar Java2Demo.jar  
501 14073 14045 0 9:48PM ttys002 0:00.00 grep Java2Demo.jar

As noted above the Garbage Collector algorithm is -XX:+UseG1GC. Does the JVM suppressed the provided -XX:+UseSerialGC jvm option or some thing other force to change the algorithm?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-39011)

1. **Sanjeev says**

[AUGUST 5, 2017 AT 4:19 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38908)

Crisp and excellent coverage.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38908)

1. [**Akhil**](http://www.facebook.com/thetechiexyz)**says**

[JULY 25, 2017 AT 2:07 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38744)

Thank you so much for sharing such a valuable information. Keep posting great articles and tutorials.!!

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38744)

1. **xionghongzhi says**

[JULY 18, 2017 AT 1:23 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38675)

SO GREAT~~~~

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38675)

1. **jean says**

[JULY 16, 2017 AT 1:13 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38655)

great article! thank you, dude

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38655)

1. [**Thach Le**](http://thachleblog.com/)**says**

[JULY 9, 2017 AT 10:32 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38607)

Very helpful. Thank you very much

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38607)

1. **Vamshi Krishna P says**

[JULY 2, 2017 AT 11:41 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38539)

Very informative article, thanks a lot Pankaj. Could you please provide the link for JDK8 article where PermGen is removed and metaspace is added.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38539)

1. **Arijit Bhadra says**

[JUNE 24, 2017 AT 4:49 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38463)

Worth of reading indeed. could you please tell me since Scala is also using JVM are they have same Memory management technique?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38463)

1. **Trupti says**

[MAY 25, 2017 AT 4:06 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38242)

Very informative.Thanks.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38242)

1. **Trupti says**

[MAY 25, 2017 AT 4:05 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38241)

Very well explained article.Thanks.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38241)

1. **Mukesh says**

[MAY 23, 2017 AT 12:14 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38226)

Good read, but needs few editing.

First line:  
Understanding JVM Memory Model, Java Memory Management are very important if you want to understand the working of Java Garbage Collection

Instead it should be:  
Understanding working of Java Garbage Collection is very important if you want to understand the JVM Memory Model, Java Memory Management

The memory model depends on the garbage collector not the other way.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38226)

1. **prabhu kvn says**

[MAY 18, 2017 AT 11:50 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38196)

This is one of the fantastic and crisp article.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38196)

1. **Rakesh says**

[MAY 12, 2017 AT 9:05 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38146)

Hi Pankaj,  
It was very basic and up to the point described  
Thanks a lot.  
It has improved my knowledge base.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38146)

1. **Avirup Das says**

[MAY 4, 2017 AT 5:50 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38082)

Hi Pankaj

This article is really helpful.

Thanks  
Avirup Das

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-38082)

1. **Hua says**

[APRIL 22, 2017 AT 5:12 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37991)

This is definitely the most awesome and valuable article describing java memory management ever!!!!

Thanks

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37991)

1. **venkatesh says**

[APRIL 13, 2017 AT 10:49 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37925)

The article is too good, and all the required information related to memory maangement is provided.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37925)

1. **Senthil V says**

[APRIL 1, 2017 AT 10:27 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37836)

Nice Article

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37836)

1. **Wojciech Fornal says**

[MARCH 31, 2017 AT 11:27 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37832)

Great article, I’ll definitely come back later for more

Thanks

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37832)

1. **Jitender says**

[MARCH 15, 2017 AT 12:27 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37717)

Nice Article. Worth a read. Can you please share the new memory model used in Java 8 as Perm Gem has been removed.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37717)

* + **Ajay Kumar says**

[MARCH 16, 2017 AT 7:54 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37721)

Very much worth-full contents….  
Thank for posting nice articles.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37721)

1. **chandan says**

[MARCH 15, 2017 AT 11:39 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37716)

thanks for such a nice tutorial

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37716)

1. **sachin says**

[FEBRUARY 6, 2017 AT 9:09 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37414)

is this the model existent even on java 8 ?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37414)

1. **Yasin says**

[JANUARY 10, 2017 AT 2:49 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37237)

did you missed native memory

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37237)

1. **shweta says**

[DECEMBER 21, 2016 AT 8:23 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37092)

Amazing article

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37092)

1. **Valerii says**

[DECEMBER 8, 2016 AT 7:13 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36988)

For the article of 2016 it would be nice to see any comments regarding removed PermGen/added Metaspace in Java8, for example

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36988)

1. **Desly Peter says**

[DECEMBER 8, 2016 AT 1:04 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36986)

This is really Helpful

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36986)

1. **Vinoo says**

[DECEMBER 4, 2016 AT 9:54 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36973)

Amazing tutorials Pankaj..Keep up the good work..Have been a fan of all your articles.Crisply written as well

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36973)

1. **James Kelly says**

[OCTOBER 27, 2016 AT 4:38 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36708)

Excellent article, most useful, thank-you very much   
I agree that this should stay as is and that a new revised article for Java 8 should be produced.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36708)

1. **Rohit Shekhar says**

[OCTOBER 6, 2016 AT 7:44 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36513)

You wrote :- Objects that are survived after many cycles of GC, are moved to the Old generation memory space.

Please clarify, Minor GC is responsible for this ? or Major GC?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36513)

1. **Rohit Shekhar says**

[OCTOBER 6, 2016 AT 5:52 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36512)

When we talk about stack and heap memory, In that concern, JVM memory is dividend into 3 parts, Young, old and Perm(mentioned in this page). And Heap Memory = Young Gen + Old Generation, Could you please confirm where stack memory remains in memory? I understand its in Perm Generation.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36512)

1. **rahul chouksey says**

[SEPTEMBER 21, 2016 AT 1:56 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36365)

good article

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36365)

1. **venkat says**

[AUGUST 28, 2016 AT 6:47 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36168)

Very nice , Great work and expatiation is extraordinary

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36168)

1. **yao says**

[AUGUST 23, 2016 AT 2:13 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36104)

O~wesome thank you so~ much!

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36104)

1. **yogesh pandey says**

[AUGUST 16, 2016 AT 11:00 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36064)

really appreciating work…great work dude..and thanks a lot..

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-36064)

1. **Paul says**

[AUGUST 5, 2016 AT 5:30 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35986)

Great article, thanks!

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35986)

1. **Leandros says**

[JULY 30, 2016 AT 2:09 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35924)

Thanks a lot. !

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35924)

1. **Anitha says**

[JULY 18, 2016 AT 12:35 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35786)

Nice article , Thank you

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35786)

1. **Sabri says**

[JULY 14, 2016 AT 3:36 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35695)

Very Nice Comprehension

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35695)

1. **SIDHANT says**

[JULY 11, 2016 AT 11:52 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35663)

AWESOME EXPLANATION

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35663)

1. **Runnerdave says**

[JULY 7, 2016 AT 7:52 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35618)

very nice article, I have been trying to understand this for months and this article just made it for me.

Thanks a lot

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35618)

1. **Deepak says**

[JULY 1, 2016 AT 11:59 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35518)

Very Nice Comprehension.. Thanks for such a complete article.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35518)

1. **Ranjan Kumar says**

[APRIL 26, 2016 AT 2:16 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34575)

Since the Perm Gen area has completely been remove, can we now change this tutorial.  
I am saying this because of my high regards for you.

You have done an excellent job by doing all what you have done.  
Also please come up with a tutorial with all new features that have been added in JDK 1.8

Thanks

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34575)

* + **AlienOnEarth says**

[MAY 19, 2016 AT 10:53 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34719)

I think author should keep the as is. This article can be be useful for versions below Java 8. Instead of rewriting this article, author should create new article with Java 8 memory model comparing with this one.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34719)

* + - [**Pankaj**](http://www.journaldev.com/)**says**

[JUNE 10, 2016 AT 11:59 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35103)

Thanks for the inputs, I will write a new one with Java 8 changes.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35103)

* + - * **ade says**

[DECEMBER 23, 2016 AT 11:35 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37104)

hi Pankaj, just read this article and got to be the best explanation of memory management I’ve seen yet!

have you got around to writing the Java 8 one yet?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-37104)

1. [**Imran Bhat**](http://mindstacksolutions.com/)**says**

[APRIL 25, 2016 AT 3:44 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34569)

Really nice article. An easy insight into the JVM memory model to begin with.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34569)

* + [**Pankaj**](http://www.journaldev.com/)**says**

[JUNE 11, 2016 AT 12:36 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35126)

Thanks Imran, appreciate your comment.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35126)

1. **Sunil Negi says**

[MARCH 25, 2016 AT 3:54 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34382)

have question around this “Perm Gen objects are garbage collected in a full garbage collection.”

Full GC = Major GC? or Full GC means complete restart of JVM?

Bcz I read somewhere Perm Gen never garbage collected

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34382)

* + **Bejond says**

[MARCH 30, 2016 AT 1:51 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34415)

Hi Sunil,

Full GC is not Major GC. Full GC cleans Young Gen, Old Gen and Perm Gen.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34415)

* + - [**Pankaj**](http://www.journaldev.com/)**says**

[JUNE 11, 2016 AT 12:36 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35127)

Thanks for pitching in and answering it for me.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35127)

1. **Sunil Negi says**

[MARCH 25, 2016 AT 2:08 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34379)

thanks! nice write up with full of information

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34379)

* + [**Pankaj**](http://www.journaldev.com/)**says**

[JUNE 11, 2016 AT 12:37 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35128)

you are welcome Sunil.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35128)

1. **JVMLover says**

[MARCH 2, 2016 AT 12:57 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34217)

Hi,

Thanks for the article.

Can anyone help me with this:  
– what’s the point of having survivor spaces ?  
– if the eden space is bigger than survivor spaces and assuming that “at a time, one of the survivor space is always empty”, how could objects be moved to the survivor space when eden is full ?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34217)

* + **Bejond says**

[MARCH 28, 2016 AT 10:30 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34405)

Hi JVMLover,

1. what’s the point of having survivor spaces ?  
Answer: There are two problems with simple mark and delete approach.  
First one is that it’s not efficient because most of the newly created objects will become unused.

So survivor spaces is aim to solve the problem. So JVM use Minor GC, which will do garbage collection when eden is full. In this way, it will avoid to collect most of the newly created objects which are unused.

After doing Minor GC, there are some objects that still in used, so these objects are moved to survivors.

Two survivors is aim to compact survivor objects on each Minor GC, to free more memory.

“if the eden space is bigger than survivor spaces and assuming that “at a time, one of the survivor space is always empty”, how could objects be moved to the survivor space when eden is full ?”  
Answer: Eden is probably bigger than survivor spaces. Survivor objects are moved to survivor space by JVM. We don’t need to know how to move them.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34405)

* + - [**Pankaj**](http://www.journaldev.com/)**says**

[JUNE 11, 2016 AT 12:37 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35129)

You have added some great information, thanks friend.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-35129)

1. **hardikn says**

[FEBRUARY 10, 2016 AT 12:42 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34059)

thanx you so much

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34059)

1. **Upendar Adepu says**

[FEBRUARY 5, 2016 AT 3:08 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34030)

Thank you. Very informative article especially for people who is not too technical.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-34030)

1. **David C. Bauer says**

[OCTOBER 29, 2015 AT 5:18 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33459)

Informative and clear write up. Thank you so much for putting this together.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33459)

1. **JS says**

[OCTOBER 26, 2015 AT 10:50 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33424)

Thank you for this scholarly yet clear information.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33424)

1. **Suhas Belekar says**

[OCTOBER 6, 2015 AT 12:19 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33311)

Thanks Pankaj! Really precise information on all the factors about GC at one place.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33311)

1. **Jason Navarro says**

[AUGUST 27, 2015 AT 2:12 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33058)

Hello Pankaj,

For a 64 bit Java 7 application experiencing high number of minor GCs, taking over two seconds, and one time over 15 seconds, what would you suggest is the issue ?

Further, noticed a pattern of a minor GC taking between 4-16 seconds, then shortly followed by a full GC taking almost about 3 seconds.

Any suggestions ?

Thank you

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33058)

1. **Ayyappan K says**

[AUGUST 26, 2015 AT 12:22 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33044)

I have few questions on GC,

1, what exactly per gen does?

2, which scenario , the memory leakage occurs?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-33044)

1. **NISHANT MEHTA says**

[AUGUST 7, 2015 AT 3:00 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32923)

Best article i found on the web which such clear explanation  
Please keep up the good work

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32923)

1. **Narendra says**

[JUNE 29, 2015 AT 11:16 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32708)

HI Pankaj,

Can you please let me clear ?

Should we use —  
-XX:PermSize  
&  
-XX:MaxPermSize

OR

-XX:PermGen  
-XX:MaxPermGen

in case of perm generation

Thanks  
Narendra

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32708)

1. **Ali says**

[MAY 25, 2015 AT 7:10 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32511)

It’s  
-XX:PermSize  
&  
-XX:MaxPermSize

not  
-XX:PermGen  
-XX:MaxPermGen

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-32511)

1. **SHRIHARI RAMACHANDRA says**

[FEBRUARY 22, 2015 AT 2:05 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31799)

Very Nice Article…

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31799)

1. **Gang says**

[JANUARY 11, 2015 AT 3:10 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31405)

Hi Pankaj,

I like your website so much and learned a lot from here. Thanks.

How can I know the garbage collection type of my JVM, or how to choose the garbage collection types on my JVM? Thanks.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31405)

1. **Kunal says**

[DECEMBER 8, 2014 AT 7:30 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31058)

Nice article.

I have one question  
When we call Runtime.getRuntime().gc() then which garbage collector gets called, is it Miner GC or Major GC

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-31058)

1. **123 says**

[DECEMBER 1, 2014 AT 4:31 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30971)

hello article is good

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30971)

1. **Srikanth Ganesan says**

[NOVEMBER 12, 2014 AT 2:37 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30717)

The Best article I found on net regarding java memory pool….thank you for sharing it

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30717)

1. **Ganesh Kar says**

[SEPTEMBER 20, 2014 AT 2:37 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30212)

Thanks Pankaj…Its really a good piece of documentation…

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30212)

1. **Pradeep Tiwari says**

[SEPTEMBER 3, 2014 AT 11:01 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30029)

This is best ever tutorial I have read on internet, Very clearly covering everything from memory allocation to Garbage Collection. Kudos.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-30029)

1. **LAFK says**

[MAY 6, 2014 AT 4:36 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28787)

That is NOT about JMM, it’s about memory management and garbage collection. Java Memory Model is about how to make memory consistent for threads and what threads interactions are allowed and what optimizations and code reorderings are fine to make memory system seem consistent to us, programmers.

See William Pugh’s pages on this, or Wikipedia (Memory Model or Consistency Model) or works by Sarita Adve or JSR-133 FAQ.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28787)

* + [**Pankaj**](http://www.journaldev.com/)**says**

[MAY 6, 2014 AT 11:15 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28795)

Yes it’s memory model from garbage collection perspective.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28795)

1. **Amishi Shah says**

[MAY 5, 2014 AT 12:36 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28779)

Hi Pankaj,

Thanks for the wonderful document.

Just one thing I was trying to understand is, how does the Survivor space(S0 and S1) gets filled up? Is there any mechanism for space getting occupied between Eden, S0 and S1 ?

Thanks,  
Amishi

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28779)

* + [**Pankaj**](http://www.journaldev.com/)**says**

[MAY 5, 2014 AT 7:29 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28782)

As explained above, Minor GC moves objects from Eden to Survivor spaces and also collects unreferenced objects from survivor spaces and move them to another one. I would suggest to run Java VisualVM and see it in action for better understanding.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28782)

1. **johnybasha says**

[MAY 4, 2014 AT 5:48 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28770)

Thanks for this useful article. Easy to understand.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28770)

1. **Peter Jerald says**

[MAY 3, 2014 AT 6:47 AM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28765)

Good write up .

Whether this above explanation about Memory model is applicable to all JVM vendors ?

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28765)

* + [**Pankaj**](http://www.journaldev.com/)**says**

[MAY 3, 2014 AT 8:40 PM](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28767)

JVM Specification defines Heap memory but it doesn’t say anything about the Young Gen/Old Gen etc. Above memory model is applicable for all Hotspot VM implementations but I am not sure about other implementations.

[Reply](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java#comment-28767)