Notes on Garbage Collection

# Permspace vs Metaspace

The main difference from a user perspective is that **Metaspace by default auto increases** its size (up to what the underlying OS provides), while PermGen always has a fixed maximum size. You can set a fixed maximum for Metaspace with JVM parameters, but you cannot make PermGen auto increase.

To a large degree it is just a change of name. Back when PermGen was introduced, there was no Java EE or dynamic class(un)loading, so once a class was loaded it was stuck in memory until the JVM shut down - thus Permanent Generation. Nowadays classes may be loaded and unloaded during the lifespan of the JVM, so Metaspace makes more sense for the area where the metadata is kept.

Both of them contain the java.lang.Class instances and both of them suffer from [ClassLoader leaks](http://java.jiderhamn.se/2011/12/11/classloader-leaks-i-how-to-find-classloader-leaks-with-eclipse-memory-analyser-mat/). Only difference is that with Metaspace default settings, it takes longer until you notice the symptoms (since it auto increases as much as it can), i.e. you just push the problem further away without solving it. OTOH I imagine the effect of running out of OS memory can be more severe than just running out of JVM PermGen, so I'm not sure it is much of an improvement.

Whether you're using a JVM with PermGen or with Metaspace, if you are doing dynamic class unloading, you should to take measures against classloader leaks, for example by using my [ClassLoader Leak Prevention library](https://github.com/mjiderhamn/classloader-leak-prevention).

Is MetaSpace by default is GC collected?

Yes, GC will run on metaspace when its getting full, it would also dynamically increase (given its allowed to) the memory allocated for metadata.

Even the PermGen is GC collected by adding the args like -XX:+CMSClassUnloadingEnabled, then what makes MetaSpace better than PermGen?

The improvement is with the dynamic expansion of the metaspace which is something permgen wasn't able to do.

MetaSpace is based on native memory, so it keeps the java objects on the disks rather than on VM?

Based on the description of metaspace, it only uses the native memory (no paging).

Based on the research by Pierre - Hugues Charbonneau ([link here](http://java.dzone.com/articles/java-8-permgen-metaspace)) its clear that the introduction of metaspace doesn't necessarily solve the OOM issue, its a bandaid to the problem at best, it attempts to dynamically resize the metaspace memory to accomodate the growing number of classes which get loaded with a possible side effect of growing uncontrollably (so long as the native memory permits it).

We can achieve the famed OOM error by setting the MaxMetaspaceSize argument to JVM and running the sample program provided.