**we're going to examine the differences between the PermGen and Metaspace memory regions** in the Java environment.

It's important to keep in mind that, starting with Java 8, the Metaspace replaces the PermGen – bringing some substantial changes.

**2. PermGen**

**PermGen (Permanent Generation) is a special heap space separated from the main memory heap**.

The JVM keeps track of loaded class metadata in the PermGen. Additionally, the JVM stores all the static content in this memory section. This includes all the static methods, primitive variables, and references to the static objects.

Furthermore, **it contains data about bytecode, names, and JIT information**. Before Java 7, the String Pool was also part of this memory. The disadvantages of the fixed pool size are listed in our [write-up](https://www.baeldung.com/java-string-pool).

The default maximum memory size for 32-bit JVM is 64 MB and 82 MB for the 64-bit version.

However, we can change the default size with the JVM options:

* *-XX:PermSize=[size]* is the initial or minimum size of the PermGen space
* *-XX:MaxPermSize=[size]* is the maximum size

Most importantly, **Oracle completely removed this memory space in the JDK 8 release.** Therefore, if we use these tuning flags in Java 8 and newer versions, we'll get the following warnings:

>> java -XX:PermSize=100m -XX:MaxPermSize=200m -version

OpenJDK 64-Bit Server VM warning: Ignoring option PermSize; support was removed **in** 8.0

OpenJDK 64-Bit Server VM warning: Ignoring option MaxPermSize; support was removed **in** 8.0

...

**With its limited memory size, PermGen is involved in generating the famous *OutOfMemoryError***. Simply put, the [class loaders](https://www.baeldung.com/java-classloaders) weren't garbage collected properly and, as a result, generated a memory leak.

Therefore, we receive a [memory space error](https://www.baeldung.com/java-gc-overhead-limit-exceeded); this happens mostly in the development environment while creating new class loaders.

**3. Metaspace**

Simply put, Metaspace is a new memory space – starting from the Java 8 version; **it has replaced the older PermGen memory space**. The most significant difference is how it handles memory allocation.

Specifically, **this native memory region grows automatically by default**.

We also have new flags to tune the memory:

* *MetaspaceSize* and *MaxMetaspaceSize –* we can set the Metaspace upper bounds.
* *MinMetaspaceFreeRatio –*is the minimum percentage of class metadata capacity free after [**garbage collection**](https://www.baeldung.com/jvm-garbage-collectors)
* *MaxMetaspaceFreeRatio*– is the maximum percentage of class metadata capacity free after a garbage collection to avoid a reduction in the amount of space

Additionally, the garbage collection process also gains some benefits from this change. The garbage collector now automatically triggers the cleaning of the dead classes once the class metadata usage reaches its maximum metaspace size.

Therefore,**with this improvement, JVM reduces the chance to get the *OutOfMemory* error**.

Despite all of these improvements, we still need to monitor and [tune the metaspace](https://www.baeldung.com/jvm-parameters) to avoid memory leaks.

**4. Summary**

In this quick write-up, we presented a brief description of PermGen and Metaspace memory regions. Additionally, we explained the key differences between each of them.

PermGen is still around with JDK 7 and older versions, but Metaspace offers more flexible and reliable memory usage for our applications.