Java Heap & GC tuning

Java "Heap" is a continuous memory region where all Objects data will be stored (by data, we mean instance of class, primitive and references). It's a big part of the process heap. It can be configured using the following parameters :

* **-Xmx** : **max heap size** (ex: -Xmx1024)
* **-Xms** : **min heap size**. Having -Xms = 1.8GB (32bit) can be bad, because you don't let memory for anything else.
* **-Xmn** : the size of the heap for the young generation, **size of Young Generation**  
  Young generation represents all the objects which have a short life of time. Young generation objects are in a specific location into the heap, where the garbage collector will pass often. All new objects are created into the young generation region (called "eden"). When an object survive is still "alive" after more than 2-3 gc cleaning, then it will be swap has an "old generation" : they are "survivor" .  
  Good size is 33%
* **-XX:NewRatio** : the same as Wmn, but using a % (dynamic fs static -Xmn option). -XX:NewRatio=3 means that the **ratio between the old and young generation** is 1:3
* **-XX:NewSize** - **Size of the young generation at JVM init**. Calculated automatically if you specify -XX:NewRatio
* **-XX:MaxNewSize** - The **largest size the young generation** can grow to (unlimited if this value is not specified at command line)
* **-XX:SurvivorRatio** : "old generation" called tenured generation, ratio, in %. For example, -XX:SurvivorRatio=6 sets the ratio between each survivor space and eden to be 1:6 (eden is where new objects are created)
* **-XX:MinHeapFreeRatio**: default is 40%. JVM will allocate memory to always have as minimum 40% of free memory. When -Xmx = -Xms, it's useless.
* **-XX:MaxHeapFreeRatio**: default is 70%. The same as Min, to avoid unecessary memory allocation.

Permanent Space : It's the third part of the memory. Here are stored classes, methods etc.

* -XX:PermSize: initial value
* -XX:MaxPermSize: max value

# Definition of Memory leak

A memory leak occurs when object references that are no longer needed are unnecessarily maintained.

A memory leak in Java is a situation where some objects are not used by application any more, but GC fails to recognize them as unused.

A memory leak occurs when memory acquired by a program for execution is never freed-up to be used by other programs and applications.

A situation where memory which is no longer needed is not released.