Java Memory Model

JVM memory is divided into separate parts.

At broad level, JVM Heap memory is physically divided into two parts –

**Young Generation** and **Old Generation**



**Young Generation**

The young generation is the place where all the new objects are created.

When the 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

* 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.

**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 **a longer time**

**Issue with Garbage Collection**

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 a long 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

### 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.

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

### 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.

### Memory Pool

Memory Pools are created by JVM memory managers to create a pool of immutable objects like String

Memory Pool can belong to Heap or Perm Gen, depending on the JVM memory manager implementation

### 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 part of the method area.

### 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 is getting referred from the method