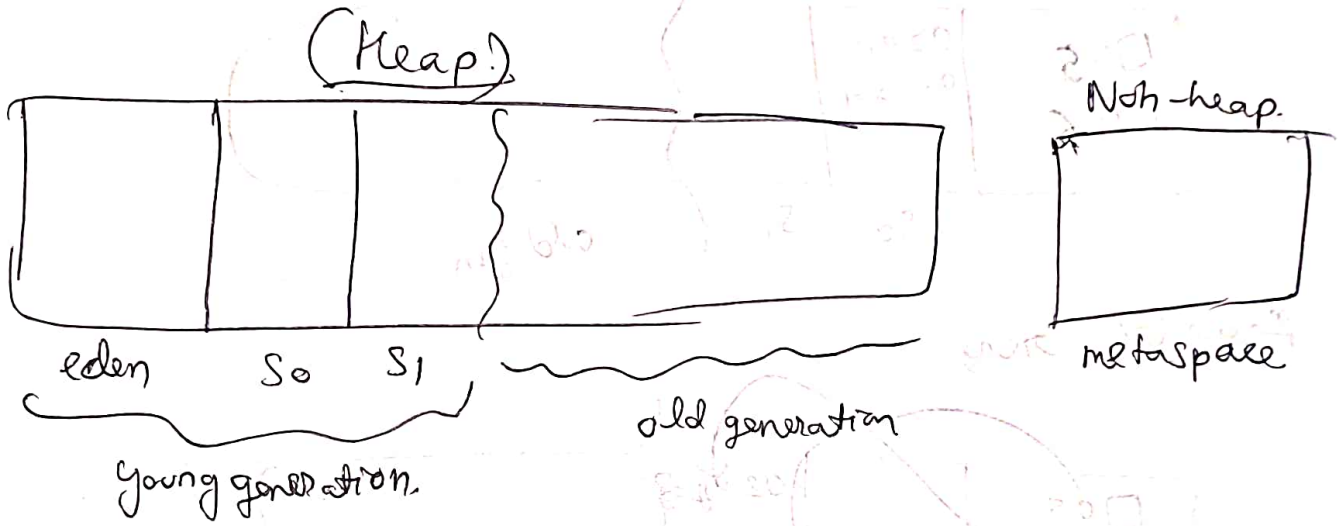
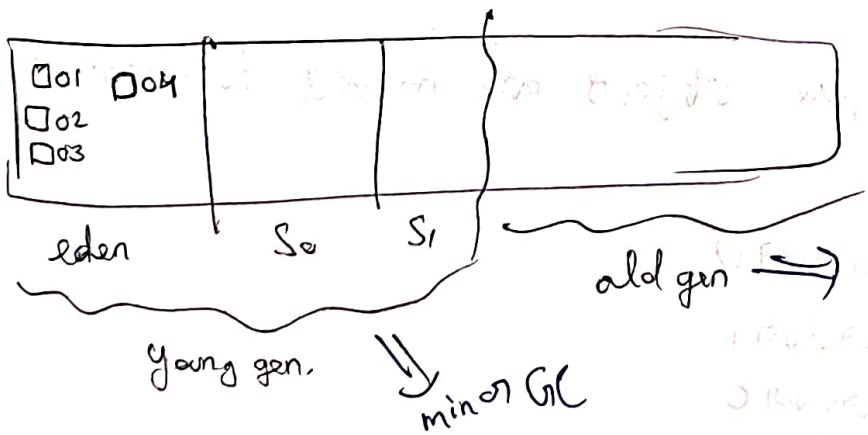
Heap Memory Managment and Garbage Collection in Java

Java heap memory divided into two parts.

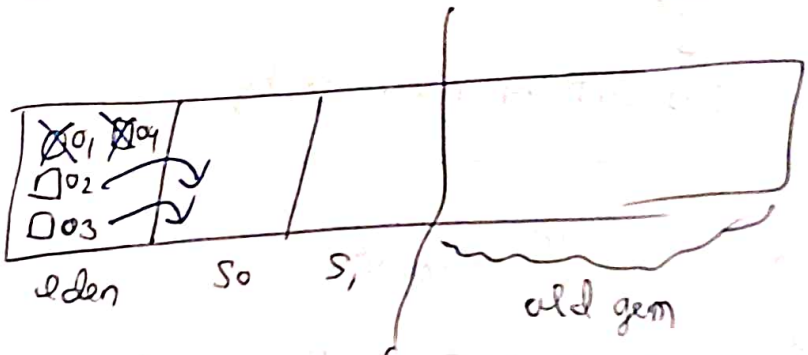


Whenever an object is created it goes to young generation's eden section. - Let's say 01, 02, 03, 04 objects were created



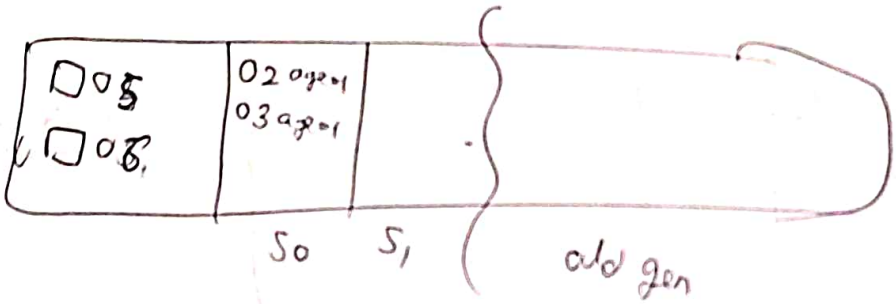
minor GC runs more frequently

lets say 1 GC (minor) cycle ran, and O1 and O4 are eligible for GC

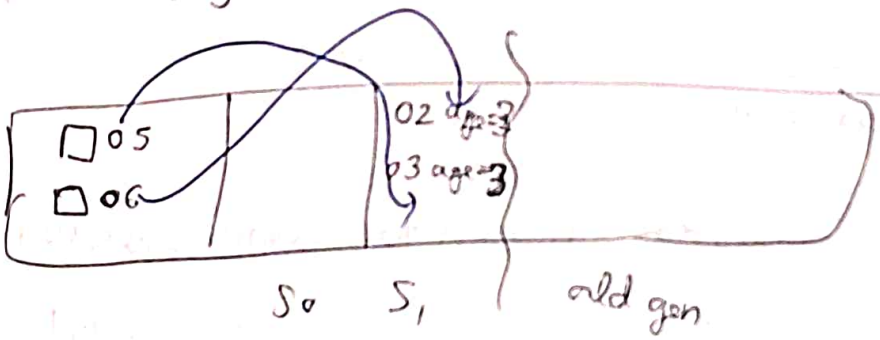


O2 and O3 move to So → Survivor 0.

Now lets say new objects O5 and O6 gets created.



Again GC runs.



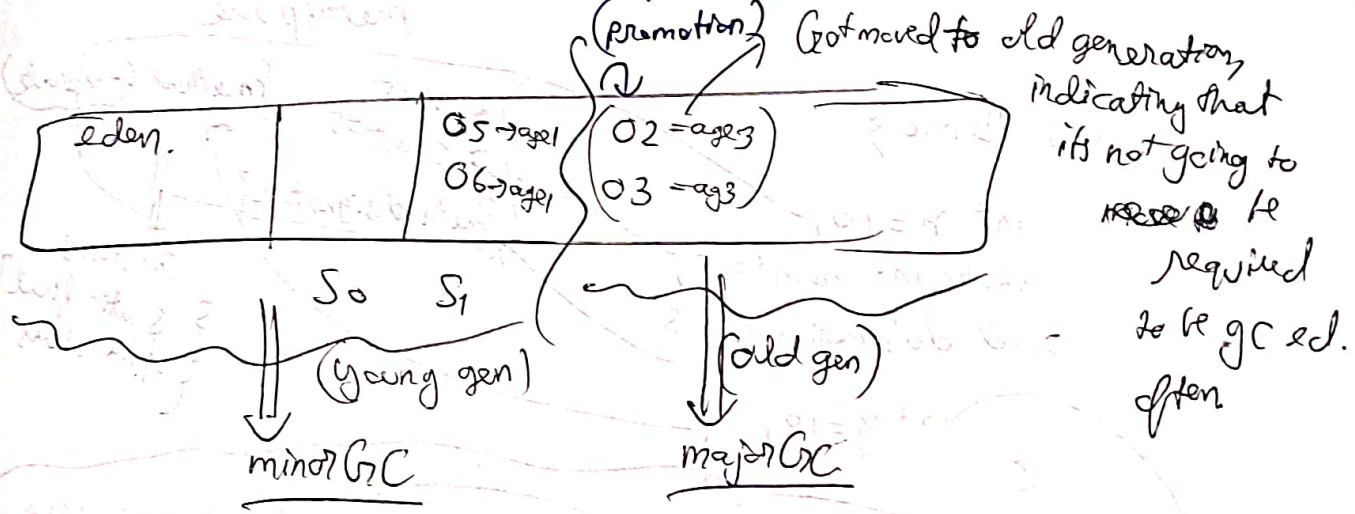
O5 and O6 move to S1.

So with each GC cycle objects are moved to alternating Survivors.

1st cycle → survivor 0
2nd cycle → survivor 1
3rd cycle → survivor 0.
⋮
So on.

In GC there is an age threshold parameter, called (threshold = 3)

if any survivor objects have survived / or have aged upto threshold, then you know these are not going to be required to be GC'ed that often

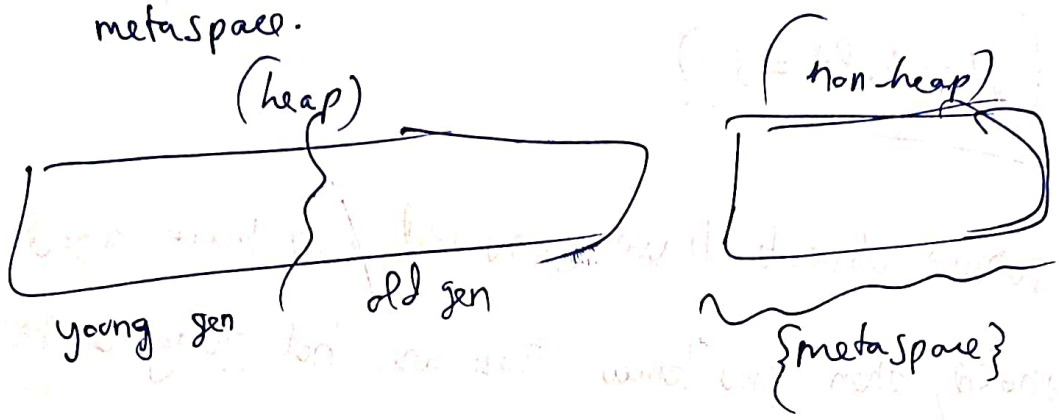


The algo of ① checking which objects need to be garbage collected and marking them → mark

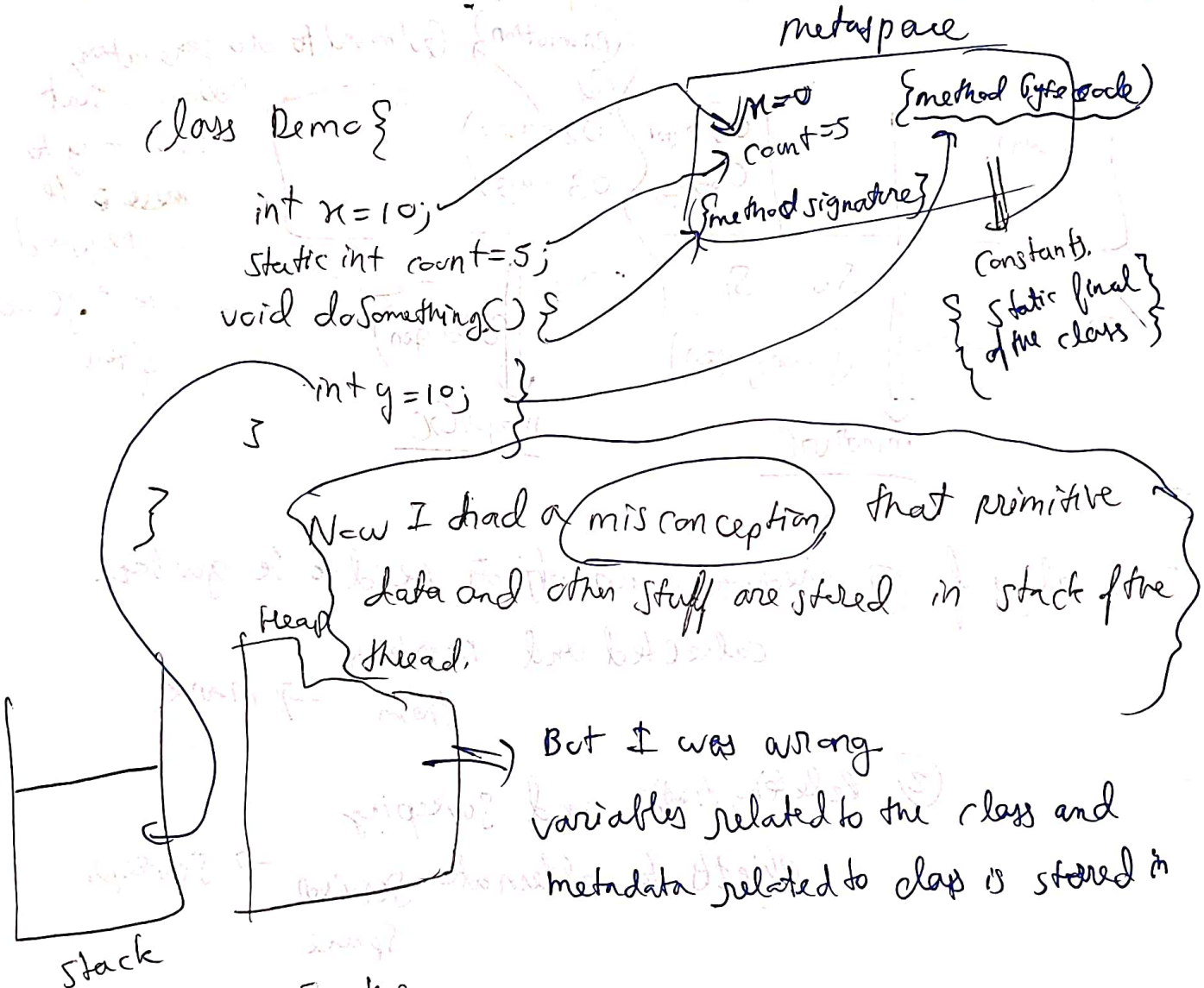
② Deleting those and sweeping objects to alternate survivor space → sweep.

This algo is called Mark & Sweep

now coming to ~~stack~~ metaspace.



Q) What is stored in Metaspace?



New I had a misconception that primitive data and other stuff are stored in stack of the thread.

But it was wrong

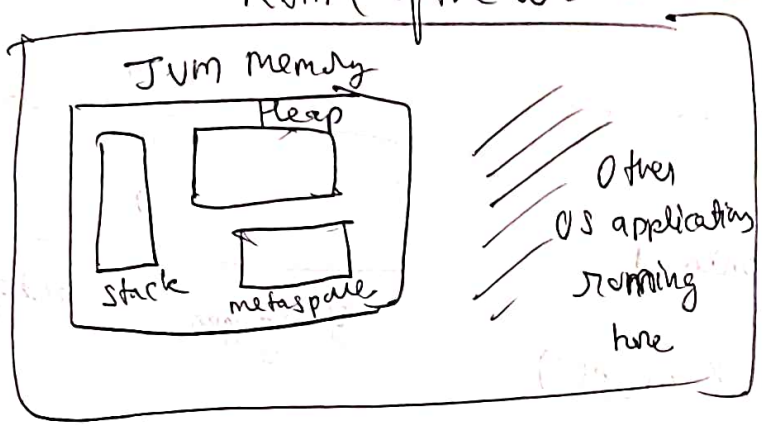
variables related to the class and metadata related to class is stored in

So the {y=10} → inside of doSomething() will be used in the stack,

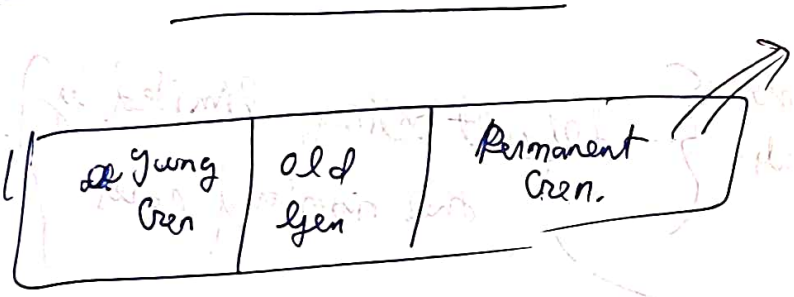
but x=10 is part of the class so in metaspace.

Now lets see the memory structure

RAM (of the whole machine).



Earlier in Java 7, metaspace was inside heap.
called Permanent Generation.

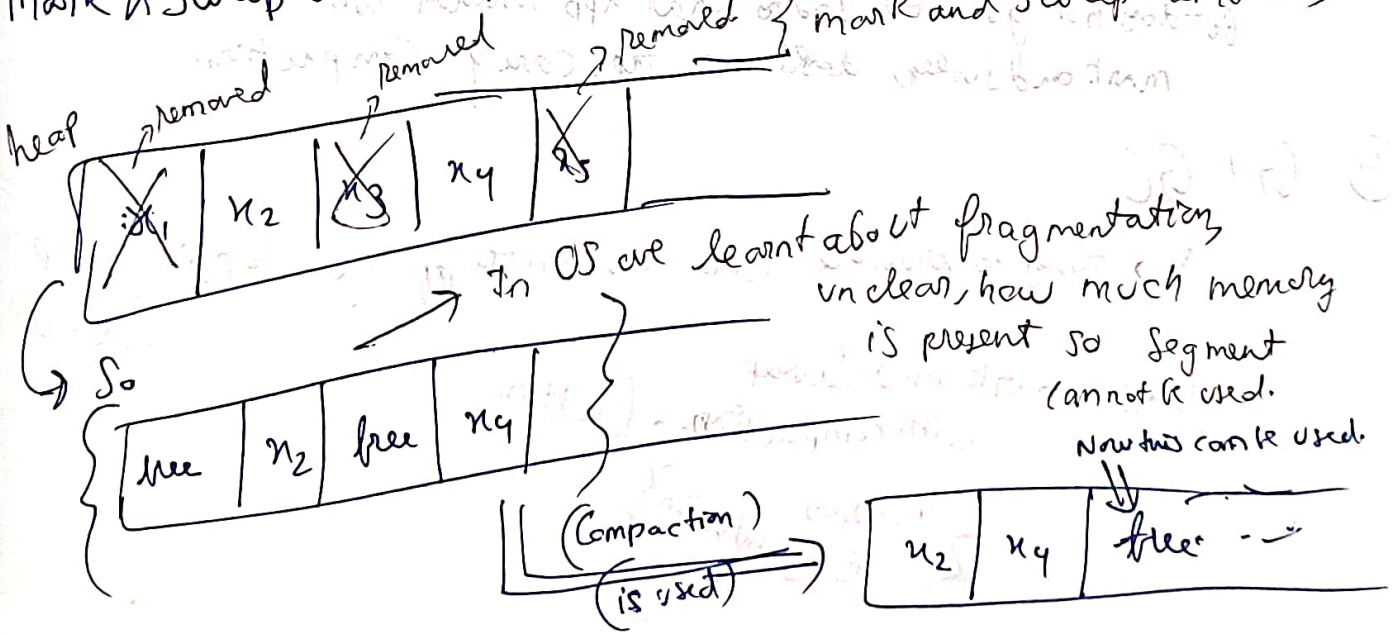


Big class with Big byte code eating up uncessary heap space, so out of heap memory error can come

So Java separated it.

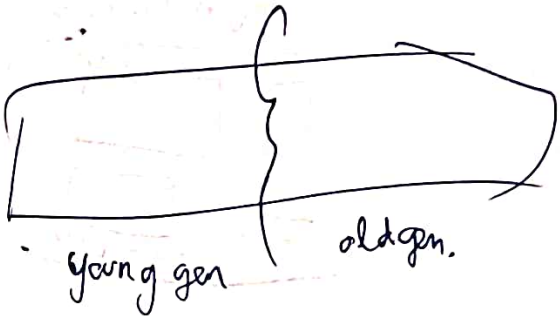
Now there is an improved mark and sweep Algo called

Mark & Sweep with Compaction.



Different Versions of GC.

① Serial GC → single thread.



During stop the world, only 1 thread doing the cleanup().

1 thread going through segments

(by), taking long time. (more time stop the world).

② Parallel GC → multiple threads. { default setting limited by the number of cores }

Faster,

threads can scan segments parallelly

③ Concurrent Mark and Sweep

↳ tries to do GC without stopping application threads. But does not guarantee 100% that app threads will run. Also this is mark and sweep, does not take care of Compaction.

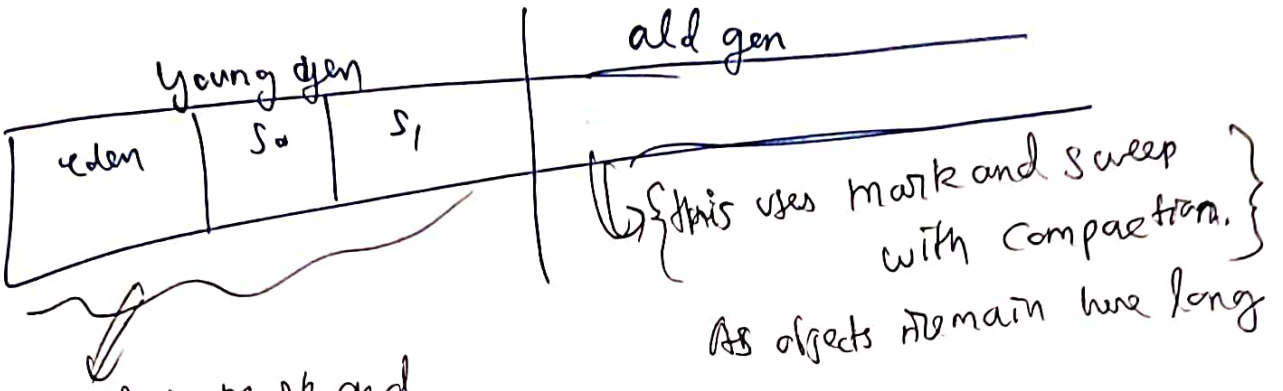
④ G1 GC

↳ tries to provide more guarantee of not stopping app threads.

Uses mark and sweep with compaction. - Better.

{ Reduces Fragmentation }

One more thing, you can run two separate algos for young and old gen.



As objects remain live long

uses plain mark and sweep.

Since whatever new objects come are written to eden,

So it just (Step 1) Copy objects from eden (old) and places them in the survivor.

(Step 2) overwrites the contents of eden with new objects

Summary → Different section using different GC Algo.