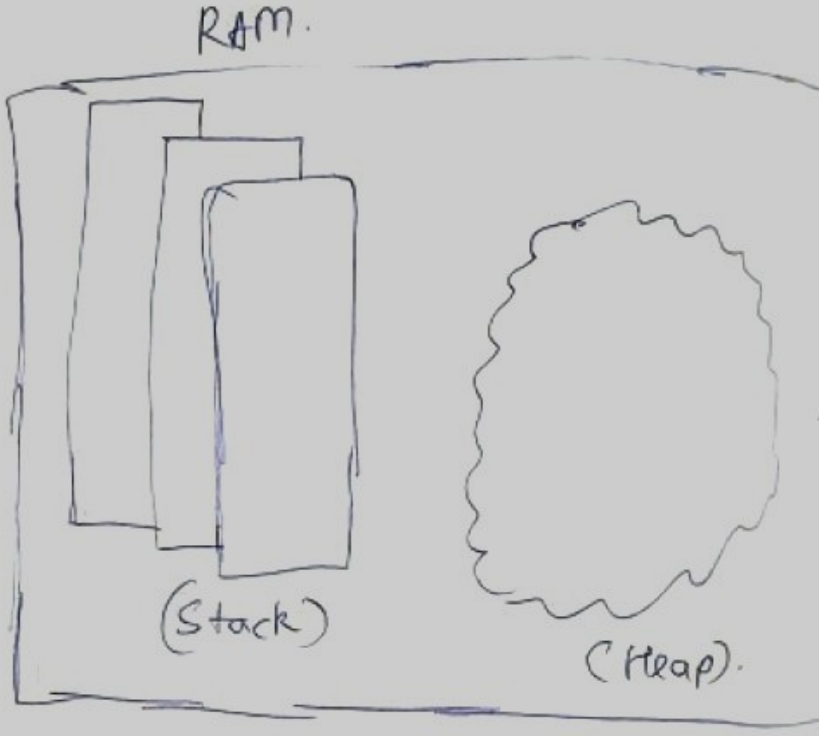
9. Java Memory Management and Garbage Collection in Depth.

2 types of Memory (RAM)

(- Stack
- Heap)

For each Process



(Stack Memory)

* temporary variables references basically
* Stack has scope for each method and inside that scope the method variables are stored.
* Each thread has its own stack
* After method scope is finished stack will pop items in LIFO fashion.

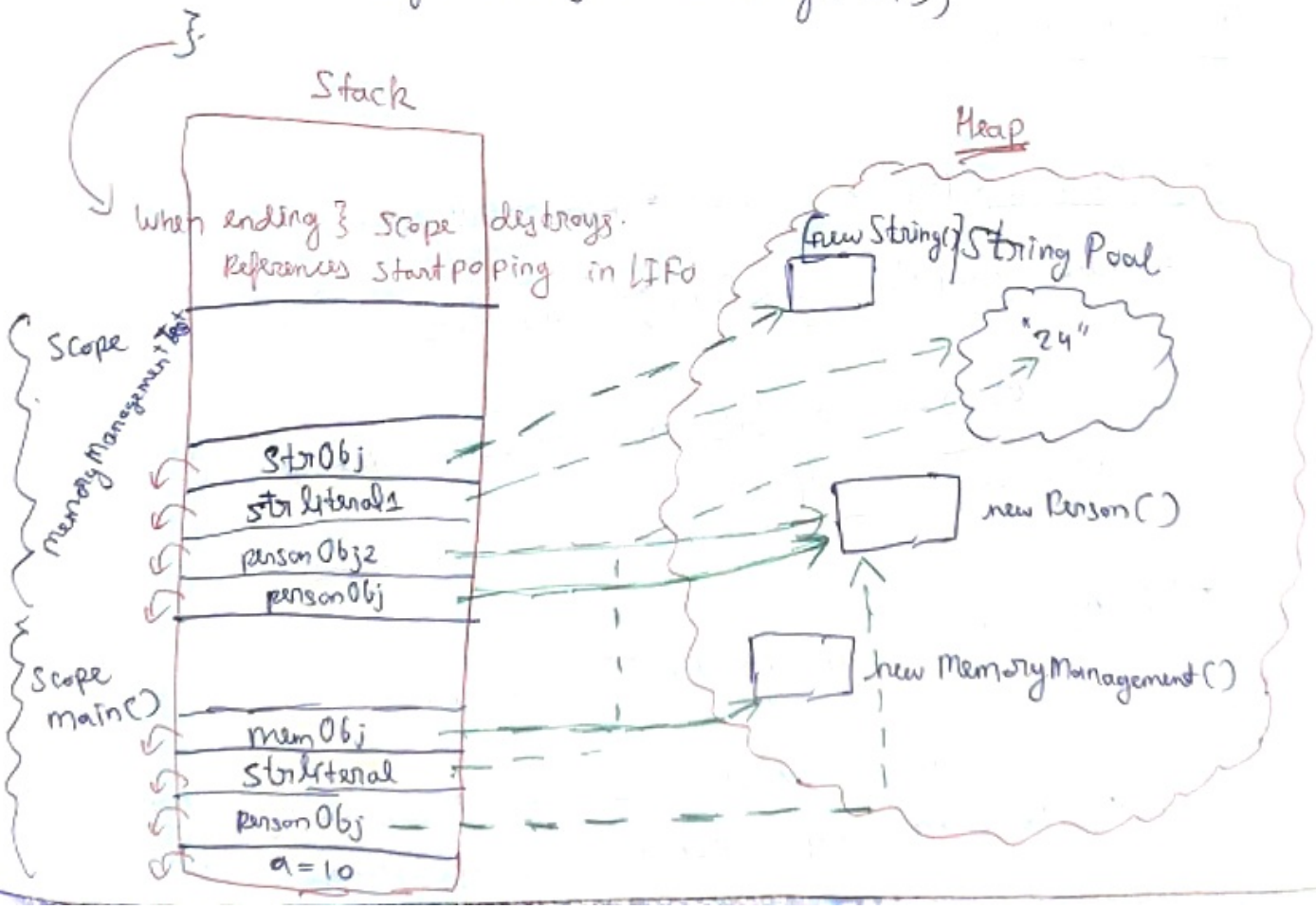
(Heap Memory)

* Used to store Objects. Actual memory Allocated Here
* No ordering of allocating memory
* GC periodically runs and cleans out unreferenced memory.
* String Pool is stored in Heap, ~~which is used~~

(Example)

```
public class MemoryManagement {

    psvm() {
        int a=10;
        Person personObj = new Person();
        String strLiteral = "24";
        MemoryManagement memObj = new MemoryManagement();
        memObj.memoryManagementTest(personObj);
    }
    private void memoryManagementTest(Person personObj){
        Person personObj2 = personObj;
        String strLiteral1 = "24";
        String strObj = new String("24");
    }
```



After all the References for an allocate memory gets pop from the stack. When Garbage Collector (GC) runs then it frees that memory

Different types of References

① (Strong Reference)

Person p = new Person() → (Strong Reference).

only when p = null / p → points to different memory and all references for new Person() is removed then only GC can remove it.

② (Weak Reference)

WeakReference<Person> weakObj = ~~new WeakPerson~~

= new WeakReference<Person>(new Person());

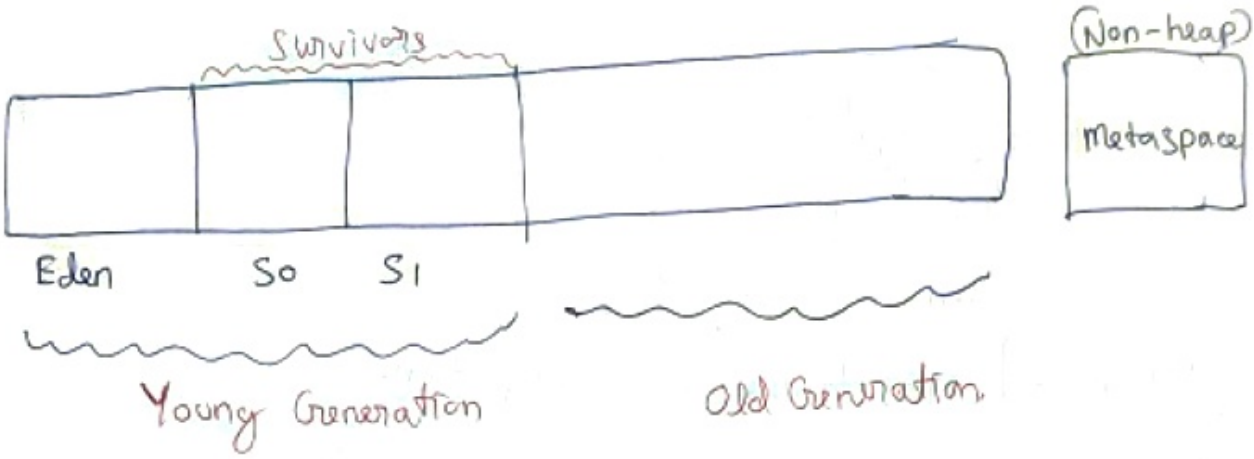
As soon as GC runs it will free up this space

~~As soon~~

③ (Soft Reference) It's a type of Weak Reference.

It tells GC that you can delete the ~~it~~ memory only when its very urgent. Like when Heap is running out of space.

Heap Memory Management and Structure.

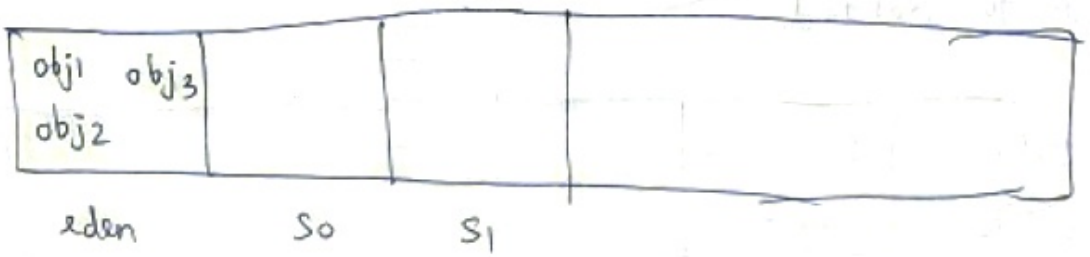


Working of GC

* When new Objects are created, they are first stored in the Eden section (Young Generation).
* Lets say few objects got unreferenced. The GC runs. It uses Mark & Sweep Algorithm. It firsts marks the objects which are to be deleted, and then sweeps them aside.

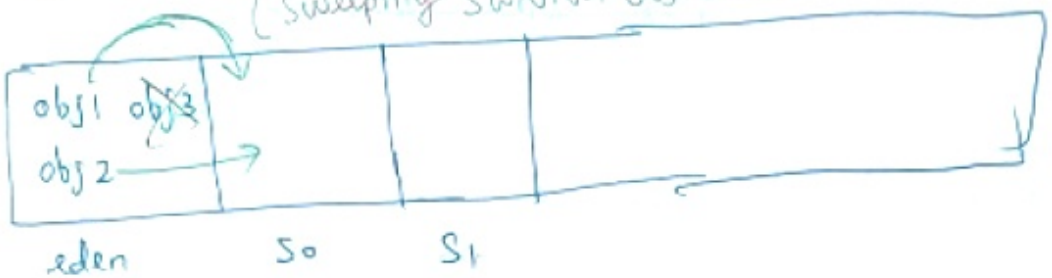
Lets see with an example.

Step1 → obj1 obj2 obj3 → 3 objects newly created.



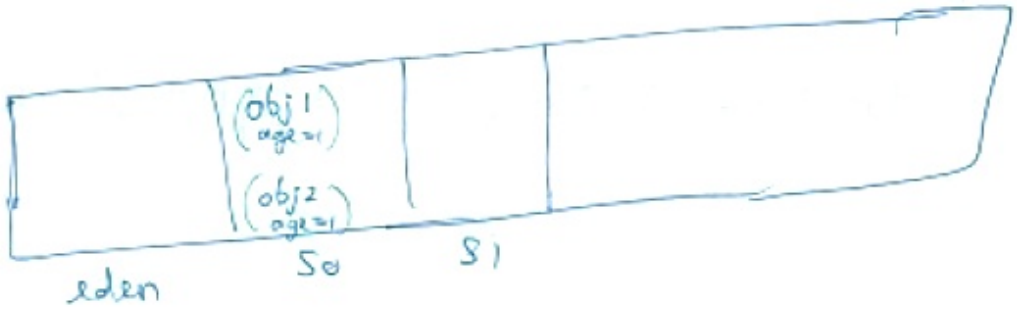
Step2

Lets say obj3 gets unreferenced.

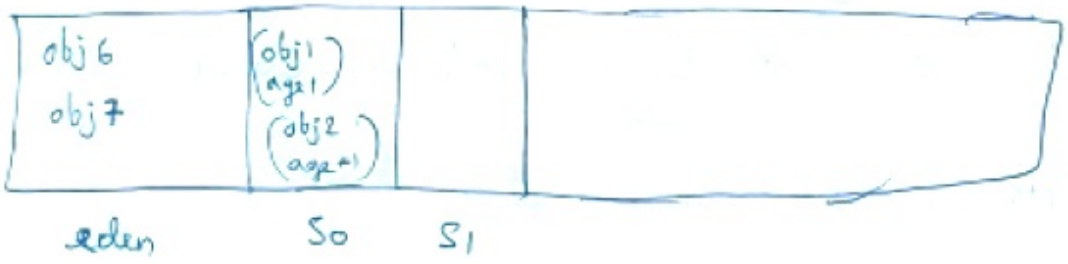


[obj1/obj2] move to the survivor section.

A new attribute called age gets added to them.

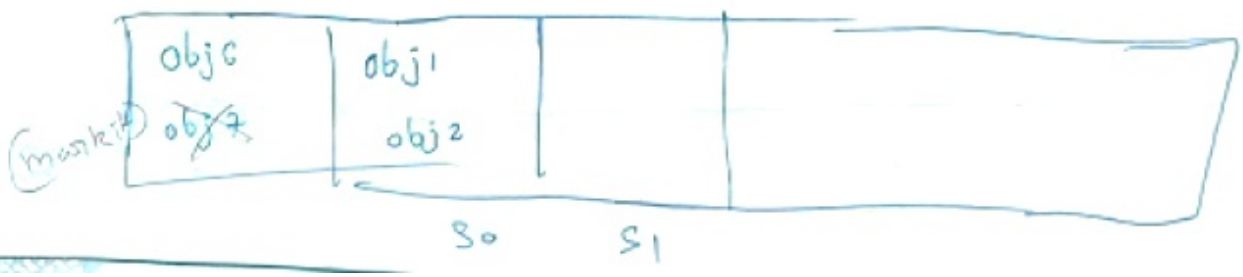


Now lets say obj6 and obj=7 gets added as well

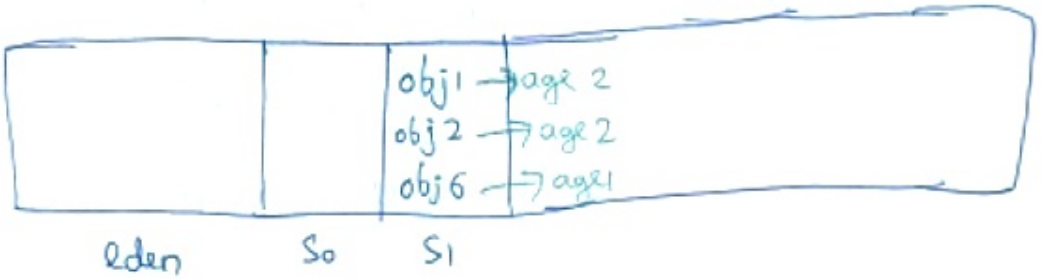


Step3

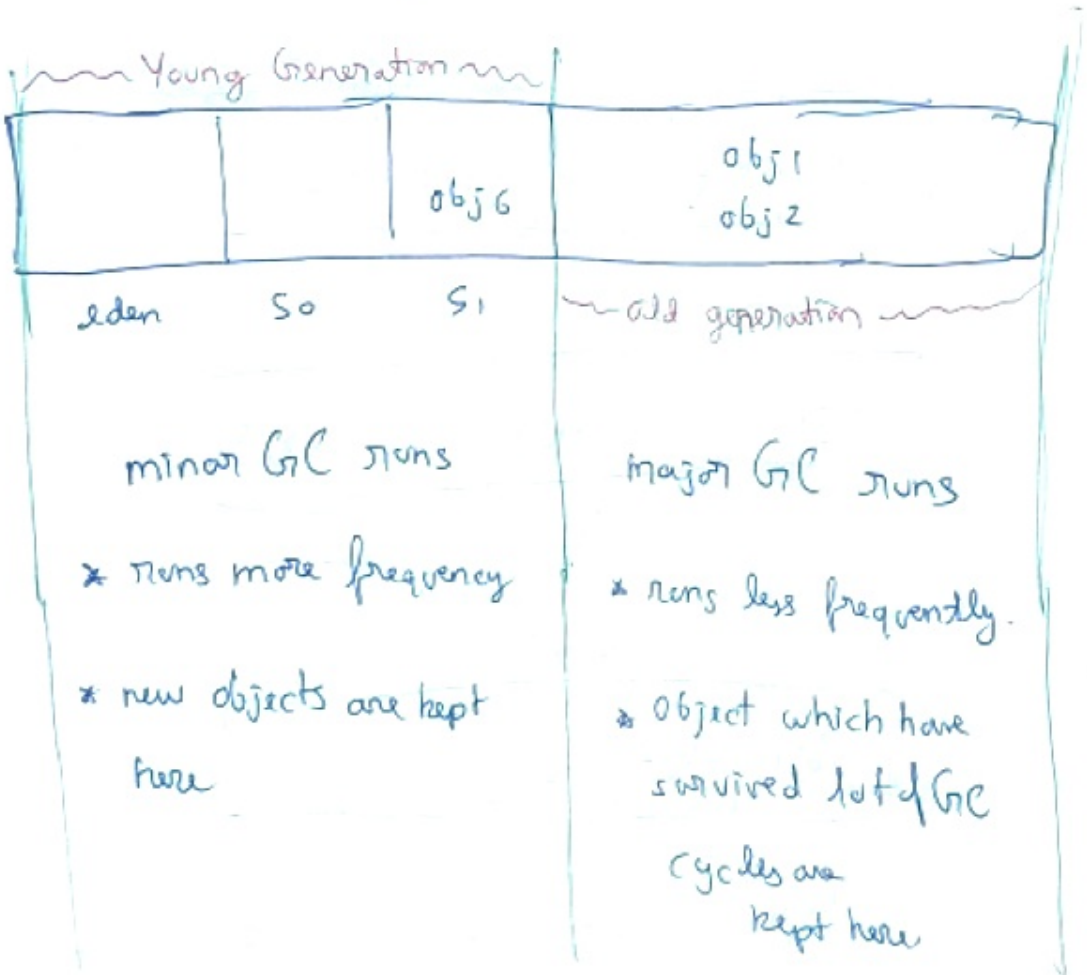
Lets say obj7 becomes unreferenced and has to be to deleted.



Now it will put the alive objects into S1 survivor section. It does this periodically basically.



In GC there is a threshold age. Let say 2.
which ever objects will achieve the threshold age, will be promoted to old generation.

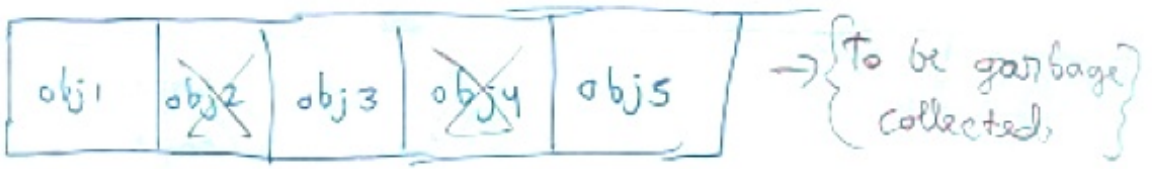


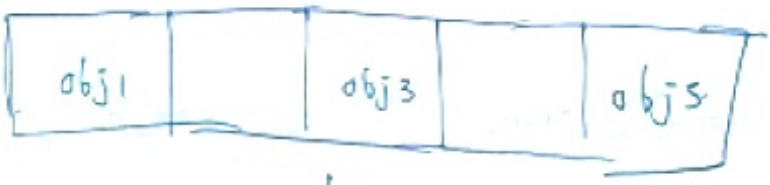
Garbage Collection, Algorithms and types

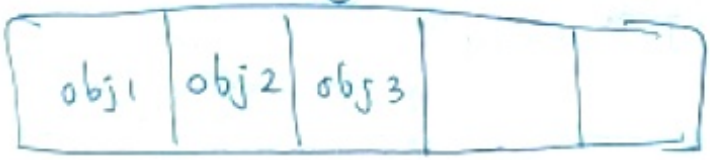
① Mark & Sweep Algo → Simple Algo. Mark which objects need deletion, and delete them and sweep surviving objects into ~~S0~~ S0/S1 from Eden.

② Mark & Sweep with Compaction →

Lets says memory block is something like.







Making continuous segment of Free Space.

versions of G.C.

1. Serial GC → only 1 thread to work as (minor GC 1st time, major GC → 2nd time)
2. Parallel GC → (default in Java 8) → [Multiple threads working together in clean up]
3. Concurrent Mark & Sweep (CMS) → [No compaction]
   ↳ tries to run GC threads in parallel with app.
4. G1 Garbage Collector [Compaction is there] Not 100% guarantee
   ↳ runs GC threads in parallel

GC [Garbage collection] is a very expensive task. When GC works the whole application stops. So if GC is slow application is slow.