JAVA: Memory Management and Garbage Collector.

By Saurav Saxena.

1. What is Memory Management ?
2. Types of Memory.

- STACK
- HEAP

□Both Stack and Heap are created by JVM and stored in RAM.

> ***Generally Heap has more memory than Stack.***

3. Stack Memory.

- Store Temporary variables and separate memory block for methods.

```
//block or temporary variables.
{
  int a = 10;
}
```

- Store Primitive data types.
- **Store reference** of the heap objects.

□ Strong reference
□When garbage collector runs strong reference object not get's deleted.

```
//Strong reference
Person obj = new Person();
```

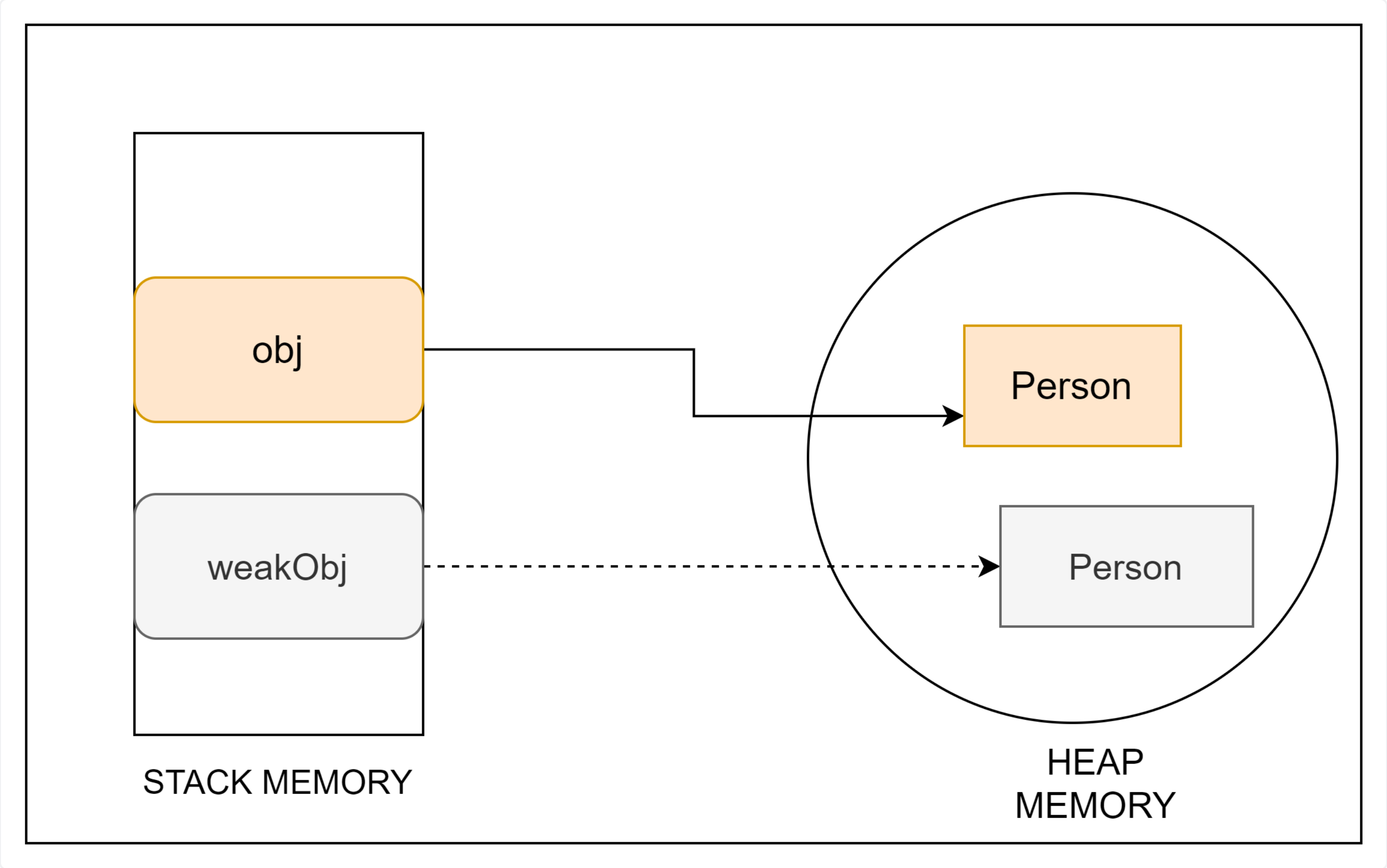
- **Weak reference**

□When garbage collector runs it deletes the weak reference object.

Soft reference

- soft reference is a type of weak reference .
- In that case when garbage collector runs soft reference object tells that hey! GC Only delete me when it's very very urgent.

```
//Weak Reference
WeakReference<Person> weakObj = new WeakReference<Person>(new Person());
```



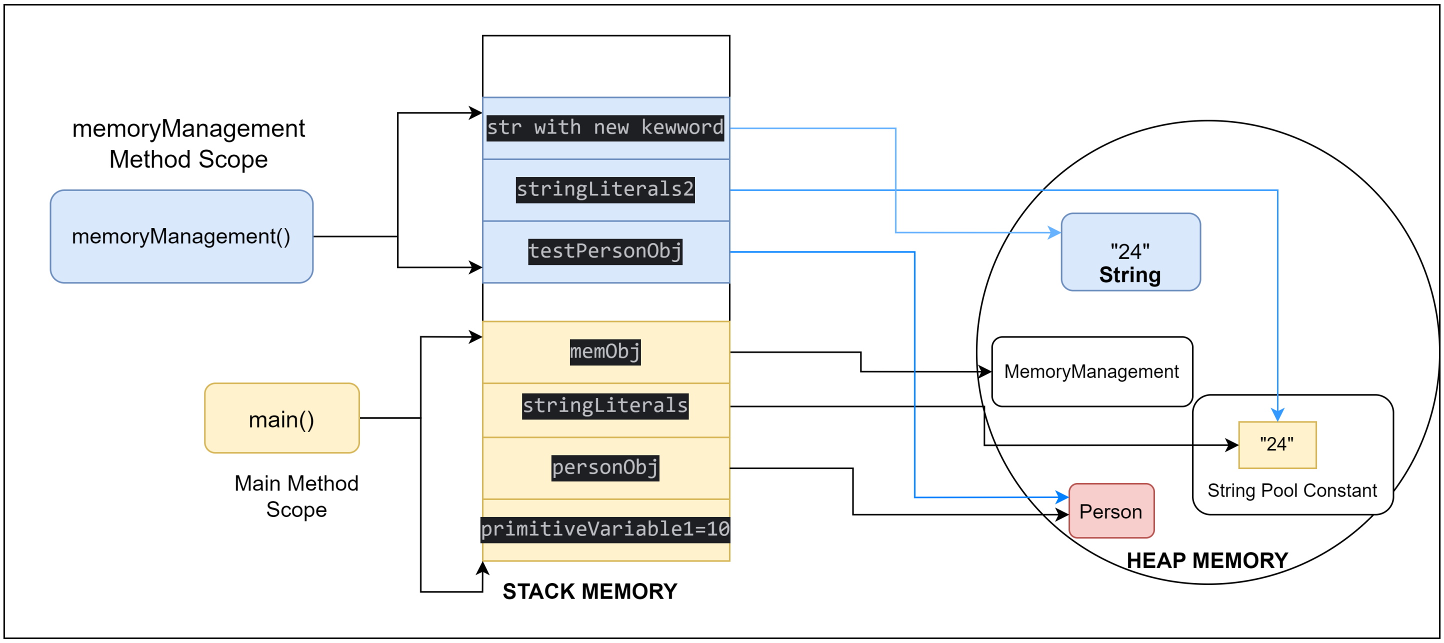
- Each thread has it's own Stack Memory but it shares same common heap memory.
- Variables within a SCOPE is only visible ans as soon as any variables goes out of the SCOPE, it's get deleted from Stack [In LIFO order].
- When Stack memory goes full, it's throws

"java.lang.StackOverflowError"

```
class Person{
    protected void profession(){
        System.out.println("I am Person");
    }
}
public class MemoryManagement{

    //main() method
    public static void main(String args[]){
        int primitiveVariable1 = 10;
        Person personObj = new Person();
        String stringLiterals = "24";
        MemoryManagement memObj = new MemoryManagement();
        memObj.memoryManagementTest(personObj);
    }

    //memoryManagementTest() method
    private void memoryManagementTest(Person person){
        Person testPersonObj=person;
        String stringLiterals2 = "24";
        String str = new String("24");
    }
}
```



- When **memoryManagementTest method** scope got closing bracket.
- In Stack memory **memoryManagementTest method** scope also got deleted as well as reference is also disconnected from heap memory in **LAST IN FIRST OUT [LIFO]** Manner.
- After deleting scope of **memoryManagementTest method** code execution encounter closing bracket of **main() method** scope, and **main method scope** is also got deleted from **Stack memory** as well as it's reference is also got disconnected from **heap memory**.
- Apart from that we have seen all reference of **Stack memory** to **Heap memory** it's get's disconnected.
- But all the **objects** and **String Pool constant literals** are still there in **heap memory** .
- And this is a serious problem,
- To resolve this problem **GARBAGE COLLECTOR** comes into picture.

4. Garbage Collector.

- It is used to delete the un-referenced Object from the heap.
- Garbage collector runs periodically and JVM Controls it.
- JVM Controls When to run garbage collector.
- System.gc() to run garbage collector manually.
- Garbage Collector scan the heap memory if it find any obj area which has not any referenced it will be deleted by garbage collector.
- If you write System.gc() in java code there is no guarantee JVM will run the garbage collector.

Multiple Way to invoking Garbage Collector.

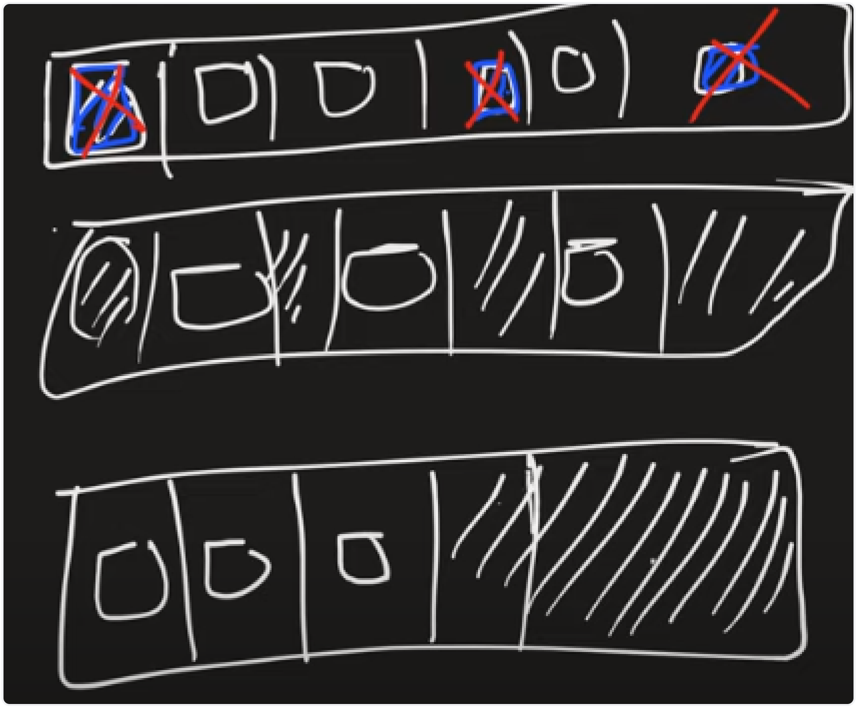
```
//Way to Invoking Garbage collector
Person p = new Person();
p=null;
Person p1 = new Person();
Person p2 = new Person();
//here pl refrencing obj got disconnected //and GC will soon delete it.
p2=p1;
//When Heap memoary is going to full then //garbage collectors call.
```

5. Heap Memory .

- Store Objects.
- There is no order of allocating the memory.
- Garbage Collector is used to delete the un-referenced objects from the heap.

 Mark and Sweep Algorithm with Compaction.
Compaction helps to organize object into sequential manner.

How Memory Compation work's see figure below.



Types of GC:

□Serial GC

- In case of Serial GC one thread working for **Minor GC** and one thread working for **Major GC**.

DisAdvantage:-

- Application Slow [Pausing Too Many times]
- **GC** is very expensive

□ Whenever **GC** Works Start All application thread will get's PAUSE.

□Parallel GC

- Multiple thread working for **GC** in Parallel but still there is a pause in application

□CMS [Concurrent Mark & Sweep]

- While your Application Thread are working concurrently GC Thread are also working.
- JVM Not give 100% guarantee that your application is not going to pause.
- JVM Try to do best concurrent but it's not give 100% guarantee.
- There is No Memory Compaction Happens.

□G1 Garbage Collector.

- **G1 Garbage collector** gives 100% guarantee to not pause in application and there is memory compaction happens.

- All the freed up memory appended in last after memory compaction.
- Throughput Increases Let's say:□1000 request/sec or min.
- **Latency** Decreases. Threads are not pausing

- Heap Memory is shared with all the threads.
- Heap also contains the String Pool.
- When Heap memory goes full, it's throws

"java.lang.OutOfMemoryError"

- Heap memory is further divided into:

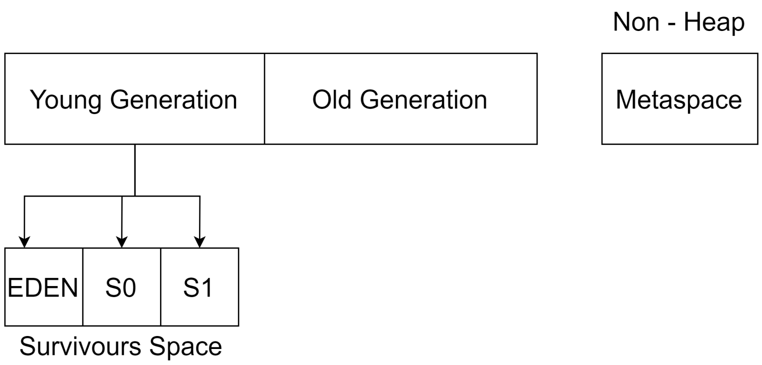
□ Young Generation [Minor GC happens Here]
□ Eden
□Survivor
□Old/ Tenured Generation
[Major GC happens here]
□ Permanent Generation [MetaSpace].
□bEfore Java 7 it was part of Heap memory
□ is separated from Heap Memory

- ***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.*
- *Currently known as* ***MEATSPACE***

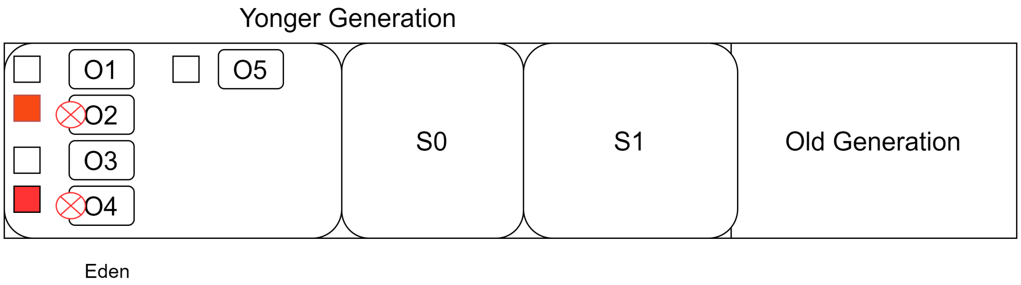
□ **MetaSpace**

- Stores Class variables
- Stores Constant [static final]
- Store Class Meta data

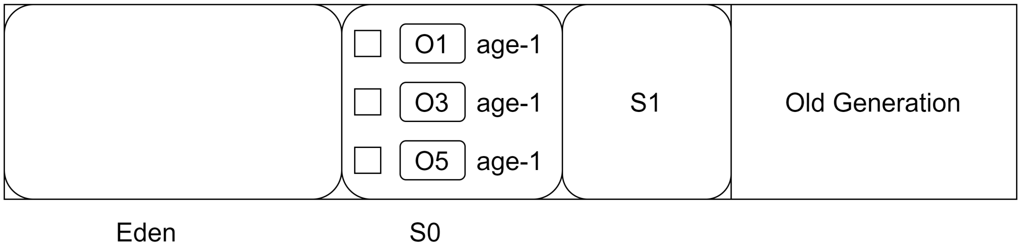
- How Objects Store In Heap Memory and it's Internal Working Step Explain I below **Diagram**.



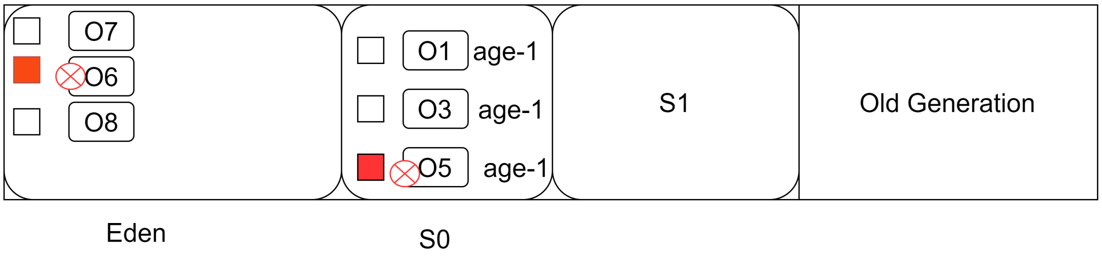
First object created in **EDEN** and Very First time **Garbage** collector calls and mark un-referenced Object for removal from Memory.

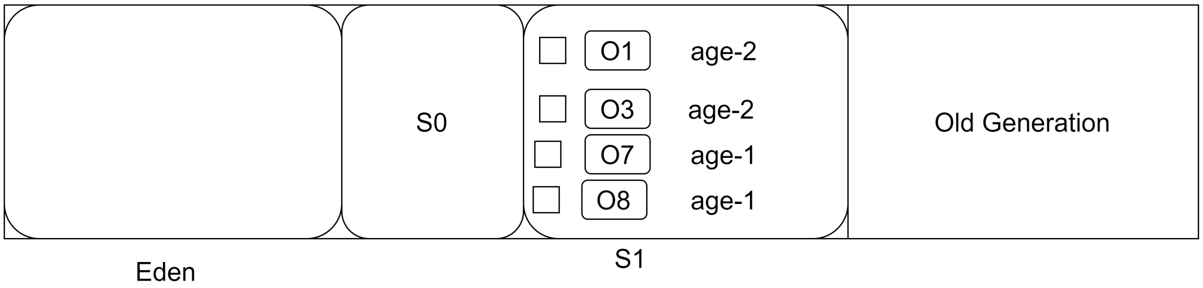


Apply Algorithm **Mark** & **Sweep** :- Sweep algo remove the **mark** objects and sweep the **survivours** objects to **S0** state.

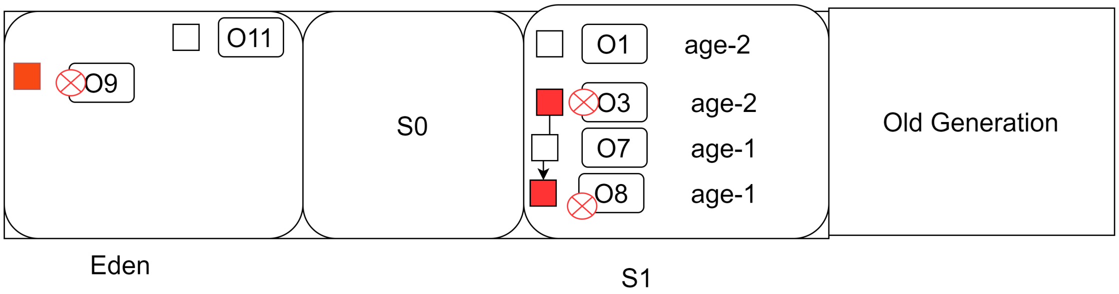


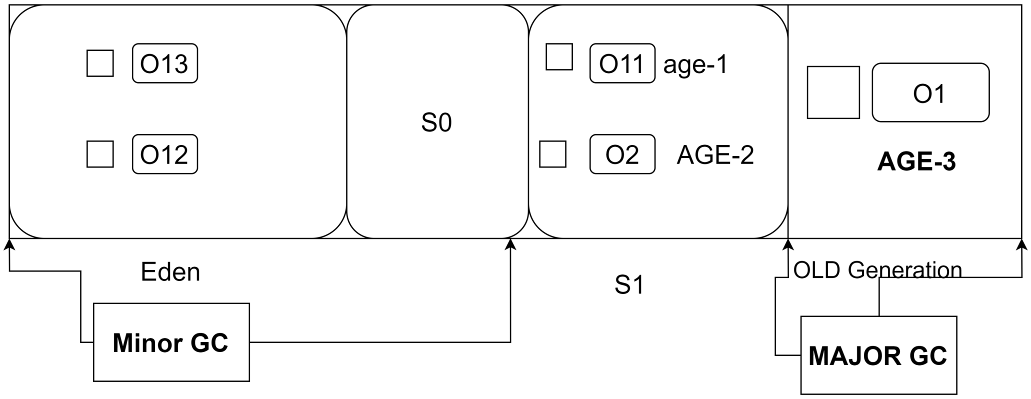
Again **Garbage** Collector Calls for 2nd time it's do the same process for **eden**, s0, s1 and so on.





Threshold value is **age=3** :-> It means that if object reach that the age limit of 3 that object will promoted to old generation. So Here **Garbage** Collector Call for **3rd** time.


O11
O9
S0
O1
age-2
O3
age-2
O7
age-1
O8
age-1
Old Generation
Eden
S1


O13
O12
S0
O11
age-1
O2
AGE-2
O1
AGE-3
Eden
S1
OLD Generation
Minor GC
MAJOR GC