Memory Management

1. STACKS
2. HEAPS

- Managed by **JVM**

STACK

- **Temporary variables -** inside particular block, stored inside stack
- **Memory Frame** - function call variables (inside stack)
- **Reference of object** - inside stack
- Each variable is visible inside its own scope
- Each Thread have its own Stack memory
- However, each thread share common Heap memory
- When variables goes outside scope, its gets deleted from Stack in a LIFO order
- Once java sees closing bracket, it starts removing from Stack
- Garbage Collector is used to delete unreferenced object from the heap.
- JVM controls, when to run garbage collector
  - We can run garbage collector using System.gc( ), though no guarantee, still depends on JVM
  - Automatic Memory Management - garbage collector runs periodically

REFERENCE TYPES

- **Strong Reference** - normal usage
- **Weak Reference** - WeakReference class
  - As soon garbage collector runs, the space occupied by object in Heap, which is weakly referred will be freed up.
- **Soft Reference** - Like weak reference, but freed up when very urgent (no more space in Heap)

HEAP MEMORY

1. Young Generation
    1. Eden
        - new Object goes here
    2. Survivor space
        1. S0
        2. S1
5. Old Generation / Tenured Generation
    - Major GC - runs less periodically
    - They are big objects present here
    - Does mark and sweep similarly

Non-Heap MetaSpace - out of Heap

- Stores class variables - static variables
- class metadata

    - constants - static final
    - When class not needed, data will be removed

Mark and Sweep Algo

3. Mark the objects, which are no more referenced, and allowed to delete
4. Sweep - removes mark objects from memory, and moves rest Objects from Eden to Survivor space. Adds age to those objects.
    - Process called Minor GC
    - Alternatively sweep algo moves objects from S0 to S1 and increase their age, and Eden will be empty.
    - Threshold age - when this is met, objects reaching such age are moved to old generation

- Mark and Sweep with Compact
    - Brings remaining objects in memory together in sequential manner

Versions of GC

3. Serial GC
    - Only 1 thread be working
    - Slow and expensive
    - Application thread pauses, if GC runs
4. Parallel GC - default Java 8
    - Multiple threads
    - Fast and less pause time
5. Concurrent Mark & Sweep (CMS)
    - Application and GC thread are working concurrently
    - Application thread do not stop with GC threads - but not guaranteed
    - No memory compaction
6. G1 GC
    - Better version of CMS
    - Application threads do not stop
    - Brings Compaction