Sample Java JVM GC Workflow for an Object

GC Workflow

1. **Creation**:
   - A new object A is created using the new keyword (e.g., Object A = new Object();).
   - Memory for object A is allocated in the **Eden space** of the **Young Generation**.
   - A constructor initializes the object, setting its fields and state.
2. **Usage**:
   - Object A is referenced by a local variable in a method or from a static field.
   - As long as the reference exists and is accessible (reachable), the object is considered "alive."
3. **Minor GC Triggered**:
   - The Eden space becomes full due to continuous allocation of new objects.
   - A **Minor GC** is triggered to free up space in the Eden.
4. **Marking Phase (Reachability Analysis)**:
   - The GC starts with **GC Roots** (active thread stacks, static fields, etc.) and traces references to identify all reachable objects.
   - Object A is found to be reachable, so it is "marked" as alive.
   - Other unreachable objects in Eden are marked as garbage.
5. **Copying (Survivor Spaces)**:
   - Live objects in the Eden space, like object A, are copied to one of the **Survivor Spaces** (S0 or S1).
   - Unreachable objects in Eden are not copied and are discarded.
   - If A was already in a Survivor Space and survived multiple GC cycles, it may be promoted to the Old Generation.
6. **Sweeping (Reclaiming Memory in Eden)**:
   - The GC sweeps through the Eden space and reclaims memory occupied by unreachable objects.
   - The freed memory becomes available for new object allocations.
7. **Promotion**:
   - If object A survives a configurable number of Minor GCs (determined by the -XX:MaxTenuringThreshold flag), it is promoted to the **Old Generation**.
   - This is because surviving multiple GC cycles indicates that the object has a longer lifespan.
8. **Old Generation Filling**:
   - Over time, the Old Generation fills up with promoted objects like A.
   - A **Full GC** (Major GC) is triggered when there is insufficient space in the Old Generation to allocate new objects or meet promotion demands.

9. **Mark-Sweep-Compact in Full GC**:
    - **Mark Phase**:
        - Similar to Minor GC, the GC identifies reachable objects in the Old Generation by traversing references from GC Roots.
    - **Sweep Phase**:
        - The GC reclaims memory from unreachable objects in the Old Generation.
    - **Compaction Phase**:
        - To address fragmentation, the GC moves remaining live objects together to create contiguous free memory, simplifying future allocations.
10. **Finalization (Optional)**:
    - If object A overrides the finalize() method, it is added to the **Finalization Queue**.
    - The finalizer thread invokes the finalize() method on A to perform any cleanup before the object is garbage collected.
    - Finalization can delay the collection of the object, so it is generally discouraged.
11. **Post-GC (Memory Reclamation)**:
    - Memory previously occupied by unreachable objects is now available for new allocations.
    - Object A remains in the Old Generation (if still reachable) or is removed (if unreachable and finalization is complete).
    - If compaction was performed, references to surviving objects are updated to their new locations.

Additional Detailing

Triggers for Minor GC and Full GC

- **Minor GC**:
    - Triggered when the Eden space is full.
    - Efficient and quick, as it operates on the Young Generation only.
- **Full GC**:
    - Triggered by:
        - Old Generation filling up.
        - Explicit call to System.gc().
        - Insufficient Metaspace or metadata pressure.
    - Slower and more resource-intensive.

Conditions for Promotion:

- An object is promoted to the Old Generation if:
    - It survives a certain number of Minor GCs (default threshold is 15, adjustable via -XX:MaxTenuringThreshold).
    - It exceeds a size limit that makes it inefficient to copy to Survivor Space (large objects may be directly allocated in the Old Generation).

Compaction Phase Importance:

- Prevents fragmentation in the Old Generation.
- Ensures that contiguous memory blocks are available for large object allocations.
- Can be resource-intensive, leading to longer pause times during Full GC.

Finalization Caveats:

- Finalization is unpredictable and expensive.
- Use of try-with-resources or explicit cleanup (e.g., close() methods) is preferred over relying on finalize().

Post-GC Effects:

- **Young Generation**:
    - Memory is cleared, and Survivor Spaces are updated.
- **Old Generation**:
    - Freed memory reduces pressure on the heap, delaying subsequent Full GCs.
- **Application Impact**:
    - Reduced pause times in Minor GC.
    - Possible longer pauses during Full GC if heap tuning is suboptimal.