Here are detailed and structured notes from the uploaded document:

Garbage Collection in Java

Introduction

1. In older languages like C++, the programmer is responsible for both object creation and destruction.
    - Common issue: Memory leaks due to neglecting object destruction.
2. In Java:
    - Programmers manage object creation but not destruction.
    - The **Garbage Collector (GC)** is an assistant running in the background to destroy unused objects and free up memory.
3. Objective of Garbage Collection:
    - Automatically reclaim memory occupied by unused objects.
    - Minimize program failures due to memory issues.

What is Garbage Collection?

- **Definition:**
    - Garbage collection is the process of automatically freeing heap memory by deleting unused objects that are no longer accessible.
- The **Garbage Collector**:
    - Runs in the background as a low-priority thread.
    - Ensures unused objects are no longer needed by the running program.
    - Cleans up memory to prevent `OutOfMemoryError`.

Dead Objects (Garbage) in Java

- **Dead Object:**
    - An object no longer accessible by any reference variable.
- **Live Object:**
    - An object that can still be accessed and used by the program.
- Example:

    ```java
    Hello h1 = new Hello();
    Hello h2 = new Hello();
    h1 = h2; // h1’s original object becomes garbage.
    ```

- The JVM automatically detects and reclaims memory occupied by garbage.

Invoking Garbage Collector

- The JVM runs garbage collection:
    1. When memory is low.
    2. Before throwing an `OutOfMemoryError`.
- **Requesting Garbage Collection:**
    - Methods to request JVM to run GC:
        1. `Runtime.getRuntime().gc()`
        2. `System.gc()`

- Example:

  ```java
  System.gc(); // Recommended way
  Runtime.getRuntime().gc(); // Alternative
  ```

Note:

- The JVM is free to ignore GC requests.
- `System.gc()` is preferred as it is static, while `Runtime.gc()` is an instance method.

Object Finalization

- **Finalization:**
  - Automatically performed on objects before their memory is freed by GC.
  - Finalization code is written in the `finalize()` method.
- **`finalize()` Method:**
  - Syntax:

    ```java
    protected void finalize() throws Throwable {
        // Finalization code
    }
    ```

  - Defined in the `Object` class and can be overridden.
  - Called by the garbage collector before an object is destroyed.

Important Points:

1. GC calls `finalize()` only once before an object is destroyed.
2. An object is not destroyed if it’s still reachable.
3. Exceptions in `finalize()`:
   - If raised and uncaught, JVM ignores them.
   - Does not terminate the program.

Behavior of Garbage Collector

1. The GC's behavior is vendor-specific and varies between JVMs.
2. Questions without precise answers:
   - Which algorithm does GC use?
   - When exactly does GC run?
   - In what order does GC destroy eligible objects?
3. **Common Algorithm:**
   - Most GCs use the **Mark-and-Sweep Algorithm**.

Memory Leaks in Java

- **Definition:**
  - Objects no longer used by the application but not eligible for garbage collection.
- **Impact:**
  - Leads to `OutOfMemoryError`.

- **Prevention:**
  - Explicitly make objects eligible for GC when no longer needed.
- **Detection:**
  - Use monitoring tools like:
    - HPJ Meter
    - IBM Tivoli
    - J Probe

Key Points to Remember

1. Garbage collection cannot guarantee immediate memory cleanup.
2. Programmers cannot control the exact timing or behavior of GC.
3. Use `System.gc()` or `Runtime.gc()` to request GC but rely on JVM’s management.

These structured notes provide a comprehensive overview of garbage collection in Java. Let me know if you'd like further refinement or elaboration!