Java Core Refresher Notes

1. Java Lifecycle and Environment Components

1.1 JDK (Java Development Kit)

- Contains:
  - JVM
  - JRE
  - Compiler
  - Debugger
  - `.java` files

1.2 JRE (Java Runtime Environment)

- Contains:
  - JVM
  - Standard libraries (`util`, `lang`, etc.)
- Used to **run** `.class` files (not to write/compile code)

1.3 JVM (Java Virtual Machine)

- Platform dependent
- Execution flow:

```
Java Code (.java)
    ↓
Compiler (JIT)
    ↓
Bytecode (.class)
    ↓
JVM
    ↓
Machine Code
    ↓
CPU
```

2. Java Program Structure

2.1 Main Method

- Must be `public` – JVM uses reflection to call it
- Class name should match the file name

🧠 3. Memory Management

🔷 3.1 Types of Memory

🔷 Stack Memory

- Stores:
  - Primitive values
  - Method-level variables
- Characteristics:
  - LIFO
  - Thread-specific
  - Cleared when scope ends

🔷 Heap Memory

- Objects created with `new` keyword
- Reference is in the stack
- Includes string literals (reference in stack)

🔷 3.2 Garbage Collection (GC)

🔷 GC Algorithms

- **Serial GC** – Single-threaded; pauses app
- **Parallel GC** – Multi-threaded; default in Java 8
- **Concurrent GC** – Works alongside app threads
- **G1 GC** – Balanced and efficient

🔷 Reference Types

- **Strong**: Normal object references
- **Weak**: `WeakReference<Object>` – GC clears even if referenced
- **Soft**: Cleared only under memory pressure
- `System.gc()` – Only a **request** to the JVM

📜 4. Heap Memory Layout

🔷 4.1 Generations

🔷 Young Generation

- Areas: **Eden**, **S0**, **S1**
- Uses **Minor GC** (fast)
- GC Process:
  1. New objects → Eden
  2. Surviving objects → S0
  3. Next GC → Surviving objects → S1

4. Objects with age 3+ → promoted to Old Gen

🔷 Old Generation

- Uses **Major GC** (slow, less frequent)
- Holds long-lived objects

🔷 4.2 Non-Heap Memory (Metaspace)

- Stores:
  - Class metadata
  - Constants
  - Static variables

🔷 4.3 GC Compaction

- **Mark and Sweep Compact**:
  Cleans unused memory and compacts active objects

🔢 5. Variables in Java

🔷 5.1 Language Traits

- **Statically typed** – Type must be declared
- **Strongly typed** – Type-safe with value limits

🔷 5.2 Naming Rules

- Can start with `$`, `_`, or letter
- No reserved keywords
- Constants: **UPPERCASE**

🔷 5.3 Variable Types

🔷 Instance / Member

- Declared in class
- Each object has its own copy

🔷 Static

- Shared across all objects
- Belongs to class
- Accessed using class name

🔷 Local

- Defined inside methods
- Limited to scope

6. Type Conversion

6.1 Widening (Implicit)

- Auto-converts smaller to larger type:

```
byte → short → int → long → float → double
```

6.2 Narrowing (Explicit)

- Requires casting:

```
int num = (int) 234L;
```

6.3 Type Promotion

- Example:

```
byte sum = 127 + 2; // Promotes to int
```

- Mixing types promotes to the largest one in the expression