The **Java Virtual Machine (JVM)** manages several types of memory areas to optimize the execution of Java applications. These memory areas include the **Heap Memory, Stack Memory, Method Area (Meta Space), Program Counter Register, Native Method Stack, and Code Cache.** Here's a detailed overview of each:

**1. Heap Memory**

Used for dynamic memory allocation where all the objects, their related instance variables and arrays are stored.

The heap is divided into several regions to optimize garbage collection and manage the lifecycle of object efficiently **by segregating short-lived and long-lived objects**.

The primary regions/divisions are the **Young Generation** and the **Old Generation**.

* **Young Generation (Young Space)**:

This is where all new objects are allocated and aged. It consists of two spaces as follows,

**Eden Space**: Where new objects are allocated.

**Survivor Spaces (S0 and S1)**: Objects that survive garbage collection in Eden Space are moved to the Survivor Spaces.

* **Old Generation (Tenured Space):**

Where objects that have survived multiple garbage collection cycles in the young generation are moved here. It stores long-lived objects.

**2. Stack Memory**

Used for storing method specific values such as **method parameters, local variables and return addresses.** Each thread has its own stack. **Stack memory is divided into frames**, where each frame corresponds to a method call. When a method is invoked, a new frame is created on the stack, and it is removed when the method call is completed.

**3. Method Area (Metaspace)**

Used for storing class level data such as **class definitions, method metadata, static variables, and constants.** It is shared among all threads. In Java 8 and later, the method area is implemented as **Metaspace** which resides in native memory. It replaces the **Permanent Generation (PermGen)** used in earlier versions.

**4. Program Counter Register**

Each thread has its own PC register, which keeps track of the address of **the JVM instruction currently being executed**. It helps in managing the execution flow of the thread. This is particularly important for **handling multi-threaded execution and thread context switching**.

**5. Native Method Stack**

Used for storing native methods written in the **language other than Java (e.g., C/C++).** Each thread has its own native method stack for executing native code.

**6. Code Cache**

### Used for storing compiled native code by the **Just-In-Time (JIT) compiler**. It helps in optimizing the execution of Java bytecode by converting it into native machine code.