**Summary :**

**JVM (Java Virtual Machine**)

**Virtual Computer**: JVM is like a virtual computer inside your actual computer.

**Runs Java Code**: It runs programs written in Java by interpreting or compiling them.

**Bytecode Interpreter**: Converts Java bytecode (compiled code) into machine-specific instructions.

**Memory Management**: Handles memory allocation and garbage collection automatically.

**Platform Independence**: Allows Java programs to run on any device or operating system with a JVM.

**Execution Environment**: Provides a secure and controlled environment for Java programs.

**Optimization**: Uses techniques like Just-In-Time (JIT) compilation to improve performance.

**Debugging**: Helps in debugging by providing tools and diagnostics for Java programs.

**Standardization**: JVM specifications ensure consistency and compatibility across different implementations.

Understanding these points helps grasp the role of JVM in executing Java programs efficiently across various platforms.

### **Summary**

* **Memory Management**: JVM uses heap, stack, and method area to manage memory efficiently.
* **Garbage Collection**: Automatic process in JVM that reclaims memory from unused objects.
* **Static Aspects**: Static variables and methods are associated with the class rather than instances, and they're initialized and loaded when the class is first referenced.

### **Memory Management in JVM**

The Java Memory Allocation is divided into following sections :

1. Heap
2. Stack
3. Code
4. [Static](https://www.guru99.com/java-static-variable-methods.html)

This division of memory is required for its effective management.

* The **code** section contains your **bytecode**.
* The **Stack** section of memory contains **methods, local variables, and reference variables.**
* The **Heap** section contains **Objects** (may also contain reference variables).
* The **Static** section contains **Static data/methods**.
* **Heap Memory**: JVM manages memory in a structure called the heap. This is where objects (instances of classes) are stored.
* **Stack Memory**: Each thread in a Java application has its own stack memory, used for storing method invocations and local variables.
* **Method Area**: It stores class-level data such as method code, static variables, and constants.

### **Garbage Collection (GC)**

* **Automatic Memory Management**: JVM automatically manages memory using a process called Garbage Collection (GC).
* **Identifying Garbage**: GC periodically checks which objects are no longer referenced by any part of the program.
* **Reclaiming Memory**: When an object is identified as garbage (i.e., no longer needed), GC frees its memory for reuse.
* **Types of GC Algorithms**: Different algorithms (e.g., Mark-Sweep, Mark-Compact, Generational GC) are used based on the heap size and application characteristics.

### **Static Aspects in JVM**

* **Static Variables**: These belong to the class rather than individual objects. They're initialized once when the class is loaded into memory.
* **Static Methods**: They can be called without creating an instance of the class. They're associated with the class, not with any specific instance.
* **Class Loading**: When a Java program starts, JVM loads classes as they are referenced. This includes loading static variables and methods into memory.