

**1) ClassLoader** The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

**2) Method Area** JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

**3) Heap** All the [Objects](https://www.guru99.com/java-oops-class-objects.html), their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

**4) JVM language Stacks** Java language Stacks store local variables, and it’s partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

**5)  PC Registers** PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

**6) Native Method Stacks** Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

**7) Execution Engine** It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

**8) Native Method interface** The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

**9) Native Method Libraries** Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

**Software Code Compilation & Execution process** In order to write and execute a software program, you need the following

**1) Editor**– To type your program into, a notepad could be used for this

**2) Compiler**– To convert your high language program into native machine code

**3) Linker**– To combine different program files reference in your main program together.

**4) Loader**– To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.

**5) Execution** – Actual execution of the code which is handled by your OS & processor.



### <https://findanyanswer.com/how-do-i-increase-metaspace-size-in-java-8> [What is Max Metaspace size?](https://findanyanswer.com/open-detail/192213A4)

From our perspective, it is important to note that Metaspace has an unlimited default maximum size. On the contrary, PermGen from Java 7 and earlier has a default maximum size of **64 MB** on 32-bit JVM and **82 MB** on the 64-bit version.

### [What is Metaspace used for?](https://findanyanswer.com/open-detail/192213A5)

**Metaspace** is memory the VM **uses to** store class metadata. Class metadata are the runtime representation of java classes within a JVM process - basically any information the JVM needs to work with a Java class. That includes, but is not limited to, runtime representation of data from the JVM class file format.

### [Is Metaspace part of heap?](https://findanyanswer.com/open-detail/192213A6)

The key difference between PermGen and **Metaspace** is this: while PermGen is **part** of Java **Heap** (Maximum size configured by -Xmx option), **Metaspace** is NOT **part of Heap**. Rather **Metaspace** is **part** of Native Memory (process memory) which is only limited by the Host Operating System.

### [What is stored in Metaspace?](https://findanyanswer.com/open-detail/192213A7)

**Metaspace** contains metadata about the application the JVM is running. It contains class definitions, method definitions, and other information about the program. The more classes you load into your app, the larger **metaspace** will be.

### [What is main memory in Java?](https://findanyanswer.com/open-detail/192213A8)

Firstly, by "**main memory**" we mean 'the **Java** heap, as seen by the JVM'. The JVM is generally free to work on a local copy of a variable. For example, a JIT compiler could create code that loads the value of a **Java** variable into a register and then works on that register.

### [What is a heap space?](https://findanyanswer.com/open-detail/192213A9)

The Java **Heap Space** is the memory “container” of you runtime Java program which provides to your Java program the proper memory **spaces** it needs (Java **Heap**, Native **Heap**) and managed by the JVM itself.

### [What is perm size in Java?](https://findanyanswer.com/open-detail/192213A10)

Basically, the "permanent generation" (whose **size** is given by **PermSize**) is used to store things that the JVM has to allocate space for, but which will not (normally) be garbage-collected (hence "permanent") (+). That means for example loaded classes and static fields.

### [What is native memory in Java?](https://findanyanswer.com/open-detail/192213A11)

**Native memory** means the **memory** area outside normal JVM **heap**, but still within the total user space **memory** spared by OS for JVM process (for example on 32-bit Windows it is by default 2 GB). Direct **memory** means you use **native memory** by means of **java**. nio. DirectByteBuffer .

### [Where static variables are stored in Java?](https://findanyanswer.com/open-detail/192213A12)

The **static variables** were **stored** in the permgen space(also called the method area). The **static variables are stored** in the Heap itself. From **Java** 8 onwards the PermGen Space have been removed and new space named as MetaSpace is introduced which is not the part of Heap any more unlike the previous Permgen Space.

### [What is permanent generation in Java?](https://findanyanswer.com/open-detail/192213A13)

The **Permanent generation** contains metadata required by the JVM to describe the classes and methods used in the application. The **permanent generation** is populated by the JVM at runtime based on classes in use by the application. PermGen has been replaced with Metaspace since **Java** 8 release.

### [How does Java GC work?](https://findanyanswer.com/open-detail/192213A14)

**Java garbage collection** is the process by which **Java** programs perform automatic memory management. **Java** programs compile to bytecode that can be run on a **Java** Virtual Machine, or **JVM** for short. The garbage collector finds these unused objects and deletes them to free up memory.

### [What is string pool and heap in Java?](https://findanyanswer.com/open-detail/192213A15)

The **Java string** constant **pool** is an area in **heap** memory where **Java** stores literal **string** values. The **heap** is an area of memory used for run-time operations. When a new variable is created and given a value, **Java** checks to see if that exact value exists in the **pool**. If not, it creates a new literal **string**.

### [What is garbage collection in Java?](https://findanyanswer.com/open-detail/192213A16)

**Java Garbage Collection**. In **java**, **garbage** means unreferenced objects. **Garbage Collection** is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

### [What is string intern in Java?](https://findanyanswer.com/open-detail/192213A17)

**String Interning** is a method of storing only one copy of each distinct **String** Value, which must be immutable. **intern**() method : In **Java**, when we perform any operation using **intern**() method, it returns a canonical representation for the **string** object. A pool is managed by **String** class.

### [What is memory management in Java?](https://findanyanswer.com/open-detail/192213A18)

In **Java**, **memory management** is the process of **allocation** and de-**allocation** of objects, called **Memory management**. **Java** does **memory management** automatically. **Java** uses an automatic **memory management** system called a garbage collector.

### [How do I fix out of memory error in Java?](https://findanyanswer.com/open-detail/192213A19)

Easy way to **solve OutOfMemoryError in java** is to increase the maximum heap size by using JVM options -Xmx512M , this will immediately **solve** your **OutOfMemoryError**.

### [What is Java class metadata?](https://findanyanswer.com/open-detail/192213A20)

It's the model of the loaded **class** base that **Java** retains at runtime in order to dynamically load, link, JIT compile, and execute **Java** code. Different design choices you make when writing your code can significantly expand or contract the amount of **metadata Java** needs to retain.