**Java Virtual machine**

* JVM is a program written to understand the byte code and coverts them into machine code
* Jvm identify operating system and processor and converts the byte code into the understandable format of that particular processor and operating system
* java program on a computer using intel processor and compile it, the compiler will produce x. class containing byte code .
* These byte code are given to JVM. Now JVM understands that we are using intel processor in our system,
* so -it converts byte code into-machine code, which is understandable by the intel processor
* JVM is system dependent, since it has to interact with the processor and operating system of the computer

**Java Development kit**

.

* **JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) that includes two things
* Development Tools(to provide an environment to develop your java programs)
* JRE (to execute your java program).

**Java Run Time Environment**

* **JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine.
* The Java Runtime Environment, or JRE, is a software layer that runs on top of a computer’s operating system software and provides the class libraries and other resources that a specific [Java](https://www.ibm.com/cloud/learn/java-explained) program needs to run.
* JRE is only used by those who only want to run Java programs that are end-users of your system.
* The JRE combines Java code created using the JDK with the necessary libraries required to run it on a JVM and then creates an instance of the JVM that executes the resulting program
* JRE loads classes, verify access to memory, and retrieves the system resource

|  |  |
| --- | --- |
| **JRE** | **JVM** |
| JVM executes Java byte code and provides an environment for executing it.  JRE is also platform dependent.  It contains class libraries and other supporting files that JVM requires to execute the program  It is the subset of JDK.  The JRE is the part of Java that creates the JVM.  JRE only contain environment to execute source code. | JVM executes Java byte code and provides an environment for executing it  JVM is highly platform dependent  Software development tools are not included in JVM.  JVM is a subset of JRE  It is the Java platform component that executes source code.  JVM bundled in both software JDK and JRE. |

**Class loaders**

* Class Loader is a part of the Java Runtime Environment
* Java classes aren’t loaded into memory all at once, but when required by an application
* At this point, the Java Class Loader is called by the JRE and these Class Loaders load classes into memory
* All classes are loaded based on their names and if any of these classes are not found then it returns a **NoClassDefFoundError** or **ClassNotFoundException**.
* Not all classes are loaded by a single Class Loader. Depending on the type of class and the path of class,
* the Class Loader that loads that particular class
* . To know the Class Loader that loads a class the getClassLoader() method is used.
* It is mainly responsible for three activities.
  + Loading
  + Linking
  + Initialization

**A Java Class loader is of three types**

1. **Bootstrap Class Loader**: A Bootstrap Class loader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java Class Loader. Bootstrap Class Loader loads classes from the location jre library Bootstrap Class Loader doesn’t have any parent Class Loaders. It is also called as the Primordial Class Loader.
2. **Extension Class Loader**: The Extension Class Loader is a child of Bootstrap Class Loader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from jre/lib/ext directory
3. **System Class Loader**: An Application Class Loader is also known as a System Class Loader. It loads the Application type classes found in the environment variable CLASSPATH, The Application Class Loader is a child class of Extension Class Loader

**Memory Areas In JVM**

1. **Method Area**

* All executing threads share this part of the JVM memory area. Class elements like constant pool, class fields, constructor codes, method codes, etc.
* Method area can be considered as a part of the heap area but stores per-class data only.
* We can say that the method area is responsible for holding class level information.

**2. Heap Memory**

* Heap Memory in java is used by java runtime to allocate memory to objects and class during a java program’s execution.
* Whenever an object is created in java, it gets stored into heap memory. A garbage collection process runs on heap memory to free up unnecessary space that is garbage collection removes those objects from the heap area that does not have any references.
* Heap memory in java is divided into the following parts:

**Here are some important points regarding java heap memory**:

* If Heap space gets full, OutOfMemory error is thrown by java.
* Access to Heap memory is slow as compared to stack memory.
* Heap memory is much more in size as compared to stack memory.
* Heap memory is not thread-safe as all objects share it.
* Automatic deallocation is not present in heap memory as it needs a garbage collector to free up space.

**3. Stack Memory**

* As the name signifies, stack memory is based on LIFO (last in, first out) principle.
* Stack memory is used for static memory allocation, and each executing thread in a java program has its own stack memory.
* Whenever a Java method is called, a new block is created in java stack memory to hold local or intermediate variables and references to other objects in the method.
* As soon as the execution of the method gets completed, the block of memory in the stack becomes empty and is used by the next method.
* The size of Stack memory is less as compared to heap memory.
* **Here are some of the important features of stack memory.**
* Stack Memory grows and shrinks itself as new methods are added and removed to stack memory, respectively.
* Stack memory gets automatically allocated and deallocated after the method completes its execution.
* Access to stack memory is fast as compared to heap memory.
* Whenever stack memory gets full, an exception called stack overflow exception is thrown by java.
* Stack memory is thread-safe as each thread has its own stack memory.

**4. PC Registers**

* The main function of pc registers is to store the address of currently executing the instruction.
* It also stores the address of threads responsible for executing current instruction.
* The size of memory allocated to pc registers is very small.
* Java applications executing in JVM does not have any effect on pc register memory or its contents.

**5. Native Area**

* This area is implemented using languages other than java.
* With the creation of new threads, memory is allocated in this area for each created thread.
* The size of the native area can be fixed or dynamic.

**Constructor chaining**

* Constructors in Java are special types of methods that are used to initialize the objects of the class.
* Constructors are called at the time of object creation of the class.
* Just like methods, although they hold a set of lines of code, they are quite different from them.
* Constructors have the same name as the Java class, but it does not have any return type.
* In Java, a new() keyword to used to create an object and every time a new object is created and one constructor is called.
* The constructor is called after the memory is allocated to the object.
* Constructor Chaining is the process of calling one constructor of a class from another constructor of the same class or another class using the current object of the class.

**Constructor Chaining with this() keyword**

If we want to call the constructor from the same class, then we use this keyword.

**Constructor Chaining with super() keyword**

If we want to call the constructor from the parent class, then we use the super keyword

**Note**: In Java, it is invalid and illegal to call the constructor directly by name. We have to use either of these two keywords to call a constructor.

**Rules for Constructor Chaining in Java**

* If you want to use Constructor Chaining in Java, you must follow the below rules:
* The this() and super() statement must always be the first statement inside the constructor.
* At least one constructor should be present in the class that has no this() keyword inside it.
* We can implement the constructor chaining in any order.

**Difference Between Class And Object**

|  |  |
| --- | --- |
| **Class** | **Object** |
| Class is used as a template for declaring and  creating the objects  When a class is created, no memory is allocated.  The class has to be declared only once.  It is declared with the class keyword  A class is used to bind data as well as methods together as a single unit.  Real-Time Example  Car Blueprint | An object is an instance of a class.  Objects are allocated memory space whenever they are created  An object is created many times as per requirement.  It is created with the new keywords in Java  Objects are like a variable of the class  Real-time Example  car |

**this()**

* this is a keyword in java
* this is a hidden reference in java
* it is used by constructor (or) method
* it always holds the address of current invoking object
* if local and instance variable names are same
* local variable will initialized to itself
* this problem is solved by using this

**Abstraction & encapsulation in shopping portal**

Abstraction

* + color,size,deliverydate

Encapsulation

* + billing

**Abstraction & encapsulation in Banking**

Abstraction

* + otp, loan amount

Encapsulation

* statement,transfer

**Stack Allocation:**

* The allocation happens on contiguous blocks of memory.
* We call it a stack memory allocation because the allocation happens in the function call stack.
* The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack.
* And whenever the function call is over, the memory for the variables is de-allocated.
* This all happens using some predefined routines in the compiler. A programmer does not have to worry about memory allocation and de-allocation of stack variables.
* This kind of memory allocation also known as Temporary memory allocation because as soon as the method finishes its execution all the data belongs to that method flushes out from the stack automatically. Means, any value stored in the stack memory scheme is accessible as long as the method hasn’t completed its execution and currently in running state

**Heap Allocation:**

* The memory is allocated during the execution of instructions written by programmers.
* Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate.
* Every time when we made an object it always creates in Heap-space and the referencing information to these objects are always stored in Stack-memory.
* Heap memory allocation isn’t as safe as Stack memory allocation was because the data stored in this space is accessible or visible to all threads. If a programmer does not handle this memory well, a [memory leak](https://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/) can happen in the program