JVM (java virtual machine) in detail in java

Contents of page >

A) What is JVM (Java virtual machine) in java?

JVM is the **virtual machine** on which **java code executes**.

JVM is responsible for **converting byte code into machine specific code**.

B) HotSpot JVM (Java Virtual Machine) Architecture >

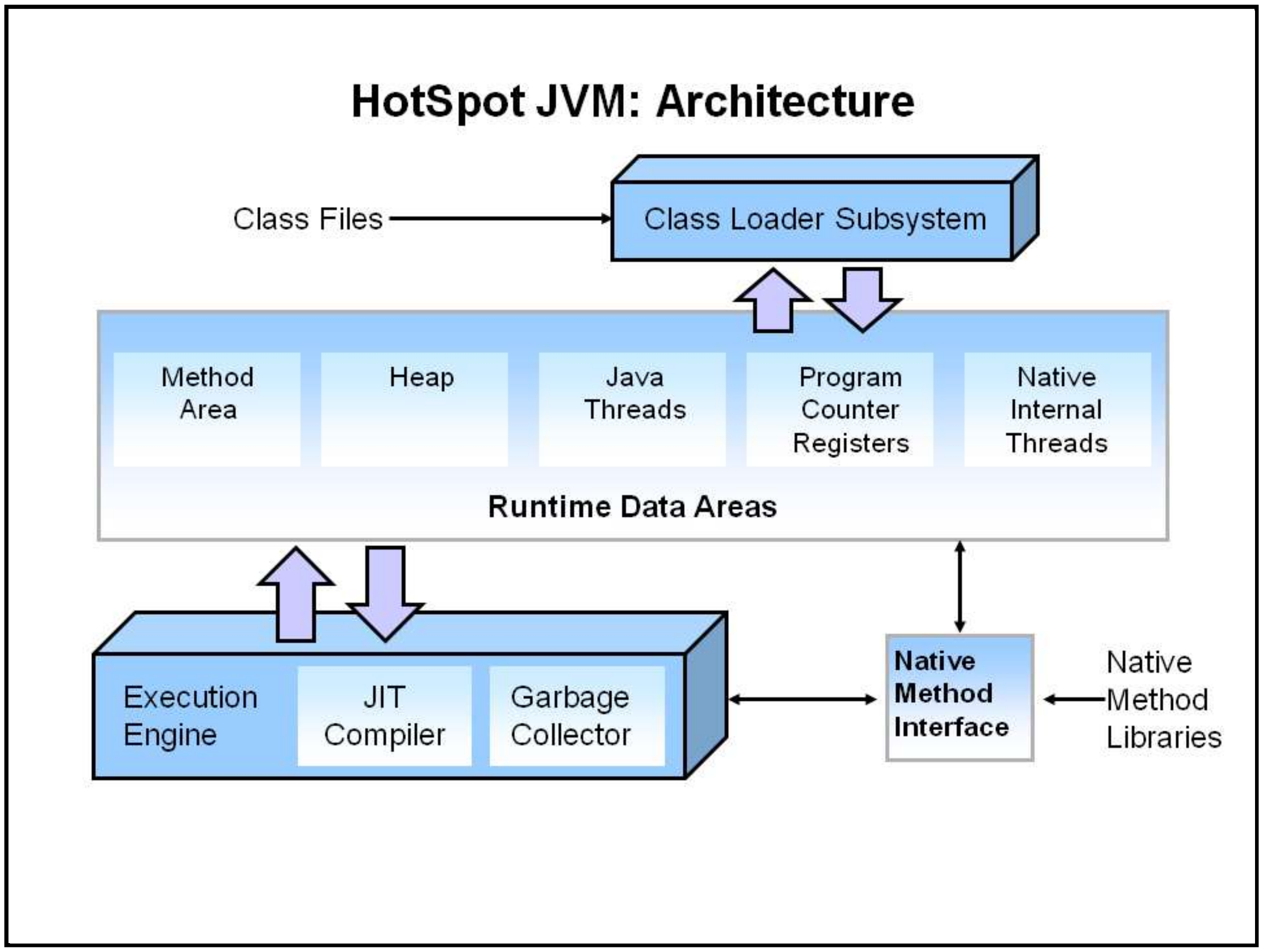

Diagram : **HotSpot JVM (Java Virtual Machine) Architecture**

Now, let's discuss each and every component of **JVM (Java Virtual Machine) Architecture** in detail.

JVM (Java Virtual Machine) **consists** of **Class Loader Subsystem**, **Runtime Data Areas** and **Execution Engine**. Let's discuss each of them in detail.

1) **Class Loader Subsystem** of JVM **>**

Classloader is a subsystem of JVM.
Classloader is used to load class files.
Classloader verifies the class file using byte code verifier. Class file will only be loaded if it is valid.

2) **Runtime Data Areas** of JVM **>**

2.1) Method Area >

Method area is also called class area.
Method area stores data for each and every class like fields,constant pool,method’s data and information.

2.2) Heap >

Heap is place where all objects are stored in JVM (java virtual machine).

Heap even contains arrays because arrays are objects.

2.3) Java Threads (Java thread Stacks) >

You must know that each and every thread has its own stack.

How stack frames are created when thread calls new method?
As we know each and every thread has its own stack. Whenever new method is called new stack frame is created and it is pushed on top of that thread's stack.

What does thread stack contains?
The stack contain

- All the local variables,
- All the parameters,
- All the return address.

Does stack stores/contains object OR what stack doesn’t contains?
Stack never stores object, but it stores object reference.

2.4) Program counter registers (PC Registers) >

Program counter registers contains >

- the address of instructions currently being executed and

- address of next instruction as well.

2.5) Native internal Threads (Native thread stack) >

Native internal threads area contains all the informations related to native platform.
Example - If we are running JVM (java application) on windows, it will contain all information related to native platform i.e. windows.
If we are running JVM (java application) on linux, it will contain all information related to native platform i.e. linux.

3) **Execution Engine** of JVM **>**

Execution Engine contains **JIT (Just In Time) Compiler** and **Garbage collector** compiler. Execution Engine also contains Interpreter.

3.1) JIT(Just In Time) compiler >

JIT compiler *compiles bytecodes to machine code at run time and* improves the performance of Java applications.

JIT Compiler internal working >

JIT compilation does require processor time and memory usage. When the JVM first starts up, lots of methods are called. Compiling all of these methods might can affect startup time significantly, though program ultimately may achieve good performance.

Methods are not compiled when they are called first time. For each and every method JVM maintains a **call count**, which is incremented every time the method is called. The methods are interpreted by JVM until call count not exceeds **JIT compilation threshold (**The JIT compilation threshold improves performance and helps the JVM to start quickly. The threshold has been selected carefully by java developers to obtain an optimal performances. Balance between startup times and long term performance is maintained**).**

Therefore, very frequently used methods are compiled as soon as JVM has started, and less used methods are compiled later.

How JIT improves performance of Most frequently used methods ?

After a method is compiled, its call **count** is **reset to zero** and **subsequent calls** to the method **increment** it **call count**. When the **call count of a method reaches** a **JIT recompilation threshold**, the **JIT compiler compiles method second time**, applying more optimizations as compared to optimizations applied in previous compilation. This process is repeated until the maximum optimization level is reached. Most frequently used methods are always optimized to maximize the performance benefits of using the JIT compiler.

Example -

Let’s say **JIT recompilation threshold = 2**

After a method is compiled, its call **count** is **reset to zero** and **subsequent calls** to the method **increment** it **call count.** When the **call count of a method reaches** a 2 (i.e. **JIT recompilation threshold)**, the **JIT compiler compiles method second time**, applying more optimizations as compared to optimizations applied in previous compilation.

3.2) Garbage Collector

Garbage Collector Garbage collection is the process by which JVM clears objects (unused objects) from heap to reclaim heap space.

Interpreter > Interpreter is responsible for reading the bytecode and then executing the instructions.

Native method libraries of JVM **>**

Native method interface is an interface that connects JVM with the native method libraries for executing native methods.

Example of Native method libraries>

If we are running JVM (java application) on windows, then Native method interface(window method interface) will connect JVM with the window methods libraries(native method libraries) for executing window methods (native methods).

You must know about **JNI**, What is **Java Native Interface**(JNI)?

You may write your application purely in java but there are certain situations where java code might need meet your requirement.
Programmers uses the JNI (Java Native Interface) to write the Java native methods when an application cannot be written purely in Java.