# Java Performance Tuning

Note that JRockit is the default JVM that comes with Oracle Weblogic Server and Hotspot is the default one for Java JDK/JRE

The Hotspot JVM is better suited for UI desktop applications whereas JRockit is made for best performance and fast executions.

**JIT Compiler**

This name (HotSpot) comes from the approach it takes toward compiling the code. In a typical program, only a small subset of code is executed frequently, and the performance of an application depends primarily on how fast those sections of code are executed. These critical sections are known as the hotspots of the application; the more the section of code is executed, the hotter that section is said to be.

When the JVM executes code, it does not begin compiling the code immediately. There are two basic reasons for this. First, if the code is going to be executed only once, then compiling it is essentially a wasted effort; it will be faster to interpret the Java bytecodes than to compile them and execute (only once) the compiled code.

But if the code in question is a frequently called method, or a loop that runs more iteration, then compiling it is worthwhile—the cycles it takes to compile the code will be outweighed by the savings in multiple executions of the faster compiled code. That trade-off is one reason that the compiler executes the interpreted code first—the compiler can figure out which methods are called frequently enough to warrant their compilation.

Java is both **interpreter** language as well as **compiler** language.

1. Java is designed to take advantage of the platform-independence of scripting languages and the native performance of compiled languages.
2. A Java class file is compiled into an intermediate language (Java bytecodes) that is then further compiled into assembly language by the JVM.
3. Compilation of the byte codes into assembly language performs a number of optimizations that greatly improve performance.

(The JIT Compiler does this)

**What is tiered compilation?**

JIT compiler comes in two flavours.

i) Client

ii) Server

Default setting is both modes enabled that is called tiered compilation.

With tiered compilation, the client compiler first compiles code. As it becomes hot, the server compiler recompiles it.

IMPORTANT THINGS TO REMEMBER

1. The client compiler is most useful when the startup of an application is the overriding performance concern.
2. Tiered compilation can achieve start up times very close to those obtained from the client compiler.

**Understanding the Code Cache**

When the JVM compiles code, it holds the set of assembly-language instructions in the code cache. The code cache has a fixed size, and once it has filled up, the JVM is not able to compile any additional code.

When the code cache fills up, the JVM will (usually) spit out a warning to that effect:

Java HotSpot(TM) 64-Bit Server VM warning: CodeCache is full. Compiler has been disabled. Java HotSpot(TM) 64-Bit Server VM

Warning: Try increasing the code cache size using

-XX: ReservedCodeCacheSize=

Default value of code cache in 64-bit server with Tiered Compilation in Java 8 is 240MB

**NOTE: The size of the code cache can be monitored using jconsole by selecting the “Memory Pool Code Cache” chart on the “Memory” panel.**

# Garbage Collectors

There are four main garbage collectors available in the JVMs

1. Serial Collector
2. Througput Collector
3. CMS Collector
4. G1 Collector

Performance of garbage collection is dominated by these basic operations:

* Finding unused objects
* Making their memory available
* Compacting the heap