**Typcial scenario:**

The source code is completely converted into machine code

**JIT scenario:**

The source code will be converted into assmebly language like structure [for ex IL(intermediate language) for C#,ByteCode for java].

The intermediate code is converted into machine language only when the application needs that is required codes are only converted to machine code

**JIT vs Non-JIT comparison:**

* In JIT not all the code is converted into machine code first a part of the code that is neccessary will be converted into machine code then if a method or functionality called is not in machine then that will be turned into machine code...it reduces burden on cpu
* As the machine code will be generated on run time....the JIT compiler will produce machine code that is optimised for running machine's cpu architecture.

**JIT Examples:**

1. In Java JIT is JVM(java virtual machine)
2. In C# it is in dot net framework
3. In Android DVM(Dalvik virtual machine)

# What is the difference between JRE,JVM and JDK?

# If you are a Java developer, it is very often that you think about understanding the JRE,JVM and JDK. Once if you understand these things, it would be quite easy for you to visualize things in logical manner. Also look at the picture below, that will clear all your questions about the JRE,JVM and JDK. This article explains about the each term and will make you understand perfectly. Hope this helps. If you are happy to receive our future updates on Java, please subscribe [here](http://www.javabeat.net/subscribe/).

## JDK (Java Development Kit)

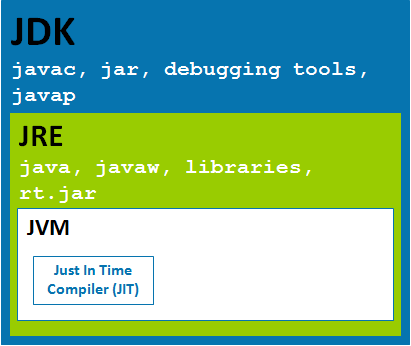
Java Developer Kit contains tools needed to develop the Java programs, and ***JRE*** to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Appletviewer, etc… Compiler converts java code into byte code. Java application launcher opens a ***JRE***, loads the class, and invokes its main method.

You need ***JDK***, if at all you want to write your own programs, and to compile the m. For running java programs, JRE is sufficient. JRE is targeted for execution of Java files i.e. **JRE** = **JVM** + Java Packages classes (like util, math, lang, awt,swing etc)+runtime libraries. ***JDK*** is mainly targeted  for java development. I.e. You can create a Java file (with the help of Java packages), compile a Java file and run a java file.

* [Java Annotations](http://www.javabeat.net/2007/08/annotations-in-java-5-0/)
* [Java Generics](http://www.javabeat.net/2007/08/generics-in-java-5-0/)

## JRE (Java Runtime Environment)

Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc. Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any java program, you need to have JRE installed in the system

[](http://www.javabeat.net/wp-content/uploads/2013/02/jvm-jre-jdk.png)The ***Java Virtual Machine*** provides a platform-independent way of executing code; programmers can concentrate on writing software, without having to be concerned with how or where it will run. **But, note that JVM itself not a platform independent . It only helps Java to be executed on the platform-independent way. When JVM has to interpret the byte codes to machine language, then it has to use some native or operating system specific language to interact with the system. One has to be very clear on platform independent concept. Even there are many JVMs written on Java, however hey too have little bit of code specific to the operating systems**

If u just want to run applets (ex: Online Yahoo games or puzzles), *JRE* needs to be installed on the machine.

## JVM (Java Virtual Machine)

As we all aware when we compile a ***Java file***, output is not an ‘exe’ but it’s a ‘.class’ file. ‘.class’ file consists of ***Java byte codes***which are understandable by JVM. Java Virtual Machine interprets the byte code into the machine code depending upon the underlying operating system and hardware combination. It is responsible for all the things like garbage collection, array bounds checking, etc… JVM is platform dependent.

The ***JVM*** is called “virtual” because it provides a machine interface that does not depend on the underlying operating system and machine hardware architecture. This independence from hardware and operating system is a cornerstone of the write-once run-anywhere value of Java programs.

There are different JVM implementations are there. These may differ in things like performance, reliability, speed, etc. These implementations will differ in those areas where Java specification doesn’t mention how to implement the features, like how the garbage collection process works is JVM dependent, Java spec doesn’t define any specific way to do this.