JVM JDK JRE

What is Java Virtual Machine (JVM)?

Java Virtual Machine, or JVM, loads, verifies and executes Java bytecode. It is known as the interpreter or the core of Java programming language because it executes Java programming.

The role of JVM in Java

JVM is specifically responsible for converting bytecode to machine-specific code and is necessary in both JDK and JRE. It is also platform-dependent and performs many functions, including memory management and security. In addition, JVM can run programs written in other programming languages that have been translated to Java bytecode.

Java Native Interface (JNI) is often referred to in connection with JVM. JNI is a programming framework that enables Java code running in JVM to communicate with (i.e., to call and be called by) applications associated with a piece of hardware and specific operating system platform. These applications are called native applications and can often be written in other languages. Native methods are used to move native code written in other languages into a Java application.

JVM components

JVM consists of three main components or subsystems:

- **Class Loader Subsystem** is responsible for loading, linking and initializing a Java class file (i.e., “Java file”), otherwise known as dynamic class loading.
- **Runtime Data Areas** contain method areas, PC registers, stack areas and threads.
- **Execution Engine** contains an interpreter, compiler and garbage collection area.

What is Java Runtime Environment (JRE)?

Java Runtime Environment, or JRE, is a set of software tools responsible for execution of the Java program or application on your system.

JRE uses heap space for dynamic memory allocation for Java objects. JRE is also used in JDB (Java Debugging).

The role of JRE in Java

If a programmer would like to execute a Java program using the Java command, they should install JRE. If they are only installing (and not developing or compiling code), then only JRE is needed.

JRE components

Besides the Java Virtual Machine, JRE is composed of a variety of other supporting software tools and features to get the most out of your Java applications.

- **Deployment solutions:** Included as part of the JRE installation are deployment technologies like Java Web Start and Java Plugin that simplify the activation of applications and provide advanced support for future Java updates.
- **Development toolkits:** JRE also contains development tools designed to help developers improve their user interface. Some of these toolkits include the following:
    - **Java 2D:** An application programming interface (API) used for drawing two-dimensional graphics in Java language. Developers can create rich user interfaces, special effects, games and animations.
    - **Abstract Window Toolkit (AWT):** A Graphical User Interface (GUI) used to create objects, buttons, scroll bars and windows.
    - **Swing:** Another lightweight GUI that uses a rich set of widgets to offer flexible, user-friendly customizations.
- **Integration libraries:** Java Runtime Environment provides several integration libraries and class libraries to assist developers in creating seamless data connections between their applications and services. Some of these libraries include the following:
    - **Java IDL:** Uses Common Object Request Brokerage Architecture (CORBA) to support distributed objects written in Java programming language.

    - **Java Database Connectivity (JDBC) API:** Provides tools for developers to write applications with access to remote relationship databases, flat files and spreadsheets.
    - **Java Naming and Directory Interface (JNDI):** A programming interface and directory service that lets clients create portable applications that can fetch information from databases using naming conventions.
- **Language and utility libraries:** Included with JRE are Java.lang. and Java.util. packages that are fundamental for the design of Java applications, package versioning, management and monitoring. Some of these packages include the following
    - **Collections framework:** A unified architecture made up of a collection of interfaces designed to improve the storage and process of application data.
    - **Concurrency utilities:** A powerful framework package with high-performance threading utilities.
    - **Preferences API:** A lightweight, cross-platform persistent API that enables multiple users on the same machine to define their own group of application preferences.
    - **Logging:** Produces log reports — such as security failures, configuration errors, and performance issues — for further analysis.
    - **Java Archive (JAR):** A platform-independent file format that enables multiple files to be bundled in JAR file format, significantly improving download speed and reducing file size.

What is Java Development Kit (JDK)?

Java Development Kit, or JDK, is a software development kit often called a superset of JRE. It is the foundational component that enables Java application and Java applet development. It is platform-specific, so separate installers are needed for each operating system (e.g., Mac, Unix and Windows).

The role of JDK in Java

JDK contains all the tools required to compile, debug and run a program developed using the Java platform. (It’s worth noting that Java programs can also be run using command line.)

JDK components

JDK includes all the Java tools, executables and binaries needed to run Java programs. This includes JRE, a compiler, a debugger, an archiver and other tools that are used in Java development.

Java SE vs. Java EE

Java is synonymous with Java Standard Edition (Java SE) or Core Java. All three euphemisms refer to the basic Java specification that includes the act of defining types and objects. Java EE, on the other hand, provides APIs and is typically used to run larger applications. The content of this blog focuses on Java SE.

How JVM, JRE and JDK work together

The diagram provides an image of how JVM, JRE and JDK fit together in the Java landscape.

JVM, JRE and JDK all have very specific functions. Without all three, Java will not operate successfuly.

JDK vs. JRE vs. JVM: Key differences

And now, for the differences:

- JDK is the development platform, while JRE is for execution.
- JVM is the foundation, or the heart of Java programming language, and ensures the program’s Java source code will be platform-agnostic.
- JVM is included in both JDK and JRE – Java programs won’t run without it.

Complementary technologies

There are many complementary technologies that can be used to enhance JVM, JRE or JDK. The following technologies are among the most frequently used:

- **Just-in-time Compiler (JIT)** is part of JVM and optimizes the conversion of bytecode to machine code. It selects similar bytecodes to compile at the same time, reducing the overall duration of bytecode to machine code compilation.
- **Javac**, another complementary tool, is a compiler that reads Java definitions and translates them into bytecode that can run on JVM.
- **Javadoc** converts API documentation from Java source code to HTML. This is useful when creating standard documentation in HTML.

JVM and container technology

Java Virtual Machine (JVM) is used to create — you guessed it — virtual machines (VMs). VMs are servers that allow multiple applications to run on the same underlying physical hardware without impacting one another. This provides better use of resources and makes it much easier and cost-effective to scale than traditional infrastructure. VMs are also easily disposable because of their independence. When you no longer need the application, you simply take the VM down.

Containers take this abstraction to the next level and virtualize the OS kernel. The absence of the OS renders containers even more lightweight, fast and flexible than VMs.