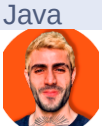
JDK vs JRE vs JVM in Java: Key Differences Explained

Updated on February 19, 2025

Java



By Anish Singh Walia

Sr Technical Writer

Introduction

Java is a powerful, platform-independent programming language widely used for developing applications. Understanding the differences between JDK, JRE, and JVM is crucial for Java developers. This article explains these three components in-depth, providing practical use cases and common debugging solutions.

The difference between JDK, JRE, and JVM is one of the popular interview questions. You might also be asked to explain JDK vs JRE vs JVM.

Understanding JDK, JRE, and JVM

Java Development Kit (JDK)

The Java Development Kit (JDK) is a software development kit that provides a set of tools and libraries for developing Java applications. It includes the Java Runtime Environment (JRE), which is necessary for running Java programs, as well as development tools such as compilers and debuggers. The JDK is essential for developers who want to write, compile, and run Java code.
Use Case: Install the JDK when developing Java applications, compiling Java code, or using Java frameworks like Spring Boot. For more information on setting up a Java development environment, refer to our tutorial on How to Install Java With Apt.

Java Runtime Environment (JRE)

The Java Runtime Environment (JRE) is a package that provides the libraries and JVM required to run Java applications. It does not include development tools, making it suitable for users who only need to run Java programs without developing them. The JRE is a subset of the JDK and is included in the JDK installation.

Use Case: Install the JRE when you only need to run Java applications but not develop them. This is typically the case for end-users who want to run Java-based software without modifying the code.

Java Virtual Machine (JVM)

The Java Virtual Machine (JVM) is a crucial component of the JRE that provides the runtime environment for Java programs. It is responsible for executing Java bytecode on any platform that supports JVM, making Java a “write once, run anywhere” language. The JVM is included in the JRE and is not installed separately.

Use Case: The JVM is automatically included with the JRE installation, so there is no need to install it separately.

Difference between JDK, JRE, and JVM

JDK, JRE, and JVM are core concepts of the Java programming language. We don’t use these concepts in programming. But, as a Java developer, we should know about them.

1. JDK

Java Development Kit aka JDK is the core component of Java Environment and provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program. JDK is a platform-specific software and that’s why we have separate installers for Windows, Mac, and Unix systems. We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes.

2. JVM

JVM is the heart of the Java programming language. When we execute a Java program, JVM is responsible for converting the byte code to the machine-specific code. JVM is also platform-dependent and provides core java functions such as memory management, garbage collection, security, etc. JVM is customizable and we can use java options to customize it. For example, allocating minimum and maximum memory to

JVM. JVM is called **virtual** because it provides an interface that does not depend on the underlying operating system and machine hardware. This independence from hardware and the operating system makes the Java programs write-once-run-anywhere.

3. JRE

JRE is the implementation of JVM. It provides a platform to execute java programs. JRE consists of JVM, Java binaries, and other classes to execute any program successfully. JRE doesn't contain any development tools such as Java compiler, debugger, JShell, etc. If you just want to execute a java program, you can install only JRE. You don't need JDK because there is no development or compilation of Java source code. Now that we have a basic understanding of JDK, JVM, and JRE, let's examine the difference between them.

Comparison Table

Component	Description	Includes	Platform-Dependent	Use Case	Installation Requirements	Develop Too
JDK	Java Development Kit	JRE, Java compiler, debugger, core classes	Yes	For development, compilation, and execution of Java programs	Separate installers for Windows, Mac, and Unix	Java com debugge JShell
JVM	Java Virtual Machine	Converts byte code to machine-specific code	Yes	Automatically included with JRE installation	Included with JRE installation	None
JRE	Java Runtime Environment	JVM, Java binaries, other classes	Yes	For executing Java programs without development or compilation	Separate installers for Windows, Mac, and Unix	None

Practical Use Cases

When to Install JDK vs JRE

If you are a **developer**, you will need to install the JDK. The JDK includes the JRE, but also provides additional tools such as the Java compiler and debugger. These tools are essential for compiling and debugging Java code.

If you only need to run Java applications, you can install the JRE. The JRE is a subset of the JDK and includes the JVM, which is necessary for running Java programs. However, it does not include the development tools provided by the JDK.

Checking Your Java Version and Environment Setup

To check your installed Java version, run:

```
java -version
javac -version  # For JDK verification
```

Copy

Ensure the correct Java version is installed and configured properly.

Understanding JVM Configurations

JVM settings can impact application performance. Adjusting the heap size helps optimize memory management:

```
java -Xms512m -Xmx1024m MyApp
```

Copy

- `-Xms` sets the initial heap size.
- `-Xmx` sets the maximum heap size.

In-depth JVM Breakdown

Just-In-Time (JIT) Compilation

The Just-In-Time (JIT) compiler is a crucial component of the JVM that significantly enhances Java performance. It dynamically converts Java bytecode into native machine code at runtime, allowing the JVM to execute Java programs more efficiently. This process occurs transparently, without the need for manual compilation or intervention. By compiling frequently executed code paths into native code, the JIT compiler reduces the overhead of interpretation and improves the overall execution speed of Java applications.

Garbage Collection

The JVM employs garbage collection to manage memory automatically, ensuring that memory is released from objects no longer in use and mitigating the risk of memory leaks. This process involves identifying objects that are no longer referenced or needed by the application and reclaiming their memory space. By doing so, garbage collection helps maintain the integrity of the system by preventing memory exhaustion and promoting efficient memory utilization.

Common Errors and Debugging

“Java Not Recognized” Error

This error often occurs due to an incorrect installation or a missing PATH variable.

Solution:

- Ensure Java is installed correctly.
- Add Java to the system PATH:

```
export PATH=$PATH:/path/to/java/bin  # Linux/macOS
setx PATH "%PATH%;C:\Path\To\Java\bin"  # Windows
```

Copy

Mismatch Between JDK and JVM Versions

A version mismatch can cause runtime errors.

Solution:

Check both versions:

```
java -version
javac -version
```

Copy

Ensure they match and update Java accordingly.

FAQs

1. What is JDK, JRE, JVM, and JIT in Java?

- JDK (Java Development Kit) includes the tools needed for Java development.
- JRE (Java Runtime Environment) provides libraries and JVM for running Java applications.
- JVM (Java Virtual Machine) executes Java bytecode.
- JIT (Just-In-Time compiler) improves execution speed by compiling bytecode to native machine code at runtime.

Component	Description	Includes	Platform-Dependent	Use Case	Installation Requirements	Develop Too
JDK	Java Development Kit	JRE, Java compiler, debugger, core classes	Yes	For development, compilation, and execution of Java programs	Separate installers for Windows, Mac, and Unix	Java cor debugge JShell
JVM	Java Virtual Machine	Converts byte code to machine-specific code	Yes	Automatically included with JRE installation	Included with JRE installation	None
JRE	Java Runtime	JVM, Java	Yes	For executing	Separate	None

Component	Description	Includes	Platform-Dependent	Use Case	Installation Requirements	Develop Too
	Environment	binaries, other classes		Java programs without development or compilation	installers for Windows, Mac, and Unix	
JIT	Just-In-Time compiler	Improves execution speed by compiling bytecode to native machine code at runtime	Yes	Automatically included with JVM	Included with JVM	None

2. What is the role of JDK in Java programming?

JDK provides the necessary tools for writing, compiling, and debugging Java applications.

3. Do I need JDK or JRE to run Java applications?

If you only need to run Java applications, JRE is sufficient. If you need to develop Java applications, you need JDK.

4. Why do I have JRE instead of JDK?

If you installed Java for running applications rather than development, JRE was likely installed instead of JDK.

5. What is the difference between JDK, JRE, and JVM?

Component	Includes	Description
JDK	JRE, Development Tools	Java Development Kit, includes tools for development, compilation, and execution of Java programs

Component	Includes	Description
JRE	JVM, Runtime Libraries	Java Runtime Environment, provides libraries and JVM for running Java applications
JVM	-	Java Virtual Machine, executes Java bytecode

6. What is the difference between JVM and JIT?

Component	Description	Functionality
JVM	Java Virtual Machine	Executes Java bytecode
JIT	Just-In-Time compiler	Compiles bytecode into machine code for better performance

JVM executes Java programs, while JIT compiles bytecode into machine code for better performance.

Conclusion

In conclusion, understanding the differences between JDK, JRE, and JVM is crucial for any Java developer. The JDK provides the tools needed for Java development, the JRE includes the JVM and libraries for running Java applications, and the JVM executes Java bytecode. The JIT compiler, included with the JVM, enhances execution speed by compiling bytecode to native machine code at runtime. By recognizing the roles of each component, developers can effectively choose the right tools for their projects and ensure efficient execution of their Java applications.

For more information on installing Java, refer to How to Install Java with apt.
To dive deeper into Java's memory management, check out Java JVM Memory Model and Memory Management in Java.
Additionally, explore Getting Started with PyPy for an alternative to traditional Python execution.

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

Learn more about our products