**A Guide to Knowing About JDK, JRE, and JVM**

Java is a high-level object-oriented programming language that has immensely made a soft corner in the hearts of millions of software developers. This programming language came into existence in the YEAR 1995 and is owned by Oracle. Who created this masterpiece? A Canadian computer scientist named James Gosling is the creator of the Java programming language. Another question that generally haunts beginner software developers is: Is Java tough to learn? The answer in one word is no! Java is easy to learn and faster to debug, and along with all these features, the programming language has fewer implementation dependencies.

Whether you are a professional Java developer or willing to step into the programming or coding world,  this guide will provide you with all the necessary ingredients that will make you a hardcore Java fan. So, without delaying much, let’s deep dive and understand some basic terminologies and important differences that you would encounter while learning Java or going to crack the Java Developer Interview.

**JDK:**

The acronym for JDK is Java Development Kit. As the name suggests, KIT, or JDK, is the prime component that has all the necessary ingredients and all the important tools required to compile and run or execute the Java program. In simple words, the Java Development Kit (JDK) offers a space that provides a facility for software developers to develop and run Java programs.

Now, is JDK a superset of JRE? What does this mean? Let’s step in and explore the meaning:

As defined above, JDK is a KIT that holds **JRE** (Java Runtime Environment) within itself along with other software development tools such as Compiler, Interpreter, and Document Generator.

**Role of JDK:**

JDK is a superset that includes JRE and JVM. It has all the important elements or tools that are needed to compile, debug, and execute Java code through the Java platform.

**Components of JDK:**

Five key components of the JDK are enlisted below:

1. **Java Compiler:** Java Development Kit (JDK) has one of the most important elements, i.e., the Java Compiler. t converts Java source code into bytecode; later, the converted bytecode is run into the JVM (Java Virtual Machine) for further processing.
2. **Java Virtual Machine (JVM):** JVM is an integral part of Compiler that interprets and executes Java bytecode and provides a runtime space for Java apps.
3. **Java Runtime Environment (JRE):** Java Runtime Environment is a subset of JDK that has a JVM (Java Virtual Machine) required to execute Java applications.
4. **Java API (Application Programming Interface):** JDK contains in-built libraries and classes that help develop Java APIs, and these APIs mainly provide functionalities to execute various tasks, such as file handling, networking, etc., so that software developers can easily and quickly build Java applications.
5. **Development Tools:** The JDK has numerous development tools that help Java developers in software development and debugging,  like Javadoc for HTML documentation and JDB, which acts as a debugger for Java apps.

**JRE:**The acronym for JVM is Java Runtime Environment. As the name suggests, Environment, JRE came into existence to provide an environment to execute Java programs. Is JRE platform-dependent? What are its compositions?

The response to this specific question is yes! JRE, aka Java Runtime Environment, is platform-dependent, just like JDK. Now, coming to the part of its compositions, JRE has JVM (Java Virtual Machine) and other related important classes.

**Role of JRE:**

In simple words, JRE is utilized to execute Java code by typing some Java commands. If a SOFTWARE  Developer decides to run a Java program then he or she must install JRE (Java Runtime Environment).

**Components of the Java Runtime Environment (JRE):**

The Java Runtime Environment (JRE) consists of several components that are essential for executing Java applications. Here are some significant components of JRE, shared below:

1. Java Virtual Machine (JVM): The JVM interprets Java bytecode and translates it into native machine centric code.
2. Class Loader: Class loaders majorly load the Java classes into the JVM. Class loaders load the files from the file system and make the files ready to run.
3. Execution Engine: The execution engine runs the Java bytecode. It has an interpreter and Just-in-Time (JIT) compiler, which change bytecode into native machine code.
4. Java Native Interface (JNI): JNI is a programming framework that allows Java code to be called by native application code written in C or C++. Through the Java Native Interface (JNI), it becomes easier to interface Java code with native platform libraries.
5. Java API (Application Programming Interface): The Java API has a wide variety of pre-built classes, offering loads of functionalities. By using all these classes and interfaces, the software developers are able to effectively create Java apps leveraging current implementations.
6. Java Class Library: The Java Class Library is a comprehensive collection of pre-compiled classes and methods organized into packages. These packages provide standard functionality for various tasks, including I/O operations, networking, and more.
7. Java Tools and Utilities: JRE offers many Java tools and utilities, like Java Compiler, Java Debugger, and many more. All these listed tools support the software development, compilation, and debugging of Java apps.
8. Java Native Interface (JNI): JNI allows Java code to interact with native code and provides a method to call native methods from Java and vice versa.

**JVM:**

The abbreviation for JVM is Java Virtual Machine. Why is it required? Why is it important? Explaining by saying that JVM is inbuilt or can be said to be already present in both the JDK and JRE. Whenever any software developer writes Java code and moves to run Java programs, then the developer uses JDK or JRE; afterwards, JVM is the prime entity to run the written code line-to-line, also termed by many developers as an interpreter.

**Role of the JVM in Java**

JVM has already been discussed in the blog post, but what is the prime role of JVM in Java? So, coming to this point, JVM is nothing but the soul of Java programming. What’s the reason behind calling it SOUL of JAVA? JVM is primarily responsible for translating or converting bytes into machine codes. Being a platform, Java Virtual Machine (JVM) facilitates software developers with multiple ultimate Java functions or methods like memory management and garbage storage features. The other important property of JVM is that it acts exceptionally well and runs the code of other programming languages after getting converted to Java bytecode.

Another question that arises while learning about JVM is virtual. By virtual, it simply means that JVM does not depend on any specific operating system or hardware to run Java programs.

**JVM Component**

Here are five essential components of the JVM:

1. Class Loader: The prime role of Class Loader is to load the Java classes into Memory. It also loads the class files of the required classes into the JVM.
2. Execution Engine: The execution engine mainly interprets the Java byte code line by line and runs the native machine code. Just-in-Time (JIT) compilation can also be used to convert bytecode into native machine code for better performance.
3. Garbage Collector (GC): It is a memory management component that automatically de-allocates memory occupied by objects of no use. Through garbage collectors, memory gets managed easily and is also protected from memory leakage.
4. Runtime Data Areas: Java Virtual Machines (JVM)) are further classified into different runtime data areas, enlisted below:

* Method Area: Stores class metadata, static fields, etc.
* Heap: It is an area where objects and their instance variables are kept.
* PC Register: The address of the JVM instruction is stored in this register.
* Native Method Stack: For executing native methods.

1. JIT Compiler (Just-In-Time Compiler): The JIT compiler translates bytecode into machine-specific code. JIT further determines the run source code and compiles it to native code to speed up the execution speed of the Java application.

**Java SE and Java EE**

In the Java programming language, there are two editions: Java SE (standard edition) and Java EE (enterprise edition). Both of the discussed editions have their own roles, and they are designed for different kinds of applications. Explore both editions to grab more information:

**What is Java SE (Standard Edition)?**

Java SE, aka Java Standard Edition, facilitates the core functionality of the Java programming language. It mainly acts as the base that helps to build all other Java editions, and that too exceptionally. Now, moving to its application part, Java SE can be beneficial or the best fit for creating desktop or server apps.

The technologies that are broadly used in Java SE are Core libraries, JavaFX, and JDBC. Along with these features, Java SE also provides the facility of “Write Once, Run Anywhere”, by using the JVM (Java Virtual Machine).

**What is Java EE (Enterprise Edition)?**

Java EE, aka Java Enterprise Edition, facilitates APIs and offers a runtime space ideal for creating, developing, and executing large-scale applications. Java EE has all the components that Java SE has, along with some other APIs that are required to build multi-tiered and enterprise-level applications.

Now, coming to the application part, Java EE is the best fit for complex enterprise-level apps, distributed systems, etc. Furthermore, the technologies used under Java EE are Servlets, JSP, JMS, JPA, etc.

**Differences between Java SE and Java EE**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Particulars** | **Java SE** | **Java EE** |
| **1** | **Definition** | Java SE, aka Java Standard Edition, is equipped with core functionalities. | Java RR (Enterprise Edition) is equipped with extended functionalities. |
| **2** | **Key Ingredients** | Core language, normal APIs | Core language, enterprise APIs |
| **3** | **Environment** | It has class libraries and virtual machines. | It has separate client, business, and enterprise layers. |
| **4** | **Application** | Used to create APIs for desktop apps | Primarily used for web apps |
| **5** | **Use Cases** | Desktop Applications | Complicated enterprise-level applications |
| **6** | **Technologies Included** | JavaFX, JDBC | Servlets, JMS, and JSP |

**How do JDK, JRE, and JVM relate together? How do all three work together?**

Without a doubt, Java is one of the oldest programming languages in the world. As in the blog post, it’s already been discussed that the Java programming language is widely used by software developers to develop robust and powerful mobile and web applications (an application stored on the server, later delivered on the internet, that runs on a browser).

Now, coming to the topic of the relationship among JDK, JRE, and JVM,

In the Java programming world: JDK, JRE, and JVM are three very important and integral part that play significant roles in developing robust Java applications. The JDK is a comprehensive software package that provides utilities that is essential & required for developing, compiling, and debugging Java applications. The JDK consists of the compiler, debugger, and other tools needed for software development. The Java Runtime Environment (JRE) is a runtime environment to run Java applications. JRE consists of the JVM and other sets of libraries required for running Java programs. JVM is platform-dependent and provides a space for Java applications across various operating systems.

In a nutshell, the JDK supports the development of Java applications, the JRE executes the Java code, and the last one, the JVM, acts as a bridge that fills the gap between the two, ensuring high performance and reliability.

**Features of JDK, JRE, and JVM**

**JDK:**

1. It provides a space for writing and executing Java code.
2. It offers the complete functionalities of Hava Virtual Machine and Java Runtime Environment.
3. One of the most important features of JDK is that it supports the Java developer in managing exceptions effectively by integrating extensions into one catch block.
4. JDK consists of a debugger, a, compiler, and many other essentials tools for software development.

**JRE:**

1. The Java Runtime Environment has a cluster of essential utilities that support JVM.
2. JRE offers several significant tools for deployment, like the. Java plug-in.  2. Java Web Start.
3. Through JRE, Java developers execute the Java code.
4. JRTE has important libraries. Some of the important libraries are: JDBC (Java Database Connectivity), JNDI (Java Naming and Directory Interface), etc.

**JVM:**

1. JVM is the prime actor that translates byte code into machine-specific code.
2. JVM offers significant functionalities like memory management and garbage collection (GC).
3. JVM uses libraries and files from JRE to run the Java program.
4. JVM runs the Java source code line-by-line (acts as an interpreter).

**Major Difference Between JDK, JRE, and JVM**

|  |  |  |
| --- | --- | --- |
| **JDK** | **JRE** | **JVM** |
| Java Development Kit. | Java Runtime Environment. | Java Virtual Machine. |
| A software development kit, that helps the developers develop Java Applications. | A software package that consists: Java Virtual Machine (JVM), Class libraries etc. | A machine that provides an environment to run Java Bytecode. |
| Has utilities to Create, Monitor and Debug source code. | Has class libraries to run Java code. | Has no tools for Software Development. |
| It is platform dependent. | It is platform dependent. | It is not platform dependent. |
| Composition of JRE & Development Tools | Combination of Class libraries & JVM | Offers a Runtime space. |

**Question & Answer:**

Q1: Briefly explain the Java Development Kit (JDK).

Ans: The Java Development Kit (JDK) is a software package provided by the renowned organization Oracle that has all the important utilities required to build robust and powerful mobile and web applications. Along with development, the Java developers can compile and execute the Java program.

Q2: Explain the special components of the Java Development Kit (JDK).

Ans: The Java Development Kit (JDK) has numerous special components, like the Java Runtime Environment (JRE), compiler, debugging tools, libraries, and many other components required to develop Java programs.

Q3: Describe the Java Runtime Environment (JRE).

Ans: The Java Runtime Environment (JRE) is nothing but a subset of the JDK (Java Development Kit) that is mainly responsible for running the Java application. JRE also consists of the Java Virtual Machine (JVM) and base libraries needed to execute Java programs.

Q4: Define the importance of the Java Virtual Machine (JVM) in Java applications.

Ans: Java Virtual Machine (JVM) is a significant part of the Java Runtime Environment (JRE). JVM is nothing but an executer that runs Java bytecode. It converts bytecode into machine-specific code.

Q5: Explain how JVM provides the facility of platform independence.

Ans: JVM provides the facility of platform independence by interpreting Bytecode. Later, it executes the Java bytecode on the host machine, where the machine gains complete hardware information and makes sure that the interpreted Java bytecode executes on numerous platforms.

Q6: Explain the Just-In-Time (JIT) compiler.

Ans: The JIT compiler changes the Java bytecode into native machine code to enhance performance by reducing the interpretation duration, which leads to improved running speed.

Q7: List out the key components of a JVM.

Ans: Java Virtual Machine (JVM) has three main key elements, i.e., class loader, execution engine, and runtime data areas. The main task of the class loader is to load classes; the role of the execution engine is to interpret bytes; and runtime data handles memory during the source code implementation.

Q8: Explain the garbage collection in JVM.

Ans: Garbage Collection (GC) in JVM de-allocates memory that is not required anymore. The main task of garbage collection (GC) is to figure out and remove the objects for the new ones.

Q9: Differentiate between JDK, JRE, and JVM.

Ans: JDK is a wholesome development kit with all the required tools and libraries for robust mobile and web application development. On the other side, JRE is responsible for running Java apps, whereas JVM runs Bytecode.

Q10: How do I install the JDK?

Ans: Java developers, whether they are newbies or professionals, can easily download and install JDK on their machines. The first step to installing the Java Development Kit is:

* Go to the official website of ORACLE.
* Search for the JDK download page.
* Choose the version that best fits your machine (operating system).
* Now, download the installer and follow all the necessary instructions to get it started.

Q11. Difference between Java SE and Java EE

Ans: Java SE means Java (Standard Edition) is for general-purpose applications, whereas Java EE (Enterprise Edition) is nothing but the extension of Java SE. It has additional APIs that enable developers to create large-scale, multi-tiered enterprise applications.

Q12. List out memory areas allocated by JVM.

Ans: There are many memory areas allocated by JVM:

* Heap
* Stack
* Class Area
* Program Counter Register

Q13. List out all the different types of loaders.

Ans: Types of classloaders are:

* Bootstrap
* Extension
* System
* Plugin

Q14. What is the purpose of the Java class library?

Ans: The motivation of the Java class library is to provide functions to accomplish several tasks, like maintaining lists or parsing strings.

Q15. What is a Java card?

Ans: Java Card, a technology through which small Java-based apps can easily and securely run on smart cards and small memory devices.