**JDK Assignment**

1. **Explain the components of the JDK.**

The component of JDK are of 3 types they are as follows –

Java Compiler (javac) : The Java Compiler (javac) is a key component of JDK that transforms Java source code (.java files) into bytecode (.class files). The generated bytecode can be executed on any platform with a Java Virtual Machine (JVM) installed, ensuring the “write once, run anywhere” philosophy of Java.

Java Virtual Machine (JVM) : The Java Virtual Machine (JVM) is the runtime engine that executes Java bytecode. It provides an abstraction layer between the Java application and the underlying operating system. The JVM enables Java programs to run independently of the hardware and operating system, enhancing portability and security.

Java Runtime Environment (JRE) : The Java Runtime Environment (JRE) is a subset of JDK that includes the JVM and essential class libraries. It is required to run Java applications on end-user systems without the need for development tools. Users can execute Java applications using the JRE, ensuring a seamless experience.

Java API : An application programming interface(API) is a connection between computers or between computer programs. In a more simple way – API is a set of ways and rules for interaction and data exchange between different programs and computers. Example : You are in a restaurant. Your are a client, you choose an order from the menu. The kitchen is the executor of your order. You need an intermediary who will report the order to the kitchen and deliver your food to your table. It could not be a chief because he is busy in cooking in the kitchen. You need someone to connect the client and the chief. And here waiter could help use – API. Waiter takes your order, report it to kitchen and then deliver the answer / food to you.

1. **Differentiate between JDK, JVM, and JRE.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Purpose | JDK | JVM | JRE |
| 1. | Definition | Software development kit for Java, including tools and libraries for developing Java application | Virtual machines that executes Java bytecode and provide a runtime environment for Java appplication | Subset of the JDK that includes the JVM and essential libraries required for executing Java applications |
| 2. | Components | -Java compiler(javac) development tools (debugger, archive tool, etc.)  -Java runtime libraries, additional libraries and APIs for development | -Interpreter for Java bytecode  - Just-In-Time(JIT)  Complier(in some JVM implementations)  -Garbage collector  -Runtime libraries | -Java Virtual Machine(JVM)  -Java runtime libraries, additional required for running Java applications |
| 3. | Purpose | Used for Java application development, including writing, compiling and debugging code | Execute Java bytecode and provides a platform-independent runtime environment for Java applications | Provides the runtime environment necessary for executing Java application, but does not include development tools like complier & debugger. |
| 4. | Example usage | Developing, compiling & debugging Java applications | Running Java applications on various platforms. | Running standalone Java applications or Java applets within web browsers. |

1. **What is the role of the JVM in Java? & How does the JVM execute Java code?**

The JVM (Java Virtual Machine) plays a crucial role in the Java ecosystem:

Role of JVM :

- It serves as an interpreter for Java bytecode, executing Java programs on any platform.

- It provides a runtime environment that abstracts away the underlying hardware and operating system details.

- It manages memory allocation and garbage collection, ensuring efficient use of resources.

- It facilitates platform independence, allowing Java programs to run unchanged on different systems with a compatible JVM.

2. Execution of Java Code :

- The Java compiler translates source code into platform-independent bytecode.

- The JVM then interprets or optionally compiles this bytecode into machine-specific instructions.

- During execution, the JVM manages memory, loads classes as needed, and handles exceptions.

- Just-In-Time (JIT) compilers in modern JVMs further optimize performance by translating frequently executed bytecode into native machine code.

- Overall, the JVM provides a runtime environment where Java code can be executed efficiently and reliably across diverse platforms.

1. **Explain the memory management system of the JVM.**

The JVM manages memory through automatic garbage collection, where unreachable objects are identified and removed to free up memory for new allocations. This process ensures efficient memory utilization and prevents memory leaks. Additionally, the JVM can optimize memory usage through techniques like generational garbage collection and adaptive sizing of memory areas. The memory management system of the JVM involves three main areas:

Heap Memory :

- This is where objects and their instance variables are stored.

- The heap is divided into two main sections: the young generation and the old generation.

- New objects are allocated in the young generation, and when it becomes full, a garbage collection process called minor GC is triggered to reclaim memory from unreachable objects.

- Objects that survive multiple minor GC cycles are promoted to the old generation.

Method Area :

- This area stores class metadata, static variables, and constant pool information.

- In older JVM implementations, this was known as the Permanent Generation (PermGen). In newer versions, it's called Metaspace.

- Metaspace dynamically adjusts its size to accommodate class metadata and reduces the risk of PermGen space errors seen in older JVMs.

Stack Memory :

- Each thread in a Java application has its own stack memory.

- Stack memory is used for storing method invocations, local variables, and partial results.

- It operates in a last-in, first-out (LIFO) manner.

- Stack memory is typically smaller than heap memory and is released when the method completes execution.

1. **What are the JIT compiler and its role in the JVM? What is the bytecode and why is it important for Java?**

JIT Compiler (Just-In-Time Compiler) :

- The JIT compiler is a component of the JVM that improves the performance of Java applications by dynamically translating Java bytecode into native machine code during runtime.

- It identifies frequently executed sections of bytecode (hot spots) and compiles them into highly optimized native code, which can execute more efficiently on the underlying hardware.

- The JIT compiler helps reduce the interpretation overhead of bytecode, resulting in faster execution of Java programs.

Bytecode :

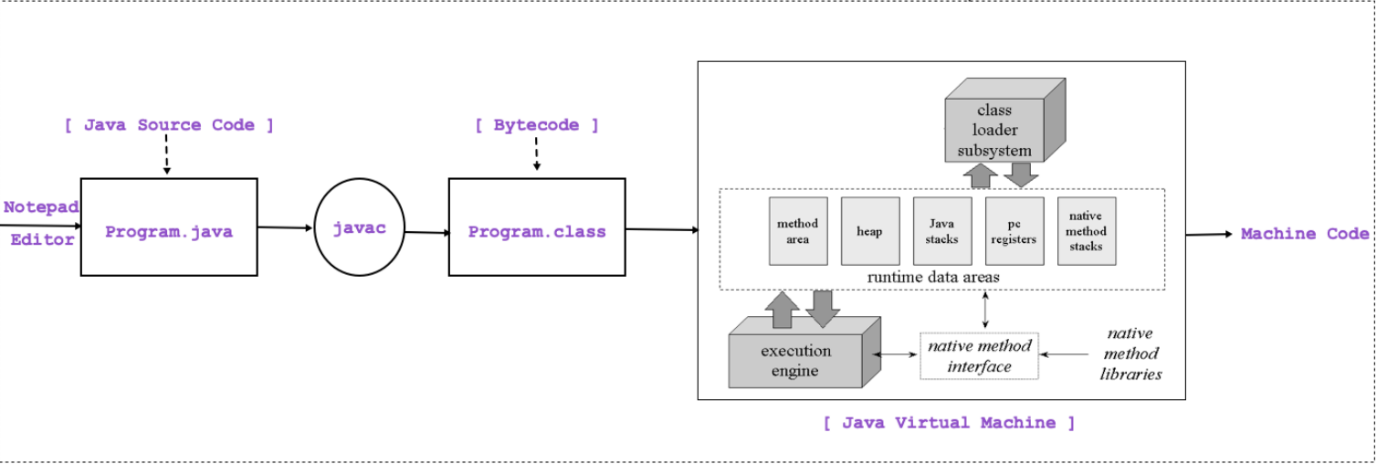
- Bytecode is the intermediate representation of Java source code after compilation by the Java compiler (javac).

- It is a platform-independent format that can be executed on any system with a compatible JVM, making Java programs inherently portable.

- Bytecode enables "write once, run anywhere" (WORA) paradigm, allowing Java applications to run unchanged across different platforms without recompilation.

- Bytecode is essential for Java's cross-platform compatibility and enables the Java ecosystem to thrive on diverse computing environments.

1. **Describe the architecture of the JVM.**



The architecture of the Java Virtual Machine (JVM) can be broken down into several key components, each playing a crucial role in executing Java bytecode efficiently. They are:

Class Loader Subsystem :

- Responsible for loading class files into memory.

- Consists of three main components:

- \*\*Bootstrap Class Loader\*\*: Loads core Java classes from the bootstrap classpath.

- \*\*Extension Class Loader\*\*: Loads classes from the extension directories.

- \*\*Application Class Loader\*\*: Loads classes from the application classpath.

Runtime Data Area :

- Divided into several memory areas:

- Method Area (Metaspace): Stores class metadata, static variables, and constant pool data.

- Heap: Stores objects and their instance variables, divided into young and old generations.

- Stack: Each thread has its own stack for method invocations and local variables.

- PC Registers: Hold the address of the current instruction being executed.

- Native Method Stack: Stores native method information.

Execution Engine :

- Responsible for executing Java bytecode.

- Consists of:

- Interpreter: Interprets bytecode instructions and executes them sequentially.

- Just-In-Time (JIT) Compiler: Compiles frequently executed bytecode into native machine code for improved performance.

- Garbage Collector Interface: Provides a framework for various garbage collection algorithms to manage memory.

1. **How does Java achieve platform independence through the JVM?**

Java achieves platform independence through the Java Virtual Machine (JVM) and the Java bytecode.

Bytecode : When you compile a Java source file, it's translated into platform-independent bytecode. This bytecode is a set of instructions meant to be executed by the JVM. It's neither specific to any hardware nor any operating system.

JVM (Java Virtual Machine): The JVM is an abstract computing machine that provides a runtime environment for executing Java bytecode. Each operating system has its own implementation of the JVM, tailored to that specific system. When you run a Java program, you don't execute the bytecode directly; instead, it's interpreted or compiled by the JVM into machine code that's specific to the underlying hardware and operating system.

By having the bytecode executed by the JVM rather than directly by the underlying hardware or operating system, Java programs can be run on any system with a compatible JVM implementation. This is what allows Java to be "write once, run anywhere," as the same bytecode can be executed on any platform with a JVM, providing platform independence.

1. **What is the significance of the class loader in Java? What is the process of garbage collection in Java.**

Class Loader in Java : The class loader in Java is responsible for loading classes into the Java Virtual Machine (JVM) dynamically at runtime. It locates and reads the binary data for a class file, which typically resides in the file system, and then creates a corresponding class object in memory. There are three built-in class loaders in Java:

1. Bootstrap Class Loader: It loads the core Java libraries (rt.jar, etc.) which are located in the bootstrap classpath.

2. Extension Class Loader: It loads classes from the extensions directories (usually $JAVA\_HOME/lib/ext).

3. Application Class Loader: It loads classes from the application classpath, which is specified by the CLASSPATH environment variable or the -classpath command-line option.

Garbage Collection in Java : Garbage collection in Java is the process of automatically reclaiming memory occupied by objects that are no longer in use by the program. Java uses a garbage collector to perform this task, which runs as a low-priority background thread in the JVM. The process of garbage collection involves the following steps:

1. Marking: The garbage collector identifies which objects in the heap are reachable and which are not. It starts from a set of root objects (such as local variables, static variables, and thread stacks) and recursively marks all objects that are reachable from them.

2.Sweeping:Once the reachable objects are identified, the garbage collector sweeps through the heap and deallocates memory for objects that are not marked as reachable. This memory is then made available for future allocations.

3. Compacting : Some garbage collectors perform heap compaction, where live objects are moved to contiguous memory locations to reduce fragmentation and improve memory locality.