Lesson-1

JShell

```
jshell> String hello(){
   ...>    return "Hello World!";
   ...> }
|  created method hello()
```

Key Components

Java Development Kit (JDK):

- Purpose: Development environment for building and compiling Java applications.
- Components: Includes JRE, javac, jar, Javadoc, and other development tools.

Java Runtime Environment (JRE):

- Purpose: Provides the runtime environment necessary to run Java applications.
- Components: Includes the JVM, core libraries, memory management and other runtime components.

Java Virtual Machine (JVM):

- Purpose: Abstract computing machine that runs Java bytecode.
- Components: Includes a class loader, bytecode verifier, and the JIT compiler.

Just-In-Time (JIT) Compiler:

- Purpose: Enhances performance by compiling bytecode into native machine code at runtime.
- Components: Part of the JVM, dynamically optimizes code execution.

What is the Interpreter?

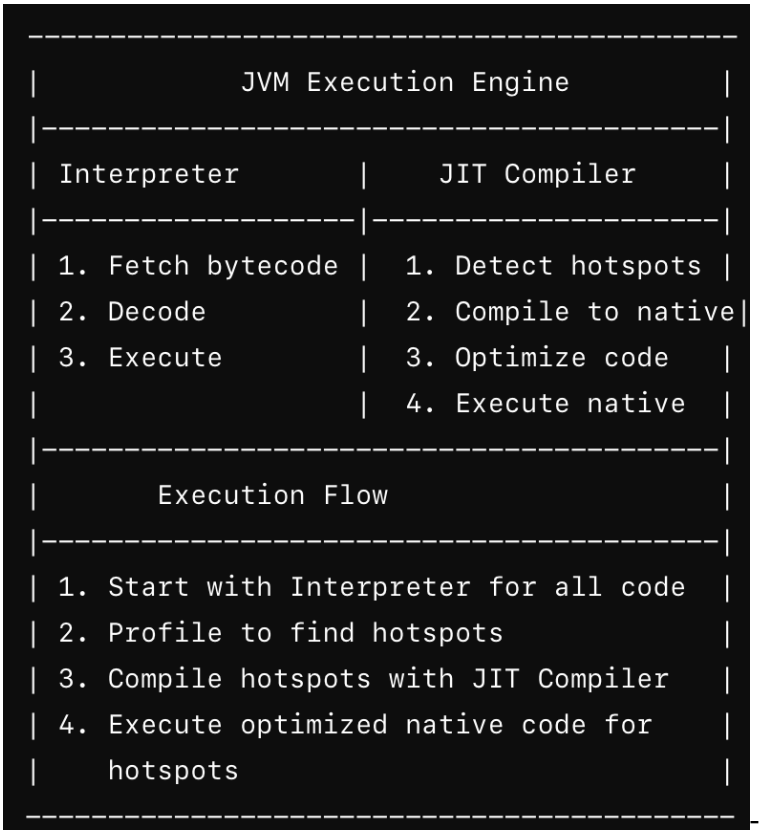
The Interpreter is a component of the JVM that directly executes the bytecode of a Java program line by line. It translates each bytecode instruction into machine code and then executes it on the fly.

How the Interpreter Works:

Fetch: Reads a bytecode instruction.
Decode: Determines what operation the instruction specifies.
Execute: Performs the operation specified by the instruction.

JIT Compiler:

The JIT (Just-In-Time) Compiler is a component of the Java Virtual Machine (JVM) that improves the performance of Java applications by converting bytecode into native machine code at runtime. This native code runs directly on the hardware, making execution faster compared to interpreting the bytecode line by line. The JIT Compiler identifies frequently executed code sections (hotspots), compiles them into optimized machine code, and executes this code for better performance.



Summary:

- Interpreter: Executes bytecode line by line, ideal for initial program execution and ensuring platform portability.
- JIT Compiler: Compiles and optimizes frequently executed code to native machine code for better performance during long-running applications.
- Combined Approach: The JVM uses both to balance the trade-off between immediate execution and long-term performance gains.
