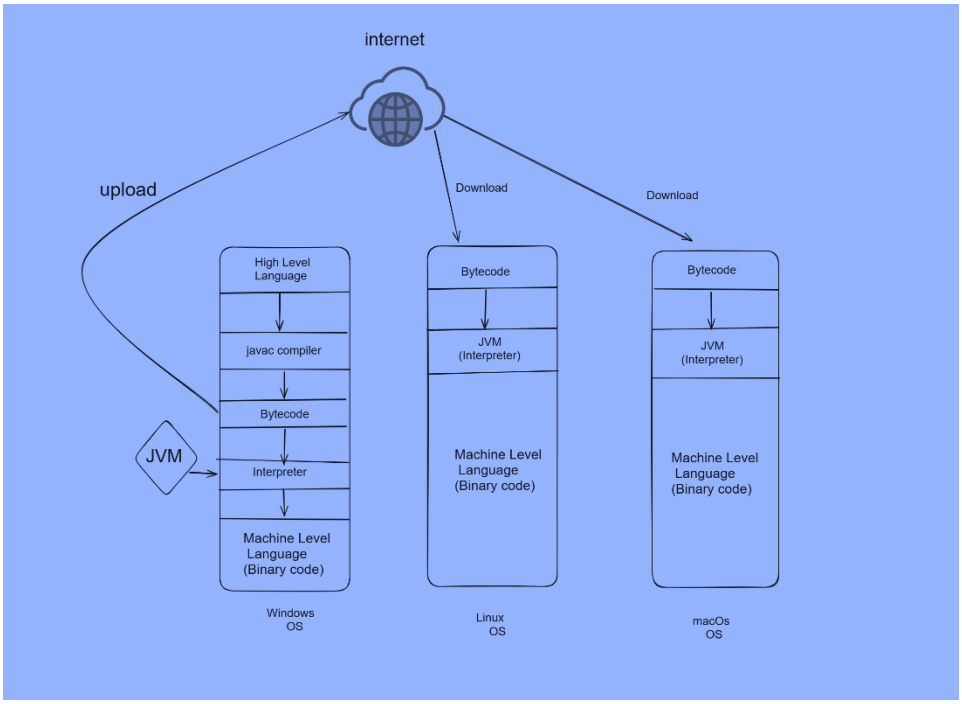
4-How Java Made Platform Independent

In this chapter, let's dive in and understand how Java is made platform independent with an example.

Example of Platform Independence

Suppose we have three systems using different operating systems: Linux, Windows, and macOS. This time, we will build an application based on Java. When we write code in Java, we use a high-level language (HLL), which uses English-like syntax and symbols. This code needs to be converted into machine-level binary code by a system software called a compiler.



Java, however, uses a hybrid compiler called the **Javac compiler**. Unlike traditional compilers that directly convert HLL code into machine language, the Javac compiler first converts the code into an intermediate form called bytecode. This bytecode is neither HLL nor MLL but lies somewhere in between.

To convert this bytecode into machine-level language (MLL), Java uses another component called the **Java Virtual Machine (JVM)**. The JVM is responsible for converting the bytecode into MLL, allowing the program to execute and produce the desired output.

Key Points About JVM

1. **Platform Dependent**: Each operating system has its own specific JVM. Therefore, JVMs differ for Windows, Linux, and macOS.
2. **Part of JDK**: When installing Java on any operating system, you download the Java Development Kit (JDK), which includes the JVM.

3. **Interpreter**: The JVM uses an interpreter to convert bytecode into machine-level language (binary code).

Example Workflow

1. **Development on Windows:**
    - Write Java code in HLL.
    - Use the Javac compiler to convert the code into bytecode.
2. **Sharing Bytecode:**
    - Upload the bytecode generated on the Windows system to the network.
    - Other systems (Linux and macOS) download the bytecode.
3. **Execution on Different Systems:**
    - Install the JDK specific to each operating system.
    - Each system's JVM, included in the JDK, interprets the bytecode.
    - The interpreter converts the bytecode into machine-level binary code.
    - The output is consistent across all operating systems.

Due to this process, Java supports the **WORA (write once, run anywhere)** feature. The bytecode, generated on one platform, can run on any other platform with a compatible JVM, providing the same output.

Moving Forward

In the next chapter, we will gain deeper knowledge about how Java internally converts bytecode to machine-level code and explore the JVM's internal architecture.

Stay curious and keep learning.