JAVA PROGRAMMING LANGUAGE

What is Java ?

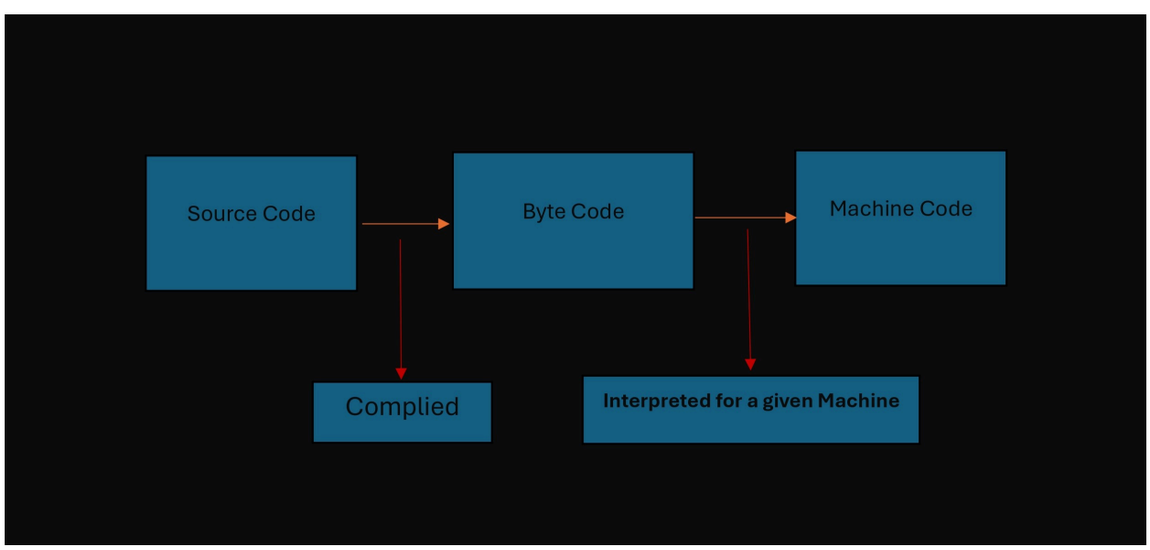
Java is an object oriented programming language ,developed by Sun MicroSystems of USA in 1991.

It was originally called oak by James Goslin . (one of the inventors of java)

JAVA - Purely Object Oriented

How Works JAVA ?

Java is Compiled into the bytecode and then it is interpreted to machine code.



1. What is JDK?

- JDK stands for **Java Development Kit**.
- It is a **software package** that provides all the tools, libraries, and executables needed to **develop, compile, and run** Java programs.
- Without JDK, you cannot write or run Java programs.

---

2. Main Components of JDK

1. **JRE (Java Runtime Environment)**
    - Provides the environment required to run Java programs.
    - Contains JVM + Libraries.
    - Used only for running programs.
2. **JVM (Java Virtual Machine)**

- Converts Java bytecode into machine code.
- Makes Java platform-independent.

1. **Compiler (`javac`)**
    - Translates `.java` (source code) into `.class` (bytecode).
2. **Interpreter / Java Launcher (java)**
    - Executes the bytecode program.
3. **Development Tools**
    - `javac` → compiler
    - `java` → runs programs
    - `javadoc` → creates documentation
    - `jar` → packages multiple classes
    - `jdb` → debugger

> JDK ⟶ JAVA DEVELOPMENT KIT
>
> Collection of tools used for developing and running java program.

> JRE ⟶ JAVA RUNTIME ENVIRONMENT
>
> Helps in Executing program developed in Java.

JVM → JAVA VIRTUAL MACHINE

JVM is a **virtual machine** that executes Java bytecode and converts it into **machine code**

3. Difference Between JDK, JRE, and JVM

Feature	JVM	JRE	JDK
Full Form	Java Virtual Machine	Java Runtime Environment	Java Development Kit
Purpose	Executes bytecode	Runs Java programs	Develops + runs Java programs
Contains	Execution engine	JVM + Libraries	JRE + Compiler + Tools
Usage	Only run	Run	Develop + run

4. How JDK Works? (Process Flow)

1. Write program – `MyProgram.java`
2. Compile with `javac` → generates **bytecode** (`MyProgram.class`)
3. JVM reads bytecode → converts to machine code for OS
4. Program executes successfully

5. Example Program (Using JDK)

```
// Save as Hello.java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, JDK!");
    }
}
```

⇒ Steps to Run:

1. Compile: `javac Hello.java`
2. Run: `java Hello`
3. Output:

```
Hello, JDK!
```

6. Types of JDK

1. **Oracle JDK** – Official, stable, but commercial use restrictions.

2. **OpenJDK** – Free, open-source version.
3. **Other builds** – Amazon Corretto, Azul Zulu, AdoptOpenJDK.

✅ **Conclusion**:

- JDK = JRE + Development Tools.
- Required for both **writing and running** Java programs.
- Core foundation of Java development.

Basic Structure of a Java Program

```
package mypackage;                          // 1. Package declaration (optional)
  public class MyFirstProgram {               // 2. Class definition (mandatory)
    public static void main(String[] args) {   // 3. Main method (mandatory)
    System.out.println("Hello, World!");       // 4. Statements and print command
   }
}
```

Explanation of Each Part

1. **Package Declaration (optional)**
   - Used to organize classes into groups (like folders).
   - Example:

     ```
     package mypackage;
     ```

2. **Class Definition (mandatory)**
   - Every Java program must have at least one class.
   - Syntax:

     ```
     public class ClassName {
        // class body
     }
     ```

3. **Main Method (mandatory)**
   - Entry point of the program; execution starts here.
   - Syntax:

     ```
     public static void main(String[] args) {
        // statements
     }
     ```

4. **Statements or Print (Program Logic)**

- Actual code you want to execute, such as input/output, loops, or conditions.
- Example:

```
System.out.println("Hello, World!");
```

---

1. **Import Statements (optional)**
   - Used to access built-in or user-defined classes from other packages.
   - Example:

```
import java.util.Scanner;   // Strating

// Code Here...

  sc.close();          // Ending
```

Java Imports – Default vs Required

Classes Imported by Default

These belong to the `java.lang` package.

They are **always available** in every Java program without needing an `import` statement.

- `String`
- `Object`
- `System`
- `Math`
- Wrapper classes: `Integer`, `Double`, `Float`, `Boolean`, `Character`
- Exception classes: `Exception`, `Throwable`, `Error`

⇒ You can use these **directly** without importing.

Classes That Require Import

For all other packages, you must explicitly use an `import` statement.

Utility Classes (`java.util`)

- `Arrays`
- `Scanner`
- `ArrayList`
- `HashMap`, `HashSet`, `LinkedList`

- `Collections`

Input/Output Classes (`java.io`)

- `File`
- `BufferedReader` , `BufferedWriter`
- `FileReader` , `FileWriter`

Date & Time Classes (`java.time`)

- `LocalDate`
- `LocalTime`
- `LocalDateTime`

Important Points

- The **file name must match the public class name** (case-sensitive).
  - Example: Class name `MyFirstProgram` → File name `MyFirstProgram.java` .
- Every statement in Java ends with a **semicolon (;)**.
- Without the `main()` method, the program won't run (except in special cases like applets or JavaFX).