What is Programming?

Programming is the process of writing instructions (called code) that a computer understands to perform a task.

What is Java?

Java is a **high-level, object-oriented, platform-independent programming language** developed by **Sun Microsystems** in **1995** (now owned by **Oracle Corporation,** since 2010). It is designed to be **simple**, **secure**, and **portable**, allowing developers to **write code once and run it anywhere** — thanks to the **Java Virtual Machine (JVM)**.

Key Points:

- **Object-Oriented**: Everything in Java is based on objects and classes.
- **Platform-Independent**: Java code runs on any device with a JVM (Write Once, Run Anywhere – WORA).
- **Compiled + Interpreted**: Java code is compiled into bytecode which is interpreted by the JVM.
- **Used For**: Web apps, mobile apps (Android), enterprise systems, desktop software, and more.

Java's origin:

---

1. The Green Project (1991)

- In **1991**, a small team of engineers at **Sun Microsystems**, led by **James Gosling**, started a secret project called **"The Green Project."**
- **Goal:** Build software for **next-generation devices** like interactive TVs, set-top boxes, and handheld devices.
- **Challenge:** Existing languages like C/C++ were platform-dependent and too complex for portable consumer electronics.
- **Solution direction:** Create a **small, reliable, platform-independent language** that could run anywhere without recompilation.
- The first prototype language was called **Oak** (named after an oak tree outside Gosling's office).

---

2. Why Oak Failed

- In **1992–1994**, Sun tried to market **Oak** to the consumer electronics industry (e.g., for interactive TV).
- **Problems:**
  - **Hardware companies weren't ready** to adopt new software platforms.
  - **Oak trademark issue:** The name "Oak" was already registered by another company.
  - **Slow industry adoption:** Interactive TV market didn't take off as expected.
- Result: The project stalled and was re-targeted.

---

3. Birth of Java (1995)

- The internet boom in **1994–1995** gave the language a new life.
- Sun Microsystems rebranded **Oak** as **Java** in **1995**.
- **Why Java?** (Rumor says it was named after Java coffee, symbolizing energy and productivity.)
- **Key features emphasized:**
  - *Write Once, Run Anywhere* (WORA) — Java programs could run on any device with a **Java Virtual Machine (JVM)**.
  - Secure and network-friendly design — perfect for web applets.
- **Milestone: Netscape Navigator** browser adopted Java applets, making Java instantly popular.

Java Program Execution Flow

1. **Java Source Code** is written in a .java file (e.g., A.java).
2. The **Java Compiler (javac)** compiles the source code into **bytecode**, saved in a .class file.
3. **Bytecode** is a platform-independent code that can run on any OS with JVM.
4. The **JVM (Java Virtual Machine)** executes the bytecode in 3 steps:
   - **Class Loader**: Loads the .class file into memory.
   - **Bytecode Verifier**: Checks bytecode for security and correctness.
   - **JIT Compiler (Just-In-Time)**: Converts bytecode into machine code for execution.
5. The final **machine code** runs on the native system (e.g., Windows, Linux, Mac).
6. This process makes Java **platform independent** –
   "**Write Once, Run Anywhere**".

Java Architecture

What is JVM (Java Virtual Machine)?

JVM is an abstract machine that enables your **Java bytecode** to run on any device or operating system.
It reads .class files (bytecode) and **converts them into machine-specific instructions**, executing them line by line.

Key Roles:

- Executes Java bytecode
- Provides **platform independence**
- Handles **memory management**, **garbage collection**, **security**, and **multithreading**

JVM = Runtime engine for Java applications

What is JRE (Java Runtime Environment)?

JRE is a **software package** that provides everything needed to **run** Java programs, but not to develop them.
It includes the **JVM**, core libraries, and supporting files.

JRE = JVM + Java Libraries

What is JDK (Java Development Kit)?

JDK is the **complete development toolkit** required to **build, compile, and run** Java applications.
It includes the **JRE**, **JVM**, and **development tools** like javac (compiler), javadoc, debugger, etc.

JDK = JRE + Development Tools

Features of Java:

1. **Simple**
   Java is easy to learn, write, and understand.
   ➤ Syntax is clean and similar to C/C++ but with fewer complex features (like pointers).

2. **Object-Oriented**
   Everything in Java is treated as an object.
   ➤ Supports concepts like Class, Object, Inheritance, Polymorphism, Encapsulation, and Abstraction.

3. **Platform Independent**
   Java code runs on any OS that has JVM.
   ➤ "**Write Once, Run Anywhere**" – thanks to bytecode and JVM.

4. **Secure**
   Java provides a secure runtime environment.
   ➤ No direct memory access, bytecode verification, and security manager.

5. **Robust**
   Java handles runtime errors through **exception handling** and strong **memory management** (garbage collection).
   ➤ Less chances of crashes.

6. **Multithreaded**
   Java can perform multiple tasks at once using **threads**.
   ➤ Useful in games, animations, and real-time applications.

7. **Architecture Neutral**
   Java bytecode is not tied to any processor or system architecture.
   ➤ Same bytecode runs on any machine with a JVM.

8. **Portable**
   Java code can be easily moved across platforms.
   ➤ No need to change the code for different machines.

9. **High Performance**
   Java is faster than other interpreted languages (like Python).
   ➤ Uses **Just-In-Time (JIT)** compiler for better performance.

10. **Distributed**
    Java supports building **distributed applications** (apps across networks).
    ➤ Uses technologies like RMI, EJB, and web services.

11. **Dynamic**
    Java loads classes and objects at **runtime**, not at compile time.
    ➤ Supports dynamic linking and reflection.

1. What are the key features of Java?

Java is a powerful, flexible, and widely-used programming language with the following key features:

- **Simple**: Easy to learn with clean syntax.
- **Object-Oriented**: Follows OOP principles (Class, Object, etc.).
- **Platform-Independent**: Runs on any OS with JVM (Write Once, Run Anywhere).
- **Secure**: No pointers, bytecode verification, and sandboxing.
- **Robust**: Strong memory management, garbage collection, and exception handling.
- **Multithreaded**: Supports concurrent execution using threads.
- **Architecture Neutral**: Bytecode is not system-specific.
- **Portable**: Code can be moved across platforms easily.
- **High Performance**: Faster execution using JIT compiler.
- **Distributed**: Supports networking and distributed computing.
- **Dynamic**: Loads classes at runtime, supports reflection.

---

2. How is Java platform-independent?

Java is platform-independent because of the **Java Virtual Machine (JVM)**.

- When you write Java code, it is compiled into **bytecode** (.class file).
- This bytecode is **not platform-specific**.
- Any device with a **JVM** can interpret and run the bytecode.

This allows Java to follow the principle:
"Write Once, Run Anywhere" (WORA).

---

3. How does Java ensure security?

Java has several built-in mechanisms to ensure security:

- **No Pointers**: Prevents direct memory access.
- **Bytecode Verification**: Checks for illegal code before execution.

- **Security Manager**: Defines access control for classes.
- **Sandboxing**: Isolates untrusted code (e.g., applets) from system resources.
- **Exception Handling**: Avoids crashes by managing runtime errors safely.

Java is widely used in banking, enterprise, and secure web applications for this reason.

---

4. What makes Java robust?

Java is considered robust because it focuses on **early error checking**, **strong memory management**, and **reliable exception handling**.

- **Compile-time & runtime error checking**
- **Automatic Garbage Collection**: Frees unused memory.
- **Exception Handling**: Catches and manages errors gracefully.
- **Strong Typing**: Prevents type mismatch and unsafe operations.

These features help avoid application crashes and memory leaks.

---

5. Difference between Java and C++ in terms of simplicity?

Feature	Java	C++
Memory Management	Automatic (Garbage Collection)	Manual (you must use new/delete)
Syntax	Cleaner and simpler	More complex (multiple inheritance, pointers, templates)
Pointers	Not supported	Fully supports pointers
Header Files	Not required	Required
Multiple Inheritance	Not supported directly (uses interfaces)	Supported (but can lead to ambiguity)
Standard Library	Rich and easy to use	Complex (e.g., STL)

Java is simpler because it removes many of the complex and error-prone features found in C++.

5. Why Java Doesn't Have Explicit Pointers?

- **Simplicity and Security**: Pointers allow direct memory access, which can lead to security issues like buffer overflow, memory corruption, etc.
- **Automatic Memory Management**: Java uses **Garbage Collection** — it handles memory for you.
- **References Instead**: Java uses **references** internally, but doesn't expose pointer arithmetic or direct memory access.

Structure of a Java Program

Each Java program must follow a certain structure to be valid and executable.

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

Explanation

1 class – Blueprint for Objects

The keyword **class** is used to define a class.
A **class is a blueprint** or template from which objects are created.
It contains variables and methods that define the behavior of objects.

---

2 public – Access Modifier

public is an **access modifier**.
It means this class or method is **accessible from anywhere** in the program.
Without public, the main() method might not be called by the JVM.

3 static – Belongs to Class, Not Object

The keyword **static** means the method belongs to the **class itself**, not to any specific object.
Because the main() method is static, the JVM can call it **without creating an object** of the class.

---

4 void – No Return Value

void means the method **does not return anything**.
The main() method just runs code; it doesn't send any value back.

---

5 main – Method Name (Entry Point)

The word **main** is the name of the method.
This is the **entry point** of any Java program.
The JVM starts executing code from the main() method.

---

6 String[] args – Command-Line Arguments

Inside the main method, we write String[] args.
This means the method takes an **array of Strings** as input.
These are called **command-line arguments**.
We can pass input values to the program through the command line.

---

7 System.out.println() – Output Statement

This line is used to **print something on the console**.
System is a predefined class.
out is a static object inside System.
println() is a method that prints and moves to the next line.
For example:

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

This will display: **Hello, World!**

Naming Conventions in Java

Naming conventions are a set of **rules or guidelines** that help developers write code that is **readable, consistent, and maintainable**.

Java is a **case-sensitive language** — which means:
MyClass and myclass are **not the same**.

---

1 Class Names – PascalCase

- Class names should start with a **capital letter**.
- If the name has multiple words, each word should also start with a **capital letter**.
- This style is called **PascalCase**.

Example:

StudentDetails

BankAccount

LoginController

---

2 Variable Names – camelCase

- Variable names should start with a **small letter**.
- If the name has multiple words, **first word is lowercase**, the rest begin with capital letters.
- This is called **camelCase** because it looks like a camel's back.

Example:

studentName

accountBalance

totalMarks

---

3 Method Names – camelCase

- Method names follow the **same rule as variable names**.

- Start with a small letter, use capital letters for additional words.
- Methods often start with verbs, like get, set, calculate, etc.

Example:

getDetails()

calculateSalary()

printReport()

---

4 Constants – UPPER_SNAKE_CASE

- Constants are variables whose values **never change**.
- Constant names are written in **all capital letters**, with words separated by **underscores**.
- This is called **UPPER_SNAKE_CASE**.

Example:

```
PI = 3.14

MAX_USERS = 100

DATABASE_URL = "localhost"
```

Comments in Java

Single-line comment

```
// This is a comment
```

Multi-line comment

```
/* This is
  a multi-line comment */
```

Documentation comment

```
/**
 * This class prints Hello
```

*/

Writing comments improves readability!

Printing to Console

println() Prints + new line

print() Prints without new line

printf() Formatted output

```
public class PrintTable {
   public static void main(String[] args) {
      System.out.printf("%-10s %5s %10s%n", "Item", "Qty", "Price");
      System.out.printf("%-10s %5d %10.2f%n", "Apple", 3, 60.50);
      System.out.printf("%-10s %5d %10.2f%n", "Banana", 2, 25.00);
   }
}
```