Java Programming Language – Introduction, History & Purpose

What is Java?

Java is a **high-level, class-based, object-oriented programming language** designed to have as few implementation dependencies as possible. It is:

- Simple
- Secure
- Portable
- Platform-independent
- Robust
- Multi-threaded
- Architecture-neutral

Java follows the principle of **"Write Once, Run Anywhere" (WORA)**, meaning code written in Java can run on any platform that supports Java without the need for recompilation.

Who Developed Java?

- **Creator:** James Gosling
- **Company:** Sun Microsystems
- **Year:** Development began in **1991**, first public release in **1995**
- James Gosling is known as the "**Father of Java**"

Original Name of Java:

- Java was originally called **Oak**.
- The name was changed to **Java** in 1995 because “Oak” was already a registered trademark.
- The name "Java" was inspired by **Java coffee** (from the Indonesian island of Java).

Why Was Java Developed?

Java was created with the following goals:

Purpose	Description
Platform Independence	Write Once, Run Anywhere
Security	Built-in security features (no pointers, sandbox environment)
Simplicity	Easier to learn than C++, avoids complex features like pointers
Object-Oriented	Supports modular, reusable, and organized code
Robust and Error-Free	Strong memory management and exception handling
Network Capable	Built-in support for network and distributed computing

Short History / Timeline of Java:

Year	Event
1991	Java project initiated by James Gosling (originally named Oak)
1995	First public version of Java released
1998	Java 2 (J2SE) released – more powerful and modular
2006	Sun Microsystems made Java open-source
2009	Oracle Corporation acquired Sun Microsystems
2014–2023	Java versions 8 to 21 released (Java 8 is still widely used)

Where is Java Used?

Java is used in a wide range of applications:

- Android App Development
- Web Applications (e.g., using Spring, JSP, Servlets)
- Enterprise Applications (e.g., ERP, Banking systems)
- Scientific and Research-based Applications
- Desktop GUI Applications (e.g., Swing, JavaFX)
- Embedded Systems
- Game Development
- Big Data and Cloud-based Applications

Java – First Program: Hello World

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

Line-by-Line Explanation:

1. `public class HelloWorld {`

- **`public`** – Access modifier: this class is accessible from anywhere.
- **`class`** – This keyword is used to define a class.
- **`HelloWorld`** – The name of the class (should match the file name `HelloWorld.java`).

Every Java program must have a class, and the file name must match the class name.

2. `public static void main(String[] args) {`

This is the **main method** — the starting point of any Java program.

- **`public`** – The method can be accessed from outside the class.
- **`static`** – Can be run without creating an object of the class.
- **`void`** – This method does not return any value.
- **`main`** – The name of the method where program execution begins.

- **`String[] args`** – An array that can hold command-line arguments.

Java always looks for the `main()` method to start executing the program.

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

This line prints the message on the console:

- **`System`** – A built-in class from the `java.lang` package.
- **`out`** – A static object of the `PrintStream` class, connected to the console.
- **`println()`** – Method used to print the string with a newline.

This line will output: **Hello, World!**

4. `Curly Braces {}`

These curly braces close the `main()` method and the `HelloWorld` class.

How This Program Works – Step-by-Step

1. **Write Code** in a file named `HelloWorld.java`.
2. **Compile** using:

   ```
   javac HelloWorld.java
   ```

   - The Java compiler converts it into **bytecode**.
   - It creates a file: `HelloWorld.class`.
3. **Run** using:

   ```
   java HelloWorld
   ```

   - This uses the **JVM (Java Virtual Machine)** to execute the `.class` file.
   - JVM reads the bytecode and displays output.

Java is **compiled and interpreted** — it compiles to bytecode and then is interpreted by the JVM.

Output:

```
Hello, World!
```

What is a Variable in Java?

A **variable** is a name given to a **memory location** where data is stored.
Think of it like a **box** that holds some value — a number, word, or any information.

Example:

```
int age = 20;
String name = "Neeraj";
```

Here:

- `age` is a variable that stores a number `20`
- `name` is a variable that stores the text `"Neeraj"`

How to Declare a Variable

Syntax:

```
dataType variableName = value;
```

Example:

```
int marks = 90;
float pi = 3.14f;
char grade = 'A';
```

Rules for Naming Variables

1. **Only letters (a-z, A-Z), digits (0-9), _, and $ allowed**
   - Valid: `marks`, `roll_no`, `amount$`, `total1`
   - Invalid: `my-name`
2. **Must not start with a digit**
   - `1age` — wrong
   - `age1` — correct
3. **Java is case-sensitive**
   - `Age` and `age` are different variables
4. **Don't use Java keywords (like `class`, `int`, `static`)**
   - `int class = 5;` — wrong
5. **Use meaningful names**
   - `a = 10;` — unclear
   - `marks = 10;` — clear

Types of Data in Java (Data Types)

Java is a **statically typed language**, which means you must define the **type of data** a variable will store.

1. Java Primitive Data Types – Table with Size and Range

Data Type	Size (in bytes)	Value Range	Example
byte	1 byte (8 bits)	-128 to 127	byte b = 100;
short	2 bytes (16 bits)	-32,768 to 32,767	short s = 15000;
int	4 bytes (32 bits)	-2,147,483,648 to 2,147,483,647	int age = 25;
long	8 bytes (64 bits)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long population = 10000000000L;
float	4 bytes (32 bits)	~±3.4e−038 to ±3.4e+038 (7 digits precision)	float pi = 3.14f;
double	8 bytes (64 bits)	~±1.7e−308 to ±1.7e+308 (15 digits precision)	double price = 99.99;
char	2 bytes (16 bits)	Unicode characters (0 to 65,535)	char grade = 'A';
boolean	1 bit (JVM uses 1 byte internally)	true or false	boolean isOn = true;

2. Non-Primitive Data Types (Also called Reference Data Types)

Data Type	Description	Example
String	Sequence of characters (text)	String name = "Neeraj";
Array	Group of values	int[] marks = {90, 80, 70};
Class	User-defined complex type	Student s = new Student();

Simple Java Program using Variables & Data Types

```
public class Main {
    public static void main(String[] args) {
        int age = 20;
        String name = "Neeraj";
        boolean isStudent = true;
        char grade = 'A';

        System.out.println(name + " is " + age + " years
old.");
        System.out.println("Student: " + isStudent);
        System.out.println("Grade: " + grade);
    }
}
```