JAVA Introduction

What Is Java?

Java is a **high-level**, **object-oriented**, and **platform-independent** programming language. It was developed by **Sun Microsystems** in 1995 and later acquired by **Oracle Corporation**. Java is widely used for building:

- Desktop applications (e.g., accounting software)
- Web applications (e.g., e-commerce sites)
- Android mobile apps
- Enterprise systems (e.g., banking platforms)

Java is known for its **robustness**, **security**, and **portability**, making it a favorite in both academic and professional settings.

Key Features of Java (Explained in Depth)

1. Platform Independent

- Java code is compiled into **bytecode**, not machine-specific instructions.
- This bytecode runs on any system with a **Java Virtual Machine (JVM)**.
- This is called **Write Once, Run Anywhere (WORA)**.
    - Example: A Java program written on Windows can run on Linux or Mac without modification.

2. Simple

- Java removes complex features like **pointers** (used in C/C++) and **multiple inheritance** (which can cause ambiguity).
- It uses a **clean syntax** and has a rich set of built-in libraries.
- Ideal for beginners because it enforces structure and readability.

3. Object-Oriented

- Everything in Java is part of a **class** or **object**.
- This promotes:

JAVA Introduction

- **Encapsulation**: Bundling data and methods together.
- **Inheritance**: Reusing code from parent classes.
- **Polymorphism**: Using one interface for different data types.
- **Abstraction**: Hiding complex implementation details.

4. Secure

- Java avoids direct memory access (no pointers).
- It runs inside a **sandbox environment**, especially in web applications.
- Features like **automatic garbage collection** prevent memory leaks.

5. Multithreading

- Java supports **multiple threads** of execution.
- Threads allow programs to perform **multiple tasks simultaneously**.
  - Example: A game can play music, respond to user input, and update graphics at the same time.

6. Just-In-Time (JIT) Compiler

- Java uses a JIT compiler to convert bytecode into **native machine code** at runtime.
- This improves performance by avoiding repeated interpretation.

Hello World Program in Java (Line-by-Line Explanation)

```
// This is a simple Java program to print Hello World!
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

JAVA Introduction

Detailed Breakdown:

- // This is a simple Java program to print Hello World!
    - This is a **single-line comment**. It's ignored by the compiler and used to explain code.
- public class HelloWorld
    - Declares a **class** named HelloWorld.
    - In Java, **every program must be inside a class**.
    - public means the class is accessible from anywhere.
- { ... }
    - Curly braces define the **body** of the class.
- public static void main(String[] args)
    - This is the **main method**—the entry point of any Java application.
    - public: Accessible from anywhere.
    - static: Belongs to the class, not an object.
    - void: Returns nothing.
    - String[] args: Accepts command-line arguments as an array of strings.
- System.out.println("Hello World!");
    - Prints the message to the **console**.
    - System: A built-in class.
    - out: A static member of System, representing the output stream.
    - println: A method that prints text followed by a newline.

Java Program Execution Flow

1. **Write Code**: Save your code in a file named HelloWorld.java.

JAVA Introduction

2. **Compile**: Use the Java compiler javac HelloWorld.java to convert code into **bytecode** (HelloWorld.class).
3. **Run**: Use java HelloWorld to execute the program.
4. **JVM**: The Java Virtual Machine reads the bytecode and converts it into **machine code** (binary) for execution.

Comments in Java (Why and How)

Comments are used to **explain code** and are ignored during execution.

Types of Comments:

- **Single-line comment**:

  // This is a comment

- **Multi-line comment**:

  /*

  This is a multi-line comment.

  Useful for explaining larger sections of code.

  */

- **Documentation comment** (used for generating docs):

  /**

  * This method prints a message.

  */

Curly Braces and Indentation

Curly braces {} define **blocks of code**. They are used in:

- Classes
- Methods

JAVA Introduction

- Loops
- Conditionals

Example:

```
public class Geeks {
   public static void main(String[] args) {
      {
         System.out.println("This is inside the block.");
      }
      System.out.println("This is outside the block.");
   }
}
```

- The inner block runs unconditionally.
- Indentation is not required by the compiler but is **essential for readability**.

Naming Conventions in Java

Consistent naming improves **readability** and **maintainability**.

Type	Convention	Example
Class	Start with uppercase	HelloWorld
Method	Start with lowercase, use camelCase	printMessage()
Variable	Same as method	userAge
Constant	All uppercase with underscores	MAX_SIZE

Famous Applications Built with Java

JAVA Introduction

Java powers many well-known platforms:

- **Android Apps** – Most Android apps use Java.
- **Netflix** – Backend services.
- **Amazon** – Core systems.
- **LinkedIn** – High-traffic handling.
- **Minecraft** – Entirely built in Java.
- **Spotify** – Server-side infrastructure.
- **Uber** – Trip management backend.
- **NASA WorldWind** – Virtual globe software.

What Can You Build with Java?

Java is used across industries and domains:

Domain	Use Cases
Mobile Apps	Android development via Android Studio
Web Development	Spring Boot, Jakarta EE
Desktop GUI	JavaFX, Swing
Enterprise Systems	Banking, ERP software
Game Development	LibGDX, jMonkeyEngine
Big Data	Hadoop, Apache Kafka
IoT	Embedded systems, smart devices
Cloud Services	AWS, Azure, Google Cloud
Scientific Tools	Data processing, simulations

Pro Tips for Learners

JAVA Introduction

- **Start with simple programs** like Hello World, then move to loops and conditionals.
- **Use comments** to explain your logic.
- **Follow naming conventions** for clarity.
- **Practice compiling and running** Java programs manually to understand the flow.
- **Explore libraries** like java.util, java.io, and java.math.
- **Use IDEs** like IntelliJ IDEA or Eclipse for better productivity.