Module 1: Introduction to Java Fundamentals

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

December 15, 2025

Module 1: Introduction to Java Fundamentals

- OOP Paradigm and Features of Java
- JVM, Bytecode, Java Program Structure
- Data Types, Variables, Naming Conventions
- Operators, Control and Looping Constructs
- One- and Multi-dimensional Arrays
- Enhanced for-loop
- Strings, StringBuffer, StringBuilder, Math Class
- Wrapper Classes

Module 1: What You Will Learn

Module 1: What You Will Learn

- Understand what Java is and how it works internally
- Explain Object-Oriented Programming and Java's features
- Describe the role of JVM and bytecode in Java execution
- Write and understand the basic structure of a Java program
- Use variables, data types, and operators correctly
- Apply control and looping constructs to solve problems
- Work with arrays and strings efficiently
- Understand memory handling using wrapper classes

What is a Programming Paradigm?

A **programming paradigm** is a style or approach to writing computer programs.

What is a Programming Paradigm?

A **programming paradigm** is a style or approach to writing computer programs.
It defines:

- How we structure our code
- How we think about solving a problem
- Rules, patterns, and principles used in programming

What is a Programming Paradigm?

A **programming paradigm** is a style or approach to writing computer programs.
It defines:

- How we structure our code
- How we think about solving a problem
- Rules, patterns, and principles used in programming

Different paradigms offer different ways of thinking:

- Like different languages used for communication
- Different tools for solving different kinds of problems

Types of Programming Paradigms

There are several major programming paradigms:

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).
- **Object-Oriented Programming (OOP)** Based on classes and objects (Java, Python, C++).

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).
- **Object-Oriented Programming (OOP)** Based on classes and objects (Java, Python, C++).
- **Functional Programming** Focus on functions and immutability (Haskell, Scala, Java Streams).

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).
- **Object-Oriented Programming (OOP)** Based on classes and objects (Java, Python, C++).
- **Functional Programming** Focus on functions and immutability (Haskell, Scala, Java Streams).
- **Event-Driven Programming** Program reacts to events (Java GUI, JavaScript, Android apps).

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).
- **Object-Oriented Programming (OOP)** Based on classes and objects (Java, Python, C++).
- **Functional Programming** Focus on functions and immutability (Haskell, Scala, Java Streams).
- **Event-Driven Programming** Program reacts to events (Java GUI, JavaScript, Android apps).
- **Declarative Programming** Specify *what* to do, not *how* to do it (SQL, HTML).

Types of Programming Paradigms

There are several major programming paradigms:

- **Procedural Programming** Step-by-step instructions (C, early Python).
- **Object-Oriented Programming (OOP)** Based on classes and objects (Java, Python, C++).
- **Functional Programming** Focus on functions and immutability (Haskell, Scala, Java Streams).
- **Event-Driven Programming** Program reacts to events (Java GUI, JavaScript, Android apps).
- **Declarative Programming** Specify *what* to do, not *how* to do it (SQL, HTML).
- **Logical Programming** Based on rules and facts (Prolog).

Object-Oriented Programming (OOP) Paradigm

What is OOP?

Object-Oriented Programming (OOP) Paradigm

What is OOP?

Object-Oriented Programming is a programming approach based on:

- **Objects** – Real-world entities represented in code
- **Classes** – Blueprints/templates for creating objects
- **Data + Functions together** – represents behavior and properties

Why Object-Oriented Paradigm?

OOP provides a structured and natural way to build software.

Why Object-Oriented Paradigm?

OOP provides a structured and natural way to build software.

- Models real-world entities (students, employees, machines, sensors)
- Makes code reusable (inheritance, classes)
- Improves readability and maintenance
- Enhances security (encapsulation)
- Suitable for large-scale applications

Why Object-Oriented Paradigm?

OOP provides a structured and natural way to build software.

- Models real-world entities (students, employees, machines, sensors)
- Makes code reusable (inheritance, classes)
- Improves readability and maintenance
- Enhances security (encapsulation)
- Suitable for large-scale applications

OOP = The foundation of modern software engineering.

Features of Java

Java is a powerful, modern, and industry-standard programming language with key features:

Features of Java

Java is a powerful, modern, and industry-standard programming language with key features:

- **Simple** – Easy to learn if you know Python
- **Object-Oriented** – Everything is based on classes and objects
- **Platform Independent** – “Write Once, Run Anywhere”
- **Secure** – No direct memory access, built-in security model
- **Robust** – Strong typing + exception handling
- **Multithreaded** – Execute multiple tasks simultaneously
- **Portable** – Works on any OS with JVM
- **High Performance** – Just-In-Time (JIT) compiler improves speed

What Does “Write Once, Run Anywhere” Mean?

Write Once, Run Anywhere (WORA) is a key feature of Java.

What Does "Write Once, Run Anywhere" Mean?

Write Once, Run Anywhere (WORA) is a key feature of Java.
It means:

- You write a Java program only once
- The same program can run on any operating system
- No need to change or rewrite the code

What Does “Write Once, Run Anywhere” Mean?

Write Once, Run Anywhere (WORA) is a key feature of Java.
It means:

- You write a Java program only once
- The same program can run on any operating system
- No need to change or rewrite the code

How is this possible?

- Java source code is compiled into **bytecode**
- Bytecode runs on the **Java Virtual Machine (JVM)**
- Every OS has its own JVM implementation

What Does “Write Once, Run Anywhere” Mean?

Write Once, Run Anywhere (WORA) is a key feature of Java.
It means:

- You write a Java program only once
- The same program can run on any operating system
- No need to change or rewrite the code

How is this possible?

- Java source code is compiled into **bytecode**
- Bytecode runs on the **Java Virtual Machine (JVM)**
- Every OS has its own JVM implementation

Result: The same Java program works on Windows, Linux, macOS, and more.

Does "Write Once, Run Anywhere" Apply to Python?

Short answer: Yes — but differently from Java.

Does "Write Once, Run Anywhere" Apply to Python?

Short answer: Yes — but differently from Java.
Python:

- Python code is **interpreted**, not compiled
- The same Python script can run on different OS
- **Only if the correct Python interpreter is installed**

Does "Write Once, Run Anywhere" Apply to Python?

Short answer: Yes — but differently from Java.
Python:

- Python code is **interpreted**, not compiled
- The same Python script can run on different OS
- **Only if the correct Python interpreter is installed**

Java:

- Java code is compiled into **bytecode**
- Bytecode runs on **JVM**, not directly on OS
- JVM hides all OS-level differences

Does “Write Once, Run Anywhere” Apply to Python?

Short answer: Yes — but differently from Java.

Python:

- Python code is **interpreted**, not compiled
- The same Python script can run on different OS
- **Only if the correct Python interpreter is installed**

Java:

- Java code is compiled into **bytecode**
- Bytecode runs on **JVM**, not directly on OS
- JVM hides all OS-level differences

Conclusion:

- Python → *Platform-dependent interpreter*
- Java → *Platform-independent bytecode via JVM*

Java Virtual Machine (JVM)

What is JVM?

Java Virtual Machine (JVM)

What is JVM?

The **Java Virtual Machine (JVM)** is a software-based engine that:

- Executes Java bytecode
- Makes Java platform-independent
- Manages memory and security

Java Virtual Machine (JVM)

What is JVM?
The **Java Virtual Machine (JVM)** is a software-based engine that:

- Executes Java bytecode
- Makes Java platform-independent
- Manages memory and security

Key Responsibilities:

- Loads and verifies bytecode
- Converts bytecode to machine code using JIT compiler
- Allocates memory and performs garbage collection
- Handles runtime errors (exceptions)

Java Virtual Machine (JVM)

What is JVM?

The **Java Virtual Machine (JVM)** is a software-based engine that:

- Executes Java bytecode
- Makes Java platform-independent
- Manages memory and security

Key Responsibilities:

- Loads and verifies bytecode
- Converts bytecode to machine code using JIT compiler
- Allocates memory and performs garbage collection
- Handles runtime errors (exceptions)

JVM = The engine that runs every Java program.

How JVM Executes a Java Program

Execution Flow:

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)
3. JVM loads the bytecode into memory (Class Loader)

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)
3. JVM loads the bytecode into memory (Class Loader)
4. Bytecode is verified for safety (Bytecode Verifier)

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)
3. JVM loads the bytecode into memory (Class Loader)
4. Bytecode is verified for safety (Bytecode Verifier)
5. JVM executes bytecode using:
    - Interpreter (line-by-line)
    - JIT (Just-In-Time) compiler for speed

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)
3. JVM loads the bytecode into memory (Class Loader)
4. Bytecode is verified for safety (Bytecode Verifier)
5. JVM executes bytecode using:
    - Interpreter (line-by-line)
    - JIT (Just-In-Time) compiler for speed
6. Garbage Collector frees unused memory

How JVM Executes a Java Program

Execution Flow:

1. You write a `.java` program (source code)
2. Java compiler (`javac`) converts it into `.class` (bytecode)
3. JVM loads the bytecode into memory (Class Loader)
4. Bytecode is verified for safety (Bytecode Verifier)
5. JVM executes bytecode using:
    - Interpreter (line-by-line)
    - JIT (Just-In-Time) compiler for speed
6. Garbage Collector frees unused memory

JVM ensures Java runs the same way on Windows, Linux, macOS, and more.

What is Bytecode?

Bytecode is an intermediate form of Java code generated after compilation.

What is Bytecode?

Bytecode is an intermediate form of Java code generated after compilation.

- Written Java program → `.java` file
- Java compiler (`javac`) converts it into `.class` file
- This `.class` file contains **Bytecode**

What is Bytecode?

Bytecode is an intermediate form of Java code generated after compilation.

- Written Java program → `.java` file
- Java compiler (`javac`) converts it into `.class` file
- This `.class` file contains **Bytecode**

Important:

- Bytecode is **not machine-specific**
- Bytecode is understood only by the JVM

What is Bytecode?

Bytecode is an intermediate form of Java code generated after compilation.

- Written Java program → `.java` file
- Java compiler (`javac`) converts it into `.class` file
- This `.class` file contains **Bytecode**

Important:

- Bytecode is **not machine-specific**
- Bytecode is understood only by the JVM

Bytecode acts as a bridge between Java source code and the operating system.

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

- **Platform Independence** Same bytecode runs on any system with a JVM.

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

- **Platform Independence** Same bytecode runs on any system with a JVM.
- **Security** JVM verifies bytecode before execution.

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

- **Platform Independence** Same bytecode runs on any system with a JVM.
- **Security** JVM verifies bytecode before execution.
- **Portability** No need to recompile Java programs for different OS.

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

- **Platform Independence** Same bytecode runs on any system with a JVM.
- **Security** JVM verifies bytecode before execution.
- **Portability** No need to recompile Java programs for different OS.
- **Performance Optimization** JVM uses JIT compiler to optimize frequently executed bytecode.

Why Does Java Use Bytecode?

Java uses bytecode to achieve:

- **Platform Independence** Same bytecode runs on any system with a JVM.
- **Security** JVM verifies bytecode before execution.
- **Portability** No need to recompile Java programs for different OS.
- **Performance Optimization** JVM uses JIT compiler to optimize frequently executed bytecode.

Bytecode is the core reason Java follows **"Write Once, Run Anywhere."**

What Does Java Bytecode Look Like?

Java Source Code

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

What Does Java Bytecode Look Like?

Java Source Code

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

Equivalent Bytecode Instructions

```
getstatic System.out
ldc "Hello World"
invokevirtual println
return
```

What Does Java Bytecode Look Like?

Java Source Code

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

Equivalent Bytecode Instructions

```
getstatic System.out
ldc "Hello World"
invokevirtual println
return
```

Note: Bytecode is not human-readable code. It is a set of instructions understood by JVM.

Bytecode vs Machine Code

Feature	**Bytecode**	**Machine Code**
Generated by	Java Compiler (`javac`)	OS / Hardware Compiler
Platform Dependent	No	Yes
Execution	Executed by JVM	Executed directly by CPU
Portability	High	Low
Security Checks	Yes (Bytecode Verifier)	No
Used in	Java	C, C++, Assembly

Java Program Execution Flow

1. Write Java source code (`.java`)

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`
3. Bytecode generated (`.class`)

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`
3. Bytecode generated (`.class`)
4. JVM loads and verifies bytecode

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`
3. Bytecode generated (`.class`)
4. JVM loads and verifies bytecode
5. JVM executes bytecode using Interpreter and JIT

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`
3. Bytecode generated (`.class`)
4. JVM loads and verifies bytecode
5. JVM executes bytecode using Interpreter and JIT
6. Program runs on the system

Java Program Execution Flow

1. Write Java source code (`.java`)
2. Compile using `javac`
3. Bytecode generated (`.class`)
4. JVM loads and verifies bytecode
5. JVM executes bytecode using Interpreter and JIT
6. Program runs on the system

Bytecode is the heart of Java's execution model.

Basic Structure of a Java Program

Every Java program follows a fixed structure.

Basic Structure of a Java Program

Every Java program follows a fixed structure.

- Program must be written inside a **class**
- Execution starts from the **main()** method
- Statements end with a semicolon (;)

Basic Structure of a Java Program

Every Java program follows a fixed structure.

- Program must be written inside a **class**
- Execution starts from the **main()** method
- Statements end with a semicolon (;)

Basic components:

- Class declaration
- main() method
- Statements inside main()

Hello World Program in Java

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

Understanding Hello World Program

- **public –** Accessible from anywhere

Understanding Hello World Program

- **public –** Accessible from anywhere
- **class –** Blueprint of the program

Understanding Hello World Program

- **public –** Accessible from anywhere
- **class –** Blueprint of the program
- **HelloWorld** – Class name (same as file name)

Understanding Hello World Program

- **public –** Accessible from anywhere
- **class –** Blueprint of the program
- **HelloWorld** – Class name (same as file name)
- **main() –** Entry point of Java program

Understanding Hello World Program

- **public** – Accessible from anywhere
- **class** – Blueprint of the program
- **HelloWorld** – Class name (same as file name)
- **main()** – Entry point of Java program
- **String[] args** – Command-line arguments

Understanding Hello World Program

- **public** – Accessible from anywhere
- **class** – Blueprint of the program
- **HelloWorld** – Class name (same as file name)
- **main()** – Entry point of Java program
- **String[] args** – Command-line arguments
- **static** – No object required to run main()

Understanding Hello World Program

- **public** – Accessible from anywhere
- **class** – Blueprint of the program
- **HelloWorld** – Class name (same as file name)
- **main()** – Entry point of Java program
- **String[] args** – Command-line arguments
- **static** – No object required to run main()
- **void** – No return value

Understanding Hello World Program

- **public** – Accessible from anywhere
- **class** – Blueprint of the program
- **HelloWorld** – Class name (same as file name)
- **main()** – Entry point of Java program
- **String[] args** – Command-line arguments
- **static** – No object required to run main()
- **void** – No return value
- **System.out.println()** – Prints output to screen

Hello World: Java vs Python

Python	**Java**

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

No class or main() required	Class and main() are mandatory
Interpreted directly by Python interpreter	Compiled to bytecode and executed via JVM

How to Compile and Run a Java Program

Step 1: Compile

```
javac HelloWorld.java
```

How to Compile and Run a Java Program

Step 1: Compile

```
javac HelloWorld.java
```

Step 2: Run

```
java HelloWorld
```

How to Compile and Run a Java Program

Step 1: Compile

```
javac HelloWorld.java
```

Step 2: Run

```
java HelloWorld
```

Output:

```
Hello, World!
```

Thank You!

Stay Connected

Premanand S

Email:	premanand.s@vit.ac.in
Phone:	+91-7358679961
LinkedIn:	linkedin.com/in/premsanand
Instagram:	instagram.com/premsanand
WhatsApp Channel:	anandsDataX
Google Scholar:	Google Scholar Profile
GitHub:	github.com/anandprems