Module 01 — Welcome to Java

ITP 120 — Java Programming I

Northern Virginia Community College
Randy Michak

🎯 Learning Objectives

By the end of this module, you will be able to:

- Explain what Java is, where it came from, and why it is widely used
- Describe how Java turns source code into a running program
- Explain the difference between the JDK, JRE, and JVM
- Set up a Java environment and run programs from the command line
- Write and explain a Hello World program line by line
- Identify Java's core syntax rules: case sensitivity, semicolons, and curly braces
- Distinguish between syntax errors, runtime errors, and logic errors
- Describe the programming cycle: design, write, compile, test, debug

1. What Is Java?

Java is a programming language — one of the most widely used on the planet. You will use it to build applications, process data, write server-side logic, and, starting right now, learn how programming actually works.

Java was created by James Gosling at Sun Microsystems and first released in 1995. The central idea: write code once, run it anywhere. Sun Microsystems was acquired by Oracle in 2010, and Oracle maintains Java today.

📖 **Key Terms**

- **Java** — A general-purpose, object-oriented programming language released in 1995
- **James Gosling** — The primary designer of Java, sometimes called the "father of Java"
- **Sun Microsystems** — The company that created Java; acquired by Oracle in 2010
- **Oracle** — The company that currently owns and maintains Java

Why Java?

There are hundreds of programming languages. Here is why Java is worth your time this semester:

- **It runs everywhere.** Java is on billions of devices — phones, servers, ATMs, smart cards, Blu-ray players. Android apps were originally written in Java.
- **Job market.** Java consistently ranks in the top three most in-demand programming languages. Companies like Amazon, Google, Netflix, and LinkedIn run Java on their back-end servers.
- **It builds good habits.** Java forces you to be explicit and structured. Those habits transfer to every other language you learn afterward.
- **Platform independence.** You write one program, and it runs on Windows, Mac, or Linux without changing a single line of code.

🧠 **Think of It Like...**

Learning Java is like learning to drive a manual transmission. It takes more focus up front, but you come out understanding how cars actually work. After Java, picking up Python or JavaScript feels easy. And it makes you a more attractive hire.

2. How Java Works

Computers do not understand English or Java. They only understand binary — ones and zeros. Java uses a two-step process: **compilation** into bytecode, then **execution by the JVM**.

Step 1: Write Source Code

You type your program into a file with a `.java` extension. This is called **source code** — the human-readable version of your program. A file named `Hello.java` is a source file.

Step 2: Compile to Bytecode

You run a tool called `javac` (the Java compiler). It reads your `.java` file and produces a `.class` file containing **bytecode**. Bytecode is not machine code, and not source code — it is a platform-neutral middle format.

Step 3: The JVM Executes the Bytecode

The **Java Virtual Machine (JVM)** reads the bytecode and translates it into instructions your specific computer can execute. Every operating system has its own JVM. That is what makes "write once, run anywhere" real.

> 🧠 **Think of It Like...**
>
> Imagine writing a recipe in English. A translator converts it into a universal recipe format (bytecode). Then any chef in the world — French, Japanese, Brazilian — reads that format and cooks it in their own kitchen (the JVM). Your original recipe runs anywhere without rewriting it.

```
Source Code          Compiler          Bytecode          JVM Executes
Hello.java   --->    javac     --->  Hello.class  --->   java Hello
(you write)       (you run javac)  (auto-created)    (you run java)
```

📖 **Key Terms**

- **Source code** — Human-readable Java code stored in a `.java` file
- **Compiler (`javac`)** — Translates source code into bytecode
- **Bytecode** — Platform-neutral code stored in a `.class` file
- **JVM (Java Virtual Machine)** — Reads bytecode and runs it on your specific machine
- **Write once, run anywhere** — One compiled program runs on any OS with a JVM installed

How Is This Different from C?

In a language like C, the compiler translates source code directly to machine code for one specific OS. A C program compiled on Windows will not run on Linux without recompiling. Java's bytecode plus JVM solves that — the same `.class` file runs on any platform with a JVM.

💡 **Tip**

The `.class` file is generated automatically by the compiler. You will never edit it. If you see `Hello.class` appear in your folder after running `javac`, that is correct — it is supposed to be there.

3. The Java Development Kit (JDK)

Before you can write and run Java programs, you need the **JDK** (Java Development Kit) installed. You will hear three acronyms: JDK, JRE, and JVM. They nest inside each other like containers.

Term	Full Name	What It Contains	Who Needs It
JVM	Java Virtual Machine	Runs bytecode on your specific machine	Anyone running Java programs

Term	Full Name	What It Contains	Who Needs It
JRE	Java Runtime Environment	JVM + standard libraries for running programs	End users who just run Java apps
JDK	Java Development Kit	JRE + `javac` compiler + dev tools	Developers — that is you

You always install the **JDK**. It contains the JRE, which contains the JVM. Think of nesting containers: JVM smallest, inside JRE, which is inside JDK.

> 🧠 **Think of It Like...**
>
> The JVM is the car engine. The JRE is the full car — engine, body, wheels, everything you need to drive. The JDK is the car plus the mechanic's full toolbox to build and fix it. Since you are building programs, you need the JDK.

What Is Inside the JDK?

- **`javac`** — The compiler. Turns `.java` source files into `.class` bytecode.
- **`java`** — The launcher. Starts the JVM and runs your compiled program.
- **`jar`** — Packages compiled files into a single distributable archive.
- **`javadoc`** — Generates HTML documentation from source code comments.
- **Standard class library** — Thousands of pre-built classes for I/O, math, networking, and more.

> 💡 **Tip**
>
> Install **JDK 17** or later. JDK 17 is a Long-Term Support (LTS) release, meaning it gets security updates for years. Download it free from **adoptium.net** (Eclipse Temurin). Works on Windows, Mac, and Linux.

4. Setting Up Your Environment

You have two main options: an **IDE** or the **command line**. You will use both this semester.

IDEs

An IDE (Integrated Development Environment) combines code editor, compiler, debugger, and project manager in one program. Your main options:

- **IntelliJ IDEA** — The most popular Java IDE. Community Edition is free. Recommended for this course.
- **Eclipse** — Free and open-source. Common in enterprise environments.
- **VS Code** — Lightweight editor with a Java extension pack.

Not ready to install? **Replit** (replit.com) and **JDoodle** (jdoodle.com) run Java in your browser with zero setup.

> 💡 **Tip**
>
> For the first few modules, use the command line instead of an IDE. The command line shows you exactly what happens at each step. IDEs hide that behind buttons. Understand the steps first, and the IDE will make much more sense afterward.

Compile and Run from the Command Line

```
# Step 1: Compile your source file
javac Hello.java

# Step 2: Run the compiled program (no .class extension)
java Hello
```

> ⚠️ **Common Mistake**
>
> Use `java Hello`, not `java Hello.class`. Java already knows to look for a `.class` file. Including the extension gives you: *Error: Could not find or load main class Hello.class*. Leave the extension off every time.

Verify Your Installation

```
java -version
openjdk version "17.0.9" 2023-10-17

javac -version
javac 17.0.9
```

Version numbers mean you are ready. `command not found` means the JDK is not on your PATH — bring that to class and we will fix it together.

5. Your First Program

Every programming course starts with Hello World. The program is trivial — the point is learning the structure Java requires. Write it, compile it, run it, then take it apart line by line.

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

Save this as `Hello.java`, then run:

```
javac Hello.java
java Hello
Hello, World!
```

You just ran your first Java program. Now let's break down every single line.

Line 1: `public class Hello {`

This declares a **class** named `Hello`. In Java, all code lives inside a class — no exceptions. The keyword `public` means this class is visible and accessible from anywhere. The opening curly brace `{` marks where the class body begins.

> ⚠️ **Critical Rule: File Name Must Match Class Name**
>
> The file name must exactly match the class name, including capitalization. Class named `Hello`? File must be `Hello.java`. Class named `MyProgram`? File must be `MyProgram.java`. Java will not compile if these differ — even one wrong capital letter breaks it.

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

This is the **main method**. When you run `java Hello`, the JVM searches for this exact signature and starts execution there. Memorize it. Write it exactly this way every time.

Keyword	What It Means Right Now
`public`	The JVM can call this method from outside the class
`static`	The JVM can call it without creating a class object first
`void`	This method does not return any value when it finishes
`main`	The specific name the JVM looks for as the entry point
`String[] args`	Accepts command-line arguments (covered in later modules)

> 💡 **Tip**
>
> Do not stress about fully understanding `static` and `String[] args` right now. Those build up over time. For now, treat `public static void main(String[] args)` as a required phrase you write exactly this way at the start of every program.

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

This prints a line of text to the console. Here is what each piece does:

- `System` — A built-in Java class representing the running environment
- `out` — The output stream connected to your terminal window
- `println` — Short for "print line" — prints text and then moves the cursor to the next line
- `"Hello, World!"` — The text to display, in double quotes (a *String literal*)
- `;` — The semicolon ending the statement

Lines 4 and 5: The Closing Braces

The two closing braces close the main method and then the class, in that order. Every opening brace needs a matching closing brace. Count them — mismatched braces are one of the most common beginner mistakes.

> 📖 **Key Terms**
>
> - **Class** — A container that holds code; all Java programs live inside at least one class
> - **Method** — A named block of code that performs a specific job
> - **main method** — The entry point of every Java application; execution starts here
> - **System.out.println()** — Prints text to the console and moves to the next line
> - **String literal** — Text enclosed in double quotes, like `"Hello, World!"`

> 🔧 **Try It Yourself**
>
> Get Hello World running, then experiment:
>
> 1. Change the text to print your own name: `"Hello, I'm [your name]!"`
> 2. Add a second `System.out.println()` inside main. What happens to the output?
> 3. Try `System.out.print()` instead of `System.out.println()`. How does the output differ?
> 4. Rename the file to `hello.java` (lowercase h) and try to compile. What error appears?

6. Java Syntax Basics

Java enforces its rules without exceptions. Break them and the compiler stops and tells you exactly where. Here are the four core syntax rules you need right now.

Rule 1: Java Is Case-Sensitive

Uppercase and lowercase letters are completely different. `System` is not `system`. `main` is not `Main`. This applies to class names, method names, variable names, and all built-in keywords.

```
// CORRECT
System.out.println("Hello");

// WRONG - lowercase 's' on System causes a compile error
system.out.println("Hello");
```

Rule 2: Every Statement Ends with a Semicolon

A semicolon is the period at the end of a sentence. Every executable statement needs one. Forget it and the compiler flags the next line with a confusing error.

```
// CORRECT
System.out.println("Hello");

// WRONG - missing semicolon causes a compile error on the line below
System.out.println("Hello")
```

Rule 3: Curly Braces Define Blocks

Opening braces `{` and closing braces `}` mark the start and end of code blocks for classes, methods, loops, and conditions. Every opening brace must have a matching closing brace. Count them carefully.

```
// CORRECT - balanced braces
public class Demo {
    public static void main(String[] args) {
        System.out.println("Works!");
    }   // closes method
}       // closes class

// WRONG - missing closing brace for the class
public class Demo {
    public static void main(String[] args) {
        System.out.println("Works!");
    }
// Error: reached end of file while parsing
```

Rule 4: Whitespace Does Not Matter (But Use It Anyway)

Java ignores extra spaces, tabs, and blank lines. Both of these compile identically — but only one of them is acceptable:

```
// Compact (valid, but completely unreadable)
public class Demo{public static void main(String[]args)
{System.out.println("Hi");}}

// Properly indented (also valid, and what every professional expects)
public class Demo {
    public static void main(String[] args) {
        System.out.println("Hi");
    }
}
```

Indentation is not optional in practice. Use it every single time.

> ⚠️ **Most Common Syntax Mistakes**
>
> - Using `system` instead of `System` (wrong capitalization)
> - Forgetting the semicolon at the end of a statement
> - Mismatched or missing curly braces
> - Using single quotes for strings — Java strings need double quotes: `"text"` not `'text'`

7. Errors

You are going to see error messages. A lot of them. Even experienced developers see error messages every single day. What matters is learning to read them and fix them quickly. There are three types of errors in Java.

Type 1: Syntax Errors (Compile-Time Errors)

A **syntax error** is a mistake in the structure of your code. Missing semicolon, misspelled keyword, unbalanced brace — these all cause syntax errors. Java catches them during compilation. Your program will not compile at all until they are fixed.

```
// WRONG - missing semicolon after println
public class Demo {
    public static void main(String[] args) {
        System.out.println("Hello")
    }
}
Demo.java:3: error: ';' expected
        System.out.println("Hello")
                                   ^
1 error
```

That error message tells you exactly what is wrong and exactly where. Line 3. Missing semicolon. Fix it there.

> 💡 **How to Read a Compiler Error**
>
> The format is: `Filename.java:LineNumber: error: description`. The caret (`^`) points to where Java got confused. Go to that line first. Fix the most-reported error first and recompile — one mistake can cause many error messages to cascade.

Type 2: Runtime Errors

A **runtime error** happens while the program is running, after it has compiled successfully. Your code is syntactically correct, but something goes wrong during execution. A classic example is dividing by zero or trying to use a null value.

```
public class Demo {
    public static void main(String[] args) {
        int x = 10;
        int y = 0;
        System.out.println(x / y);  // dividing by zero at runtime
    }
}
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Demo.main(Demo.java:5)
```

The program compiled fine. It crashed when it tried to execute line 5. The error message tells you which line, what went wrong, and what kind of exception it was.

Type 3: Logic Errors

A **logic error** is the sneakiest type. The program compiles and runs without crashing — but it produces the wrong result. Java cannot catch logic errors because the code is technically valid. You have to catch them by testing carefully.

```
public class Demo {
    public static void main(String[] args) {
        int total = 5 + 3;
        System.out.println("The answer is: " + total);
    }
}
The answer is: 8
```

If you wanted `5 * 3` but typed `5 + 3`, the program runs fine and prints 8. Java has no way to know 8 was wrong — only you know the expected result. Logic errors are fixed through testing and debugging.

Error Type	When It Occurs	Example	How to Fix
Syntax error	At compile time	Missing semicolon	Read the compiler error message and fix the flagged line
Runtime error	While running	Divide by zero	Read the exception message; check input values
Logic error	While running (no crash)	Wrong calculation	Test your program with known inputs and expected outputs

> **🔧 Try It Yourself**
>
> Practice causing errors intentionally so you know what they look like:
>
> 1. Remove the semicolon from your Hello World `println` line. Read the compiler error.
> 2. Rename the class to `hello` (lowercase) but keep the file as `Hello.java`. What happens?
> 3. Delete one closing brace. What error message appears?
> 4. Change `System` to `system`. What does the compiler say?

8. The Programming Process

Writing code is not just typing until it works. Professional programmers follow a process. Every step matters, and skipping steps leads to bugs that are painful to track down later.

The Five-Step Cycle

Step 1: Design
Before writing a single line of code, think about what your program needs to do. What inputs does it receive? What output should it produce? What steps will get you from input to output? Write it out in plain English or a simple diagram. This is called an *algorithm* — a step-by-step plan for solving a problem.

Step 2: Write
Translate your design into Java code. Focus on one piece at a time. Do not try to write everything at once. Write a small chunk, think about whether it matches your design, then continue.

Step 3: Compile
Run `javac` and see if your code compiles. Fix any syntax errors the compiler reports. If there are multiple errors, fix them from the top down — one early error can cause a cascade of fake errors below it.

Step 4: Test
Run your program and verify it does what you intended. Test with different inputs. Try

edge cases — what happens with zero, negative numbers, empty input? Does the output match what you designed?

Step 5: Debug
If the output is wrong, you have a logic error. Read your code carefully. Add extra `println` statements to print intermediate values and see where things go off track. This is called *debugging* — the process of finding and fixing errors in a working program.

> 🧠 **Think of It Like...**
>
> Building a bookshelf: you start with a design and measurements (Step 1), then cut the wood (Step 2), then test that the pieces fit together (Step 3-4), and sand down anything that does not line up (Step 5). Skipping the design step and just cutting wood is how you end up with a pile of expensive lumber and no shelf.

> 💡 **Tip**
>
> Do not write 50 lines of code and then try to compile. Write 5-10 lines, compile and test, then continue. Small, frequent compilations mean smaller problems to fix when something goes wrong. Beginners often write large amounts of code and then face an overwhelming wall of errors.

🔧 Try It Yourself: Full Cycle

Follow the five steps for this simple program:

Goal: Write a program that prints your full name, your major, and the current semester on three separate lines.

1. **Design:** What three things does it need to print? Write them on paper.
2. **Write:** Create a new file called `MyInfo.java` with a class named `MyInfo` and three `println` statements.
3. **Compile:** Run `javac MyInfo.java`. Fix any errors.
4. **Test:** Run `java MyInfo`. Does it print exactly what you planned?
5. **Debug:** If anything looks wrong, find the line and fix it.

📋 Module Summary

Module 01 Key Takeaways

- **Java** was created in 1995 by James Gosling at Sun Microsystems; now maintained by Oracle
- **How Java works:** source code (`.java`) → compiler (`javac`) → bytecode (`.class`) → JVM runs it
- **"Write once, run anywhere"** means one compiled program runs on any OS with a JVM
- **JDK** contains the JRE, which contains the JVM; always install the JDK for development
- **Compile:** `javac Hello.java` — **Run:** `java Hello` (no `.class` extension)
- **File name must match class name** exactly, including capitalization
- **Every program needs** a class and a `public static void main(String[] args)` method
- **Core syntax rules:** Java is case-sensitive; statements end with `;` ; braces must balance
- **Three error types:** syntax (compile-time), runtime (crash), logic (wrong output)
- **Programming cycle:** Design → Write → Compile → Test → Debug

Vocabulary Review

Term	Definition
Java	A general-purpose, object-oriented programming language created in 1995 by James Gosling at Sun Microsystems
Source code	Human-readable Java code written in a `.java` file
Compiler	

Term	Definition
	The tool (`javac`) that translates Java source code into bytecode
Bytecode	Platform-neutral code stored in a `.class` file; output of the Java compiler
JVM	Java Virtual Machine; reads bytecode and executes it on a specific machine
JRE	Java Runtime Environment; the JVM plus standard libraries needed to run Java programs
JDK	Java Development Kit; the JRE plus development tools including the compiler
Write once, run anywhere	Java's platform independence: one compiled program runs on any OS with a JVM
Class	A container that holds code; every Java program lives inside at least one class
Method	A named block of code that performs a specific task
main method	The entry point of a Java application: `public static void main(String[] args)`
System.out.println()	Prints a line of text to the console and moves to the next line
String literal	Text enclosed in double quotes, like `"Hello, World!"`
Syntax error	A compile-time error caused by violating Java's grammar rules
Runtime error	An error that occurs while a program is running, causing it to crash
Logic error	An error where a program runs without crashing but produces incorrect results

Term	Definition
Algorithm	A step-by-step plan for solving a problem
Debugging	The process of finding and fixing errors in a program

Knowledge Check

Answer each question below. Choose the single best answer for each.

1. Who created Java, and in what year was it first released?

A. James Gosling, 1995

B. Linus Torvalds, 1991

C. Bill Gates, 1990

D. Guido van Rossum, 2000

2. Which file extension does Java *source code* use?

A. `.exe`

B. `.java`

C. `.jar`

D. `.class`

3. What does the Java compiler (`javac`) produce when it compiles a `.java` file?

A. A machine code executable that runs directly on the CPU

B. Bytecode stored in a `.class` file

C. A new `.java` file with errors removed

D. A compressed archive that the JRE executes

4. Which of the following best describes the JDK?

A. A program that only runs bytecode; it does not compile

B. A lightweight viewer for reading Java documentation

C. A subset of the JRE that contains only the JVM

D. A full development toolkit containing the compiler, JRE, and development tools

5. A student saves a class named `MyProgram` in a file called `myprogram.java`. What will happen when they try to compile?

A. The compiler reports an error because the file name does not match the class name

B. Java automatically corrects the file name and compiles

C. The compiler produces a warning but still compiles

D. The program compiles and runs normally

6. A student writes the following code. What error will occur and when?

```
public class Broken {
    public static void main(String[] args) {
        System.out.println("Hello")
    }
}
```

A. A runtime error — the program crashes when it tries to print

B. A logic error — the output will be incorrect

C. A syntax error — the compiler will report a missing semicolon

D. No error — semicolons are optional in Java

7. Which command correctly *runs* a compiled Java program named `Hello`?

A. `run Hello.java`

B. `javac Hello`

C. `java Hello`

D. `java Hello.class`

8. Examine the following code. It compiles and runs without crashing, but prints `15` when the correct answer should be `50`. What type of error is this?

```
public class Calc {
    public static void main(String[] args) {
        int result = 10 + 5;   // should be 10 * 5
        System.out.println(result);
    }
}
```

A. Syntax error

B. Logic error

C. Runtime error

D. Compile-time error

9. What is the correct order of the Java programming cycle?

A. Write → Compile → Test → Design → Debug

B. Compile → Design → Write → Debug → Test

C. Design → Write → Compile → Test → Debug

D. Test → Design → Compile → Write → Debug

10. Which of the following is the correct way to print text to the console in Java?

A. `print("Hello");`

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

C. `console.log("Hello");`

D. `System.out.println('Hello');`

✅ Answer Key

Question	Answer	Explanation
1	**A**	James Gosling created Java in 1995 at Sun Microsystems.
2	**B**	Java source code files use the `.java` extension.
3	**B**	The compiler produces bytecode stored in a `.class` file.
4	**D**	The JDK contains the compiler, JRE, and development tools.
5	**A**	The file name must exactly match the class name, including capitalization.
6	**C**	A missing semicolon is a syntax error caught at compile time.
7	**C**	`java Hello` runs the compiled program (no `.class` extension).
8	**B**	The program runs but gives wrong output — that is a logic error.
9	**C**	Design → Write → Compile → Test → Debug.
10	**B**	`System.out.println("Hello");` is the correct syntax.