Document Generated: 12/15/2025

Learning Style: Virtual Classroom

Technology: Java

Difficulty: Beginner

Course Duration: 5 Days

Getting Started with Programming, OO & Java Basics for Non-Developers (TT2000)

About this course:

Learning to program opens up a world of possibilities, whether you are looking to build applications, improve your problem-solving skills, or just understand how software works. Getting Started with Programming, OO, and Java 21 Basics for

Non-Developers is a hands-on, expert-led course designed to make coding approachable, even if you have never written a line of code before. You will learn how programs work, how to think like a developer, and how to write and organize Java code in a way that makes sense. With plenty of hands-on practice, you will gain confidence using Java 21’s latest features, working with IDEs, and understanding key concepts like variables, loops, methods, and object-oriented programming.

The average salary of a Java Developer is **$90,992** per year.

Course Objective:

In this course, you will gain the essential programming skills needed to write, structure, and troubleshoot Java applications with confidence. By the end, you will be able to:

- Write and run Java programs using IntelliJ (or Eclipse, if requested) and the Java Development Kit (JDK) to understand how code is compiled and executed.
- Use variables, loops, conditionals, and methods to control program flow and manage data efficiently.
- Apply object-oriented programming principles, including classes, objects, inheritance, and polymorphism, to design well-structured applications.
- Work with Java's core features, such as arrays, strings, exceptions, and collections, to build functional and organized code.
- Understand and apply best practices in writing clean, reusable, and maintainable Java code.
- Build confidence in troubleshooting errors, debugging programs, and thinking like a developer.

Audience:

- Technically-minded attendees who want or who want to begin the process of becoming an OO application developer
- Technical team members from non-development roles, re-skilling to move into software and application development roles within an organization
- Recent college graduates looking to apply their college experience to programming skills in a professional environment, or perhaps needing to learn the best practices and standards for programming within their new organization
- Technical managers tasked with overseeing programming teams, or

development projects, where basic coding knowledge and exposure will be useful in project oversight or communications needs

Prerequisites:

- Basic computer literacy: Familiarity with computer operating systems, file management, and general navigation to ensure a smooth learning experience.

- Foundational knowledge of IT concepts: Understanding of essential IT terminologies and concepts, such as computer networks, software applications, and data storage.

- Analytical thinking: Ability to analyze problems and think critically to develop logical solutions, fostering a programmer's mindset.

Course Outline:

1. Overview of Computer Programming

- Explain what a program is
- Explain why there are different types of languages
- Explain what a compiler is
- Explain what an interpreter is
- Lab: Matching Terms

2. Features of a Program

- Understand what the entry and exit points of an application are
- Explain what variables are
- Explain what programming instructions are
- Explain what errors and exceptions are
- Understand what programming algorithms are

3. Software Development Life Cycle

- Explain the purpose of the software development life cycle
- Explain what each phase is for
- Explain the difference between the software development life cycle and a methodology

4. Thinking in Objects

- Understand the difference between a class and an object
- Deconstruct an object into attributes and operations

- Map an object to a class
- Define inheritance
- Lab: Designing an Application

5. The Java Platform

- Introduce the Java Platform
- Explore the Java Standard Edition
- Discuss the lifecycle of a Java Program
- Explain the responsibilities of the JVM
- Executing Java programs
- Garbage Collection
- Documentation and Code Reuse

6. Using the JDK

- Explain the JDK’s file structure
- Use the command line compiler to compile a Java class
- Use the command line Java interpreter to run a Java application class
- Lab: Exploring MemoryViewer

7. The IntelliJ Paradigm

- Introduce the IntelliJ IDE
- The Basics of the IntelliJ interface
- IntelliJ Projects and Modules
- Creating and running Java applications
- Tutorial: Working with IntelliJ 2023.2 (Community Edition)

8. Writing a Simple Class

- Write a Java class that does not explicitly extend another class
- Define instance variables for a Java class
- Create object instances
- Primitives vs Object References
- Implement a main method to create an instance of the defined class
- Java keywords and reserved words
- Lab: Create a Simple Class

9. Adding Methods to the Class

- Write a class with accessor methods to read and write instance variables
- Write a constructor to initialize an instance with data
- Write a constructor that calls other constructors of the class to benefit from code reuse
- Use the this keyword to distinguish local variables from instance variables
- Lab: Create a Class with Methods

10. Exploring Object-Oriented Programming

- Real-World Objects
- Classes and Objects
- Object Behavior
- Methods and Messages
- Lab: Define and use a New Java class

11. Inheritance, Abstraction, and Polymorphism

- Encapsulation
- Inheritance
- Method Overriding
- Polymorphism
- Lab: Define and use Another Java Class

12. Language Statements

- Arithmetic operators
- Operators to increment and decrement numbers
- Comparison operators
- Logical operators
- Return type of comparison and logical operators
- Use for loops
- Swtch Expressions
- Switch Expressions and yield
- Lab: Looping (optional)
- Lab: Language Statements
- Lab: Switch Expressions

13. Using Strings and Text Blocks

- Create an instance of the String class
- Test if two strings are equal
- Perform a case-insensitive equality test
- Contrast String, StringBuffer, and StringBuilder
- Compact Strings
- Text Blocks
- Unicode support
- Lab: Fun with Strings
- Lab: Using StringBuffers and StringBuilders

14. Fields and Variables

- Discuss Block Scoping Rules
- Distinguish between instance variables and method variables within a method
- Explain the difference between the terms field and variable
- List the default values for instance variables
- Final and Static fields and methods
- Lab: Field Test

15. Specializing in a Subclass

- Constructing a class that extends another class
- Implementing equals and toString
- Writing constructors that pass initialization data to parent constructor
- Using instanceof to verify type of an object reference
- Pattern matching for instanceof
- Overriding subclass methods
- Safely casting references to a more refined type
- Lab: Creating Subclasses

16. Using Arrays

- Declaring an array reference
- Allocating an array
- Initializing the entries in an array
- Writing methods with a variable number of arguments
- Lab: Creating an Array

17. Formatting Strings

- Format a String using the formatter syntax
- Apply text formatting
- Use String.format and System.out.printf
- Lab: Textblocks

18. Records

- Data objects in Java
- Introduce records as carrier of immutable data
- Defining records
- The Canonical constructor
- Compact constructors
- Lab: Records

19. Java Packages and Visibility

- Use the package keyword to define a class within a specific package
- Discuss levels of accessibility/visibility
- Using the import keyword to declare references to classes in a specific package
- Using the standard type naming conventions
- Visibility in the Java Modular System
- Correctly executing a Java application class
- The Java Modular System
- Defining Modules

20. Utility Classes

- Introduce the wrapper classes

- Explain Autoboxing and Unboxing
- Converting String representations of primitive numbers into their primitive types
- Defining Enumerations
- Using static imports
- Deprecating classes and methods
- Lab: Enumerations

21. Java Date/Time

- The Date and Calendar classes
- Introduce the new Date/Time API
- LocalDate, LocalDateTime, etc.
- Formatting Dates
- Working with time zones
- Manipulate date/time values

22. Inheritance and Polymorphism

- Write a subclass with a method that overrides a method in the superclass
- Group objects by their common supertype
- Utilize polymorphism
- Cast a supertype reference to a valid subtype reference
- Use the final keyword on methods and classes to prevent overriding
- Lab: Salaries - Polymorphism

23. Interfaces and Abstract Classes

- Define supertype contracts using abstract classes
- Implement concrete classes based on abstract classes
- Define supertype contracts using interfaces
- Implement concrete classes based on interfaces
- Explain advantage of interfaces over abstract classes
- Explain advantage of abstract classes over interfaces
- Lab: Interfaces

24. Introduction to Exception Handling

- Introduce the Exception architecture
- Defining a try/catch blocks
- Checked vs Unchecked exceptions
- Lab: Exceptions

25. Exceptions

- Defining your own application exceptions
- Automatic closure of resources
- Suppressed exceptions
- Handling multiple exceptions in one catch
- Enhanced try-with-resources

- Helpful NullPointerException(s)
- Lab: Exceptional
- Lab: Helpful Nullpointers

26. Building Java Applications

- Explain the steps involved in building applications
- Define the build process
- Introduce build scripts
- Explain the standard folder layout
- Resolving project dependencies
- Tutorial: Importing code Using Maven

27. Introduction to Generics

- Generics and Subtyping
- Bounded Wildcards
- Generic Methods
- Legacy Calls To Generics
- When Generics Should Be Used
- Lab: DynamicArray
- Lab: Adding Generics to Dynamic Array

28. Collections

- Provide an overview of the Collection API
- Review the different collection implementations (Set, List and Queue)
- Explore how generics are used with collections
- Examine iterators for working with collections
- Lab: Create a simple Game using Collections