# Core Java 25 Programming Developer's Workshop (TT2100) Eğitimi

Süre: **5 Gün**

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

## Overview

**Get hands-on with modern Java to build scalable apps, boost performance, streamline workflows, and create enterprise-ready solutions**

Throughout this five-day, hands-on Java training course, students learn the best practices for writing great object-oriented programs in Java 25, using sound development techniques, modern features for improved performance, and new capabilities that accelerate rapid application development.

This course explores key enhancements introduced since Java 11, including Records, Sealed Classes, Text Blocks, Pattern Matching, Virtual Threads, and numerous API updates. In addition, Java 25 continues the trend of simplifying the language for newcomers by providing streamlined syntax, enhanced pattern matching, and a more approachable set of defaults that make it easier for developers new to Java to quickly write correct, readable, and efficient code without getting bogged down in boilerplate.

With Java 25, learning the language as a beginner has become more approachable than ever. New features such as Compact Source Files and Instance Main Methods allow students to create and run small programs with minimal setup, eliminating much of the boilerplate code that used to overwhelm newcomers. Flexible Constructor Bodies simplify class creation by reducing strict syntax requirements, making it easier to focus on concepts instead of technical hurdles. Additionally, Pattern Matching for Primitives streamlines common coding tasks by replacing verbose casting with more natural and readable code. These improvements reflect Javaâ€™s ongoing effort to lower the learning curve, helping new developers get started faster while still building a foundation that is compatible with professional software development practices.

Developers leaving this course will be well-prepared to work on Java 8, Java 11, Java 17, and Java 21 projects, while also being ready to contribute effectively to modern projects using Java 25.

**Who Should Attend?**

This is a foundational Java programming course designed for attendees who already have prior development experience in another programming language. Participants should be familiar with basic programming concepts such as variables, control structures, functions/methods, and data structures.

## Prerequisites

Participants should be familiar with basic programming concepts such as variables, control structures, functions/methods, and data structures.

# Core Java 25 Programming Developer's Workshop (TT2100) Eğitimi

Süre: **5 Gün**

BİLGİNÇ
IT Academy
info@bilginc.com
bilginc.com

## What You Will Learn

This skills-centric course is about 50% hands-on lab and 50% lecture , designed to train attendees in basic OO coding and Java development skills, coupling the most current, effective techniques with the soundest industry practices. Throughout the course students will be led through a series of progressively advanced topics, where each topic consists of lecture, group discussion, comprehensive hands-on lab exercises, and lab review.

Our engaging instructors and mentors are highly experienced practitioners who bring years of current " on-the-job " experience into every classroom. Working in a hands-on learning environment, guided by our expert team, attendees will learn to:

- Understand what object-oriented (OO) programming is and recognize the advantages it provides in today's software development world.
- Gain a solid grasp of the fundamentals of the Java language, including its importance, uses, strengths, and limitations.
- Connect the basics of the Java language to OO programming and the Java Object Model.
- Learn to use Java's exception handling features to build more reliable applications.
- Work with the Java Modular System (Project Jigsaw) to create organized, maintainable applications.
- Design and implement classes that demonstrate inheritance and polymorphism.
- Use collections, generics, autoboxing, and enumerations to efficiently manage data.
- Process large volumes of data using lambda expressions and the Stream API.
- Define and implement abstract, static, and private methods in interfaces.
- Take advantage of Java development tooling available in modern programming environments.
- Write modern Java code using switch expressions for more concise and expressive branching logic.
- Use text blocks to create clean, multi-line string literals.
- Apply pattern matching for instanceof to write safer and more readable type checks.
- Introduce records as immutable data carriers to simplify domain models.
- Use pattern matching in switch statements to simplify conditional logic.
- Apply record patterns to deconstruct and access data directly within records.

**Specific Java 25 features that are covered in the course include:**

- Learn how to use **primitive types in patterns**, allowing pattern matching to work directly with primitive values for safer and more concise code.
- Understand **module import declarations**, which simplify modular programming by making module dependencies clearer and easier to manage.
- Explore **flexible constructor bodies**, which give developers greater control over how constructors are structured and how initialization logic is handled.
- Practice writing **compact source files and instance main methods**, which reduce boilerplate and make it easier for beginners to create simple Java applications.

## Outline

**1) The Java Platform**

Java stands out in the software world not only as a programming language but also as an adaptable

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

platform. Understanding Java means exploring its broader ecosystem: from the Standard Edition (SE) used for everyday applications, to the Enterprise tailored for large-scale systems. You will see how Java's design enables code to run on many devices and operating systems without changes, thanks to the Java Virtual Machine (JVM). You'll also get to know the essential tools provided in the Java Development Kit (JDK), which make building, debugging, and documenting Java applications possible. This lesson explains the step-by-step process of transforming Java code into running applications, the automatic memory management through garbage collection, and the dynamic runtime features that keep Java secure and efficient. Gaining these insights will equip you to understand why Java is trusted for everything from desktop software to smart devices and enterprise solutions.

- 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

**2) Using the JDK**

Understanding how Java organizes code and documentation is essential for creating reliable and maintainable applications. You will explore the structure of the Java Development Kit (JDK) and the importance of tools and APIs that support your work. Learning how to use packages, JARs, and modules will help you keep your projects organized and scalable. You will also see how to set up your Windows environment for Java development, ensuring all necessary commands and libraries are within reach. The lesson guides you through the process of compiling and running Java programs, clarifying file naming rules and output directories. You will discover how documentation, especially with Javadoc, supports code reuse and collaboration by making your codebase easier to understand and maintain.

- 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
- Documentation and Code Reuse

**3) Using the IntelliJ IDE**

IntelliJ IDEA is a powerful tool designed to make your software development workflow more efficient and enjoyable. As you use IntelliJ, you'll notice how it integrates a wide range of development tasks into one cohesive environment, from writing code and managing resources to compiling, running, and debugging your applications. The platform's advanced features, such as intelligent code assistance, robust project organization, and deep integration with build tools, version control systems, and databases, are aimed at streamlining your work and reducing repetitive tasks.

Understanding how IntelliJ organizes projects, modules, libraries, and facets will help you keep your projects tidy and scalable. You will also see how features like wizards, templates, and instant code suggestions can speed up your coding and help maintain high standards of code quality. The environment offers flexible ways to view and manage your codebase, as well as tools for customizing coding styles and navigating large projects with ease. By mastering these foundational aspects of IntelliJ, you set yourself up for more productive, reliable, and maintainable development work.

BİLGİNÇ IT Academy

info@bilginc.com

bilginc.com


- Introduce the IntelliJ IDE
- The Basics of the IntelliJ interface
- IntelliJ Projects and Modules
- Creating and running Java applications
- Tutorial: Working with IntelliJ 2025 (Community Edition)

**4) Writing a Simple Class**

Java programming centers around the concept of classes, which organize both data and functionality in a unified structure. By understanding how classes act as templates, you can efficiently define the properties and behaviors that objects will possess. This lesson explains the difference between fields and methods, clarifies how to create and initialize objects, and explores how access modifiers influence the visibility of different components in your code. You will also encounter the distinction between primitive types and object references, learn how to use the dot operator for interacting with objects, and discover the methods for displaying output to the console. Additionally, it is essential to be aware of Java's reserved words and keywords to avoid naming conflicts and maintain clear, error-free code.

- 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

**5) Adding Methods to the Class**

In this lesson, you will explore how Java harnesses the power of methods to define object behavior and manage data. You'll learn the essential structure of a method, including how modifiers, return types, parameters, and the method body come together to perform specific actions on objects. As you move forward, you'll see how parameters act as inputs, allowing methods to process information dynamically. You'll also discover how the `this` keyword helps manage variable naming conflicts and how Java uses accessors and mutators to control data access and modification, reinforcing the principles of encapsulation. The lesson covers the distinctions between overloaded and overridden methods, providing insights into how Java supports flexible and maintainable code. You'll dig into constructors, special methods that initialize objects—and see how their proper use ensures robust object creation. Finally, you'll learn about Java annotations, which add metadata to your code, and how to indicate deprecated features to guide future development.

- 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
- Introducing annotations
- Deprecating classes and methods

**6) Object-Oriented Programming**

Object-oriented programming models software around real-world concepts, allowing you to create programs that are flexible, scalable, and easy to maintain. In this lesson, you will explore how objects encapsulate both state (data) and behavior (operations), and how classes provide the templates for

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

these objects. By examining features such as interfaces, encapsulation, inheritance, and polymorphism, you will see how code can evolve and adapt to change with minimal disruption. You'll also learn how objects relate to each other, mirroring complex relationships found in real systems. Through practical examples, such as designing a television or managing vehicles, you'll gain insight into how abstraction and object relationships form the backbone of robust software solutions.

- Real-World Objects
- Classes and Objects
- Object Behavior
- Methods and Messages

**7) Language Statements**

Operators and control statements are at the heart of Java programming, providing the tools you need to manipulate data, make decisions, and repeat actions. This lesson explores a wide array of operators, from basic arithmetic to more advanced bitwise and logical operations, equipping you to write concise and effective code. You'll encounter shorthand ways to update variables, learn how comparison and logical operators drive decisions, and see how the ternary operator streamlines conditional assignments. The lesson also delves into looping constructs, including for, while, and do-while loops, as well as the mechanisms to control flow within these loops. To tackle complex branching, you'll discover how both traditional switch statements and modern switch expressions can simplify your code, allowing for clear and maintainable multi-path logic. By mastering these constructs, you strengthen your ability to craft robust, flexible, and efficient Java programs.

- Arithmetic operators
- Operators to increment and decrement numbers
- Comparison operators
- Logical operators
- Return type of comparison and logical operators
- Use for loops
- Switch Expressions
- Switch Expressions and yield

**8) Using Strings and Text Blocks**

Understanding how Java manages and manipulates strings is vital for efficient and robust programming. Strings play a fundamental role in handling text data, and Java offers a range of tools and classes to make string processing both powerful and straightforward. You will explore how strings are stored, how immutability impacts performance and behavior, and the various methods that allow for comparison, modification, and formatting. The lesson covers practical classes like StringBuffer and StringBuilder, which help with mutable strings, and introduces newer features like StringJoiner and text blocks for more readable and manageable code. Important distinctions in string comparison and whitespace management will also be addressed. Additionally, you will see how Java has evolved to optimize memory usage for strings and how its Unicode support keeps pace with the world’s languages and symbols. With these concepts, you will be prepared to write clear, efficient, and modern Java code that handles text seamlessly.

- Create an instance of the String class
- Test if two strings are equal
- Perform a case-insensitive equality test

BİLGİNÇ IT Academy

info@bilginc.com

bilginc.com


- Contrast String, StringBuffer, and StringBuilder
- Compact Strings
- Text Blocks
- Unicode support

**9) Fields and Variables**

Understanding how variables operate in Java is crucial for writing reliable and maintainable code. You'll explore the specific roles that fields, local variables, and instance variables play within classes and methods. By examining Java's data types, you'll see how primitive values, object references, and arrays are stored and manipulated. The lesson covers how variables are initialized, why block scoping matters, and how naming conflicts are resolved using the `this` keyword. You'll also learn about Java's type inference capabilities using `var`, including best practices and limitations. Finally, you'll gain insight into the importance of static and final variables, and how static methods are used to provide utility functions without requiring object creation. Each concept builds a foundation for writing robust, clear, and efficient Java programs.

- 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

**10) Specializing in a Subclass**

Inheritance is a cornerstone of object-oriented programming in Java. By extending a class, you can build on existing functionality, reducing repetition and making your code easier to maintain. You'll see how to use the `extends` keyword to create subclasses that inherit data and behaviors from superclasses, and how to customize these behaviors by overriding methods. This lesson also covers how Java handles object references through upcasting and downcasting, and how type checking with `instanceof` and pattern matching can make your code safer and clearer. Since every class in Java is ultimately derived from the Object class, you'll learn the importance of overriding its fundamental methods for equality, string representation, and hashing. Constructor chaining and the correct use of constructors are also explored to ensure that objects are initialized properly in complex class hierarchies. By mastering these concepts, you'll gain the tools needed to structure robust, flexible, and maintainable Java applications.

- 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
- Overriding subclass methods
- Pattern matching for instanceof
- Safely casting references to a more refined type

**11) Using Arrays**

Arrays in Java offer a way to organize and manage a collection of data items efficiently. By understanding how arrays work, you can create structures that hold fixed sequences of elements, all sharing the same data type. This lesson covers how to declare and allocate arrays, fill them with data,

BİLGİNÇ IT Academy

info@bilginc.com

bilginc.com


and access individual elements using index positions. You will also see how multidimensional arrays can be set up to represent more complex data arrangements. Managing array data often involves copying or expanding arrays, and you'll learn how Java's built-in methods streamline these tasks. Methods sometimes need to accept a flexible number of inputs, and arrays are key to supporting this, both through manual approaches and the use of Java's varargs feature. Key rules, such as array bounds and argument placement, are highlighted to help you avoid common errors and write robust, adaptable code.

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

**12) Records**

In this lesson, you will explore how Java models data using both traditional data objects and the newer record feature. You'll begin by examining the standard patterns and code requirements for creating data objects, including constructors, getters, and essential methods like `equals`, `hashCode`, and `toString`. As you progress, you'll see how modern IDEs can automate much of this task but still require careful management to avoid errors.

The lesson then introduces Java records, a powerful feature designed to simplify the creation of immutable data classes. You'll learn how records automatically generate much of the necessary code, their structure, and how to define them efficiently. By working with canonical, alternative, and compact constructors, you'll uncover ways to validate and initialize record data. Additionally, you'll see how records can be extended with custom and static methods, while remaining immutable and concise. Finally, you'll examine how generic records increase flexibility and reusability in your code, enabling you to create versatile data containers with type safety.

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

**13) Java Packages and Visibility**

When programming in Java, organizing your code is essential for building scalable and maintainable software. You will encounter situations where the same class name can mean very different things in different contexts, leading to potential conflicts and confusion. Java provides powerful tools to address these issues, including packages, access modifiers, and modules. Packages help you logically group related classes, avoid naming clashes, and maintain a tidy project structure. Access modifiers control what parts of your code can interact with each other, helping secure sensitive data and internal logic. As your projects grow, Java's modular system allows you to bundle groups of packages together, control what gets shared, and manage dependencies efficiently. Throughout this lesson, you will see how these features work together to produce organized, clear, and professional Java code. Following naming conventions will also improve code readability and collaboration across teams.

- 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

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

- Using the standard type naming conventions
- Introduce the Java Modular System
- Visibility in the Java Modular System

**14) Running Java applications**

Java 25 brings a fresh approach that makes it easier than ever to start programming, whether you are brand new or already have experience. In the past, setting up your first Java program often meant wrestling with lots of structure and rules before you could even see anything happen on screen. Now, you can focus on the most important part, what your code does, rather than how to organize it. With Java 25, you can write and run programs using just a single file, skipping folders and complex project setups. The language now supports a scripting-like style, letting you try out ideas and get results right away. Features like implicit classes, simplified main methods, and the new IO class for input and output all work together to help you learn and experiment with less effort. You will also see how Java now lets you run source files directly, compile code on the fly, and handle input and output in a much simpler way. These improvements are designed to make Java more approachable and efficient, so you can spend more time creating and less time setting things up.

- Identify key simplifications introduced in Java 25 for getting started with programming.
- Describe the use and benefits of implicit classes in Java 25.
- Demonstrate how to run Java source files directly without manual compilation.
- Explain the lazy compilation and execution process in Java 25.
- Write main methods using the simplified syntax options introduced in Java 25.
- Recognize how Java determines the entry point when multiple main methods exist.
- Use the new IO class to print output and read input in Java 25 programs.

**15) Utility Classes**

Understanding how Java manages basic data types, dates, and formatted strings is crucial for building efficient and reliable applications. Wrapper classes make it possible to treat primitive values as objects, which is essential when working with libraries or APIs that require objects. Converting between primitives and Strings, or handling numbers with high precision, is handled through specialized classes and methods. You will see how autoboxing and unboxing simplify code, and where caution is required for performance or reliability.

Enums provide a safer and clearer way to work with fixed sets of constants, and static imports help streamline your code. Working with dates and times in Java has evolved, with the java.time API offering powerful classes for handling date-only, time-only, and combined date-time values, as well as supporting operations like calculating the difference between two dates or formatting output for different regions. These concepts, along with flexible string formatting, give you the tools to write code that is not only correct and efficient, but also readable and maintainable.

- Introduce the wrapper classes
- Explain Autoboxing and Unboxing
- Converting String representations of primitive numbers into their primitive types
- Defining Enumerations
- Using static imports
- Introduce the Date/Time API
- LocalDate / LocalDateTime etc.
- Apply text formatting

- Using System.out.printf

**16) Inheritance and Polymorphism**

Polymorphism is a central pillar of object-oriented programming, giving you the flexibility to design systems where different objects can be treated uniformly through a common interface. By learning how Java applies polymorphism, you gain the ability to write code that is both reusable and adaptable. You'll see how subclassing allows you to create specialized versions of a parent class, each capable of providing its own implementation of a method. You'll also examine how casting between class types works, including the differences between upcasting and downcasting, and how to use the instanceof operator to maintain type safety. The lesson also explores how the final keyword helps control inheritance, method overriding, and variable immutability, providing essential tools for robust Java programming.

- 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

**17) Interfaces and Abstract Classes**

Understanding how to structure code for flexibility and maintainability is crucial in Java programming. This lesson explores the core principles behind separating what an object can do from how it does it. You will see how Java formalizes these capabilities through interfaces and abstract classes, providing tools to define contracts and organize code. By examining abstract classes, you'll learn how partial blueprints guide subclass development. Interfaces, on the other hand, show how to declare abilities that many unrelated classes can share. The lesson also covers advanced interface features like default, static, and private methods, which add new layers of power and convenience to your designs. Through practical examples, you will gain insight into polymorphism, multi-interface implementation, type checking, and the evolving nature of interfaces, empowering you to create robust, adaptable software.

- 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

**18) Sealed Classes**

Understanding how Java controls the relationships between types is crucial for building reliable and maintainable software. You will explore interface contracts, which set the expectations for what a class or interface should deliver, and see how Java enforces these expectations through inheritance and implementation rules. The lesson delves into the challenges of controlling which classes can implement or extend an interface using traditional methods such as final classes and package-private constructors. Next, you will learn about sealed types, a feature that allows you to explicitly define and limit which classes are permitted to be part of a hierarchy. This approach provides greater predictability and clarity in your code by reducing accidental or unwanted extensions. You will also see how sealed classes and interfaces work in practice, how to declare permitted subclasses, and how

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

these constructs compare to final and abstract classes. To help you make informed decisions in your own projects, you will review the benefits, real-world scenarios, and limitations of using sealed classes in Java.

- Introduce sealed classes
- The sealed and permits modifier
- Sealed interfaces
- Sealed classes and pattern matching

19) Pattern Matching
You will explore advanced features in Java that streamline type checks and control flow within your code. Learn how the `instanceof` operator and pattern matching work together to make type safety more robust and your code more concise. Discover how modern Java switch statements have evolved to handle a broader range of types and conditions, including support for null values, exhaustive matching, and integration with enums, sealed classes, and interfaces. You'll also work with records and see how pattern matching and type inference can simplify handling both simple and nested data structures. By mastering these features, you can write clearer, safer, and more expressive Java programs.

- Pattern Matching in switch statements
- Pattern Matching and sealed classes
- Record Patterns

**20) Introduction to Exception Handling**

Dealing with unexpected problems is an essential part of programming, and Java provides clear structures for managing these situations. This lesson explores how Java identifies, organizes, and handles errors that arise during program execution. You'll see how exceptions differ from regular errors, and how Java's system directs you to address them through specialized objects and structured code blocks. By understanding how exceptions are classified and used, you'll gain the ability to write software that reacts predictably to problems such as missing files or invalid data. You'll also discover how the exception hierarchy allows you to manage both general and specific error cases, and why Java distinguishes between exceptions you must handle and those you may safely ignore. With practical examples, this lesson will strengthen your skills in making robust, reliable, and maintainable Java applications that can cope with both anticipated and unforeseen issues.

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

**21) Exceptions**

In this lesson, you will learn how to create and manage custom exceptions in Java, making your error handling more specific and effective for your applications. You will explore how Java exceptions are structured as classes, giving you the ability to extend and customize them to reflect unique scenarios in your code. The lesson will guide you through providing meaningful constructors and additional fields in custom exception classes, enhancing the clarity and usefulness of error messages. You will also discover strategies for handling resources safely, including the use of try-with-resources and the finally block, ensuring that resources are always properly released. Key techniques such as multi-catch, suppressed exception handling, and improvements in NullPointerException diagnostics will be

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

covered, equipping you to build more robust and maintainable Java applications. The lesson will conclude with practical guidance on when to use exceptions and why leveraging standard exceptions leads to cleaner, more predictable code.

- Defining your own application exceptions
- Automatic closure of resources
- Suppressed exceptions
- Handling multiple exceptions in one catch
- Enhanced try-with-resources
- Helpful NullPointerException(s)

**22) Building Java Applications**

After writing your source code, the next critical phase in application development involves several structured steps to transform your code into a working product. These steps include compiling your code, managing resources, generating documentation, and running tests, all of which can be labor-intensive if handled manually. Tools like Maven automate and simplify these processes, making software development more efficient and organized. Understanding how to use Maven, including how to set up a project, manage dependencies, and import your project into popular IDEs like Eclipse and IntelliJ, is essential for modern Java development. This lesson provides practical guidance on using Maven to streamline your workflow and create robust, maintainable applications.

- Explain the steps involved in building applications
- Define the build process
- Introduce build scripts
- Explain the standard folder layout
- Resolving project dependencies

**23) Introduction to Generics**

Generics are a key feature in Java that make code more robust and easier to maintain. By introducing type parameters, you gain the ability to define classes, interfaces, and methods that work safely with different data types. This means you can write code that is both flexible and type-safe, helping you catch errors at compile time instead of at runtime. Generics are especially useful when working with collections, such as lists or maps, as they prevent accidental mixing of unrelated object types and eliminate the need for explicit casting. Through this lesson, you will see how generics provide safer, cleaner, and more efficient ways to handle data structures and operations in Java, making your code more reliable and easier to understand.

- Explain the purpose of generics in Java.
- Identify the risks of using non-generic collections.
- Describe how to define and use generic classes.
- Implement generic methods in Java.
- Distinguish between raw types and parameterized types.
- Apply type parameter naming conventions in generic code.
- Demonstrate the use of bounded types and bounded wildcards.

**24) Introducing Lambda Expressions and Functional Interfaces**

Functional and object-oriented programming represent two fundamental approaches to structuring and organizing code in Java. By understanding both, you can choose the strategy that best fits your

BİLGİNÇ IT Academy
info@bilginc.com
bilginc.com

coding challenges, whether you need reliability and predictability or rich data modeling with stateful objects. This lesson introduces functional interfaces, a cornerstone of functional programming in Java, and shows how they enable the use of lambda expressions for more concise, expressive code. You'll learn how these concepts lead to cleaner design, improved modularity, and greater flexibility. Through practical examples, you'll see how to implement reusable functions, leverage generic interfaces, and apply method or constructor references for streamlined code. Along the way, you'll discover how lambdas differ from traditional anonymous classes and how to use them effectively for better code clarity and maintainability.

- Identify the core differences between functional and object-oriented programming.
- Explain the purpose and structure of functional interfaces in Java.
- Demonstrate how to implement functional interfaces with lambda expressions.
- Utilize utility methods that accept functional interfaces as parameters.
- Optimize lambda expressions for clarity and conciseness.
- Use generic functional interfaces for flexible code reuse.

**25) Collections**

Arrays in Java offer a simple way to group and manage data, but their fixed size and rigid structure can make code less adaptable and efficient. When dealing with dynamic or large-scale data management, you need tools that can grow, shrink, and respond flexibly to changes. The Java Collections Framework was designed for exactly this purpose. By learning the essential interfaces and their implementations, you gain the ability to organize, manipulate, and access your data in ways that are both powerful and efficient. Understanding how to use these interfaces, as well as the new features introduced in recent Java versions, will help you write cleaner, more maintainable code. This lesson will guide you through the most important concepts, tools, and best practices for working with collections, so you can confidently manage data in modern Java applications.

- 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
- Sequenced Collections

**26) Using Collections**

Java’s Collection Framework offers a versatile set of tools for managing and manipulating groups of objects. You will explore the distinct interfaces that shape how elements are stored, accessed, and modified, from ensuring uniqueness in sets to managing key-value associations in maps. By learning about advanced interfaces such as SortedSet and NavigableSet, you’ll see how ordered data can be efficiently retrieved and maintained. Sorting is a key theme, with a focus on both natural and custom orderings using Comparable and Comparator. Modern Java features, such as lambdas and method references, enable functional programming approaches to collections, making data processing concise and expressive. You’ll also discover how to create immutable and thread-safe collections for robust application design, and why thoughtful selection of collection types and implementations matters for performance. Finally, practical considerations like setting initial capacity will be highlighted to help you write efficient, scalable Java programs.

- Collection Sorting
- Comparators

BİLGİNÇ IT Academy

info@bilginc.com

bilginc.com


- Using the Right Collection
- Lambda expressions in Collections
- Sequenced Sets

**27) Streams**

Collections are foundational to how Java applications handle and process data. When working with lists or other collection types, performing operations such as searching, grouping, and aggregating data becomes a regular task. Unlike SQL, which provides a straightforward, declarative syntax for these operations, Java's traditional approach often involves explicit, repetitive coding to filter and process collections. The Stream API transforms this experience, offering a set of tools to express complex data manipulations in a clear and concise way. By learning how to use streams, you can write more readable, maintainable code that takes full advantage of Java's capabilities, including parallel processing. This lesson equips you with the concepts and practical skills needed to filter, map, and collect data using streams, and shows how to harness advanced features like optional values, numeric streams, and stream sources. You will see how these tools can handle both simple and large-scale data processing tasks efficiently.

- Understanding the problem with collections in Java
- Thinking of program solutions in a declarative way
- Use the Stream API to process collections of data
- Understand the difference between intermediate and terminal stream operations
- Filtering elements from a Stream
- Finding element(s) within a Stream
- Collecting the elements from a Stream into a List

**28) Collectors**

Collectors are essential tools for anyone working with Java Streams. You will learn how collectors help transform, organize, and analyze data as it flows through a stream. This lesson introduces the core collector operations: accumulating elements into collections, grouping and partitioning data based on properties or conditions, and summarizing information to extract key insights. Real-world examples, such as categorizing flight data, will illustrate each concept in action. You will also explore how to combine Stream operations with collectors for tasks like creating lists, sets, and maps, producing formatted strings, and performing summary calculations. By understanding and applying collectors, you gain the ability to manage complex data with concise, efficient code.

- Using different ways to collect the items from a Stream
- Grouping elements within a stream
- Gathering statistics about numeric property of elements in a stream

**Additional Topic: Time Permitting**

These topics will be included in your course materials but may or may not be presented during the live class depending on the pace of the course and attendee skill level and participation.

**1) Introduction to Annotations**

Annotations in Java are a powerful feature that allow you to add meaningful metadata directly to your code. These annotations act as instructions for the compiler, tools, or frameworks, but are not part of your core business logic. By learning how to use annotations, you gain the ability to influence how your

BİLGİNÇ IT Academy

info@bilginc.com

bilginc.com


code is processed, validated, and executed—often making configuration simpler and more intuitive. You'll encounter standard annotations provided by Java, as well as those created by various frameworks and libraries. Creating your own custom annotations lets you define reusable patterns or rules for your projects. Understanding where and how annotations are retained and applied is essential, particularly when integrating with tools that use reflection to inspect or modify behavior at runtime. This lesson explores these concepts and demonstrates how annotations enhance code organization, maintainability, and functionality, especially in enterprise and testing environments.

- Discuss how annotations work in Java
- Understand what is required to work with Java’s annotations
- Use annotations
- Other technologies that are using annotations

**2) Java Data Access JDBC API**

JDBC is a foundational technology for Java applications that interact with databases. By using JDBC, you can connect your Java programs to a wide range of data sources, not just traditional relational database management systems. This lesson guides you through the essential concepts of JDBC, from understanding its standard interfaces and the role of database drivers, to writing SQL commands that retrieve and modify data. You will learn how to establish database connections, use Statement and PreparedStatement objects for executing queries, and process the results with ResultSet. Additionally, you will see how to perform data modifications and control transactions to maintain data integrity. Mapping between SQL and Java data types is also covered, ensuring you can work confidently with different kinds of information. Through practical examples and clear explanations, you will gain the skills needed to integrate robust database functionality into your Java applications

- Connecting to a database using JDBC
- Executing a statement against a database that returns a ResultSet
- Setting up and working with PreparedStatements
- Extracting multiple rows of data from a ResultSet
- Inserting, updating and deleting rows in a table