Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: JBCP • **Updated:** 18 Nov 2026
https://mockattempt.com/training/java-backend-developer-career-program

JBCP • JAVA ENTERPRISE BACKEND & MICROSERVICES ENGINEERING **BEGINNER TO JOB-READY**

100% PLACEMENT GUARANTEED* CAREER PROGRAM (6-MO REFUND*)

Java Backend Developer – Industry-Ready Career Program

Go from Java 21 fundamentals to building, testing, containerizing, deploying, and defending production-style Spring Boot 3 & Microservices backend applications across 24 structured weekly modules, 6 mandatory projects, and 120+ DSA problems.

Duration: 24 Weeks (6 Months) • Beginner to Job-Ready (600 Hrs) • **Modules:** 24 • **Live Topics:** 439 • **Subtopics:** 3,512+ • **Labs & Tasks:** 878 • **Mocks / AI Vivas:** 439 / 439 • **Projects:** 24 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹15,000 (Net Tuition ₹12,712 + Included 18% GST ₹2,288) • **Split Option:** 3 × $53 (0% Installments) • **Effective ROI:** ~₹25/Guided Hr (~₹34/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Career Live Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 28 Weeks (600 Guided Live + Lab Hours))

✓ **Trainer LeetCode & HackerRank Mentorship (Java 21 LTS & PostgreSQL SQL): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

✓ **6-Month Placement Guarantee* + Online AI Assessment (OA) Support: Complete 100% assigned repositories & score ≥90% across all Topic Mock Tests & Voice AI Interviews → 100+ verified IT job openings within 6 months with active Senior Trainer coaching to crack employer Online AI Assessments (OA), or 100% net tuition refund*.**

Core Technologies & Engineering Stack: Java 21 LTS • Core Java & OOP • Java Collections Framework • Generics, Lambdas & Stream API • JVM & Multithreading • Data Structures & Algorithms (120+) • Git & GitHub PR Workflow • PostgreSQL & SQL • MySQL Fundamentals • JDBC & HikariCP • JPA & Hibernate 6 • Spring Framework 6 • Spring Boot 3.x & REST APIs • Spring Data JPA & Validation • Spring Security 6 & JWT • Microservices & Spring Cloud • Netflix Eureka & API Gateway • Resilience4j • Apache Kafka • Redis & MongoDB Fundamentals • JUnit 5, Mockito & Testcontainers • Swagger / OpenAPI & Postman • Docker, Linux & GitHub Actions CI/CD • Maven & IntelliJ IDEA

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Core Java: Write structured, clean, object-oriented Java 21 applications with packages, enums, and custom exceptions.
2. Advanced Java: Work fluently with Collections internals, Generics, Lambdas, Stream API, Optional, JVM memory, and multithreading.
3. Data Structures & Algorithms: Solve 100–150 curated Easy and Medium (plus selected Hard) Java coding problems with clear Big-O analysis.
4. Database & SQL: Design normalized 3NF PostgreSQL schemas, composite indexes, joins, subqueries, ACID transactions, window functions, and CTEs.
5. JDBC & Hibernate/JPA: Connect Java backends via pooled JDBC and implement N+1-free ORM persistence, relationships, and @Version locking.
6. Spring Framework & Spring Boot 3: Understand IoC/DI/beans/AOP and develop validated, documented, paginated production-style REST APIs.
7. Spring Security & JWT: Implement BCrypt password security, stateless JWT authentication, RBAC roles, and resource-ownership guards.
8. Microservices, Kafka & Redis: Build independently deployable services with Eureka, API Gateway, Resilience4j, Kafka event pipelines (Outbox/DLQ), and Redis caching.
9. Testing, Git, Docker & CI/CD: Write JUnit 5, Mockito & Testcontainers tests, collaborate via GitHub PRs, containerize with Docker Compose, and explain your entire backend architecture in technical interviews.
10. Crack LeetCode and HackerRank problems in Java 21 LTS & PostgreSQL SQL with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
11. Receive active Senior Trainer coaching during your 6-month (180-day) Placement Tenure to crack employer Online AI Assessments (OA), timed coding rounds, and automated screening tests.
12. Automated Testing, Debugging & Quality Gates (Java 21 LTS, Core Java & OOP, Java Collections Framework, Generics, Lambdas & Stream API, JVM & Multithreading) — Automated Java 21 LTS, Core Java & OOP, Java Collections Framework, Generics, Lambdas & Stream API, JVM & Multithreading test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.

Eligibility & Prerequisites: No prior Java or backend framework experience required — Month 1 starts from JDK 21 installation and programming fundamentals. • Consistent weekly commitment across live sessions, hands-on coding assignments, GitHub commits, and continuous DSA practice. •

Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Dedication to write original code in your GitHub portfolio and complete the mentor-guided Live Client Production Sprint.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (24 MODULES • 439 LIVE TOPICS • 3,512+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Core Java Fundamentals (Weeks 1–4) (59 Topics / 472 Subtopics) • Modules 5–8: Month 2 – Advanced Core Java, DSA and Git (Weeks 5–8) (88 Topics / 704 Subtopics) • Modules 9–12: Month 3 – Database, SQL, JDBC and Hibernate (Weeks 9–12) (73 Topics / 584 Subtopics) • Modules 13–16: Month 4 – Spring Framework and Spring Boot (Weeks 13–16) (75 Topics / 600 Subtopics) • Modules 17–20: Month 5 – Microservices and Industry Backend Engineering (Weeks 17–20) (73 Topics / 584 Subtopics) • Modules 21–24: Month 6 – Testing, Deployment, Capstone and Placement Preparation (Weeks 21–24) (71 Topics / 568 Subtopics)

MODULE 1 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 13 TOPICS (104 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Java Introduction and Programming Fundamentals

Master the Java 21 LTS platform, JDK/JRE/JVM architecture, bytecode compilation, primitive & reference types, type casting, operators, console I/O, Javadoc conventions, and day-one Git repository setup.

Objectives: ✓ Differentiate JDK, JRE, and JVM responsibilities and configure Java 21 LTS with JAVA_HOME and PATH. • ✓ Compile and execute Java source files via javac/java while tracing bytecode verification and execution. • ✓ Declare local, instance, and static variables using appropriate primitive and reference data types. • ✓ Apply widening/narrowing type conversions and arithmetic, relational, logical, bitwise, and ternary operators accurately. • ✓ Read structured console input with Scanner and BufferedReader, write clean self-documenting code, and commit daily exercises to Git.

1.1 Introduction to Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: History • Features • Platform independence • Enterprise applications • Production Edge-Case & Pitfall Guard (Introduction to Java) • Hands-On Java Implementation & Verification (Introduction to Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Java

Purpose: Selecting Java 21 LTS for high-throughput banking, e-commerce, and enterprise backend microservices.

What Topic Is Next → Next Topic 1.2: Java Architecture (JDK, JRE and JVM)

1.2 Java Architecture (JDK, JRE and JVM) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JDK • JRE • JVM components • Write Once Run Anywhere (WORA) • Production Edge-Case & Pitfall Guard (Java Architecture (JDK, JRE and JVM)) • Hands-On Java Implementation & Verification (Java Architecture (JDK, JRE and JVM)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Java Architecture (JDK, JRE and JVM)

Purpose: Choosing minimal JRE/jlink container base images for production Docker deployments vs full JDK images in CI build stages.

What Topic Is Next → Next Topic 1.3: Java Installation and Environment Setup

1.3 Java Installation and Environment Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JDK 21 LTS installation • JAVA_HOME configuration • OS PATH setup • IntelliJ IDEA & Maven workspace verification • Production Edge-Case & Pitfall Guard (Java Installation and Environment Setup) • Hands-On Java Implementation & Verification (Java Installation and Environment Setup) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Java Installation

Purpose: Standardizing developer and CI runner environments on Java 21 LTS so Maven builds behave identically across Windows, Linux, and macOS.

What Topic Is Next → Next Topic 1.4: First Java Program

1.4 First Java Program [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class declaration • Public static void main(String[] args) signature • System.out printing • CLI arguments • Production Edge-Case & Pitfall Guard (First Java Program) • Hands-On Java Implementation & Verification (First Java Program) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for First Java Program

Purpose: Bootstrapping command-line utilities, batch jobs, and Spring Boot entry-point classes cleanly.

What Topic Is Next → Next Topic 1.5: Java Compilation and Bytecode Execution

1.5 Java Compilation and Bytecode Execution [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Source code (.java) • Javac compiler checks • Bytecode (.class) • Class loader verification • Production Edge-Case & Pitfall Guard (Java Compilation and Bytecode Execution) • Hands-On Java Implementation & Verification (Java Compilation and Bytecode Execution) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Java Compilation

Purpose: Diagnosing NoClassDefFoundError vs ClassNotFoundException and inspecting compiled `.class` artifacts in Maven `target/classes`.

What Topic Is Next → Next Topic 1.6: Variables in Java

1.6 Variables in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Local variables • Instance fields • Static class variables • Variable scope • Production Edge-Case & Pitfall Guard (Variables in Java) • Hands-On Java Implementation & Verification (Variables in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Variables in Java

Purpose: Preventing shared-state bugs in backend services by keeping request data in method-local variables rather than mutable static fields.

What Topic Is Next → Next Topic 1.7: Data Types in Java

1.7 Data Types in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 8 primitive types (byte • Short • Int • Long • Production Edge-Case & Pitfall Guard (Data Types in Java) • Hands-On Java Implementation & Verification (Data Types in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Types in Java

Purpose: Choosing `long` for database primary keys and epoch timestamps, and `BigDecimal` over `double` for exact monetary calculations.

What Topic Is Next → Next Topic 1.8: Type Conversion and Casting

1.8 Type Conversion and Casting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implicit widening conversion • Explicit narrowing cast • Numeric promotion in expressions • Truncation • Production Edge-Case & Pitfall Guard (Type Conversion and Casting) • Hands-On Java Implementation & Verification (Type Conversion and Casting) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Conversion

Purpose: Safely converting external API payload numbers, percentages, and pagination counts without silent data truncation.

What Topic Is Next → Next Topic 1.9: Operators in Java

1.9 Operators in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Arithmetic • Relational • Short-circuit logical • Assignment • Production Edge-Case & Pitfall Guard (Operators in Java) • Hands-On Java Implementation & Verification (Operators in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Operators in Java

Purpose: Writing guard expressions with short-circuit `&&`/`||`, checking parity/flags, and evaluating concise conditional expressions.

What Topic Is Next → Next Topic 1.10: User Input (Scanner and BufferedReader)

1.10 User Input (Scanner and BufferedReader) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Java.util.Scanner token parsing • Newline buffer pitfalls • InputStreamReader + BufferedReader fast line reading • Input validation • Production Edge-Case & Pitfall Guard (User Input (Scanner and BufferedReader)) • Hands-On Java Implementation & Verification (User Input (Scanner and BufferedReader)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User Input (Scanner and BufferedReader)

Purpose: Reading interactive CLI inputs in console tools and parsing large input streams efficiently in competitive/OA coding tests.

What Topic Is Next → Next Topic 1.11: Comments and Javadoc Documentation

1.11 Comments and Javadoc Documentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-line (//) • Multi-line (/* */) • Javadoc (/** */) tags (@param • @return • Production Edge-Case & Pitfall Guard (Comments and Javadoc Documentation) • Hands-On Java Implementation & Verification (Comments and Javadoc Documentation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comments

Purpose: Documenting public service contracts and utility APIs so IDE tooltips and Maven Javadoc builds stay accurate for teammates.

What Topic Is Next → Next Topic 1.12: Java Coding Conventions and Clean Code Basics

1.12 Java Coding Conventions and Clean Code Basics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PascalCase classes • CamelCase methods/variables • UPPER_SNAKE_CASE constants • Package naming • Production Edge-Case & Pitfall Guard (Java Coding Conventions and Clean Code Basics) • Hands-On Java Implementation & Verification (Java Coding Conventions and Clean Code Basics) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Java Coding Conventions

Purpose: Passing automated Checkstyle/SonarQube pull-request reviews in enterprise Java backend teams.

What Topic Is Next → Next Topic 1.13: Week 1 Git Setup and Daily Commit Discipline

1.13 Week 1 Git Setup and Daily Commit Discipline [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Initializing local Git repo • Configuring user.name/user.email • .gitignore for Java/Maven/IntelliJ • Staging • Production Edge-Case & Pitfall Guard (Week 1 Git Setup and Daily Commit Discipline) • Hands-On Java Implementation & Verification (Week 1 Git Setup and Daily Commit Discipline) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Week 1 Git Setup

Purpose: Maintaining a verifiable public GitHub commit trail from Week 1 onward rather than uploading zip files at the end of the course.

What Topic Is Next → Module 1 Capstone (Week 1 Lab Build: CLI Financial & Academic Utility Suite) → Next Topic 2.1: Conditional Statements

Module 1 Hands-On Capstone Project (100 Marks): Week 1 Lab Build: CLI Financial & Academic Utility Suite — Build a cleanly packaged Java 21 command-line utility suite combining Student Profile Printer, Simple/Compound Interest Calculator, Employee Net Salary Slip, Temperature Converter, and Shape...

MODULE 2 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 12 TOPICS (96 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Control Statements and Arrays

Build strong algorithmic control-flow logic using conditionals, modern Java 21 switch expressions, iterative loops, pattern synthesis, 1D & multidimensional arrays, matrix algebra, and java.util.Arrays utilities.

Objectives: ✓ Write bug-free branching logic using guard clauses, if-else ladders, and modern Java 21 exhaustive switch expressions. • ✓ Control iteration cleanly with for, while, do-while, enhanced for-each, break, and continue. • ✓ Solve nested-loop dry-run problems across star, numeric, and character patterns to sharpen index reasoning. • ✓ Allocate, traverse, insert, delete, reverse, deduplicate, and search 1D and 2D arrays in-place with boundary safety. • ✓ Implement matrix addition, subtraction, transpose, and multiplication and leverage java.util.Arrays helper methods.

2.1 Conditional Statements [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: If-else • Nested if • Else-if ladder • Guard clauses • Production Edge-Case & Pitfall Guard (Conditional Statements) • Hands-On Java Implementation & Verification (Conditional Statements) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Conditional Statements

Purpose: Validating API request business rules such as credit eligibility, tax slabs, and order discount tiers cleanly.

What Topic Is Next → Next Topic 2.2: Switch Statements and Switch Expressions

2.2 Switch Statements and Switch Expressions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Traditional switch-case fallthrough vs modern Java 14–21 arrow switch expressions (`->`) • `yield` • Exhaustiveness • Production Edge-Case & Pitfall Guard (Switch Statements and Switch Expressions) • Hands-On Java Implementation & Verification (Switch Statements and Switch Expressions) • Switch Statements and Switch Expressions — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Switch Statements

Purpose: Mapping payment statuses, HTTP codes, and order states cleanly without accidental case fallthrough bugs.

What Topic Is Next → Next Topic 2.3: Loops (for, while, do-while)

2.3 Loops (for, while, do-while) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deterministic `for` loops • Condition-driven `while` loops • Menu-driven `do-while` loops • Infinite-loop prevention • Production Edge-Case & Pitfall Guard (Loops (for, while, do-while)) • Hands-On Java Implementation & Verification (Loops (for, while, do-while)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loops (for, while, do-while)

Purpose: Paginating batch database records, retrying transient operations with bounded counters, and running CLI menus.

What Topic Is Next → Next Topic 2.4: Loop Control (break, continue and Nested Loops)

2.4 Loop Control (break, continue and Nested Loops) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Early exit with `break` • Skipping iterations with `continue` • Labeled breaks • Nested loop complexity awareness • Production Edge-Case & Pitfall Guard (Loop Control (break, continue and Nested Loops)) • Hands-On Java Implementation & Verification (Loop Control (break, continue and Nested Loops)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loop Control (break, continue and Nested Loops)

Purpose: Short-circuiting search loops immediately once a matching record is found and skipping malformed CSV rows.

What Topic Is Next → Next Topic 2.5: Patterns (Star, Number and Character Patterns)

2.5 Patterns (Star, Number and Character Patterns) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Row-column coordinate mapping • Pyramid/diamond star patterns • Floyd's number triangle • Character matrices • Production Edge-Case & Pitfall Guard (Patterns (Star, Number and Character Patterns)) • Hands-On Java Implementation & Verification (Patterns (Star, Number and Character Patterns)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Patterns (Star, Number and Character Patterns)

Purpose: Developing rock-solid mental dry-running of nested loop invariants, boundary conditions, and 2D grid coordinates.

What Topic Is Next → Next Topic 2.6: Arrays Introduction, Declaration and Initialization

2.6 Arrays Introduction, Declaration and Initialization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contiguous heap allocation • Zero/default initialization • `.length` property • Static vs dynamic array literals • Production Edge-Case & Pitfall Guard (Arrays Introduction, Declaration and Initialization) • Hands-On Java Implementation & Verification (Arrays Introduction, Declaration and Initialization) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Arrays Introduction

Purpose: Storing fixed-size lookup tables, byte buffers, and low-overhead numeric sequences in performance-critical Java code.

What Topic Is Next → Next Topic 2.7: Array Operations (Traversal, Insertion, Deletion and Searching)

2.7 Array Operations (Traversal, Insertion, Deletion and Searching) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Forward/backward traversal • Shifting elements for insertion • Deletion • Linear search • Production Edge-Case & Pitfall Guard (Array Operations (Traversal, Insertion, Deletion and Searching)) • Hands-On Java Implementation & Verification (Array Operations (Traversal, Insertion, Deletion and Searching)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Operations (Traversal, Insertion, Deletion and...

Purpose: Understanding why `ArrayList` insertion/deletion in the middle costs `O(n)` element shifts under the hood.

What Topic Is Next → Next Topic 2.8: One-Dimensional Arrays Manipulation

2.8 One-Dimensional Arrays Manipulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Finding second-largest element • Counting frequencies • Finding duplicates • Removing duplicates from sorted arrays • Production Edge-Case & Pitfall Guard (One-Dimensional Arrays Manipulation) • Hands-On Java Implementation & Verification (One-Dimensional Arrays Manipulation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for One-Dimensional Arrays Manipulation

Purpose: Solving frequent OA screening problems and implementing in-memory analytics over numeric metrics.

What Topic Is Next → Next Topic 2.9: Multidimensional Arrays

2.9 Multidimensional Arrays [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 2D rectangular arrays • Jagged arrays (arrays of arrays) • Row-major memory layout • Nested traversal • Production Edge-Case & Pitfall Guard (Multidimensional Arrays) • Hands-On Java Implementation & Verification (Multidimensional Arrays) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multidimensional Arrays

Purpose: Representing seating charts, tabular exports, dynamic programming tables, and adjacency matrices.

What Topic Is Next → Next Topic 2.10: Matrix Operations (Addition, Subtraction, Transpose and Multiplication)

2.10 Matrix Operations (Addition, Subtraction, Transpose and Multiplication) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dimension compatibility validation • Matrix addition/subtraction • In-place/out-of-place transpose • `O(m*n*p)` multiplication • Production Edge-Case & Pitfall Guard (Matrix Operations (Addition, Subtraction, Transpose and Multiplication)) • Hands-On Java Implementation & Verification (Matrix Operations (Addition, Subtraction, Transpose and Multiplication)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Matrix Operations (Addition, Subtraction, Transpose...

Purpose: Implementing tabular transformations, image/grid rotations, and batch scoring computations accurately.

What Topic Is Next → Next Topic 2.11: Enhanced For Loop (For-Each)

2.11 Enhanced For Loop (For-Each) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-only iteration syntax • Readability advantages • When index-based `for` is required instead • Nested for-each over 2D arrays • Production Edge-Case & Pitfall Guard (Enhanced For Loop (For-Each)) • Hands-On Java Implementation & Verification (Enhanced For Loop (For-Each)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enhanced For Loop (For-Each)

Purpose: Writing concise, bug-free read-only aggregation loops over arrays and collections in service layers.

What Topic Is Next → Next Topic 2.12: Array Utilities (java.util.Arrays)

2.12 Array Utilities (java.util.Arrays) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Arrays.sort • Arrays.binarySearch • Arrays.copyOf • Arrays.copyOfRange • Production Edge-Case & Pitfall Guard (Array Utilities (java.util.Arrays)) • Hands-On Java Implementation & Verification (Array Utilities (java.util.Arrays)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Utilities (java.util.Arrays)

Purpose: Defensive array copying, fast Dual-Pivot Quicksort/TimSort sorting, and clean debug logging of 1D/2D arrays.

What Topic Is Next → Module 2 Capstone (Week 2 Lab Build: Matrix & Array Analytics Engine) → Next Topic 3.1: Class and Object

Module 2 Hands-On Capstone Project (100 Marks): Week 2 Lab Build: Matrix & Array Analytics Engine — Build an interactive Java 21 array and matrix workbench supporting min/max, in-place reversal, duplicate removal, second-largest finder, frequency counter, missing-number solver, and validat...

MODULE 3 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Object-Oriented Programming (OOP)

Design clean, extensible domain models in Java 21 using classes, objects, constructors, encapsulation, inheritance, compile-time & runtime polymorphism, abstract classes, modern interfaces, final invariants, and Object contracts.

Objectives: ✓ Model real business entities using cohesive classes, constructors, `this` chaining, and `static` members. • ✓ Enforce encapsulation and invariant validation so invalid object state cannot be constructed. • ✓ Design clean inheritance hierarchies and prefer composition/interfaces where appropriate. • ✓ Distinguish compile-time overloading from runtime overriding and dynamic method dispatch. • ✓ Implement abstract classes, multi-interface contracts, `final` immutability, and proper `toString()`, `equals()`, and `hashCode()` overrides.

3.1 Class and Object [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class blueprint definition • Heap object instantiation with `new` • Stack object references • Null references • Production Edge-Case & Pitfall Guard (Class and Object) • Hands-On Java Implementation & Verification (Class and Object) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Class

Purpose: Modeling domain entities (`Student`, `BankAccount`, `Order`) with cohesive state and domain methods.

What Topic Is Next → Next Topic 3.2: Methods in Java

3.2 Methods in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Method signatures • Parameter passing (pass-by-value of primitives • Object references) • Return types • Production Edge-Case & Pitfall Guard (Methods in Java) • Hands-On Java Implementation & Verification (Methods in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Methods in Java

Purpose: Decomposing complex business workflows into small, testable, intention-revealing domain methods.

What Topic Is Next → Next Topic 3.3: Constructors and Constructor Overloading

3.3 Constructors and Constructor Overloading [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default compiler constructor • Parameterized constructors • Constructor overloading • Invariant validation during construction • Production Edge-Case & Pitfall Guard (Constructors and Constructor Overloading) • Hands-On Java Implementation & Verification (Constructors and Constructor Overloading) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Constructors

Purpose: Guaranteeing domain objects and Spring beans are always created in a valid, ready-to-use state.

What Topic Is Next → Next Topic 3.4: this Keyword

3.4 this Keyword [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Disambiguating shadowed fields • Constructor chaining via `this(...)` • Passing/returning current instance in fluent APIs • Production Edge-Case & Pitfall Guard (this Keyword) • Hands-On Java Implementation & Verification (this Keyword) • this Keyword — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for this Keyword

Purpose: Eliminating duplicate initialization logic across overloaded constructors and building fluent query/request builders.

What Topic Is Next → Next Topic 3.5: static Keyword

3.5 static Keyword [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Static fields • Static utility methods • Static initialization blocks • Class-level memory in Metaspace/Heap • Production Edge-Case & Pitfall Guard (static Keyword) • Hands-On Java Implementation & Verification (static Keyword) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for static Keyword

Purpose: Defining immutable constants (`public static final`) and stateless factory/utility methods (`LocalDate.of`, `Collections.emptyList`).

What Topic Is Next → Next Topic 3.6: Encapsulation

3.6 Encapsulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Private state • Controlled getters/setters • Invariant enforcement • Defensive copying • Production Edge-Case & Pitfall Guard (Encapsulation) • Hands-On Java Implementation & Verification (Encapsulation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Encapsulation

Purpose: Protecting sensitive account balances, credit limits, and order state transitions from arbitrary external mutation.

What Topic Is Next → Next Topic 3.7: Inheritance

3.7 Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: IS-A relationship • `extends` • Single • Multilevel • Production Edge-Case & Pitfall Guard (Inheritance) • Hands-On Java Implementation & Verification (Inheritance) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inheritance

Purpose: Sharing base auditing fields (`createdAt`, `updatedAt`) or common vehicle/employee traits across specialized subtypes.

What Topic Is Next → Next Topic 3.8: super Keyword

3.8 super Keyword [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Invoking parent constructors via `super(...)` • Calling overridden parent methods via `super.method()` • Constructor execution order • Production Edge-Case & Pitfall Guard (super Keyword) • Hands-On Java Implementation & Verification (super Keyword) • super Keyword — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for super Keyword

Purpose: Initializing immutable parent fields in custom exception classes and specialized domain subclasses.

What Topic Is Next → Next Topic 3.9: Method Overloading (Compile-Time Polymorphism)

3.9 Method Overloading (Compile-Time Polymorphism) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Same method name with distinct parameter lists (arity • Types • Order) • Type promotion rules • Production Edge-Case & Pitfall Guard (Method Overloading (Compile-Time Polymorphism)) • Hands-On Java Implementation & Verification (Method Overloading (Compile-Time Polymorphism)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Method Overloading (Compile-Time Polymorphism)

Purpose: Providing ergonomic API entry points (`search(long id)`, `search(String email)`, `search(String dept, int page)`).

What Topic Is Next → Next Topic 3.10: Method Overriding (Runtime Polymorphism)

3.10 Method Overriding (Runtime Polymorphism) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: @Override annotation • Covariant return types • Access-level & checked-exception rules • Dynamic dispatch on the runtime object • Production Edge-Case & Pitfall Guard (Method Overriding (Runtime Polymorphism)) • Hands-On Java Implementation & Verification (Method Overriding (Runtime Polymorphism)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Method Overriding (Runtime Polymorphism)

Purpose: Customizing `calculatePay()` or `calculateArea()` per subtype while callers depend only on the base contract.

What Topic Is Next → Next Topic 3.11: Polymorphism (Static vs Dynamic Binding)

3.11 Polymorphism (Static vs Dynamic Binding) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Early binding (private • Static • Final methods • Fields) vs late virtual method invocation • Production Edge-Case & Pitfall Guard (Polymorphism (Static vs Dynamic Binding)) • Hands-On Java Implementation & Verification (Polymorphism (Static vs Dynamic Binding)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Polymorphism (Static vs Dynamic Binding)

Purpose: Processing heterogeneous collections of `Notification` or `Payment` strategies uniformly without `if/else` type chains.

What Topic Is Next → Next Topic 3.12: Abstraction and Abstract Classes

3.12 Abstraction and Abstract Classes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Abstract classes • Abstract vs concrete template methods • Protected constructors • Partial implementation sharing • Production Edge-Case & Pitfall Guard (Abstraction and Abstract Classes) • Hands-On Java Implementation & Verification (Abstraction and Abstract Classes) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Abstraction

Purpose: Implementing the Template Method pattern (e.g., `AbstractReportExporter` handling headers/footers while subclasses render rows).

What Topic Is Next → Next Topic 3.13: Interfaces and Multiple Interface Implementation

3.13 Interfaces and Multiple Interface Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pure capability contracts • `implements` • Multiple interface inheritance • Loose coupling • Production Edge-Case & Pitfall Guard (Interfaces and Multiple Interface Implementation) • Hands-On Java Implementation & Verification (Interfaces and Multiple Interface Implementation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interfaces

Purpose: Decoupling Spring Boot controllers from service implementations (`PaymentGateway` implemented by `UpiPaymentGateway` & `CardPaymentGateway`).

What Topic Is Next → Next Topic 3.14: Modern Interface Features (Default, Static and Private Methods)

3.14 Modern Interface Features (Default, Static and Private Methods) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Java 8+ `default` methods for backward-compatible evolution • `static` factory methods on interfaces • `private` helper methods • Diamond conflict resolution • Production Edge-Case & Pitfall Guard (Modern Interface Features (Default, Static and Private Methods)) • Hands-On Java Implementation & Verification (Modern Interface Features (Default, Static and Private Methods)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern Interface Features (Default, Static and Priva...

Purpose: Adding new convenience methods (like `List.sort` or `Comparator.thenComparing`) to interfaces without breaking existing implementors.

What Topic Is Next → Next Topic 3.15: final Keyword (Variables, Methods and Classes)

3.15 final Keyword (Variables, Methods and Classes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Blank final fields • Immutable value objects • Preventing method overriding (`final` methods) • Non-extensible `final` classes • Production Edge-Case & Pitfall Guard (final Keyword (Variables, Methods and Classes)) • Hands-On Java Implementation & Verification (final Keyword (Variables, Methods and Classes)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for final Keyword (Variables, Methods and Classes)

Purpose: Designing thread-safe immutable DTOs, security tokens, and Spring constructor-injected `private final` dependencies.

What Topic Is Next → Next Topic 3.16: Object Class (toString, equals, hashCode and getClass)

3.16 Object Class (toString, equals, hashCode and getClass) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Java.lang.Object root methods • Readable `toString()` • Reflexive/symmetric/transitive `equals()` • `Objects.hash()` contract • Production Edge-Case & Pitfall Guard (Object Class (toString, equals, hashCode and getClass)) • Hands-On Java Implementation & Verification (Object Class (toString, equals, hashCode and getClass)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object Class (toString, equals, hashCode and getClas...

Purpose: Ensuring domain entities and composite keys behave accurately in logs, assertions, `HashSet`, and `HashMap`.

What Topic Is Next → Module 3 Capstone (Week 3 Lab Build: Multi-Domain OOP Banking & Library Core) → Next Topic 4.1: String Immutability and String Pool

Module 3 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Multi-Domain OOP Banking & Library Core — Design an extensible OOP library and banking core demonstrating inheritance, interfaces (`PaymentGateway`), abstract template classes, runtime polymorphism, `final` value objects, and contra...

MODULE 4 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Strings, Exception Handling and Core Utilities

Master Java String immutability & String Pool internals, StringBuilder/StringBuffer, wrappers & autoboxing, structured checked/unchecked/custom exception handling, assertions, packages, access modifiers, and rich enums — culminating in Mini-Project 1.

Objectives: ✓ Explain String immutability and String Constant Pool behavior and choose StringBuilder for loop concatenations. • ✓ Compare strings safely and parse/transform text using core String API methods. • ✓ Prevent NullPointerException and unboxing pitfalls when working with Wrapper classes. • ✓ Design clean exception hierarchies using try-catch-finally, try-with-resources, throw/throws, and custom domain exceptions. • ✓ Organize multi-package Java applications with strict access modifiers and type-safe Enums for Mini-Project 1.

4.1 String Immutability and String Pool [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: String literal interning in Heap String Pool vs `new String()` • Immutability benefits (security • Caching • Thread safety) • Production Edge-Case & Pitfall Guard (String Immutability and String Pool) • Hands-On Java Implementation & Verification (String Immutability and String Pool) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Immutability

Purpose: Understanding memory footprint and hash caching when millions of JSON keys and identifiers pass through a Java backend.

What Topic Is Next → Next Topic 4.2: String Comparison (==, equals, equalsIgnoreCase, compareTo)

4.2 String Comparison (==, equals, equalsIgnoreCase, compareTo) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reference identity (`==`) vs content equality (`equals`) • Null-safe constant-first comparisons • `equalsIgnoreCase` • Lexicographical `compareTo` • Production Edge-Case & Pitfall Guard (String Comparison (==, equals, equalsIgnoreCase, compareTo)) • Hands-On Java Implementation & Verification (String Comparison (==, equals, equalsIgnoreCase, compareTo)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Comparison (==, equals, equalsIgnoreCase, com...

Purpose: Validating role codes, email addresses (case-insensitive), and sorting names deterministically.

What Topic Is Next → Next Topic 4.3: String Methods

4.3 String Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Length • IsBlank • Strip • CharAt • Production Edge-Case & Pitfall Guard (String Methods) • Hands-On Java Implementation & Verification (String Methods) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Methods

Purpose: Sanitizing form/API inputs, parsing CSV rows, masking PII phone/card numbers, and formatting notifications.

What Topic Is Next → Next Topic 4.4: StringBuilder

4.4 StringBuilder [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mutable character buffer • Capacity growth (`old * 2 + 2`) • Append • Insert • Production Edge-Case & Pitfall Guard (StringBuilder) • Hands-On Java Implementation & Verification (StringBuilder) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java

Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for StringBuilder

Purpose: Constructing dynamic SQL clauses, CSV exports, and formatted console tables in `O(n)` time.

What Topic Is Next → Next Topic 4.5: StringBuffer

4.5 StringBuffer [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Synchronized thread-safe mutable strings • Comparison with StringBuilder • String • Legacy API interoperability • Production Edge-Case & Pitfall Guard (StringBuffer) • Hands-On Java Implementation & Verification (StringBuffer) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for StringBuffer

Purpose: Knowing why `StringBuilder` is preferred in method-local code while recognizing `StringBuffer` in JDK regex/legacy APIs.

What Topic Is Next → Next Topic 4.6: Wrapper Classes

4.6 Wrapper Classes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Byte • Short • Integer • Long • Production Edge-Case & Pitfall Guard (Wrapper Classes) • Hands-On Java Implementation & Verification (Wrapper Classes) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Wrapper Classes

Purpose: Representing nullable database columns (`Integer managerId`) in JPA entities and generic collections (`List<Long>`).

What Topic Is Next → Next Topic 4.7: Autoboxing and Unboxing

4.7 Autoboxing and Unboxing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automatic compiler conversion between primitives • Wrappers • Hidden `NullPointerException` on unboxing • Loop allocation overhead • Production Edge-Case & Pitfall Guard (Autoboxing and Unboxing) • Hands-On Java Implementation & Verification (Autoboxing and Unboxing) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Autoboxing

Purpose: Avoiding silent `NullPointerException` when unboxing nullable DTO/entity fields (`Boolean isActive`, `Long count`) in services.

What Topic Is Next → Next Topic 4.8: Exception Handling Hierarchy

4.8 Exception Handling Hierarchy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Throwable -> Error (OutOfMemoryError • StackOverflowError) vs Exception (Checked vs RuntimeException) • Call-stack unwinding • Production Edge-Case & Pitfall Guard (Exception Handling Hierarchy) • Hands-On Java Implementation & Verification (Exception Handling Hierarchy) • Exception Handling Hierarchy — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Exception Handling Hierarchy

Purpose: Designing predictable failure boundaries so bad user requests return clean errors without crashing the application process.

What Topic Is Next → Next Topic 4.9: try-catch Blocks

4.9 try-catch Blocks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Specific-to-general catch ordering • Multi-catch (`catch (IOException | NumberFormatException e)`) • Preserving root cause • Meaningful error messages • Production Edge-Case & Pitfall Guard (try-catch Blocks) • Hands-On Java Implementation & Verification (try-catch Blocks) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for try-catch Blocks

Purpose: Translating low-level parsing or I/O failures into actionable user feedback or structured logs.

What Topic Is Next → Next Topic 4.10: finally Block and Resource Cleanup

4.10 finally Block and Resource Cleanup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Guaranteed execution of `finally` • `try-with-resources` (`AutoCloseable`) • Suppressed exceptions • Why `return` inside `finally` is dangerous • Production Edge-Case & Pitfall Guard (finally Block and Resource Cleanup) • Hands-On Java Implementation & Verification (finally Block and Resource Cleanup) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for finally Block

Purpose: Preventing file-descriptor, socket, and database-connection leaks even when exceptions occur mid-method.

What Topic Is Next → Next Topic 4.11: throw and throws

4.11 throw and throws [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicitly throwing exceptions with `throw` • Declaring checked propagation via `throws` • Exception chaining (`new CustomEx(msg • Cause)`) • Production Edge-Case & Pitfall Guard (throw and throws) • Hands-On Java Implementation & Verification (throw and throws) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for throw

Purpose: Failing fast at service boundaries when preconditions fail while preserving the original stack trace cause.

What Topic Is Next → Next Topic 4.12: Custom Exceptions

4.12 Custom Exceptions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing domain-specific exceptions (`StudentNotFoundException` • `DuplicateRollNumberException`) • Error codes • Constructors • Production Edge-Case & Pitfall Guard (Custom Exceptions) • Hands-On Java Implementation & Verification (Custom Exceptions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Exceptions

Purpose: Mapping domain errors cleanly to HTTP 404 / 409 / 422 status codes in Spring Boot `@RestControllerAdvice` later in Month 4.

What Topic Is Next → Next Topic 4.13: Checked Exceptions

4.13 Checked Exceptions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Compile-time enforced recovery (`IOException` • `SQLException` • `ParseException`) • Handle-or-declare rule • Production Edge-Case & Pitfall Guard (Checked Exceptions) • Hands-On Java Implementation & Verification (Checked Exceptions) • Idiomatic JBCP Implementation: clean modular architecture, strict

contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Checked Exceptions

Purpose: Handling recoverable external resource failures (missing config file, network timeout, SQL connectivity).

What Topic Is Next → Next Topic 4.14: Unchecked Exceptions

4.14 Unchecked Exceptions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: RuntimeException subclasses (`NullPointerException` • `IllegalArgumentException` • `IllegalStateException` • `IndexOutOfBoundsException`) • Production Edge-Case & Pitfall Guard (Unchecked Exceptions) • Hands-On Java Implementation & Verification (Unchecked Exceptions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Unchecked Exceptions

Purpose: Using unchecked domain exceptions in modern Spring/Hibernate transaction boundaries (which roll back on `RuntimeException` by default!).

What Topic Is Next → Next Topic 4.15: Java Assertions

4.15 Java Assertions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `assert condition : message • ` syntax • Enabling assertions via `-ea` JVM flag • Why assertions never replace public input validation • Production Edge-Case & Pitfall Guard (Java Assertions) • Hands-On Java Implementation & Verification (Java Assertions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Java Assertions

Purpose: Verifying internal algorithm invariants during local development and test runs.

What Topic Is Next → Next Topic 4.16: Packages in Java

4.16 Packages in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reverse-domain naming (`com.mockattempt.student.*`) • Directory mapping • `import` vs wildcard imports • Layered package organization • Production Edge-Case & Pitfall Guard (Packages in Java) • Hands-On Java Implementation & Verification (Packages in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Packages in Java

Purpose: Structuring enterprise backends cleanly into `model`, `repository`, `service`, `exception`, and `ui`/`controller` packages.

What Topic Is Next → Next Topic 4.17: Access Modifiers (public, protected, default, private)

4.17 Access Modifiers (public, protected, default, private) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class-level vs member-level visibility matrix across same class • Same package • Subclass in another package • World • Production Edge-Case & Pitfall Guard (Access Modifiers (public, protected, default, private)) • Hands-On Java Implementation & Verification (Access Modifiers (public, protected, default, private)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Access Modifiers (public, protected, default, privat...

Purpose: Keeping repository helpers package-private (`default`) while exposing only narrow `public` service APIs.

What Topic Is Next → Next Topic 4.18: Enums in Java

4.18 Enums in Java [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Type-safe constants • Enum constructors • Fields • Methods • Production Edge-Case & Pitfall Guard (Enums in Java) • Hands-On Java Implementation & Verification (Enums in Java) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enums in Java

Purpose: Modeling fixed domain sets (`Grade`, `OrderStatus`, `Role`, `Department`) with embedded behavior instead of fragile magic strings.

What Topic Is Next → Module 4 Capstone (Mini-Project 1: Student Management Console Application) → Next Topic 5.1: Collections Introduction and Architecture

Module 4 Hands-On Capstone Project (100 Marks): Mini-Project 1: Student Management Console Application — Build a modular, menu-driven Core Java 21 Student Management Console Application applying strict OOP encapsulation, multi-package architecture, enum grade calculation, input validation, and...

MODULE 5 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Java Collections Framework

Deep-dive into every core interface and implementation of `java.util` Collections — List, ArrayList, LinkedList, Vector, Stack, Queue, PriorityQueue, Deque, Set, HashSet, LinkedHashSet, TreeSet, Map, HashMap, LinkedHashMap, TreeMap, Hashtable, Iterators, Comparable, Comparator, and the equals/hashCode contract.

Objectives: ✓ Select the optimal Collection or Map data structure based on lookup, insertion, ordering, and memory complexity. • ✓ Explain the internal bucket/node/red-black-tree mechanics of ArrayList, LinkedList, HashMap, and TreeMap in interviews. • ✓ Traverse and mutate collections safely without triggering ConcurrentModificationException. • ✓ Implement natural ordering (`Comparable`) and multi-field custom sorting (`Comparator`). • ✓ Preserve the `equals()` and `hashCode()` contract so custom keys never get lost in HashSet/HashMap.

5.1 Collections Introduction and Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Iterable -> Collection hierarchy (List • Set • Queue/Deque) • Separate Map hierarchy • Production Edge-Case & Pitfall Guard (Collections Introduction and Architecture) • Hands-On Java Implementation & Verification (Collections Introduction and Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Collections Introduction

Purpose: Choosing standard JDK data structures for DTO lists, deduplicated IDs, caches, and task queues.

What Topic Is Next → Next Topic 5.2: List Interface

5.2 List Interface [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ordered indexed sequence contract • Positional access • Duplicate allowance • `List.of` • Production Edge-Case & Pitfall Guard (List Interface) • Hands-On Java Implementation & Verification (List Interface) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns •

Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for List Interface

Purpose: Returning ordered query results and paginated DTO slices from backend service methods.

What Topic Is Next → Next Topic 5.3: ArrayList

5.3 ArrayList [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic `Object[]` backing array • Default capacity (10) • 1.5x growth resizing • `O(1)` random access • Production Edge-Case & Pitfall Guard (ArrayList) • Hands-On Java Implementation & Verification (ArrayList) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ArrayList

Purpose: Default choice for 90%+ of sequential read-heavy entity and DTO lists in Spring Boot APIs.

What Topic Is Next → Next Topic 5.4: LinkedList

5.4 LinkedList [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Doubly-linked `Node<E>` (`prev` • `item` • `next`) structure • `O(1)` head/tail operations • Production Edge-Case & Pitfall Guard (LinkedList) • Hands-On Java Implementation & Verification (LinkedList) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LinkedList

Purpose: Understanding doubly-linked node pointers and when `ArrayDeque` outperforms `LinkedList` due to CPU cache locality.

What Topic Is Next → Next Topic 5.5: Vector

5.5 Vector [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Legacy synchronized dynamic array • 2x capacity growth • Method-level lock overhead • Modern alternatives (`ArrayList` • Production Edge-Case & Pitfall Guard (Vector) • Hands-On Java Implementation & Verification (Vector) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Vector

Purpose: Maintaining legacy enterprise codebases and answering interview comparisons between `Vector` and `ArrayList`.

What Topic Is Next → Next Topic 5.6: Stack

5.6 Stack [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: LIFO (Last-In-First-Out) push • Pop • Peek operations • Why `java.util.Stack` extends `Vector` • Production Edge-Case & Pitfall Guard (Stack) • Hands-On Java Implementation & Verification (Stack) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stack

Purpose: Implementing undo history, expression parsing, DFS traversal, and bracket validation.

What Topic Is Next → Next Topic 5.7: Queue Interface

5.7 Queue Interface [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: FIFO (First-In-First-Out) processing contract: `offer` vs `add` • `poll` vs `remove` • `peek` vs `element` • Production Edge-Case & Pitfall Guard (Queue Interface) • Hands-On Java Implementation & Verification (Queue Interface) • Queue Interface — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Queue Interface

Purpose: Buffering incoming jobs, breadth-first search (BFS), and ordered request processing.

What Topic Is Next → Next Topic 5.8: PriorityQueue

5.8 PriorityQueue [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Binary heap internal array • Natural min-heap ordering vs custom `Comparator` max-heap • `O(log n)` offer/poll • `O(1)` peek • Production Edge-Case & Pitfall Guard (PriorityQueue) • Hands-On Java Implementation & Verification (PriorityQueue) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PriorityQueue

Purpose: Scheduling urgent support tickets, Top-K analytics, and Dijkstra shortest-path algorithms.

What Topic Is Next → Next Topic 5.9: Deque and ArrayDeque

5.9 Deque and ArrayDeque [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Double-ended queue (`offerFirst` • `offerLast` • `pollFirst` • `pollLast`) • Production Edge-Case & Pitfall Guard (Deque and ArrayDeque) • Hands-On Java Implementation & Verification (Deque and ArrayDeque) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deque

Purpose: High-performance sliding-window maximums, work-stealing queues, and both LIFO stack & FIFO queue workloads.

What Topic Is Next → Next Topic 5.10: Set Interface

5.10 Set Interface [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mathematical uniqueness contract • `Set.of` immutable sets • Set union (`addAll`) • Intersection (`retainAll`) • Production Edge-Case & Pitfall Guard (Set Interface) • Hands-On Java Implementation & Verification (Set Interface) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Set Interface

Purpose: Representing unique user roles/permissions (`Set<Role>`) and computing permission diffs cleanly.

What Topic Is Next → Next Topic 5.11: HashSet

5.11 HashSet [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Backed internally by `HashMap<E • Object>` (`PRESENT` dummy value) • `O(1)` average add/contains/remove • Unordered iteration • Production Edge-Case & Pitfall Guard (HashSet) • Hands-On Java Implementation & Verification (HashSet) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HashSet

Purpose: Deduplicating email batches, checking blacklisted JWT IDs, and fast membership checks.

What Topic Is Next → Next Topic 5.12: LinkedHashSet

5.12 LinkedHashSet [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hash table + doubly-linked list across entries • Preserving predictable insertion order with `O(1)` lookup performance • Production Edge-Case & Pitfall Guard (LinkedHashSet) • Hands-On Java Implementation & Verification (LinkedHashSet) • LinkedHashSet — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LinkedHashSet • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Deduplicating UI filter tags or SQL column lists while preserving exact user-specified order.

What Topic Is Next → Next Topic 5.13: TreeSet

5.13 TreeSet [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Backed by `TreeMap` (Red-Black self-balancing BST) • `O(log n)` operations • `NavigableSet` range queries (`headSet` • `tailSet` • Production Edge-Case & Pitfall Guard (TreeSet) • Hands-On Java Implementation & Verification (TreeSet) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TreeSet

Purpose: Maintaining sorted leaderboards, price brackets, and nearest-slot time lookups (`ceiling`/`floor`).

What Topic Is Next → Next Topic 5.14: Map Interface

5.14 Map Interface [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Key-value associations • `put` • `getOrDefault` • `putIfAbsent` • Production Edge-Case & Pitfall Guard (Map Interface) • Hands-On Java Implementation & Verification (Map Interface) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Map Interface

Purpose: Indexing entities by ID, counting event frequencies with `merge`, and building grouped lookup caches.

What Topic Is Next → Next Topic 5.15: HashMap Internals

5.15 HashMap Internals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Node<K • V>[] table • Hash spreading `(h = key.hashCode()) ^ (h >>> 16)` • Index `(n - 1) & hash` • Production Edge-Case & Pitfall Guard (HashMap Internals) • Hands-On Java Implementation & Verification (HashMap Internals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HashMap Internals

Purpose: Fast `O(1)` in-memory lookups and explaining hash collision degradation & load-factor tuning in interviews.

What Topic Is Next → Next Topic 5.16: LinkedHashMap

5.16 LinkedHashMap [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Insertion-order vs access-order (`accessOrder = true`) doubly-linked entries • Overriding `removeEldestEntry` for bounded LRU caches • Production Edge-Case & Pitfall Guard (LinkedHashMap) • Hands-On Java Implementation & Verification (LinkedHashMap) • LinkedHashMap — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LinkedHashMap • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Building lightweight bounded LRU (Least Recently Used) in-memory caches and deterministic JSON payload maps.

What Topic Is Next → Next Topic 5.17: TreeMap

5.17 TreeMap [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Red-Black tree sorted keys • `O(log n)` get/put • `NavigableMap` (`firstEntry` • `lastEntry` • Production Edge-Case & Pitfall Guard (TreeMap) • Hands-On Java Implementation & Verification (TreeMap) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TreeMap

Purpose: Tax-slab bracket lookups (`floorEntry(income)`), time-series range queries, and canonical request-header signing.

What Topic Is Next → Next Topic 5.18: Hashtable

5.18 Hashtable [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Legacy synchronized dictionary • No null keys or null values • Coarse object-wide locking vs `ConcurrentHashMap` • Production Edge-Case & Pitfall Guard (Hashtable) • Hands-On Java Implementation & Verification (Hashtable) • Hashtable — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hashtable

Purpose: Understanding `Properties` (which extends `Hashtable`) and why `ConcurrentHashMap` replaced `Hashtable` in concurrent servers.

What Topic Is Next → Next Topic 5.19: Iterator and Fail-Fast Behaviour

5.19 Iterator and Fail-Fast Behaviour [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `hasNext()` • `next()` • Safe `iterator.remove()` • `modCount` tracking • Production Edge-Case & Pitfall Guard (Iterator and Fail-Fast Behaviour) • Hands-On Java Implementation & Verification (Iterator and Fail-Fast Behaviour) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Iterator

Purpose: Safely pruning expired sessions or inactive items while traversing a mutable collection.

What Topic Is Next → Next Topic 5.20: ListIterator

5.20 ListIterator [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bidirectional traversal (`hasPrevious` • `previous`) • Index inspection (`nextIndex`) • In-place `set(e)` replacement • Production Edge-Case & Pitfall Guard (ListIterator) • Hands-On Java Implementation & Verification (ListIterator) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ListIterator

Purpose: Walking ordered workflows forward/backward and replacing normalized elements in-place.

What Topic Is Next → Next Topic 5.21: Comparable (Natural Ordering)

5.21 Comparable (Natural Ordering) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Comparable<T>` interface • `compareTo(T other)` contract (sign symmetry • Transitivity) • `Integer.compare` / `Long.compare` overflow safety • Production Edge-Case & Pitfall Guard (Comparable (Natural Ordering)) • Hands-On Java Implementation & Verification (Comparable (Natural Ordering)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comparable (Natural Ordering)

Purpose: Defining default natural sort order on domain models (`Employee` ordered by `employeeId` or `Invoice` by `dueDate`).

What Topic Is Next → Next Topic 5.22: Comparator (Custom Multi-Field Sorting)

5.22 Comparator (Custom Multi-Field Sorting) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Comparator.comparing` • `comparingDouble` • `reversed()` • `thenComparing` • Production Edge-Case & Pitfall Guard (Comparator (Custom Multi-Field Sorting)) • Hands-On Java Implementation & Verification (Comparator (Custom Multi-Field Sorting)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comparator (Custom Multi-Field Sorting)

Purpose: Supporting dynamic API sort options (`sort=salary,desc&sort=name,asc`) in backend search endpoints.

What Topic Is Next → Next Topic 5.23: equals and hashCode Contract in Collections

5.23 equals and hashCode Contract in Collections [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why equal objects MUST produce identical hash codes • Hash collision handling via `equals` • Immutable key fields • JPA entity equality nuances • Production Edge-Case & Pitfall Guard (equals and hashCode Contract in Collections) • Hands-On Java Implementation & Verification (equals and hashCode Contract in Collections) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for equals

Purpose: Preventing duplicate business keys and lost cache entries across `HashSet`, `HashMap`, and Hibernate persistence sets.

What Topic Is Next → Module 5 Capstone (Week 5 Lab Build: Collections-Powered Task & Payroll Store) → Next Topic 6.1: Generics (Classes, Interfaces and Methods)

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Lab Build: Collections-Powered Task & Payroll Store — Implement an in-memory Employee & Priority Task engine using `ArrayList`, `HashSet` deduplication, `HashMap` word/department indexing, `Comparator` multi-field sorting, and `PriorityQueue` u...

MODULE 6 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Generics, Java 8+ and Modern Java Features

Write expressive, type-safe modern Java 21 code using Generics, PECS wildcards, Lambdas, Functional Interfaces, Method References, Stream API pipelines, Collectors, Optional, java.time Date/Time API, Records, Sealed Classes, Pattern Matching, and NIO.2 Files.

Objectives: ✓ Eliminate unsafe raw casts using generic classes, generic methods, bounded type parameters, and PECS (`? extends` / `? super`) wildcards. • ✓ Compose clean functional pipelines using Lambdas, Predicate, Function, Consumer, Supplier, and Method References. • ✓ Transform, filter, flatten, sort, reduce, and group datasets declaratively with the Java Stream API and Collectors. • ✓ Eliminate null-check boilerplate with `Optional`, handle time zones cleanly with `java.time`, and use Java 21 Records, Switch Pattern Matching, Text Blocks, and Sealed Classes. • ✓ Read and write UTF-8 files safely using modern `java.nio.file.Path` and `Files`.

6.1 Generics (Classes, Interfaces and Methods) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Compile-time type safety • Type parameters (`<T • R>`) • Generic `ApiResponse<T>` wrappers • Production Edge-Case & Pitfall Guard (Generics (Classes, Interfaces and Methods)) • Hands-On Java Implementation & Verification (Generics (Classes, Interfaces and Methods)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generics (Classes, Interfaces and Methods)

Purpose: Building reusable `PagedResponse<T>` and `Result<T>` envelopes shared across all REST controllers.

What Topic Is Next → Next Topic 6.2: Bounded Generics

6.2 Bounded Generics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Upper bounds (`<T extends Number & Comparable<T>>`) • Restricting type parameters to domain capabilities • Production Edge-Case & Pitfall Guard (Bounded Generics) • Hands-On Java Implementation & Verification (Bounded Generics) • Bounded Generics — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bounded Generics • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Writing reusable numeric aggregation or `AuditableEntity` repository helpers with compile-time guarantees.

What Topic Is Next → Next Topic 6.3: Wildcards (?, extends and super — PECS)

6.3 Wildcards (?, extends and super — PECS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unbounded `?` • Covariance `? extends T` (Producer Extends) • Contravariance `? super T` (Consumer Super) • Invariance of `List<T>` • Production Edge-Case & Pitfall Guard (Wildcards (?, extends and super — PECS)) • Hands-On Java Implementation & Verification (Wildcards (?, extends and super — PECS)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Wildcards (?, extends and super — PECS)

Purpose: Designing flexible library/service utility methods like `Collections.copy(dest, src)` that accept subtype collections.

What Topic Is Next → Next Topic 6.4: Lambda Expressions

6.4 Lambda Expressions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@FunctionalInterface` • Concise `(params) -> expression` syntax • Target typing • Effectively final variable capture • Production Edge-Case & Pitfall Guard (Lambda Expressions) • Hands-On Java Implementation & Verification (Lambda Expressions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lambda Expressions

Purpose: Passing concise filtering, transformation, and callback strategies without verbose anonymous inner classes.

What Topic Is Next → Next Topic 6.5: Core Functional Interfaces (Predicate, Function, Consumer, Supplier)

6.5 Core Functional Interfaces (Predicate, Function, Consumer, Supplier) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `java.util.function` contracts: `test(T)` • `apply(T)` • `accept(T)` • `get()` • Production Edge-Case & Pitfall Guard (Core Functional Interfaces (Predicate, Function, Consumer, Supplier)) • Hands-On Java Implementation & Verification (Core Functional Interfaces (Predicate, Function, Consumer, Supplier)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Functional Interfaces (Predicate, Function, Con...

Purpose: Composing reusable business validation rules (`Predicate<Order>`), DTO mappers (`Function<Entity, Dto>`), and lazy defaults (`Supplier`).

What Topic Is Next → Next Topic 6.6: Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer)

6.6 Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Two-argument functional contracts (`BiPredicate` • `BiFunction` • `BiConsumer` • `BinaryOperator`) • Production Edge-Case & Pitfall Guard (Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer)) • Hands-On Java Implementation & Verification (Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bi-Arity Functional Interfaces (BiPredicate, BiFunct...

Purpose: Combining two domain inputs (e.g., `(Order, Coupon) -> DiscountedTotal`) and iterating `Map` entries cleanly.

What Topic Is Next → Next Topic 6.7: Method References

6.7 Method References [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Static (`Integer::parseInt`) • Bound instance (`logger::info`) • Unbound instance (`String::toLowerCase`) • Constructor (`ArrayList::new`) references • Production Edge-Case & Pitfall Guard (Method References) • Hands-On Java Implementation & Verification (Method References) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Method References

Purpose: Making Stream pipelines (`map(Employee::department)`, `map(UserDto::fromEntity)`) self-documenting and concise.

What Topic Is Next → Next Topic 6.8: Stream API Architecture and Creation

6.8 Stream API Architecture and Creation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `collection.stream()` • `Stream.of` • `Arrays.stream` • `IntStream.rangeClosed` • Production Edge-Case & Pitfall Guard (Stream API Architecture and Creation) • Hands-On Java Implementation & Verification (Stream API Architecture and Creation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stream API Architecture

Purpose: Expressing declarative in-memory data transformations without mutating source collections.

What Topic Is Next → Next Topic 6.9: Stream Intermediate Operations

6.9 Stream Intermediate Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `filter` • `map` • `mapToDouble` • `flatMap` • Production Edge-Case & Pitfall Guard (Stream Intermediate Operations) • Hands-On Java Implementation & Verification (Stream Intermediate Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stream Intermediate Operations

Purpose: Filtering active employees by salary, flattening nested order line-items with `flatMap`, and paginating in-memory slices.

What Topic Is Next → Next Topic 6.10: Stream Terminal Operations

6.10 Stream Terminal Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `toList()` • `collect` • `reduce` • `count` • Production Edge-Case & Pitfall Guard (Stream Terminal Operations) • Hands-On Java Implementation & Verification (Stream Terminal Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stream Terminal Operations

Purpose: Finding the highest-paid employee (`max`), checking policy compliance (`allMatch`), and aggregating totals (`reduce`).

What Topic Is Next → Next Topic 6.11: Collectors (groupingBy, partitioningBy, joining, mapping)

6.11 Collectors (groupingBy, partitioningBy, joining, mapping) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Collectors.toMap` (with merge function!) • `groupingBy` + downstream `counting`/`averagingDouble`/`mapping` • `partitioningBy` • `joining` • Production Edge-Case & Pitfall Guard (Collectors (groupingBy, partitioningBy, joining, mapping)) • Hands-On Java Implementation & Verification (Collectors (groupingBy, partitioningBy, joining, mapping)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Collectors (groupingBy, partitioningBy, joining, map...

Purpose: Grouping orders by status, computing department headcount & average salary maps, and building comma-separated summaries.

What Topic Is Next → Next Topic 6.12: Optional for Null-Safe Return Contracts

6.12 Optional for Null-Safe Return Contracts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Optional.of` • `Optional.ofNullable` • `Optional.empty` • `map` • Production Edge-Case & Pitfall Guard (Optional for Null-Safe Return Contracts) • Hands-On Java Implementation & Verification (Optional for Null-Safe Return Contracts) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optional for Null-Safe Return Contracts

Purpose: Handling `JpaRepository.findById(id).orElseThrow(() -> new NotFoundException(id))` cleanly across every Spring Boot service.

What Topic Is Next → Next Topic 6.13: Date and Time API (java.time)

6.13 Date and Time API (java.time) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable `LocalDate` • `LocalTime` • `LocalDateTime` • `Instant` • Production Edge-Case & Pitfall Guard (Date and Time API (java.time)) • Hands-On Java Implementation & Verification (Date and Time API (java.time)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Date

Purpose: Storing UTC `Instant` timestamps in PostgreSQL and converting to `Asia/Kolkata` (`ZonedDateTime`) at API presentation boundaries.

What Topic Is Next → Next Topic 6.14: Interface Default and Static Enhancements

6.14 Interface Default and Static Enhancements [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Evolving functional • Domain interfaces with `default` combinators • `static` factories without breaking lambdas • Production Edge-Case & Pitfall Guard (Interface Default and Static Enhancements) • Hands-On Java Implementation & Verification (Interface Default and Static Enhancements) • Interface Default and Static Enhancements — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interface Default

Purpose: Understanding how JDK added `.stream()`, `.forEach()`, and `Comparator.comparing()` while keeping single-abstract-method compatibility.

What Topic Is Next → Next Topic 6.15: Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching)

6.15 Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable `record` DTOs & compact constructors • `sealed` class hierarchies (`permits`) • Multi-line SQL/JSON text blocks (`\"\"\"`) • `switch` pattern matching • Production Edge-Case & Pitfall Guard (Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching)) • Hands-On Java Implementation & Verification (Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modern Java 17/21 Features (Records, Sealed Classes,...

Purpose: Writing zero-boilerplate Spring Boot request/response DTOs as `record`s and modeling exhaustive payment outcomes with `sealed interface`.

What Topic Is Next → Next Topic 6.16: File I/O and NIO.2 (Path, Files and Streams)

6.16 File I/O and NIO.2 (Path, Files and Streams) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `java.nio.file.Path` • `Files.readString` • `Files.writeString` • `Files.lines` (try-with-resources!) • Production Edge-Case & Pitfall Guard (File I/O and NIO.2 (Path, Files and Streams)) • Hands-On Java Implementation & Verification (File I/O and NIO.2 (Path, Files and Streams)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File I/O

Purpose: Exporting/importing CSV employee records in Month 2 Mini-Project 2 and streaming large log files without loading gigabytes into heap.

What Topic Is Next → Module 6 Capstone (Week 6 Lab Build: Functional Stream Analytics & NIO File Exporter) → Next Topic 7.1: JVM Architecture Overview

Module 6 Hands-On Capstone Project (100 Marks): Week 6 Lab Build: Functional Stream Analytics & NIO File Exporter — Build a declarative Java 21 Stream analytics service grouping employees by department, finding top earners, handling missing values safely via `Optional`, modeling sealed results with `recor...

MODULE 7 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: JVM, Multithreading and Concurrency

Understand JVM class loading, Heap/Stack/Metaspace memory layout, Garbage Collection & JIT compilation, and build thread-safe concurrent Java 21 systems using Threads, synchronization, volatile, Atomics, ExecutorService, CompletableFuture, and Concurrent Collections.

Objectives: ✓ Explain JVM ClassLoader phases, Heap vs Thread Stack vs Metaspace memory regions, GC roots/collectors (G1GC/ZGC), and JIT compilation. • ✓ Create and coordinate threads cleanly while understanding OS thread lifecycle states. • ✓ Eliminate race conditions and deadlocks using intrinsic locks, lock ordering, `volatile`, and `java.util.concurrent.atomic` classes. • ✓ Execute parallel workloads safely via bounded `ExecutorService` thread pools, `Callable`/`Future`, and non-blocking `CompletableFuture` pipelines. • ✓ Use `ConcurrentHashMap` and `BlockingQueue` to solve producer-consumer and concurrent caching workloads.

7.1 JVM Architecture Overview [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ClassLoader subsystem • Runtime Data Areas • Execution Engine (Interpreter + JIT + GC) • Native Method Interface (JNI) • Production Edge-Case & Pitfall Guard (JVM Architecture Overview) • Hands-On Java Implementation & Verification (JVM Architecture Overview) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JVM Architecture Overview

Purpose: Sizing container memory (`-Xms`, `-Xmx`) and diagnosing production latency spikes in Spring Boot services.

What Topic Is Next → Next Topic 7.2: Class Loading Subsystem

7.2 Class Loading Subsystem [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bootstrap • Platform • Application ClassLoaders • Parents-Delegation model • Production Edge-Case & Pitfall Guard (Class Loading Subsystem) • Hands-On Java Implementation & Verification (Class Loading Subsystem) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Class Loading Subsystem

Purpose: Troubleshooting classpath conflicts and Spring Boot fat-JAR `LaunchedURLClassLoader` behavior.

What Topic Is Next → Next Topic 7.3: JVM Memory Management (Heap, Stack and Metaspace)

7.3 JVM Memory Management (Heap, Stack and Metaspace) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Young Gen (Eden • Survivor S0/S1) • Old Gen • Per-thread JVM Stacks & frames • Production Edge-Case & Pitfall Guard (JVM Memory Management (Heap, Stack and Metaspace)) • Hands-On Java Implementation & Verification (JVM Memory Management (Heap, Stack and Metaspace)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JVM Memory Management (Heap, Stack and Metaspace)

Purpose: Recognizing why unbounded static caches cause Heap OOM while infinite recursion causes StackOverflowError.

What Topic Is Next → Next Topic 7.4: Garbage Collection Fundamentals

7.4 Garbage Collection Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reachability analysis from GC Roots • Mark-Sweep-Compact • Minor vs Major/Mixed GC • Stop-The-World pauses • Production Edge-Case & Pitfall Guard (Garbage Collection Fundamentals) • Hands-On Java Implementation & Verification (Garbage Collection Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Garbage Collection Fundamentals

Purpose: Preventing memory leaks caused by unclosed listeners, ThreadLocals, or unbounded static maps in long-running servers.

What Topic Is Next → Next Topic 7.5: JIT Compiler (Just-In-Time Compilation)

7.5 JIT Compiler (Just-In-Time Compilation) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: HotSpot profiling • C1 (client/fast compile) & C2 (tiered server optimization) • Method inlining • Escape analysis • Production Edge-Case & Pitfall Guard (JIT Compiler (Just-In-Time Compilation)) • Hands-On Java Implementation & Verification (JIT Compiler (Just-In-Time Compilation)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JIT Compiler (Just-In-Time Compilation)

Purpose: Understanding why Java backend throughput improves after JVM warm-up and why microbenchmarks need warm-up iterations.

What Topic Is Next → Next Topic 7.6: Thread Fundamentals (Process vs Thread)

7.6 Thread Fundamentals (Process vs Thread) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: OS Process isolation vs lightweight shared-heap threads • Context switching • Concurrency vs parallelism • Java 21 platform vs virtual threads intro • Production Edge-Case & Pitfall Guard (Thread Fundamentals (Process vs Thread)) • Hands-On Java Implementation & Verification (Thread Fundamentals (Process vs Thread)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Fundamentals (Process vs Thread)

Purpose: Understanding how embedded Tomcat handles hundreds of simultaneous HTTP requests inside a single Spring Boot JVM process.

What Topic Is Next → Next Topic 7.7: Thread Creation (Thread Class and Runnable Interface)

7.7 Thread Creation (Thread Class and Runnable Interface) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing `Runnable` (preferred composition) vs extending `Thread` • Passing lambdas to `new Thread(...)` • Java 21 `Thread.ofVirtual()` preview • Production Edge-Case & Pitfall Guard (Thread Creation (Thread Class and Runnable Interface)) • Hands-On Java Implementation & Verification (Thread Creation (Thread Class and Runnable Interface)) • Thread Creation (Thread Class and Runnable Interface) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Creation (Thread Class and Runnable Interface...

Purpose: Decoupling task logic (`Runnable`) from the execution mechanism (Thread or ExecutorService).

What Topic Is Next → Next Topic 7.8: Thread Lifecycle States

7.8 Thread Lifecycle States [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Thread.State` enum: `NEW` • `RUNNABLE` • `BLOCKED` • `WAITING` • Production Edge-Case & Pitfall Guard (Thread Lifecycle States) • Hands-On Java Implementation & Verification (Thread Lifecycle States) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Lifecycle States

Purpose: Reading production thread dumps (`jstack`) to spot threads stuck in `BLOCKED` (lock contention) or `WAITING` (pool exhaustion).

What Topic Is Next → Next Topic 7.9: Thread Methods (start, run, sleep, join, interrupt)

7.9 Thread Methods (start, run, sleep, join, interrupt) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Coordinating completion with `join()` • Cooperative cancellation via `interrupt()` & `isInterrupted()` • Restoring interrupt status • Production Edge-Case & Pitfall Guard (Thread Methods (start, run, sleep, join, interrupt)) • Hands-On Java Implementation & Verification (Thread Methods (start, run, sleep, join, interrupt)) • Thread Methods (start, run, sleep, join, interrupt) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Methods (start, run, sleep, join, interrupt)

Purpose: Gracefully shutting down background workers when a Spring Boot container receives `SIGTERM`.

What Topic Is Next → Next Topic 7.10: Synchronization (synchronized Methods and Blocks)

7.10 Synchronization (synchronized Methods and Blocks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Intrinsic monitor locks • Mutual exclusion + happens-before memory visibility • Narrow `synchronized (lock)` blocks vs coarse method locks • Production Edge-Case & Pitfall Guard (Synchronization (synchronized Methods and Blocks)) • Hands-On Java Implementation & Verification (Synchronization (synchronized Methods and Blocks)) • Synchronization (synchronized Methods and Blocks) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronization (synchronized Methods and Blocks)

Purpose: Preventing double-booking in seat/ticket reservation critical sections.

What Topic Is Next → Next Topic 7.11: Race Conditions and Shared Mutable State

7.11 Race Conditions and Shared Mutable State [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Check-then-act (`if (!contains) put`) • Read-modify-write (`count++`) hazards • Stateless service design in Spring • Production Edge-Case & Pitfall Guard (Race Conditions and Shared Mutable State) • Hands-On Java Implementation & Verification (Race Conditions and Shared Mutable State) • Race Conditions and Shared Mutable State — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Race Conditions

Purpose: Keeping Spring `@Service` and `@RestController` singletons 100% stateless so concurrent HTTP threads never corrupt one another's data.

What Topic Is Next → Next Topic 7.12: Deadlocks (Causes and Prevention)

7.12 Deadlocks (Causes and Prevention) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Coffman conditions • Circular lock acquisition (`Thread 1: A->B` vs `Thread 2: B->A`) • Consistent global lock ordering • `ReentrantLock.tryLock(timeout)` • Production Edge-Case & Pitfall Guard (Deadlocks (Causes and Prevention)) • Hands-On Java Implementation & Verification (Deadlocks (Causes and Prevention)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deadlocks (Causes and Prevention)

Purpose: Preventing deadlocks in multi-account wallet transfers by acquiring account locks in deterministic ID order.

What Topic Is Next → Next Topic 7.13: volatile Keyword and Memory Visibility

7.13 volatile Keyword and Memory Visibility [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: CPU cache coherence • Java Memory Model happens-before guarantee • When `volatile` suffices (single-writer boolean flag) vs when it fails (`count++`) • Production Edge-Case & Pitfall Guard (volatile Keyword and Memory Visibility) • Hands-On Java Implementation & Verification (volatile Keyword and Memory Visibility) • volatile Keyword and Memory Visibility — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for volatile Keyword

Purpose: Signaling a background consumer loop (`private volatile boolean running = true;`) to stop immediately across threads.

What Topic Is Next → Next Topic 7.14: Atomic Classes (AtomicInteger, AtomicLong, LongAdder)

7.14 Atomic Classes (AtomicInteger, AtomicLong, LongAdder) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lock-free Compare- • -Swap (CAS) hardware instructions • `incrementAndGet` • `compareAndSet` • Production Edge-Case & Pitfall Guard (Atomic Classes (AtomicInteger, AtomicLong, LongAdder)) • Hands-On Java Implementation & Verification (Atomic Classes (AtomicInteger, AtomicLong, LongAdder)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Atomic Classes (AtomicInteger, AtomicLong, LongAdder...

Purpose: Tracking thread-safe API request metrics, rate-limit counters, and sequence IDs without lock blocking.

What Topic Is Next → Next Topic 7.15: ExecutorService and Thread Pools

7.15 ExecutorService and Thread Pools [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed • Cached • Scheduled • Custom `ThreadPoolExecutor` (corePoolSize • Production Edge-Case & Pitfall Guard (ExecutorService and Thread Pools) • Hands-On Java Implementation & Verification (ExecutorService and Thread Pools) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ExecutorService

Purpose: Executing bounded parallel batch jobs and `@Async` email/report tasks without exhausting JVM memory.

What Topic Is Next → Next Topic 7.16: Callable and Future

7.16 Callable and Future [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Callable<V>` returning typed results & throwing checked exceptions • `Future.get(timeout • Unit)` • `invokeAll` • Production Edge-Case & Pitfall Guard (Callable and Future) • Hands-On Java Implementation & Verification (Callable and Future) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Callable

Purpose: Running parallel downstream service checks with strict SLA timeouts (`future.get(2, TimeUnit.SECONDS)`).

What Topic Is Next → Next Topic 7.17: CompletableFuture (Asynchronous Programming)

7.17 CompletableFuture (Asynchronous Programming) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `supplyAsync` • `thenApply` • `thenCompose` • `thenCombine` • Production Edge-Case & Pitfall Guard (CompletableFuture (Asynchronous Programming)) • Hands-On Java Implementation & Verification (CompletableFuture (Asynchronous Programming)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CompletableFuture (Asynchronous Programming)

Purpose: Fetching User Profile, Order Summary, and Loyalty Points concurrently in an API aggregator in `max(t1, t2, t3)` instead of `t1 + t2 + t3`.

What Topic Is Next → Next Topic 7.18: Concurrent Collections (ConcurrentHashMap and BlockingQueue)

7.18 Concurrent Collections (ConcurrentHashMap and BlockingQueue) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fine-grained bucket/CAS concurrency in `ConcurrentHashMap` • Atomic `computeIfAbsent` • Bounded `ArrayBlockingQueue` (`put`/`take`) for Producer-Consumer • Production Edge-Case & Pitfall Guard (Concurrent Collections (ConcurrentHashMap and BlockingQueue)) • Hands-On Java Implementation & Verification (Concurrent Collections (ConcurrentHashMap and BlockingQueue)) • Concurrent Collections (ConcurrentHashMap and BlockingQueue) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Concurrent Collections (ConcurrentHashMap and Blocki...

Purpose: Implementing thread-safe in-memory token caches and backpressure-safe Producer-Consumer audit pipelines.

What Topic Is Next → Module 7 Capstone (Week 7 Lab Build: Thread-Safe Ticket Booking & Async Aggregator) → Next Topic 8.1: Algorithm Fundamentals

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Lab Build: Thread-Safe Ticket Booking & Async Aggregator — Develop a concurrent seat-reservation simulator and bounded `ArrayBlockingQueue` producer-consumer audit pipeline using locks, `AtomicLong` counters, `ExecutorService`, and `CompletableFutur...

MODULE 8 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 31 TOPICS (248 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Data Structures and Algorithms Foundation & Git/GitHub Workflow

Establish formal Big-O algorithmic problem-solving across arrays, two pointers, sliding window, prefix sum, searching, sorting, recursion, backtracking, linked lists, stacks, queues, and hashing — alongside structured enterprise Git & GitHub branching, PRs, conflict resolution, and Mini-Project 2.

Objectives: ✓ Analyze worst-case and average-case Time & Space Complexity (`O(1)` to `O(2^n)`) before writing code. • ✓ Recognize and apply Two Pointers, Sliding Window, Prefix Sum, Binary Search, Divide-and-Conquer Sorting, Recursion, and Hashing patterns. • ✓ Implement Singly/Doubly Linked Lists, Array/Linked Stacks, and Circular/Deque Queues from scratch in Java 21. • ✓ Execute professional Git workflows: branching, merging, conflict resolution, rebasing, stashing, `reset` vs `revert`, PR reviews, and `.gitignore` hygiene. • ✓ Deliver Month 2 Mini-Project 2 (Employee Management Application) on GitHub alongside 35–45 solved DSA exercises.

8.1 Algorithm Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Definition of an algorithm • Correctness invariants • Edge-case modeling • Dry-running • Production Edge-Case & Pitfall Guard (Algorithm Fundamentals) • Hands-On Java Implementation & Verification (Algorithm Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Algorithm Fundamentals

Purpose: Approaching unfamiliar OA and live coding interview problems methodically instead of guessing.

What Topic Is Next → Next Topic 8.2: Time and Space Complexity Analysis

8.2 Time and Space Complexity Analysis [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Counting elementary operations vs input size `n` • Auxiliary space vs total space • Recursion call-stack space • Amortized complexity • Production Edge-Case & Pitfall Guard (Time and Space Complexity Analysis) • Hands-On Java Implementation & Verification (Time and Space Complexity Analysis) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Time

Purpose: Predicting whether an API endpoint or SQL-post-processing loop will time out when tenant records grow from 100 to 100,000.

What Topic Is Next → Next Topic 8.3: Big O Notation (O(1), O(log n), O(n), O(n log n), O(n²))

8.3 Big O Notation (O(1), O(log n), O(n), O(n log n), O(n²)) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Asymptotic upper bounds • Dropping constants & lower-order terms • Input-constraint-to-acceptable-Big-O cheat sheet • Production Edge-Case & Pitfall Guard (Big O Notation (O(1), O(log n), O(n), O(n log n), O(n²))) • Hands-On Java Implementation & Verification (Big O Notation (O(1), O(log n), O(n), O(n log n), O(n²))) • Big O Notation (O(1), O(log n), O(n), O(n log n), O(n²)) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Big O Notation (O(1), O(log n), O(n), O(n log n), O(...

Purpose: Knowing immediately that `n <= 10^5` requires `O(n)` or `O(n log n)` and rules out `O(n^2)` nested scans.

What Topic Is Next → Next Topic 8.4: Array Problem Patterns

8.4 Array Problem Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: In-place reversal algorithm for `k`-step array rotation • Kadane's maximum subarray sum • Dutch National Flag 3-way partition • Production Edge-Case & Pitfall Guard (Array Problem Patterns) • Hands-On Java Implementation & Verification (Array Problem Patterns) • Array Problem Patterns — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Problem Patterns

Purpose: Solving classic coding screening problems in `O(n)` time and `O(1)` auxiliary space.

What Topic Is Next → Next Topic 8.5: Two Pointers Technique

8.5 Two Pointers Technique [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Converging left/right pointers on sorted arrays (pair sum • Container water • Palindrome) • Fast/slow write-read pointers (move zeroes • Production Edge-Case & Pitfall Guard (Two Pointers Technique) • Hands-On Java Implementation & Verification (Two Pointers Technique) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Two Pointers Technique

Purpose: Reducing `O(n^2)` nested pair scans to `O(n)` linear passes without extra hash memory.

What Topic Is Next → Next Topic 8.6: Sliding Window (Fixed and Variable Window)

8.6 Sliding Window (Fixed and Variable Window) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed-size `k` rolling window sum/average • Variable expand-shrink `[left..right]` window (longest unique substring • Min-size subarray sum) • Production Edge-Case & Pitfall Guard (Sliding Window (Fixed and Variable Window)) • Hands-On Java Implementation & Verification (Sliding Window (Fixed and Variable Window)) • Sliding Window (Fixed and Variable Window) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sliding Window (Fixed and Variable Window)

Purpose: Computing rolling 5-minute API error rates and streaming analytics without recomputing from scratch.

What Topic Is Next → Next Topic 8.7: Prefix Sum and Subarray Queries

8.7 Prefix Sum and Subarray Queries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `prefix[i]` construction for `O(1)` range sum `[L..R]` • Prefix Sum + HashMap for `Subarray Sum Equals K` (including negative numbers) • Production Edge-Case & Pitfall Guard (Prefix Sum and Subarray Queries) • Hands-On Java Implementation & Verification (Prefix Sum and Subarray Queries) • Prefix Sum and Subarray Queries — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prefix Sum • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Answering thousands of financial date-range ledger queries in `O(1)` time after a single `O(n)` precomputation.

What Topic Is Next → Next Topic 8.8: Searching (Linear and Binary Search)

8.8 Searching (Linear and Binary Search) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sequential scan vs `O(log n)` Binary Search • Overflow-safe midpoint `low + (high - low) / 2` • Lower-bound (first occurrence) • Upper-bound • Production Edge-Case & Pitfall Guard (Searching (Linear and Binary Search)) • Hands-On Java Implementation & Verification (Searching (Linear and Binary Search)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Searching (Linear and Binary Search)

Purpose: Fast timestamp log bisection, sorted index lookups, and monotonic threshold search.

What Topic Is Next → Next Topic 8.9: Elementary Sorting (Bubble, Selection and Insertion Sort)

8.9 Elementary Sorting (Bubble, Selection and Insertion Sort) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Swapping invariants • Stability (Bubble & Insertion are stable • Selection is unstable) • `O(n)` best-case adaptive Insertion Sort on nearly sorted data • Production Edge-Case & Pitfall Guard (Elementary Sorting (Bubble, Selection and Insertion Sort)) • Hands-On Java Implementation & Verification (Elementary Sorting (Bubble, Selection and Insertion Sort)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Elementary Sorting (Bubble, Selection and Insertion...

Purpose: Understanding why JDK `TimSort` switches to Insertion Sort for small sub-arrays (`n < 32`).

What Topic Is Next → Next Topic 8.10: Efficient Divide-and-Conquer Sorting (Merge Sort and Quicksort)

8.10 Efficient Divide-and-Conquer Sorting (Merge Sort and Quicksort) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Merge Sort `O(n log n)` worst-case & stability (`O(n)` buffer) vs Quicksort Lomuto/Hoare partitioning (`O(n log n)` average • Randomized pivot) • Production Edge-Case & Pitfall Guard (Efficient Divide-and-Conquer Sorting (Merge Sort and Quicksort)) • Hands-On Java Implementation & Verification (Efficient Divide-and-Conquer Sorting (Merge Sort and Quicksort)) • Efficient Divide-and-Conquer Sorting (Merge Sort and Quicksort) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Efficient Divide-and-Conquer Sorting (Merge Sort and... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining why `Arrays.sort(int[])` uses Dual-Pivot Quicksort while `Arrays.sort(Object[])` uses stable TimSort.

What Topic Is Next → Next Topic 8.11: Recursion Fundamentals

8.11 Recursion Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base case • Recursive leap of faith • Call-stack frame trace • Recurrence relations • Production Edge-Case & Pitfall Guard (Recursion Fundamentals) • Hands-On Java Implementation & Verification (Recursion Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Recursion Fundamentals

Purpose: Traversing hierarchical category trees, org charts, and nested JSON AST structures.

What Topic Is Next → Next Topic 8.12: Backtracking (Subsets, Combinations and Permutations)

8.12 Backtracking (Subsets, Combinations and Permutations) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choose -> Explore -> Unchoose state-space tree template • Pruning invalid branches early • Production Edge-Case & Pitfall Guard (Backtracking (Subsets, Combinations and Permutations)) • Hands-On Java Implementation & Verification (Backtracking (Subsets, Combinations and Permutations)) • Backtracking (Subsets, Combinations and Permutations) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backtracking (Subsets, Combinations and Permutations... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Generating valid schedule combinations, rule configurations, and constraint-satisfaction solutions.

What Topic Is Next → Next Topic 8.13: Linked List Data Structures (Singly, Doubly and Circular)

8.13 Linked List Data Structures (Singly, Doubly and Circular) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom `ListNode` implementation • Dummy sentinel head technique • In-place reversal • Middle node • Production Edge-Case & Pitfall Guard (Linked List Data Structures (Singly, Doubly and Circular)) • Hands-On Java Implementation & Verification (Linked List Data Structures (Singly, Doubly and Circular)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linked List Data Structures (Singly, Doubly and Circ...

Purpose: Mastering pointer manipulation for coding rounds and understanding LRU cache doubly-linked node splicing.

What Topic Is Next → Next Topic 8.14: Stack Implementations and Monotonic Stack Intro

8.14 Stack Implementations and Monotonic Stack Intro [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom bounded Array Stack & Linked-Node Stack • Valid Parentheses • MinStack in `O(1)` • Next Greater Element • Production Edge-Case & Pitfall Guard (Stack Implementations and Monotonic Stack Intro) • Hands-On Java Implementation & Verification (Stack Implementations and Monotonic Stack Intro) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stack Implementations

Purpose: Expression evaluation, syntax validation, and `O(n)` span/next-greater analytics.

What Topic Is Next → Next Topic 8.15: Queue Implementations (Simple, Circular and Deque)

8.15 Queue Implementations (Simple, Circular and Deque) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ring-buffer Circular Queue using `(tail + 1) % capacity` • Queue using Two Stacks • Monotonic Deque for Sliding Window Maximum • Production Edge-Case & Pitfall Guard (Queue Implementations (Simple, Circular and Deque)) • Hands-On Java Implementation & Verification (Queue Implementations (Simple, Circular and Deque)) • Queue Implementations (Simple, Circular and Deque) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Queue Implementations (Simple, Circular and Deque)

Purpose: Understanding ring buffers used in high-throughput loggers, Netty, and Kafka network buffers.

What Topic Is Next → Next Topic 8.16: Hashing Problem Solving (HashMap and HashSet)

8.16 Hashing Problem Solving (HashMap and HashSet) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unsorted Two-Sum in `O(n)` • Group Anagrams • Longest Consecutive Sequence in `O(n)` • First Non-Repeating Character • Production Edge-Case & Pitfall Guard (Hashing Problem Solving (HashMap and HashSet)) • Hands-On Java Implementation & Verification (Hashing Problem Solving (HashMap and HashSet)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hashing Problem Solving (HashMap and HashSet)

Purpose: Trading `O(n)` memory for `O(1)` average key lookups to eliminate quadratic nested loops.

What Topic Is Next → Next Topic 8.17: Version Control Systems Overview

8.17 Version Control Systems Overview [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Centralized vs Distributed VCS • Content-addressable snapshots (SHA-1/SHA-256) • Why every engineering team relies on Git • Production Edge-Case & Pitfall Guard (Version Control Systems Overview) • Hands-On Java Implementation & Verification (Version Control Systems Overview) • Version Control Systems Overview — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Version Control Systems Overview

Purpose: Auditing who changed any line of backend code (`git blame`), rolling back bad releases, and parallel team development.

What Topic Is Next → Next Topic 8.18: Git Installation and Global Configuration

8.18 Git Installation and Global Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git config --global user.name` • `user.email` • `init.defaultBranch main` • `core.autocrlf` (Windows vs Linux line endings) • Production Edge-Case & Pitfall Guard (Git Installation and Global Configuration) • Hands-On Java Implementation & Verification (Git Installation and Global Configuration) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Git Installation

Purpose: Preventing noisy CRLF whole-file diffs between Windows dev laptops and Linux CI/Docker containers.

What Topic Is Next → Next Topic 8.19: Local and Remote Repositories

8.19 Local and Remote Repositories [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Working Tree vs Staging Index vs Local `.git` Object DB vs Remote (`origin`) • `git remote -v` • Production Edge-Case & Pitfall Guard (Local and Remote Repositories) • Hands-On Java Implementation & Verification (Local and Remote Repositories) • Local and Remote Repositories — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Local • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding exactly what moves locally vs over the network during staging, committing, fetching, and pushing.

What Topic Is Next → Next Topic 8.20: Basic Git Commands (init, status, add, diff, commit, log)

8.20 Basic Git Commands (init, status, add, diff, commit, log) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Selective staging (`git add -p`) • `git diff --staged` • Atomic commits • `git log --oneline --graph --decorate` • Production Edge-Case & Pitfall Guard (Basic Git Commands (init, status, add, diff, commit, log)) • Hands-On Java Implementation & Verification (Basic Git Commands (init, status, add, diff, commit, log)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Basic Git Commands (init, status, add, diff, commit,...

Purpose: Reviewing every staged hunk with `git diff --staged` before committing so debug prints or temp credentials never slip in.

What Topic Is Next → Next Topic 8.21: Remote Git Commands (clone, fetch, pull, push)

8.21 Remote Git Commands (clone, fetch, pull, push) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git clone` • `git fetch` (safe metadata sync) vs `git pull` (`fetch` + `merge`/`rebase`) • `git push -u origin <branch>` • Production Edge-Case & Pitfall Guard (Remote Git Commands (clone, fetch, pull, push)) • Hands-On Java Implementation & Verification (Remote Git Commands (clone, fetch, pull, push)) • Remote Git Commands (clone, fetch, pull, push) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Remote Git Commands (clone, fetch, pull, push)

Purpose: Inspecting teammate updates safely via `git fetch origin` before integrating them into your working branch.

What Topic Is Next → Next Topic 8.22: Branching (branch, switch, checkout)

8.22 Branching (branch, switch, checkout) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lightweight branch pointers • `git switch -c feat/...` • Naming conventions (`feat/` • `fix/` • Production Edge-Case & Pitfall Guard (Branching (branch, switch, checkout)) • Hands-On Java Implementation & Verification (Branching (branch, switch, checkout)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Branching (branch, switch, checkout)

Purpose: Isolating every Jira/GitHub ticket in its own feature branch so `main` stays deployable at all times.

What Topic Is Next → Next Topic 8.23: Merging and Resolving Merge Conflicts

8.23 Merging and Resolving Merge Conflicts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast-forward vs 3-way merge commits • Reading `<<<<<<< HEAD` • `=======` • `>>>>>>>` conflict markers • Production Edge-Case & Pitfall Guard (Merging and Resolving Merge Conflicts) • Hands-On Java Implementation & Verification (Merging and Resolving Merge Conflicts) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Merging

Purpose: Cleanly combining concurrent changes to `EmployeeService.java` made by two developers.

What Topic Is Next → Next Topic 8.24: Rebasing Fundamentals

8.24 Rebasing Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git rebase main` linear history replay • Interactive rebase (`git rebase -i`) for cleaning local WIP commits • Golden Rule of Rebasing • Production Edge-Case & Pitfall Guard (Rebasing Fundamentals) • Hands-On Java Implementation & Verification (Rebasing Fundamentals) • Rebasing Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rebasing Fundamentals

Purpose: Keeping your unmerged feature branch cleanly on top of the latest `main` before opening a Pull Request.

What Topic Is Next → Next Topic 8.25: Stashing Temporary Changes

8.25 Stashing Temporary Changes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git stash push -m "..."` • `git stash list` • `git stash pop` • `git stash apply` • Production Edge-Case & Pitfall Guard (Stashing Temporary Changes) • Hands-On Java Implementation & Verification (Stashing Temporary Changes) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stashing Temporary Changes

Purpose: Saving half-finished feature edits cleanly when an urgent production hotfix branch switch is needed.

What Topic Is Next → Next Topic 8.26: Reset vs Revert (Safe History Management)

8.26 Reset vs Revert (Safe History Management) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git reset --soft / --mixed / --hard` (local history rewrite) vs `git revert <sha>` (safe inverse commit for pushed shared branches) • Production Edge-Case & Pitfall Guard (Reset vs Revert (Safe History Management)) • Hands-On Java Implementation & Verification (Reset vs Revert (Safe History Management)) • Reset vs Revert (Safe History Management) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reset vs Revert (Safe History Management) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Rolling back a broken commit on shared `main` safely using `git revert` without rewriting public history.

What Topic Is Next → Next Topic 8.27: .gitignore Hygiene and Secret Prevention

8.27 .gitignore Hygiene and Secret Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ignoring `target/` • `.idea/` • `.env` • `application-local.yml` • Production Edge-Case & Pitfall Guard (.gitignore Hygiene and Secret Prevention) • Hands-On Java Implementation & Verification (.gitignore Hygiene and Secret Prevention) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for .gitignore Hygiene

Purpose: Preventing database passwords, JWT signing keys, and 50MB JAR artifacts from ever entering Git history.

What Topic Is Next → Next Topic 8.28: GitHub Repository Creation and Protection

8.28 GitHub Repository Creation and Protection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Public/private repo setup • Branch protection rules • Topics/tags • Releases • Production Edge-Case & Pitfall Guard (GitHub Repository Creation and Protection) • Hands-On Java Implementation & Verification (GitHub Repository Creation and Protection) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Repository Creation

Purpose: Presenting clean, recruiter-ready portfolio repositories on GitHub with clear topics, releases, and branch rules.

What Topic Is Next → Next Topic 8.29: Pull Requests and Code Review Workflow

8.29 Pull Requests and Code Review Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Opening descriptive PRs (Problem • Solution • Testing Done) • Line-by-line review comments • Production Edge-Case & Pitfall Guard (Pull Requests and Code Review Workflow) • Hands-On Java Implementation & Verification (Pull Requests and Code Review Workflow) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pull Requests

Purpose: Working identically to agile backend engineering teams where zero code merges to `main` without PR review + passing CI.

What Topic Is Next → Next Topic 8.30: Team Collaboration and Feature-Branch Workflow

8.30 Team Collaboration and Feature-Branch Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Trunk-based / GitHub Flow feature branching • Syncing with `origin/main` • Pair code reviews • Avoiding long-lived drift branches • Production Edge-Case & Pitfall Guard (Team Collaboration and Feature-Branch Workflow) • Hands-On Java Implementation & Verification (Team Collaboration and Feature-Branch Workflow) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Team Collaboration

Purpose: Collaborating smoothly across microservice modules without stepping on teammates' files.

What Topic Is Next → Next Topic 8.31: Git Best Practices (Conventional Commits & Engineering README)

8.31 Git Best Practices (Conventional Commits & Engineering README) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Imperative Conventional Commits (`feat:` • `fix:` • `refactor:` • `test:` • Production Edge-Case & Pitfall Guard (Git Best Practices (Conventional Commits & Engineering README)) • Hands-On Java Implementation & Verification (Git Best Practices (Conventional Commits & Engineering README)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Git Best Practices (Conventional Commits & Engineeri...

Purpose: Making hiring managers immediately understand your project architecture within 30 seconds of opening your GitHub link.

What Topic Is Next → Module 8 Capstone (Mini-Project 2: Employee Management Application (Collections, Streams, File I/O & Git)) → Next Topic 9.1: Database, DBMS and RDBMS Introduction

Module 8 Hands-On Capstone Project (100 Marks): Mini-Project 2: Employee Management Application (Collections, Streams, File I/O & Git) — Develop an enterprise-style Core Java 21 Employee Management Application utilizing OOP, `HashMap`/`ArrayList`/`TreeSet` Collections, `Comparator` sorting, Java Stream API analytics, `Optiona...

MODULE 9 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Database Fundamentals and SQL

Build complete fluency in relational database fundamentals and SQL using PostgreSQL (with MySQL syntax comparisons) — DDL, DML, DQL, constraints, 1:1 / 1:N / M:N relationships, INNER/OUTER/SELF joins, scalar & correlated subqueries, aggregations, and set operations.

Objectives: ✓ Design relational tables in PostgreSQL with accurate data types and PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and NOT NULL constraints. • ✓ Write safe DML (`INSERT`, `UPDATE`, `DELETE`) and expressive `SELECT` queries with filtering, pattern matching, and sorting. • ✓ Aggregate business metrics using `GROUP BY` and `HAVING` while respecting SQL logical execution order (`FROM` -> `WHERE` -> `GROUP BY` -> `HAVING` -> `SELECT` -> `ORDER BY`). • ✓ Model 1:1, 1:N, and M:N join tables and write INNER, LEFT, RIGHT, FULL OUTER, and SELF joins accurately. • ✓ Solve multi-step reporting questions using single-row, multi-row, correlated subqueries, and `UNION` / `INTERSECT` / `EXCEPT`.

9.1 Database, DBMS and RDBMS Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: File systems vs DBMS vs RDBMS • Codd's relational model • PostgreSQL vs MySQL overview • Client-server connection lifecycle • Production Edge-Case & Pitfall Guard (Database, DBMS and RDBMS Introduction) • Hands-On Java Implementation & Verification (Database, DBMS and RDBMS Introduction) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Choosing PostgreSQL as the system-of-record store for financial, e-commerce, and HR backend systems.

What Topic Is Next → Next Topic 9.2: Relational Database Architecture

9.2 Relational Database Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Schemas • Tables (relations) • Rows (tuples) • Columns (attributes) • Production Edge-Case & Pitfall Guard (Relational Database Architecture) • Hands-On Java Implementation & Verification (Relational Database Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational Database Architecture

Purpose: Isolating multi-module tables cleanly inside PostgreSQL schemas (`hr`, `payroll`, `audit`).

What Topic Is Next → Next Topic 9.3: SQL Data Types

9.3 SQL Data Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BIGINT` • `INTEGER` • `NUMERIC(p • S)` • Production Edge-Case & Pitfall Guard (SQL Data Types) • Hands-On Java Implementation & Verification (SQL Data Types) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Data Types

Purpose: Using `NUMERIC(12,2)` for exact money amounts and `TIMESTAMPTZ` for timezone-safe UTC timestamps.

What Topic Is Next → Next Topic 9.4: SQL Command Categories (DDL, DML, DQL, DCL and TCL)

9.4 SQL Command Categories (DDL, DML, DQL, DCL and TCL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: DDL (`CREATE/ALTER/DROP/TRUNCATE`) • DML (`INSERT/UPDATE/DELETE`) • DQL (`SELECT`) • DCL (`GRANT/REVOKE`) • Production Edge-Case & Pitfall Guard (SQL Command Categories (DDL, DML, DQL, DCL and TCL)) • Hands-On Java Implementation & Verification (SQL Command Categories (DDL, DML, DQL, DCL and TCL)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Command Categories (DDL, DML, DQL, DCL and TCL)

Purpose: Separating Flyway/Liquibase schema migrations (DDL) from runtime repository operations (DML/DQL/TCL) and least-privilege DB roles (DCL).

What Topic Is Next → Next Topic 9.5: Database and Table Creation (CREATE, ALTER, DROP)

9.5 Database and Table Creation (CREATE, ALTER, DROP) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE TABLE` • `GENERATED ALWAYS AS IDENTITY` • `ALTER TABLE ADD/ALTER/DROP COLUMN` • `DROP TABLE IF EXISTS` • Production Edge-Case & Pitfall Guard (Database and Table Creation (CREATE, ALTER, DROP)) • Hands-On Java Implementation & Verification (Database and Table Creation (CREATE, ALTER, DROP)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Writing repeatable, forward-compatible SQL migration scripts for production schema evolution.

What Topic Is Next → Next Topic 9.6: Data Manipulation (INSERT, UPDATE, DELETE)

9.6 Data Manipulation (INSERT, UPDATE, DELETE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-row `INSERT ... RETURNING` • Targeted `UPDATE ... WHERE` • Safe `DELETE ... WHERE` • `ON CONFLICT` upserts • Production Edge-Case & Pitfall Guard (Data Manipulation (INSERT, UPDATE, DELETE)) • Hands-On Java Implementation & Verification (Data Manipulation (INSERT, UPDATE, DELETE)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Manipulation (INSERT, UPDATE, DELETE)

Purpose: Fetching generated IDs in one round-trip with `INSERT ... RETURNING id` and preventing full-table accidental updates.

What Topic Is Next → Next Topic 9.7: Data Retrieval (SELECT and DISTINCT)

9.7 Data Retrieval (SELECT and DISTINCT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit column projection (avoiding `SELECT *`) • Column aliases • Computed expressions • `COALESCE` • Production Edge-Case & Pitfall Guard (Data Retrieval (SELECT and DISTINCT)) • Hands-On Java Implementation & Verification (Data Retrieval (SELECT and DISTINCT)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Retrieval (SELECT and DISTINCT)

Purpose: Fetching only required DTO columns over the network to reduce database I/O and memory consumption.

What Topic Is Next → Next Topic 9.8: Filtering Rows (WHERE, AND, OR, NOT)

9.8 Filtering Rows (WHERE, AND, OR, NOT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Predicate evaluation • Operator precedence (` • ` binds tighter than `OR`!) • Explicit parentheses • Production Edge-Case & Pitfall Guard (Filtering Rows (WHERE, AND, OR, NOT)) • Hands-On Java Implementation & Verification (Filtering Rows (WHERE, AND, OR, NOT)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Filtering Rows (WHERE, AND, OR, NOT)

Purpose: Building accurate search filters in HR and e-commerce product catalogs.

What Topic Is Next → Next Topic 9.9: SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)

9.9 SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Set membership `IN` • Inclusive range `BETWEEN` • Wildcard `LIKE` (`%` • `_`) / PostgreSQL `ILIKE` • Production Edge-Case & Pitfall Guard (SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)) • Hands-On Java Implementation & Verification (SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)

Purpose: Implementing prefix autocomplete (`LIKE 'rav%'`) and handling nullable `manager_id` / `terminated_at` columns.

What Topic Is Next → Next Topic 9.10: Sorting Results (ORDER BY)

9.10 Sorting Results (ORDER BY) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-column `ASC`/`DESC` sorting • `NULLS LAST` / `NULLS FIRST` • Deterministic tie-breaker columns (`ORDER BY salary DESC • Id ASC`) • Production Edge-Case & Pitfall Guard (Sorting Results (ORDER BY)) • Hands-On Java Implementation & Verification (Sorting Results (ORDER BY)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sorting Results (ORDER BY)

Purpose: Preventing duplicate or skipped records across paginated API pages by always including a unique tie-breaker column.

What Topic Is Next → Next Topic 9.11: Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)

9.11 Aggregation Functions (COUNT, SUM, AVG, MIN, MAX) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `COUNT(*)` vs `COUNT(column)` (ignores NULLs) vs `COUNT(DISTINCT col)` • `SUM` • `ROUND(AVG(...) • 2)` • Production Edge-Case & Pitfall Guard (Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)) • Hands-On Java Implementation & Verification (Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)

Purpose: Generating dashboard KPI cards (total payroll, headcount, unassigned employees, average salary) in a single SQL query.

What Topic Is Next → Next Topic 9.12: Grouping Data (GROUP BY and HAVING)

9.12 Grouping Data (GROUP BY and HAVING) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping keys • Non-aggregated column rule • Pre-aggregation `WHERE` filtering vs post-aggregation `HAVING` filtering • Production Edge-Case & Pitfall Guard (Grouping Data (GROUP BY and HAVING)) • Hands-On Java Implementation & Verification (Grouping Data (GROUP BY and HAVING)) • Grouping Data (GROUP BY and HAVING) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Grouping Data (GROUP BY and HAVING)

Purpose: Finding departments with more than 5 active employees or average salary above ₹80,000.

What Topic Is Next → Next Topic 9.13: Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)

9.13 Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative data integrity • `ON DELETE RESTRICT / CASCADE / SET NULL` • `CHECK (salary > 0)` • Composite unique constraints • Production Edge-Case & Pitfall Guard (Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)) • Hands-On Java Implementation & Verification (Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNI...

Purpose: Guaranteeing zero orphan foreign keys and zero negative prices/salaries even if a bug slips into application code.

What Topic Is Next → Next Topic 9.14: Entity Relationships (1:1, 1:N and M:N)

9.14 Entity Relationships (1:1, 1:N and M:N) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: One-to-One (`employee_profiles` unique FK) • One-to-Many (`departments` -> `employees`) • Many-to-Many (`employees` <-> `projects` via composite-PK join table) • Production Edge-Case & Pitfall Guard (Entity Relationships (1:1, 1:N and M:N)) • Hands-On Java Implementation & Verification (Entity Relationships (1:1, 1:N and M:N)) • Entity Relationships (1:1, 1:N and M:N) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Entity Relationships (1:1, 1:N and M:N)

Purpose: Designing clean join tables (`employee_projects` with `assigned_at`, `role`) for enterprise domain models.

What Topic Is Next → Next Topic 9.15: SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)

9.15 SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ANSI `JOIN ... ON` syntax • Preserving unmatched left rows with `LEFT JOIN` • Finding anti-joins (`LEFT JOIN ... WHERE right.id IS NULL`) • `FULL OUTER JOIN` • Production Edge-Case & Pitfall Guard (SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)) • Hands-On Java Implementation & Verification (SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)

Purpose: Listing all departments including newly created departments with 0 employees (`LEFT JOIN` + `COUNT(e.id)`).

What Topic Is Next → Next Topic 9.16: Self Joins

9.16 Self Joins [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Joining a table to itself using mandatory table aliases (`emp` vs `mgr`) to query hierarchical parent-child rows • Production Edge-Case & Pitfall Guard (Self Joins) • Hands-On Java Implementation & Verification (Self Joins) • Self Joins — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Self Joins • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Listing every employee alongside their reporting manager's name (including CEO via `LEFT JOIN`).

What Topic Is Next → Next Topic 9.17: Subqueries (Scalar and Multi-Row)

9.17 Subqueries (Scalar and Multi-Row) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar subqueries (`salary > (SELECT AVG(salary)...)`) • `IN` subqueries • Derived inline tables in `FROM` • `NOT IN` NULL trap! • Production Edge-Case & Pitfall Guard (Subqueries (Scalar and Multi-Row)) • Hands-On Java Implementation & Verification (Subqueries (Scalar and Multi-Row)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Subqueries (Scalar and Multi-Row)

Purpose: Comparing individual performance or pricing against company-wide or category benchmarks.

What Topic Is Next → Next Topic 9.18: Correlated Subqueries and EXISTS

9.18 Correlated Subqueries and EXISTS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Outer-row-dependent subqueries • `EXISTS` / `NOT EXISTS` short-circuit semi-joins • Comparing against per-department averages • Production Edge-Case & Pitfall Guard (Correlated Subqueries and EXISTS) • Hands-On Java Implementation & Verification (Correlated Subqueries and EXISTS) • Correlated Subqueries and EXISTS — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Correlated Subqueries

Purpose: Finding employees earning above *their own department's* average salary or customers who placed at least one order this month.

What Topic Is Next → Next Topic 9.19: Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)

9.19 Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Column-count & type compatibility • Deduplicating `UNION` vs fast `UNION ALL` • Common rows `INTERSECT` • Set difference `EXCEPT` • Production Edge-Case & Pitfall Guard (Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)) • Hands-On Java Implementation & Verification (Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)

Purpose: Combining active and archived ledger feeds (`UNION ALL`) or auditing discrepancies between two tables (`EXCEPT`).

What Topic Is Next → Module 9 Capstone (Week 9 Lab Build: PostgreSQL Enterprise HR Schema & 40-Query Pack) → Next Topic 10.1: Database Normalization (1NF, 2NF, 3NF and BCNF)

Module 9 Hands-On Capstone Project (100 Marks): Week 9 Lab Build: PostgreSQL Enterprise HR Schema & 40-Query Pack — Author a complete PostgreSQL DDL schema (`departments`, `employees`, `projects`, `employee_projects`) with PK/FK/UNIQUE/CHECK constraints and 40 analytical queries covering INNER/LEFT/SELF j...

MODULE 10 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Advanced SQL and Database Design

Design normalized, high-performance relational schemas in PostgreSQL (with MySQL comparisons) — 1NF–BCNF normalization, composite B-Tree indexing, EXPLAIN ANALYZE query plans, views & materialized views, functions/procedures/triggers, ACID isolation levels, locking/deadlocks, keyset pagination, window functions, and recursive CTEs.

Objectives: ✓ Normalize raw business spreadsheets from unnormalized form through 1NF, 2NF, 3NF, and BCNF while knowing when controlled denormalization makes sense. • ✓ Create single-column, composite, partial, and unique indexes and verify index usage via `EXPLAIN ANALYZE`. • ✓ Manage Views, Materialized Views, SQL Functions, Stored Procedures, and Audit Triggers responsibly. • ✓ Reason accurately about ACID properties, Transaction Isolation Levels, row locks (`SELECT ... FOR UPDATE`), and database deadlocks. • ✓ Write enterprise reporting queries using Keyset Pagination, Window Functions (`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LAG`, `LEAD`), and Recursive CTEs.

10.1 Database Normalization (1NF, 2NF, 3NF and BCNF) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Atomic attributes (1NF) • Eliminating partial dependencies on composite keys (2NF) • Eliminating transitive dependencies (3NF) • Boyce-Codd Normal Form (BCNF) • Production Edge-Case & Pitfall Guard (Database Normalization (1NF, 2NF, 3NF and BCNF)) • Hands-On Java Implementation & Verification (Database Normalization (1NF, 2NF, 3NF and BCNF)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Normalization (1NF, 2NF, 3NF and BCNF)

Purpose: Eliminating update, insertion, and deletion anomalies when designing order, inventory, and HR schemas.

What Topic Is Next → Next Topic 10.2: Strategic Denormalization

10.2 Strategic Denormalization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-heavy vs write-heavy trade-offs • Snapshotting historical facts (`order_items.unit_price_at_purchase`) • Summary caches • Production Edge-Case & Pitfall Guard (Strategic Denormalization) • Hands-On Java Implementation & Verification (Strategic Denormalization) • Strategic Denormalization — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strategic Denormalization

Purpose: Storing `unit_price` snapshot on `order_items` so future catalog price edits never alter historical invoices!

What Topic Is Next → Next Topic 10.3: Database Indexes (Single-Column, Composite and Unique)

10.3 Database Indexes (Single-Column, Composite and Unique) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: B-Tree structure (`O(log n)`) • Composite index leftmost-prefix rule `(department_id • Salary)` • Unique & partial indexes • Production Edge-Case & Pitfall Guard (Database Indexes (Single-Column, Composite and Unique)) • Hands-On Java Implementation & Verification (Database Indexes (Single-Column, Composite and Unique)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Indexes (Single-Column, Composite and Uniqu...

Purpose: Accelerating foreign-key joins (PostgreSQL does NOT auto-index FK columns!) and multi-column search filters.

What Topic Is Next → Next Topic 10.4: Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

10.4 Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Seq Scan` vs `Index Scan` vs `Index Only Scan` vs `Bitmap Heap Scan` • `Nested Loop` / `Hash Join` / `Merge Join` • Estimated vs actual rows • Production Edge-Case & Pitfall Guard (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Hands-On Java Implementation & Verification (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production

Runtime Execution & Memory Lifecycle for Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

Purpose: Proving with empirical millisecond evidence why an API query is slow before deploying an index migration.

What Topic Is Next → Next Topic 10.5: SQL Query Optimization

10.5 SQL Query Optimization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: SARGable predicates (avoiding functions on indexed columns like `LOWER(email)` without functional index) • Avoiding `SELECT *` • Covering indexes • Production Edge-Case & Pitfall Guard (SQL Query Optimization) • Hands-On Java Implementation & Verification (SQL Query Optimization) • SQL Query Optimization — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Query Optimization

Purpose: Rewriting `WHERE EXTRACT(YEAR FROM created_at) = 2026` into a SARGable range `created_at >= '2026-01-01' AND created_at < '2027-01-01'` so the B-Tree index is used!

What Topic Is Next → Next Topic 10.6: Database Views

10.6 Database Views [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE OR REPLACE VIEW` • Security column masking (hiding salary/PII) • Simplifying complex reusable joins • Production Edge-Case & Pitfall Guard (Database Views) • Hands-On Java Implementation & Verification (Database Views) • Database Views — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Views

Purpose: Exposing a clean `v_public_employee_directory` view without sensitive payroll columns to reporting consumers.

What Topic Is Next → Next Topic 10.7: Materialized Views

10.7 Materialized Views [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE MATERIALIZED VIEW` • Persisted query results on disk • Indexing materialized views • `REFRESH MATERIALIZED VIEW CONCURRENTLY` • Production Edge-Case & Pitfall Guard (Materialized Views) • Hands-On Java Implementation & Verification (Materialized Views) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Materialized Views

Purpose: Precomputing heavy analytical dashboards refreshed every 15 minutes instead of scanning millions of rows on every page load.

What Topic Is Next → Next Topic 10.8: Stored Procedures

10.8 Stored Procedures [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PostgreSQL `CREATE PROCEDURE` (`plpgsql`) • `CALL` • Transactional control inside procedures • When logic belongs in Java Service vs DB procedure • Production Edge-Case & Pitfall Guard (Stored Procedures) • Hands-On Java Implementation & Verification (Stored Procedures) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stored Procedures

Purpose: Executing bulk archival or data-cleanup DBA routines close to the data engine.

What Topic Is Next → Next Topic 10.9: User-Defined SQL Functions

10.9 User-Defined SQL Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar • Table-returning functions (`CREATE FUNCTION ... RETURNS`) • SQL vs PL/pgSQL functions • `IMMUTABLE` / `STABLE` / `VOLATILE` markers • Production Edge-Case & Pitfall Guard (User-Defined SQL Functions) • Hands-On Java Implementation & Verification (User-Defined SQL Functions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User-Defined SQL Functions

Purpose: Encapsulating reusable fiscal-year or net-compensation calculations inside reporting SQL queries.

What Topic Is Next → Next Topic 10.10: Database Triggers

10.10 Database Triggers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BEFORE` vs `AFTER` row/statement triggers • `NEW` • `OLD` record variables • Automated `updated_at` stamping • Production Edge-Case & Pitfall Guard (Database Triggers) • Hands-On Java Implementation & Verification (Database Triggers) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Triggers

Purpose: Recording tamper-evident salary-change audit rows automatically whenever `hr.employees.salary` changes.

What Topic Is Next → Next Topic 10.11: Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)

10.11 Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: All-or-nothing atomic work units • `BEGIN` • `COMMIT` • `ROLLBACK` • Production Edge-Case & Pitfall Guard (Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)) • Hands-On Java Implementation & Verification (Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)

Purpose: Ensuring wallet debit + ledger credit + order status update either all commit together or all roll back cleanly on failure.

What Topic Is Next → Next Topic 10.12: ACID Properties

10.12 ACID Properties [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Atomicity (WAL undo/abort) • Consistency (constraints/invariants) • Isolation (MVCC concurrency) • Durability (WAL fsync crash recovery) • Production Edge-Case & Pitfall Guard (ACID Properties) • Hands-On Java Implementation & Verification (ACID Properties) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ACID Properties

Purpose: Explaining in backend interviews *how* PostgreSQL guarantees zero corrupted balances across power loss and concurrent traffic.

What Topic Is Next → Next Topic 10.13: Transaction Isolation Levels

10.13 Transaction Isolation Levels [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dirty Read • Non-Repeatable Read • Phantom Read • Write Skew • Production Edge-Case & Pitfall Guard (Transaction Isolation Levels) • Hands-On Java Implementation & Verification (Transaction Isolation Levels) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transaction Isolation Levels

Purpose: Preventing concurrent double-spend or lost updates by choosing Repeatable Read / Serializable or explicit row locking.

What Topic Is Next → Next Topic 10.14: Database Locking (Shared and Exclusive Locks)

10.14 Database Locking (Shared and Exclusive Locks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Row-level `FOR SHARE` vs `FOR UPDATE` • `FOR UPDATE SKIP LOCKED` (job queues) • Pessimistic vs optimistic concurrency control • Production Edge-Case & Pitfall Guard (Database Locking (Shared and Exclusive Locks)) • Hands-On Java Implementation & Verification (Database Locking (Shared and Exclusive Locks)) • Database Locking (Shared and Exclusive Locks) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Locking (Shared and Exclusive Locks)

Purpose: Locking an inventory or wallet row with `SELECT ... FOR UPDATE` during checkout, or polling outbox events via `FOR UPDATE SKIP LOCKED`.

What Topic Is Next → Next Topic 10.15: Database Deadlocks

10.15 Database Deadlocks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cross-row lock cycles between two transactions • PostgreSQL deadlock detector (`deadlock_timeout`) • Automatic victim abort (`SQLSTATE 40P01`) • Consistent primary-key update ordering • Production Edge-Case & Pitfall Guard (Database Deadlocks) • Hands-On Java Implementation & Verification (Database Deadlocks) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Deadlocks

Purpose: Sorting item IDs ascending before batch `UPDATE` / `FOR UPDATE` statements so concurrent checkouts never deadlock!

What Topic Is Next → Next Topic 10.16: Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)

10.16 Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `OFFSET 100000 LIMIT 20` still scans & discards 100 • 000 rows • Vs `O(log n)` Keyset/Cursor pagination (`WHERE (created_at • Id) < (? • Production Edge-Case & Pitfall Guard (Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)) • Hands-On Java Implementation & Verification (Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Efficient SQL Pagination (LIMIT/OFFSET vs Keyset See...

Purpose: Keeping infinite-scroll feeds and large order-export APIs sub-millisecond at page 5,000.

What Topic Is Next → Next Topic 10.17: Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)

10.17 Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `OVER (PARTITION BY ... ORDER BY ...)` • Top-N per group (`DENSE_RANK`) • Running totals (`SUM(...) OVER`) • Month-over-month deltas (`LAG`/`LEAD`) • Production Edge-Case & Pitfall Guard (Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)) • Hands-On Java Implementation & Verification (Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD...

Purpose: Finding the Top 3 highest-paid employees *per department* (the #1 SQL interview problem!) and computing running account balances.

What Topic Is Next → Next Topic 10.18: Common Table Expressions (CTEs and Recursive CTEs)

10.18 Common Table Expressions (CTEs and Recursive CTEs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WITH` readable modular queries • Multi-CTE pipelines • `WITH RECURSIVE` anchor + recursive union for org-chart & category hierarchies • Production Edge-Case & Pitfall Guard (Common Table Expressions (CTEs and Recursive CTEs)) • Hands-On Java Implementation & Verification (Common Table Expressions (CTEs and Recursive CTEs)) • Common Table Expressions (CTEs and Recursive CTEs) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Common Table Expressions (CTEs and Recursive CTEs)

Purpose: Traversing an arbitrary-depth reporting hierarchy (`manager_id`) or multi-level e-commerce category tree in a single SQL query.

What Topic Is Next → Module 10 Capstone (Week 10 Lab Build: 3NF Schema Normalization, Index Tuning & Window Analytics) → Next Topic 11.1: JDBC Architecture and Workflow

Module 10 Hands-On Capstone Project (100 Marks): Week 10 Lab Build: 3NF Schema Normalization, Index Tuning & Window Analytics — Normalize an unnormalized enterprise dataset into 3NF/BCNF, benchmark composite B-Tree indexes on 100k rows using `EXPLAIN ANALYZE`, and build Top-N per department `DENSE_RANK` + `WITH RECUR...

MODULE 11 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 11: JDBC and Database Connectivity

Connect Java 21 applications directly to PostgreSQL using the JDBC API — Type-4 PostgreSQL driver, Connections, PreparedStatement, ResultSet mapping, CRUD, batch inserts, manual ACID transactions, HikariCP DataSource connection pooling, SQL injection prevention, DAO architecture, and try-with-resources.

Objectives: ✓ Understand how Spring Data JPA and Hibernate execute SQL under the hood via standard `java.sql` JDBC interfaces. • ✓ Execute parameterized CRUD and batch operations using `PreparedStatement` with zero SQL injection vulnerabilities. • ✓ Control multi-statement ACID transactions manually (`setAutoCommit(false)`, `commit()`, `rollback()`). • ✓ Configure `DataSource` connection pooling (HikariCP) instead of opening raw TCP connections per query. • ✓ Structure clean DAO (Data Access Object) classes with `try-with-resources` and custom persistence exception translation.

11.1 JDBC Architecture and Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `java.sql` / `javax.sql` interfaces vs vendor driver implementations (`org.postgresql:postgresql`) • 6-step query lifecycle • Production Edge-Case & Pitfall Guard (JDBC Architecture and Workflow) • Hands-On Java Implementation & Verification (JDBC Architecture and Workflow) • JDBC Architecture and Workflow — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JDBC Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding the exact foundational layer beneath Spring `JdbcClient`, `JdbcTemplate`, MyBatis, and Hibernate.

What Topic Is Next → Next Topic 11.2: JDBC Drivers and Modern SPI Loading

11.2 JDBC Drivers and Modern SPI Loading [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Historical Driver Types 1–4 • Why pure-Java Type-4 drivers dominate production • Automatic `ServiceLoader` discovery (no `Class.forName` needed) • Production Edge-Case & Pitfall Guard (JDBC Drivers and Modern SPI Loading) • Hands-On Java Implementation & Verification (JDBC Drivers and Modern SPI Loading) • JDBC Drivers and Modern SPI Loading — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JDBC Drivers

Purpose: Configuring clean JDBC URLs (`jdbc:postgresql://localhost:5432/academy_hr`) via environment variables.

What Topic Is Next → Next Topic 11.3: Establishing Database Connections

11.3 Establishing Database Connections [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DriverManager.getConnection` • Network TCP/TLS handshake cost • Connection validation (`isValid`) • Externalized credentials • Production Edge-Case & Pitfall Guard (Establishing Database Connections) • Hands-On Java Implementation & Verification (Establishing Database Connections) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Establishing Database Connections

Purpose: Verifying database reachability at startup and keeping secrets out of Git.

What Topic Is Next → Next Topic 11.4: Statement Interface and Static DDL/SQL

11.4 Statement Interface and Static DDL/SQL [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Connection.createStatement()` • `executeQuery` vs `executeUpdate` vs `execute` • Why `Statement` must NEVER take user input • Production Edge-Case & Pitfall Guard (Statement Interface and Static DDL/SQL) • Hands-On Java Implementation & Verification (Statement Interface and Static DDL/SQL) • Statement Interface and Static DDL/SQL — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Statement Interface

Purpose: Running fixed, parameterless schema bootstrap scripts in integration test setups.

What Topic Is Next → Next Topic 11.5: PreparedStatement and Parameterized Queries

11.5 PreparedStatement and Parameterized Queries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `?` positional placeholders (1-based index!) • `setLong` • `setString` • `setBigDecimal` • Production Edge-Case & Pitfall Guard (PreparedStatement and Parameterized Queries) • Hands-On Java Implementation & Verification (PreparedStatement and Parameterized Queries) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PreparedStatement

Purpose: Executing 100% injection-safe parameterized SQL and retrieving generated primary keys cleanly.

What Topic Is Next → Next Topic 11.6: CallableStatement (Calling Stored Procedures & Functions)

11.6 CallableStatement (Calling Stored Procedures & Functions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `prepareCall("CALL hr.give_department_hike(? • ?)")` • `IN` / `OUT` / `INOUT` parameters • `registerOutParameter` • Production Edge-Case & Pitfall Guard (CallableStatement (Calling Stored Procedures & Functions)) • Hands-On Java Implementation & Verification (CallableStatement (Calling Stored Procedures & Functions)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CallableStatement (Calling Stored Procedures & Funct...

Purpose: Invoking PostgreSQL stored procedures or legacy financial functions cleanly from Java.

What Topic Is Next → Next Topic 11.7: ResultSet Navigation and Row Mapping

11.7 ResultSet Navigation and Row Mapping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Forward-only cursor `rs.next()` • Column-name getters (`getLong("id")` • `getBigDecimal("salary")` • `getObject("hired_at" • Production Edge-Case & Pitfall Guard (ResultSet Navigation and Row Mapping) • Hands-On Java Implementation & Verification (ResultSet Navigation and Row Mapping) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ResultSet Navigation

Purpose: Mapping relational rows into immutable Java 21 DTO/domain objects without `NULL` corruption.

What Topic Is Next → Next Topic 11.8: JDBC CRUD Operations

11.8 JDBC CRUD Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Complete Create • Read-by-ID (`Optional<Employee>`) • Read-All-Paginated • Update (checking affected row count) • Production Edge-Case & Pitfall Guard (JDBC CRUD Operations) • Hands-On Java Implementation & Verification (JDBC CRUD Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JDBC CRUD Operations

Purpose: Implementing deterministic data access methods with explicit affected-row verification.

What Topic Is Next → Next Topic 11.9: JDBC Batch Processing

11.9 JDBC Batch Processing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `addBatch()` • `executeBatch()` • PostgreSQL `reWriteBatchedInserts=true` driver flag • Chunking batches of 500–1 • Production Edge-Case & Pitfall Guard (JDBC Batch Processing) • Hands-On Java Implementation & Verification (JDBC Batch Processing) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JDBC Batch Processing

Purpose: Importing 10,000 payroll rows in ~50ms via batched network round-trips instead of 10,000 individual round-trips (~8 seconds).

What Topic Is Next → Next Topic 11.10: JDBC Transaction Management (Commit and Rollback)

11.10 JDBC Transaction Management (Commit and Rollback) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `conn.setAutoCommit(false)` • Executing multiple `PreparedStatement`s on the SAME `Connection` • `conn.commit()` • `conn.rollback()` in `catch` • Production Edge-Case & Pitfall Guard (JDBC Transaction Management (Commit and Rollback)) • Hands-On Java Implementation & Verification (JDBC Transaction Management (Commit and Rollback)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JDBC Transaction Management (Commit and Rollback)

Purpose: Transferring budget across departments and writing an audit log atomically on one connection.

What Topic Is Next → Next Topic 11.11: Connection Pooling Concepts and HikariCP

11.11 Connection Pooling Concepts and HikariCP [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why connection creation is expensive • Pool sizing formula (`connections = (core_count * 2) + disk_spindles`) • `maximumPoolSize` • `connectionTimeout` • Production Edge-Case & Pitfall Guard (Connection Pooling Concepts and HikariCP) • Hands-On Java Implementation & Verification (Connection Pooling Concepts and HikariCP) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection Pooling Concepts

Purpose: Configuring HikariCP (Spring Boot's default pool!) so 200 concurrent HTTP threads share a lean pool of 10–20 PostgreSQL connections.

What Topic Is Next → Next Topic 11.12: javax.sql.DataSource Abstraction

11.12 javax.sql.DataSource Abstraction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Injecting `DataSource` into DAOs instead of `DriverManager` • Swapping H2/Testcontainers in tests vs HikariCP PostgreSQL in prod • Production Edge-Case & Pitfall Guard (javax.sql.DataSource Abstraction) • Hands-On Java Implementation & Verification (javax.sql.DataSource Abstraction) • javax.sql.DataSource Abstraction — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for javax.sql.DataSource Abstraction • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Making DAO classes 100% decoupled from connection creation so integration tests inject test databases effortlessly.

What Topic Is Next → Next Topic 11.13: SQL Injection Vulnerabilities and Prevention

11.13 SQL Injection Vulnerabilities and Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How tautology (`' OR '1'='1`) • UNION-based • Stacked-query injections work • Bind-parameter separation of SQL AST from data • Production Edge-Case & Pitfall Guard (SQL Injection Vulnerabilities and Prevention) • Hands-On Java Implementation & Verification (SQL Injection Vulnerabilities and Prevention) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Injection Vulnerabilities

Purpose: Securing dynamic search and sort endpoints (bind `?` for values + strict enum/Set whitelist for dynamic sort column names!).

What Topic Is Next → Next Topic 11.14: DAO (Data Access Object) Design Pattern

11.14 DAO (Data Access Object) Design Pattern [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Interface `EmployeeDao` + `JdbcEmployeeDao` implementation • Separating SQL persistence from domain service rules • Production Edge-Case & Pitfall Guard (DAO (Data Access Object) Design Pattern) • Hands-On Java Implementation & Verification (DAO (Data Access Object) Design Pattern) • DAO (Data Access Object) Design Pattern — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DAO (Data Access Object) Design Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping SQL strings isolated in the persistence layer so Month 3 Week 12 can swap in a JPA/Hibernate implementation cleanly.

What Topic Is Next → Next Topic 11.15: SQLException Handling and SQLState Codes

11.15 SQLException Handling and SQLState Codes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `e.getSQLState()` • PostgreSQL unique violation (`23505`) & foreign-key violation (`23503`) • Wrapping checked `SQLException` into meaningful unchecked `DataAccessException` • Production Edge-Case & Pitfall Guard (SQLException Handling and SQLState Codes) • Hands-On Java Implementation & Verification (SQLException Handling and SQLState Codes) • SQLException Handling and SQLState Codes — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQLException Handling

Purpose: Translating PostgreSQL `23505` duplicate-email constraint errors into clean `DuplicateEmailException` (HTTP 409 Conflict).

What Topic Is Next → Next Topic 11.16: Try-With-Resources in JDBC

11.16 Try-With-Resources in JDBC [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested automatic closing order (`ResultSet` -> `PreparedStatement` -> `Connection` returned to pool) • Suppressed exceptions • Production Edge-Case & Pitfall Guard (Try-With-Resources in JDBC) • Hands-On Java Implementation & Verification (Try-With-Resources in JDBC) • Try-With-Resources in JDBC — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Try-With-Resources in JDBC • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Guaranteeing zero connection leaks back to HikariCP even when SQL queries throw exceptions.

What Topic Is Next → Module 11 Capstone (Week 11 Lab Build: Pooled JDBC Employee DAO Application) → Next Topic 12.1: ORM (Object-Relational Mapping) Concepts

Module 11 Hands-On Capstone Project (100 Marks): Week 11 Lab Build: Pooled JDBC Employee DAO Application — Create a HikariCP-pooled JDBC Employee & Department persistence application implementing the DAO pattern, parameterized `PreparedStatement` CRUD, batch imports, manual ACID commit/rollback,...

MODULE 12 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 12: JPA, Hibernate and ORM

Master Jakarta Persistence (JPA 3.1) and Hibernate 6 ORM — entity lifecycle states, EntityManager persistence context, primary-key strategies, 1:1 / 1:N / M:1 / M:N associations, lazy vs eager fetching, cascading, JPQL, native & derived queries, caching, solving N+1 queries, optimistic locking (@Version), pagination, and Mini-Project 3.

Objectives: ✓ Explain the object-relational impedance mismatch and distinguish Jakarta Persistence (JPA specification) from Hibernate 6 (provider). • ✓ Map relational schemas cleanly using `@Entity`, `@Table`, `@Id`, `@Column`, `@ManyToOne`, `@OneToMany`, `@OneToOne`, and `@ManyToMany`. • ✓ Trace Transient, Managed, Detached, and Removed entity states inside the Persistence Context (First-Level Cache) and dirty checking. • ✓ Write optimized JPQL, JOIN FETCH, DTO projections, and Pageable queries while eliminating LazyInitializationException and N+1 select storms. • ✓ Protect concurrent updates with `@Version` optimistic locking and deliver Mini-Project 3 (Database-Driven Employee Management System).

12.1 ORM (Object-Relational Mapping) Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object graph vs relational tables impedance mismatch (granularity • Inheritance • Associations • Navigation) • Production Edge-Case & Pitfall Guard (ORM (Object-Relational Mapping) Concepts) • Hands-On Java Implementation & Verification (ORM (Object-Relational Mapping) Concepts) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ORM (Object-Relational Mapping) Concepts

Purpose: Eliminating repetitive CRUD mapping boilerplate while retaining full SQL awareness for complex queries.

What Topic Is Next → Next Topic 12.2: Hibernate 6 Architecture and Configuration

12.2 Hibernate 6 Architecture and Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SessionFactory` (`EntityManagerFactory`) • `Session` (`EntityManager`) • `Transaction` • JDBC/HikariCP integration • Production Edge-Case & Pitfall Guard (Hibernate 6 Architecture and Configuration) • Hands-On Java Implementation & Verification (Hibernate 6 Architecture and Configuration) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hibernate 6 Architecture

Purpose: Configuring `ddl-auto: validate` in production so Hibernate verifies schema compatibility without altering live tables unpredictably.

What Topic Is Next → Next Topic 12.3: Jakarta Persistence (JPA) Specification vs Hibernate

12.3 Jakarta Persistence (JPA) Specification vs Hibernate [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `jakarta.persistence.*` standard API vs Hibernate provider implementation • Portability • `javax` -> `jakarta` namespace in Java 17/21 & Spring Boot 3 • Production Edge-Case & Pitfall Guard (Jakarta Persistence (JPA) Specification vs Hibernate) • Hands-On Java Implementation & Verification (Jakarta Persistence (JPA) Specification vs Hibernate) • Jakarta Persistence (JPA) Specification vs Hibernate — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Jakarta Persistence (JPA) Specification vs Hibernate

Purpose: Importing `jakarta.persistence.*` annotations consistently across all Spring Boot 3.x enterprise microservices.

What Topic Is Next → Next Topic 12.4: Entity Classes and Entity Lifecycle

12.4 Entity Classes and Entity Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Entity` rules (protected/public no-arg constructor • Non-final class) • Transient -> Managed -> Detached -> Removed states • Production Edge-Case & Pitfall Guard (Entity Classes and Entity Lifecycle) • Hands-On Java Implementation & Verification (Entity Classes and Entity Lifecycle) • Entity Classes and Entity Lifecycle — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Entity Classes

Purpose: Understanding why modifying a Managed entity inside a transaction automatically issues an `UPDATE` at flush time (dirty checking!).

What Topic Is Next → Next Topic 12.5: EntityManager and Persistence Context Operations

12.5 EntityManager and Persistence Context Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `EntityManager` thread-unsafety (one per request/transaction) • Unit-of-work transactional write-behind • `flush()` • `clear()` • Production Edge-Case & Pitfall Guard (EntityManager and Persistence Context Operations) • Hands-On Java Implementation & Verification (EntityManager and Persistence Context Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for EntityManager

Purpose: Batch-persisting entities cleanly by calling `flush()` + `clear()` periodically so the Persistence Context does not bloat heap.

What Topic Is Next → Next Topic 12.6: Persistence Operations (persist, find, merge, remove)

12.6 Persistence Operations (persist, find, merge, remove) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `persist` (new entity) • `find` vs `getReference` (lazy proxy) • `merge` (returns managed copy!) • `remove` • Production Edge-Case & Pitfall Guard (Persistence Operations (persist, find, merge, remove)) • Hands-On Java Implementation & Verification (Persistence Operations (persist, find, merge, remove)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Persistence Operations (persist, find, merge, remove...

Purpose: Using `em.getReference(Department.class, deptId)` to set a foreign key without firing an unnecessary `SELECT` on `departments`!

What Topic Is Next → Next Topic 12.7: Primary Key Identifier Generation Strategies

12.7 Primary Key Identifier Generation Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `GenerationType.IDENTITY` • `SEQUENCE` (`@SequenceGenerator` with `allocationSize` for JDBC batching!) • `UUID` • Production Edge-Case & Pitfall Guard (Primary Key Identifier Generation Strategies) • Hands-On Java Implementation & Verification (Primary Key Identifier Generation Strategies) • Primary Key Identifier Generation Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Primary Key Identifier Generation Strategies

Purpose: Choosing PostgreSQL `SEQUENCE` when high-throughput Hibernate batch inserts are required (`IDENTITY` disables Hibernate insert batching!).

What Topic Is Next → Next Topic 12.8: Entity Relationships (OneToOne, OneToMany, ManyToOne, ManyToMany)

12.8 Entity Relationships (OneToOne, OneToMany, ManyToOne, ManyToMany) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Owning side (holds `@JoinColumn` FK) vs inverse `mappedBy` side • Bidirectional helper methods (`addEmployee`) • `@JoinTable` vs explicit join entity • Production Edge-Case & Pitfall Guard (Entity Relationships (OneToOne, OneToMany, ManyToOne, ManyToMany)) • Hands-On Java Implementation & Verification (Entity Relationships (OneToOne, OneToMany, ManyToOne, ManyToMany)) • Entity Relationships (OneToOne, OneToMany, ManyToOne, ManyToMany) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Entity Relationships (OneToOne, OneToMany, ManyToOne...

Purpose: Modeling `Department` (1) <-> (N) `Employee` where `Employee` owns `department_id` via `@ManyToOne(fetch = FetchType.LAZY)`.

What Topic Is Next → Next Topic 12.9: Mapping Annotations (@Table, @Column, @JoinColumn, @Enumerated)

12.9 Mapping Annotations (@Table, @Column, @JoinColumn, @Enumerated) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Schema/index metadata on `@Table` • `@Column(nullable • Length • Unique • Production Edge-Case & Pitfall Guard (Mapping Annotations (@Table, @Column, @JoinColumn, @Enumerated)) • Hands-On Java Implementation & Verification (Mapping Annotations (@Table, @Column, @JoinColumn, @Enumerated)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mapping Annotations (@Table, @Column, @JoinColumn, @...

Purpose: Persisting Java enums as readable strings (`EnumType.STRING`) and exact `BigDecimal` money columns (`precision = 12, scale = 2`).

What Topic Is Next → Next Topic 12.10: Fetching Strategies (Lazy vs Eager Loading)

12.10 Fetching Strategies (Lazy vs Eager Loading) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JPA defaults (`@ManyToOne`/`@OneToOne` default to EAGER! • `@OneToMany`/`@ManyToMany` default to LAZY) • Why ALL associations should be `LAZY` • `LazyInitializationException` • Production Edge-Case & Pitfall Guard (Fetching Strategies (Lazy vs Eager Loading)) • Hands-On Java Implementation & Verification (Fetching Strategies (Lazy vs Eager Loading)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Fetching Strategies (Lazy vs Eager Loading)

Purpose: Explicitly setting `fetch = FetchType.LAZY` on every `@ManyToOne` and `@OneToOne` so loading an `Employee` never drags in the entire object graph unintentionally!

What Topic Is Next → Next Topic 12.11: Cascading and Orphan Removal

12.11 Cascading and Orphan Removal [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CascadeType.PERSIST` • `MERGE` • `REMOVE` • `ALL` • Production Edge-Case & Pitfall Guard (Cascading and Orphan Removal) • Hands-On Java Implementation & Verification (Cascading and Orphan Removal) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cascading

Purpose: Saving an `Order` together with its `OrderItem` children atomically while NEVER cascading `REMOVE` onto shared lookup entities like `Department` or `Role`!

What Topic Is Next → Next Topic 12.12: JPQL (Java Persistence Query Language)

12.12 JPQL (Java Persistence Query Language) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Entity/field-centric query syntax • Named `:param` bindings • `JOIN` / `LEFT JOIN FETCH` • Constructor DTO expressions (`SELECT new ...EmployeeSummaryDto(...)`) • Production Edge-Case & Pitfall Guard (JPQL (Java Persistence Query Language)) • Hands-On Java Implementation & Verification (JPQL (Java Persistence Query Language)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JPQL (Java Persistence Query Language)

Purpose: Projecting read-only search results directly into Java 21 `record` DTOs with zero persistence-context dirty-checking overhead.

What Topic Is Next → Next Topic 12.13: Native SQL Queries through JPA

12.13 Native SQL Queries through JPA [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `createNativeQuery` • Mapping scalar/tuple projections • When to drop down to PostgreSQL-specific SQL (window functions • Recursive CTEs • Production Edge-Case & Pitfall Guard (Native SQL Queries through JPA) • Hands-On Java Implementation & Verification (Native SQL Queries through JPA) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Native SQL Queries through JPA

Purpose: Running PostgreSQL recursive org-chart CTEs or `DENSE_RANK` analytics cleanly within the same JPA transaction.

What Topic Is Next → Next Topic 12.14: Derived Repository Query Fundamentals

12.14 Derived Repository Query Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Method-name query derivation concepts (`findByEmailIgnoreCase` • `existsByCode` • `findByDepartmentIdAndActiveTrue`) • Keeping method names readable vs `@Query` • Production Edge-Case & Pitfall Guard (Derived Repository Query Fundamentals) • Hands-On Java Implementation & Verification (Derived Repository Query Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Derived Repository Query Fundamentals

Purpose: Preparing seamlessly for Month 4 Spring Data `JpaRepository` interfaces.

What Topic Is Next → Next Topic 12.15: JPA Transaction Boundaries

12.15 JPA Transaction Boundaries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-write vs read-only transactions • Why all writes require an active transaction • Rollback on `RuntimeException` • Keeping transactions short • Production Edge-Case & Pitfall Guard (JPA Transaction Boundaries) • Hands-On Java Implementation & Verification (JPA Transaction Boundaries) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JPA Transaction Boundaries

Purpose: Marking read-only query transactions so Hibernate skips dirty-checking snapshots and cuts memory/CPU overhead.

What Topic Is Next → Next Topic 12.16: Persistence Context and First-Level (L1) Cache

12.16 Persistence Context and First-Level (L1) Cache [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Session-scoped identity map (`em.find(Emp • 1)` twice fires only ONE `SELECT`) • Repeatable entity reference identity (`e1 == e2`) • Write-behind flushing • Production Edge-Case & Pitfall Guard (Persistence Context and First-Level (L1) Cache) • Hands-On Java Implementation & Verification (Persistence Context and First-Level (L1) Cache) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Persistence Context

Purpose: Guaranteeing object identity consistency within a single request transaction.

What Topic Is Next → Next Topic 12.17: Second-Level (L2) Cache Fundamentals

12.17 Second-Level (L2) Cache Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: SessionFactory-scoped shared cache vs L1 session cache • Good candidates (rarely-mutated reference data like Countries/Departments) vs bad candidates (volatile balances) • Redis comparison • Production Edge-Case & Pitfall Guard (Second-Level (L2) Cache Fundamentals) • Hands-On Java Implementation & Verification (Second-Level (L2) Cache Fundamentals) • Second-Level (L2) Cache Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Second-Level (L2) Cache Fundamentals

Purpose: Understanding L1 vs L2 cache trade-offs and why distributed microservices often prefer explicit Spring Cache + Redis.

What Topic Is Next → Next Topic 12.18: The N+1 Query Problem (Causes and Resolution)

12.18 The N+1 Query Problem (Causes and Resolution) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading `1` list of `N` Employees then lazily accessing `emp.getDepartment().getName()` in a loop (`1 + N` SQL round-trips!) • Fixing via `JOIN FETCH` • `@EntityGraph` • `@BatchSize` • Production Edge-Case & Pitfall Guard (The N+1 Query Problem (Causes and Resolution)) • Hands-On Java Implementation & Verification (The N+1 Query Problem (Causes and Resolution)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The N+1 Query Problem (Causes and Resolution)

Purpose: Eliminating the #1 cause of slow Spring Boot + Hibernate production endpoints!

What Topic Is Next → Next Topic 12.19: Optimistic Locking (@Version)

12.19 Optimistic Locking (@Version) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Version private Long version • `WHERE id = ? • Version = ?` • `OptimisticLockException` on concurrent lost-update collisions • Production Edge-Case & Pitfall Guard (Optimistic Locking (@Version)) • Hands-On Java Implementation & Verification (Optimistic Locking (@Version)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optimistic Locking (@Version)

Purpose: Preventing two HR admins editing the same employee record simultaneously from silently overwriting each other's changes without holding long DB locks.

What Topic Is Next → Next Topic 12.20: Pagination and Sorting with JPA

12.20 Pagination and Sorting with JPA [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `setFirstResult(offset)` + `setMaxResults(size)` + deterministic `ORDER BY` • Separate `COUNT(e.id)` total query • Two-step ID pagination for collection graphs • Production Edge-Case & Pitfall Guard (Pagination and Sorting with JPA) • Hands-On Java Implementation & Verification (Pagination and Sorting with JPA) • Pagination and Sorting with JPA — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pagination

Purpose: Powering fast, searchable, sortable paginated employee tables in Mini-Project 3.

What Topic Is Next → Module 12 Capstone (Mini-Project 3: Database-Driven Employee Management System (PostgreSQL, JDBC & Hibernate/JPA)) → Next Topic 13.1: Spring Ecosystem and Architecture

Module 12 Hands-On Capstone Project (100 Marks): Mini-Project 3: Database-Driven Employee Management System (PostgreSQL, JDBC & Hibernate/JPA) — Design a normalized PostgreSQL HR database and implement a production-grade persistence layer in Java 21 combining JDBC analytics/batch utilities with Jakarta Persistence (JPA 3.1) & Hiberna...

MODULE 13 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 13: Spring Framework Fundamentals

Understand the core engine of Spring Framework 6 — Inversion of Control (IoC), Dependency Injection (DI), ApplicationContext, bean scopes & lifecycle hooks, Java configuration, component scanning, stereotypes, constructor injection, @Qualifier/@Primary, Profiles, @Value/@ConfigurationProperties, Spring AOP proxies, and Spring MVC DispatcherServlet.

Objectives: ✓ Explain Inversion of Control (IoC) and Dependency Injection (DI) without framework jargon by contrasting tight coupling vs constructor injection. • ✓ Trace how `ApplicationContext` instantiates, wires, initializes, and destroys Spring beans across Singleton and Prototype scopes. • ✓ Configure beans cleanly using `@Configuration`/`@Bean`, `@ComponentScan`, and `@Component`, `@Service`, `@Repository`, `@Controller` stereotypes. • ✓ Resolve multi-implementation ambiguity deterministically with `@Primary` and `@Qualifier` and externalize environment settings via `@Profile` and properties. • ✓ Explain Spring AOP proxy mechanics (`@Aspect`, `@Around`) and the Spring MVC `DispatcherServlet` HTTP request lifecycle.

13.1 Spring Ecosystem and Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Framework 6 baseline (Java 17/21 + Jakarta EE 10) • Modular ecosystem (Boot • Data • Security • Production Edge-Case & Pitfall Guard (Spring Ecosystem and Architecture) • Hands-On Java Implementation & Verification (Spring Ecosystem and Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Ecosystem

Purpose: Building enterprise-grade Java backends without heavyweight application-server lock-in.

What Topic Is Next → Next Topic 13.2: Spring Modules Overview (Core, Context, AOP, Web, Data, Security)

13.2 Spring Modules Overview (Core, Context, AOP, Web, Data, Security) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Responsibilities • Dependency boundaries across Core Container • AOP/Instrumentation • Data Access/Transactions • Production Edge-Case & Pitfall Guard (Spring Modules Overview (Core, Context, AOP, Web, Data, Security)) • Hands-On Java Implementation & Verification (Spring Modules Overview (Core, Context, AOP, Web, Data, Security)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Modules Overview (Core, Context, AOP, Web, Da...

Purpose: Importing only the exact Spring capabilities required by a microservice.

What Topic Is Next → Next Topic 13.3: Inversion of Control (IoC)

13.3 Inversion of Control (IoC) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hollywood Principle ("Don't call us • We'll call you") • Delegating object graph creation • Lifecycle management to the IoC container • Production Edge-Case & Pitfall Guard (Inversion of Control (IoC)) • Hands-On Java Implementation & Verification (Inversion of Control (IoC)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inversion of Control (IoC)

Purpose: Decoupling business services from concrete infrastructure creation so implementations can swap for tests or profiles.

What Topic Is Next → Next Topic 13.4: Dependency Injection (DI)

13.4 Dependency Injection (DI) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Supplying dependencies via constructorcontracts • Testability with mocks • Loose coupling against interfaces • Production Edge-Case & Pitfall Guard (Dependency Injection (DI)) • Hands-On Java Implementation & Verification (Dependency Injection (DI)) • Dependency Injection (DI) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Injection (DI)

Purpose: Injecting `PaymentGateway` and `OrderRepository` interfaces into `CheckoutService` so JUnit tests pass fast in-memory fakes/mocks.

What Topic Is Next → Next Topic 13.5: Spring Beans

13.5 Spring Beans [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: What qualifies as a Spring-managed Bean vs domain entities/DTOs • Bean names • `BeanDefinition` metadata • Production Edge-Case & Pitfall Guard (Spring Beans) • Hands-On Java Implementation & Verification (Spring Beans) • Spring Beans — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Beans

Purpose: Registering stateless services, repositories, controllers, and clients as Spring Beans while keeping request DTOs and JPA entities outside the IoC container!

What Topic Is Next → Next Topic 13.6: BeanFactory Fundamentals

13.6 BeanFactory Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Root `BeanFactory` interface • Lazy bean instantiation semantics vs eager `ApplicationContext` startup validation • Production Edge-Case & Pitfall Guard (BeanFactory Fundamentals) • Hands-On Java Implementation & Verification (BeanFactory Fundamentals) • BeanFactory Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BeanFactory Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding Spring's internal container hierarchy for senior Java interviews.

What Topic Is Next → Next Topic 13.7: ApplicationContext Container

13.7 ApplicationContext Container [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AnnotationConfigApplicationContext` • `ConfigurableApplicationContext` • Eager singleton pre-instantiation • `ApplicationEventPublisher` • Production Edge-Case & Pitfall Guard (ApplicationContext Container) • Hands-On Java Implementation & Verification (ApplicationContext Container) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ApplicationContext Container

Purpose: Failing fast at startup if any dependency, property, or bean wiring is invalid before traffic hits the instance.

What Topic Is Next → Next Topic 13.8: Bean Scopes (Singleton, Prototype, Request, Session)

13.8 Bean Scopes (Singleton, Prototype, Request, Session) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default `singleton` (one instance per Spring container • NOT JVM!) • `prototype` (new instance per lookup) • Web `request` & `session` scopes • Production Edge-Case & Pitfall Guard (Bean Scopes (Singleton, Prototype, Request, Session)) • Hands-On Java Implementation & Verification (Bean Scopes (Singleton, Prototype, Request, Session)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bean Scopes (Singleton, Prototype, Request, Session)

Purpose: Keeping 99% of Spring beans `singleton` and 100% stateless for zero per-request allocation overhead.

What Topic Is Next → Next Topic 13.9: Bean Lifecycle (Initialization and Destruction)

13.9 Bean Lifecycle (Initialization and Destruction) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Constructor -> DI -> BeanPostProcessors (AOP proxies) -> `@PostConstruct` -> Ready -> `@PreDestroy` / `AutoCloseable.close()` on graceful shutdown • Production Edge-Case & Pitfall Guard (Bean Lifecycle (Initialization and Destruction)) • Hands-On Java Implementation & Verification (Bean Lifecycle (Initialization and Destruction)) • Bean Lifecycle (Initialization and Destruction) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bean Lifecycle (Initialization and Destruction) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Warming up reference caches on `@PostConstruct` and flushing thread pools/connections cleanly on `@PreDestroy`.

What Topic Is Next → Next Topic 13.10: Configuration Styles (Legacy XML vs Modern Java @Configuration)

13.10 Configuration Styles (Legacy XML vs Modern Java @Configuration) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading legacy `beans.xml` in brownfield banking apps vs type-safe `@Configuration` + `@Bean` factory methods for third-party SDK classes (`ObjectMapper` • `PasswordEncoder` • `RestClient`) • Production Edge-Case & Pitfall Guard (Configuration Styles (Legacy XML vs Modern Java @Configuration)) • Hands-On Java Implementation & Verification (Configuration Styles (Legacy XML vs Modern Java @Configuration)) • Configuration Styles (Legacy XML vs Modern Java @Configuration) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Configuration Styles (Legacy XML vs Modern Java @Con...

Purpose: Registering third-party library classes (which you cannot edit to add `@Component`) via `@Bean` methods.

What Topic Is Next → Next Topic 13.11: Component Scanning (@ComponentScan)

13.11 Component Scanning (@ComponentScan) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base-package discovery rules • Why placing `@SpringBootApplication` at root package `com.mockattempt.shop` scans all sub-packages automatically • Production Edge-Case & Pitfall Guard (Component Scanning (@ComponentScan)) • Hands-On Java Implementation & Verification (Component Scanning (@ComponentScan)) • Component Scanning (@ComponentScan) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Component Scanning (@ComponentScan) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Organizing packages cleanly so controllers, services, and repositories are discovered automatically.

What Topic Is Next → Next Topic 13.12: Stereotype Annotations (@Component, @Service, @Repository, @Controller)

13.12 Stereotype Annotations (@Component, @Service, @Repository, @Controller) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Semantic specialization of `@Component` • Automatic SQL/JPA persistence exception translation (`DataAccessException`) enabled specifically by `@Repository`! • Production Edge-Case & Pitfall Guard (Stereotype Annotations (@Component, @Service, @Repository, @Controller)) • Hands-On Java Implementation & Verification (Stereotype Annotations (@Component, @Service, @Repository, @Controller)) • Stereotype Annotations (@Component, @Service, @Repository, @Controller) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stereotype Annotations (@Component, @Service, @Repos... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using `@Repository` on custom JDBC/JPA DAOs so vendor exceptions translate uniformly into Spring's `DataAccessException` hierarchy.

What Topic Is Next → Next Topic 13.13: Autowiring (Constructor vs Setter vs Field Injection)

13.13 Autowiring (Constructor vs Setter vs Field Injection) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why single-constructor injection (no `@Autowired` annotation even needed!) is Spring's official standard: `final` immutability • NPE-proof • Trivial JUnit construction • Fast circular-dependency detection • Production Edge-Case & Pitfall Guard (Autowiring (Constructor vs Setter vs Field Injection)) • Hands-On Java Implementation & Verification (Autowiring (Constructor vs Setter vs Field Injection)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Autowiring (Constructor vs Setter vs Field Injection...

Purpose: Writing `@Service` classes with `private final` fields and a clean constructor in every production project.

What Topic Is Next → Next Topic 13.14: @Qualifier (Resolving Multiple Implementations)

13.14 @Qualifier (Resolving Multiple Implementations) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `NoUniqueBeanDefinitionException` • Disambiguating `RazorpayGateway` vs `StripeGateway` explicitly via `@Qualifier("razorpayGateway")` • Production Edge-Case & Pitfall Guard (@Qualifier (Resolving Multiple Implementations)) • Hands-On Java Implementation & Verification (@Qualifier (Resolving Multiple Implementations)) • @Qualifier (Resolving Multiple Implementations) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for @Qualifier (Resolving Multiple Implementations) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Injecting a specific notification channel (`smsSender` vs `emailSender`) or secondary read-replica `DataSource`.

What Topic Is Next → Next Topic 13.15: @Primary (Default Bean Selection)

13.15 @Primary (Default Bean Selection) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Marking the standard default implementation with `@Primary` while allowing specialized callers to override via `@Qualifier` • Production Edge-Case & Pitfall Guard (@Primary (Default Bean Selection)) • Hands-On Java Implementation & Verification (@Primary (Default Bean Selection)) • @Primary (Default Bean Selection) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for @Primary (Default Bean Selection) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Setting PostgreSQL primary `DataSource` or `EmailNotifier` as `@Primary` default across 90% of callers.

What Topic Is Next → Next Topic 13.16: Spring Profiles (@Profile)

13.16 Spring Profiles (@Profile) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Environment-specific bean activation (`dev` • `test` • `prod`) • `!prod` negation • Production Edge-Case & Pitfall Guard (Spring Profiles (@Profile)) • Hands-On Java Implementation & Verification (Spring Profiles (@Profile)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Profiles (@Profile)

Purpose: Activating a sandbox mock payment gateway in `dev`/`test` and the live gateway only under `@Profile("prod")`.

What Topic Is Next → Next Topic 13.17: Externalized Properties (@Value and @PropertySource)

13.17 Externalized Properties (@Value and @PropertySource) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Value("${app.tax.gst-rate:18.0}")` safe defaults • `Environment` • Strongly-typed Java 21 `record` `@ConfigurationProperties` • Production Edge-Case & Pitfall Guard (Externalized Properties (@Value and @PropertySource)) • Hands-On Java Implementation & Verification (Externalized Properties (@Value and @PropertySource)) • Externalized Properties (@Value and @PropertySource) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Externalized Properties (@Value and @PropertySource)

Purpose: Binding grouped JWT, Kafka, and S3 settings cleanly into validated immutable Java 21 `record` configuration classes.

What Topic Is Next → Next Topic 13.18: Spring AOP (Aspect, Advice, Pointcut and Join Point)

13.18 Spring AOP (Aspect, Advice, Pointcut and Join Point) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cross-cutting concerns (latency logging • Auditing • Security • Transactions) • Production Edge-Case & Pitfall Guard (Spring AOP (Aspect, Advice, Pointcut and Join Point)) • Hands-On Java Implementation & Verification (Spring AOP (Aspect, Advice, Pointcut and Join Point)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring AOP (Aspect, Advice, Pointcut and Join Point)

Purpose: Logging execution latency of all `@Service` methods automatically and understanding why `@Transactional` uses Spring AOP proxies!

What Topic Is Next → Next Topic 13.19: Spring MVC and DispatcherServlet Lifecycle

13.19 Spring MVC and DispatcherServlet Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Front Controller pattern: `DispatcherServlet` -> `HandlerMapping` -> `HandlerInterceptor` -> `@Controller` -> `HttpMessageConverter` (Jackson) / `ExceptionResolver` • Production Edge-Case & Pitfall Guard (Spring MVC and DispatcherServlet Lifecycle) • Hands-On Java Implementation & Verification (Spring MVC and DispatcherServlet Lifecycle) • Spring MVC and DispatcherServlet Lifecycle — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring MVC • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Tracing end-to-end how an incoming HTTP JSON request becomes a Java DTO, executes business logic, and serializes an HTTP response.

What Topic Is Next → Module 13 Capstone (Week 13 Lab Build: Modular Spring Core Notification & Billing Container) → Next Topic 14.1: Spring Boot 3 Introduction and Architecture

Module 13 Hands-On Capstone Project (100 Marks): Week 13 Lab Build: Modular Spring Core Notification & Billing Container — Build a constructor-injected Spring Framework 6 module switching cleanly between Sandbox and Production gateways via `@Profile`, `@Primary`, `@Qualifier`, `@ConfigurationProperties`, and an...

MODULE 14 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 14: Spring Boot Fundamentals and REST API Development

Build clean, idiomatic Spring Boot 3.x REST APIs — Spring Initializr, Starter BOM dependency management, @Conditional auto-configuration internals, embedded Tomcat, YAML profiles, REST architectural constraints, HTTP methods/status codes, @RestController mappings, Java 21 record DTOs, Jackson JSON serialization, API versioning, Postman test suites, and OpenAPI 3 / Swagger UI.

Objectives: ✓ Bootstrap Spring Boot 3.x + Java 21 projects via Spring Initializr and explain how Starter POMs and `@EnableAutoConfiguration` work internally. • ✓ Configure embedded Tomcat, YAML multi-environment profiles (`dev`, `test`, `prod`), and Spring Boot Actuator health checks. • ✓ Design clean resource-oriented REST endpoints using proper HTTP verbs (`GET`, `POST`, `PUT`, `PATCH`, `DELETE`) and accurate HTTP status codes (`200`, `201`, `204`, `400`, `401`, `403`, `404`, `409`, `422`). • ✓ Bind path variables, query parameters, headers, and `@RequestBody` JSON payloads into immutable Java 21 `record` DTOs via Jackson. • ✓ Test APIs systematically with Postman collections and publish interactive OpenAPI 3 / Swagger UI documentation (`springdoc-openapi`).

14.1 Spring Boot 3 Introduction and Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Opinionated convention-over-configuration • Standalone executable JARs • Production-ready Actuator metrics/health • Zero XML • Production Edge-Case & Pitfall Guard (Spring Boot 3 Introduction and Architecture) • Hands-On Java Implementation & Verification (Spring Boot 3 Introduction and Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Boot 3 Introduction

Purpose: Shipping standardized, cloud-ready Java 21 backend services in minutes instead of days of servlet XML plumbing.

What Topic Is Next → Next Topic 14.2: Project Creation with Spring Initializr

14.2 Project Creation with Spring Initializr [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Maven wrapper (`./mvnw`) • `spring-boot-starter-parent` BOM • Java 21 target • Choosing starters (`web` • Production Edge-Case & Pitfall Guard (Project Creation with Spring Initializr) • Hands-On Java Implementation & Verification (Project Creation with Spring Initializr) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Project Creation with Spring Initializr

Purpose: Guaranteeing reproducible builds on every developer laptop and CI runner via committed `./mvnw` / `mvnw.cmd`.

What Topic Is Next → Next Topic 14.3: Spring Boot Starters and BOM Dependency Management

14.3 Spring Boot Starters and BOM Dependency Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Curated transitive dependency bundles (`spring-boot-starter-web` • `data-jpa` • `security` • `test`) • Production Edge-Case & Pitfall Guard (Spring Boot Starters and BOM Dependency Management) • Hands-On Java Implementation & Verification (Spring Boot Starters and BOM Dependency Management) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Boot Starters

Purpose: Eliminating `NoSuchMethodError` dependency-hell clashes across Jackson, Hibernate, and Spring modules.

What Topic Is Next → Next Topic 14.4: Auto-Configuration Internals

14.4 Auto-Configuration Internals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AutoConfiguration.imports` • `@ConditionalOnClass` • `@ConditionalOnMissingBean` • `@ConditionalOnProperty` • Production Edge-Case & Pitfall Guard (Auto-Configuration Internals) • Hands-On Java Implementation & Verification (Auto-Configuration Internals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Auto-Configuration Internals

Purpose: Customizing Jackson `ObjectMapper` or `SecurityFilterChain` cleanly because Boot backs off automatically via `@ConditionalOnMissingBean`!

What Topic Is Next → Next Topic 14.5: Embedded Tomcat Server Configuration

14.5 Embedded Tomcat Server Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `server.port` • Graceful shutdown (`server.shutdown=graceful`) • Tomcat thread pool (`server.tomcat.threads.max`) • Java 21 virtual threads toggle (`spring.threads.virtual.enabled`) • Production Edge-Case & Pitfall Guard (Embedded Tomcat Server Configuration) • Hands-On Java Implementation & Verification (Embedded Tomcat Server Configuration) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Embedded Tomcat Server Configuration

Purpose: Enabling `server.shutdown=graceful` so rolling Docker deployments finish in-flight HTTP requests without dropping user checkouts!

What Topic Is Next → Next Topic 14.6: Layered Application Package Structure

14.6 Layered Application Package Structure [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clean separation across `controller` • `service` • `repository` • `domain`/`entity` • Production Edge-Case & Pitfall Guard (Layered Application Package Structure) • Hands-On Java Implementation & Verification (Layered Application Package Structure) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Layered Application Package Structure

Purpose: Enforcing strict inward dependency rules (`controller -> service -> repository`) with zero reverse imports.

What Topic Is Next → Next Topic 14.7: Configuration with application.properties and YAML

14.7 Configuration with application.properties and YAML [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hierarchical `application.yml` • `${ENV_VAR:default}` placeholders • External config precedence (CLI args > Env Vars > Profile YAML > Base YAML) • Production Edge-Case & Pitfall Guard (Configuration with application.properties and YAML) • Hands-On Java Implementation & Verification (Configuration with application.properties and YAML) • Configuration with application.properties and YAML — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Configuration with application.properties

Purpose: Injecting production DB URLs and JWT secrets via OS environment variables without rebuilding the Docker image.

What Topic Is Next → Next Topic 14.8: Environment Profiles (dev, test, prod)

14.8 Environment Profiles (dev, test, prod) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `application-dev.yml` • `application-test.yml` • `application-prod.yml` • Hiding stack traces • Production Edge-Case & Pitfall Guard (Environment Profiles (dev, test, prod)) • Hands-On Java Implementation & Verification (Environment Profiles (dev, test, prod)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Profiles (dev, test, prod)

Purpose: Enabling verbose SQL logging in `dev`, Testcontainers in `test`, and hardened JSON logging in `prod`.

What Topic Is Next → Next Topic 14.9: REST Architectural Principles and HTTP Semantics

14.9 REST Architectural Principles and HTTP Semantics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Statelessness • Plural noun resource URIs (`/api/v1/products/{id}`) • Representations • Idempotency • Production Edge-Case & Pitfall Guard (REST Architectural Principles and HTTP Semantics) • Hands-On Java Implementation & Verification (REST Architectural Principles and HTTP Semantics) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST Architectural Principles

Purpose: Designing predictable enterprise APIs that mobile, web, and partner services integrate with effortlessly.

What Topic Is Next → Next Topic 14.10: HTTP Methods (GET, POST, PUT, PATCH, DELETE)

14.10 HTTP Methods (GET, POST, PUT, PATCH, DELETE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Safe & idempotent `GET` • Non-idempotent `POST` (create/action) • Idempotent full replacement `PUT` • Partial update `PATCH` • Production Edge-Case & Pitfall Guard (HTTP Methods (GET, POST, PUT, PATCH, DELETE)) • Hands-On Java Implementation & Verification (HTTP Methods (GET, POST, PUT, PATCH, DELETE)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Methods (GET, POST, PUT, PATCH, DELETE)

Purpose: Understanding why network retries are safe for `GET`/`PUT`/`DELETE` while `POST /orders` needs an `Idempotency-Key`!

What Topic Is Next → Next Topic 14.11: HTTP Status Codes (2xx, 3xx, 4xx, 5xx)

14.11 HTTP Status Codes (2xx, 3xx, 4xx, 5xx) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `200 OK` • `201 Created` (+ `Location` header!) • `204 No Content` • `400 Bad Request` • Production Edge-Case & Pitfall Guard (HTTP Status Codes (2xx, 3xx, 4xx, 5xx)) • Hands-On Java Implementation & Verification (HTTP Status Codes (2xx, 3xx, 4xx, 5xx)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Status Codes (2xx, 3xx, 4xx, 5xx)

Purpose: Enabling frontend clients and API gateways to distinguish auth expiry (`401`) from permission denial (`403`), duplicate SKU (`409`), and server faults (`500`).

What Topic Is Next → Next Topic 14.12: @RestController vs @Controller

14.12 @RestController vs @Controller [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@RestController` (`@Controller` + `@ResponseBody`) • Automatic JSON message conversion vs view-template rendering • Production Edge-Case & Pitfall Guard (@RestController vs @Controller) • Hands-On Java Implementation & Verification (@RestController vs @Controller) • @RestController vs @Controller — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for @RestController vs @Controller • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Writing thin, clean HTTP adapter controllers that delegate all business rules to `@Service` beans.

What Topic Is Next → Next Topic 14.13: @RequestMapping and Composed Method Mappings

14.13 @RequestMapping and Composed Method Mappings [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class-level `@RequestMapping("/api/v1/products")` + method-level `@GetMapping` • `@PostMapping` • `@PutMapping` • `@PatchMapping` • Production Edge-Case & Pitfall Guard (@RequestMapping and Composed Method Mappings) • Hands-On Java Implementation & Verification (@RequestMapping and Composed Method Mappings) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for @RequestMapping

Purpose: Keeping endpoint routes consistent, self-documenting, and free of ambiguous path collisions.

What Topic Is Next → Next Topic 14.14: Extracting @PathVariable, @RequestParam and @RequestHeader

14.14 Extracting @PathVariable, @RequestParam and @RequestHeader [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Resource identifiers in `@PathVariable` • Optional filtering/pagination in `@RequestParam(defaultValue = ...)` • Metadata like `Idempotency-Key` in `@RequestHeader` • Production Edge-Case & Pitfall Guard (Extracting @PathVariable, @RequestParam and @RequestHeader) • Hands-On Java Implementation & Verification (Extracting @PathVariable, @RequestParam and @RequestHeader) • Extracting @PathVariable, @RequestParam and @RequestHeader — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Extracting @PathVariable

Purpose: Supporting clean query filters (`GET /api/v1/products?category=books&minPrice=199&page=0&size=20`).

What Topic Is Next → Next Topic 14.15: @RequestBody Deserialization

14.15 @RequestBody Deserialization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Binding JSON request payloads into immutable Java 21 `record` request DTOs • Handling malformed JSON (`HttpMessageNotReadableException` -> HTTP 400) • Production Edge-Case & Pitfall Guard (@RequestBody Deserialization) • Hands-On Java Implementation & Verification (@RequestBody Deserialization) • @RequestBody Deserialization — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for @RequestBody Deserialization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Accepting strongly-typed JSON payloads (`CreateProductRequest`) with zero setter mutation.

What Topic Is Next → Next Topic 14.16: ResponseEntity (Status Codes, Headers and Envelopes)

14.16 ResponseEntity (Status Codes, Headers and Envelopes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ResponseEntity.ok()` • `.created(uri)` • `.noContent().build()` • `.status(HttpStatus.ACCEPTED)` • Production Edge-Case & Pitfall Guard (ResponseEntity (Status Codes, Headers and Envelopes)) • Hands-On Java Implementation & Verification (ResponseEntity (Status Codes, Headers and Envelopes)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ResponseEntity (Status Codes, Headers and Envelopes)

Purpose: Controlling exact HTTP status codes and response headers per endpoint.

What Topic Is Next → Next Topic 14.17: DTO Pattern (Separating API Contracts from JPA Entities)

14.17 DTO Pattern (Separating API Contracts from JPA Entities) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why exposing `@Entity` directly causes mass-assignment security flaws (`role: "ADMIN"`) • Infinite bidirectional JSON recursion • `LazyInitializationException` • Breaking API coupling • Production Edge-Case & Pitfall Guard (DTO Pattern (Separating API Contracts from JPA Entities)) • Hands-On Java Implementation & Verification (DTO Pattern (Separating API Contracts from JPA Entities)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DTO Pattern (Separating API Contracts from JPA Entit...

Purpose: Defining dedicated `CreateUserRequest`, `UpdateProfileRequest`, and `UserSummaryResponse` Java 21 `record` DTOs.

What Topic Is Next → Next Topic 14.18: JSON Serialization with Jackson

14.18 JSON Serialization with Jackson [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ObjectMapper` • `JavaTimeModule` ISO-8601 dates (`WRITE_DATES_AS_TIMESTAMPS: false`) • `@JsonProperty` • `@JsonInclude(NON_NULL)` • Production Edge-Case & Pitfall Guard (JSON Serialization with Jackson) • Hands-On Java Implementation & Verification (JSON Serialization with Jackson) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Serialization with Jackson

Purpose: Ensuring `Instant` and `LocalDate` serialize as standard ISO-8601 strings (`"2026-11-17T14:30:00Z"`) across all endpoints.

What Topic Is Next → Next Topic 14.19: REST API Design Best Practices and Versioning

14.19 REST API Design Best Practices and Versioning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: URI versioning (`/api/v1/...`) • Backward-compatible additive changes • Filtering/sorting/pagination query conventions • Idempotency keys • Production Edge-Case & Pitfall Guard (REST API Design Best Practices and Versioning) • Hands-On Java Implementation & Verification (REST API Design Best Practices and Versioning) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST API Design Best Practices

Purpose: Evolving `/api/v1/orders` to `/api/v2/orders` cleanly without breaking deployed mobile clients.

What Topic Is Next → Next Topic 14.20: API Testing with Postman

14.20 API Testing with Postman [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Postman Environments (`{{baseUrl}}` • `{{jwtToken}}`) • Collection folders • Automated post-response status/JSON schema assertions • Production Edge-Case & Pitfall Guard (API Testing with Postman) • Hands-On Java Implementation & Verification (API Testing with Postman) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Testing with Postman

Purpose: Sharing an executable Postman collection in every repository so reviewers and QA verify all endpoints in one click.

What Topic Is Next → Next Topic 14.21: OpenAPI 3 and Swagger Documentation (springdoc-openapi)

14.21 OpenAPI 3 and Swagger Documentation (springdoc-openapi) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `springdoc-openapi-starter-webmvc-ui` • `@Tag` • `@Operation` • `@ApiResponses` • Production Edge-Case & Pitfall Guard (OpenAPI 3 and Swagger Documentation (springdoc-openapi)) • Hands-On Java Implementation & Verification (OpenAPI 3 and Swagger Documentation (springdoc-openapi)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OpenAPI 3

Purpose: Generating live, always-up-to-date OpenAPI 3 contracts (`/v3/api-docs`) for frontend teams and capstone evaluation.

What Topic Is Next → Module 14 Capstone (Week 14 Lab Build: Documented Spring Boot 3 Product Catalog REST API) → Next Topic 15.1: Layered Backend Architecture (Controller, Service, Repository)

Module 14 Hands-On Capstone Project (100 Marks): Week 14 Lab Build: Documented Spring Boot 3 Product Catalog REST API — Develop a resource-oriented `/api/v1/products` and `/api/v1/categories` Spring Boot 3 REST API using Java 21 `record` DTOs, `ResponseEntity` status semantics, OpenAPI 3 / Swagger UI, and an...

MODULE 15 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 15: Spring Boot, JPA, Validation and Exception Handling

Connect Spring Boot 3 REST APIs to PostgreSQL using Spring Data JPA, Jakarta Bean Validation, custom validators, centralized @RestControllerAdvice RFC-7807 ProblemDetail error responses, Pageable/Sort/Specification dynamic search, @Transactional propagation & isolation, SLF4J structured logging, and secret management.

Objectives: ✓ Architect cohesive Controller -> Service -> JpaRepository layers with clean DTO <-> Entity mapping. • ✓ Enforce input validation at API boundaries using `@Valid`, standard Jakarta constraints, and custom cross-field validators. • ✓ Centralize all error responses consistently via `@RestControllerAdvice` and `@ExceptionHandler` (`ProblemDetail` / standard error envelope). • ✓ Implement server-side pagination (`Pageable`), dynamic whitelisted sorting (`Sort`), and composable multi-filter queries (`JpaSpecificationExecutor`). • ✓ Configure `@Transactional` boundaries (`readOnly = true`, propagation, isolation, rollback rules), structured SLF4J logging, and externalized secrets.

15.1 Layered Backend Architecture (Controller, Service, Repository) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Thin HTTP Controller • Transactional domain Service • Persistence Repository • Strict unidirectional layer contracts • Production Edge-Case & Pitfall Guard (Layered Backend Architecture (Controller, Service, Repository)) • Hands-On Java Implementation & Verification (Layered Backend Architecture (Controller, Service, Repository)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Layered Backend Architecture (Controller, Service, R...

Purpose: Keeping business invariants reusable across REST controllers, Kafka listeners, and scheduled jobs.

What Topic Is Next → Next Topic 15.2: Spring Data JpaRepository and CRUD Layer

15.2 Spring Data JpaRepository and CRUD Layer [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JpaRepository<Product • Long>` • `save` • `findById` • Production Edge-Case & Pitfall Guard (Spring Data JpaRepository and CRUD Layer) • Hands-On Java Implementation & Verification (Spring Data JpaRepository and CRUD Layer) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Data JpaRepository

Purpose: Disabling `spring.jpa.open-in-view=false` so lazy queries never fire accidentally in the web view/JSON layer while holding HikariCP connections!

What Topic Is Next → Next Topic 15.3: Service Layer Business Logic Design

15.3 Service Layer Business Logic Design [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Orchestrating domain checks (stock availability • Duplicate SKU guard • Price computation) • Keeping entities rich rather than anemic • Production Edge-Case & Pitfall Guard (Service Layer Business Logic Design) • Hands-On Java Implementation & Verification (Service Layer Business Logic Design) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Layer Business Logic Design

Purpose: Validating category existence, SKU uniqueness, and stock rules atomically inside `ProductService`.

What Topic Is Next → Next Topic 15.4: DTO-to-Entity and Entity-to-DTO Mapping

15.4 DTO-to-Entity and Entity-to-DTO Mapping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deterministic dedicated Mapper components / static factory methods (`ProductResponse.from(entity)`) • MapStruct overview • Avoiding reflection-heavy `BeanUtils` • Production Edge-Case & Pitfall Guard (DTO-to-Entity and Entity-to-DTO Mapping) • Hands-On Java Implementation & Verification (DTO-to-Entity and Entity-to-DTO Mapping) • DTO-to-Entity and Entity-to-DTO Mapping — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DTO-to-Entity

Purpose: Compile-time-checked mapping where adding a new DTO field immediately surfaces in the compiler.

What Topic Is Next → Next Topic 15.5: Jakarta Bean Validation Fundamentals

15.5 Jakarta Bean Validation Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-boot-starter-validation` (Hibernate Validator) • Triggering validation via `@Valid` on `@RequestBody` • `@Validated` on path/query params • Production Edge-Case & Pitfall Guard (Jakarta Bean Validation Fundamentals) • Hands-On Java Implementation & Verification (Jakarta Bean Validation Fundamentals) • Jakarta Bean Validation Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Jakarta Bean Validation Fundamentals

Purpose: Rejecting malformed client payloads at the HTTP edge before a single database query runs.

What Topic Is Next → Next Topic 15.6: Standard Validation Annotations (@NotNull, @NotBlank, @Size, @Email, @Min, @Max, @Positive)

15.6 Standard Validation Annotations (@NotNull, @NotBlank, @Size, @Email, @Min, @Max, @Positive) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Difference between `@NotNull` • `@NotEmpty` • `@NotBlank` (trims whitespace!) • `@DecimalMin` • Production Edge-Case & Pitfall Guard (Standard Validation Annotations (@NotNull, @NotBlank, @Size, @Email, @Min, @Max, @Positive)) • Hands-On Java Implementation & Verification (Standard Validation Annotations (@NotNull, @NotBlank, @Size, @Email, @Min, @Max, @Positive)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Standard Validation Annotations (@NotNull, @NotBlank...

Purpose: Validating strings (`@NotBlank @Size(max=120)`), emails (`@Email`), prices (`@NotNull @Positive`), and nested cart items (`@NotEmpty List<@Valid CartItemReq>`).

What Topic Is Next → Next Topic 15.7: Custom Validation Annotations and ConstraintValidator

15.7 Custom Validation Annotations and ConstraintValidator [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Defining `@ValidSku` or class-level `@DateRangeValid` + `ConstraintValidator<A • T>` for reusable domain/cross-field rules • Production Edge-Case & Pitfall Guard (Custom Validation Annotations and ConstraintValidator) • Hands-On Java Implementation & Verification (Custom Validation Annotations and ConstraintValidator) • Custom Validation Annotations and ConstraintValidator — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Validation Annotations • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Validating cross-field rules like `discountPrice <= mrpPrice` or `endDate >= startDate` cleanly.

What Topic Is Next → Next Topic 15.8: Controller-Scoped @ExceptionHandler

15.8 Controller-Scoped @ExceptionHandler [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Handling specific exceptions locally inside a controller vs why cross-controller duplication leads to `@RestControllerAdvice` • Production Edge-Case & Pitfall Guard (Controller-Scoped @ExceptionHandler) • Hands-On Java Implementation & Verification (Controller-Scoped @ExceptionHandler) • Controller-Scoped @ExceptionHandler — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Controller-Scoped @ExceptionHandler • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Handling a specialized payment webhook signature exception within `WebhookController`.

What Topic Is Next → Next Topic 15.9: Global Exception Handling (@ControllerAdvice and @RestControllerAdvice)

15.9 Global Exception Handling (@ControllerAdvice and @RestControllerAdvice) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Centralized exception translation for `ResourceNotFoundException` (404) • `DuplicateResourceException` (409) • `MethodArgumentNotValidException` (400 field map) • `OptimisticLockingFailureException` (409) • Production Edge-Case & Pitfall Guard (Global Exception Handling (@ControllerAdvice and @RestControllerAdvice)) • Hands-On Java Implementation & Verification (Global Exception Handling (@ControllerAdvice and @RestControllerAdvice)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Global Exception Handling (@ControllerAdvice and @Re...

Purpose: Guaranteeing every single error across the entire backend returns one predictable JSON schema.

What Topic Is Next → Next Topic 15.10: Standard Error Response Design (RFC 7807 ProblemDetail)

15.10 Standard Error Response Design (RFC 7807 ProblemDetail) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring 6 native `ProblemDetail` (`type` • `title` • `status` • `detail` • Production Edge-Case & Pitfall Guard (Standard Error Response Design (RFC 7807 ProblemDetail)) • Hands-On Java Implementation & Verification (Standard Error Response Design (RFC 7807 ProblemDetail)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Standard Error Response Design (RFC 7807 ProblemDeta...

Purpose: Returning structured field-by-field validation errors (`{"name": "must not be blank", "price": "must be greater than 0"}`) for instant UI form highlighting.

What Topic Is Next → Next Topic 15.11: Spring Data Pagination (Page, Slice, PageRequest and Pageable)

15.11 Spring Data Pagination (Page, Slice, PageRequest and Pageable) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `PageRequest.of(page • Math.min(size • 100) • Sort)` • Production Edge-Case & Pitfall Guard (Spring Data Pagination (Page, Slice, PageRequest and Pageable)) • Hands-On Java Implementation & Verification (Spring Data Pagination (Page, Slice, PageRequest and Pageable)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Data Pagination (Page, Slice, PageRequest and...

Purpose: Capping `size <= 100` so a malicious client requesting `?size=10000000` cannot trigger a JVM `OutOfMemoryError`!

What Topic Is Next → Next Topic 15.12: Dynamic Sorting (Sort and Whitelisted Sort Fields)

15.12 Dynamic Sorting (Sort and Whitelisted Sort Fields) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Sort.by(Sort.Order.desc("price") • Sort.Order.asc("id"))` • Validating client `sortBy` strings against allowed entity fields to prevent `PropertyReferenceException` (500) • Production Edge-Case & Pitfall Guard (Dynamic Sorting (Sort and Whitelisted Sort Fields)) • Hands-On Java Implementation & Verification (Dynamic Sorting (Sort and Whitelisted Sort Fields)) • Dynamic Sorting (Sort and Whitelisted Sort Fields) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Sorting (Sort and Whitelisted Sort Fields)

Purpose: Supporting `sort=price,asc` or `sort=createdAt,desc` cleanly on catalog and order endpoints.

What Topic Is Next → Next Topic 15.13: Dynamic Query Filtering with JPA Specifications

15.13 Dynamic Query Filtering with JPA Specifications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JpaSpecificationExecutor<Product>` • Composable `Specification<Product>` predicates (`hasCategory` • `priceBetween` • `nameContains` • Production Edge-Case & Pitfall Guard (Dynamic Query Filtering with JPA Specifications) • Hands-On Java Implementation & Verification (Dynamic Query Filtering with

JPA Specifications) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Query Filtering with JPA Specifications

Purpose: Supporting any combination of 6 optional search filters (`category`, `brand`, `minPrice`, `maxPrice`, `inStock`, `keyword`) without 64 repository methods!

What Topic Is Next → Next Topic 15.14: Declarative Transactions (@Transactional)

15.14 Declarative Transactions (@Transactional) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Transactional` vs `@Transactional(readOnly = true)` • Default rollback on unchecked `RuntimeException`/`Error` ONLY (`rollbackFor = Exception.class` for checked!) • Proxy self-invocation rule • Production Edge-Case & Pitfall Guard (Declarative Transactions (@Transactional)) • Hands-On Java Implementation & Verification (Declarative Transactions (@Transactional)) • Declarative Transactions (@Transactional) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Declarative Transactions (@Transactional)

Purpose: Putting `@Transactional(readOnly = true)` at class level on read-heavy `@Service`s and `@Transactional` on mutating methods!

What Topic Is Next → Next Topic 15.15: Transaction Propagation (REQUIRED, REQUIRES_NEW, MANDATORY)

15.15 Transaction Propagation (REQUIRED, REQUIRES_NEW, MANDATORY) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Propagation.REQUIRED` (join existing or start new) • `REQUIRES_NEW` (suspend outer tx • Commit independent tx — e.g. • Failed-login security audit!) • Production Edge-Case & Pitfall Guard (Transaction Propagation (REQUIRED, REQUIRES_NEW, MANDATORY)) • Hands-On Java Implementation & Verification (Transaction Propagation (REQUIRED, REQUIRES_NEW, MANDATORY)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transaction Propagation (REQUIRED, REQUIRES_NEW, MAN...

Purpose: Persisting a payment-failure audit log via `REQUIRES_NEW` on a separate collaborator bean even when the outer checkout transaction rolls back!

What Topic Is Next → Next Topic 15.16: Transaction Isolation Configuration in Spring

15.16 Transaction Isolation Configuration in Spring [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Transactional(isolation = Isolation.READ_COMMITTED / REPEATABLE_READ / SERIALIZABLE)` • Combining with `@Lock(LockModeType.PESSIMISTIC_WRITE)` or `@Version` • Production Edge-Case & Pitfall Guard (Transaction Isolation Configuration in Spring) • Hands-On Java Implementation & Verification (Transaction Isolation Configuration in Spring) • Transaction Isolation Configuration in Spring — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transaction Isolation Configuration in Spring • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing concurrent stock overselling when 50 users buy the last 2 flash-sale items simultaneously.

What Topic Is Next → Next Topic 15.17: Structured Application Logging (SLF4J and Logback)

15.17 Structured Application Logging (SLF4J and Logback) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `LoggerFactory.getLogger` • Parameterized placeholders `log.info("Order {} placed by user {}" • OrderId • UserId)` (never string `+`!) • Production Edge-Case & Pitfall Guard (Structured Application Logging (SLF4J and Logback)) • Hands-On Java Implementation & Verification (Structured Application Logging (SLF4J and Logback)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Structured Application Logging (SLF4J and Logback)

Purpose: Attaching an MDC `X-Correlation-Id` filter so every log line of an HTTP request can be grepped instantly in production.

What Topic Is Next → Next Topic 15.18: Configuration Security and External Secrets

15.18 Configuration Security and External Secrets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 12-Factor App config via OS env vars (`DB_PASSWORD` • `JWT_SECRET`) • `.env.example` template committed while `.env` is git-ignored • Fail-fast `@NotBlank` startup validation on `@ConfigurationProperties` • Production Edge-Case & Pitfall Guard (Configuration Security and External Secrets) • Hands-On Java Implementation & Verification (Configuration Security and External Secrets) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Configuration Security

Purpose: Ensuring the app refuses to boot in `prod` if `JWT_SECRET` is missing or shorter than 32 bytes instead of falling back to `"secret123"`.

What Topic Is Next → Module 15 Capstone (Week 15 Lab Build: Transactional JPA Search API with ProblemDetail Errors) → Next Topic 16.1: Security Fundamentals (Authentication vs Authorization)

Module 15 Hands-On Capstone Project (100 Marks): Week 15 Lab Build: Transactional JPA Search API with ProblemDetail Errors — Connect your Catalog & Order REST API to PostgreSQL with `JpaRepository`, composable `Specification` filters, `Pageable` pagination, Jakarta Bean Validation, `@RestControllerAdvice` RFC-7807...

MODULE 16 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 17 TOPICS (136 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 16: Spring Security and Authentication

Secure Spring Boot 3 REST APIs end-to-end using Spring Security 6 — Authentication vs Authorization, SecurityFilterChain lambda DSL, BCrypt password hashing, UserDetails & UserDetailsService, AuthenticationManager, RBAC, JWT header.payload.signature internals, stateless OncePerRequestFilter validation, CORS, CSRF trade-offs, OAuth2 Resource Server concepts, @PreAuthorize method security, security testing, and Major Project 4.

Objectives: ✓ Trace how requests pass through the Spring Security 6 `SecurityFilterChain` before reaching `DispatcherServlet`. • ✓ Hash passwords irreversibly with `BCryptPasswordEncoder` and load database identities via `UserDetailsService`. • ✓ Implement stateless HMAC-SHA256 / RSA JWT issuance and `OncePerRequestFilter` validation populating `SecurityContextHolder`. • ✓ Enforce Role-Based and Permission-Based Access Control (`ADMIN`, `SELLER`, `CUSTOMER`) at both URL and `@PreAuthorize` method levels. • ✓ Configure strict CORS origins, stateless CSRF policy, custom `401`/`403` JSON entry points, `@WithMockUser` security tests, and deliver Major Project 4 (Secure E-Commerce Backend API).

16.1 Security Fundamentals (Authentication vs Authorization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Principal identity verification (AuthN -> 401 when missing/invalid) vs privilege verification (AuthZ -> 403 when insufficient) • Defense-in-Depth • OWASP API Top 10 (BOLA/IDOR!) • Production Edge-Case & Pitfall Guard (Security Fundamentals (Authentication vs Authorization)) • Hands-On Java Implementation & Verification (Security Fundamentals (Authentication vs Authorization)) • Security Fundamentals (Authentication vs Authorization) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Security Fundamentals (Authentication vs Authorizati...

Purpose: Preventing Broken Object Level Authorization (BOLA) by checking `order.getUserId().equals(currentUser.id())` in addition to role checks.

What Topic Is Next → Next Topic 16.2: Spring Security 6 Filter Chain Architecture

16.2 Spring Security 6 Filter Chain Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DelegatingFilterProxy` -> `FilterChainProxy` -> ordered `SecurityFilterChain` filters -> `SecurityContextHolder` -> `DispatcherServlet` • Production Edge-Case & Pitfall Guard (Spring Security 6 Filter Chain Architecture) • Hands-On Java Implementation & Verification (Spring Security 6 Filter Chain Architecture) • Spring Security 6 Filter Chain Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Security 6 Filter Chain Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding why `@RestControllerAdvice` does NOT catch exceptions thrown inside a Security Filter unless delegated via `AuthenticationEntryPoint` / `AccessDeniedHandler...

What Topic Is Next → Next Topic 16.3: SecurityFilterChain Configuration (Lambda DSL)

16.3 SecurityFilterChain Configuration (Lambda DSL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern Spring Security 6 `@Bean SecurityFilterChain` lambda DSL (`authorizeHttpRequests` • `requestMatchers` • `sessionManagement` • `exceptionHandling`) • Production Edge-Case & Pitfall Guard (SecurityFilterChain Configuration (Lambda DSL)) • Hands-On Java Implementation & Verification (SecurityFilterChain Configuration (Lambda DSL)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SecurityFilterChain Configuration (Lambda DSL)

Purpose: Allowing public `GET /api/v1/products/**` and `POST /api/v1/auth/**` while requiring authentication on everything else (`anyRequest().authenticated()`).

What Topic Is Next → Next Topic 16.4: Password Security (PasswordEncoder and BCrypt)

16.4 Password Security (PasswordEncoder and BCrypt) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why MD5/SHA-256 are broken for passwords • Adaptive salted work-factor hashing with `BCryptPasswordEncoder(12)` • `encode()` on registration vs `matches()` on login • Production Edge-Case & Pitfall Guard (Password Security (PasswordEncoder and BCrypt)) • Hands-On Java Implementation & Verification (Password Security (PasswordEncoder and BCrypt)) • Password Security (PasswordEncoder and BCrypt) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Password Security (PasswordEncoder and BCrypt)

Purpose: Guaranteeing every stored password hash has a unique per-user cryptographic salt automatically embedded in the `$2a$12$...` string.

What Topic Is Next → Next Topic 16.5: UserDetails Identity Model

16.5 UserDetails Identity Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing/adapting `UserDetails` (`getUsername` • `getPassword` • `getAuthorities` • `isEnabled` • Production Edge-Case & Pitfall Guard (UserDetails Identity Model) • Hands-On Java Implementation & Verification (UserDetails Identity Model) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for UserDetails Identity Model

Purpose: Creating a clean `AuthenticatedUser` principal record wrapping `userId`, `email`, and `GrantedAuthority` roles without coupling JPA `@Entity` directly to Spring Security.

What Topic Is Next → Next Topic 16.6: UserDetailsService Implementation

16.6 UserDetailsService Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading user credentials & roles from PostgreSQL via `UserRepository.findByEmailIgnoreCase` • Throwing `UsernameNotFoundException` without user-enumeration leaks • Production Edge-Case & Pitfall Guard (UserDetailsService Implementation) • Hands-On Java Implementation & Verification (UserDetailsService Implementation) • UserDetailsService Implementation — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for UserDetailsService Implementation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Bridging PostgreSQL `users` + `roles` tables into Spring Security's authentication provider.

What Topic Is Next → Next Topic 16.7: AuthenticationManager and DaoAuthenticationProvider Workflow

16.7 AuthenticationManager and DaoAuthenticationProvider Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AuthenticationConfiguration.getAuthenticationManager()` • `UsernamePasswordAuthenticationToken` • Credential verification • Disabled/locked account checks • Production Edge-Case & Pitfall Guard (AuthenticationManager and DaoAuthenticationProvider Workflow) • Hands-On Java Implementation & Verification (AuthenticationManager and DaoAuthenticationProvider Workflow) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for AuthenticationManager

Purpose: Authenticating `/api/v1/auth/login` cleanly via `authenticationManager.authenticate(...)` before minting a JWT.

What Topic Is Next → Next Topic 16.8: Role-Based and Permission-Based Authorization (RBAC)

16.8 Role-Based and Permission-Based Authorization (RBAC) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Coarse roles (`ROLE_ADMIN` • `ROLE_CUSTOMER`) vs fine-grained authorities (`product:write` • `order:refund`) • `@AuthenticationPrincipal` in controllers • Production Edge-Case & Pitfall Guard (Role-Based and Permission-Based Authorization (RBAC)) • Hands-On Java Implementation & Verification (Role-Based and Permission-Based Authorization (RBAC)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend

Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based

Purpose: Extracting the logged-in user's ID directly from `@AuthenticationPrincipal AppUserPrincipal principal` — NEVER trusting a `userId` sent in client JSON!

What Topic Is Next → Next Topic 16.9: JWT Structure and Claims (Header, Payload, Signature)

16.9 JWT Structure and Claims (Header, Payload, Signature) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base64Url `header.payload.signature` • Standard claims (`sub` • `iss` • `iat` • Production Edge-Case & Pitfall Guard (JWT Structure and Claims (Header, Payload, Signature)) • Hands-On Java Implementation & Verification (JWT Structure and Claims (Header, Payload, Signature)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JWT Structure

Purpose: Stateless horizontal scaling where any microservice replica verifies the cryptographic signature without sticky sessions.

What Topic Is Next → Next Topic 16.10: JWT Creation and Cryptographic Validation

16.10 JWT Creation and Cryptographic Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 256-bit+ HMAC-SHA256 (`HS256`) / RSA (`RS256` in microservices) signing with `jjwt` or `spring-boot-starter-oauth2-resource-server` `NimbusJwtEncoder`/`Decoder` • Strict algorithm/expiry/issuer checks • Refresh-token concept • Production Edge-Case & Pitfall Guard (JWT Creation and Cryptographic Validation) • Hands-On Java Implementation & Verification (JWT Creation and Cryptographic Validation) • JWT Creation and Cryptographic Validation — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JWT Creation

Purpose: Issuing short-lived (15–30 min) signed access tokens and rejecting expired, tampered, or `alg: none` tokens.

What Topic Is Next → Next Topic 16.11: Custom JWT OncePerRequestFilter

16.11 Custom JWT OncePerRequestFilter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Extending `OncePerRequestFilter` • Extracting `Authorization: Bearer <token>` • Validating token • Setting `UsernamePasswordAuthenticationToken` on `SecurityContextHolder` • Production Edge-Case & Pitfall Guard (Custom JWT OncePerRequestFilter) • Hands-On Java Implementation & Verification (Custom JWT OncePerRequestFilter) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom JWT OncePerRequestFilter

Purpose: Authenticating Bearer requests cleanly once per HTTP dispatch without redundant filter execution on error forwards.

What Topic Is Next → Next Topic 16.12: Stateless Session Configuration

16.12 Stateless Session Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SessionCreationPolicy.STATELESS` • Ensuring zero `JSESSIONID` cookies or server-side `HttpSession` memory growth • Production Edge-Case & Pitfall Guard (Stateless Session Configuration) • Hands-On Java Implementation & Verification (Stateless Session Configuration) • Stateless Session Configuration — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stateless Session Configuration • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running 5 stateless Spring Boot containers behind a load balancer where any container can serve any request.

What Topic Is Next → Next Topic 16.13: CORS (Cross-Origin Resource Sharing)

16.13 CORS (Cross-Origin Resource Sharing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Browser Same-Origin Policy • `OPTIONS` preflight requests • Configuring explicit `CorsConfigurationSource` (`allowedOrigins` from env config • `allowedMethods` • Production Edge-Case & Pitfall Guard (CORS (Cross-Origin Resource Sharing)) • Hands-On Java Implementation & Verification (CORS (Cross-Origin Resource Sharing)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CORS (Cross-Origin Resource Sharing)

Purpose: Allowing your authorized web frontend domain to call `/api/v1/**` with `Authorization: Bearer` headers cleanly.

What Topic Is Next → Next Topic 16.14: CSRF Protection and Stateless API Trade-offs

16.14 CSRF Protection and Stateless API Trade-offs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Cross-Site Request Forgery exploits ambient browser cookies • Why cookie-based web apps MUST keep CSRF enabled vs why pure `Authorization: Bearer` header APIs disable CSRF • Production Edge-Case & Pitfall Guard (CSRF Protection and Stateless API Trade-offs) • Hands-On Java Implementation & Verification (CSRF Protection and Stateless API Trade-offs) • CSRF Protection and Stateless API Trade-offs — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSRF Protection • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining accurately in security interviews *when* disabling CSRF is safe (pure `Bearer` header auth, zero auth cookies) and *when* it is vulnerable (cookie-stored JWTs!...

What Topic Is Next → Next Topic 16.15: OAuth2 and OpenID Connect (OIDC) Fundamentals

16.15 OAuth2 and OpenID Connect (OIDC) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Authorization Code + PKCE flow • Client vs Authorization Server (Keycloak/Auth0/Google) vs Resource Server (`oauth2ResourceServer(oauth2 -> oauth2.jwt(...))`) • JWKS public keys • Production Edge-Case & Pitfall Guard (OAuth2 and OpenID Connect (OIDC) Fundamentals) • Hands-On Java Implementation & Verification (OAuth2 and OpenID Connect (OIDC) Fundamentals) • OAuth2 and OpenID Connect (OIDC) Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OAuth2

Purpose: Understanding how enterprise microservices validate asymmetric `RS256` JWTs via JWKS endpoints without sharing private signing keys.

What Topic Is Next → Next Topic 16.16: Method-Level Security (@EnableMethodSecurity, @PreAuthorize)

16.16 Method-Level Security (@EnableMethodSecurity, @PreAuthorize) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableMethodSecurity` • SpEL expressions `@PreAuthorize("hasRole('ADMIN')")` • `@PreAuthorize("#userId == authentication.principal.userId or hasRole('ADMIN')")` • Production Edge-Case & Pitfall Guard (Method-Level Security (@EnableMethodSecurity, @PreAuthorize)) • Hands-On Java Implementation & Verification (Method-Level Security (@EnableMethodSecurity, @PreAuthorize)) • Method-Level Security (@EnableMethodSecurity, @PreAuthorize) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Method-Level Security (@EnableMethodSecurity, @PreAu...

Purpose: Enforcing defense-in-depth authorization directly on `@Service` methods even if a new controller endpoint is added later.

What Topic Is Next → Next Topic 16.17: Security Integration Testing (spring-security-test)

16.17 Security Integration Testing (spring-security-test) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@WithMockUser(roles = "ADMIN")` • Testing unauthenticated `401` • Wrong-role `403` • Valid-JWT `200`/`201` • Production Edge-Case & Pitfall Guard (Security Integration Testing (spring-security-test)) • Hands-On Java Implementation & Verification (Security Integration Testing (spring-security-test)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Security Integration Testing (spring-security-test)

Purpose: Guaranteeing via automated CI tests that no future refactor accidentally exposes an admin or private endpoint.

What Topic Is Next → Module 16 Capstone (Major Project 4: Secure E-Commerce Backend REST API (Spring Boot 3, JPA & JWT)) → Next Topic 17.1: Monolithic Architecture (Strengths, Modularity and Scaling Limits)

Module 16 Hands-On Capstone Project (100 Marks): Major Project 4: Secure E-Commerce Backend REST API (Spring Boot 3, JPA & JWT) — Architect and build a complete, production-style monolithic Spring Boot 3 REST backend for an E-Commerce platform featuring stateless JWT authentication, role-based access control (`ADMIN`,...

MODULE 17 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 17: Microservices Architecture Fundamentals

Transition from modular Spring Boot monoliths to distributed Microservices — architectural trade-offs, Domain-Driven Design Bounded Contexts, database-per-service ownership, synchronous HTTP clients (OpenFeign, RestClient/RestTemplate, WebClient), Netflix Eureka service discovery, Spring Cloud Gateway, client-side load balancing, Spring Cloud Config, and distributed correlation IDs.

Objectives: ✓ Evaluate honestly when a modular monolith is better vs when independent Microservices are justified. • ✓ Decompose a commerce domain into clean Bounded Contexts (`user`, `product`, `order`, `inventory`, `payment`, `notification`) with strict database-per-service boundaries. • ✓ Implement synchronous service-to-service calls using declarative `OpenFeign` and modern `RestClient`/`WebClient` with explicit timeouts. • ✓ Register dynamic service instances with Eureka, route traffic through Spring Cloud Gateway, and externalize configs via Spring Cloud Config. • ✓ Propagate `X-Correlation-Id` / trace headers across service hops so distributed requests remain debuggable.

17.1 Monolithic Architecture (Strengths, Modularity and Scaling Limits) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single deployable artifact • Local ACID transactions • Low operational overhead vs deployment contention • Blast radius • Production Edge-Case & Pitfall Guard (Monolithic Architecture (Strengths, Modularity and Scaling Limits)) • Hands-On Java Implementation & Verification (Monolithic Architecture (Strengths, Modularity and Scaling Limits)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Monolithic Architecture (Strengths, Modularity and S...

Purpose: Understanding why well-engineered startups begin as a clean modular monolith before extracting high-scale microservices.

What Topic Is Next → Next Topic 17.2: Microservices Architecture (Benefits and Distributed Challenges)

17.2 Microservices Architecture (Benefits and Distributed Challenges) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Independent deployability • Fault isolation • Per-service scaling • Database-per-service autonomy vs network fallacies • Production Edge-Case & Pitfall Guard (Microservices Architecture (Benefits and Distributed Challenges)) • Hands-On Java Implementation & Verification (Microservices Architecture (Benefits and Distributed Challenges)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microservices Architecture (Benefits and Distributed...

Purpose: Scaling `product-catalog-service` to 10 read replicas during a flash sale while keeping `billing-service` at 2 replicas.

What Topic Is Next → Next Topic 17.3: Service Decomposition Strategies

17.3 Service Decomposition Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decomposing by business capability • Subdomain rather than technical layers • Sizing boundaries around transactional invariants • Production Edge-Case & Pitfall Guard (Service Decomposition Strategies) • Hands-On Java Implementation & Verification (Service Decomposition Strategies) • Service Decomposition Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Decomposition Strategies

Purpose: Splitting `Catalog`, `Order`, `Inventory`, `Payment`, and `Notification` so each owns a clear business lifecycle.

What Topic Is Next → Next Topic 17.4: Bounded Context and DDD Fundamentals

17.4 Bounded Context and DDD Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ubiquitous Language • How `"Product"` means title/description/SEO in `Catalog` Context vs SKU/warehouse-bin/qty in `Inventory` Context • Anti-Corruption Layer • Production Edge-Case & Pitfall Guard (Bounded Context and DDD Fundamentals) • Hands-On Java Implementation & Verification (Bounded Context and DDD Fundamentals) • Bounded Context and DDD Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bounded Context

Purpose: Avoiding one bloated 80-field `GodProduct` class shared across every microservice.

What Topic Is Next → Next Topic 17.5: Service Communication (Synchronous vs Asynchronous)

17.5 Service Communication (Synchronous vs Asynchronous) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Request-response HTTP/REST (immediate query/validation) vs event-driven Kafka messaging (temporal decoupling • Resilience when downstream is offline) • Production Edge-Case & Pitfall Guard (Service Communication (Synchronous vs Asynchronous)) • Hands-On Java Implementation & Verification (Service Communication (Synchronous vs Asynchronous)) • Service Communication (Synchronous vs Asynchronous) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Communication (Synchronous vs Asynchronous) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Validating SKU price synchronously while triggering email/analytics notifications asynchronously via Kafka.

What Topic Is Next → Next Topic 17.6: Service-to-Service REST Communication

17.6 Service-to-Service REST Communication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contract DTOs • Connection & read timeouts (NEVER leave infinite defaults!) • Propagating auth/correlation headers • Handling downstream `4xx` vs `5xx` • Production Edge-Case & Pitfall Guard (Service-to-Service REST Communication) • Hands-On Java Implementation & Verification (Service-to-Service REST Communication) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service-to-Service REST Communication

Purpose: `order-service` verifying product pricing & active status from `product-service` with a 2-second read timeout.

What Topic Is Next → Next Topic 17.7: Spring Cloud OpenFeign (Declarative REST Clients)

17.7 Spring Cloud OpenFeign (Declarative REST Clients) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableFeignClients` • `@FeignClient(name = "product-service")` • Declarative Spring MVC annotations • `ErrorDecoder` • Production Edge-Case & Pitfall Guard (Spring Cloud OpenFeign (Declarative REST Clients)) • Hands-On Java Implementation & Verification (Spring Cloud OpenFeign (Declarative REST Clients)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud OpenFeign (Declarative REST Clients)

Purpose: Calling `productClient.getProductBySku(sku)` as a clean, type-safe Java interface integrated with Eureka load balancing.

What Topic Is Next → Next Topic 17.8: RestTemplate Legacy Awareness & Modern Spring RestClient

17.8 RestTemplate Legacy Awareness & Modern Spring RestClient [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Maintaining `RestTemplate` in existing codebases vs migrating to Spring 6.1+ fluent synchronous `RestClient` • Production Edge-Case & Pitfall Guard (RestTemplate Legacy Awareness & Modern Spring RestClient) • Hands-On Java Implementation & Verification (RestTemplate Legacy Awareness & Modern Spring RestClient) • RestTemplate Legacy Awareness & Modern Spring RestClient — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RestTemplate Legacy Awareness • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Reading brownfield `RestTemplate` code confidently while writing clean fluent `RestClient` calls in new Spring Boot 3.2+ apps.

What Topic Is Next → Next Topic 17.9: WebClient (Non-Blocking & Reactive HTTP Client)

17.9 WebClient (Non-Blocking & Reactive HTTP Client) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WebClient.builder()` • `Mono`/`Flux` basics • Parallel scatter-gather fan-out calls • Timeout & error status handlers (`onStatus`) • Production Edge-Case & Pitfall Guard (WebClient (Non-Blocking & Reactive HTTP Client)) • Hands-On Java Implementation & Verification (WebClient (Non-Blocking & Reactive HTTP Client)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WebClient (Non-Blocking & Reactive HTTP Client)

Purpose: Firing concurrent non-blocking HTTP requests to multiple downstream services and combining responses cleanly.

What Topic Is Next → Next Topic 17.10: Service Discovery Concepts

17.10 Service Discovery Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why hardcoded `localhost:8082` breaks in autoscaled container environments • Client-side vs server-side service discovery • Heartbeats & eviction • Production Edge-Case & Pitfall Guard (Service Discovery Concepts) • Hands-On Java Implementation & Verification (Service Discovery Concepts) • Service Discovery Concepts — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Discovery Concepts

Purpose: Resolving logical service names (`lb://inventory-service`) dynamically across scaling instances.

What Topic Is Next → Next Topic 17.11: Netflix Eureka Service Registry and Discovery

17.11 Netflix Eureka Service Registry and Discovery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eureka Server registry dashboard (`8761`) • Instance registration • 30s heartbeats • Self-preservation mode • Production Edge-Case & Pitfall Guard (Netflix Eureka Service Registry and Discovery) • Hands-On Java Implementation & Verification (Netflix Eureka Service Registry and Discovery) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Netflix Eureka Service Registry

Purpose: Running a local & staging Eureka discovery registry where all microservices register and discover peers automatically.

What Topic Is Next → Next Topic 17.12: API Gateway Responsibilities

17.12 API Gateway Responsibilities [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single public edge entry point • Path routing • Centralized JWT verification • Rate limiting • Production Edge-Case & Pitfall Guard (API Gateway Responsibilities) • Hands-On Java Implementation & Verification (API Gateway Responsibilities) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Gateway Responsibilities

Purpose: Exposing only `api-gateway:8080` to external clients while keeping internal microservices on private Docker networks.

What Topic Is Next → Next Topic 17.13: Spring Cloud Gateway Implementation

17.13 Spring Cloud Gateway Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-cloud-starter-gateway` • Route Predicates (`Path`) • GatewayFilters (`AddRequestHeader` • `StripPrefix` • Production Edge-Case & Pitfall Guard (Spring Cloud Gateway Implementation) • Hands-On Java Implementation & Verification (Spring Cloud Gateway Implementation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud Gateway Implementation

Purpose: Configuring declarative YAML routes forwarding `/api/v1/orders/**` to `lb://order-service`.

What Topic Is Next → Next Topic 17.14: Client-Side Load Balancing

17.14 Client-Side Load Balancing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Cloud LoadBalancer (`@LoadBalanced`) • Round-Robin instance selection across multiple running ports (`--server.port=0`) • Production Edge-Case & Pitfall Guard (Client-Side Load Balancing) • Hands-On Java Implementation & Verification (Client-Side Load Balancing) • Client-Side Load Balancing — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client-Side Load Balancing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Distributing `order-service` traffic evenly across two running `product-service` instances and skipping unhealthy ones.

What Topic Is Next → Next Topic 17.15: Centralized Configuration Management Concepts

17.15 Centralized Configuration Management Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why managing 12 separate `application.yml` files across 6 services x 3 environments drifts without centralized config • Production Edge-Case & Pitfall Guard (Centralized Configuration Management Concepts) • Hands-On Java Implementation & Verification (Centralized Configuration Management Concepts) • Centralized Configuration Management Concepts — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized Configuration Management Concepts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Storing version-controlled service configuration in a central Git-backed config repository.

What Topic Is Next → Next Topic 17.16: Spring Cloud Config Server & Client

17.16 Spring Cloud Config Server & Client [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableConfigServer` • `spring.config.import=optional:configserver:http://localhost:8888` • Actuator `/refresh` awareness • Production Edge-Case & Pitfall Guard (Spring Cloud Config Server & Client) • Hands-On Java Implementation & Verification (Spring Cloud Config Server & Client) • Spring Cloud Config Server & Client — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud Config Server

Purpose: Serving environment-specific configuration from `config-server` to all microservices at startup.

What Topic Is Next → Next Topic 17.17: Distributed Tracing Concepts

17.17 Distributed Tracing Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Trace ID (spans entire multi-service request) vs Span ID (single hop/operation) • W3C `traceparent` header propagation • Production Edge-Case & Pitfall Guard (Distributed Tracing Concepts) • Hands-On Java Implementation & Verification (Distributed Tracing Concepts) • Distributed Tracing Concepts — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Tracing Concepts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Reconstructing exact millisecond timelines across `Gateway -> Order -> Inventory -> PostgreSQL` when debugging latency.

What Topic Is Next → Next Topic 17.18: Correlation ID Request Tracking

17.18 Correlation ID Request Tracking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Gateway GlobalFilter generating `X-Correlation-Id` if absent • Propagating via Feign `RequestInterceptor` • SLF4J MDC into every log line & error response • Production Edge-Case & Pitfall Guard (Correlation ID Request Tracking) • Hands-On Java Implementation & Verification (Correlation ID Request Tracking) • Correlation ID Request Tracking — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Correlation ID Request Tracking

Purpose: Giving support engineers a `correlationId` in API error responses that maps directly to every microservice's logs.

What Topic Is Next → Module 17 Capstone (Week 17 Lab Build: Eureka + Spring Cloud Gateway + OpenFeign Service Mesh) → Next Topic 18.1: Spring Cloud Ecosystem and Release Train Compatibility

Module 17 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Eureka + Spring Cloud Gateway + OpenFeign Service Mesh — Extract `product-service` and `order-service` behind `api-gateway` and `discovery-server` (Eureka) with declarative `OpenFeign` calls, Spring Cloud Config externalization, and `X-Correlation...

MODULE 18 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 18: Spring Cloud and Distributed Systems

Engineer fault-tolerant distributed Java backends — Eureka Server/Client hardening, Gateway JWT edge security, Spring Cloud LoadBalancer, Resilience4j Circuit Breakers, Retries with exponential backoff, Rate Limiters, Timeouts & Fallbacks, Distributed Transactions & Saga Pattern (Choreography vs Orchestration), Eventual Consistency, Idempotency Keys, Micrometer Tracing, and Observability.

Objectives: ✓ Secure the edge `api-gateway` with JWT authentication and forward verified user identity headers to internal services. • ✓ Prevent cascading microservice outages using `Resilience4j` Circuit Breakers, TimeLimiters, bounded Retries, RateLimiters, and graceful Fallbacks. • ✓ Explain why 2-Phase Commit (2PC) fails to scale across microservices and design Saga Choreography/Orchestration workflows with compensating actions. • ✓ Design idempotent `POST` / payment / consumer operations using `Idempotency-Key` tables so retries never double-charge. • ✓ Instrument logs, metrics (`Actuator` + `Micrometer`), and distributed traces (`Micrometer Tracing` + Zipkin) for production observability.

18.1 Spring Cloud Ecosystem and Release Train Compatibility [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Cloud 2023/2024 release trains compatible with Spring Boot 3.x • BOM alignment (`spring-cloud-dependencies`) • Key cloud-native starters • Production Edge-Case & Pitfall Guard (Spring Cloud Ecosystem and Release Train Compatibility) • Hands-On Java Implementation & Verification (Spring Cloud Ecosystem and Release Train Compatibility) • Spring Cloud Ecosystem and Release Train Compatibility — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud Ecosystem

Purpose: Importing `spring-cloud-dependencies` BOM so Gateway, Eureka, OpenFeign, Config, and CircuitBreaker versions always match Boot 3.x.

What Topic Is Next → Next Topic 18.2: Eureka Server Registry Setup and Health Tuning

18.2 Eureka Server Registry Setup and Health Tuning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standalone & peer-aware Eureka Server config • Actuator health endpoints • Lease expiration & renewal intervals in dev vs prod • Production Edge-Case & Pitfall Guard (Eureka Server Registry Setup and Health Tuning) • Hands-On Java Implementation & Verification (Eureka Server Registry Setup and Health Tuning) • Eureka Server Registry Setup and Health Tuning — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eureka Server Registry Setup

Purpose: Keeping a clean service registry health view across all running Major Project 5 services.

What Topic Is Next → Next Topic 18.3: Eureka Client Registration and Metadata

18.3 Eureka Client Registration and Metadata [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring.application.name` (critical service ID!) • `prefer-ip-address: true` in Docker containers • Graceful deregistration on shutdown • Production Edge-Case & Pitfall Guard (Eureka Client Registration and Metadata) • Hands-On Java Implementation & Verification (Eureka Client Registration and Metadata) • Eureka Client Registration and Metadata — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eureka Client Registration

Purpose: Setting `prefer-ip-address: true` so Docker containers advertise reachable container IPs rather than unresolvable random container hostnames.

What Topic Is Next → Next Topic 18.4: Spring Cloud Gateway Routing and Custom Filters

18.4 Spring Cloud Gateway Routing and Custom Filters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Path/Method/Header predicates • `RewritePath` • Custom `GlobalFilter` & `GatewayFilterFactory` for request logging • Header sanitization • Production Edge-Case & Pitfall Guard (Spring Cloud Gateway Routing and Custom Filters) • Hands-On Java Implementation & Verification (Spring Cloud Gateway Routing and Custom Filters) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud Gateway Routing

Purpose: Stripping spoofed client-supplied `X-User-Id` headers at the gateway before injecting verified JWT claims!

What Topic Is Next → Next Topic 18.5: API Gateway Security (Edge JWT Authentication)

18.5 API Gateway Security (Edge JWT Authentication) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating Bearer JWTs at Spring Cloud Gateway (`SecurityWebFilterChain`) • Propagating verified principal claims/tokens downstream • Production Edge-Case & Pitfall Guard (API Gateway Security (Edge JWT Authentication)) • Hands-On Java Implementation & Verification (API Gateway Security (Edge JWT Authentication)) • API Gateway Security (Edge JWT Authentication) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Gateway Security (Edge JWT Authentication) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rejecting unauthenticated or expired tokens right at the perimeter before internal services spend CPU or DB connections.

What Topic Is Next → Next Topic 18.6: Spring Cloud LoadBalancer Customization

18.6 Spring Cloud LoadBalancer Customization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Round-Robin vs Random load balancing • Health-check-aware instance filtering • Caching discovered instance lists • Production Edge-Case & Pitfall Guard (Spring Cloud LoadBalancer Customization) • Hands-On Java Implementation & Verification (Spring Cloud LoadBalancer Customization) • Spring Cloud LoadBalancer Customization — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cloud LoadBalancer Customization

Purpose: Routing inter-service calls only to `UP` instances without hitting crashed containers.

What Topic Is Next → Next Topic 18.7: Circuit Breaker Pattern (Preventing Cascading Failures)

18.7 Circuit Breaker Pattern (Preventing Cascading Failures) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why a slow downstream service takes down the entire microservice fleet via thread-pool saturation • `CLOSED` -> `OPEN` -> `HALF_OPEN` state machine • Production Edge-Case & Pitfall Guard (Circuit Breaker Pattern (Preventing Cascading Failures)) • Hands-On Java Implementation & Verification (Circuit Breaker Pattern (Preventing Cascading Failures)) • Circuit Breaker Pattern (Preventing Cascading Failures) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Circuit Breaker Pattern (Preventing Cascading Failur... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Failing fast in < 1ms when `payment-gateway` is down (`OPEN` state) instead of tying up 200 Tomcat threads waiting on timeouts.

What Topic Is Next → Next Topic 18.8: Resilience4j CircuitBreaker Implementation

18.8 Resilience4j CircuitBreaker Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-cloud-starter-circuitbreaker-resilience4j` • `@CircuitBreaker(name = "inventory" • FallbackMethod = "...")` • `slidingWindowSize` • Production Edge-Case & Pitfall Guard (Resilience4j CircuitBreaker Implementation) • Hands-On Java Implementation & Verification (Resilience4j CircuitBreaker Implementation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Resilience4j CircuitBreaker Implementation

Purpose: Wrapping `inventoryClient` calls in `order-service` with Resilience4j and monitoring live circuit state via Actuator.

What Topic Is Next → Next Topic 18.9: Retry Policies and Exponential Backoff with Jitter

18.9 Retry Policies and Exponential Backoff with Jitter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Retry` configuration • MaxAttempts • Exponential backoff + random jitter (preventing retry storms/thundering herd!) • Retrying ONLY transient idempotent operations • Production Edge-Case & Pitfall Guard (Retry Policies and Exponential Backoff with Jitter) • Hands-On Java Implementation & Verification (Retry Policies and Exponential Backoff with Jitter) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Retry Policies

Purpose: Retrying a transient network `SocketTimeoutException` 3 times (`200ms -> 400ms -> 800ms` + jitter) safely.

What Topic Is Next → Next Topic 18.10: Rate Limiting and Request Throttling

18.10 Rate Limiting and Request Throttling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Protecting login/OTP/checkout APIs from abuse • Traffic spikes using Resilience4j `@RateLimiter` (`limitForPeriod` • `limitRefreshPeriod`) -> HTTP `429 Too Many Requests` • Production Edge-Case & Pitfall Guard (Rate Limiting and Request Throttling) • Hands-On Java Implementation & Verification (Rate Limiting and Request Throttling) • Rate Limiting and Request Throttling — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rate Limiting

Purpose: Throttling `/api/v1/auth/login` and `/api/v1/orders` bursts cleanly with `429` + `Retry-After` header.

What Topic Is Next → Next Topic 18.11: Timeouts and Bulkhead Isolation

18.11 Timeouts and Bulkhead Isolation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Socket read timeouts • Resilience4j `@TimeLimiter` / `@Bulkhead` (isolating thread/semaphore pools per downstream dependency so one slow partner cannot starve healthy endpoints) • Production Edge-Case & Pitfall Guard (Timeouts and Bulkhead Isolation) • Hands-On Java Implementation & Verification (Timeouts and Bulkhead Isolation) • Timeouts and Bulkhead Isolation — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Timeouts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Capping concurrent outbound calls to a slow shipping provider via `@Bulkhead` while catalog browsing stays 100% responsive.

What Topic Is Next → Next Topic 18.12: Graceful Fallback Strategies

18.12 Graceful Fallback Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cached stale read fallback • Default degraded response • Queuing order as `PENDING_VERIFICATION` vs explicit `503 Service Unavailable` when safety requires • Production Edge-Case & Pitfall Guard (Graceful Fallback Strategies) • Hands-On Java Implementation & Verification (Graceful Fallback Strategies) • Graceful Fallback Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graceful Fallback Strategies

Purpose: Serving cached product recommendations when the recommendation service is down, while refusing unsafe blind financial debits!

What Topic Is Next → Next Topic 18.13: Distributed Transactions Problem Space

18.13 Distributed Transactions Problem Space [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why classic XA / Two-Phase Commit (2PC) locks resources across services • Kills availability (CAP theorem trade-offs) in modern cloud microservices • Production Edge-Case & Pitfall Guard (Distributed Transactions Problem Space) • Hands-On Java Implementation & Verification (Distributed Transactions Problem Space) • Distributed Transactions Problem Space — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Transactions Problem Space • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Coordinating `Order Creation` -> `Inventory Reservation` -> `Payment Capture` across 3 independent PostgreSQL databases.

What Topic Is Next → Next Topic 18.14: Saga Pattern (Choreography vs Orchestration)

18.14 Saga Pattern (Choreography vs Orchestration) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Local ACID transactions per service + compensating transactions (`ReleaseStock` if `PaymentFailed` • `CancelOrder`) • Event Choreography (peer events) vs Saga Orchestrator • Production Edge-Case & Pitfall Guard (Saga Pattern (Choreography vs Orchestration)) • Hands-On Java Implementation & Verification (Saga Pattern (Choreography vs Orchestration)) • Saga Pattern (Choreography vs Orchestration) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Saga Pattern (Choreography vs Orchestration)

Purpose: Executing `PENDING -> STOCK_RESERVED -> PAID -> CONFIRMED` (or compensating `STOCK_RELEASED -> CANCELLED` on payment failure!).

What Topic Is Next → Next Topic 18.15: Eventual Consistency in Distributed Systems

18.15 Eventual Consistency in Distributed Systems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strong consistency within an aggregate vs eventual consistency across Bounded Contexts • Intermediate lifecycle states (`PENDING_PAYMENT`) • UI/API polling & webhooks • Production Edge-Case & Pitfall Guard (Eventual Consistency in Distributed Systems) • Hands-On Java Implementation & Verification (Eventual Consistency in Distributed Systems) • Eventual Consistency in Distributed Systems — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eventual Consistency in Distributed Systems

Purpose: Returning `202 Accepted` / `status: PENDING` immediately on order submission while async Saga steps converge within milliseconds.

What Topic Is Next → Next Topic 18.16: Idempotency in Distributed APIs

18.16 Idempotency in Distributed APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Idempotency-Key` UUID header • `idempotency_keys` PostgreSQL unique constraint table storing `(key • User_id • Request_hash • Production Edge-Case & Pitfall Guard (Idempotency in Distributed APIs) • Hands-On Java Implementation & Verification (Idempotency in Distributed APIs) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Idempotency in Distributed APIs

Purpose: Guaranteeing that if a mobile app retries `POST /api/v1/orders` 3 times on flaky 4G, only ONE order and ONE payment are created!

What Topic Is Next → Next Topic 18.17: Centralized Config Server Operations

18.17 Centralized Config Server Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Environment profiles in Config Server • Health checks • Fail-fast vs optional config startup • Encrypting sensitive config properties • Production Edge-Case & Pitfall Guard (Centralized Config Server Operations) • Hands-On Java Implementation & Verification (Centralized Config Server Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized Config Server Operations

Purpose: Managing shared Kafka broker addresses, Eureka URLs, and Resilience4j thresholds cleanly across 6 services.

What Topic Is Next → Next Topic 18.18: Distributed Tracing with Micrometer Tracing

18.18 Distributed Tracing with Micrometer Tracing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Boot 3 `micrometer-tracing-bridge-brave` + `zipkin-reporter-brave` (replaces deprecated Spring Cloud Sleuth!) • Automatic `[service • TraceId • SpanId]` log pattern • Production Edge-Case & Pitfall Guard (Distributed Tracing with Micrometer Tracing) • Hands-On Java Implementation & Verification (Distributed Tracing with Micrometer Tracing) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Tracing with Micrometer Tracing

Purpose: Visualizing end-to-end request waterfall spans across Gateway, Order, Product, and Inventory in Zipkin.

What Topic Is Next → Next Topic 18.19: Three Pillars of Observability (Logs, Metrics and Traces)

18.19 Three Pillars of Observability (Logs, Metrics and Traces) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structured JSON logs (what happened) • Actuator + Micrometer Prometheus metrics (`/actuator/health` • `/actuator/metrics` • JVM/HikariCP/HTTP p95 latency) • Production Edge-Case & Pitfall Guard (Three Pillars of Observability (Logs, Metrics and Traces)) • Hands-On Java Implementation & Verification (Three Pillars of Observability (Logs, Metrics and Traces)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Three Pillars of Observability (Logs, Metrics and Tr...

Purpose: Monitoring `http.server.requests` p95 latency, HikariCP active connections, and custom `orders.placed` counters via Micrometer `MeterRegistry`.

What Topic Is Next → Module 18 Capstone (Week 18 Lab Build: Fault-Tolerant Resilience4j & Idempotent Saga Checkout) → Next Topic 19.1: Messaging Fundamentals and Event-Driven Architecture

Module 18 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: Fault-Tolerant Resilience4j & Idempotent Saga Checkout — Harden `order-service` -> `inventory-service` -> `payment-service` interactions using Resilience4j CircuitBreakers, exponential-backoff Retries, RateLimiters, `Idempotency-Key` deduplication...

MODULE 19 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 19: Apache Kafka and Asynchronous Communication

Build high-throughput, fault-tolerant event-driven Java backends with Apache Kafka and Spring for Apache Kafka — brokers, topics, partitions, keys, producers (`KafkaTemplate`), consumers (`@KafkaListener`), consumer groups, offsets & commits, replication/acks, JSON/Avro serialization, retry topics, Dead Letter Queues (DLQ), ordering guarantees, idempotent consumers, and the Transactional Outbox Pattern.

Objectives: ✓ Contrast traditional point-to-point queues with Apache Kafka's partitioned, append-only distributed commit log. • ✓ Publish keyed domain events reliably using Spring `KafkaTemplate` with `acks=all` and idempotent producer settings. • ✓ Scale parallel consumption using Consumer Groups while understanding partition-to-consumer cardinality and offset commits. • ✓ Handle poison-pill messages cleanly with `DefaultErrorHandler` / `@RetryableTopic` backoff retries and Dead Letter Topics (`.DLT`). • ✓ Guarantee per-entity ordering via partition keys, prevent duplicate side effects via idempotent `processed_events` tables, and avoid dual-write bugs via the Transactional Outbox Pattern.

19.1 Messaging Fundamentals and Event-Driven Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Producers • Brokers • Consumers • Point-to-point queues vs pub/sub event streams • Production Edge-Case & Pitfall Guard (Messaging Fundamentals and Event-Driven Architecture) • Hands-On Java Implementation & Verification (Messaging Fundamentals and Event-Driven Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Messaging Fundamentals

Purpose: Allowing `order-service` to accept orders in 15ms even while `notification-service` (SMTP email) experiences temporary slowness.

What Topic Is Next → Next Topic 19.2: Synchronous REST vs Asynchronous Kafka Trade-offs

19.2 Synchronous REST vs Asynchronous Kafka Trade-offs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Latency decoupling • Spike buffering/backpressure • Multi-subscriber fan-out vs debugging complexity • Eventual consistency • Production Edge-Case & Pitfall Guard (Synchronous REST vs Asynchronous Kafka Trade-offs) • Hands-On Java Implementation & Verification (Synchronous REST vs Asynchronous Kafka Trade-offs) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronous REST vs Asynchronous Kafka Trade-offs

Purpose: Having `inventory-service`, `notification-service`, and `analytics-service` all react independently to one `order.placed` topic without changing `order-service`!

What Topic Is Next → Next Topic 19.3: Kafka Architecture (Brokers, Topics, Partitions and KRaft)

19.3 Kafka Architecture (Brokers, Topics, Partitions and KRaft) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable append-only partition logs • Partition keys (`hash(key) % numPartitions`) • Retention by time/size (messages are NOT deleted on read!) • Modern KRaft metadata mode • Production Edge-Case & Pitfall Guard (Kafka Architecture (Brokers, Topics, Partitions and KRaft)) • Hands-On Java Implementation & Verification (Kafka Architecture (Brokers, Topics, Partitions and KRaft)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Architecture (Brokers, Topics, Partitions and...

Purpose: Replaying historical events for a brand-new analytics consumer because Kafka retains events for 7 days regardless of prior reads.

What Topic Is Next → Next Topic 19.4: Kafka Producers (Partitioning, Batching and Acks)

19.4 Kafka Producers (Partitioning, Batching and Acks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Key-based murmur2 partitioning • `acks=0` vs `acks=1` vs `acks=all` • `enable.idempotence=true` (default in modern Kafka!) • `linger.ms` & `batch.size` • Production Edge-Case & Pitfall Guard (Kafka Producers (Partitioning, Batching and Acks)) • Hands-On Java Implementation & Verification (Kafka Producers (Partitioning, Batching and Acks)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Producers (Partitioning, Batching and Acks)

Purpose: Passing `String.valueOf(orderId)` as the message key so all events for Order #42 land in the same partition in exact chronological order!

What Topic Is Next → Next Topic 19.5: Kafka Consumers (Poll Loop and Deserialization)

19.5 Kafka Consumers (Poll Loop and Deserialization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pull-based `poll()` model • `auto.offset.reset` (`earliest` vs `latest`) • Heartbeats vs `max.poll.interval.ms` • Production Edge-Case & Pitfall Guard (Kafka Consumers (Poll Loop and Deserialization)) • Hands-On Java Implementation & Verification (Kafka Consumers (Poll Loop and Deserialization)) • Kafka Consumers (Poll Loop and Deserialization) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Consumers (Poll Loop and Deserialization)

Purpose: Setting `auto-offset-reset: earliest` in microservices so a newly deployed consumer group never misses events published right before it booted.

What Topic Is Next → Next Topic 19.6: Consumer Groups and Parallelism

19.6 Consumer Groups and Parallelism [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Load-sharing partitions within one `group.id` vs independent broadcast across different `group.id`s • Why `activeConsumers <= partitionCount` • Production Edge-Case & Pitfall Guard (Consumer Groups and Parallelism) • Hands-On Java Implementation & Verification (Consumer Groups and Parallelism) • Consumer Groups and Parallelism — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Consumer Groups • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running 3 `inventory-service` instances (`group.id=inventory`) sharing 3 partitions in parallel while `notification-service` (`group.id=notifications`) receives its own f...

What Topic Is Next → Next Topic 19.7: Offsets and Commit Strategies

19.7 Offsets and Commit Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Current offset vs committed offset (`__consumer_offsets`) • Auto-commit risks (data loss if crash before processing finishes!) vs Spring Kafka `RECORD`/`BATCH` commit-after-process • Production Edge-Case & Pitfall Guard (Offsets and Commit Strategies) • Hands-On Java Implementation & Verification (Offsets and Commit Strategies) • Offsets and Commit Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Offsets • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring offsets commit ONLY *after* database persistence succeeds (`enable-auto-commit: false` + Spring `AckMode.RECORD`), guaranteeing At-Least-Once delivery.

What Topic Is Next → Next Topic 19.8: Partition Replication, ISR and Fault Tolerance

19.8 Partition Replication, ISR and Fault Tolerance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replication factor (`RF=3` in prod) • Partition Leader vs Followers • In-Sync Replicas (`ISR`) • `min.insync.replicas=2` + `acks=all` durability guarantee • Production Edge-Case & Pitfall Guard (Partition Replication, ISR and Fault Tolerance) • Hands-On Java Implementation & Verification (Partition Replication, ISR and Fault Tolerance) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Partition Replication

Purpose: Guaranteeing zero committed event loss even if a Kafka broker disk dies abruptly.

What Topic Is Next → Next Topic 19.9: Message Serialization (JSON and Apache Avro Fundamentals)

19.9 Message Serialization (JSON and Apache Avro Fundamentals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JsonSerializer` / `ErrorHandlingDeserializer` wrapping `JsonDeserializer` (critical for poison pills!) • Backward/forward schema evolution & Avro Schema Registry overview • Production Edge-Case & Pitfall Guard (Message Serialization (JSON and Apache Avro Fundamentals)) • Hands-On Java Implementation & Verification (Message Serialization (JSON and Apache Avro Fundamentals)) • Message Serialization (JSON and Apache Avro Fundamentals) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Serialization (JSON and Apache Avro Fundamen... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Wrapping `JsonDeserializer` inside `ErrorHandlingDeserializer` so a single malformed JSON message routes to DLQ instead of bricking the consumer in an infinite deserializ...

What Topic Is Next → Next Topic 19.10: Spring for Apache Kafka Integration

19.10 Spring for Apache Kafka Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-kafka` starter • `KafkaTemplate<String • Object>` • `NewTopic` `@Bean` auto-creation • Production Edge-Case & Pitfall Guard (Spring for Apache Kafka Integration) • Hands-On Java Implementation & Verification (Spring for Apache Kafka Integration) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring for Apache Kafka Integration

Purpose: Auto-provisioning `order-events`, `inventory-events`, and `payment-events` topics cleanly via `TopicBuilder`.

What Topic Is Next → Next Topic 19.11: Kafka Listeners (@KafkaListener)

19.11 Kafka Listeners (@KafkaListener) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@KafkaListener(topics = ... • GroupId = ...)` • Extracting `@Payload` • `@Header(KafkaHeaders.RECEIVED_KEY)` • Production Edge-Case & Pitfall Guard (Kafka Listeners (@KafkaListener)) • Hands-On Java Implementation & Verification (Kafka Listeners (@KafkaListener)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Listeners (@KafkaListener)

Purpose: Logging topic, partition, offset, and `eventId` cleanly in every consumer handler.

What Topic Is Next → Next Topic 19.12: Consumer Error Handling

19.12 Consumer Error Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Transient errors (DB/network blip -> retry!) vs non-retryable errors (`IllegalArgumentException` / invalid data -> skip straight to DLT!) • `DefaultErrorHandler` + `FixedBackOff` / `ExponentialBackOff` • Production Edge-Case & Pitfall Guard (Consumer Error Handling) • Hands-On Java Implementation & Verification (Consumer Error Handling) • Consumer Error Handling — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Consumer Error Handling • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing a single bad event from blocking 10,000 valid subsequent events behind it in the partition.

What Topic Is Next → Next Topic 19.13: Retry Topics (@RetryableTopic)

19.13 Retry Topics (@RetryableTopic) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Non-blocking delayed retries via Spring Kafka `@RetryableTopic(attempts = "4" • Backoff = @Backoff(delay = 1000 • Multiplier = 2.0))` • Main partition stays unblocked! • Production Edge-Case & Pitfall Guard (Retry Topics (@RetryableTopic)) • Hands-On Java Implementation & Verification (Retry Topics (@RetryableTopic)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Retry Topics (@RetryableTopic)

Purpose: Retrying temporary SMTP/webhook failures across `-retry-0`, `-retry-1` topics without stalling real-time order events on the main topic.

What Topic Is Next → Next Topic 19.14: Dead Letter Queue (DLQ / DLT)

19.14 Dead Letter Queue (DLQ / DLT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DeadLetterPublishingRecoverer` • `@DltHandler` • Storing diagnostic failure headers (`kafka_dlt-exception-message`) • Alerting & replay runbooks • Production Edge-Case & Pitfall Guard (Dead Letter Queue (DLQ / DLT)) • Hands-On Java Implementation & Verification (Dead Letter Queue (DLQ / DLT)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dead Letter Queue (DLQ / DLT)

Purpose: Capturing exhausted failed messages into `order-events.DLT` + persisting an admin alert record so zero customer events vanish silently.

What Topic Is Next → Next Topic 19.15: Event-Driven Design (Domain Events vs Integration Events)

19.15 Event-Driven Design (Domain Events vs Integration Events) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Internal Domain Events vs public versioned cross-service Integration Events • Thin notification events vs self-contained Event-Carried State Transfer • Schema backward compatibility (additive optional fields!) • Production Edge-Case & Pitfall Guard (Event-Driven Design (Domain Events vs Integration Events)) • Hands-On Java Implementation & Verification (Event-Driven Design (Domain Events vs Integration Events)) • Event-Driven Design (Domain Events vs Integration Events) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Design (Domain Events vs Integration Ev...

Purpose: Including `eventId`, `eventType`, `schemaVersion`, `occurredAt`, and `correlationId` metadata on every integration event.

What Topic Is Next → Next Topic 19.16: Message Ordering Guarantees

19.16 Message Ordering Guarantees [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Per-partition FIFO guarantee • Choosing the right partition key (`orderId` for order lifecycle • `accountId` for ledger) • `max.in.flight.requests.per.connection <= 5` with idempotent producer • Production Edge-Case & Pitfall Guard (Message Ordering Guarantees) • Hands-On Java Implementation & Verification (Message Ordering Guarantees) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Ordering Guarantees

Purpose: Ensuring `OrderPlaced -> OrderPaid -> OrderShipped` for `orderId=9001` are always processed in exact sequence.

What Topic Is Next → Next Topic 19.17: Idempotent Consumers (Deduplicating At-Least-Once Delivery)

19.17 Idempotent Consumers (Deduplicating At-Least-Once Delivery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why network/rebalance hiccups cause duplicate deliveries in Kafka ("At-Least-Once") • Inserting `event_id UUID PRIMARY KEY` into `processed_events` inside the SAME local `@Transactional` DB commit as the business write! • Production Edge-Case & Pitfall Guard (Idempotent Consumers (Deduplicating At-Least-Once Delivery)) • Hands-On Java Implementation & Verification (Idempotent Consumers (Deduplicating At-Least-Once Delivery)) • Idempotent Consumers (Deduplicating At-Least-Once Delivery) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Idempotent Consumers (Deduplicating At-Least-Once De... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring that if `PaymentConfirmedEvent(eventId=abc)` is delivered twice after a rebalance, inventory is deducted and email sent ONLY ONCE!

What Topic Is Next → Next Topic 19.18: Transactional Outbox Pattern

19.18 Transactional Outbox Pattern [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: The Dual-Write bug (`orderRepo.save()` commits in PostgreSQL • Then JVM crashes 1ms before `kafkaTemplate.send()` -> lost event!) • Solving by writing `Order` + `OutboxEvent` row in ONE PostgreSQL transaction • Relaying via `@Scheduled` poller / CDC • Production Edge-Case & Pitfall Guard (Transactional Outbox Pattern) • Hands-On Java Implementation & Verification (Transactional Outbox Pattern) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transactional Outbox Pattern

Purpose: Achieving 100% reliable event publishing in financial and e-commerce order services without distributed XA transactions.

What Topic Is Next → Module 19 Capstone (Week 19 Lab Build: Event-Driven Kafka Order & Notification Pipeline) → Next Topic 20.1: NoSQL vs Relational Databases (Polyglot Persistence)

Module 19 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Event-Driven Kafka Order & Notification Pipeline — Implement asynchronous `OrderPlaced`, `InventoryUpdated`, `PaymentConfirmed`, and `EmailNotification` workflows using Spring Kafka, partition keys, `@RetryableTopic`, `.DLT`

Dead Letter Queu...

MODULE 20 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 20: Redis, NoSQL and Backend Performance

Accelerate distributed backends using Polyglot Persistence (MongoDB document fundamentals) and Redis in-memory caching & rate limiting — MongoDB documents/indexes/aggregations, Spring Data MongoDB, Redis Strings/Hashes/Lists/Sets, Cache-Aside pattern, Spring Cache (@Cacheable/@CachePut/@CacheEvict), TTL & stampede prevention, HikariCP & N+1 tuning, and Redis rate limiting — culminating in Major Project 5.

Objectives: ✓ Choose objectively between PostgreSQL (relational ACID), MongoDB (flexible document schemas), and Redis (sub-millisecond in-memory state). • ✓ Model documents, create compound indexes, run aggregation pipelines, and build `MongoRepository` services in Spring Data MongoDB. • ✓ Use Redis Strings, Hashes, Lists, and Sets and implement Cache-Aside caching cleanly with `@Cacheable`, `@CachePut`, and `@CacheEvict`. • ✓ Configure randomized TTLs, JSON Redis serialization, cache invalidation on writes, and Redis-backed API rate limiting. • ✓ Deliver Month 5 Deliverable: Major Project 5 — Distributed E-Commerce Microservices Application (`Gateway`, `Eureka`, `User`, `Product`, `Order`, `Inventory`, `Payment`, `Notification`, `Kafka`, `Redis`, `PostgreSQL`).

20.1 NoSQL vs Relational Databases (Polyglot Persistence) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Document • Key-Value • Wide-Column • Graph stores vs RDBMS • Production Edge-Case & Pitfall Guard (NoSQL vs Relational Databases (Polyglot Persistence)) • Hands-On Java Implementation & Verification (NoSQL vs Relational Databases (Polyglot Persistence)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NoSQL vs Relational Databases (Polyglot Persistence)

Purpose: Matching each microservice's storage engine to its exact access pattern instead of forcing one tool everywhere.

What Topic Is Next → Next Topic 20.2: MongoDB Fundamentals

20.2 MongoDB Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: BSON binary JSON documents • Collections • Dynamic/schema-validated document structure • `ObjectId` 12-byte structure • Production Edge-Case & Pitfall Guard (MongoDB Fundamentals) • Hands-On Java Implementation & Verification (MongoDB Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Fundamentals

Purpose: Storing polymorphic category attributes (laptops have `ramGb`/`cpu`; shirts have `fabric`/`chestIn`) without 40 nullable SQL columns.

What Topic Is Next → Next Topic 20.3: MongoDB Data Modeling (Embedding vs Referencing)

20.3 MongoDB Data Modeling (Embedding vs Referencing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Embedding bounded 1:few sub-documents (`addresses` on `user`) vs Referencing unbounded 1:million collections (`reviews` referencing `productId`) • 16MB BSON document limit! • Production Edge-Case & Pitfall Guard (MongoDB Data Modeling (Embedding vs Referencing)) • Hands-On Java Implementation & Verification (MongoDB Data Modeling (Embedding vs Referencing)) • MongoDB Data Modeling (Embedding vs Referencing) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Data Modeling (Embedding vs Referencing) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Embedding 3–8 product specification pairs inside `ProductDoc` while keeping unbounded customer `Reviews` as a separate collection!

What Topic Is Next → Next Topic 20.4: MongoDB CRUD Operations

20.4 MongoDB CRUD Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `insertOne`/`insertMany` • `find` projections • `$gt`/`$in` filters • Atomic `$set` / `$inc` / `$push` updates (never overwrite whole doc blindly!) • Production Edge-Case & Pitfall Guard (MongoDB CRUD Operations) • Hands-On Java Implementation & Verification (MongoDB CRUD Operations) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB CRUD Operations

Purpose: Atomically incrementing view counters (`$inc: { views: 1 }`) without read-modify-write lost updates.

What Topic Is Next → Next Topic 20.5: MongoDB Indexing

20.5 MongoDB Indexing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default `_id` index • Compound indexes & ESR rule (`Equality -> Sort -> Range`) • Text/TTL indexes (`expireAfterSeconds`) • `.explain("executionStats")` (`IXSCAN` vs `COLLSCAN`) • Production Edge-Case & Pitfall Guard (MongoDB Indexing) • Hands-On Java Implementation & Verification (MongoDB Indexing) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Indexing

Purpose: Auto-expiring audit/OTP documents via a MongoDB TTL index and verifying `IXSCAN` via `explain`.

What Topic Is Next → Next Topic 20.6: MongoDB Aggregation Pipeline Fundamentals

20.6 MongoDB Aggregation Pipeline Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage `$match` (push early to use indexes!) -> `$group` (`$avg` • `$sum`) -> `$sort` -> `$limit` -> `$project` • Production Edge-Case & Pitfall Guard (MongoDB Aggregation Pipeline Fundamentals) • Hands-On Java Implementation & Verification (MongoDB Aggregation Pipeline Fundamentals) • MongoDB Aggregation Pipeline Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Aggregation Pipeline Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Computing average star rating and rating-count breakdown per `productId` directly inside MongoDB.

What Topic Is Next → Next Topic 20.7: Spring Data MongoDB (MongoRepository and MongoTemplate)

20.7 Spring Data MongoDB (MongoRepository and MongoTemplate) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Document(collection = "reviews")` • `MongoRepository<ReviewDoc • String>` • Derived queries • Production Edge-Case & Pitfall Guard (Spring Data MongoDB (MongoRepository and MongoTemplate)) • Hands-On Java Implementation & Verification (Spring Data MongoDB (MongoRepository and MongoTemplate)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Data MongoDB (MongoRepository and MongoTempla...

Purpose: Integrating a clean `ReviewRepository` microservice module with Spring Boot 3.

What Topic Is Next → Next Topic 20.8: Redis In-Memory Data Store Architecture

20.8 Redis In-Memory Data Store Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-threaded event-loop command execution (atomic commands • Sub-millisecond latency!) • RDB snapshots & AOF persistence modes • Lettuce non-blocking Spring Boot client • Production Edge-Case & Pitfall Guard (Redis In-Memory Data Store Architecture) • Hands-On Java Implementation & Verification (Redis In-Memory Data Store Architecture) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis In-Memory Data Store Architecture

Purpose: Serving hot product details, active promos, token blacklists, and rate-limit counters in < 1ms.

What Topic Is Next → Next Topic 20.9: Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)

20.9 Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strings (`GET/SET EX/INCR`) • Hashes (`HSET/HGETALL` for user carts!) • Lists (`LPUSH/RPOP`) • Sets (`SADD/SISMEMBER`) • Production Edge-Case & Pitfall Guard (Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)) • Hands-On Java Implementation & Verification (Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Core Data Structures (Strings, Hashes, Lists,...

Purpose: Storing active shopping carts as Redis Hashes (`cart:{userId}` -> `field=productId, value=qty`) with automatic 7-day TTL!

What Topic Is Next → Next Topic 20.10: Caching Patterns (Cache-Aside, Write-Through, Read-Through)

20.10 Caching Patterns (Cache-Aside, Write-Through, Read-Through) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cache-Aside (Read: check Redis -> on miss read PostgreSQL -> populate Redis with TTL • Write: update PostgreSQL -> evict Redis key!) • Cache hit ratio formulas • Production Edge-Case & Pitfall Guard (Caching Patterns (Cache-Aside, Write-Through, Read-Through)) • Hands-On Java Implementation & Verification (Caching Patterns (Cache-Aside, Write-Through, Read-Through)) • Caching Patterns (Cache-Aside, Write-Through, Read-Through) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Caching Patterns (Cache-Aside, Write-Through, Read-T...

Purpose: Offloading 90%+ of repetitive `GET /api/v1/products/{id}` queries from PostgreSQL during high-traffic sales.

What Topic Is Next → Next Topic 20.11: Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)

20.11 Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableCaching` • Configuring `RedisCacheManager` with `GenericJackson2JsonRedisSerializer` (not unreadable JDK binary serialization!) • SpEL keys `key = "#id"` • `unless = "#result == null"` • Production Edge-Case & Pitfall Guard (Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)) • Hands-On Java Implementation & Verification (Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Cache Abstraction (@Cacheable, @CachePut, @Ca...

Purpose: Declaratively caching `getProductById` with `@Cacheable(cacheNames = "products", key = "#id")` and evicting on `updateProduct`/`deleteProduct`!

What Topic Is Next → Next Topic 20.12: Cache Invalidation, Penetration and Stampede Protection

20.12 Cache Invalidation, Penetration and Stampede Protection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why cache invalidation is hard • Cache Penetration (caching short-TTL empty sentinel or Bloom filter for non-existent IDs) • Cache Avalanche (adding random TTL jitter!) • Cache Stampede (`sync = true` lock) • Production Edge-Case & Pitfall Guard (Cache Invalidation, Penetration and Stampede Protection) • Hands-On Java Implementation & Verification (Cache Invalidation, Penetration and Stampede Protection) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache Invalidation

Purpose: Using `@Cacheable(..., sync = true)` on hot homepage/banner keys so 500 concurrent threads don't stampede PostgreSQL simultaneously when TTL expires!

What Topic Is Next → Next Topic 20.13: Time-To-Live (TTL) and Eviction Policies

20.13 Time-To-Live (TTL) and Eviction Policies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mandatory TTL on every cache key (`EXPIRE`) • `maxmemory` policies (`allkeys-lru` • `volatile-lru` • `noeviction`) • Production Edge-Case & Pitfall Guard (Time-To-Live (TTL) and Eviction Policies) • Hands-On Java Implementation & Verification (Time-To-Live (TTL) and Eviction Policies) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Time-To-Live (TTL)

Purpose: Setting 15-minute TTL on catalog pages, 5-minute TTL on OTPs, and exact remaining-JWT-lifetime TTL on revoked token `jti` blacklists!

What Topic Is Next → Next Topic 20.14: Backend Database and API Performance Profiling

20.14 Backend Database and API Performance Profiling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Measuring p50/p95/p99 latency (not misleading averages!) • SQL query count assertions • Payload compression • Avoiding chatty N-call loops • Production Edge-Case & Pitfall Guard (Backend Database and API Performance Profiling) • Hands-On Java Implementation & Verification (Backend Database and API Performance Profiling) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backend Database

Purpose: Running a 50-concurrent-user JMeter test plan to measure p95 latency before and after Redis caching + SQL indexing.

What Topic Is Next → Next Topic 20.15: Connection Pooling Tuning in Microservices

20.15 Connection Pooling Tuning in Microservices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sizing HikariCP per microservice (`total_connections = replicas * maxPoolSize < PostgreSQL max_connections`) • `leak-detection-threshold` • Fast `connection-timeout` • Production Edge-Case & Pitfall Guard (Connection Pooling Tuning in Microservices) • Hands-On Java Implementation & Verification (Connection Pooling Tuning in Microservices) • Connection Pooling Tuning in Microservices — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection Pooling Tuning in Microservices

Purpose: Configuring `maximum-pool-size: 10` across 6 microservices (60 total connections < PostgreSQL default 100 limit).

What Topic Is Next → Next Topic 20.16: Detecting and Eliminating N+1 Queries at Scale

20.16 Detecting and Eliminating N+1 Queries at Scale [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EntityGraph(attributePaths = {"category"})` • Constructor DTO projections • Two-query ID-page + `IN (:ids)` collection fetch • Hibernate statistics check • Production Edge-Case & Pitfall Guard (Detecting and Eliminating N+1 Queries at Scale) • Hands-On Java Implementation & Verification (Detecting and Eliminating N+1 Queries at Scale) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Detecting

Purpose: Fetching 20 paginated orders + all their `OrderItem` lines in exactly 2 SQL queries instead of 21 queries (`1 + 20`)!

What Topic Is Next → Next Topic 20.17: High-Volume API Pagination Strategies

20.17 High-Volume API Pagination Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Slice<T>` (avoids expensive `SELECT COUNT(*)` on millions of rows!) • Cursor-based `?afterId=1050&limit=20` pagination for feeds/events • Production Edge-Case & Pitfall Guard (High-Volume API Pagination Strategies) • Hands-On Java Implementation & Verification (High-Volume API Pagination Strategies) • High-Volume API Pagination Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for High-Volume API Pagination Strategies • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Powering sub-5ms order history and job-listing feeds without full-table `COUNT(*)` bottlenecks.

What Topic Is Next → Next Topic 20.18: Redis-Based Distributed Rate Limiting

20.18 Redis-Based Distributed Rate Limiting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why JVM-local rate limiters leak `N * limit` across `N` replicas • Atomic Redis fixed-window (`INCR` + `EXPIRE` Lua script) & Spring Cloud Gateway `RequestRateLimiter` token bucket • Production Edge-Case & Pitfall Guard (Redis-Based Distributed Rate Limiting) • Hands-On Java Implementation & Verification (Redis-Based Distributed Rate Limiting) • Redis-Based Distributed Rate Limiting — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis-Based Distributed Rate Limiting • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enforcing a cluster-wide limit of `10 req/sec` per IP/API-key at `api-gateway` backed by shared Redis!

What Topic Is Next → Module 20 Capstone (Major Project 5: Distributed E-Commerce Microservices Application) → Next Topic 21.1: Software Testing Fundamentals (Unit, Integration and System Testing)

Module 20 Hands-On Capstone Project (100 Marks): Major Project 5: Distributed E-Commerce Microservices Application — Evolve the commerce domain into a resilient, event-driven distributed Microservices system consisting of independently deployable Spring Boot 3 services (`API Gateway`, `Eureka Server`, `Use...

MODULE 21 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 21: Testing and Quality Engineering

Build automated quality confidence across every backend layer — the Testing Pyramid, JUnit 5 lifecycle & parameterized tests, Mockito mocks/stubs/spies/captors, Spring Boot Test slices (@WebMvcTest + MockMvc, @DataJpaTest), full-stack integration testing with real Docker PostgreSQL via Testcontainers, Postman/OpenAPI contract checks, JaCoCo coverage, and regression discipline.

Objectives: ✓ Balance fast isolated unit tests, focused Spring slice tests (`@WebMvcTest`, `@DataJpaTest`), and real-database integration tests (`@SpringBootTest` + `Testcontainers`). • ✓ Write expressive JUnit 5 tests using `@BeforeEach`, `@Nested`, `@ParameterizedTest`, and clear `Arrange -> Act -> Assert` structure. • ✓ Isolate `@Service` business logic in < 50ms using `@ExtendWith(MockitoExtension.class)`, `@Mock`, `@InjectMocks`, `when(...).thenReturn(...)`, and `verify(...)`. • ✓ Test HTTP status codes, `@Valid` validation errors, and JWT/RBAC rules via `@WebMvcTest` + `MockMvc`, and verify PostgreSQL queries against real containerized Postgres via `Testcontainers`. • ✓ Measure meaningful JaCoCo branch/line coverage and prevent production regressions.

21.1 Software Testing Fundamentals (Unit, Integration and System Testing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deterministic fast unit tests vs component/DB integration tests vs end-to-end API contract tests • Arrange-Act-Assert (Given-When-Then) discipline • Production Edge-Case & Pitfall Guard (Software Testing Fundamentals (Unit, Integration and System Testing)) • Hands-On Java Implementation & Verification (Software Testing Fundamentals (Unit, Integration and System Testing)) • Software Testing Fundamentals (Unit, Integration and System Testing) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Software Testing Fundamentals (Unit, Integration and... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Catching broken business calculations, SQL bugs, and security regressions automatically before any PR merges.

What Topic Is Next → Next Topic 21.2: The Testing Pyramid in Spring Boot Backends

21.2 The Testing Pyramid in Spring Boot Backends [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ~70% fast Mockito/domain unit tests • ~20% `@WebMvcTest`/`@DataJpaTest`/`Testcontainers` slice & integration tests • ~10% E2E smoke workflows • Production Edge-Case & Pitfall Guard (The Testing Pyramid in Spring Boot Backends) • Hands-On Java Implementation & Verification (The Testing Pyramid

in Spring Boot Backends) • The Testing Pyramid in Spring Boot Backends — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The Testing Pyramid in Spring Boot Backends

Purpose: Keeping `./mvnw test` under 30 seconds so developers actually run tests before every commit.

What Topic Is Next → Next Topic 21.3: JUnit 5 (Jupiter) Architecture and Lifecycle

21.3 JUnit 5 (Jupiter) Architecture and Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Test` • `@DisplayName` • `@BeforeEach` • `@AfterEach` • Production Edge-Case & Pitfall Guard (JUnit 5 (Jupiter) Architecture and Lifecycle) • Hands-On Java Implementation & Verification (JUnit 5 (Jupiter) Architecture and Lifecycle) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JUnit 5 (Jupiter) Architecture

Purpose: Organizing `OrderServiceTest` into readable `@Nested class WhenPlacingOrder` and `@Nested class WhenCancellingOrder` suites.

What Topic Is Next → Next Topic 21.4: JUnit 5 Assertions and Exception Verification

21.4 JUnit 5 Assertions and Exception Verification [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `assertEquals` • `assertTrue` • `assertNotNull` • `assertAll` (reports ALL failures at once!) • Production Edge-Case & Pitfall Guard (JUnit 5 Assertions and Exception Verification) • Hands-On Java Implementation & Verification (JUnit 5 Assertions and Exception Verification) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JUnit 5 Assertions

Purpose: Verifying both happy-path return DTO fields (`assertAll`) and exact domain exceptions (`assertThrows`) when business rules reject bad inputs.

What Topic Is Next → Next Topic 21.5: Parameterized Tests (@ParameterizedTest)

21.5 Parameterized Tests (@ParameterizedTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@ValueSource` • `@CsvSource` • `@EnumSource` • `@MethodSource` for testing 10 boundary/invalid inputs with zero copy-pasted test methods • Production Edge-Case & Pitfall Guard (Parameterized Tests (@ParameterizedTest)) • Hands-On Java Implementation & Verification (Parameterized Tests (@ParameterizedTest)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Parameterized Tests (@ParameterizedTest)

Purpose: Testing tax slabs, coupon rules, SKU regexes, and password strength across 12 edge cases in one concise method.

What Topic Is Next → Next Topic 21.6: Mockito Fundamentals (@ExtendWith(MockitoExtension.class))

21.6 Mockito Fundamentals (@ExtendWith(MockitoExtension.class)) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pure Java + Mockito unit tests (zero Spring context startup!) • `@Mock` • `@InjectMocks` • Stubbing `when(repo.findById(1L)).thenReturn(Optional.of(entity))` • Production Edge-Case & Pitfall Guard (Mockito Fundamentals (@ExtendWith(MockitoExtension.class))) • Hands-On Java Implementation & Verification (Mockito Fundamentals (@ExtendWith(MockitoExtension.class))) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mockito Fundamentals (@ExtendWith(MockitoExtension.c...

Purpose: Testing 15 complex `OrderService` branches (out of stock, expired coupon, duplicate SKU, inactive user) in < 100ms total!

What Topic Is Next → Next Topic 21.7: Test Doubles (Mock vs Stub vs Spy vs Fake)

21.7 Test Doubles (Mock vs Stub vs Spy vs Fake) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stubs (canned answers) • Mocks (interaction verification) • `@Spy` (partial real object — use sparingly!) • In-memory Fakes (`FakeClock` • Production Edge-Case & Pitfall Guard (Test Doubles (Mock vs Stub vs Spy vs Fake)) • Hands-On Java Implementation & Verification (Test Doubles (Mock vs Stub vs Spy vs Fake)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test Doubles (Mock vs Stub vs Spy vs Fake)

Purpose: Passing `Clock.fixed(Instant.parse("2026-11-17T00:00:00Z"), ZoneOffset.UTC)` instead of mocking static `Instant.now()`!

What Topic Is Next → Next Topic 21.8: Interaction Verification (verify, ArgumentCaptor and Matchers)

21.8 Interaction Verification (verify, ArgumentCaptor and Matchers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `verify(kafkaTemplate).send(...)` • `verify(repo • Never()).save(any())` on validation failure • `ArgumentCaptor<Order>` to inspect exact saved fields • Production Edge-Case & Pitfall Guard (Interaction Verification (verify, ArgumentCaptor and Matchers)) • Hands-On Java Implementation & Verification (Interaction Verification (verify, ArgumentCaptor and Matchers)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interaction Verification (verify, ArgumentCaptor and...

Purpose: Proving `orderRepo.save(...)` was NEVER called when stock check failed and capturing the exact `Order` persisted on success.

What Topic Is Next → Next Topic 21.9: Full Context Integration Testing (@SpringBootTest)

21.9 Full Context Integration Testing (@SpringBootTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WebEnvironment.RANDOM_PORT` vs `MOCK` • `TestRestTemplate` / `RestTestClient` • `@ActiveProfiles("test")` • Minimizing context reloads across test classes via shared base config • Production Edge-Case & Pitfall Guard (Full Context Integration Testing (@SpringBootTest)) • Hands-On Java Implementation & Verification (Full Context Integration Testing (@SpringBootTest)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Full Context Integration Testing (@SpringBootTest)

Purpose: Verifying full end-to-end wiring from HTTP FilterChain -> Controller -> Transactional Service -> JPA -> Database.

What Topic Is Next → Next Topic 21.10: Web Layer Testing with MockMvc

21.10 Web Layer Testing with MockMvc [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mockMvc.perform(post(...).contentType(APPLICATION_JSON).content(...))` • `status()` • `header()` • `jsonPath("$.id")` • Production Edge-Case & Pitfall Guard (Web Layer Testing with MockMvc) • Hands-On Java Implementation & Verification (Web Layer Testing with MockMvc) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Web Layer Testing with MockMvc

Purpose: Asserting exact HTTP status codes (`201`, `400`, `401`, `403`, `404`, `409`), `Location` headers, and JSON field contracts without opening a TCP port.

What Topic Is Next → Next Topic 21.11: Repository Slice Testing (@DataJpaTest)

21.11 Repository Slice Testing (@DataJpaTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Focused JPA/Hibernate slice • Automatic per-test transaction rollback • `TestEntityManager` (`persistAndFlush` + `clear` to test real DB reads rather than L1 cache!) • Production Edge-Case & Pitfall Guard (Repository Slice Testing (@DataJpaTest)) • Hands-On Java Implementation & Verification (Repository Slice Testing (@DataJpaTest)) • Repository Slice Testing (@DataJpaTest) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repository Slice Testing (@DataJpaTest)

Purpose: Verifying custom `@Query` JPQL, unique constraints, Specifications, and pagination queries fast and cleanly.

What Topic Is Next → Next Topic 21.12: Controller Slice Testing (@WebMvcTest)

21.12 Controller Slice Testing (@WebMvcTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading ONLY web layer (`@RestController` • `@RestControllerAdvice` • Jackson • Bean Validation • Production Edge-Case & Pitfall Guard (Controller Slice Testing (@WebMvcTest)) • Hands-On Java Implementation & Verification (Controller Slice Testing (@WebMvcTest)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Controller Slice Testing (@WebMvcTest)

Purpose: Testing all 25 HTTP endpoints, request validations, and role guards in seconds with zero database startup.

What Topic Is Next → Next Topic 21.13: Database and API Integration Testing

21.13 Database and API Integration Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end HTTP -> Service -> JPA -> PostgreSQL flow verification • Test data isolation (`@Sql` / clean setup) • Testing real transaction rollbacks & optimistic locks • Production Edge-Case & Pitfall Guard (Database and API Integration Testing) • Hands-On Java Implementation & Verification (Database and API Integration Testing) • Database and API Integration Testing — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Verifying that a failed checkout truly leaves `orders` empty and `products.stock` untouched in PostgreSQL.

What Topic Is Next → Next Topic 21.14: Testcontainers (Real PostgreSQL & Redis in Docker Tests)

21.14 Testcontainers (Real PostgreSQL & Redis in Docker Tests) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `org.testcontainers:postgresql` • `@ServiceConnection` (Spring Boot 3.1+ zero-boilerplate wiring!) • Catching PostgreSQL-specific SQL/JSONB/constraint bugs that H2 misses • Production Edge-Case & Pitfall Guard (Testcontainers (Real PostgreSQL & Redis in Docker Tests)) • Hands-On Java Implementation & Verification (Testcontainers (Real PostgreSQL & Redis in Docker Tests)) • Testcontainers (Real PostgreSQL & Redis in Docker Tests) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testcontainers (Real PostgreSQL & Redis in Docker Te...

Purpose: Running CI integration tests against a real ephemeral `postgres:16-alpine` container via `@ServiceConnection`!

What Topic Is Next → Next Topic 21.15: Automated API Contract Testing with Postman / Newman

21.15 Automated API Contract Testing with Postman / Newman [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chaining Postman requests (Register -> Login capture `token` -> Create Product capture `id` -> Place Order -> Assert Stock) • CLI execution via Newman in CI • Production Edge-Case & Pitfall Guard (Automated API Contract Testing with Postman / Newman) • Hands-On Java Implementation & Verification (Automated API Contract Testing with Postman / Newman) • Automated API Contract Testing with Postman / Newman — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automated API Contract Testing with Postman / Newman • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Shipping a self-verifying `postman_collection.json` alongside your Capstone repository.

What Topic Is Next → Next Topic 21.16: OpenAPI & Swagger Contract Verification

21.16 OpenAPI & Swagger Contract Verification [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying `/v3/api-docs` generates valid OpenAPI 3.0/3.1 JSON without schema conflicts • Documenting `400`/`401`/`403`/`404` responses accurately • Production Edge-Case & Pitfall Guard (OpenAPI & Swagger Contract Verification) • Hands-On Java Implementation & Verification (OpenAPI & Swagger Contract Verification) • OpenAPI & Swagger Contract Verification — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OpenAPI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Adding a fast `MockMvc` test asserting `GET /v3/api-docs` returns `200 OK` so broken Swagger annotations never break runtime docs.

What Topic Is Next → Next Topic 21.17: Clean Code, SOLID Principles and Maintainability

21.17 Clean Code, SOLID Principles and Maintainability [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single Responsibility • Dependency Inversion • Small pure functions • Guard clauses • Production Edge-Case & Pitfall Guard (Clean Code, SOLID Principles and Maintainability) • Hands-On Java Implementation & Verification (Clean Code, SOLID Principles and Maintainability) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Clean Code

Purpose: Keeping backend codebases easy to extend, review, and explain clearly during live code walkthrough interviews.

What Topic Is Next → Next Topic 21.18: Code Coverage with JaCoCo

21.18 Code Coverage with JaCoCo [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `jacoco-maven-plugin` • Line vs branch coverage (`if/else` paths!) • Excluding auto-generated DTOs/Application main • Why meaningful assertions beat vanity 100% getter tests • Production Edge-Case & Pitfall Guard (Code Coverage with JaCoCo) • Hands-On Java Implementation & Verification (Code Coverage with JaCoCo) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Code Coverage with JaCoCo

Purpose: Generating `target/site/jacoco/index.html` in Maven to spot untested error/security branches in `@Service` classes.

What Topic Is Next → Next Topic 21.19: Regression Testing Discipline (Bug -> Failing Test -> Fix)

21.19 Regression Testing Discipline (Bug -> Failing Test -> Fix) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reproducing every reported bug as a failing automated test FIRST • Fixing the code until green • Keeping that regression guard in CI permanently • Production Edge-Case & Pitfall Guard (Regression Testing Discipline (Bug -> Failing Test -> Fix)) • Hands-On Java Implementation & Verification (Regression Testing Discipline (Bug -> Failing Test -> Fix)) • Regression Testing Discipline (Bug -> Failing Test -> Fix) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Regression Testing Discipline (Bug -> Failing Test -...

Purpose: Ensuring fixed production edge cases (e.g., rounding error or expired-coupon bypass) can NEVER silently return months later.

What Topic Is Next → Module 21 Capstone (Week 21 Lab Build: Full-Pyramid JUnit 5, Mockito & Testcontainers Suite) → Next Topic 22.1: Linux Architecture and Server Distributions

Module 21 Hands-On Capstone Project (100 Marks): Week 21 Lab Build: Full-Pyramid JUnit 5, Mockito & Testcontainers Suite — Verify a Spring Boot backend with >= 75% service coverage using JUnit 5 `@ParameterizedTest`, Mockito unit tests, `@WebMvcTest` + `MockMvc` security/validation slice tests, `@DataJpaTest`, a...

MODULE 22 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 22: Docker, Linux and CI/CD Fundamentals

Package, run, network, and automate Java 21 + Spring Boot 3 backends at a practical production-ready level — Linux CLI/permissions/processes/networking, multi-stage Java 21 Dockerfiles, container volumes & networks, multi-service Docker Compose stacks (Spring Boot + PostgreSQL + Redis + Kafka), Maven build lifecycles, GitHub Actions CI/CD pipelines, cloud deployment fundamentals, and secret hygiene.

Objectives: ✓ Navigate Linux servers confidently using essential filesystem, permission (`chmod`/`chown`), process (`ps`/`top`/`kill`), networking (`ss`/`curl`), and environment commands. • ✓ Author optimized, non-root, multi-stage Java 21 `Dockerfile`s (`maven:3.9-eclipse-temurin-21` build stage -> slim `eclipse-temurin:21-jre-alpine` runtime stage). • ✓ Manage container logs, volumes, bridge networks, and health-checked `docker-compose.yml` stacks running Spring Boot + PostgreSQL + Redis + Kafka. • ✓ Automate `mvn -B verify` + Docker image builds on every push/PR using GitHub Actions workflows. • ✓ Deploy a containerized Spring Boot backend cleanly with `prod` profiles, externalized environment variables, and strict secret management.

22.1 Linux Architecture and Server Distributions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Kernel vs shell/userland • Ubuntu/Debian vs RHEL/Rocky vs minimal Alpine containers • Filesystem hierarchy (`/etc` • `/var/log` • Production Edge-Case & Pitfall Guard (Linux Architecture and Server Distributions) • Hands-On Java Implementation & Verification (Linux Architecture and Server Distributions) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Architecture

Purpose: Understanding the Linux runtime environment inside 99% of production cloud VMs and Docker containers.

What Topic Is Next → Next Topic 22.2: Essential Linux Terminal Commands

22.2 Essential Linux Terminal Commands [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pwd` • `ls -la` • `cd` • `mkdir -p` • Production Edge-Case & Pitfall Guard (Essential Linux Terminal Commands) • Hands-On Java Implementation & Verification (Essential Linux Terminal Commands) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Essential Linux Terminal Commands

Purpose: Tailing live Spring Boot logs (`tail -f app.log | grep ERROR`) and searching config files over SSH.

What Topic Is Next → Next Topic 22.3: Linux File Permissions (chmod and chown)

22.3 Linux File Permissions (chmod and chown) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Owner / Group / Others (`rwx` = `4 • 1`) • `chmod 755` (`mvnw` executable bit!) • `chmod 600` (SSH keys & secret env files) • Production Edge-Case & Pitfall Guard (Linux File Permissions (chmod and chown)) • Hands-On Java Implementation & Verification (Linux File Permissions (chmod and chown)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux File Permissions (chmod and chown)

Purpose: Fixing `Permission denied: ./mvnw` in Linux CI via `chmod +x mvnw` (`git update-index --chmod=+x mvnw`) and running containers as non-root `appuser`.

What Topic Is Next → Next Topic 22.4: Linux Process Management (ps, top, jps, kill)

22.4 Linux Process Management (ps, top, jps, kill) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ps aux | grep java` • `top`/`htop` CPU & RSS memory • JDK `jps`/`jstack` • Graceful `kill -15` (`SIGTERM`) vs last-resort `kill -9` (`SIGKILL`) • Production Edge-Case & Pitfall Guard (Linux Process Management (ps, top, jps, kill)) • Hands-On Java Implementation & Verification (Linux Process Management (ps, top, jps, kill)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Process Management (ps, top, jps, kill)

Purpose: Sending `SIGTERM` (`kill -15`) so Spring Boot's graceful shutdown finishes active requests and closes HikariCP/Kafka cleanly.

What Topic Is Next → Next Topic 22.5: Linux Networking Basics (IP, Ports, DNS, HTTP, curl)

22.5 Linux Networking Basics (IP, Ports, DNS, HTTP, curl) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `127.0.0.1` loopback vs `0.0.0.0` all interfaces • Checking listening ports (`ss -tulpn` / `lsof -i :8080`) • DNS (`nslookup`/`dig`) • `curl -i -X POST` health & API debugging • Production Edge-Case & Pitfall Guard (Linux Networking Basics (IP, Ports, DNS, HTTP, curl)) • Hands-On Java Implementation &

Verification (Linux Networking Basics (IP, Ports, DNS, HTTP, curl)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Networking Basics (IP, Ports, DNS, HTTP, curl)

Purpose: Diagnosing `Address already in use: 8080` or `Connection refused` (binding `127.0.0.1` instead of `0.0.0.0` inside Docker!) in 10 seconds.

What Topic Is Next → Next Topic 22.6: Environment Variables in Linux & Containers

22.6 Environment Variables in Linux & Containers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `export KEY=VAL` • `env`/`printenv` • Spring Boot relaxed binding (`SPRING_DATASOURCE_URL` -> `spring.datasource.url`) • Production Edge-Case & Pitfall Guard (Environment Variables in Linux & Containers) • Hands-On Java Implementation & Verification (Environment Variables in Linux & Containers) • Environment Variables in Linux & Containers — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Variables in Linux

Purpose: Overriding any `application.yml` key in Docker/Linux simply by setting uppercase underscore environment variables!

What Topic Is Next → Next Topic 22.7: Docker and Containerization Fundamentals

22.7 Docker and Containerization Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Containers (shared Linux kernel + cgroups/namespaces isolation • MBs • Sub-second start) vs traditional VMs (full guest OS • GBs) • Production Edge-Case & Pitfall Guard (Docker and Containerization Fundamentals) • Hands-On Java Implementation & Verification (Docker and Containerization Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker

Purpose: Packaging Java 21 + exact OS libraries + JAR into an immutable image that runs identically locally, in CI, and in cloud prod.

What Topic Is Next → Next Topic 22.8: Docker Images and Layer Caching

22.8 Docker Images and Layer Caching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-only union filesystem layers • Tags (`eclipse-temurin:21-jre-alpine` vs mutable `:latest`) • Layer cache invalidation order • `.dockerignore`! • Production Edge-Case & Pitfall Guard (Docker Images and Layer Caching) • Hands-On Java Implementation & Verification (Docker Images and Layer Caching) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Images

Purpose: Copying `pom.xml` and downloading Maven dependencies in a cached layer BEFORE `COPY src ./src` so code-only rebuilds finish in seconds!

What Topic Is Next → Next Topic 22.9: Running and Managing Containers

22.9 Running and Managing Containers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Port publishing `-p 8080:8080` (`host:container`) • Detached `-d` • `--name` • `-e` / `--env-file` • Production Edge-Case & Pitfall Guard (Running and Managing Containers) • Hands-On Java Implementation & Verification (Running and Managing Containers) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Running

Purpose: Mapping host port `8081` to container port `8080` (`-p 8081:8080`) cleanly without rebuilding the JAR.

What Topic Is Next → Next Topic 22.10: Authoring Production Java 21 Multi-Stage Dockerfiles

22.10 Authoring Production Java 21 Multi-Stage Dockerfiles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stage 1 (`maven:3.9-eclipse-temurin-21` compiles JAR) -> Stage 2 (`eclipse-temurin:21-jre-alpine` copies only JAR • Runs as non-root `USER` • `ENTRYPOINT ["java" • "-XX:MaxRAMPercentage=75.0" • Production Edge-Case & Pitfall Guard (Authoring Production Java 21 Multi-Stage Dockerfiles) • Hands-On Java Implementation & Verification (Authoring Production Java 21 Multi-Stage Dockerfiles) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authoring Production Java 21 Multi-Stage Dockerfiles

Purpose: Shrinking image size from ~750MB (JDK + Maven cache) down to ~190MB (slim JRE + JAR) with smaller attack surface and proper `SIGTERM` handling!

What Topic Is Next → Next Topic 22.11: Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)

22.11 Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `docker build -t` • `docker ps -a` • `docker logs -f --tail 100` • `docker exec -it <container> sh` • Production Edge-Case & Pitfall Guard (Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)) • Hands-On Java Implementation & Verification (Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Essential Docker CLI Commands (build, run, ps, logs,...

Purpose: Inspecting startup failure logs (`docker logs`) and container memory usage (`docker stats`) rapidly.

What Topic Is Next → Next Topic 22.12: Docker Volumes and Persistent Database Storage

22.12 Docker Volumes and Persistent Database Storage [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why container writable layers vanish on `docker rm` • Named volumes (`pg_data:/var/lib/postgresql/data`) vs bind mounts • Production Edge-Case & Pitfall Guard (Docker Volumes and Persistent Database Storage) • Hands-On Java Implementation & Verification (Docker Volumes and Persistent Database Storage) • Docker Volumes and Persistent Database Storage — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Volumes • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Persisting PostgreSQL tables and Kafka logs across container restarts and image upgrades via named Docker volumes.

What Topic Is Next → Next Topic 22.13: Docker Container Networking

22.13 Docker Container Networking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: User-defined bridge networks • Built-in Docker DNS resolution by service/container name (`jdbc:postgresql://postgres:5432/jobportal` — NOT `localhost`!) • Production Edge-Case & Pitfall Guard (Docker Container Networking) • Hands-On Java Implementation & Verification (Docker Container Networking) • Docker Container Networking — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts &

non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Container Networking • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Connecting `order-service` to `postgres:5432`, `redis:6379`, and `kafka:9092` via Docker DNS service names.

What Topic Is Next → Next Topic 22.14: Docker Compose for Multi-Container Applications

22.14 Docker Compose for Multi-Container Applications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `compose.yaml` (`services` • `networks` • `volumes` • `environment` • Production Edge-Case & Pitfall Guard (Docker Compose for Multi-Container Applications) • Hands-On Java Implementation & Verification (Docker Compose for Multi-Container Applications) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Compose for Multi-Container Applications

Purpose: Using `depends_on` with `condition: service_healthy` (`pg_isready`) so Spring Boot waits until PostgreSQL is genuinely accepting SQL connections before booting!

What Topic Is Next → Next Topic 22.15: Containerized PostgreSQL Setup and Init Scripts

22.15 Containerized PostgreSQL Setup and Init Scripts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `postgres:16-alpine` • `POSTGRES_DB`/`USER`/`PASSWORD` • `pg_isready` healthcheck • `/docker-entrypoint-initdb.d/` multi-database bootstrap SQL • Production Edge-Case & Pitfall Guard (Containerized PostgreSQL Setup and Init Scripts) • Hands-On Java Implementation & Verification (Containerized PostgreSQL Setup and Init Scripts) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Containerized PostgreSQL Setup

Purpose: Initializing isolated databases (`user_db`, `job_db`, `application_db`) cleanly inside one local dev PostgreSQL container.

What Topic Is Next → Next Topic 22.16: Multi-Service Spring Boot + Kafka + Redis Compose Stack

22.16 Multi-Service Spring Boot + Kafka + Redis Compose Stack [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Orchestrating Eureka • Gateway • Microservices • PostgreSQL • Production Edge-Case & Pitfall Guard (Multi-Service Spring Boot + Kafka + Redis Compose Stack) • Hands-On Java Implementation & Verification (Multi-Service Spring Boot + Kafka + Redis Compose Stack) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Service Spring Boot + Kafka + Redis Compose St...

Purpose: Booting the complete Final Capstone architecture locally with a single `docker compose up --build` command.

What Topic Is Next → Next Topic 22.17: CI/CD Fundamentals (Continuous Integration & Continuous Delivery)

22.17 CI/CD Fundamentals (Continuous Integration & Continuous Delivery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automated build + test + lint + container build on every PR (CI) vs automated staging/production release deployment (CD) • Fast feedback loops • Production Edge-Case & Pitfall Guard (CI/CD Fundamentals (Continuous Integration & Continuous Delivery)) • Hands-On Java Implementation & Verification (CI/CD Fundamentals (Continuous Integration & Continuous Delivery)) • CI/CD Fundamentals (Continuous Integration & Continuous Delivery) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CI/CD Fundamentals (Continuous Integration & Continu... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Blocking broken code or failing JUnit tests from ever merging into `main`.

What Topic Is Next → Next Topic 22.18: GitHub Actions Workflow Fundamentals

22.18 GitHub Actions Workflow Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.github/workflows/ci.yml` • Triggers (`on: [push • Pull_request]`) • Ubuntu runners • Production Edge-Case & Pitfall Guard (GitHub Actions Workflow Fundamentals) • Hands-On Java Implementation & Verification (GitHub Actions Workflow Fundamentals) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Actions Workflow Fundamentals

Purpose: Getting free, automatic green/red PR status checks on every student and team GitHub repository.

What Topic Is Next → Next Topic 22.19: Automated Build, Test and Package Pipeline

22.19 Automated Build, Test and Package Pipeline [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `./mvnw -B verify` in GitHub Actions (Docker is pre-installed on GitHub Ubuntu runners so Testcontainers works out of the box!) • Uploading JaCoCo/Surefire artifacts • Building Docker image • Production Edge-Case & Pitfall Guard (Automated Build, Test and Package Pipeline) • Hands-On Java Implementation & Verification (Automated Build, Test and Package Pipeline) • Automated Build, Test and Package Pipeline — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automated Build

Purpose: Executing unit tests,PostgreSQL Testcontainers integration tests, JAR packaging, and `docker build` validation on every Pull Request.

What Topic Is Next → Next Topic 22.20: Maven Build Lifecycle Mastery (clean, compile, test, package, verify)

22.20 Maven Build Lifecycle Mastery (clean, compile, test, package, verify) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standard Maven phases (`validate -> compile -> test -> package -> integration-test -> verify -> install`) • Surefire (`*Test`) vs Failsafe (`*IT`) • Multi-module parent `pom.xml` basics • Production Edge-Case & Pitfall Guard (Maven Build Lifecycle Mastery (clean, compile, test, package, verify)) • Hands-On Java Implementation & Verification (Maven Build Lifecycle Mastery (clean, compile, test, package, verify)) • Maven Build Lifecycle Mastery (clean, compile, test, package, verify) — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Maven Build Lifecycle Mastery (clean, compile, test,...

Purpose: Sharing common Java 21, Spring Boot 3, and Spring Cloud versions cleanly across microservices via a root parent `pom.xml`.

What Topic Is Next → Next Topic 22.21: Deploying a Spring Boot Backend Application

22.21 Deploying a Spring Boot Backend Application [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deploying containerized Spring Boot + managed PostgreSQL/Redis to a cloud VM / container platform (Render/Railway/VPS) • Reverse proxy TLS (Nginx/Caddy) • `/actuator/health` readiness probes • Production Edge-Case & Pitfall Guard (Deploying a Spring Boot Backend Application) • Hands-On Java Implementation & Verification (Deploying a Spring Boot Backend Application) • Deploying a Spring Boot Backend Application — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deploying a Spring Boot Backend Application

Purpose: Publishing a live HTTPS URL + Swagger UI for your Capstone so interviewers can test your live API during technical rounds.

What Topic Is Next → Next Topic 22.22: Production Profiles and Environment Configuration

22.22 Production Profiles and Environment Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SPRING_PROFILES_ACTIVE=prod` • Disabling `show-sql` & verbose debug logs • Setting `ddl-auto: validate` • Configuring CORS allowed frontend origins • Production Edge-Case & Pitfall Guard (Production Profiles and Environment Configuration) • Hands-On Java Implementation & Verification (Production Profiles and Environment Configuration) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production Profiles

Purpose: Ensuring production boot uses validated schemas, structured logs, and bounded heap (`-XX:MaxRAMPercentage=75.0`).

What Topic Is Next → Next Topic 22.23: Secrets Management and Secure Configuration Practices

22.23 Secrets Management and Secure Configuration Practices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: GitHub Actions Encrypted Secrets (`${{ secrets.DB_PASSWORD }}`) • Docker Compose `.env` + `.env.example` • Rotating leaked keys immediately • Never baking secrets into `Dockerfile` `ENV`/`ARG` layers! • Production Edge-Case & Pitfall Guard (Secrets Management and Secure Configuration Practices) • Hands-On Java Implementation & Verification (Secrets Management and Secure Configuration Practices) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secrets Management

Purpose: Injecting `JWT_SECRET`, `DB_PASSWORD`, and registry tokens at runtime only so `docker inspect` / image layers never leak credentials.

What Topic Is Next → Module 22 Capstone (Week 22 Lab Build: Multi-Stage Docker Compose & GitHub Actions CI Pipeline) → Next Topic 23.1: Arrays Interview Revision (Two Pointers, Sliding Window, Prefix Sum)

Module 22 Hands-On Capstone Project (100 Marks): Week 22 Lab Build: Multi-Stage Docker Compose & GitHub Actions CI Pipeline — Containerize a Spring Boot 3 + PostgreSQL + Redis + Kafka stack using a non-root multi-stage Java 21 `Dockerfile`, health-checked `docker-compose.yml`, and automated `.github/workflows/ci.ym...

MODULE 23 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 23: DSA Revision, Coding Tests and Technical Interviews

Consolidate 6 months of continuous DSA problem-solving (100–150 curated problems) and full-stack Java Backend interview mastery — Arrays, Strings, HashMap, Linked List, Stack, Queue, Binary Search on Answer, Sorting, Recursion/Backtracking, Binary Trees & BSTs, Traversals, Graphs (BFS/DFS), Dynamic Programming (Memoization & Tabulation), Complexity Walkthroughs, and structured Core Java, SQL, Spring Boot, Microservices, and Project Architecture mock interviews.

Objectives: ✓ Identify optimal DSA patterns within 2 minutes of reading a coding assessment problem statement and constraints. • ✓ Implement bug-free Java 21 solutions for Arrays, Strings, Hashing, Linked Lists, Monotonic Stacks/Deques, Answer-Space Binary Search, Trees/BSTs, Graph BFS/DFS, and introductory Dynamic Programming. • ✓ Articulate worst-case Time and Auxiliary Space complexity clearly aloud while coding. • ✓ Answer deep-dive technical interview questions across Core Java/JVM/Concurrency, SQL/Indexing/ACID, Spring Boot/JPA/Security, and Microservices/Kafka/Redis with concrete production examples. • ✓ Walk interviewers through your own GitHub project architecture, schema trade-offs, failure handling, and code line-by-line.

23.1 Arrays Interview Revision (Two Pointers, Sliding Window, Prefix Sum) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Product of Array Except Self • Trapping Rain Water • Subarray Sum Equals K • Merge Overlapping Intervals • Production Edge-Case & Pitfall Guard (Arrays Interview Revision (Two Pointers, Sliding Window, Prefix Sum)) • Hands-On Java Implementation & Verification (Arrays Interview Revision (Two Pointers, Sliding Window, Prefix Sum)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Arrays Interview Revision (Two Pointers, Sliding Win...

Purpose: Cracking the highest-frequency Round-1 online coding screening patterns in Java.

What Topic Is Next → Next Topic 23.2: Strings Interview Revision (Frequency, Palindrome, Anagrams, Patterns)

23.2 Strings Interview Revision (Frequency, Palindrome, Anagrams, Patterns) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Longest Palindromic Substring (expand around center) • Minimum Window Substring • Find All Anagrams in a String (`int[26]` window) • String Compression • Production Edge-Case & Pitfall Guard (Strings Interview Revision (Frequency, Palindrome, Anagrams, Patterns)) • Hands-On Java Implementation & Verification (Strings Interview Revision (Frequency, Palindrome, Anagrams, Patterns)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strings Interview Revision (Frequency, Palindrome, A...

Purpose: Using fixed `int[26]` / `int[128]` frequency arrays for fast `O(1)`-overhead character window matching.

What Topic Is Next → Next Topic 23.3: HashMap Lookup & Frequency Interview Problems

23.3 HashMap Lookup & Frequency Interview Problems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: LRU Cache (`HashMap` + custom Doubly Linked List in `O(1)`!) • Top K Frequent Elements • Subarray Sums Divisible by K • Copy List with Random Pointer • Production Edge-Case & Pitfall Guard (HashMap Lookup & Frequency Interview Problems) • Hands-On Java Implementation & Verification (HashMap Lookup & Frequency Interview Problems) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HashMap Lookup

Purpose: Implementing a thread-clear `O(1)` `LRUCache` from scratch — a classic product-company Java coding round favorite!

What Topic Is Next → Next Topic 23.4: Linked List Interview Patterns (Reversal, Cycle Entry, Merge)

23.4 Linked List Interview Patterns (Reversal, Cycle Entry, Merge) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reverse Nodes in k-Group • Detect Cycle Entry Node (Floyd phase 2) • Merge Two/K Sorted Lists (`PriorityQueue`) • Remove Nth Node From End • Production Edge-Case & Pitfall Guard (Linked List Interview Patterns (Reversal, Cycle Entry, Merge)) • Hands-On Java Implementation & Verification (Linked List Interview Patterns (Reversal, Cycle Entry, Merge)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linked List Interview Patterns (Reversal, Cycle Entr...

Purpose: Demonstrating rock-solid pointer invariant handling without NullPointerExceptions on 0-node or 1-node edge cases.

What Topic Is Next → Next Topic 23.5: Stack Interview Problems (Monotonic Stack & Expressions)

23.5 Stack Interview Problems (Monotonic Stack & Expressions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Daily Temperatures • Next Greater Element II (circular) • Largest Rectangle in Histogram • Evaluate Reverse Polish Notation • Production Edge-Case & Pitfall Guard (Stack Interview Problems (Monotonic Stack & Expressions)) • Hands-On Java Implementation & Verification (Stack Interview Problems (Monotonic Stack & Expressions)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stack Interview Problems (Monotonic Stack & Expressi...

Purpose: Solving next-warmer-day / stock-span / histogram area problems in `O(n)` using a monotonic index `ArrayDeque<Integer>`.

What Topic Is Next → Next Topic 23.6: Queue & Monotonic Deque Interview Problems

23.6 Queue & Monotonic Deque Interview Problems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sliding Window Maximum in `O(n)` (`ArrayDeque`) • Rotten Oranges multi-source BFS queue • First Non-Repeating Character in a Stream • Production Edge-Case & Pitfall Guard (Queue & Monotonic Deque Interview Problems) • Hands-On Java Implementation & Verification (Queue & Monotonic Deque Interview Problems) • Queue & Monotonic Deque Interview Problems — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Queue

Purpose: Maintaining rolling window maximums in `O(1)` amortized time per element.

What Topic Is Next → Next Topic 23.7: Binary Search Variations & Binary Search on Answer Space

23.7 Binary Search Variations & Binary Search on Answer Space [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Search in Rotated Sorted Array • Find Peak Element • Binary Search on Monotonic Answer Space (Koko Eating Bananas / Ship Packages Within D Days) • Production Edge-Case & Pitfall Guard (Binary Search Variations & Binary Search on Answer Space) • Hands-On Java Implementation & Verification (Binary Search Variations & Binary Search on Answer Space) • Binary Search Variations & Binary Search on Answer Space — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Binary Search Variations

Purpose: Recognizing `"Minimize the maximum capacity/speed such that feasible(mid) is true"` as Binary Search on `[low..high]` answer space!

What Topic Is Next → Next Topic 23.8: Sorting-Based Interview Problems

23.8 Sorting-Based Interview Problems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Meeting Rooms II (min conference rooms via Sweep Line / Min-Heap) • Non-Overlapping Intervals • Custom Comparator Largest Number • Kth Largest (`PriorityQueue` / Quickselect) • Production Edge-Case & Pitfall Guard (Sorting-Based Interview Problems) • Hands-On Java Implementation & Verification (Sorting-Based Interview Problems) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sorting-Based Interview Problems

Purpose: Solving interval scheduling and Top-K selection problems cleanly.

What Topic Is Next → Next Topic 23.9: Recursion & Backtracking Interview Problems

23.9 Recursion & Backtracking Interview Problems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generate Parentheses • Combination Sum • Word Search on 2D Grid (in-place visited mark + restore) • N-Queens pruning • Production Edge-Case & Pitfall Guard (Recursion & Backtracking Interview Problems) • Hands-On Java Implementation & Verification (Recursion & Backtracking Interview Problems) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Recursion

Purpose: Pruning invalid state-space branches early (e.g., `openCount < n` and `closeCount < openCount` in Generate Parentheses).

What Topic Is Next → Next Topic 23.10: Binary Trees and Binary Search Trees (BST)

23.10 Binary Trees and Binary Search Trees (BST) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Max Depth • Diameter of Binary Tree • Invert Tree • Lowest Common Ancestor (LCA) • Production Edge-Case & Pitfall Guard (Binary Trees and Binary Search Trees (BST)) • Hands-On Java Implementation & Verification (Binary Trees and Binary Search Trees (BST)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Binary Trees

Purpose: Validating BST global range constraints rather than only checking immediate `left.val < node.val < right.val`.

What Topic Is Next → Next Topic 23.11: Tree Traversals (Inorder, Preorder, Postorder & Level-Order BFS)

23.11 Tree Traversals (Inorder, Preorder, Postorder & Level-Order BFS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recursive & iterative stack traversals • Level-Order BFS (`Queue`) • Right-Side View • Zigzag Level Order • Production Edge-Case & Pitfall Guard (Tree Traversals (Inorder, Preorder, Postorder & Level-Order BFS)) • Hands-On Java Implementation & Verification (Tree Traversals (Inorder, Preorder, Postorder & Level-Order BFS)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tree Traversals (Inorder, Preorder, Postorder & Leve...

Purpose: Using `int levelSize = queue.size()` snapshot per BFS level to process tree levels cleanly.

What Topic Is Next → Next Topic 23.12: Graph Algorithms (BFS, DFS, Cycle Detection & Topological Sort)

23.12 Graph Algorithms (BFS, DFS, Cycle Detection & Topological Sort) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Adjacency List `List<List<Integer>>` • Number of Islands • Shortest Unweighted Path (BFS) • Course Schedule prerequisites cycle check (Kahn's indegree BFS / 3-color DFS) • Production Edge-Case & Pitfall Guard (Graph Algorithms (BFS, DFS, Cycle Detection & Topological Sort)) • Hands-On Java Implementation & Verification (Graph Algorithms (BFS, DFS, Cycle Detection & Topological Sort)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graph Algorithms (BFS, DFS, Cycle Detection & Topolo...

Purpose: Modeling task dependency graphs (Course Schedule / Maven build order) via Kahn's Topological Sort!

What Topic Is Next → Next Topic 23.13: Dynamic Programming (Memoization & Tabulation Fundamentals)

23.13 Dynamic Programming (Memoization & Tabulation Fundamentals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Overlapping subproblems + optimal substructure • Top-Down Memoization (`Integer[] memo`) -> Bottom-Up Tabulation -> Space Optimization: Climbing Stairs • House Robber • Coin Change • Production Edge-Case & Pitfall Guard (Dynamic Programming (Memoization & Tabulation Fundamentals)) • Hands-On Java Implementation & Verification (Dynamic Programming (Memoization & Tabulation Fundamentals)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Programming (Memoization & Tabulation Fundam...

Purpose: Converting exponential `O(2^n)` recursion into `O(n)` / `O(n*amount)` polynomial DP systematically.

What Topic Is Next → Next Topic 23.14: Complexity Analysis Walkthroughs in Live Interviews

23.14 Complexity Analysis Walkthroughs in Live Interviews [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deriving tight bounds for Heapify (`O(n)`) • Backtracking (`O(n * 2^n)`) • Graph (`O(V + E)`) • DP state count x transition cost • Production Edge-Case & Pitfall Guard (Complexity Analysis Walkthroughs in Live Interviews) • Hands-On Java Implementation & Verification (Complexity Analysis Walkthroughs in Live Interviews) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Complexity Analysis Walkthroughs in Live Interviews

Purpose: Ending every interview coding solution with a crisp, accurate 30-second Time & Space justification.

What Topic Is Next → Next Topic 23.15: Core Java, JVM & Concurrency Interview Deep-Dive

23.15 Core Java, JVM & Concurrency Interview Deep-Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: String Pool • `equals`/`hashCode` & `HashMap` Java 8+ treeification • `ConcurrentHashMap` vs `Hashtable` • `volatile` vs `AtomicInteger` vs `synchronized` • Production Edge-Case & Pitfall Guard (Core Java, JVM & Concurrency Interview Deep-Dive) • Hands-On Java Implementation & Verification (Core Java, JVM & Concurrency Interview Deep-Dive) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Java

Purpose: Answering Round-1/Round-2 Core Java backend engineering questions with internal JDK 21 precision.

What Topic Is Next → Next Topic 23.16: Database, SQL, Indexing & JPA Interview Deep-Dive

23.16 Database, SQL, Indexing & JPA Interview Deep-Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composite B-Tree leftmost prefix • `LEFT JOIN` `ON` vs `WHERE` • `NOT IN` NULL trap • Window Functions • Production Edge-Case & Pitfall Guard (Database, SQL, Indexing & JPA Interview Deep-Dive) • Hands-On Java Implementation & Verification (Database, SQL, Indexing & JPA Interview Deep-Dive) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Writing live SQL queries (`Top 3 per department`, `2nd highest salary`, `unmatched records`) and diagnosing Hibernate SQL issues.

What Topic Is Next → Next Topic 23.17: Spring Boot, Data JPA & Security Interview Deep-Dive

23.17 Spring Boot, Data JPA & Security Interview Deep-Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-configuration `@Conditional` mechanics • Bean scopes & AOP proxy self-invocation trap • `@Transactional` propagation (`REQUIRES_NEW`) & rollback rules • `SecurityFilterChain` • Production Edge-Case & Pitfall Guard (Spring Boot, Data JPA & Security Interview Deep-Dive) • Hands-On Java Implementation & Verification (Spring Boot, Data JPA & Security Interview Deep-Dive) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spring Boot

Purpose: Demonstrating production Spring Boot 3 maturity rather than annotation memorization.

What Topic Is Next → Next Topic 23.18: Microservices, Spring Cloud, Kafka & Redis Interview Deep-Dive

23.18 Microservices, Spring Cloud, Kafka & Redis Interview Deep-Dive [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Gateway + Eureka flow • Resilience4j CircuitBreaker states • Saga compensating transactions • Kafka consumer groups/offsets/ordering/DLQ/Outbox • Production Edge-Case & Pitfall Guard (Microservices, Spring Cloud, Kafka & Redis Interview Deep-Dive) • Hands-On Java Implementation & Verification (Microservices, Spring Cloud, Kafka & Redis Interview Deep-Dive) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microservices

Purpose: Designing resilient distributed workflows on a virtual whiteboard during backend system-design rounds.

What Topic Is Next → Next Topic 23.19: Project Architecture Defense & Code Walkthrough Mastery

23.19 Project Architecture Defense & Code Walkthrough Mastery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring a crisp 3-minute architecture pitch • Defending PostgreSQL schema/indexes • Showing JWT/RBAC • Walking through Kafka/Redis/Resilience4j flows • Production Edge-Case & Pitfall Guard (Project Architecture Defense & Code Walkthrough Mastery) • Hands-On Java Implementation & Verification (Project Architecture Defense & Code Walkthrough Mastery) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Project Architecture Defense

Purpose: Turning the final 20 minutes of every interview ("Tell me about a backend project you built") into your strongest hiring signal.

What Topic Is Next → Module 23 Capstone (Week 23 Lab Build: 120-Problem DSA Portfolio & Architecture Defense Pack) → Next Topic 24.1: Job Portal Domain Modeling, Bounded Contexts & ER Schema Design

Module 23 Hands-On Capstone Project (100 Marks): Week 23 Lab Build: 120-Problem DSA Portfolio & Architecture Defense Pack — Complete the curated Java 21 DSA problem repository with documented Time/Space complexities (`Arrays` through `DP`) and deliver a recorded 10-minute system architecture & code walkthrough de...

MODULE 24 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 10 TOPICS (80 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 24: Final Industry Capstone Project and Placement Readiness

Design, implement, test, containerize, document, deploy, and defend the Final Industry Capstone — a distributed Job Portal Backend System built with Java 21, Spring Boot 3, Spring Security/JWT, Spring Data JPA, PostgreSQL, Spring Cloud Gateway, Eureka, Apache Kafka, Redis, JUnit 5/Mockito, Swagger/OpenAPI, and Docker Compose.

Objectives: ✓ Architect and build all core business modules of the Job Portal Backend System: Auth/User, Candidate, Employer, Job, Application, and Notification services. • ✓ Implement JWT authentication, RBAC (`CANDIDATE`, `EMPLOYER`, `ADMIN`), ownership guards, and dynamic paginated/cached Job Search APIs. • ✓ Connect asynchronous application lifecycle notifications (`ApplicationSubmitted`, `ApplicationShortlisted`, `InterviewScheduled`) over Apache Kafka with idempotency and DLQ. • ✓ Package the complete system with JUnit 5/Mockito/Testcontainers tests, OpenAPI/Swagger docs, Postman collection, ER & Architecture diagrams, and `docker-compose.yml`. • ✓ Deploy and deliver a live end-to-end Capstone demonstration and technical viva defense for placement readiness.

24.1 Job Portal Domain Modeling, Bounded Contexts & ER Schema Design

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing schemas • Constraints • Composite indexes for Users • Candidates (skills/resume metadata) • Production Edge-Case & Pitfall Guard (Job Portal Domain Modeling, Bounded Contexts & ER Schema Design) • Hands-On Java Implementation & Verification (Job Portal Domain Modeling, Bounded Contexts & ER Schema Design) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Job Portal Domain Modeling

Purpose: Preventing duplicate job applications at the database level via `UNIQUE (job_id, candidate_id)` and indexing `(status, location, min_salary)`.

What Topic Is Next → Next Topic 24.2: User & Authentication Service (Registration, Login, JWT & RBAC)

24.2 User & Authentication Service (Registration, Login, JWT & RBAC)

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: BCrypt registration for `CANDIDATE` • `EMPLOYER` accounts • JWT access/refresh token issuance • Account status checks • Production Edge-Case & Pitfall Guard (User & Authentication Service (Registration, Login, JWT & RBAC)) • Hands-On Java Implementation & Verification (User & Authentication Service (Registration, Login, JWT & RBAC)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User

Purpose: Enforcing strict role separation so only `EMPLOYER`s post jobs and only `CANDIDATE`s submit applications.

What Topic Is Next → Next Topic 24.3: Candidate & Employer Profile Services

24.3 Candidate & Employer Profile Services

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Candidate profile • Education • Experience years • Skill tags & resume metadata • Production Edge-Case & Pitfall Guard (Candidate & Employer Profile Services) • Hands-On Java Implementation & Verification (Candidate & Employer Profile Services) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Candidate

Purpose: Ensuring Employer A can update ONLY Employer A's company profile and job postings.

What Topic Is Next → Next Topic 24.4: Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)

24.4 Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Job posting • Status lifecycle (`DRAFT` • `OPEN` • `CLOSED`) • Production Edge-Case & Pitfall Guard (Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)) • Hands-On Java Implementation & Verification (Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Job Service

Purpose: Supporting `GET /api/v1/jobs?q=java&location=Bengaluru&workMode=HYBRID&minSalary=600000&page=0&size=20&sort=newest`.

What Topic Is Next → Next Topic 24.5: Application Service & Status Workflow

24.5 Application Service & Status Workflow

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Applying to an `OPEN` job (`Idempotency-Key` + `UNIQUE(job_id • Candidate_id)`) • Tracking states (`APPLIED -> UNDER_REVIEW -> SHORTLISTED -> INTERVIEW -> OFFERED / REJECTED`) • Employer vs candidate views • Production Edge-Case & Pitfall Guard (Application Service & Status Workflow) • Hands-On Java Implementation & Verification (Application Service & Status Workflow) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Service

Purpose: Validating via `job-service` (OpenFeign + Resilience4j) that a job is `OPEN` before accepting a candidate's application.

What Topic Is Next → Next Topic 24.6: Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)

24.6 Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Publishing `ApplicationSubmittedEvent` & `ApplicationStatusChangedEvent` (keyed by `applicationId`) • `notification-service` `@KafkaListener` with idempotency table • `@RetryableTopic` • `.DLT` • Production Edge-Case & Pitfall Guard (Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)) • Hands-On Java Implementation & Verification (Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for

Kafka Notification Pipeline (Asynchronous Events, Re...

Purpose: Notifying employers on new applications and candidates on shortlist/interview updates asynchronously without slowing down API responses.

What Topic Is Next → Next Topic 24.7: API Gateway, Eureka Discovery & Redis Caching Integration

24.7 API Gateway, Eureka Discovery & Redis Caching Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Routing all external requests through `api-gateway` (`8080`) + Eureka (`8761`) • Redis `@Cacheable` on hot `GET /api/v1/jobs/{id}` with `@CacheEvict` on job edits • Gateway Redis rate limiting • Production Edge-Case & Pitfall Guard (API Gateway, Eureka Discovery & Redis Caching Integration) • Hands-On Java Implementation & Verification (API Gateway, Eureka Discovery & Redis Caching Integration) • API Gateway, Eureka Discovery & Redis Caching Integration — Technical Interview & Code Walkthrough Readiness • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Gateway

Purpose: Protecting login/apply endpoints with Redis rate limits and serving hot job detail views in < 2ms.

What Topic Is Next → Next Topic 24.8: Comprehensive Test Suite & OpenAPI Documentation

24.8 Comprehensive Test Suite & OpenAPI Documentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JUnit 5 + Mockito service unit tests • `@WebMvcTest` RBAC/validation tests • PostgreSQL `Testcontainers` integration tests • `springdoc-openapi` Swagger UI across services • Production Edge-Case & Pitfall Guard (Comprehensive Test Suite & OpenAPI Documentation) • Hands-On Java Implementation & Verification (Comprehensive Test Suite & OpenAPI Documentation) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comprehensive Test Suite

Purpose: Proving to technical interviewers with one `./mvnw verify` run that your Capstone is genuinely engineered, not a fragile demo.

What Topic Is Next → Next Topic 24.9: Docker Compose Packaging, GitHub Actions CI & Cloud Deployment

24.9 Docker Compose Packaging, GitHub Actions CI & Cloud Deployment [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage Java 21 `Dockerfile`s • Root `docker-compose.yml` with healthchecks (`Postgres` • `Redis` • `Kafka` • Production Edge-Case & Pitfall Guard (Docker Compose Packaging, GitHub Actions CI & Cloud Deployment) • Hands-On Java Implementation & Verification (Docker Compose Packaging, GitHub Actions CI & Cloud Deployment) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Compose Packaging

Purpose: Enabling any reviewer or recruiter to clone the repo, copy `.env.example` -> `.env`, and boot the entire system with `docker compose up -d`.

What Topic Is Next → Next Topic 24.10: Capstone Technical README, Postman Suite & Placement Viva Defense

24.10 Capstone Technical README, Postman Suite & Placement Viva Defense [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Polishing portfolio `README.md` (architecture diagram • ER diagram • Tech choices • Curl/Postman quickstart • Production Edge-Case & Pitfall Guard (Capstone Technical README, Postman Suite & Placement Viva Defense) • Hands-On Java Implementation & Verification (Capstone Technical README, Postman Suite & Placement Viva Defense) • Idiomatic JBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Java Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Technical README

Purpose: Converting your 6-month engineering effort into concrete, verifiable resume bullets and confident technical interview performance.

What Topic Is Next → Module 24 Capstone → End-of-Course Live Client Project: Job Portal Backend System

Module 24 Hands-On Capstone Project (100 Marks): Final Industry Capstone: Job Portal Backend System (Distributed Java 21, Spring Boot 3, Kafka, Redis & Docker) — Design, engineer, test, containerize, deploy, and defend a portfolio-ready distributed Job Portal Backend System comprising `User Service`, `Candidate Service`, `Employer Service`, `Job Serv...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Job Portal Backend System • **Confirmed Role:** Java Backend Developer Project Intern

Domain & Production Infra: Live Enterprise Java Enterprise Backend & Microservices Engineering Production Engagement (Distributed Java 21, Spring Boot 3, Spring Security/JWT, PostgreSQL, Eureka, API Gateway, Kafka, Redis & Docker Compose Microservices)

- Architected and implemented 'Job Portal Backend System' using Java 21 LTS, Core Java & OOP, Java Collections Framework, Generics, Lambdas & Stream API, JVM & Multithreading under Senior Working IT Professional code review.
- Engineered production authentication/RBAC, schema migrations, input validation, caching/security hardening, and clean modular layers.
- Completed automated unit/integration test suites, Docker/IaC packaging, and Git pull-request review sprints.
- Deployed and defended the complete production build with observability, runbooks, and 1-on-1 architecture viva sign-off.

Target Job Roles: Java Backend Developer – Industry-Ready Career Program Developer, Associate Java Enterprise Backend & Microservices Engineering Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 – ₹8.5 LPA • $55K–$85K Global **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 – ₹19.0 LPA • $85K–$130K Global **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 – ₹38.0+ LPA • $130K–$185K+ Global

Optional Worthwhile Global Certifications (100% Exam-Success Mentorship Available): **Oracle Certified Professional: Java SE 21 Developer (1Z0-830):** ₹10.0 LPA – ₹24.0 LPA (+25–35% CTC Edge) • **OpenJS Node.js Services Developer (JSNSD):** ₹14.0 LPA – ₹30.0 LPA (API & Security Exam) • **MongoDB Certified Associate Developer (Node.js) (C100DEV):** ₹10.0 LPA – ₹24.0 LPA (+25% Data Backend Edge)