Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: JAVA • **Updated:** 18 Nov 2026
https://mockattempt.com/courses/core-java-training

JAVA • PROGRAMMING & BACKEND | **BEGINNER TO ADVANCED** | **100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER**

Core & Advanced Java Programming (Java 21 LTS)

Go from zero programming background to writing clean, object-oriented, concurrent Java 21 LTS applications mastering OOP, HashMap/Collections internals, Generics, Lambdas, Stream API, Optional, Records, File I/O, and Multithreading.

Duration: 7 Weeks • Weekend Live (Sat–Sun • 2h/Day) (80 Hrs) • **Modules:** 7 • **Live Topics:** 116 • **Subtopics:** 928+ • **Labs & Tasks:** 232 • **Mocks / AI Vivas:** 116 / 116 • **Projects:** 7 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,999 (Net Tuition ₹1,694 + Included 18% GST ₹305) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹25/Guided Hr (~₹25/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online 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) • 10–12 Weeks Weekend Track)

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

Core Technologies & Engineering Stack: Java 21 LTS • OOP & Interfaces • Strings & Arrays • Exception Handling • Collections Framework • HashMap Internals • Generics & Records • Lambdas & Stream API • JVM & Multithreading • NIO.2 File I/O & Maven

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Build modular, cleanly packaged Java 21 applications using strict OOP encapsulation, interfaces, abstract classes, and custom exceptions.
2. Choose optimal Java Collections (`ArrayList`, `HashSet`, `HashMap`, `TreeMap`, `PriorityQueue`) with clear internal hashing & Big-O understanding.
3. Transform data declaratively using Generics, Functional Interfaces, Lambdas, Stream API, `Optional`, and Java `record`s.
4. Understand JVM memory regions and write thread-safe concurrent code with locks, `ExecutorService`, and `CompletableFuture`.
5. Crack LeetCode and HackerRank problems in Java 21 LTS with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (Java 21 LTS, OOP & Interfaces, Strings & Arrays, Exception Handling, Collections Framework) — Automated Java 21 LTS, OOP & Interfaces, Strings & Arrays, Exception Handling, Collections Framework test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Core & Advanced Java Programming (Java 21 LTS) topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular Java 21 LTS, OOP & Interfaces, Strings & Arrays, Exception Handling, Collections Framework production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Core & Advanced Java Programming (Java 21 LTS).

Eligibility & Prerequisites: Zero prior programming experience required — starts from JDK 21 installation and first Java program. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews. • 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 (7 MODULES • 116 LIVE TOPICS • 928+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Core Java 21 & Object-Oriented Programming (Weeks 1–4) (59 Topics / 472 Subtopics) • Modules 5–7: Month 2 – Collections, Streams, JVM & Concurrency (Weeks 5–7) (57 Topics / 456 Subtopics)

MODULE 1 OF 7 • STAGE 1 • 11 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 Pitfall Guard (Introduction to Java) • Hands-On Verification (Introduction to Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Java Architecture (JDK, JRE and JVM)) • Hands-On Verification (Java Architecture (JDK, JRE and JVM)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Java Installation and Environment Setup) • Hands-On Verification (Java Installation and Environment Setup) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (First Java Program) • Hands-On Verification (First Java Program) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Java Compilation and Bytecode Execution) • Hands-On Verification (Java Compilation and Bytecode Execution) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Variables in Java) • Hands-On Verification (Variables in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Data Types in Java) • Hands-On Verification (Data Types in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Type Conversion and Casting) • Hands-On Verification (Type Conversion and Casting) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Operators in Java) • Hands-On Verification (Operators in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (User Input (Scanner and BufferedReader)) • Hands-On Verification (User Input (Scanner and BufferedReader)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Comments and Javadoc Documentation) • Hands-On Verification (Comments and Javadoc Documentation) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Java Coding Conventions and Clean Code Basics) • Hands-On Verification (Java Coding Conventions and Clean Code Basics) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Week 1 Git Setup and Daily Commit Discipline) • Hands-On Verification (Week 1 Git Setup and Daily Commit Discipline) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 1 • 11 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 Pitfall Guard (Conditional Statements) • Hands-On Verification (Conditional Statements) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Switch Statements and Switch Expressions) • Hands-On Verification (Switch Statements and Switch Expressions) • Switch Statements and Switch Expressions — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Loops (for, while, do-while)) • Hands-On Verification (Loops (for, while, do-while)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Loop Control (break, continue and Nested Loops)) • Hands-On Verification (Loop Control (break, continue and Nested Loops)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Patterns (Star, Number and Character Patterns)) • Hands-On Verification (Patterns (Star, Number and Character Patterns)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Arrays Introduction, Declaration and Initialization) • Hands-On Verification (Arrays Introduction, Declaration and Initialization) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Array Operations (Traversal, Insertion, Deletion and Searching)) • Hands-On Verification (Array Operations (Traversal, Insertion, Deletion and Searching)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (One-Dimensional Arrays Manipulation) • Hands-On Verification (One-Dimensional Arrays Manipulation) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Multidimensional Arrays) • Hands-On Verification (Multidimensional Arrays) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Matrix Operations (Addition, Subtraction, Transpose and Multiplication)) • Hands-On Verification (Matrix Operations (Addition, Subtraction, Transpose and Multiplication)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Enhanced For Loop (For-Each)) • Hands-On Verification (Enhanced For Loop (For-Each)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Array Utilities (java.util.Arrays)) • Hands-On Verification (Array Utilities (java.util.Arrays)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 1 • 11 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 Pitfall Guard (Class and Object) • Hands-On Verification (Class and Object) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Methods in Java) • Hands-On Verification (Methods in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Constructors and Constructor Overloading) • Hands-On Verification (Constructors and Constructor Overloading) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (this Keyword) • Hands-On Verification (this Keyword) • this Keyword — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (static Keyword) • Hands-On Verification (static Keyword) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Encapsulation) • Hands-On Verification (Encapsulation) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Inheritance) • Hands-On Verification (Inheritance) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (super Keyword) • Hands-On Verification (super Keyword) • super Keyword — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Method Overloading (Compile-Time Polymorphism)) • Hands-On Verification (Method Overloading (Compile-Time Polymorphism)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Method Overriding (Runtime Polymorphism)) • Hands-On Verification (Method Overriding (Runtime Polymorphism)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Polymorphism (Static vs Dynamic Binding)) • Hands-On Verification (Polymorphism (Static vs Dynamic Binding)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Abstraction and Abstract Classes) • Hands-On Verification (Abstraction and Abstract Classes) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Interfaces and Multiple Interface Implementation) • Hands-On Verification (Interfaces and Multiple Interface Implementation) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Modern Interface Features (Default, Static and Private Methods)) • Hands-On Verification (Modern Interface Features (Default, Static and Private Methods)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (final Keyword (Variables, Methods and Classes)) • Hands-On Verification (final Keyword (Variables, Methods and Classes)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Object Class (toString, equals, hashCode and getClass)) • Hands-On Verification (Object Class (toString, equals, hashCode and getClass)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 1 • 11 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 Pitfall Guard (String Immutability and String Pool) • Hands-On Verification (String Immutability and String Pool) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (String Comparison (==, equals, equalsIgnoreCase, compareTo)) • Hands-On Verification (String Comparison (==, equals, equalsIgnoreCase, compareTo)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (String Methods) • Hands-On Verification (String Methods) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (StringBuilder) • Hands-On Verification (StringBuilder) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (StringBuffer) • Hands-On Verification (StringBuffer) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Wrapper Classes) • Hands-On Verification (Wrapper Classes) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Autoboxing and Unboxing) • Hands-On Verification (Autoboxing and Unboxing) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Exception Handling Hierarchy) • Hands-On Verification (Exception Handling Hierarchy) • Exception Handling Hierarchy — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (try-catch Blocks) • Hands-On Verification (try-catch Blocks) • Idiomatic JAVA Implementation: immutable Records, Generics,

Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (finally Block and Resource Cleanup) • Hands-On Verification (finally Block and Resource Cleanup) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (throw and throws) • Hands-On Verification (throw and throws) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Custom Exceptions) • Hands-On Verification (Custom Exceptions) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Checked Exceptions) • Hands-On Verification (Checked Exceptions) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Unchecked Exceptions) • Hands-On Verification (Unchecked Exceptions) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Java Assertions) • Hands-On Verification (Java Assertions) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Packages in Java) • Hands-On Verification (Packages in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Access Modifiers (public, protected, default, private)) • Hands-On Verification (Access Modifiers (public, protected, default, private)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Enums in Java) • Hands-On Verification (Enums in Java) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 2 • 11 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 Pitfall Guard (Collections Introduction and Architecture) • Hands-On Verification (Collections Introduction and Architecture) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (List Interface) • Hands-On Verification (List Interface) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (ArrayList) • Hands-On Verification (ArrayList) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (LinkedList) • Hands-On Verification (LinkedList) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Vector) • Hands-On Verification (Vector) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Stack) • Hands-On Verification (Stack) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Queue Interface) • Hands-On Verification (Queue Interface) • Queue Interface — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (PriorityQueue) • Hands-On Verification (PriorityQueue) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Deque and ArrayDeque) • Hands-On Verification (Deque and ArrayDeque) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Set Interface) • Hands-On Verification (Set Interface) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (HashSet) • Hands-On Verification (HashSet) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (LinkedHashSet) • Hands-On Verification (LinkedHashSet) • LinkedHashSet — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for LinkedHashSet • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

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 Pitfall Guard (TreeSet) • Hands-On Verification (TreeSet) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Map Interface) • Hands-On Verification (Map Interface) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (HashMap Internals) • Hands-On Verification (HashMap Internals) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (LinkedHashMap) • Hands-On Verification (LinkedHashMap) • LinkedHashMap — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for LinkedHashMap • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

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 Pitfall Guard (TreeMap) • Hands-On Verification (TreeMap) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Hashtable) • Hands-On Verification (Hashtable) • Hashtable — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Iterator and Fail-Fast Behaviour) • Hands-On Verification (Iterator and Fail-Fast Behaviour) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (ListIterator) • Hands-On Verification (ListIterator) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Comparable (Natural Ordering)) • Hands-On Verification (Comparable (Natural Ordering)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Comparator (Custom Multi-Field Sorting)) • Hands-On Verification (Comparator (Custom Multi-Field Sorting)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (equals and hashCode Contract in Collections) • Hands-On Verification (equals and hashCode Contract in Collections) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 2 • 10 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 Pitfall Guard (Generics (Classes, Interfaces and Methods)) • Hands-On Verification (Generics (Classes, Interfaces and Methods)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Bounded Generics) • Hands-On Verification (Bounded Generics) • Bounded Generics — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Bounded Generics • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

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 Pitfall Guard (Wildcards (?, extends and super — PECS)) • Hands-On Verification (Wildcards (?, extends and super — PECS)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Lambda Expressions) • Hands-On Verification (Lambda Expressions) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Core Functional Interfaces (Predicate, Function, Consumer, Supplier)) • Hands-On Verification (Core Functional Interfaces (Predicate, Function, Consumer, Supplier)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer)) • Hands-On Verification (Bi-Arity Functional Interfaces (BiPredicate, BiFunction, BiConsumer)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Method References) • Hands-On Verification (Method References) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Stream API Architecture and Creation) • Hands-On Verification (Stream API Architecture and Creation) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Stream Intermediate Operations) • Hands-On Verification (Stream Intermediate Operations) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Stream Terminal Operations) • Hands-On Verification (Stream Terminal Operations) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Collectors (groupingBy, partitioningBy, joining, mapping)) • Hands-On Verification (Collectors (groupingBy, partitioningBy, joining, mapping)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Optional for Null-Safe Return Contracts) • Hands-On Verification (Optional for Null-Safe Return Contracts) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Date and Time API (java.time)) • Hands-On Verification (Date and Time API (java.time)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Interface Default and Static Enhancements) • Hands-On Verification (Interface Default and Static Enhancements) • Interface Default and Static Enhancements — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching)) • Hands-On Verification (Modern Java 17/21 Features (Records, Sealed Classes, Text Blocks, Pattern Matching)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (File I/O and NIO.2 (Path, Files and Streams)) • Hands-On Verification (File I/O and NIO.2 (Path, Files and Streams)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 7 • STAGE 2 • 10 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 Pitfall Guard (JVM Architecture Overview) • Hands-On Verification (JVM Architecture Overview) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Class Loading Subsystem) • Hands-On Verification (Class Loading Subsystem) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (JVM Memory Management (Heap, Stack and Metaspace)) • Hands-On Verification (JVM Memory Management (Heap, Stack and Metaspace)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Garbage Collection Fundamentals) • Hands-On Verification (Garbage Collection Fundamentals) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (JIT Compiler (Just-In-Time Compilation)) • Hands-On Verification (JIT Compiler (Just-In-Time Compilation)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Thread Fundamentals (Process vs Thread)) • Hands-On Verification (Thread Fundamentals (Process vs Thread)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Thread Creation (Thread Class and Runnable Interface)) • Hands-On Verification (Thread Creation (Thread Class and Runnable Interface)) • Thread Creation (Thread Class and Runnable Interface) — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Thread Lifecycle States) • Hands-On Verification (Thread Lifecycle States) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Thread Methods (start, run, sleep, join, interrupt)) • Hands-On Verification (Thread Methods (start, run, sleep, join, interrupt)) • Thread Methods (start, run, sleep, join, interrupt) — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Synchronization (synchronized Methods and Blocks)) • Hands-On Verification (Synchronization (synchronized Methods and Blocks)) • Synchronization (synchronized Methods and Blocks) — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Race Conditions and Shared Mutable State) • Hands-On Verification (Race Conditions and Shared Mutable State) • Race Conditions and Shared Mutable State — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Deadlocks (Causes and Prevention)) • Hands-On Verification (Deadlocks (Causes and Prevention)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (volatile Keyword and Memory Visibility) • Hands-On Verification (volatile Keyword and Memory Visibility) • volatile Keyword and Memory Visibility — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Atomic Classes (AtomicInteger, AtomicLong, LongAdder)) • Hands-On Verification (Atomic Classes (AtomicInteger, AtomicLong, LongAdder)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (ExecutorService and Thread Pools) • Hands-On Verification (ExecutorService and Thread Pools) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Callable and Future) • Hands-On Verification (Callable and Future) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (CompletableFuture (Asynchronous Programming)) • Hands-On Verification (CompletableFuture (Asynchronous Programming)) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 Pitfall Guard (Concurrent Collections (ConcurrentHashMap and BlockingQueue)) • Hands-On Verification (Concurrent Collections (ConcurrentHashMap and BlockingQueue)) • Concurrent Collections (ConcurrentHashMap and BlockingQueue) — Interview Readiness • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT 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 → End-of-Course Live Client Project: Core & Advanced Java Programming (Java 21 LTS) — Enterprise Client Production Platform

Module 7 Hands-On Capstone Project (100 Marks): Capstone: Core Java 21 Banking, Payroll & Thread-Safe Seat Reservation Engine — Develop a concurrent seat-reservation simulator and bounded `ArrayBlockingQueue` producer-consumer audit pipeline using locks, `AtomicLong` counters, `ExecutorService`, and `CompletableFutur...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Core & Advanced Java Programming (Java 21 LTS) — Enterprise Client Production Platform • **Confirmed Role:** Core & Advanced Java Programming (Java 21 LTS) Engineering Project Intern

Domain & Production Infra: Live Enterprise Programming & Backend Client Engagement (Java 21 LTS, OOP & Interfaces, Strings & Arrays, Exception Handling, Collections Framework • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular Java 21 LTS, OOP & Interfaces, Strings & Arrays, Exception Handling, Collections Framework production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Core & Advanced Java Programming (Java 21 LTS) Developer, Associate Programming & Backend 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

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/core-java-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.