MockAttempt

TECH ACADEMY • OFFICIAL COURSE SYLLABUS BROCHURE

MockAttempt Academy Pvt. Ltd. • Live Architect Mentorship, AI Assessments & Client Production

Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: JAVA • **Updated:** 04 Nov 2026
https://mockattempt.com/courses/java-training

JAVA • PROGRAMMING & BACKEND **BEGINNER TO JOB-READY JAVA DEVELOPER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Java Training

Master Java from JVM internals, OOPs, Strings, Exceptions & Multithreading to Collections, Generics, Java 8+ Streams & MySQL JDBC — with runnable Java programs, 386 Topic Mock Tests (17,860 MCQs), and Live Voice + MockAttempt AI Technical Interviews at the most affordable fee.

Duration: 16–18 Weeks • Live + Hands-On Projects (209 Hrs) • **Modules:** 16 • **Live Topics:** 53 • **Subtopics:** 424+ • **Labs & Tasks:** 106 • **Mocks / AI Vivas:** 386 / 386 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹2,499 (Net Tuition ₹2,118 + Included 18% GST ₹381) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹12/Guided Hr (~₹47/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 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 • JVM & Memory Model • OOPs & Solid • Multithreading • Collections & Generics • Lambda & Stream API • JDBC & MySQL

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Understand JDK, JRE, JVM ClassLoader, Heap vs Stack memory, and JIT compilation internals
2. Write clean object-oriented Java programs using Inheritance, Polymorphism, Interfaces, Exceptions, and Multithreading
3. Master ArrayList, HashSet, HashMap, Generics, Java 8+ Lambdas/Streams, and SQL-injection-safe MySQL JDBC
4. Practice 386 Timed Mock Exams (17,500 Java MCQs) and 386 Spoken Voice + MockAttempt AI Technical Interviews
5. Java 21 LTS features in production: Records, Sealed Classes, Pattern Matching, Streams, and Optional safety.
6. Crack LeetCode and HackerRank problems in Java 21 LTS with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. High-throughput concurrency using Project Loom Virtual Threads, `CompletableFuture`, and lock-free collections.
8. JVM memory model literacy (Heap generations, G1/ZGC garbage collection, thread dumps, and leak profiling).

Eligibility & Prerequisites: Zero prior Java experience required — starts from JDK setup, bytecode compilation, and Hello World • Any basic laptop/PC capable of running JDK 17/21 and VS Code / IntelliJ Community Edition • 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 live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 53 LIVE TOPICS • 424+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (17 Topics / 136 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (14 Topics / 112 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (10 Topics / 80 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (12 Topics / 96 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Java Fundamentals, JVM/JDK/JRE Architecture, Data Types & Operators

Master Java platform independence (WORA), JDK vs JRE vs JVM internals, primitive vs reference types, type promotion, and operators.

Objectives: ✓ Features of Java — Platform Independence, Bytecode & Object-Oriented Design — Write Once Run Anywhere (WORA), javac bytecode generation, robustness, security, multithreading, and enterprise portability. • ✓ JVM Architecture — ClassLoader, Heap, Stack, Metaspace & JIT Compiler — Bootstrap/Application ClassLoaders, Bytecode Verifier, Runtime Data Areas (Heap, Thread Stacks, PC Register, Metaspace), Execution Engine, and Garbage Collection. • ✓ Java Data Types — 8 Primitives, Wrapper Autoboxing & Type Casting — Exact bit widths of byte, short, int, long, float, double, char (UTF-16), and boolean, widening vs narrowing casts, and Integer caching. • ✓ Java Operators — Arithmetic, Bitwise, Shift (>> vs >>>), Short-Circuit & Ternary — Pre/post increment (++i vs i++), integer vs floating division, signed (>>) vs unsigned right shift (>>>), && vs &, and ternary ? :. • ✓ Build, verify, and ship Module Capstone: Smart CLI GST Invoice, Loan EMI & Income Tax Calculator in Java.

1.1 Features of Java — Platform Independence, Bytecode & Object-Oriented Design

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Bytecode & Platform independent • Core Syntax, CLI & Production APIs: javac, java, public static void main(String[] args), System.out.println() • Deep-Dive Mechanics: WORA & Object-oriented • Advanced Patterns: Robust & Main method • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Java Platform & Course Enrollment Banner • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Features of Java

Purpose: To understand and apply Features of Java — Platform Independence, Bytecode & Object-Oriented Design in real Java systems — specifically: Write Once Run Anywhere (WORA), j...

What Topic Is Next → Next Topic 1.2: JVM Architecture — ClassLoader, Heap, Stack, Metaspace & JIT Compiler

1.2 JVM Architecture — ClassLoader, Heap, Stack, Metaspace & JIT Compiler

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ClassLoader & Heap • Core Syntax, CLI & Production APIs: Runtime.getRuntime(), ClassLoader, System.gc(), JIT HotSpot • Deep-Dive Mechanics: Stack & Metaspace • Advanced Patterns: JIT compiler & Garbage Collection • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: JVM Runtime Memory & Processor Inspector • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for JVM Architecture

Purpose: To understand and apply JVM Architecture — ClassLoader, Heap, Stack, Metaspace & JIT Compiler in real Java systems — specifically: Bootstrap/Application ClassLoaders, Byt...

What Topic Is Next → Next Topic 1.3: Java Data Types — 8 Primitives, Wrapper Autoboxing & Type Casting

1.3 Java Data Types — 8 Primitives, Wrapper Autoboxing & Type Casting

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Primitive types & Widening • Core Syntax, CLI & Production APIs: byte, short, int, long (L), float (f), double, char, boolean • Deep-Dive Mechanics: Narrowing cast & UTF-16 char • Advanced Patterns: Long suffix L & Autoboxing • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Primitive Ranges, Widening & Explicit Narrowing Casts • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Data Types

Purpose: To understand and apply Java Data Types — 8 Primitives, Wrapper Autoboxing & Type Casting in real Java systems — specifically: Exact bit widths of byte, short, int, long,...

What Topic Is Next → Next Topic 1.4: Java Operators — Arithmetic, Bitwise, Shift (>> vs >>>), Short-Circuit & Ternary

1.4 Java Operators — Arithmetic, Bitwise, Shift (>> vs >>>), Short-Circuit & Ternary

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Post-increment & Short-circuit && • Core Syntax, CLI & Production APIs: ++, --, /, %, &&, ||, &, |, ^, <<, >>, >>>, ? : • Deep-Dive Mechanics: Unsigned shift >>> & Ternary • Advanced Patterns: Integer division & Precedence • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Short-Circuit Logic, Bitwise Shifts & Increment Evaluation • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Operators

Purpose: To understand and apply Java Operators — Arithmetic, Bitwise, Shift (>> vs >>>), Short-Circuit & Ternary in real Java systems — specifically: Pre/post increment (++i vs i...

What Topic Is Next → Module 1 Capstone (Smart CLI GST Invoice, Loan EMI & Income Tax Calculator in Java) → Next Topic 2.1: Java Conditional Branching — if, else-if Ladders & Guard Clauses

Module 1 Hands-On Capstone Project (100 Marks): Smart CLI GST Invoice, Loan EMI & Income Tax Calculator in Java — Build an interactive console billing & financial calculator in Java 21 using Scanner, formatted printf tables, compound/EMI math, and strict numeric range validation.

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Java Control Flow, Switch Expressions, Loops & Arrays

Write structured algorithmic logic with if-else ladders, modern Java switch expressions (->), labeled loops, and 1D/2D/jagged arrays.

Objectives: ✓ Java Conditional Branching — if, else-if Ladders & Guard Clauses — Boolean condition requirements in Java (no truthy ints!), nested conditionals, guard clauses, and ternary expressions. • ✓ Java Switch Statements & Modern Java 14+ Switch Expressions — Classic switch with break vs fall-through, supported selector types (int, String, enum), and modern arrow case -> with yield. • ✓ Java Loops — for, Enhanced for-each, while, do-while & Labeled break — Indexed iteration, read-only enhanced for-each over arrays/Iterables, do-while post-check loops, and breaking outer nested loops via labels. • ✓ Java Arrays — 1D, Multidimensional, Jagged Arrays & java.util.Arrays — Contiguous heap allocation, default zero/null initialization, .length property, jagged 2D arrays, Arrays.sort() Dual-Pivot Quicksort, and Arrays.binarySearch(). • ✓ Build, verify, and ship Module Capstone: Batch Exam Scorecard & 2D Matrix Analytics Engine.

2.1 Java Conditional Branching — if, else-if Ladders & Guard Clauses [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Boolean condition & Else-if ladder • Core Syntax, CLI & Production APIs: if (booleanExpr) { ... } else if (...) { ... } else { ... } • Deep-Dive Mechanics: Guard clause & Block scope • Advanced Patterns: Ternary & Dangling else • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Eligibility & Scholarship Tier Evaluator • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Conditional Branching

Purpose: To understand and apply Java Conditional Branching — if, else-if Ladders & Guard Clauses in real Java systems — specifically: Boolean condition requirements in Java (no t...

What Topic Is Next → Next Topic 2.2: Java Switch Statements & Modern Java 14+ Switch Expressions

2.2 Java Switch Statements & Modern Java 14+ Switch Expressions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Switch expression & Fall-through • Core Syntax, CLI & Production APIs: switch (val) { case "A" -> ...; default -> ...; } • Deep-Dive Mechanics: Tableswitch & Yield • Advanced Patterns: String switch & Enum case • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Modern Java Arrow Switch Expression Role Router • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Switch Statements

Purpose: To understand and apply Java Switch Statements & Modern Java 14+ Switch Expressions in real Java systems — specifically: Classic switch with break vs fall-through, suppor...

What Topic Is Next → Next Topic 2.3: Java Loops — for, Enhanced for-each, while, do-while & Labeled break

2.3 Java Loops — for, Enhanced for-each, while, do-while & Labeled break [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: For-each & Do-while • Core Syntax, CLI & Production APIs: for (int i = 0; i < n; i++), for (T item : items), outer: break outer; • Deep-Dive Mechanics: Labeled break & Continue • Advanced Patterns: Off-by-one & Iterator • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Prime Sieve Check & Labeled 2D Matrix Search • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Loops

Purpose: To understand and apply Java Loops — for, Enhanced for-each, while, do-while & Labeled break in real Java systems — specifically: Indexed iteration, read-only enhanced fo...

What Topic Is Next → Next Topic 2.4: Java Arrays — 1D, Multidimensional, Jagged Arrays & java.util.Arrays

2.4 Java Arrays — 1D, Multidimensional, Jagged Arrays & java.util.Arrays [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Contiguous heap & Arr.length • Core Syntax, CLI & Production APIs: new int[n], arr.length, Arrays.sort(), Arrays.toString(), System.arraycopy() • Deep-Dive Mechanics: Jagged array & Arrays.sort • Advanced Patterns: Arrays.toString & ArrayIndexOutOfBoundsException • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Array Sorting, Binary Search & Jagged Row Sums • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Arrays

Purpose: To understand and apply Java Arrays — 1D, Multidimensional, Jagged Arrays & java.util.Arrays in real Java systems — specifically: Contiguous heap allocation, default zero...

What Topic Is Next → Module 2 Capstone (Batch Exam Scorecard & 2D Matrix Analytics Engine) → Next Topic 3.1: Java Classes & Objects — Fields, Methods, Object Creation & References

Module 2 Hands-On Capstone Project (100 Marks): Batch Exam Scorecard & 2D Matrix Analytics Engine — Develop a Java console application that stores multi-subject marks for a cohort in a 2D array, computes student-wise percentages, toppers, subject averages, and binary-searches scores using...

MODULE 3 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Java OOPs Foundations — Classes, Constructors, static & this

Build modular object-oriented Java programs using classes, overloaded constructors, constructor chaining (this()), and static class members.

Objectives: ✓ Java Classes & Objects — Fields, Methods, Object Creation & References — Class blueprints, new heap allocation, reference variables, default field values vs uninitialized local variables, and anonymous objects. • ✓ Constructors in Java — Default, Parameterized, Overloading & Copy Pattern — Implicit no-arg constructor rule, constructor overloading, initialization order, private constructors (Singleton/utility classes), and defensive copying. • ✓ The static Keyword — Static Variables, Methods, Blocks & Nested Classes — Class-level state shared across instances, static initializer blocks, utility methods, and why static contexts cannot reference this/super. • ✓ The this Keyword — Shadowing Resolution, Constructor Chaining & Fluent APIs — Disambiguating shadowed fields (this.name = name), delegating constructors via this(...), passing current instance, and fluent method chaining (return this). • ✓ Build, verify, and ship Module Capstone: Automated Student Admission & Unique Roll-Number Generator.

3.1 Java Classes & Objects — Fields, Methods, Object Creation & References [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: New operator & Object reference • Core Syntax, CLI & Production APIs: class Student { ... }, new Student(), obj.method() • Deep-Dive Mechanics: Instance field & NullPointerException • Advanced Patterns: TLAB heap & Method invocation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Course Blueprint & Object Reference Behavior • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Classes

Purpose: To understand and apply Java Classes & Objects — Fields, Methods, Object Creation & References in real Java systems — specifically: Class blueprints, new heap allocation,...

What Topic Is Next → Next Topic 3.2: Constructors in Java — Default, Parameterized, Overloading & Copy Pattern

3.2 Constructors in Java — Default, Parameterized, Overloading & Copy Pattern [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: No-arg suppression & Constructor overloading • Core Syntax, CLI & Production APIs: public ClassName(...) { ... }, private ClassName() {} • Deep-Dive Mechanics: Copy constructor & Private constructor • Advanced Patterns: Initialization order & Singleton • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Overloaded Constructors & Copy Constructor Pattern • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Constructors in Java

Purpose: To understand and apply Constructors in Java — Default, Parameterized, Overloading & Copy Pattern in real Java systems — specifically: Implicit no-arg constructor rule, c...

What Topic Is Next → Next Topic 3.3: The static Keyword — Static Variables, Methods, Blocks & Nested Classes

3.3 The static Keyword — Static Variables, Methods, Blocks & Nested Classes [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Static field & Static block • Core Syntax, CLI & Production APIs: static int count;, static { ... }, public static class Builder • Deep-Dive Mechanics: Invokestatic & Static nested class • Advanced Patterns: Class loading & Shared state • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Static ID Counter & Static Initialization Block • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for The static Keyword

Purpose: To understand and apply The static Keyword — Static Variables, Methods, Blocks & Nested Classes in real Java systems — specifically: Class-level state shared across insta...

What Topic Is Next → Next Topic 3.4: The this Keyword — Shadowing Resolution, Constructor Chaining & Fluent APIs

3.4 The this Keyword — Shadowing Resolution, Constructor Chaining & Fluent APIs [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Field shadowing & This() first statement • Core Syntax, CLI & Production APIs: this.field = field;, this(...);, return this; • Deep-Dive Mechanics: Fluent builder & Aload_0 • Advanced Patterns: Instance reference & Chaining • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Fluent Query Builder Using return this • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for The this Keyword

Purpose: To understand and apply The this Keyword — Shadowing Resolution, Constructor Chaining & Fluent APIs in real Java systems — specifically: Disambiguating shadowed fields (t...

What Topic Is Next → Module 3 Capstone (Automated Student Admission & Unique Roll-Number Generator) → Next Topic 4.1: Inheritance in Java — extends, super Keyword, IS-A vs HAS-A Composition

Module 3 Hands-On Capstone Project (100 Marks): Automated Student Admission & Unique Roll-Number Generator — Create an OOP student enrollment system where a static sequence counter auto-issues unique student IDs (MAA-1001, MAA-1002...), overloaded constructors support scholarship/regular admissions...

MODULE 4 OF 16 • STAGE 1 • 15 LIVE + LAB HOURS • 5 TOPICS (40 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Java OOPs Pillars — Inheritance, Polymorphism, Abstraction & Encapsulation

Master extends, super(), compile-time overloading vs runtime dynamic method dispatch, abstract classes, Java 8+ interfaces, and immutable encapsulation.

Objectives: ✓ Inheritance in Java — extends, super Keyword, IS-A vs HAS-A Composition — Single/multilevel/hierarchical inheritance, why Java forbids multiple class inheritance (Diamond Problem), super() constructor chaining, and final classes. • ✓ Polymorphism — Method Overloading vs Method Overriding & Dynamic Dispatch — Compile-time overloading resolution vs runtime invokevirtual overriding, @Override annotation, covariant return types, and why static/private methods are hidden, not overridden. • ✓ Abstract Classes & Template Method Pattern in Java — Partial abstraction (0 to 100%), mixing abstract methods with concrete state/constructors, and structuring reusable workflows via the Template Method pattern. • ✓ Java Interfaces — Multiple Implementation, default & static Methods — Contract-driven architecture, implicit public static final constants, implementing multiple interfaces, Java 8 default/static methods, and resolving diamond default conflicts. • ✓ Encapsulation, Access Modifiers & Immutable Classes in Java — Four visibility levels (private, default/package-private, protected, public), invariant-checking setters, defensive copying, and building thread-safe immutable classes. • ✓ Build, verify, and ship Module Capstone: Polymorphic Payment Gateway (UPI, Card, NetBanking) & Audit Ledger.

4.1 Inheritance in Java — extends, super Keyword, IS-A vs HAS-A Composition [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Extends & Super() • Core Syntax, CLI & Production APIs: class Child extends Parent, super(...), super.method(), final class • Deep-Dive Mechanics: Diamond problem & IS-A vs HAS-A • Advanced Patterns: Final class & Constructor order • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inheritance Hierarchy & super() Initialization Order • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Inheritance in Java

Purpose: To understand and apply Inheritance in Java — extends, super Keyword, IS-A vs HAS-A Composition in real Java systems — specifically: Single/multilevel/hierarchical inheri...

What Topic Is Next → Next Topic 4.2: Polymorphism — Method Overloading vs Method Overriding & Dynamic Dispatch

4.2 Polymorphism — Method Overloading vs Method Overriding & Dynamic Dispatch [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Overloading vs overriding & Dynamic method dispatch • Core Syntax, CLI & Production APIs: @Override, Parent ref = new Child();, instanceof pattern matching • Deep-Dive Mechanics: @Override & Covariant return • Advanced Patterns: Method hiding & Instanceof • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Runtime Dynamic Method Dispatch Showcase • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Polymorphism

Purpose: To understand and apply Polymorphism — Method Overloading vs Method Overriding & Dynamic Dispatch in real Java systems — specifically: Compile-time overloading resolution...

What Topic Is Next → Next Topic 4.3: Abstract Classes & Template Method Pattern in Java

4.3 Abstract Classes & Template Method Pattern in Java [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Abstract class & Abstract method • Core Syntax, CLI & Production APIs: public abstract class Base { abstract void step(); } • Deep-Dive Mechanics: Template method & Constructors in abstract • Advanced Patterns: Illegal modifiers & Partial abstraction • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Template Method Pattern Using an Abstract Report Generator • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Abstract Classes

Purpose: To understand and apply Abstract Classes & Template Method Pattern in Java in real Java systems — specifically: Partial abstraction (0 to 100%), mixing abstract methods w...

What Topic Is Next → Next Topic 4.4: Java Interfaces — Multiple Implementation, default & static Methods

4.4 Java Interfaces — Multiple Implementation, default & static Methods [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Implements & Multiple interfaces • Core Syntax, CLI & Production APIs: interface Payable { ... }, default void log(), InterfaceA.super.m() • Deep-Dive Mechanics: Default method & Public static final • Advanced Patterns: Interface.super & Loose coupling • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Multiple Interfaces & Default Method Resolution • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Interfaces

Purpose: To understand and apply Java Interfaces — Multiple Implementation, default & static Methods in real Java systems — specifically: Contract-driven architecture, implicit pu...

What Topic Is Next → Next Topic 4.5: Encapsulation, Access Modifiers & Immutable Classes in Java

4.5 Encapsulation, Access Modifiers & Immutable Classes in Java [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Private fields & Access modifiers • Core Syntax, CLI & Production APIs: private final T field;, public T getField(), defensive copy • Deep-Dive Mechanics: Protected vs package & Immutable class • Advanced Patterns: Defensive copy & Invariant validation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Encapsulated Bank Wallet with Validation Guards • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Encapsulation

Purpose: To understand and apply Encapsulation, Access Modifiers & Immutable Classes in Java in real Java systems — specifically: Four visibility levels (private, default/package-...

What Topic Is Next → Module 4 Capstone (Polymorphic Payment Gateway (UPI, Card, NetBanking) & Audit Ledger) → Next Topic 5.1: Java Strings — Immutability, String Constant Pool & == vs .equals()

Module 4 Hands-On Capstone Project (100 Marks): Polymorphic Payment Gateway (UPI, Card, NetBanking) & Audit Ledger — Architect an extensible Java payment processing engine using a PaymentGateway interface, an AbstractPayment base class for common validation/receipts, concrete UpiPayment and CardPayment sub...

MODULE 5 OF 16 • STAGE 2 • 13 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Java Strings, StringBuilder, Regex & Exception Handling

Understand String Constant Pool interning, O(n) StringBuilder mutation, java.util.regex Pattern/Matcher, and Checked vs Unchecked exceptions.

Objectives: ✓ Java Strings — Immutability, String Constant Pool & == vs .equals() — Why java.lang.String is final and immutable, String Constant Pool (SCP) interning, hashcode caching, Compact Strings (byte[]), and core methods. • ✓ StringBuilder vs StringBuffer — Mutable Text Manipulation in O(n) — Dynamic resizable buffer capacity, .append(), .insert(), .delete(), .reverse(), and thread-safe synchronized StringBuffer vs fast unsynchronized StringBuilder. • ✓ Java Regular Expressions — Pattern, Matcher & Text Extraction — java.util.regex.Pattern compilation, Matcher.find() vs matches(), capture groups group(1), and String.replaceAll(). • ✓ Exception Handling — Checked vs Unchecked, try-with-resources & Custom Exceptions — Throwable -> Error vs Exception -> RuntimeException hierarchy, multi-catch, finally guarantees, AutoCloseable try-with-resources, and throw vs throws. • ✓ Build, verify, and ship Module Capstone: Fault-Tolerant KYC Document Validator & Log Masking Engine.

5.1 Java Strings — Immutability, String Constant Pool & == vs .equals() [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: String Constant Pool & Immutability • Core Syntax, CLI & Production APIs: .equals(), .equalsIgnoreCase(), .substring(), .split(), .intern() • Deep-Dive Mechanics: == vs equals & Intern() • Advanced Patterns: HashCode caching & Compact Strings • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: String Pool Reference Identity vs .equals() Verification • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Strings

Purpose: To understand and apply Java Strings — Immutability, String Constant Pool & == vs .equals() in real Java systems — specifically: Why java.lang.String is final and immutab...

What Topic Is Next → Next Topic 5.2: StringBuilder vs StringBuffer — Mutable Text Manipulation in O(n)

5.2 StringBuilder vs StringBuffer — Mutable Text Manipulation in O(n) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: StringBuilder & StringBuffer • Core Syntax, CLI & Production APIs: new StringBuilder(capacity), .append(), .reverse(), .toString() • Deep-Dive Mechanics: Mutable buffer & Append O(1) • Advanced Patterns: Reverse() & Equals pitfall • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Fast CSV Builder & Palindrome Checker with StringBuilder • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for StringBuilder vs StringBuffer

Purpose: To understand and apply StringBuilder vs StringBuffer — Mutable Text Manipulation in O(n) in real Java systems — specifically: Dynamic resizable buffer capacity, .append(...

What Topic Is Next → Next Topic 5.3: Java Regular Expressions — Pattern, Matcher & Text Extraction

5.3 Java Regular Expressions — Pattern, Matcher & Text Extraction [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Pattern.compile & Matcher.find • Core Syntax, CLI & Production APIs: Pattern.compile(), matcher.find(), matcher.group(), replaceAll() • Deep-Dive Mechanics: Double backslash & Capture groups • Advanced Patterns: Thread-safe Pattern & ReplaceAll • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Compiled Regex Extractor for Student Codes & Fees • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Regular Expressions

Purpose: To understand and apply Java Regular Expressions — Pattern, Matcher & Text Extraction in real Java systems — specifically: java.util.regex.Pattern compilation, Matcher.fi...

What Topic Is Next → Next Topic 5.4: Exception Handling — Checked vs Unchecked, try-with-resources & Custom Exceptions

5.4 Exception Handling — Checked vs Unchecked, try-with-resources & Custom Exceptions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Checked vs Unchecked & Try-with-resources • Core Syntax, CLI & Production APIs: try (Resource r = ...) { } catch (ExA | ExB e) { } finally { } • Deep-Dive Mechanics: Throw vs throws & AutoCloseable • Advanced Patterns: Finally & Custom exception • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: AutoCloseable Resource & Custom Unchecked Exception Flow • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Exception Handling

Purpose: To understand and apply Exception Handling — Checked vs Unchecked, try-with-resources & Custom Exceptions in real Java systems — specifically: Throwable -> Error vs Excep...

What Topic Is Next → Module 5 Capstone (Fault-Tolerant KYC Document Validator & Log Masking Engine) → Next Topic 6.1: Multithreading & Thread Lifecycle — Runnable, start() vs run() & join()

Module 5 Hands-On Capstone Project (100 Marks): Fault-Tolerant KYC Document Validator & Log Masking Engine — Build a Java validation & PII-masking service that validates emails, Indian mobile numbers, and student IDs using compiled Regex Patterns, masks sensitive digits via StringBuilder, and throw...

MODULE 6 OF 16 • STAGE 2 • 13 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Java Multithreading, Synchronization & Executor Thread Pools

Write safe concurrent Java systems using Thread/Runnable/Callable, thread lifecycle states, synchronized monitors, volatile, and ExecutorService pools.

Objectives: ✓ Multithreading & Thread Lifecycle — Runnable, start() vs run() & join() — NEW -> RUNNABLE -> BLOCKED/WAITING/TIMED_WAITING -> TERMINATED states, implementing Runnable vs extending Thread, sleep(), and join(). • ✓ Synchronization, Intrinsic Monitors, volatile & Race Condition Prevention — Atomicity & visibility hazards (count++), object monitor locks (`synchronized`), static class locks, `volatile` happens-before visibility, and preventing Deadlock. • ✓ ExecutorService Thread Pools, Callable<V>, Future & CompletableFuture — Reusing worker threads via Executors.newFixedThreadPool(), returning results/exceptions from Callable<V>, Future.get(), and graceful pool shutdown. • ✓ Build, verify, and ship Module Capstone: Thread-Safe Flash Sale Seat Booking & ExecutorService Worker Engine.

6.1 Multithreading & Thread Lifecycle — Runnable, start() vs run() & join() [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Start vs run & Runnable • Core Syntax, CLI & Production APIs: new Thread(runnable).start(), Thread.sleep(), thread.join() • Deep-Dive Mechanics: Thread states & IllegalThreadStateException • Advanced Patterns: Join() & Virtual Threads • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Parallel Worker Thread Execution & join() Coordination • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Multithreading

Purpose: To understand and apply Multithreading & Thread Lifecycle — Runnable, start() vs run() & join() in real Java systems — specifically: NEW -> RUNNABLE -> BLOCKED/WAITING/TI...

What Topic Is Next → Next Topic 6.2: Synchronization, Intrinsic Monitors, volatile & Race Condition Prevention

6.2 Synchronization, Intrinsic Monitors, volatile & Race Condition Prevention [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Race condition & Intrinsic monitor • Core Syntax, CLI & Production APIs: synchronized (lock) { ... }, volatile boolean, AtomicInteger • Deep-Dive Mechanics: Happens-before & Volatile vs AtomicInteger • Advanced Patterns: Deadlock lock ordering & Synchronized block • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Thread-Safe Counter Under Concurrent Contention • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Synchronization

Purpose: To understand and apply Synchronization, Intrinsic Monitors, volatile & Race Condition Prevention in real Java systems — specifically: Atomicity & visibility hazards (cou...

What Topic Is Next → Next Topic 6.3: ExecutorService Thread Pools, Callable<V>, Future & CompletableFuture

6.3 ExecutorService Thread Pools, Callable<V>, Future & CompletableFuture [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ExecutorService & Callable vs Runnable • Core Syntax, CLI & Production APIs: ExecutorService, Callable<T>, Future<T>, pool.shutdown() • Deep-Dive Mechanics: Future.get() & Pool.shutdown() • Advanced Patterns: BlockingQueue & CompletableFuture • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Callable Task Submission & Future Result Aggregation • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for ExecutorService Thread Pools

Purpose: To understand and apply ExecutorService Thread Pools, Callable<V>, Future & CompletableFuture in real Java systems — specifically: Reusing worker threads via Executors.ne...

What Topic Is Next → Module 6 Capstone (Thread-Safe Flash Sale Seat Booking & ExecutorService Worker Engine) → Next Topic 7.1: File Handling in Java — File vs java.nio.file.Path & Files API

Module 6 Hands-On Capstone Project (100 Marks): Thread-Safe Flash Sale Seat Booking & ExecutorService Worker Engine — Simulate 200 concurrent student booking requests competing for 25 limited workshop seats using an ExecutorService thread pool, synchronized/ReentrantLock seat allocation, Callable<String> re...

MODULE 7 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Java File I/O, Buffered Streams & Object Serialization

Process files and binary/character streams efficiently using java.io, java.nio.file.Files, BufferedReader, and Serializable + transient.

Objectives: ✓ File Handling in Java — File vs java.nio.file.Path & Files API — Creating, inspecting, reading, and writing files using modern java.nio.file.Path/Files alongside classic java.io.File. • ✓ BufferedReader & BufferedWriter — High-Throughput Character Streaming — Decorator pattern wrapping Reader/InputStreamReader with an 8 KB RAM buffer to minimize expensive OS kernel read() syscalls. • ✓ Java Serialization — Serializable, serialVersionUID & transient Keyword — Converting object graphs to byte streams via ObjectOutputStream, version compatibility with serialVersionUID, and excluding passwords/secrets via transient. • ✓ Build, verify, and ship Module Capstone: Buffered CSV Attendance Importer & Binary Snapshot Serializer.

7.1 File Handling in Java — File vs java.nio.file.Path & Files API

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Path.of & Java.nio.file.Files • Core Syntax, CLI & Production APIs: Path.of(...), Files.readString(), Files.writeString(), Files.lines() • Deep-Dive Mechanics: StandardCharsets.UTF_8 & File descriptor • Advanced Patterns: Byte vs Char stream & IOException • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cross-Platform Path Resolution & Normalization Demo • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for File Handling in Java

Purpose: To understand and apply File Handling in Java — File vs java.nio.file.Path & Files API in real Java systems — specifically: Creating, inspecting, reading, and writing fil...

What Topic Is Next → Next Topic 7.2: BufferedReader & BufferedWriter — High-Throughput Character Streaming

7.2 BufferedReader & BufferedWriter — High-Throughput Character Streaming

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: BufferedReader & Decorator pattern • Core Syntax, CLI & Production APIs: new BufferedReader(new StringReader(...)), .readLine(), .lines() • Deep-Dive Mechanics: ReadLine() EOF null & 8KB buffer • Advanced Patterns: InputStreamReader & Try-with-resources • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Buffered Line-by-Line Score Stream Aggregation • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for BufferedReader

Purpose: To understand and apply BufferedReader & BufferedWriter — High-Throughput Character Streaming in real Java systems — specifically: Decorator pattern wrapping Reader/Input...

What Topic Is Next → Next Topic 7.3: Java Serialization — Serializable, serialVersionUID & transient Keyword

7.3 Java Serialization — Serializable, serialVersionUID & transient Keyword

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Serializable marker & SerialVersionUID • Core Syntax, CLI & Production APIs: implements Serializable, private static final long serialVersionUID, transient • Deep-Dive Mechanics: Transient & NotSerializableException • Advanced Patterns: ObjectOutputStream & No constructor on deserialize • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: In-Memory Serialization Verifying transient Password Masking • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Serialization

Purpose: To understand and apply Java Serialization — Serializable, serialVersionUID & transient Keyword in real Java systems — specifically: Converting object graphs to byte stre...

What Topic Is Next → Module 7 Capstone (Buffered CSV Attendance Importer & Binary Snapshot Serializer) → Next Topic 8.1: Java ArrayList vs LinkedList — Dynamic Resizing, Fail-Fast Iterators & Complexity

Module 7 Hands-On Capstone Project (100 Marks): Buffered CSV Attendance Importer & Binary Snapshot Serializer — Create a Java file-processing utility that streams student attendance CSV logs via BufferedReader inside try-with-resources, flags < 75% attendance alerts, and persists binary state snapshot...

MODULE 8 OF 16 • STAGE 2 • 15 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Java Collections Framework, HashMap Internals & Generics

Master dynamic lists, hash sets, Java 8+ treeified HashMap internals (hashCode/equals contract), Comparable/Comparator, and bounded Generics.

Objectives: ✓ Java ArrayList vs LinkedList — Dynamic Resizing, Fail-Fast Iterators & Complexity — Object[] backing array, 1.5x growth formula, O(1) random get(i) vs O(n) middle shift, Iterator.remove(), and List.of() unmodifiable lists. • ✓ Java HashSet, LinkedHashSet & TreeSet — Uniqueness & Equals/HashCode Contract — How HashSet delegates to a backing HashMap<E, PRESENT>, LinkedHashSet insertion order, TreeSet Red-Black O(log n) sorted order, and custom object deduplication. • ✓ Java HashMap Internals — Bucket Hashing, Load Factor 0.75 & Treeification — Node<K,V>[] array, `(n - 1) & hash` index calculation, load factor 0.75 resize doubling, Java 8 collision chain conversion to Red-Black Tree (TREEIFY_THRESHOLD = 8), and ConcurrentHashMap. • ✓ Java Generics — Type Parameters, Bounded Wildcards (PECS) & Type Erasure — Compile-time type safety without ClassCastException, generic methods `<T>`, upper bounds `? extends T` (Producer), lower bounds `? super T` (Consumer), and JVM Type Erasure. • ✓ Build, verify, and ship Module Capstone: Generic In-Memory Course Search, Deduplicated Tag Index & Leaderboard Engine.

8.1 Java ArrayList vs LinkedList — Dynamic Resizing, Fail-Fast Iterators & Complexity

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 1.5x resizing & O(1) random access • Core Syntax, CLI & Production APIs: List<T> list = new ArrayList<>(), .add(), .get(), .removeIf(), Collections.sort() • Deep-Dive Mechanics: Fail-fast modCount & ConcurrentModificationException • Advanced Patterns: RemoveIf & ArrayList vs LinkedList • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Conditional Removal & Custom Sorting on ArrayList • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java ArrayList vs LinkedList

Purpose: To understand and apply Java ArrayList vs LinkedList — Dynamic Resizing, Fail-Fast Iterators & Complexity in real Java systems — specifically: Object[] backing array, 1.5...

What Topic Is Next → Next Topic 8.2: Java HashSet, LinkedHashSet & TreeSet — Uniqueness & Equals/HashCode Contract

8.2 Java HashSet, LinkedHashSet & TreeSet — Uniqueness & Equals/HashCode Contract [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Backing HashMap & Equals and hashCode • Core Syntax, CLI & Production APIs: new HashSet<>(), new LinkedHashSet<>(), new TreeSet<>(), Objects.hash() • Deep-Dive Mechanics: LinkedHashSet order & TreeSet O(log n) • Advanced Patterns: Deduplication & Null handling • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Preserving Unique Insertion Order vs Sorted Order with Sets • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java HashSet

Purpose: To understand and apply Java HashSet, LinkedHashSet & TreeSet — Uniqueness & Equals/HashCode Contract in real Java systems — specifically: How HashSet delegates to a back...

What Topic Is Next → Next Topic 8.3: Java HashMap Internals — Bucket Hashing, Load Factor 0.75 & Treeification

8.3 Java HashMap Internals — Bucket Hashing, Load Factor 0.75 & Treeification [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Load factor 0.75 & TREEIFY_THRESHOLD 8 • Core Syntax, CLI & Production APIs: map.getOrDefault(), map.computeIfAbsent(), map.merge(), entrySet() • Deep-Dive Mechanics: Immutable keys & ComputeIfAbsent • Advanced Patterns: ConcurrentHashMap & (n - 1) & hash • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Frequency Aggregation Using HashMap.merge() & getOrDefault() • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java HashMap Internals

Purpose: To understand and apply Java HashMap Internals — Bucket Hashing, Load Factor 0.75 & Treeification in real Java systems — specifically: Node<K,V>[] array, `(n - 1) & hash`...

What Topic Is Next → Next Topic 8.4: Java Generics — Type Parameters, Bounded Wildcards (PECS) & Type Erasure

8.4 Java Generics — Type Parameters, Bounded Wildcards (PECS) & Type Erasure [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Type Erasure & PECS rule • Core Syntax, CLI & Production APIs: class Box<T>, <T extends Comparable<T>>, List<? extends Number> • Deep-Dive Mechanics: ? extends vs ? super & Invariance • Advanced Patterns: Bounded type <T extends> & ClassCastException prevention • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Bounded Generic Upper-Bound Sum (? extends Number) • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Generics

Purpose: To understand and apply Java Generics — Type Parameters, Bounded Wildcards (PECS) & Type Erasure in real Java systems — specifically: Compile-time type safety without Cla...

What Topic Is Next → Module 8 Capstone (Generic In-Memory Course Search, Deduplicated Tag Index & Leaderboard Engine) → Next Topic 9.1: Java Lambda Expressions, @FunctionalInterface & Method References (::)

Module 8 Hands-On Capstone Project (100 Marks): Generic In-Memory Course Search, Deduplicated Tag Index & Leaderboard Engine — Build an in-memory catalog & student leaderboard engine in Java using HashMap<String, Course> for O(1) code lookup, HashSet<String> for unique skill tags, ArrayList + Comparator chaining for...

MODULE 9 OF 16 • STAGE 3 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Modern Java (Java 8–21) — Lambda Expressions & Stream API

Write concise, declarative data pipelines using @FunctionalInterface, Predicate/Function/Consumer/Supplier, method references, Streams, Collectors, and Optional.

Objectives: ✓ Java Lambda Expressions, @FunctionalInterface & Method References (::) — Single Abstract Method (SAM) interfaces, Predicate<T>, Function<T,R>, Consumer<T>, Supplier<T>, effectively final variable capture, and invokedynamic. • ✓ Java Stream API, Lazy Evaluation, Collectors & Optional<T> — Lazy intermediate operations (filter, map, flatMap, distinct, sorted) vs terminal operations (collect, reduce, count, findFirst), groupingBy, and Optional. • ✓ Build, verify, and ship Module Capstone: Declarative E-Commerce Analytics & Grouping Engine with Java Streams.

9.1 Java Lambda Expressions, @FunctionalInterface & Method References (::) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @FunctionalInterface & Effectively final • Core Syntax, CLI & Production APIs: @FunctionalInterface, (a, b) -> expr, ClassName::methodName, Predicate<T> • Deep-Dive Mechanics: Invokedynamic & Predicate & Function • Advanced Patterns: Method reference :: & SAM interface • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Functional Predicate Composition & Method Reference Pipeline • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Lambda Expressions

Purpose: To understand and apply Java Lambda Expressions, @FunctionalInterface & Method References (::) in real Java systems — specifically: Single Abstract Method (SAM) interface...

What Topic Is Next → Next Topic 9.2: Java Stream API, Lazy Evaluation, Collectors & Optional<T>

9.2 Java Stream API, Lazy Evaluation, Collectors & Optional<T> [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Lazy intermediate ops & Single-use stream • Core Syntax, CLI & Production APIs: .stream().filter(...).map(...).collect(Collectors.groupingBy(...)), Optional.ofNullable() • Deep-Dive Mechanics: Collectors.groupingBy & FlatMap • Advanced Patterns: Optional.orElseGet & Short-circuit • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Stream Grouping, Filtering & Revenue Aggregation • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java Stream API

Purpose: To understand and apply Java Stream API, Lazy Evaluation, Collectors & Optional<T> in real Java systems — specifically: Lazy intermediate operations (filter, map, flatMap...

What Topic Is Next → Module 9 Capstone (Declarative E-Commerce Analytics & Grouping Engine with Java Streams) → Next Topic 10.1: Java JDBC Architecture & MySQL Database Connectivity

Module 9 Hands-On Capstone Project (100 Marks): Declarative E-Commerce Analytics & Grouping Engine with Java Streams — Build an analytical reporting engine over 1,000+ order records using pure Java Stream pipelines: compute category-wise revenue (`Collectors.groupingBy` + `summingDouble`), top 3 customers, p...

MODULE 10 OF 16 • STAGE 3 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Java JDBC Database Connectivity (MySQL), PreparedStatement & Transactions

Connect Java backend services to MySQL/MariaDB safely using JDBC Type-4 drivers, SQL-injection-proof PreparedStatement, batch updates, and ACID transactions.

Objectives: ✓ Java JDBC Architecture & MySQL Database Connectivity — JDBC API architecture, Type-4 Pure Java MySQL Connector/J driver, JDBC connection URLs, DriverManager / DataSource pooling (HikariCP), and ResultSet cursor iteration. • ✓ JDBC PreparedStatement, SQL Injection Defense, Batching & ACID Transactions — Precompiled parameterized SQL (`?`), `setString`/`setInt`, `executeQuery` vs `executeUpdate`, `addBatch`/`executeBatch`, and `conn.setAutoCommit(false)` commit/rollback. • ✓ Build, verify, and ship Module Capstone: JDBC + MySQL Student Admission, Fee Ledger & Certificate DAO Layer.

10.1 Java JDBC Architecture & MySQL Database Connectivity

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Type-4 Driver & DriverManager • Core Syntax, CLI & Production APIs: DriverManager.getConnection(url, user, pass), ResultSet.next(), HikariDataSource • Deep-Dive Mechanics: ResultSet 1-based index & Try-with-resources • Advanced Patterns: Connection pooling & SQLException • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: JDBC Connection URL Builder & DAO Configuration Pattern • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java JDBC Architecture

Purpose: To understand and apply Java JDBC Architecture & MySQL Database Connectivity in real Java systems — specifically: JDBC API architecture, Type-4 Pure Java MySQL Connector/...

What Topic Is Next → Next Topic 10.2: JDBC PreparedStatement, SQL Injection Defense, Batching & ACID Transactions

10.2 JDBC PreparedStatement, SQL Injection Defense, Batching & ACID Transactions

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: PreparedStatement ? & SQL injection defense • Core Syntax, CLI & Production APIs: conn.prepareStatement(sql), ps.setString(1, val), ps.addBatch(), conn.commit(), conn.rollbac... • Deep-Dive Mechanics: ExecuteQuery vs executeUpdate & ExecuteBatch() • Advanced Patterns: SetAutoCommit(false) & Commit and rollback • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Parameterized SQL Specification & Transaction Guard Verifier • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for JDBC PreparedStatement

Purpose: To understand and apply JDBC PreparedStatement, SQL Injection Defense, Batching & ACID Transactions in real Java systems — specifically: Precompiled parameterized SQL (`?...

What Topic Is Next → Module 10 Capstone (JDBC + MySQL Student Admission, Fee Ledger & Certificate DAO Layer) → Next Topic 11.1: Immutable Java Records & Compact Canonical Constructors

Module 10 Hands-On Capstone Project (100 Marks): JDBC + MySQL Student Admission, Fee Ledger & Certificate DAO Layer — Build a clean Data Access Object (DAO) persistence layer in Java 21 connecting to MySQL/MariaDB via JDBC, supporting parameterized student CRUD, batch mock-question insertion (`addBatch`/`ex...

MODULE 11 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Modern Java 17/21 Features: Records, Sealed Types & Pattern Matching

Production-grade Java Training engineering module covering Modern Java 17/21 Features: Records, Sealed Types & Pattern Matching with hands-on labs and capstone build.

Objectives: ✓ Immutable Java Records & Compact Canonical Constructors — Validate invariants once at construction and eliminate verbose POJO boilerplate. • ✓ Sealed Interfaces & Exhaustive Switch Pattern Matching — Guarantee every transaction state is handled at compile time without default-branch bugs. • ✓ Text Blocks, Optional Chains & java.time UTC APIs — Write clean SQL/JSON literals and timezone-safe billing schedules in enterprise Java. • ✓ Build, verify, and ship Module Capstone: Type-Safe Banking Domain Engine in Modern Java 21.

11.1 Immutable Java Records & Compact Canonical Constructors

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Compact Canonical Constructors • Integration with Sealed Interfaces & Exhaustive Switch Pattern Matching • Live Debugging, Telemetry & Technical Interview Trade • What Is Immutable Java Records & Compact Canonical Constructors & Why Industry Teams Use It • In Java Training, Immutable Java Records & Compact Canonical Constructors exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Immutable Java Records & Compact Canonical Constructors in real Java systems — specifically: Validate invariants once at construction and eliminat...

What Topic Is Next → Next Topic 11.2: Sealed Interfaces & Exhaustive Switch Pattern Matching

11.2 Sealed Interfaces & Exhaustive Switch Pattern Matching

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Exhaustive Switch Pattern Matching • Integration with Text Blocks, Optional Chains & java.time UTC APIs • Live Debugging, Telemetry & Technical Interview Trade • What Is Sealed Interfaces & Exhaustive Switch Pattern Matching & Why Industry Teams Use It • In Java Training, Sealed Interfaces & Exhaustive Switch Pattern Matching exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Sealed Interfaces & Exhaustive Switch Pattern Matching in real Java systems — specifically: Guarantee every transaction state is handled at compil...

What Topic Is Next → Next Topic 11.3: Text Blocks, Optional Chains & java.time UTC APIs

11.3 Text Blocks, Optional Chains & java.time UTC APIs

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Optional Chains • Integration with Immutable Java Records & Compact Canonical Constructors • Live Debugging, Telemetry & Technical Interview Trade • What Is Text Blocks, Optional Chains & java.time UTC APIs & Why Industry Teams Use It • In Java Training, Text Blocks, Optional Chains & java.time UTC APIs exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Text Blocks, Optional Chains & java.time UTC APIs in real Java systems — specifically: Write clean SQL/JSON literals and timezone-safe billing sch...

What Topic Is Next → Module 11 Capstone (Type-Safe Banking Domain Engine in Modern Java 21) → Next Topic 12.1: ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture

Module 11 Hands-On Capstone Project (100 Marks): Type-Safe Banking Domain Engine in Modern Java 21 — Design, implement, test, and document 'Type-Safe Banking Domain Engine in Modern Java 21' applying every production concept from Module 11 (Modern Java 17/21 Features: Records, Sealed Types...

MODULE 12 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Java Concurrency, ExecutorService, Locks & Virtual Threads

Production-grade Java Training engineering module covering Java Concurrency, ExecutorService, Locks & Virtual Threads with hands-on labs and capstone build.

Objectives: ✓ ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture — Execute parallel I/O tasks with strict timeouts instead of spawning unbounded raw Threads. • ✓ ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock — Protect shared in-memory caches from race conditions and deadlocks under heavy load. • ✓ Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale — Handle tens of thousands of concurrent socket/JDBC operations with lightweight virtual threads. • ✓ Build, verify, and ship Module Capstone: High-Throughput Multi-Threaded Order Matching Engine.

12.1 ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Callable/Future • Integration with ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock • Live Debugging, Telemetry & Technical Interview Trade • What Is ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture & Why Industry Teams Use It • In Java Training, ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture in real Java systems — specifically: Execute parallel I/O tasks with strict timeout...

What Topic Is Next → Next Topic 12.2: ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock

12.2 ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of AtomicReference • Integration with Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale • Live Debugging, Telemetry & Technical Interview Trade • What Is ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock & Why Industry Teams Use It • In Java Training, ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply ConcurrentHashMap, AtomicReference & ReentrantReadWriteLock in real Java systems — specifically: Protect shared in-memory caches from race conditi...

What Topic Is Next → Next Topic 12.3: Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale

12.3 Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with ThreadPoolExecutor Sizing, Callable/Future & CompletableFuture • Live Debugging, Telemetry & Technical Interview Trade • What Is Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale & Why Industry Teams Use It • In Java Training, Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Java 21 Virtual Threads (Project Loom) for Blocking I/O Scale in real Java systems — specifically: Handle tens of thousands of concurrent socket/J...

What Topic Is Next → Module 12 Capstone (High-Throughput Multi-Threaded Order Matching Engine) → Next Topic 13.1: Stack Frames vs Heap Generations (Eden, Survivor, Old Gen)

Module 12 Hands-On Capstone Project (100 Marks): High-Throughput Multi-Threaded Order Matching Engine — Design, implement, test, and document 'High-Throughput Multi-Threaded Order Matching Engine' applying every production concept from Module 12 (Java Concurrency, ExecutorService, Locks & Virt...

MODULE 13 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

JVM Memory Architecture, Garbage Collection & Performance Tuning

Production-grade Java Training engineering module covering JVM Memory Architecture, Garbage Collection & Performance Tuning with hands-on labs and capstone build.

Objectives: ✓ Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) — Prevent OutOfMemoryError leaks caused by unbounded static maps and unclosed streams. • ✓ G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis — Diagnose CPU spikes and deadlocks rapidly using jcmd, jstack, and Flight Recorder. • ✓ JDBC Connection Pooling (HikariCP), PreparedStatements & Batching — Execute parameterized SQL 20x faster while completely blocking SQL injection. • ✓ Build, verify, and ship Module Capstone: Low-Latency JVM Profiler & Leak Diagnostic Lab.

13.1 Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Survivor • Integration with G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis • Live Debugging, Telemetry & Technical Interview Trade • What Is Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) & Why Industry Teams Use It • In Java Training, Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) in real Java systems — specifically: Prevent OutOfMemoryError leaks caused by unbounded...

What Topic Is Next → Next Topic 13.2: G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis

13.2 G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of ZGC Tuning Flags plus Heap/Thread Dump Analysis • Integration with JDBC Connection Pooling (HikariCP), PreparedStatements & Batching • Live Debugging, Telemetry & Technical Interview Trade • What Is G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis & Why Industry Teams Use It • In Java Training, G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply G1GC & ZGC Tuning Flags plus Heap/Thread Dump Analysis in real Java systems — specifically: Diagnose CPU spikes and deadlocks rapidly using jcmd,...

What Topic Is Next → Next Topic 13.3: JDBC Connection Pooling (HikariCP), PreparedStatements & Batching

13.3 JDBC Connection Pooling (HikariCP), PreparedStatements & Batching [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of PreparedStatements • Integration with Stack Frames vs Heap Generations (Eden, Survivor, Old Gen) • Live Debugging, Telemetry & Technical Interview Trade • What Is JDBC Connection Pooling (HikariCP), PreparedStatements & Batching & Why Industry Teams Use It • In Java Training, JDBC Connection Pooling (HikariCP), PreparedStatements & Batching exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply JDBC Connection Pooling (HikariCP), PreparedStatements & Batching in real Java systems — specifically: Execute parameterized SQL 20x faster while...

What Topic Is Next → Module 13 Capstone (Low-Latency JVM Profiler & Leak Diagnostic Lab) → Next Topic 14.1: SOLID Principles & Dependency Inversion in Real Java Codebases

Module 13 Hands-On Capstone Project (100 Marks): Low-Latency JVM Profiler & Leak Diagnostic Lab — Design, implement, test, and document 'Low-Latency JVM Profiler & Leak Diagnostic Lab' applying every production concept from Module 13 (JVM Memory Architecture, Garbage Collection & Perform...

MODULE 14 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

SOLID Design Principles, GoF Patterns, Maven & JUnit 5/Mockito

Production-grade Java Training engineering module covering SOLID Design Principles, GoF Patterns, Maven & JUnit 5/Mockito with hands-on labs and capstone build.

Objectives: ✓ SOLID Principles & Dependency Inversion in Real Java Codebases — Keep business rules decoupled so swapping SMS/email/payment providers takes minutes. • ✓ Factory, Builder, Strategy & Observer Patterns in Production — Replace sprawling 500-line if/else chains with clean, extensible polymorphic strategies. • ✓ Maven Multi-Module Builds, JUnit 5 Assertions & Mockito Stubbing — Verify every Java service edge case automatically in CI before packaging JAR artifacts. • ✓ Build, verify, and ship Module Capstone: Modular Enterprise Invoice & Notification Library with JUnit 5.

14.1 SOLID Principles & Dependency Inversion in Real Java Codebases [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Dependency Inversion in Real Java Codebases • Integration with Factory, Builder, Strategy & Observer Patterns in Production • Live Debugging, Telemetry & Technical Interview Trade • What Is SOLID Principles & Dependency Inversion in Real Java Codebases & Why Industry Teams Use It • In Java Training, SOLID Principles & Dependency Inversion in Real Java Codebases exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply SOLID Principles & Dependency Inversion in Real Java Codebases in real Java systems — specifically: Keep business rules decoupled so swapping SMS/...

What Topic Is Next → Next Topic 14.2: Factory, Builder, Strategy & Observer Patterns in Production

14.2 Factory, Builder, Strategy & Observer Patterns in Production [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Builder • Integration with Maven Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is Factory, Builder, Strategy & Observer Patterns in Production & Why Industry Teams Use It • In Java Training, Factory, Builder, Strategy & Observer Patterns in Production exists for one clear real • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Factory, Builder, Strategy & Observer Patterns in Production in real Java systems — specifically: Replace sprawling 500-line if/else chains with c...

What Topic Is Next → Next Topic 14.3: Maven Multi-Module Builds, JUnit 5 Assertions & Mockito Stubbing

14.3 Maven Multi-Module Builds, JUnit 5 Assertions & Mockito Stubbing [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of JUnit 5 Assertions • Integration with SOLID Principles & Dependency Inversion in Real Java Codebases • Live Debugging, Telemetry & Technical Interview Trade • What Is Maven Multi • In Java Training, Maven Multi • Even if you come from a non • Runnable Code Implementation (JAVA)

Purpose: To understand and apply Maven Multi-Module Builds, JUnit 5 Assertions & Mockito Stubbing in real Java systems — specifically: Verify every Java service edge case automati...

What Topic Is Next → Module 14 Capstone (Modular Enterprise Invoice & Notification Library with JUnit 5) → Next Topic 15.1: Immutable Java Records, Compact Constructors & Defensive Validation

Module 14 Hands-On Capstone Project (100 Marks): Modular Enterprise Invoice & Notification Library with JUnit 5 — Design, implement, test, and document 'Modular Enterprise Invoice & Notification Library with JUnit 5' applying every production concept from Module 14 (SOLID Design Principles, GoF Patterns...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Java 21 Modern Language Features: Records, Sealed Types & Virtual Threads

Self-explanatory list-and-bullet production mastery module covering Java 21 Modern Language Features: Records, Sealed Types & Virtual Threads in Java Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Immutable Java Records, Compact Constructors & Defensive Validation — Purpose: Model domain DTOs and value objects concisely with zero boilerplate mutability bugs. • ✓ Sealed Classes, Exhaustive Switch Pattern Matching & Null-Safe Design — Purpose: Let the Java compiler guarantee every business state transition is handled explicitly. • ✓ Java 21 Virtual Threads (Project Loom) & Structured Concurrency — Purpose: Scale I/O-bound services to tens of thousands of concurrent tasks with simple readable code. • ✓ Build, verify, and document 'High-Concurrency Java 21 Virtual-Thread Order Processor & Pattern-Matched Domain' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Immutable Java Records, Compact Constructors & Defensive Validation [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Immutable Java Records, Compact Constructors & Defensive Validation • Performance Optimization, Complexity & Latency Impact of Immutable Java Records, Compact Constructors & Defensive V... • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Immutable Java Records • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply Immutable Java Records, Compact Constructors & Defensive Validation in real Java systems — specifically: Model domain DTOs and value objects conci...

What Topic Is Next → Next Topic 15.2: Sealed Classes, Exhaustive Switch Pattern Matching & Null-Safe Design

15.2 Sealed Classes, Exhaustive Switch Pattern Matching & Null-Safe Design [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Sealed Classes, Exhaustive Switch Pattern Matching & Null • Performance Optimization, Complexity & Latency Impact of Sealed Classes, Exhaustive Switch Pattern Matching & Null • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Sealed Classes • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply Sealed Classes, Exhaustive Switch Pattern Matching & Null-Safe Design in real Java systems — specifically: Let the Java compiler guarantee every b...

What Topic Is Next → Next Topic 15.3: Java 21 Virtual Threads (Project Loom) & Structured Concurrency

15.3 Java 21 Virtual Threads (Project Loom) & Structured Concurrency [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Java 21 Virtual Threads (Project Loom) & Structured Concurrency • Performance Optimization, Complexity & Latency Impact of Java 21 Virtual Threads (Project Loom) & Structured Concur... • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Java 21 Virtual Threads (Project Loom) • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply Java 21 Virtual Threads (Project Loom) & Structured Concurrency in real Java systems — specifically: Scale I/O-bound services to tens of thousands...

What Topic Is Next → Module 15 Capstone (High-Concurrency Java 21 Virtual-Thread Order Processor & Pattern-Matched Domain) → Next Topic 16.1: ClassLoading, Stack vs Heap Memory, String Pool & Garbage Collection Mechanics

Module 15 Hands-On Capstone Project (100 Marks): High-Concurrency Java 21 Virtual-Thread Order Processor & Pattern-Matched Domain — Architect, implement, test, and document 'High-Concurrency Java 21 Virtual-Thread Order Processor & Pattern-Matched Domain' applying every bullet-wise engineering concept from Module 15 (Jav...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

JVM Internals, JUnit 5 Testing Discipline & Enterprise Java Interviews

Self-explanatory list-and-bullet production mastery module covering JVM Internals, JUnit 5 Testing Discipline & Enterprise Java Interviews in Java Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ ClassLoading, Stack vs Heap Memory, String Pool & Garbage Collection Mechanics — Purpose: Explain how the JVM allocates, interns, and reclaims memory under real enterprise workloads. • ✓ Clean Unit Testing with JUnit 5 Parameterized Tests, Assertions & Test Doubles — Purpose: Verify business rules and boundary exceptions automatically on every Maven/Gradle build. • ✓ Core Java Design Patterns, Concurrency Traps & SDE Interview Readiness — Purpose: Solve equals/hashCode contracts, deadlock prevention, and SOLID refactoring challenges. • ✓ Build, verify, and document 'JUnit 5 + Mockito Verification Suite & JVM GC Diagnostic Lab' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 ClassLoading, Stack vs Heap Memory, String Pool & Garbage Collection Mechanics [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for ClassLoading, Stack vs Heap Memory, String Pool & Garbage Collection Mecha... • Performance Optimization, Complexity & Latency Impact of ClassLoading, Stack vs Heap Memory, String Pool & Garbage.... • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for ClassLoading • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply ClassLoading, Stack vs Heap Memory, String Pool & Garbage Collection Mechanics in real Java systems — specifically: Explain how the JVM allocates,...

What Topic Is Next → Next Topic 16.2: Clean Unit Testing with JUnit 5 Parameterized Tests, Assertions & Test Doubles

16.2 Clean Unit Testing with JUnit 5 Parameterized Tests, Assertions & Test Doubles [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Clean Unit Testing with JUnit 5 Parameterized Tests, Assertions & Test Dou... • Performance Optimization, Complexity & Latency Impact of Clean Unit Testing with JUnit 5 Parameterized Tests, Asser... • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Clean Unit Testing with JUnit 5 Parameterized Tests • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply Clean Unit Testing with JUnit 5 Parameterized Tests, Assertions & Test Doubles in real Java systems — specifically: Verify business rules and boun...

What Topic Is Next → Next Topic 16.3: Core Java Design Patterns, Concurrency Traps & SDE Interview Readiness

16.3 Core Java Design Patterns, Concurrency Traps & SDE Interview Readiness [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Core Java Design Patterns, Concurrency Traps & SDE Interview Readiness • Performance Optimization, Complexity & Latency Impact of Core Java Design Patterns, Concurrency Traps & SDE Intervi... • Live Debugging, Stack • Runnable Code Implementation (JAVA) • Hands • Idiomatic JAVA Implementation: immutable Records, Generics, Streams & Virtual Threads • JDK 21 JVM Stack/Heap & JIT Execution & Memory Lifecycle for Core Java Design Patterns • Production Hardening & Security: defensive immutability, null-safety & thread-confinement

Purpose: To understand and apply Core Java Design Patterns, Concurrency Traps & SDE Interview Readiness in real Java systems — specifically: Solve equals/hashCode contracts, deadl...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: CoreBank Concurrent Account Ledger, Multithreaded Batch & JDBC Audit Engine

Module 16 Hands-On Capstone Project (100 Marks): JUnit 5 + Mockito Verification Suite & JVM GC Diagnostic Lab — Architect, implement, test, and document 'JUnit 5 + Mockito Verification Suite & JVM GC Diagnostic Lab' applying every bullet-wise engineering concept from Module 16 (JVM Internals, JUnit 5...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: CoreBank Concurrent Account Ledger, Multithreaded Batch & JDBC Audit Engine • **Confirmed Role:** Core & Advanced Java 21 Engineering Intern

Domain & Production Infra: BFSI Core Banking & Financial Compliance Client (Java 21 LTS + Virtual Threads + Maven + HikariCP JDBC + JUnit 5 Linux Server)

- Designed clean SOLID domain models using Java 21 records, sealed interfaces, generics, and immutable value objects.
- Implemented thread-safe concurrent settlement workers with ExecutorService, Virtual Threads, and CompletableFuture.
- Built transactional JDBC repositories with prepared statements, HikariCP connection pooling, and batch audit logging.
- Verified business invariants with JUnit 5 test suites and deployed packaged JAR services to live client production.

Target Job Roles: Java Software Engineer, Associate Enterprise Backend Developer, Core Java & JDBC Developer, Graduate Engineer Trainee (Java)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.8 LPA – ₹8.5 LPA **Mid-Level Java Backend Engineer (1–3 Yrs):** ₹9.0 LPA – ₹18.0 LPA **Senior Java Architect / Staff Engineer (3–6+ Yrs):** ₹19.0 LPA – ₹38.0+ LPA