Clean Code and Test-Driven Development in Java

Duration: 4 Days
Available Languages: English German

Audience

Software Crafters, Software Developers, Software Testers, XP Coaches.

Precondition

Solid knowledge of the Java programming language.

Goals

Learn the benefits, mechanics, and nuts and bolts of developing software using Test-Driven Development.

Contents

Test-Driven Development (TDD) is a software development practice from Extreme Programming (XP) and Software Craft. TDD increases code coverage, leads to fast tests, and supports continuous refactoring and continuous design improvement. Some benefits of TDD include developing faster with fewer errors, reducing debug time, lean development, better design, quick feedback, and eliminating fear for continuous refactoring. Last but not least, TDD drives decoupled and thus better quality software architecture.

- Software Architecture Fundamentals for TDD
    - The Two Values of Software
    - The ATP-Trinity of project code
    - The Importance Priorities: Automation > Test > Production
    - The five major design smells
    - Cohesion and Coupling
    - What is "testable" and how is it related to maintainability / Clean Code?
    - Test Automation Pyramid
    - TDD in the context of Agile and XP
- Unit Testing fundamentals
    - The job of and work split between test frameworks
    - Anatomy of xUnit frameworks
    - JUnit 3, 4, 5; TestNG
    - Inner Workings of test frameworks
    - The Single-Assert Rule
- TDD fundamentals
    - The Three Laws of Test-Driven Development

  - # The Red-Green-Refactor Cycle
  - # The FAIR/FIRST principles
  - # How to Start
  - # ZOMBIES - Zero One Many Boundaries Interfaces Exceptions Simplicity
- Test Doubles (Stubbing and Mocking)
  - # The Ontology of Test Doubles
  - # The Two Schools of TDD: Stateism ("Chicago School") vs Mockism ("London School")
  - # Mocking and Coupling
  - # Working with Mockito and PowerMockito
  - # Avoiding tautological tests
- BDD - Behavior Driven Development
  - # 3 Amigos
  - # Specification by Example
  - # Gherkin
  - # Cucumber Java
  - # Unit Testing vs Acceptance Testing
  - # Given-When-Then vs 4 A's
- Intermediate TDD
  - # TPP - Transformation Priority Premise
  - # Transformation vs Refactoring
  - # Starting Points
  - # The Sequence for Tests
  - # TCR - test && commit || revert
  - # Hamcrest Matchers
- ATDD - Acceptance Test-Driven Development
  - # Test Automation Pyramid
  - # Using BDD on different layers
  - # Acceptance Test Step Definitions with Selenium and Appium
  - # Integration Test Step Definitions with HTTP client
  - # Unit Test Step Definitions
- TDD for Legacy Code
  - # Legacy Code Change Matrix
  - # Refactoring
  - # Characterization Testing
- TDD and Dependency Injection Frameworks
  - # Spring
  - # Spring4JUnit Test Runner, SpringExtension, SpringBootTest
  - # Configuring injection for tests: ContextConfiguration, TestPropertySource, RestClientTest, MockBean, etc
  - # Cucumber and Spring
  - # TDD performance with Spring
- Design Patterns
  - # The 5 Design Smells
  - # The SOLID Principles
  - # The Package Principles
  - # Package Metrics
  - # Creational Patterns
  - # Structural Patterns

    - # Behavioral Patterns
    - # Architecture Patterns
- Refactoring
    - # Code Smells
    - # Basic refactoring: Rename, Inline, Extract, Move
    - # Refactoring of conditional logic
    - # OOP Refactoring
    - # Refactor code smells using patterns
- Outlook
    - # Migrating from JUnit 4 to JUnit 5
    - # Test Architecture
    - # Working Effectively with Legacy Code
    - # How to migrate Test-Last to Test-First
    - # TDD and the SOLID principles
    - # TDD and Agile Development (Scrum, XP, Kanban, Lean)
    - # TDD and Software Craftsmanship
    - # TDD and Pair Programming, Mob/Ensemble Programming - Ping Pong
    - # TDD and Continuous Integration / Trunk-Based Development
    - # TDD and Continuous Delivery / DevOps

The course uses OpenJDK 18, Maven 3.8.5, Gradle 7.4.2, JUnit Platform 1.8.2, JUnit Jupiter 5.8.2, Cucumber 7.3.4, Pitest 1.8.0, and Spring Boot 2.7.0. The preferred IDE is IntelliJ IDEA 2022.1.1. Differences between JUnit 5, 4, and 3, and TestNG are covered in detail.

The course language is Java. Nelkinda also offers this course in other languages, for example, C, C++, C#, JavaScript, Kotlin, Python, Swift, and TypeScript.

Event Type

This is a full-day open (anyone can register) instructor-led classroom training about Test-Driven Development in Java. The number of seats is limited to ensure the best quality training for the participants. The course fee includes snacks and lunch.

Trainer

Your trainer for this event is Christian Hujer.

Christian Hujer has 21 years of experience in TDD and 25 years of experience in Java. He's been training developers and teams for organizations like BNP Paribas, Elsevier, Ford, Giesecke & Devrient, Nokia, SUN Microsystems, Volkswagen, and many others.

Booking

Contact Siddhesh Nikude, +91-95-52572354, training@nelkinda.com