TRAINING CATALOG

VICTOR RENTEA CONSULTING SRL, 2024

victorrentea@gmail.com ◆ (+4) 0720 019 564 ◆ @victorrentea ◆ LinkedIn Profile

Testimonials on VictorRentea.ro, LinkedIn, and Facebook

Technologies continuously change, tech stacks evolve, the mindset shifts, and the best practices constantly align, so…

⚠ Find the most up-to-date version of this file at: https://victorrentea.ro/catalog

Training Topics

The topics can be tailored to fit your needs: sub-items removed, added, or combined.

General Terms & Conditions

- Dates/Availability: workshops are scheduled typically 1-3 months in advance.
- Base fee = <request a quote> / **day**
- Group size: **max 20-25** participants.
- Training of **7 hours/day**, e.g., 9:00 - 17:00 CET, with 1 hour lunch break at half-time.
- Delivered as **online webinar**, via Zoom, Teams, Webex, or any other online meeting tool.
- Included: **training slides** packed with links for further study, and **all the code** zipped with git history.
- Included: **prep-call** to tailor the agenda to the needs of participants (30-60 minutes).
- Included: **prep-assignments** for selected topics.
- Included: **follow-up support** via **email**, and **live consultancy/coaching** (1 hour for every training day).
- Option: **on-site training** (outside Bucharest) = +50% BASE FEE, minimum **2 days**, **unlimited** attendees.
- Option: **split in 4 online hours/day** with double no. of days (for deeper learning) = +20% BASE FEE.
- Option: **large groups (over 25)** and/or **record the session** = +30% BASE FEE.
    - ⚠ Shy people will interact less + we can't dive too deep into the specifics of one project.

CLEAN CODE (2 DAYS)

Learn how to fix typical code smells and fall in love with continuous refactoring towards better code quality.

Workshop language: (ideally the participants should share one language)

- Java, Kotlin, Scala, C#
- TypeScript, JavaScript, PHP, Python

Target Audience:
- Developer looking to upgrade the quality of the code you write.
- Senior/Tech Lead/Coach/Chapter Lead concerned with helping your colleagues develop their skills.

After this workshop you will be able to:
- Identify common Code Smells and compare various ways to fix them.
- Refactor without introducing bugs, by splitting large changes in safe baby-steps.
- Boost your productivity and focus by mastering your IDE refactoring support.
- Become a better team player by coding, reviewing, and pairing like a pro.
- Learn state-of-the-art best practices of your programming language.

What to expect: A mix of short refactoring exercises with debates about design options, attitude, teamwork, and more, in an entertaining style spiced with real-life stories and analogies (preview). Concepts will be applied via code review of selected prep-assignments, live-refactoring of code samples from the audience (for private sessions), and hands-on refactoring by the participants working in mini-groups.

Option: Half-days with more hands-on: Split the online training in 4 hours / day, and allow the participants to work independently during the 2nd half of the day on refactoring exercises/self-study that I can provide.

Prep work:
1. Practice refactoring ideally in pair programming
a) Trivia Kata (est: 3-5 hours) – **MANDATORY**: send me the solutions for review **before the training** ⚠
Java: https://github.com/victorrentea/kata-trivia-java (.zip)
Non-Java: https://github.com/jbrains/trivia
b) Refactor to FP (est: 3 hours, optional): https://github.com/victorrentea/kata-streams-java (Java)

2. Send me code samples to refactor together.
- Send me **2-3 samples** of ≤ **200-300 lines** of your codebase.
- The samples should manifest common issues, debated at code reviews, perhaps refactored repeatedly.
- At least 1 participant should be **familiar with the functionality**.
- (Optional) Please try to make the extracted code sample compile.
- Without an NDA, anonymize the code: e.g., rename Account→Basket, Transfer→Apple, etc. (AI can help)

3. Read
- From *Clean Code* book by Uncle Bob (Robert C. Martin) - Chapters 1-11
- From *Refactoring* book by Martin Fowler - Chapter 2 + 3 (est: 30 minutes)
- Code Smells Catalog: https://refactoring.guru/refactoring/smells

4. Import the project
- Clone and import in IDE the project: Java, TypeScript, or PHP.

5. Master your JetBrains IDE (eg IntelliJ)
- Install plugins: SonarLint, Key Promoter, and Code Metrics.
- Discover the tricks I'll use: https://youtu.be/ZiOMQRujfMM and https://youtu.be/ZiOMQRujfMM

6. Send me a **list with the GitHub ID** for all participants the evening before the training. I will invite them to collaborate to a public Git (needed for the independent exercise during the second day).

Agenda

The agenda should be adjusted for your team's needs in a prep-call.

- **Fundamentals**
  - Refactoring Opportunities: boy-scout, pre-done, pre-change, post-frustration, post-rush
  - Refactoring Stoppers: time, fear, branching, focus
  - Teamwork: code review, conventional commit, pair programming
  - Split large refactoring in safe baby-steps using full IDE support
  - Refactoring Legacy Code: Mikado Method, Golden Master Testing (optional)
- **Code Smells** – overview of the most damaging micro-design issues today & fixing tactics:
- **Expressive Code**
  - Good names: explanatory variables, methods, and constants
  - Function signature: numbers of parameters, flags, defaults, extract parameter object
  - Function depth and cognitive complexity; guards
  - Single Responsibility (SRP) and Single-Level of Abstraction (SLAb) principles
  - The 3 rules of a clean switch
  - Error handling best practices
- **Clean Object-Oriented Programming (OOP)**
  - Missing Abstraction – identifying and extracting Value Objects
  - Encapsulation: move behavior next to state, enforce domain constraints
  - Primitive obsession, micro-types
- **Clean Functional Programming (FP)**
  - Pure Functions in practice
  - Immutable Objects: benefits and tradeoffs
  - Replacing loops with FP pipelines
  - Mastering pipelines operators: .map, .filter, .flatMap, …
  - FP anti-patterns and abuse: side-effects, tuples, FP chain wreck, complex .reduce, Try monad
- **Language-specific Best Practices** (spread throughout the content)
  - **Java**: Optional<>, checked exceptions, switch, mutability
  - **PHP**: type hints, collections, ?, default params
  - **Kotlin/Scala**: tuples, extension functions, ?
  - **Python**: type hints, named tuples, comprehensions, **, @dataclass
  - **TypeScript/JavaScript**: type vs interface vs class, config object, async,
- **IDE Refactoring Moves**: extract/inline variables, methods, parameters, constants & interfaces, etc…
- **Live-Refactoring** (1-2 hours): Victor on production code snippets from the audience
- **Code review** of selected prep-assignments (1 hour)
- **Hands-on** (2-2.5 hours): Participants refactoring in mini-groups
- **Other Debates,** triggered by the audience questions

Follow-up Refactoring Exercises (optional, after the workshop):
- **Java Stream Katas** (practice FP): https://github.com/victorrentea/kata-streams-java
- **Gilded Rose** (practice OOP): https://github.com/emilybache/GildedRose-Refactoring-Kata
- **Supermarket** (practice modeling): https://github.com/victorrentea/kata-supermarket-java
- **Catalog of exercises** (refactoring, TDD, unit testing): https://kata-log.rocks/refactoring

UNIT TESTING (2 DAYS)

This training aims to upgrade developers from *I have to write tests* to *Living a better life thanks to tests*. Constantly looking through the maintenance perspective, we go from immediate test-writing tricks and techniques to best practices in the design of tests, and up to what writing tests can tell us about the design of our production code. A deep discussion will revolve around best practices and pitfalls of using mocks in our tests. Along the way, we'll understand what Test-Driven Development and Behavior-Driven Development mean, as well as some special testing techniques like Golden Master, Characterization Tests, Approval Tests, and more. Then, at the end, we'll see how we can cover more code with wider-scoped integration tests using Spring Boot (if the group speaks Java).

Main Goals:
- Motivate developers to test thoroughly their code
- Understand the design insights provided by testing
- Review the state-of-the-art testing techniques of today

Ideal audience: Developers that have
- Written/fixed at least 100 Unit Tests
- Working knowledge with mocking frameworks (e.g., Mockito)
- At least 2-3 years of experience and/or bright-minded

(The content can be scaled down for less experienced groups)

Prep-work:
- **Read** about different levels of testing - Practical Test Pyramid
- **Coding Exercises** to practice writing unit tests and mocks **BEFORE** the training, (mandatory for those with less working experience) https://github.com/victorrentea/unit-testing-katas - (Java)
- [optional] **Watch** a Test-Driven Development Coding Kata: https://www.pentalog.com/pentabar/tdd-live-coding-kata
- **Send me samples of Unit Tests** (for dedicated sessions only)
  - 1-3 typical test classes **of ≤ 100-200 lines** + production code under test
  - **If making the extracted sample compile is too difficult,** don't send me any code code around it - I will create the missing code to make it compile
  - At least one participant should be familiar with the code.

Deliverable in: Java, Kotlin, Scala, PHP, TypeScript, or C#

Code: https://github.com/victorrentea/unit-testing

Agenda
- **Fundamentals**
  - Why test, why test-first, cost of tests, test-around-bugs
  - Key Concepts: Line/Branch Coverage, Mutation Testing, Test Isolation, Flaky Test
  - Testing Strategies: Pyramid vs Honeycomb
  - Types of tests: Unit Tests (solitary vs social), Integration Tests
  - Test anatomy, naming tests, single assert rule
- **Test-Driven Development** (TDD) – 2 hours
  - Rules of TDD, Red-Green-Refactor workflow
  - TDD Styles: Chicago("triangulation") vs. London("outside-in")
  - TDD Pros / Cons, TDD in real-life
  - **Exercise:** Classic TDD Coding Kata
  - Exercise: Outside-in TDD Coding Kata [optional]
- **Testing Techniques**
  - Creating test data
  - Expressive asserts with Assert4J (assertThat)
  - Parameterized Tests – uses/abuses
  - Using ".feature" files: best practices, Behavior-Driven Development (BDD/ATDD)
  - Testing Legacy Code: Characterization Testing and Golden-Master technique
  - Approval Testing
  - Subcutaneous tests, @VisibleForTesting
  - Nested test classes
  - Extending JUnit Framework – examples
- **Using Mocks** – 2 hours
  - Reasons to Love (3) and Hate (3) Mocks

  - Basic Features: stubbing, verification, argument matchers, captors, mocking statics…
  - Advanced/Dangerous Features: lenient, any, doReturn, spy, times, noMoreInteractions
  - **Exercise:** Covering Legacy Code with tests
- **Design Hints from Unit Testing** - 4 hours
  - What am I testing here? Syndrome
  - Precise method signatures
  - Object Mother (over stubbing getters)
  - Agnostic Domain Tests (aka Dependency Inversion Principle)
  - Split by Layers of Abstraction (over partial mocks)
  - Split unrelated complexity (over bloated shared test fixture)
  - Mock Roles, not Objects! (over fine-grained tests)
  - Decouple complexity, Functional Core/Imperative Shell segregation
  - Immutable objects (over temporal coupling)
- **Testing with Spring Framework** – 4-6 hours
  - Testing with DI: @Import, @MockBean, @Primary, profiles, test properties, @DirtiesContext
  - Testing with Persistence: in-memory DB, @Transactional tests, cleanup, @Sql, Testcontainers
  - Testing with External APIs: essentials of WireMock, recording proxy; vs MockServer
  - Testing your API: MockMvc, strategies, building custom testing DSLs, testing authorization
  - Tests with MQ
  - Slice tests (@DataJpa, @Web..)
  - OpenAPI-freeze approval test
  - **Exercise**: refactoring from a mock-based test to an end-to-end system test
  - Optimizing run time of integration tests
  - [optional] Consumer-Driven Contract Tests with Pact 1h
  - [optional] Spring Cloud Contract 1h
- **Code review** of unit tests code from projects of participants [optional]

SOFTWARE ARCHITECTURE (2 DAYS)

An architect has no simple, clean decision to take - everything is a trade-off with pros and cons to evaluate within the constraints of the project. But only after a decision is made, the real struggle starts - everyone in the team must learn to apply it in their daily work, and understand the WHY behind it in order to adjust that decision if the problem constraints shift. Join this deep dive into the mainstream backend architecture styles, understand pros and cons of typical design decisions, and learn how to gradually evolve your architecture to support more complexity.

The first day explains the foundational design principles, overviews key Domain-Driven Design tactics, and assembles the classic Layered Architecture showing what Evolutionary Architecture means in practice. The second day starts with a comparison of classical concentric architectural styles (Hexagonal-, Clean-, Onion-), along with common pragmatic simplifications. However, during more than half of day we study how to slice a monolithic application into modules, to tame the complexity that would otherwise eventually make the code unmaintainable.

The concepts will be demonstrated by refactoring 2 simple Java applications, explaining how modern frameworks simplify our work, keeping the focus on architectural decisions and tradeoffs.

After this workshop you will be able to:
● Develop a firm understanding of the core architectural principles.
● Learn to analyze tradeoffs of design decisions.
● Master the state-of-the-art backend architectural styles.
● Implement architectural changes incrementally, without large, risky rewrites.

This workshop is for you if:
● You are a developer that wants to become an architect.
● You are an architect or technical coach that wants an update on current trends in architectures.
● You are looking to debate emerging design practices you observed.

You can prepare in advance:
● **CLONE & IMPORT in IDE** the code (optionally try to solve in advance some TODOs in the readme.md):
1) https://github.com/victorrentea/clean-architecture and
2) https://github.com/victorrentea/spring-modulith
● **READ** a compilation of architectural styles: https://herbertograca.com/2017/11/16/explicit-architecture
● **READ** an influential article https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
● **WATCH** introduction to Modular Monolith: https://youtu.be/Xo3rsiZYsJQ
● **STUDY** a diagram: https://docs.google.com/drawings/d/1BzSoiYaOSD-A49w8X4MMx7X717KJdEhnb-BF5uU9h6o
● **THINK**: compile and send me before the training a bullet list of topics you'd like to debate🔥.
● **READ:** the links in the agenda below

Agenda:

- **Fundamentals**
  - Principles: SOLID, DRY, KISS & CUPID
  - Complexity vs Coupling vs Data
  - Constraints, NFRs, tradeoffs, and "it depends"
  - Documenting Design: UML, ADR, C4
  - Role of an Architect. How to become one?
- **The Domain Model**
  - Ubiquitous Language and Bounded Context (Domain-Driven Design)
  - Value Objects for a Supple Domain Model
  - Rich Domain Model with behavior and constraints
  - Debate: is ORM a friend or a foe?
  - **Exercise**: refactoring Domain Model from 'anemic' to 'rich'
- **Exposing your API**
  - Contract- vs Code-first API development
  - The Dark side of Automatic Mappers
  - Command-Query Responsibility Segregation (CQRS)
  - Task-Based UI
  - API Versioning Strategies, SemVer
- **Consuming their API**
  - Adapter Pattern and Dependency Inversion
  - Enforce boundaries with Architecture Unit Tests
  - **Exercise**: extract infrastructure concerns out of core logic
- **Layered & Evolutionary Architecture**
  - Over-engineer vs Under-design
  - Application Service (in DDD), and Separation by Layers of Abstraction
  - Vertical-Slice Architecture (VSA)
  - **Exercise**: push complexity in Domain Services
- **Concentric Architectures**
  - Hexagonal Architecture (aka Ports-and-Adapters)
  - Onion Architecture
  - The 5 Rules of a Clean Architecture
  - Criticisms and Pragmatic Simplifications
  - **Quiz**
- **Finding Service Boundaries** - Heuristics
  - Screaming Architecture – partition code first by business capability, not by layers
  - Bounded Contexts, Team Size, Code Volatility, Conway's Law
  - Platform Team (as in Team Topologies)
- **Modular Monolith Architecture** (Modulith)
  - What is a "Module"
  - Module contract: internal/external, calls/events/plugins, encapsulation
  - Module data decoupling stages
  - Module interaction patterns, breaking cyclic dependencies
  - Extracting a module as a microservice (for the right! reasons)
  - **Exercise:** Extract a new functional module in an application using Spring Modulith

PRAGMATIC DOMAIN-DRIVEN DESIGN (2-3 DAYS)

This training includes the Software Architecture topic (see above), which covers many DDD lessons. In addition, a DDD Workshop would also include:

Prep-work: Participants with no prior contact with Domain-Driven Design philosophy should read
- DDD Terms: read at least pages 1-25 from: https://www.domainlanguage.com/ddd/reference/
- Domain-Driven Design Distilled *by Vaughn Vernon* (on Amazon), or
- DDD Quickly *by InfoQ*: https://www.infoq.com/minibooks/domain-driven-design-quickly/ (free)

Code sample using Aggregate design: https://github.com/victorrentea/ddd

Agenda (in addition to Software Architecture)**:**
- **DDD Philosophy** (1-2 hours)
  - DDD Philosophy, when to use/avoid, typical misconceptions
  - Ubiquitous Language – why it matters
  - Key values of DDD: Biz Collaboration, Domain Distillation, Exploration, and Bounded Contexts
  - DDD Pitfalls: DDD God Fallacy, overengineering, internal framework mania, Tech-tiger bias
  - "Light DDD": 10 Takeaways applicable in any non-DDD Project
- **Event Storming Workshop** (2 hours), including PO/BA/DE
  - Identifying Pivotal Events, Commands, Views, and Aggregates
  - Experiencing Collaborative Domain discovery
- **Aggregate Design** (3-4 hours)
  - Aggregate as the consistency boundary for changes
  - Domain Events, Event Sourcing (preview)
- **Java & Framework tricks:**
  - Hibernate tricks (consistent @Entity, immutable @Embeddable, ID Type)
  - [Java] Spring tricks: @DomainEvents, Stereotypes, Aspects, Spring Data JPA Specifications

Microservice Patterns (2 days)

This workshop covers the core principles of microservices, their advantages, design, and communication patterns. Attendees will learn how to switch from traditional single systems to building distributed systems and how to keep them running smoothly. The training also touches on maintaining consistency, message-based integration, and event-driven architectures.

Agenda

- **Fundamentals 10m**
  - Definition and History of Microservices
  - Benefits and Drawbacks of Microservices
  - CAP Theorem & Eventual Consistency
  - Integration Paradigms: RPC, Messaging, Shared DB, Files
- **Finding Service Boundaries –** A Gallery of Heuristics, including
  - Technical vs Functional partitioning (aka Screaming Architecture)
  - Business Capability vs Data
  - Bounded Contexts for a sharp Domain Model
  - Conway's Law
  - Cognitive Load and the Platform Team (Team Topologies)
- **Migrating from Monolith to Microservices**
  - Strangler Pattern
  - Modular Monolith Architecture
- **API Design and Management**
  - REST APIs, Swagger/OpenAPI, GraphQL (brief)
  - Versioning Strategies (semver.org)
  - Contract Testing & Automation: Pact.io and Spring Cloud Contract
  - API Gateway & BFF Pattern
  - Service Discovery
- **Resilience Patterns**
  - Catastrophic Failures examples
  - Fallback Patterns
  - Identifying failure units
  - Timeouts, Retry, Idempotency, and Circuit Breaker
  - Load Throttling: Concurrency and Rate Limiter
  - Load Balancing and Auto-scaling
  - Sidecar pattern (Istio)
- **Message-Based Integration**
  - Messaging vs REST (or other RPC) – use-cases
  - Messaging Features: persistent q, priority q, DLQ, consumer groups
  - Messaging Patterns: fire-and-forget, request-reply, claim check
  - Events vs Commands
- **Event-Driven Architectures**
  - Four types of Event-Driven Architectures (talk)
  - Kafka fundamentals and common patterns
  - Event-Sourcing: patterns and challenges
  - **Round table**: event-streaming war stories
- **Distributed Consistency**
  - Outbox/Inbox Pattern, Change Data Capture (Debezium)
  - Choreography vs Orchestration, Smart Endpoints and Dumb Pipes philosphy
  - Compensations, Reservations, Saga Pattern
  - **Exercise**: designing a food delivery saga
- **Operating & Debugging**
  - Health Checks & Liveness-Management
  - Request tracing & centralized logging
  - Metrics: Prometheus/Grafana
  - Distributed tracing bottlenecks, eg with Zipkin
- **Security**
  - Authentication / Authorization Strategies
  - Top 10 API Vulnerabilities (OWASP) - optional
  - Secret Management
- **Testing**
  - Honeycomb Testing Strategy
  - End-to-end testing: who owns the service mocks?

DESIGN PATTERNS AND PRINCIPLES (2 DAYS)

Mastering Design Patterns helps you understand existing code, identify, and compare design alternatives, and better communicate your ideas with your colleagues/architects. Since developers are fluent in code, the workshop is heavily code-driven, but the focus stays on the evergreen design principles and values. In the agenda, you'll find a range of patterns from simple ones like Singleton, Factory Method, or Strategy, up to strategical ones like Visitor and Saga Pattern, but don't expect to hear the dogmatic definition of them. Instead, we'll explore together pitfalls, tradeoffs, and alternative designs that will fill your toolbox for writing better, simpler code in your actual project. Along the way we'll also explore patterns (like Proxy) that underly mainstream framework today, as well as inter-system collaboration patterns, like commands and events, choreography, and orchestration. One of the most important takeaways, however, will probably be the Clean Architecture (aka Onion), implemented with a pragmatic code-first mindset.

By the end of this live online course, you'll understand:
- The most important Design Patterns in use today
- The principles underlying these Patterns
- The benefits and tradeoffs that Patterns introduce

And you'll be able to:
- Identify design patterns applicable to real-life problems, and weigh their pros/cons
- Find a simple solution to the problem at hand after considering design alternatives and evolution paths
- Read complex code faster and deeper understand the frameworks and libraries you use
- Refactor code in baby-steps towards Design Patterns

This Online Training is for you because…
- You can read Java and you want to upgrade your design skills
  (* coding can also be performed in PHP, C#, TypeScript, Scala, or Kotlin)
- You are considering an Architect career path
- You want to design a new system, or just review the design of your current one

If you want the best of this training, you can prepare in advance by:
- (MUST) **READ**: 5-10 minutes about each pattern in the agenda below from:
  - This website: https://refactoring.guru/design-patterns or
  - This book: Head First Design Patterns
- (RECOMMENDED) **CODING**: apply as many design patterns as possible you can to this exercise:
  - https://github.com/victorrentea/design-patterns/blob/master/src/main/java/victor/training/patterns/assignment/
- (OPTIONAL) **READ** (15 minutes): this very influential article about Clean Architecture:
  https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

⚠ **IMPORTANT**: The patterns listed in the agenda target a **backend developer** using Java/Kotlin/Scala (± Spring), C#, or PHP. For developers developing Web Frontends (Angular, React..), Games (eg Unity) or Scripts (eg python), we can tailor the agenda for their needs during a prep-call. Even if the principles stay the same, their materialization in patterns is highly dependent of the application challenges, language and frameworks used.

Code: https://github.com/victorrentea/design-patterns

The agenda is fully configurable to suit your team's needs.

- Strategy (OOP) vs FP alternative vs Chain of Responsibility (aka “Filters” Pattern)
- Template Method (OOP) vs Passing-a-Function (FP) + variations (Loan Pattern)
- Visitor Pattern vs Matching Sealed Classes (Java21,Scala,Kotlin)
- State Machines & the State Pattern

Intercepting Calls – 1.5 hours

- Decorator, Proxy (OOP), Aspect Oriented Programming (AOP) and Execute-Around (FP)

Creating Objects – 1.5 hours

- Factory Method
- Singleton, Dependency Injection, Object Pool - overview
- Builder – pattern or anti-pattern?

Propagating Change - 2 hours

- Observer Pattern: in-memory (eg Spring Events) or messaging
- Command Pattern: in-memory (eg tasks to thread pool) or messaging
- Observer vs Command

Wrap-up – 30 min

- Anti-Patterns + typical workarounds
- Recap & Quiz.

MODERN JAVA PATTERNS (1-2 DAYS)

Discover the best new features of the Java language and learn modern design patterns enabled by them.

Optional Preparation:

- The participants are assumed to be already fluent with the Java 8 lambda/Stream syntax. To fill any gaps you can watch this Java 8 Stream API screencast on YouTube, then solve these exercises: https://github.com/victorrentea/kata-streams-java.

Agenda

- **Functional Programming**
  - Advanced Stream use-cases
  - Syntax quirks: method references, target typing, effectively final
  - Exceptions: wrapping as runtime and the Try monad
- **Fighting NULL**
  - Null Object pattern
  - Optional best practices & abuse
  - Annotations
- **Immutability**
  - Records (17) and Data-Oriented Programming (DOP); best practices and limitations
  - Do we still need Lombok?
  - Immutable collections
- **Intercepting calls**
  - Proxy pattern (OOP)
  - Aspect-Oriented Programming (AOP) – Spring example
  - Execute-Around pattern (FP)
- **Forking behavior**
  - Strategy pattern
  - Attaching behavior to enum
  - Return-switch-enum pattern (17)
  - Filter pattern
- **Fill behavior bits**
  - Template Method pattern (OOP)
  - Loan pattern (FP)
- **Behavior per subtype**
  - Visitor pattern (OOP)
  - Switch on sealed classes (21)
- **Concurrency**:
  - Parallel stream (8) pitfalls
  - CompletableFuture (8)
  - Virtual Threads (Project Loom) (21)
  - Structured Concurrency (25)
- **Strings**: formatted, text blocks (17), interpolation (±24)
- [optional] Interfaces with default (8) and private (17) methods – use-cases
- Platform improvements: GraalVM and nativeimage, super-fast Garbage Collectors
- Future of Java Language

SPRING FRAMEWORK (3 DAYS)

The duration can be reduced to 2 days if we remove some chapters from the agenda.
Must-have a prep-call to tailor this long agenda to what your team really needs / uses. 👌

Target audience:
a) Entry-level developers with no/little prior contact with Spring
b) Experienced developers with hands-on experience with Spring
⚠ For the best experience, try to avoid mixing people from both categories in the same group.

Prep work:
- [MUST] Prepare questions, debate topics, and challenging code samples from your project
- [MUST] Clone the project and mvn install it from https://github.com/victorrentea/spring
- Juniors can get familiar with the basics via a video course on Udemy/Pluralsight/Coursera …
- More experienced developers can try out samples from baeldung.com covering topics in the agenda

Agenda
- **Spring Container** (2h)
  - Defining Beans: component-scanning, @Import, and @Bean
  - Dependency Injection Styles + Lombok tricks
  - Bean Lifecycle @Scope
  - Spring Profiles and ConditionalOn…
- **Events and Startup** (.5h): @EventListener best practices, initializers
- **Configuration** (.5h)**:** @Value, @ConfigurationProperties, config sources, Cloud Config Server
- **Spring Boot** (.5h):
  - Library version management, starters
  - Convention over Configuration: AutoConfiguration, devtools
- **Aspects** 1h: concepts, proxy implementation, pitfalls, writing an @Aspect
- **Transaction Management** 2-4h
  - Database ACID transaction, anomalies
  - @Transactional mechanics: propagation, rollbacks, read-only tx, after-tx hooks
  - Best-practices and pitfalls using Transactions, tuning performance [opt]
  - JPA integration: write-behind, auto-flushing, lazy-loading, and 1st level cache
- **Multi-threading**: Spring thread pools, @Async, and non-blocking REST APIs; @Scheduled
- **Caching**: annotation and programmatic, ⚠Pitfalls (stale cache, distributed cache, hit/miss ratio)
- **REST Endpoints** 2h
  - REST API design best practices, validation, documentation
  - @RestControllerAdvice
  - [optional] Introduction to WebFlux 1h *(see my Reactive Programming training for a deep-dive)*
  - [optional] WebSockets: concepts, debug, queue/topic, security, common patterns, testing
- **Observability** 1h
  - Spring Actuator features, adding endpoints and health checks
  - Exposing metrics with Micrometer
- **Spring Security** 2-4h
  - Spring Security Architecture, Custom Security Filters
  - Authentication via: User-Password login form, Basic/api-key, Preauth Headers, JWT Token
  - Authorization: URL pattern vs annotations, role vs feature-based authz, data-security, testing
  - Spring OAuth2 integration (example using KeyCloak) – prep video: Intro to OAuth
  - *(For more please see my 'Secure Coding' training)*
- **Messages on Queues** 1-2h
  - Concepts: durability, topic vs queue, reply queue, consumer group, tracing, DLQ, correlationID
  - Spring Cloud Stream (the new Functional-Style)
  - [optional] IntegrationFlows DSL
- **Testing** 4h+
  - Testing with DI: @Import, @MockBean, @Primary, profiles, test properties, @DirtiesContext
  - Testing with Persistence: in-memory DB, @Transactional tests, cleanup, @Sql, Testcontainers
  - Testing with External APIs: essentials of WireMock, recording proxy; vs MockServer
  - Testing your API: MockMvc, strategies, building custom testing DSLs, testing authorization
  - OpenAPI-freeze approval test
  - **Exercise**: refactoring from a mock-based test to an end-to-end system test
  - Optimizing run time of integration tests
  - [optional] Consumer-Driven Contract Tests with Pact 1h
- **Spring Batch [optional] 2-3h**

    - Concepts: Step, Job, Listener, Execution Context, Transaction control, Resume
    - Using Listeners to track metadata
    - Fine-Tuning Performance: Chunking, Parallel Steps, Multi-Threading processing
- **Wrap-up:**
    - Real-life scenarios - Brainstorming
    - Spring Overview: main features, strengths and weaknesses, best learning sources

SECURE CODING IN JAVA (2 DAYS)

Overview

This workshop starts by reviewing the essential cryptography and web-security concepts and techniques. The top web attacks identified by OWASP (https://owasp.org/) are then explained and fixed with Spring Framework in several ways comparing the tradeoffs of each option. We'll then dive into the two main coordinates of security: authentication and authorization and we'll explore the mainstream practices as well as some advanced use-cases that will lead us to discover many details of the Spring Security framework.

This Training is for you because:

- You had security incidents in production.
- You got a pen-test/security audit, and you want to prevent the issues discovered from recurring.
- Security is an early concern in your project, due to its sensitive nature.
- You are looking for a review of your team's secure coding practices.

Prerequisites: Prior contact with security concerns is recommended

Prep-work:

- **Read** a bit in advance about the topics in the agenda that interest you the most. (request links)
- **Watch** Introduction to OAuth: https://www.youtube.com/watch?v=996OiexHze0

The agenda will be adapted for the technical stack / security techniques in use, ideally guided by the client pen-testing findings.

Agenda

- **Cryptography** Fundamentals (2 hour)
  - Hashing, Salting, bcrypt, Encryption (symmetric / asymmetric), Digital Signatures
  - Certificates, Certificate Authorities, self-signed certificate, keytool overview
  - [optional] Java workshop experimenting all the above topics (raw)
  - TLS, mTLS; [optional] Exposing https endpoints on a Spring Boot App
  - Web Sessions and Cookies
  - Security Architecture Overview: DMZ vs Intranet, WAF, Security Proxy
- **OWASP Web App Top 10** Vulnerabilities (link)
  - Cross-Site Request Forgery (CSRF)
  - Cross-Origin Resource Sharing (CORS)
  - Injection of SQL/NoSQL/JPQL, javascript (XSS, CSP), OS commands, url parts (SSRF)
  - Risks of handling files: upload, zips, XML/YAML attacks, insecure deserialization, viruses
  - **Vulnerable Dependencies** (CVE): fixing strategies; security scanning tools
- **OWASP API Security Top 10** Vulnerabilities (link)
  - Broken Function- or Object- Level Authorization
  - Excessive Data Exposure
  - Lack of Resources & Rate Limiting
  - Mass Assignment
  - Flaws in Configuration, Monitoring, and Deployment
- **Securing Applications with Spring**
  - Authentication: Form Login, Basic, API Token, Pre-Auth headers, JWT token
  - Function-Level Authorization: URL-patterns and annotations, Role- vs Authority- based
  - Spring Security Architecture, writing a custom filter, debugging
  - Object-Level Authorization: data jurisdiction, permission evaluator, data visibility
  - Unit-Testing Backend Authorization
- **OAuth2** (4-5 hours)
  - Tokens, Actors, Front- vs Back- channel, Single-Sign On (SSO)
  - Flows: Authorization Code, Client Credentials, ~~Implicit Flow~~ Authorization Code+PKCE
  - Social Login (eg "login with Google"), OpenID Connect
  - Workshop: Spring OAuth using KeyCloak
  - [optional deep-dive] Example attacks on OAuth; dissecting the exchanges (2 hours)

JPA/HIBERNATE PERFORMANCE TUNING (2 DAYS)

Everyone thinks they know Hibernate. And then they have their first performance issue. Or they try to deeply refactor their JPA model. And then, they search for this training.

By the end of the live online course, you'll understand:
● Why Hibernate is by far the most complex Java library in broad use
● The internal behavior of Hibernate that has an impact on performance
● The exact lifecycle of an Entity
● How Hibernate controls transactions

And you'll be able to:
● Locate the performance bottlenecks
● Fix the root cause of performance issues and improve the overall performance of your application
● Write highly efficient read queries
● Optimize writes to DB and batches
● Design safe and performant Entity models with Hibernate

This Online Training is for you because…
● You are an experienced professional with strong practical experience with Hibernate
● [ideally] You had prior contact with tuning the performance of the interaction with a Relational Database.

Prep work (optional):
- **ORM Exercise** ("fill in the annotations"): https://github.com/victorrentea/jpa-assignment/
- **Problems from the project:** send me a series of questions / topics to debate

Code: https://github.com/victorrentea/jpa

Agenda:
- Chapter 1: **Efficient Searching**
  - Lazy Loading: pitfalls and best practices
  - N+1 Queries Problem + solutions; debate
  - Paginated queries: pitfalls, best practices, UX Tricks
  - Dynamic Queries with jpql+=, criteria, metamodel, specifications, fixed JPQL, and QueryDSL
  - Tuning JPQL and SQL – practical hints
- Chapter 2: **Designing a Clean, Performant Entity model**
  - Primitive links vs JPA links
  - Persisting enums via Hibernate type mappers
  - Bidirectional JPA Links – best practices
  - Rich Deep Modeling using @Embeddables, ElementCollections and safe-guarding entities
  - Advanced mapping: @MapsId, Composite PKs
  - List<> vs Set<> of Children; implementing hashCode on Entities
- Chapter 3: **Efficient Transactions**
  - Spring @Transactional: mechanics, propagation, error handling, after commit hooks
  - Persistence Context as 1st Level Cache and Write Buffer (aka write-behind)
  - Auto-flush dirty entities: embrace or reject?
  - How to detect and avoid Connection Starvation issues
  - Transaction management best practices
- Chapter 4: **Batch Inserting**
  - Generating Primary Keys efficiently
  - getReference()
  - Batching inserts
  - High-performance Spring Batch jobs – hands-on (on demand)
- Chapter 5: **Caching (optional)**
  - Enabling Second-level Cache for Entities and Queries; tuning
  - Comparison with Spring @Cacheable
- Chapter 6: **Working with Large Files**
  - Domain modelling alternatives
  - Streaming CLOB/BLOB data
- Chapter 7: **Updating Entities**
  - Mastering .merge() operator
  - Concurrency Control using optimistic and pessimistic locks
  - [opt] Bulk update JPQL queries - pitfalls

JAVA PERFORMANCE (3 DAYS)

A deep dive in Java Concurrency, Memory Management, Profiling, and Benchmarking, spiced with under-the-hood details of the JVM and debates around typical performance issues of Java projects. The workshop is packed with experiments distilled from real-life projects to explore and demonstrate different performance issues.

Prerequisites: Solid knowledge of the Java Language. Ideally: prior contact with performance issues.

Prep-work:
- Clone + Import Profiling Project: https://github.com/victorrentea/performance-profiling
- Install tools: VisualVM, Java Mission Control
- Clone + Import Main Project: https://github.com/victorrentea/performance
- Think of performance issues you encountered by now; ask your friends for more ☺
- [optional] Study in advance heap-dumps of some memory leaks, available here
- (optional, if using in project) CompletableFuture Self-study: Intro Video and Workshop Code

Agenda:
- **Introduction:**
  - War stories from participants (round table)
  - Key Concepts, Metrics and Questions
  - Principles and Strategies to Improve Performance
  - Common bottlenecks of backend systems

- **Tracing the Bottleneck**:
  - Zooming in on the bottleneck via Distributed Tracing and Zipkin
  - Understanding and using Micrometer Metrics,
  - Execution profiling with Java Flight Recorder via Glowroot, Java Mission Control, and IntelliJ
  - Load testing principles and techniques: Gatling, jMeter, K6
  - ***Exercise:*** Understand and use a flame graph to trace bottlenecks in a sample app

- **Multi-threading**
  - Thread Pools mechanics, queue size, thread count, usage patterns
  - Multi-threading Risks**:** race bugs, deadlocks, thread pool starvation
  - Concurrency Control: synchronized, atomic primitives, synchronized+concurrent collections
  - [upon request] Concurrency Primitives: Lock, Semaphore, CyclicBarrier, wait-notify
  - ***Exercise***: Designing a thread-safe concurrent workflow
  - Non-blocking concurrency with CompletableFutures, Reactive and Virtual Threads (java21)
  - Spring Framework support: ThreadPoolTaskExecutor, non-blocking HTTP Endpoints, @Async
  - ***Exercise***: parallelizing a non-blocking REST API
  - Parallel streams: mechanics, best practices, pitfalls

- **[optional] CompletableFuture** deep dive workshop (4h – 8h with participants hands-on) - code
  - Combining, Chaining
  - Exception handling
  - Controlling Parallelism
  - Best Practices and Common Patterns for Non-blocking Flows
  - Testing & Debugging
  - Advanced: Tracing (ThreadLocal), Monitoring (Micrometer) & Profiling (JFR)

- **Memory Management**
  - Java Memory Model: Thread Stacks, Metaspace, Heap (old/young/survivor), TLAB
  - Garbage Collector: Key concepts, Monitoring, GC Algorithms
  - Techniques for using less memory
  - Thread Local: best practices, propagation over thread pools, pitfalls
  - Heap Dump analysis: retained/shallow heap, GC Roots, profiling allocations
  - ***Exercise***: Tracing ten types of Memory Leaks (find some heapdumps here)

- **Caching [optional]**
  - Principles, cost-benefit-risk of caching, core parameters
  - What & Where to cache data: comparison
  - Maintaining cache consistency: eviction, TTL, distributed cache, best practices

- **Just-In-Time (JIT) Compiler Overview**
    - JIT mechanics, dynamic optimizations, writing JIT-friendly code
    - GraalVM nativeimage
    - ***Exercise:*** Writing micro-benchmarks with Java Measuring Harness (JMH)

- **Tuning JPA Performance 1-2 days** (not included ⚠) -- *see the JPA topic in my agenda.*
    - *In many Java applications, DB and JPA usage becomes a bottleneck.*

REACTIVE PROGRAMMING (3 DAYS)

Deliverable in: Java or Kotlin; This training can cover:
- ➢ Project Reactor + Spring WebFlux, and/or
- ➢ rxJava - Android/UI flavored

Overview:
The first challenge when approaching reactive programming is understanding WHY and WHEN NOT to use it. Starting from 'classic' blocking web endpoints, we'll benchmark the bottleneck that occurs under heavy load and then explore the different historical alternatives to Reactor. Besides exploring dozens of operators, the focus will remain on understanding the signals that drive all reactive flows, as this is the key to unlocking the mysteries of backpressure, hot publishers and avoiding common pitfalls when starting to work with Reactor. We will then use this knowledge to approach several typical reactive flows, drawing conclusions about the best way to write the code to be maintainable and safe. After a roundtrip of Spring WebFlux features and quirks, we'll then talk about 2 tough tasks: testing reactive flows and progressive migration of blocking code to reactive flows, and if there's time left, about performance tuning.

This training is for you because:
- Your project uses Spring 5 WebFlux
- You want to optimize a heavy-loaded API or a heavy batch workflow
- You had bugs and long headaches when trying to use Reactor in your project
- Or simply you just want to master the most complex programming paradigm in the world

Prerequisites:
- Fluency with Functional Programming concepts: .filter() .map(), lambda syntax, immutable objects
- Prior contact with multi-threaded code

Preparation:
⚠ This is a very challenging workshop ⚠
It's imperative to **allocate 2-8 hours to go through the prep work**, alone or in pair programming (ask your manager)
- **The Main Workshop**:
  During the workshop, we will spend a lot of time solving some prepared problems (eg C1_Creation.java) for which we'll try to pass some pre-written unit tests (eg C1_CreationTest.java). Please🙏 spend several hours on that code in advance to get more familiar with the concepts we'll be juggling with for long hard hours.

- Alternatively, check out the ProjectReactor's official **lite workshop** with more explained theory is this (est work time ≥ 3-4 hours): https://tech.io/playgrounds/929/reactive-programming-with-reactor-3/Intro
- If you get stuck, you can find the solutions on our project's Git (eg: 'Part01FluxSolved.java')

- **A good workshop** (free slides and code): https://github.com/nurkiewicz/reactor-workshop

Code: https://github.com/victorrentea/reactive

Agenda
- **Introduction**
  - [optional, basics] Recap: functional pipelines with Java8 Streams
  - Blocking REST Endpoints: trace a thread pool starvation issue
  - Alternatives to Reactive: CompletableFuture, coroutines(kt), Virtual Threads (Project Loom)
  - Reactive Streams Spec
  - Mono, Flux
  - Understanding Marble Diagrams
  - Signals: next, complete, error, request, subscribe, cancel
- **Core Operators**
  - Immediate: `just, empty, error, never, range, fromIterable`
  - Delayed: `delay, interval, delayElements`
  - Immediate Operators: `distinct, filter, map, first, take, contains, all`

- **Enriching Data**
  - `flatMap` vs `concatMap` vs `flatMapSequential`
  - Batching requests: `buffer`, `bufferWindow`
  - Handling Empty: `defaultIfEmpty, switchIfEmpty`
  - Async filtering with `.filterWhen`
  - Parallelizing work: Tuples, `zip`, `zipWith`, `zipWhen`
  - Use-case Context design pattern
- **Performing Side Effects**
  - Fire-and-forget using doOnNext
  - Parallelization, cancellation
- **[Hands-on] Migrating a blocking codebase to Reactive Programming**
  - Schedulers: CPU vs IO, best practices, ParallelFlux
  - Calling blocking code in a safe way on boundedElastic scheduler
  - CompletableFuture integration
  - Bridging to a callback-based model: Sinks
  - Reactive REST: Endpoints, and WebClient
  - Reactive NoSQL: Mongo, Cassandra, Redis
  - Reactive SQL: R2DBC
  - Reactive Messages: send/receive
  - Kafka Streams [optional]: KStream, KTable, Spring Cloud Stream Functions
- **Infinite Fluxes**
  - Preventing cancellation with onErrorContinue
  - GroupedFlux
  - Time series operators: scan, reduce, window, buffer, sample, merge, concat, combineLatest
  - **Hot vs Cold Publisher**
  - In-memory broadcast using Sinks
  - Caching Hot Publishers: `replay, cache, connect, autoConnect`
- **Resilience**
  - Error handling patterns
  - Timeout and Retry
  - Back-pressure
  - Throttling: rate-limiting, resilience4j integration
  - Challenges of distributed systems under heavy load - brainstorming
- **Testing**
  - StepVerifier vs .block()
  - TestPublisher
  - PublisherProbe + custom extension
  - Detecting blocking code with BlockHound
  - Controlling Virtual Time [optional]
- **Debugging, Monitoring and Tuning**
  - Propagating metadata via Reactor Context
  - Checkpoints and Hooks.onOperatorDebug
  - Monitoring using `.metrics, .elapsed, .timed`
- **Common Pitfalls and Workarounds**
  - Not subscribing / Resubscribing to Cold Publisher
  - Loosing Data Signals (empty)
  - .subscribe()
  - Blocking in non-blocking threads
  - Unexpected Cancelation
- **Review of Code** Sample from the audience

ONE-TO-ONE COACHING OR CONSULTANCY

Intense one-to-one interactive sessions of 1-2 hours with 1 developer or a small team (max 5). The most time-efficient way to grow your (team's) skills.

Activities can include:

- Pair Programming sessions.
- (optional) Home assignments between sessions.
- Code/Architecture Reviews.
- Tuning performance or tracing very complex bugs.
- Career orientation.
- Discussions on any concepts from my training curricula above, but also new custom topics.

Important: The hourly fee is considerably lower (≅1/3) than the training fee.
The timeslots are booked 1-2 weeks in advance, during lunch time.

LEGAL ENTITY DETAILS

- **Company Name**: Victor Rentea Consulting SRL
- **Fiscal Address**: Dristorului 91-95, Ap 1106, Bucharest 031538, Romania
- **VAT Code** (CUI): RO41987600
- **Romanian Commerce Registry Number:** J40/16655/2019