



# Clean Code

Duration 3 day(s) (CLEAN-CODE-03)

Design and write a clean code, improve an existing code

## Description

The introduction of agile methods has made it easier to build software that meets real needs, and to improve the delivery process. The clean code is a complementary set of technical practices, to ensure the durability of the software built. A clean code is indeed the sine qua non of a robust software (low maintenance cost) and scalable (can be adapted to new needs).

## Goals

- Deepen object-oriented programming principles to write state-of-the-art code
- Realize and put into practice these principles with simple and striking examples
- Put under control the technical debt by refactoring
- Learn techniques specific to legacy applications
- Acquire a synthetic vision of the most used clean design methods

## *Public*

- Developers - Architects

## *Prerequisites*

- Experience in Java

## *Structure*

50% Theory, 50% Practice

## Program

# The software quality imperative

- Consequences of insufficient quality
- The infernal cycle: writing, rewriting
- The concept of technical debt
- Tools and processes: necessary, but not sufficient
  - Quality control tools
  - Process and formatting
  - Limitations
- What is clean code?

# Principles of clean design

- General principles
  - Founding principles of OOP
  - The four principles of Kent Beck
  - Importance of naming
  - Common sense by acronyms: YAGNI / KISS / DRY / POLA
  - Some functional programming principles
- Minimize coupling, maximize cohesion
  - SOLID principles
  - Cohesion and coupling
  - Stability and instability
- Supple design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions
  - Defensive Programming
  - Conceptual Contours
  - Standalone Classes and Closure of Operations
  - Declarative Style of Design
  - Bonus: the principle of symmetry

# Improve the quality of existing code: smells and refactors

- The concepts of smell and refactor
- The smells of Martin Fowler
  - Duplicated method
  - Duplicated class
  - Long method
  - Long class
  - Primitive obsession
  - Brief overview of other smells
- Java effect
  - Equality .. or not
  - Immutability with holes

- Instant obsolescence
- Hide this exception that I can not see
- ArrayList obsession
- Weak design
  - Technical Modules: Service-Dao-Entity
  - Generate getters and setters
  - Death by nesting: the devil's staircase
  - Packages unpacked
  - To be or not to be: the inappropriate relationship Is-not-a
  - Javadoc and alibi tests
  - The editorialist: the intelligence buried in the comments
  - The night of living codes
- Properly modify a legacy application

# Panorama of other clean design methods

- Test-first design at the service of quality
- Software Craftsmanship
- Standard design bricks
- The business domain as the core of the software: the Domain-driven Design approach