sages

Course code: **REFAKT**

Course title: **Writing better code by using refactoring techniques and design patterns**

Days: 3

Description:

Course intended for:

The training provides a thorough understanding of refactoring techniques and the context, in which they should be used. It starts with a discussion on the code quality and methods, which allow us to determine that the source code is of low quality. Afterwards, the participants are presented with the rules that the programmers should use during their work in order to aim at development of high quality code. The main part of the training consists of workshops on refactoring techniques (including method composition, simplification of conditional expressions) and design patterns based on a GoF set (Gang-of-Four).

Course objective:

- To be able to assess the quality of the source code used,
- Indicate imperfections in the code, name them and state why they exert negative impact on the application quality,
- Understand different refactoring techniques and be able to use them on a low-quality code,
- Understand the context of use of a given design pattern and be able to implement it.

Requirements:

The training participant should have basic experience in object-oriented programming. The preferred language is Java.

Course parameters:

3*8 hours (3*7 net hours) of lectures and workshops in the following proportion: 80% workshops, discussions; 20% - lectures.

Group size: no more than 8-10 participants.

Sages sp. z o.o., ul. Nowogrodzka 62c, 02-002 Warszawa
tel. +48 22 203 56 00 fax: +48 22 203 56 01
e-mail: office@sages.io web: www.sages.io

sages

Course curriculum:

1. Introduction
    - Design pattern – a definition
    - Are design patterns an answer to the shortcomings of a given programming language?
2. The code quality and its assessment
    - How to measure the quality of a source code?
    - Code Smells
        - Code duplication
        - Methods based on use of a great number of parameters
        - Long methods
        - Huge classes
        - God Class
        - Dependence on implementation details of a different class
        - Use of names, which do not describe anything
    - Anti-patterns
        - Cut-and-Paste Programming
        - Spaghetti Code
        - Programming to Implementation
        - Tight coupling
        - Golden Hammer
        - Poltergeist
        - Boat Anchor

        - Dead End
        - Ambiguous Viewpoint
        - Lava Flow
        - Mushroom Management
    - Code quality improvement
        - (Missing) tests of components undergoing changes
        - The so-called Clean Code
        - Encapsulation
        - The principle of responsibility
        - Composition above inheritance
        - Programming through interfaces
    - Technical debt
3. Refactoring techniques
    - Introduction
    - Development of methods
        - Introduce Explaining Variable
        - Split Temporary Variable
        - Replace Template with Query, Inline Temp, Inline Method and Extract Method
        - Remove Assignments to Parameters
        - Substitute Algorithm
    - Method call simplification
        - Replace Constructor with Factory Method, ErrorCode with Exception, Exception with Test, Parameter with Explicit Methods, Parameter with Method,

        - Introduce Parameter Object
        - Encapsulate Downcast
    - Transfer of properties between objects
        - Move Method and Move Field
        - Extract Class
        - Inline Class
        - Hide Delegate
        - Remove Middle Man
        - Introduce Foreign Method
        - Introduce Local Extension
    - Data organization and modeling
        - Replace Array with Object, Data Value with Object, Magic Number with Symbolic Constant, Record with Data Class, Subclass with Fields, Type Code with Class, Type Code with Strategy/State, Type Code with Subclasses
        - Encapsulate Collection, Encapsulate Field
        - Change Bidirectional Association to Unidirectional, Unidirectional Association to Bidirectional, Reference to Value, Value to Reference
    - Simplification of conditional expressions
        - Decompose Conditional Expressions, Consolidate Conditional Expressions and Consolidate Duplicate Conditional Expressions
        - Remove Control Flag
        - Replace Nested Conditional with Guard Clauses
        - Replace Conditional with Polymorphism
        - Null Object
    - Generalizations

        - Pull Up Constructor Body, Field, Method
        - Push Down Field, Method
        - Collapse Hierarchy
        - Extract Interface, Subclass, Superclass
        - Replace Delegation with Inheritance, Inheritance with Delegation

4. Design patterns
    - Introduction
    - GoF patterns
        - Creational: Builder, Prototype, Factory Method, Abstract Factory, Singleton
        - Structural: Facade, Proxy, Composite, Adapter, Decorator, Bridge
        - Behavioral: Command, Observer, State, Strategy, Chain of Responsibility, Mediator, Visitor, Template Method
    - Nuances of use of individual patterns
5. Summary