Kod szkolenia: **REFAKT/JAV**

Tytuł szkolenia: **Użycie wzorców projektowych przy tworzeniu kodu i refaktoryzacji**

Dni: 4

Opis:

Adresaci szkolenia:

Szkolenie adresowane jest do programistów Java, którzy chcieliby poznać wzorce projektowe i prawidłowo stosować je w kodzie, tak przy tworzeniu nowych rozwiązań, jak i refaktoryzacji już zastanego kodu. Również osoby pragnące poznać podstawowe kompetencje w zakresie projektowania uznają to szkolenia za bardzo przydatne.

Cel szkolenia:

Celem szkolenia jest zdobycie umiejętności poprawnego stosowania wzorców. Począwszy od właściwej identyfikacji wymaganego wzorca, poprzez adaptację do specyfiki problemu (podstawy projektowania w UML), a na kodowaniu rozwiązania kończąc.
Drugim wymiarem szkolenia jest wykrywanie złych rozwiązań i ich refaktoryzacja z użyciem poznanych wzorców. Tu przydatna okazuje się znajomość poprawnych zasad projektowania oraz antywzorców, z którymi zapoznajemy, aby łatwiej zidentyfikować problemy w istniejącym kodzie.
Na szkoleniu duży nacisk kładzie się na poprawne stosowanie wzorców, dlatego mimo że grupą odbiorczą są programiści, przedstawiane są podstawy projektowania. Dzięki temu nie tylko wprowadzamy przemyślane rozwiązania, ale w niektórych przypadkach również generujemy dla nich kod na podstawie diagramów klas

Wymagania:

Od uczestników wymaga się umiejętności programowania w języku Java.

Parametry szkolenia

4 dni po 8 godzin. Łącznie 32 godziny szkolenia.
Szkolenie prowadzone jest również w wersji 3 dniowej – bez modułu o refaktoryzacji, ale z prostymi ćwiczeniami z refaktoryzacji przy poznawaniu wzorców.

Program szkolenia:

1. Wprowadzenie do UML

- Czym jest modelowanie obiektowe
- UML – czym jest, a czym nie
- Rozwój UML
- Różnice między diagramem a modelem
- Przegląd diagramów UML
- Wybrane diagramy UML
  - Diagramy klas
  - Opis diagramu klas
    - Identyfikacja klas metodą rzeczownikową
  - Diagramy sekwencji
    - Opis diagramu sekwencji
    - Poprawne tworzenie projektowych diagramów sekwencji

2. Wprowadzenie do wzorców z podstawami projektowania obiektowego
   - Enkapsulacja
   - High Cohension
   - Loose Coupling
   - Command-Query Separation
   - Problemy z dziedziczeniem w Javie
   - Wprowadzenie do wzorców
   - Różne rodzaje wzorców
   - GRASP (General Responsibility Assignment Software Patterns)
     - Information Expert
     - Creator
     - Controller
     - Polymorphism
     - Pure Fabrication
     - Indirection
     - Protected Variations
     - S.O.L.I.D (SOLID-ne programowanie)
       - Single Responsibility Principle
       - Open-Close Principle
       - Liskov Substitution Principle
       - Interface Segregation Principle
       - Dependency Inversion Principle
3. Wzorce GOF
   - Wzorce konstrukcyjne
     - Abstract Factory
     - Builder
     - Factory Method
     - Prototype
     - Singleton
   - Wzorce strukturalne
     - Adapter
     - Bridge
     - Composite
     - Decorator

        - Façade
        - Flyweight
        - Proxy
    - Wzorce behawioralne
        - Chain of responsibility
        - Command
        - Interpreter
        - Iterator
        - Mediator
        - Memento
        - Observer
        - State
        - Strategy
        - Template Method
        - Visitor
4. Przegląd wybranych antywzorców
    - Busy Waiting
    - Circilar Dependency
    - Golden Hammer
    - Hardcoding
    - Lava Flow
    - Object Orgy
    - Spaghetti Code
    - The Blob (God Object)
5. Refaktoryzacja do wzorców
    - Czym jest refaktoryzacja
    - Kiedy warto refaktoryzować? (code smell)
    - Wybrane symptomy złego kodu
        - Contrivied Complexity
        - Cyclomatic complexity
        - Duplicated code
        - Excessive return of dataFeature envy
        - Excessive use of literals
        - Excessively long identifiers
        - Excessively short identifiers
        - Inappropriate intimacy
        - Large class
        - Large method
        - Lazy class
        - Orphan variable or constant class
        - Refused bequest
        - Too many parameters
    - Refaktoryzacja a testy jednostkowe
    - Wsparcie narzędzi przy refaktoryzacji
    - Techniki refaktoryzacji z użyciem wzorców
        - Compose Method

- Chain constructors
- Encapsulate Classes with Factory
- Encapsulate Composite With Builder
- Extract Adapter
- Extract Composite
- Extract Parameter
- Form Template Method
- Inline Singleton
- Introduce Null Object
- Introduce Polymorphic Creation With Factory Method
- Limit Instantiation with Singleton
- Move Accumulation to Collecting Parameter
- Move Accumulation to Visitor
- Move Creation Knowdlego to Factory
- Move Embellishment to Deorator
- Replace Conditional Dispatcher with Command
- Replace Conditional Logic with Strategy
- Replace Constructors with Creation Methods
- Replace Hard-Coded Notifications with Observer
- Replace Implicit Tree with Composite
- Replace One/Many Distinctions with Composite
- Replace State-Altering Conditionals with State
- Replace Type Code with Class
- Unify Interfaces
- Unify Interfaces with Adapter

6. (OPCJONALNE) Osiąganie elastyczności w Javie przy pomocy refleksji
    - Obiekty tworzone na podstawie nazwy klasy
    - Dynamicznie wołane metody na podstawie nazwy
    - Dynamiczne proxy