Kod szkolenia: **REFAKT/JAV**

Tytuł szkolenia: **Wzorce projektowe i refaktoryzacja kodu w języku Java**

Design patterns and refactoring in Java

Dni: 4

Opis:

Cel szkolenia

Celem szkolenia jest zdobycie umiejętności poprawnego stosowania wzorców projektowych. 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 się, aby łatwiej zidentyfikować problemy w istniejącym kodzie. Na szkoleniu duży nacisk kładzie się na poprawne stosowanie wzorców, dlatego mimo że grupą docelową 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.

Mocne strony szkolenia

Poprawne techniki refaktoryzacji do wzorców z użyciem narzędzi na gotowym i generowanym kodzie. Połączenie siły IDE z narzędziem do modelowania UML.

Grupa docelowa

programista

Adresaci szkolenia

Programiści Java, chcący poznać wzorce projektowe i prawidłowo stosować je przy refaktoryzacji oraz tworzeniu nowych rozwiązań.

Wymagania dla uczestników

Umiejętność programowania w języku Java.

sages

Parametry szkolenia

4 dni wykładów i warsztatów. Szkolenie prowadzone jest również w wersji 3 dniowej – bez modułów o antywzorcach i refaktoryzacji, ale z prostymi ćwiczeniami z refaktoryzacji przy poznawaniu wzorców. Maksymalna liczba uczestników: 8-10

Program szkolenia:

1. Niezbędnik UML dla wzorców (8 h)
    - Wprowadzenie do UML
    - Diagram klas
    - Diagram sekwencji
    - Modelowanie w omówionych diagramach
2. Wprowadzenie do wzorców i podstawy projektowania obiektowego (1,5 h)
    - 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)
    - S.O.L.I.D (SOLID-ne programowanie)
3. Wzorce GOF (17 h)
    - Konstrukcyjne: Abstract Factory, Builder, Factory Method, Prototype, Singleton
    - Strukturalne: Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
    - Behawioralne: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor
4. Przegląd wybranych antywzorców (0,5 h)
    - Busy Waiting
    - Circular Dependency
    - Golden Hammer
    - Hardcoding
    - Lava Flow
    - Object Orgy
    - Spaghetti Code
    - The Blob (God Object)
5. Refaktoryzacja do wzorców (5 h)
    - Czym jest refaktoryzacja
    - Kiedy warto refaktoryzować? (code smells)
    - 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

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: biuro@sages.com.pl web: www.sages.com.pl

- 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. Podstawy dynamicznego wołania metod (opcjonalne)
   - Tworzenie obiektów na podstawie nazwy klasy
   - Refleksja
   - Invokedynamic
   - Dynamiczne proxy