SZKOLENIE ŚREDNIO ZAAWANSOWANE

Wzorce projektowe i refaktoryzacja w języku Java

REFAKT/JAV

Czas trwania: 4 dni

Osiąganie wysokiej jakości, czytelnego i elastycznego kodu dzięki użyciu wzorców GOF i narzędzi do refaktoryzacji

Cele szkolenia

- 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
- Wykrywanie złych rozwiązań i ich refaktoryzacja z użyciem poznanych wzorców
- Duży nacisk na poprawne stosowanie wzorców (przedstawiane są podstawy projektowania, mimo że grupą docelową są programiści)
- Wprowadzanie przemyślanych rozwiązań, oraz generowanie dla nich kodu na podstawie diagramów klas

Zalety

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

Dla kogo?

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

Wymagania

- Umiejętność programowania w języku Java

Program

1. Niezbędnik UML dla wzorców
    a. Wprowadzenie do UML
    b. Diagram klas
    c. Diagram sekwencji
    d. Modelowanie w omówionych diagramach
2. Wprowadzenie do wzorców i podstawy projektowania obiektowego
    a. Enkapsulacja
    b. High Cohension
    c. Loose Coupling
    d. Command-Query Separation
    e. Problemy z dziedziczeniem w Javie
    f. Wprowadzenie do wzorców
    g. Różne rodzaje wzorców
    h. GRASP (General Responsibility Assignment Software Patterns)
    i. S.O.L.I.D (SOLID-ne programowanie)
3. Wzorce GOF
    a. Konstrukcyjne: Abstract Factory, Builder, Factory Method, Prototype, Singleton
    b. Strukturalne: Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
    c. Behawioralne: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor
4. Przegląd wybranych antywzorców
    a. Busy Waiting
    b. Circular Dependency
    c. Golden Hammer
    d. Hardcoding
    e. Lava Flow
    f. Object Orgy
    g. Spaghetti Code
    h. The Blob (God Object)
5. Refaktoryzacja do wzorców
    a. Czym jest refaktoryzacja
    b. Kiedy warto refaktoryzować? (code smells)
    c. Wybrane symptomy złego kodu: Contrivied Complexity, Cyclomatic complexity, Duplicated code
    d. Excessive return of dataFeature envy, Excessive use of literals
    e. Excessively long identifiers, Excessively short identifiers
    f. Inappropriate intimacy, Large class, Large method, Lazy class
    g. Orphan variable or constant class, Refused bequest, Too many parameters
    h. Refaktoryzacja a testy jednostkowe
    i. Wsparcie narzędzi przy refaktoryzacji
    j. Techniki refaktoryzacji z użyciem wzorców
    k. Compose Method, Chain Constructors, Encapsulate Classes with Factory

l. Encapsulate Composite With Builder, Extract Adapter, Extract Composite
m. Extract Parameter, Form Template Method, Inline Singleton
n. Introduce Null Object, Introduce Polymorphic Creation With Factory Method
o. Limit Instantiation with Singleton, Move Accumulation to Collecting Parameter
p. Move Accumulation to Visitor, Move Creation Knowledge to Factory
q. Move Embellishment to Deorator, Replace Conditional Dispatcher with Command
r. Replace Conditional Logic with Strategy, Replace Constructors with Creation Methods
s. Replace Hard-Coded Notifications with Observer, Replace Implicit Tree with Composite
t. Replace One/Many Distinctions with Composite
u. Replace State-Altering Conditionals with State, Replace Type Code with Class
v. Unify Interfaces, Unify Interfaces with Adapter

6. Podstawy dynamicznego wołania metod (opcjonalne)
   a. Tworzenie obiektów na podstawie nazwy klasy
   b. Refleksja
   c. Invokedynamic
   d. Dynamiczne proxy