

Cezary Hołub
Wrocław 2025

Akademia Techniczno-Informatyczna w Naukach Stosowanych

Przedmiot	Projektowanie i Programowanie Aplikacji Biznesowych
Semestr	Zima 2024/2025

Lab 9 08.02.2025

Materiały do ćwiczeń

**Proszę przekazywać efekty pracy na Git'a aby ich nie stracić.
W Git'cie, powinny być commity od wszystkich członków grupy a nie tylko od jednego.**

Uwaga!!! Poniższe kody źródłowe i konfiguracja są przykładowe każda grupa opiera się o swój dotychczasowy projekt

Wprowadzenie:

Testy jednostkowe - Weryfikowanie poprawności działania pojedynczych elementów (jednostek) programu w izolacji od reszty systemu. Zaletą testów jednostkowych jest możliwość wykonywania na bieżąco w pełni zautomatyzowanych testów na modyfikowanych elementach programu, co umożliwia często wychwycenie błędu natychmiast po jego pojawieniu się i szybką jego lokalizację zanim dojdzie do wprowadzenia błędnego fragmentu do programu. Testy jednostkowe są również formą specyfikacji.

Testy jednostkowe w Javie - JUnit

- JUnit jest to narzędzie – platforma programistyczna – framework w ramach technologii Java służące do tworzenia testów jednostkowych dla oprogramowania napisanego w języku Java
- JUnit jest przedstawicielem całej rodziny frameworków do testowania jednostkowego nazywanej wspólnie xUnit
- JUnit znajduje się w pakiecie
 - junit.framework w przypadku JUnit 3.8 i starszych
 - org.junit w przypadku JUnit 4.0 i nowszych

ZADANIE 1

Przygotowanie środowiska projektu pod testy Junit

Testy jednostkowe ze względu na osoby kontekst uruchamiania, oraz całkowitą separację od kodu produkcyjnego, wymagają osobnego pliku konfiguracyjnego Springa – `applicationContext-test.xml`. Dla ułatwienia, nasz kontekst testowy będzie kopią produkcyjnego. Normalnie powinno się tego unikać, testy powinny pracować w izolacji od zależności zewnętrznych, dotyczy to w szczególności braku zależności od baz danych. Nowoczesna konstrukcja Springa i JUnit przychodzi z pomocą i dla takiego przypadku, można globalnie dla testów zdefiniować operacje rollback. Nawet jak testy będą modyfikować zawartość bazy danych w czasie swojego działania, tuż po skończeniu testów rollback wycofa wszystkie zmiany.

Przebieg ćwiczenia:

1. Proszę dodać zależności do biblioteki JUnit w pliku pom.xml
2. Proszę umieścić plik applicationContext-test.xml w katalogu /src/test/resources/
3. Proszę utworzyć plik applicationContext-test.xml o podobnej treści (UWAGA!!! Poniżej jest przykład, każdy może mieć inne parametry i ustawienia w pliku ze względu na organizację jego projektu)

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="pl.sos" /> <!-- Chcemy automatycznie obsługiwać
wszystkie klasy z adnotacjami w tym
    pakiecie. UWAGA!!! każdy projekt może mieć oczywiście inna strukturę pakietów
-->

    <context:annotation-config /> <!-- To na przyszłość, pozwoli nam korzystać z adnotacji
także w klasach,
    które byśmy skonfigurowali ręcznie -->

    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
        <property name="url" value="jdbc:hsqldb:hsqldb://localhost/testdb" />
        <property name="username" value="sa" />
        <property name="password" value="" />
    </bean>

    <bean id="entityManagerFactoryBean"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="packagesToScan" value="WWSIS.sos.model" />
        <property name="jpaVendorAdapter">
            <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                <property name="showSql" value="true" />
                <property name="databasePlatform"
value="org.hibernate.dialect.HSQLDialect" />
            </bean>
        </property>
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">validate</prop>
            </props>
        </property>
```

```

        </bean>

        <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"
/>

        <bean
class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />

        <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
            <property name="entityManagerFactory" ref="entityManagerFactoryBean" />
        </bean>
        <bean id="StudentDao" class="WWSIS.sos.dao.impl.StudentImpl" />
        <bean id="PrzedmiotyDao" class="WWSIS.sos.dao.impl.PrzedmiotyDaoImpl" />
    </beans>

```

TEORIA

Konwencje nazewnictwa

W ramach testów przetestujemy metody publiczne z klas DAO które stworzyliśmy w rozdziale powyżej: StudentDaoImpl, PrzedmiotyDaoImpl, etc.. Konwencja nazewnictwa wymaga aby nowo tworzony pakiet z klasami testów nazwać, tak jak oryginalny z którego klasy będziemy testować. W naszym przypadku stworzymy pakiet WWSIS.sos.dao. Konwencja nazewnictwa dla klas testowych wymaga dodania słowa Test przed nazwą klasy którą testujemy, np. klasa testowa dla klasy StudentDaoImpl będzie się nazywać TestStudentDaoImpl. **(klas testowych będzie tyle ile jest klas DAO w projekcie)**

ZADANIE 2

Implementacja klasy testowej TestStudentDao*

*Nazwa klasy jest przykładowa. Klasa ta odpowiada za implementację logiki służącej obsłudze podstawowych operacji Studenta.

Przebieg ćwiczenia:

1. Proszę napisać testy JUnit dla wszystkich metod publicznych z klasy StudentDao. Powinny pojawiać się przypadki testowe (metody testowe) dla **skrajnych przypadków i najczęściej występujących przypadków dla każdej metody publicznej** z klasy StudentDao
2. Poniżej przykładowy kod początku klasy testowej

```
package WWSIS.sos.dao;
```

```
import static org.junit.Assert.assertFalse;
```

```

import static org.junit.Assert.assertTrue;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.annotation.Rollback;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.transaction.annotation.Transactional;
import WWSIS.sos.model.Student;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:applicationContext-test.xml"})
@Transactional
@Rollback(true)
public class TestStudentDao {

    @Autowired
    StudentDao studentDAO;

    Student newStudent;

    @Before
    public void setUp() {
...
    }

    @Test
    public void testPokazPlanZajec() {
...

    }

    . . .
    . . .
}

```

ZADANIE 3

Implementacja klasy testowej Test [xxx] Dao*

*Nazwa klasy jest przykładowa. Klasa ta odpowiada za implementację logiki służącej obsłudze [xxx]

Przebieg ćwiczenia:

1. Proszę napisać testy JUnit dla wszystkich metod publicznych z pozostałych klas DAO,. Powinny pojawiać

się przypadki testowe (metody testowe) dla **skrajnych przypadków i najczęściej występujących przypadków dla każdej metody publicznej** z pozostałych klas typu DAO

2. Poniżej przykładowy kod początku klasy testowej

```
package WWSIS.sos.dao;

import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.annotation.Rollback;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.transaction.annotation.Transactional;
import WWSIS.sos.dao.[xxx]Dao;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:applicationContext-test.xml"})
@Transactional
@Rollback(true)
public class Test[xxx]Dao {

    @Autowired
    [xxx]Dao [xxx]DAO;

    @Before
    public void setUp() {
        ...
    }

    @Test
    public void test[xxx]() {
        ...
    }
}
```