Cezary Hołub
Wrocław 2025

Akademia Techniczno-Informatyczna w Naukach Stosowanych

Przedmiot	Projektowanie i Programowanie Aplikacji Biznesowych
Semestr	Zima 2024/2025

Lab 7 26.01.2025

Materiały do ćwiczeń

Proszę przekazywać efekty pracy na Git'a aby ich nie stracić.

Wprowadzenie:
Dysponując schematem bazy danych i danymi testowymi, oraz konfigurację Srpinga pod naszę naszą bazę danych, należy stworzyć kolejną warstwę naszego SOS, będzie to zalążek warstwy DAO (ang. Data Access Object). Stwórzmy obiekty Java które reprezentują tabele i kolumny z naszej bazy danych. Będą to proste obiekty typu POJO. Obiekty te będą "najniżej" w strukturze warstw naszej aplikacji. Obiekty te będą reprezentować warstwę modelu z wzorca MVC. Technika łączenia i odwzorowywania obiektów z/na relacyjną bazę danych nazywa się ORM. Implementacją idei ORM będzie u nas framework Hibernate.

Data Access Object – komponent dostarczający jednolity interfejs do komunikacji między aplikacją a źródłem danych (np. bazą danych czy plikiem). Jest często łączony z innymi wzorcami projektowymi. Dzięki DAO, aplikacja nie musi znać sposobu oraz ostatecznego miejsca składowania swoich danych, a ewentualne modyfikacje któregoś z czynników nie pociągają za sobą konieczności modyfikowania jej kodu źródłowego. Komponent ten jest często stosowany w modelu MVC (Model-View-Controller) do oddzielenia dostępu do danych od logiki biznesowej i warstwy prezentacji. Gotowe narzędzia do korzystania z DAO wchodzą w skład wielu popularnych języków programowania oraz platform np. Java EE.

Plain Old Java Object (POJO) – termin używany przez zwolenników idei mówiącej, że im prostszy design tym lepiej. Używa się go dla określenia obiektów, będących zwyczajnymi obiektami Java, nie zaś obiektami specjalnymi, w szczególności Enterprise JavaBeans. Obiekty POJO będą czymś w rodzaju odpowiednika tabel z naszej bazy danych. Są to obiekty posiadające tylko pola settery i gettery.

Mapowanie obiektowo-relacyjne (ang. **O**bject-**R**elational **M**apping ORM) – sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze. Implementacja takiego odwzorowania stosowana jest m.in. w przypadku, gdy tworzony system w języku Java łączący się z bazą HSQLDB.

ZADANIE 1	Standard **JPA** i obiekty **POJO**

JPA (Java Persistence API) – standard, który znacznie upraszcza obsługę bazy danych z poziomu aplikacji. JPA to standard z grupy tzw. ORM (ang. Object-Relational Mapping), co w wolnym tłumaczeniu oznacza mapowanie z modelu obiektowego (takiego, jaki używamy w aplikacjach Java) do modelu relacyjnego (takiego, z jakiego korzystają bazy danych).

Tak naprawdę aby korzystać z JPA będziemy potrzebowali jednej adnotacji na każdej klasie oraz jednej na jej polu (identyfikatorze), którą chcemy też przechowywać w bazie danych . Jednocześnie możemy (i będziemy) modyfikować domyślne zachowanie JPA za pomocą dodatkowych adnotacji aby powiedzieć dokładnie, co chcemy, i w jaki sposób, żeby było zrobione. To kolejny przykład elastyczności i wygody, którą dostajemy dzięki podejściu **convention-over-configuration**.

Adnotacje w JPA

W przypadku baz danych mówimy o encjach i tak najważniejszą adnotacją w przypadku JPA jest właśnie `@Entity`. Adnotacja ta nie wymaga żadnych parametrów (możemy wybrać nazwę - pewnego rodzaju alias, używany później w zapytaniach, ale zostaniemy przy domyślnym - nazwie klasy), umieszczamy ją nad całą klasą, która ma być mapowana:

```
@Entity
public class Ksiazka {
    @Id
    private Long id;

    //...
}
```

Klasa taka, którą adnotujemy za pomocą `@Entity`, musi mieć publiczne gettery i settery dla każdego pola. Założenia przyjęte przez JPA w takiej sytuacji są następujące:

- tabela nazywa się tak jak klasa
- kolumny nazywają się tak, jak pola i są odpowiedniego typu z możliwością wpisania null

Powyżej widzimy także adnotację `@Id` - wskazuje nam ona, że dane pole jest identyfikatorem (unikalnym) tego obiektu. Najczęściej jest to pole typu `Long` o nazwie id lub podobnej. Bardzo często można spotkać się z adnotacją `@GeneratedValue`, co wskazuje, że identyfikator ten jest generowany automatycznie w momencie zapisu do bazy danych.

Zmiana parametrów tabeli

Bardzo często mamy już gotową bazę danych (UWAGA!!! W naszym przypadku tak jest), którą chcemy zmapować i nie chcąc zmieniać jej struktury musimy dopasować nasze mapowanie (np. nazwę tabeli). Służy do tego adnotacja `@Table` - najczęściej wykorzystujemy jej parametr name, jak na poniższym przykładzie:

```
@Entity
@Table(name="ksiazki")
public class Ksiazka {
    //...
}
```

Taki zapis spowoduje, że klasa `Ksiazka` będzie mapowana na tabelę "`ksiazki`" zamiast domyślnej tabeli "`Ksiazka`"

Zmiana parametrów kolumny
Podobnie jak w przypadku tabeli, domyślną nazwą kolumny jest nazwa pola. Najczęściej jednak nie jest to zgodne z konwencją nazewnictwa w bazie danych (gdzie często zamiast camelCase używamy nazw_z_podkreslnikami). Aby zmienić nazwę kolumny lub zmodyfikować jej atrybuty możemy użyć adnotacji `@Column` jak na przykładzie poniżej:

```
@Entity
@Table(name="ksiazki")
public class Ksiazka {

    @Column(name="tytul_ksiazki", nullable=false)
    private String tytulKsiazki;
    //...
}
```

Widzimy tutaj dwie rzeczy: po pierwsze wskazujemy, że pole `tytulKsiazki` będzie zapisywane w kolumnie o nazwie `tytul_ksiazki`, na dodatek nie może mieć wartości `NULL` (ustawiliśmy atrybut nullable na `false` [domyślnie jest `true`] - jeśli spróbujemy zapisać w bazie danych obiekt, który w tym polu będzie miał wartość `null`, otrzymamy błąd bazy danych.

Tutorial:
http://www.tutorialspoint.com/jpa/jpa_introduction.htm
http://galaxy.eti.pg.gda.pl/katedry/kask/dydaktyka/Platformy_technologiczne/Java/2013/jpa.pdf

Przebieg ćwiczenia:

1. Będziemy definiować klasy które są odpowiednikiem modelu we wzorcu MVC.Klasy te powinny mieć wydzielony pakiet najlepiej z nazwą "`model`". Proszę stworzyć pakiet typu `pl.wwsis.sos.model` lub `WWSIS.sos.model` - to tylko przykład, każdy w swoim projekcie może mieć inaczej
2. Proszę w tym projekcie stworzyć klasy odpowiadające ilościowo i nazewniczo tabelom z bazy danych. Prosze pamiętać że dzięki adnotacją `@Table(name=...)` i `@Column(name=...)`, nazwy nie muszą być identyczne, ale dobrze aby było bardzo bliskie.
3. Proszę odpowiednio zaadnotować klasę (odpowiada tabeli) i pola (odpowiadają kolumnie).

Adnotacje muszą uwzględniać do jest identyfikatorem co może być null a co nie. Może też dołożyć constarainty na np. maksymalną długość czy unikalność wartości w kolumnie.

4. Proszę zadbać o settery i gettery
5. Prosze zmodyfikować u siebie fragment pliku konfiguracyjnego Springa, tak aby wszystkie parametry odpowiadały naszej bazie danych i naszej konfiguracji. Tym razem proszę zwrócić szczególną uwagę na "`packageToScan`"

```
<tx:annotation-driven />
<bean id="entityManagerFactoryBean"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="pl.wwsis.sos.model" />
    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
           <property name="showSql" value="false" />
           <property name="databasePlatform" value="org.hibernate.dialect.MySQLDialect" />
        </bean>
    </property>
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">update</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>
```

Zgrubna charakterystyka systemu SOS

SOS = System Obsługi Studenta. Webowy system obsługi studenta pozwalający na zarządzanie , zapisami na przedmioty, podgląd ocen, wpłat czesnego, wypożyczeń z biblioteki, planu zajęć opisu przedmiotów i innych.

Zgrubny cel i zakres systemu (naprowadzenie na tematykę wymagań, to tylko podpowiedź). Każdy z poniższych punktów należy rozbić na konkretne wymagania.

1. System ma umożliwiać studentom zapisywanie/wypisywanie się na przedmioty (proszę pamiętać o wymaganiach towarzyszących i pośrednich)
2. System ma umożliwiać studentom podgląd ocen z przedmiotów
3. System ma umożliwiać studentom podgląd przedmiotów
4. System ma umożliwiać studentom podgląd planu zajęć
5. System ma umożliwiać pełne zarządzanie kontem użytkownika systemu (*jako podpowiedź mamy tu takie konkretne wymagania(już po rozbiciu): rejestracja konta, logowanie, wylogowanie, edycja konta, edycja emaila, edycja hasła, usunięcie konta, przypomnienie hasła, blokowanie konta po trzech nieudanych próbach logowania)*
6. System ma się integrować z systemem księgowym uczelni (tylko podgląd czy, ile oraz kiedy zapłacił czesne)
7. System ma się integrować z systemem biblioteczny (tylko podgląd stanu konta studenta: wypożyczone, przeterminowane, oddane, oczekujące w kolejce)