Cezary Hołub
Wrocław 2025

Akademia Techniczno-Informatyczna w Naukach Stosowanych

Przedmiot	Projektowanie i programowanie aplikacji biznesowych
Semestr	Zima 2024/2025

Lab 8 26.01.2025

Materiały do ćwiczeń

Wprowadzenie:

Dysponując warstwą encji danych (proste obiekty typu POJO). Możemy przystąpić do konstrukcji warstwy DAO, która to pracuje na encjach.

DAO (ang. *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 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.

Warstwa "Modelu" z wzorca **M**VC, tak naprawdę dzieli się na kolejną trójstopniową hierarchię

1. definicja tabel w SQL - dwa laboratoria do tyłu
2. definicja encji (obiekty typu POJO) - jedno laboratorium do tyłu
3. DAO - obecne laboratorium. DAO składa się z interfejsów i klas implementujących te interfejsy

dodatkowo, **metodom z warstwy DAO muszą towarzyszyć testy jednostkowe.**

Metody z DAO pracują na encjach! Oznacza to, że parametry wejściowe, oraz typy zwracane metody, powinny być całym encjami, lub polami wchodzącymi w skład encji. Oczywiście encje można grupować w listy (np. pobranie wszystkich użytkowników to będzie lista encji typu User). Pusty typ zwracany czy pusty parametr też są dopuszczalne.

Przydatne linki do dzisiejszego laboratorium:

- http://www.objectdb.com/java/jpa dobry tutorial o definiwaniu całej warstwy danych
- http://www.mkyong.com/hibernate/hibernate-query-examples-hql/ HQL/JPQL w pigułce
- http://docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html interfejs EntityManagera. Warto się zapoznać**. Wiele operacji na danych, mamy za darmo z EntityManagera! Nie trzeba pisać zapytań w JPQL.**
- http://www.tutorialspoint.com/spring/spring_bean_definition.htm tutorial o beanach Springowych
- http://www.tutorialspoint.com/spring/index.htm całościowy tutorial o Springu
- https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/ - dobre wyjaśnienie relacji

ZADANIE 1	Definicja interfejsów warstwy DAO

W celu stworzenia warstwy DAO skorzystamy ze znanego wzorca projektowego "Wzorzec mostu" (https://pl.wikipedia.org/wiki/Most_(wzorzec_projektowy)) czyli oddzielimy abstrsakcję (intefejsy) od implementacji (klasy). Zbudujemy trzy interfejsy w których będą deklaracje metod wykonujące operacje na naszej warstwie danych

Przebieg ćwiczenia:

1. Proszę dodać pakiet w którym będą nasze interfejsy (np. `com.wwsis.sos.dao`). Ostatnie słowo, czyli "dao" jest obowiązkowe, umożliwia przejrzystą nawigację po kodzie i jest zgodne z powszechnymi praktykami panującymi od lat w branży IT.
2. Proszę dodać tyle interfejsów ile jest klas encyjnych, każdy w nazwie powinien kończyć się słowem "Dao". Jest to kolejna konwencja nazewnicza. Wszystkie interfejsy będące w pakiecie który kończy się `.dao` na powinny mieć również w nazwie `Dao` na końcu, **np.:**
    a. `StudentDao`
    b. `PrzedmiotyDao`
    c. `ZapisyDao`
    d. `BibliotekaDao`
    e. `UzytkownikDao`
    f. `etc.`

    Ilość interfejsów zależy głównie od ilości klas encyjnych.
3. W interfejsach `*Dao` deklarujemy metody które będą implementować logikę opisaną w fazie projektowania (głównie: diagram klas oraz wymagania funkcjonalne), **np.:**
    4. W interfejsie `PrzedmiotyDao` mogą znaleźć się metody `pobierzDostępnePrzedmiot(...)` oraz `pobierzSzczegołyPrzedmiotu(...)`
    5. W interfejsie `CzesneDao` moą znaleźć się metody `pobierzZaległeWpłaty(...)`, `pobierzWszystkieWplaty(...)`, `pobierzSzczegolyWplat(...)`

Teoria	Standard **JPQL (HQL)** i praca z **EntityManagerem**

JPQL to skrót od Java Persistence Query Language. Język ten oferuje mnóstwo możliwości, podobnie do języka SQL. Naszym celem nie jest poznać go dokładnie. W przypadku podstawowych operacji, możemy te pojęcia utożsamiać (mając jednak świadomość, że są to inne rzeczy!). Język ten służy do wykonywanie zapytań **na obiektach (encjach) nie na tabelach**, w sposób bardzo podobny do języka SQL. Zapytań możemy użyć np. do pobrania listy obiektów.

Aby wykonać zapytanie musimy najpierw je utworzyć za pomocą metody `entityManager.createQuery` a następnie pobrać wynik. Aby pobrać listę książek potrzebujemy więc wykonać poniższy kod:

```
Query query = entityManager.createQuery("SELECT k FROM Ksiazki k");
List<Ksiazka> ksiazki = query.getResultList();
```

Jeśli nasze zapytanie zwraca tylko jeden element, możemy użyć metody `query.getSingleResult()`;

EntityManager został zdefiniowany na poprzednim laboratorium w pliku konfiguracyjnym Springa. Możemy go używać w klasach typu DAO. Sposób użycia w przykładowej klasie DAO - adnotujemy anotacją `@PersistenceContext`

```
@Transactional
public class KsiazkaDaoImpl implements KsiazkaDao {

	@PersistenceContext
	EntityManager entityManager;

	@Override
	public List<Ksizka> getAllBooks() {
		Query query = entityManager.createQuery("SELECT k FROM Ksiazki k");
		List<Ksiazka> ksiazki = query.getResultList();
		return ksiazki;


	}
```

Podstawowe operacje w JPQL

- **FROM**

```
String hql = "FROM Employee";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **AS**

```
String hql = "FROM Employee AS E";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **SELECT**

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **WHERE**

```
String hql = "FROM Employee E WHERE E.id = 10";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **ORDER BY**

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **GROUP BY**

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E GROUP BY E.firstName";
Query query = entityManager.createQuery(hql);
List results = query.getResultList()
```

- **INSERT** - wstawianie nowego rekordu **musi być połączone z operacja SELECT**

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  +
             "SELECT firstName, lastName, salary FROM old_employee";
Query query = Query query = entityManager.createQuery(hql);

int result = query.executeUpdate();
```

Parametry w zapytaniach JPQL, do zapytań można przekazywać parametry. Parameter w zapytaniu poprzedza znak "**:**". Przekazanie parametru wymaga wykonania metody `setParameter` na obiekcie klasy `Query`. Pierwszy parametr metody to nazwa pola z zapytania JPQL, drugi parametr to nazwa parametru, którego wartość chcemy przekazać.

```
import javax.persistence.EntityManager;
import javax.persistence.Query;
...
@SuppressWarnings("unchecked")
public List<Author> getAuthorsByLastName(String lastName) {
    String queryString = "SELECT a FROM Author a " +
                         "WHERE a.lastName IS NULL OR LOWER(a.lastName) = LOWER(:lastName)";
    Query query = getEntityManager().createQuery(queryString);
    query.setParameter("lastName", lastName);
```

```
    return query.getResultList();
}
```

- **UPDATE**

```
String hql = "UPDATE Employee set salary = :salary WHERE id = :employee_id";
Query query = entityManager.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
```

- **DELETE**

```
String hql = "DELETE FROM Employee WHERE id = :employee_id";
Query query = entityManager.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
```

EntityManager i jego metody. Jak już było wcześniej wspomniane, niektóre operacje na danych mamy za darmo od EntityManagera. Nie trzeba pisać zapytania w JPQL. Dwa przykłady:

- metoda `find`

Metoda find służy do pobierania jednego obiektu na podstawie jego klucza. Kluczem do wyszukiwania obiektu jest to pole, które posiada adnotację @Id. Przykładowe wywołanie:

```
entityManager.find(Kot.class, 1L);
```

pobierze nam obiekt typu Kot, który ma id o wartości 1.
Jeśli metoda ta nie znajdzie danego obiektu w bazie danych, zwraca wartość null.

- metoda `persist`

Metda do złuży do zapisu i aktualizacji obiektu do bazu danych. Aktualizacji dokonuje się wtedy, gdy obiekt, który chcemy zapisać ma takie samo Id (pole z adnotacją @Id) jako obiekt który już jest w bazie. EM sam potrafi rozróżnić i zarządzać kiedy jest robiona aktualizacja a kiedy zapis. Zwalnia nas to z obowiązku myślenia o tym.

```
@Override
public void insertMessage(Message m) {
	m.setAuthorId(5);
	entityManager.persist(m);
}
```

W powyższym przykładzie dysponujemy obiektem "`m`" klasy `Message`. W tym obiekcie nadpisujemy pole `authorId` nową wartością - 5. Następnie cały obiekt zapisujemy ("persystujemy") do bazy.

Zachęcam do zapoznania się z możliwościami EntityManagera
http://docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html

ZADANIE 3	Implementacja metod z interfejsów

Znamy podstawy JPQL, wiemy do czego służy EntityManager. Posiadają interfejsy DAO, czas napisać implementację ich metod. Idąc za Wzorcem Mostu, implementacja będzie w klasach. Jedna klasa implementuje jeden interfejs, klasy powinny znaleźć się w osobnym pakiecie. Nazwę pakietu dla klas również obowiązuje konwencja nazewnicza. Jeżeli pakiet z interfejsami nazywał się `com.wwsis.sos.dao` konwencja nakazuje dodanie "`impl`" na koniec, czyli mamy pakiet `com.wwsis.sos.dao.impl` i w niem znajdą się nasze klasy DAO.

Klasy obowiązuje konwencja nazewnicza. Jeżeli klasa implementuje jakiś interfejs dodajemy słowo "`Impl`" w nazwie klasy. Dla na interfejsu `StudentDao`, klasa go implementująca to `StudentDaoImpl`

Przebieg ćwiczenia:

1. Stwórz trzy klasy implementujące interfejsy: **np.:** `StudentDaoImpl, PrzedmiotyDaoImpl BibliotekaDaoImpl, PrzedmiotyDaoImpl`
2. Dokonaj implementacji wszystkich metod z interfejsów

ZADANIE 4	Dodanie beanów odpowiedzialnych za DAO do pliku konfiguracyjnego Springa

Nasze klasy DAO to tak zwane Springowe "beany". Springowe beany powinny być znane Springowi, czyli muszą się gdzieś zarejestrować. Miejscem tworzenia beanów, jest nasz plik XML z konfiguracją Springa (np. applicationContext.xml). Bean to taka specjalna klasa, która jest zarządzana przez Springowy kontener IoC (ang. *Inversion of Control*). Klasa stająca się beanem ma tą właściwość, że można ją "wstrzykiwać" do innych klas.
Nasze klasy typu DAO powinny być właśnie "wstrzyknięte" do warstwy serwisów. Obrazując to inaczej klasy DAO (warstwa Model) będą "wstrzyknięte" do warstwy Controller z wzorca MVC.
Więcej o beanach, tutaj: http://www.tutorialspoint.com/spring/spring_bean_definition.htm

Przykład deklaracji bean w pliku konfiguracyjnym Springa, np.::

```
<bean id="StudentDao" class="com.wwsis.sos.dao.impl.StudentDaoImpl"/>
```

Przebieg ćwiczenia:

1. Dokonaj deklaracji trzech beanów Springowych odpowiadającym trzem klasom DAO z powyższych ćwiczeń

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)