Arbeitsblatt JPA

Ziele
Ziel dieses Arbeitsblattes ist es, einzelne Aspekte von JPA genauer kennen zu lernen, zu verstehen oder zu vertiefen.

Ausgangslage
Im Gitlab-Projekt https://gitlab.fhnw.ch/eaf/hs21/02 steht das Projekt lab-jpa zur Verfügung welches für diese Aufgaben verwendet werden soll. Importieren Sie dieses Projekt in ihre IDE.

Wie im Movie-Rental Projekt verwenden wir auch in diesem Projekt Spring Boot um den EntityManager und die Transaktionen zu erzeugen. Bei den einzelnen Testprogrammen (die jedoch nicht als JUnit-Tests realisiert sind sondern als einfache Java-Programme mit einer main-Methode) wird mit der Annotation `@EntityScan` jeweils angegeben, in welchem Paket die Modellklassen definiert sind. Jedes Testprogramm hat folgenden Rahmen:

```
@SpringBootApplication
@EntityScan(basePackageClasses=Customer.class)
public class TestX implements CommandLineRunner {

	@PersistenceContext
	EntityManager em;

	public static void main(String[] args) {
		new SpringApplicationBuilder(TestX.class).run(args);
	}

	@Override
	@Transactional
	public void run(String... args) throws Exception {
	}
}
```

Die Datei `application.properties` definiert die in-memory Datenbank mit der URL `jdbc:h2:mem:lab-jpa-db`.

Spring-Boot ist zudem so konfiguriert, dass jeweils auch ein Webserver gestartet wird damit die Datenbank mit der h2-Konsole betrachtet werden kann. Die Konsequenz ist, dass ein gestartetes Programm jeweils weiterläuft bis es explizit gestoppt wird, und ein zweiter Webserver kann unter demselben Port nicht geöffnet werden. Einige der Programme enden daher explizit mit der Anweisung `System.exit(0)`.

Die Nummern der Testprogramme (`Test1`, `Test2`, …) beziehen sich jeweils auf die folgenden Aufgaben.

Aufgaben:

1) **Unifizierung im Persistenzkontext**
Im Programm `Test1` wird der Kunde mit ID=1 zweimal mit der Methode `find` des Entity-Managers geladen:

```
Customer c1 = em.find(Customer.class, 1);
Customer c2 = em.find(Customer.class, 1);
```

Vergleichen Sie danach diese beiden Objekte mit einem Referenzvergleich (==). Sind diese beiden Objekte identisch oder nicht?

Was für ein Resultat erhalten Sie beim Referenzvergleich, wenn Sie zwischen den beiden `em.find`-Aufrufen den Persistenzkontext leeren?

```
Customer c1 = em.find(Customer.class, 1);
em.clear();
Customer c2 = em.find(Customer.class, 1);
```

Was für ein Resultat erhalten Sie beim Referenzvergleich, wenn Sie nach dem laden der Entität c1 diese mit `em.detach(c1)` explizit aus dem Persistenzkontext entfernen?

Wiederholen Sie dieses Experiment mit zwei unterschiedlichen Entity-Managern:

```
Customer c1 = emf.createEntityManager().find(Customer.class, 1);
Customer c2 = emf.createEntityManager().find(Customer.class, 1);
```

In diesem Beispiel haben wir uns von Spring eine Instanz der Klasse `EntityManagerFactory` einstecken lassen, über die wir dann `EntityManager`-Instanzen erzeugen können. Die Annotation lautet `@PersistenceUnit`.

```
@PersistenceUnit
EntityManagerFactory emf;
```

2) **OneToOne Association**
Definieren Sie zwischen `Customer` und `Address` eine „bidirektionale“ 1:1 Assoziation und lassen Sie das *mappedBy* Attribut auf beiden Seiten weg.

```
@Entity
public class Customer {

    @OneToOne
    private Address address;

    …
}
```

```
@Entity
public class Address {

    @OneToOne
    private Customer customer;

    …
}
```

a) Wie sieht das generierte DB Schema aus? In welcher Tabelle ist der Fremdschlüssel definiert?

b) Was für ein Objektmodell haben Sie damit abgebildet?

Das Testprogramm *Test2* bewirkt lediglich, dass die Datenbank neu erstellt und die H2-Console verfügbar wird.

3) **Bidirectional OneToOne Association**
Ergänzen Sie nun den Code den Sie im letzten Task in der Klasse `Address` hinzugefügt haben so, dass eine *normale* bidirektionale 1:1 Assoziation zwischen den Entitäten `Customer` und `Address` definiert wird. Sie können nach dieser Änderung auch nochmals das Testprogramm `Test2` starten um zu prüfen, wie das generierte Schema nach dieser Änderung aussieht.

Wir betrachten nun den folgenden Code aus der Klasse `Test3`:

```
Customer c = new Customer("Gosling", 55);
Address a = new Address("Infinite Loop 1", "Cupertino");
c.setAddress(a);

em.persist(a);
em.persist(c);
```

Fragen:

a) Ist die Reihenfolge der beiden Aufrufe em.persist(a) und em.persist(c) wichtig?

b) Ist der Aufruf von em.persist(a) nötig?

4) **Unidirectional OneToMany Association**
Definieren Sie in der Klasse `Customer` zwischen `Customer` und `Order` eine **unidirektionale 1:n** Beziehung (`OneToMany`), d.h. es soll möglich sein, dass ein Kunde mehrere Bestellungen hat.

Mit der Definition der Assoziation muss auch eine Methode `addOrder` implementiert werden. Diese soll eine neue Bestellung einem Kunden zuordnen (d.h. der Liste der Bestellungen hinzufügen).

a) Wie sieht das von JPA generierte DB Schema aus?

b) Kann die Assoziation auch anders in der Datenbank abgelegt werden?
Hinweis: Annotation `@JoinColumn`

c) Was passiert (bei beiden Varianten), wenn im Testprogramm `Test4` der Kommentar bei `c2.addOrder(o1);` gelöscht wird?