tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

xHIT 2017/18, Gruppe 2

Distributed Computing DezSys-GK8.3

Komponentenbasierte Programmierung

Armin Freudenthaler

14. Juni 2018

Bewertung:		Version:	0.1
Betreuer:	BORM	Begonnen:	24.5.18
		Beendet:	0.0.18

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt die Anwendung von komponentenbasierter Programmierung mittels Webframeworks.

1.1 Ziele

Das Ziel dieser Übung ist die automatisierte Persistierung und Verwendung von Objekten eines vorgegebenen Domänenmodells mittels eines Frameworks. Dabei sollen die CRUD-Operationen der verwendeten API zur Anwendung kommen.

Die Persistierung soll mittels der Java Persistence API (JPA) realisiert werden.

1.2 Voraussetzungen

- Grundlagen zu Java und das Anwenden neuer Application Programming Interfaces (APIs)
- Verständnis über relationale Datenbanken und dessen Anbindung mittels höherer Programmiersprachen (JDBC/ODBC)
- Verständnis von UML und Build-Tools

1.3 Aufgabenstellung

Erstellen Sie von folgendem Modell Persistenzklassen und implementieren Sie diese mittels JPA:

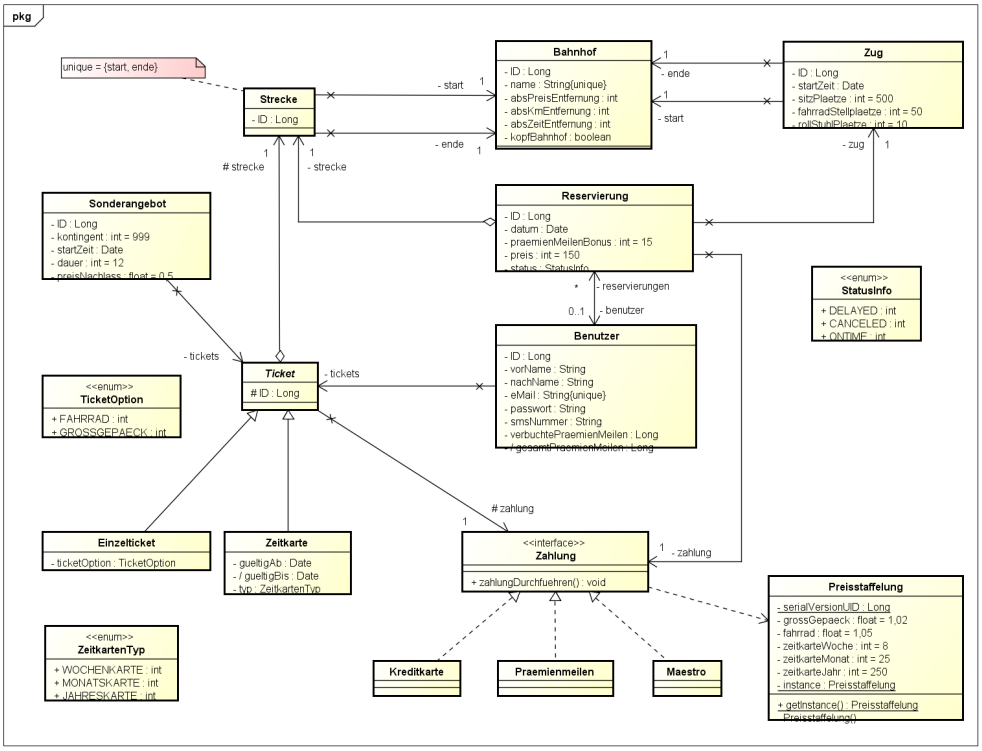

Abbildung 1: Westbahn

Suche
Die Suche nach Zügen muss auf jeden Fall die Auswahl des Abfahrts- und Ankunftsortes (nur folgende Bahnhöfe sind möglich: Wien Westbhf, Wien Hütteldorf, St. Pölten, Amstetten, Linz, Wels, Attnang-Puchheim, Salzburg) ermöglichen. Dies führt zur Anzeige der möglichen Abfahrten, die zur Vereinfachung an jedem Tag zur selben Zeit stattfinden. Des weiteren wird auch die Dauer der Fahrt angezeigt.

In dieser Liste kann nun eine gewünschte Abfahrtszeit ausgewählt werden. Die Auswahl der Zeit führt zu einer automatischen Weiterleitung zum Ticketshop.

Um sich die Auslastung der reservierten Sitzplätze anzusehen, muss bei dem Suchlisting noch das Datum ausgewählt werden. Dieses Service steht jedoch nur registrierten Benutzern zur Verfügung.

Ticketshop
Man kann Einzeltickets kaufen, Reservierungen für bestimmte Züge durchführen und Zeitkarten erwerben. Dabei sind folgende Angaben notwendig:

Einzeltickets: Strecke (Abfahrt/Ankunft), Anzahl der Tickets, Optionen (Fahrrad, Großgepäck) Reservierung: Strecke (Abfahrt/Ankunft), Art der Reservierung (Sitzplatz, Fahrrad, Rollstuhlstellplatz), Reisetag und Zug (Datum/Uhrzeit) Zeitkarte: Strecke, Zeitraum (Wochen- und Monatskarte)

Um einen Überblick zu erhalten, kann der Warenkorb beliebig befüllt und jederzeit angezeigt werden. Es sind keine Änderungen erlaubt, jedoch können einzelne Posten wieder gelöscht werden.

Die Funktion „Zur Kassa gehen“ soll die Bezahlung und den Ausdruck der Tickets sowie die Zusendung per eMail/SMS ermöglichen. Dabei ist für die Bezahlung nur ein Schein-Service zu verwenden um zum Beispiel eine Kreditkarten- bzw. Maestrotransaktion zu simulieren.

Prämienmeilen
Benutzer können sich am System registrieren um getätigte Käufe und Reservierungen einzusehen. Diese führen nämlich zu Prämienmeilen, die weitere Vergünstigungen ermöglichen. Um diese beim nächsten Einkauf nützen zu können, muss sich der Benutzer einloggen und wird beim „Zur Kassa gehen“ gefragt, ob er die Prämienmeilen für diesen Kauf einlösen möchte.

Instant Notification System der Warteliste
Der Kunde soll über Änderungen bezüglich seiner Reservierung (Verspätung bzw. Stornierung) mittels ausgesuchtem Service (eMail bzw. SMS) benachrichtigt werden. Bei ausgelasteten Zügen soll auch die Möglichkeit einer Anfrage an reservierte Plätze möglich sein. Dabei kann ein Zuggast um einen Platz ansuchen, bei entsprechender Änderung einer schon getätigten Reservierung wird der ansuchende Kunde informiert und es wird automatisch seine Reservierung angenommen.

Sonderangebote
Für festzulegende Fahrtstrecken soll es ermöglicht werden, dass ein fixes Kontingent von Tickets (z.b.: 999) zu einem verbilligten Preis (z.b.: 50% Reduktion) angeboten wird. Diese Angebote haben neben dem Kontingent auch eine zeitliche Beschränkung. Der Start wird mit Datum und Uhrzeit festgelegt. Die Dauer wird in Stunden angegeben. Diese Angebote werden automatisch durch Ablauf der Dauer beendet.

1.3.1 Task 1: Mapping

Schreiben Sie für alle oben definierten Klassen und Relationen entsprechende Hibernate JPA Implementierungen (javax.persistence.*). Bis auf die Klasse Reservierung sollen dafür die Annotationen verwendet

werden. Die Klasse Reservierung soll mittels XML Mapping definiert werden.

1.3.2 Task 2: Named Queries

Schreiben Sie folgende NamedQueries (kein plain SQL und auch keine Inline-Queries) für das Domänenmodell aus Task1. Die Queries sollen die entsprechenden Parameter akzeptieren und die gewünschten Typen zurückliefern:

- Finde alle Reservierungen für einen bestimmten Benutzer, der durch die eMail-Adresse definiert wird.
- Liste alle Benutzer auf, die eine Monatskarte besitzen.
- Liste alle Tickets für eine bestimmte Strecke aus (durch Anfangs- und Endbahnhof definiert), wo keine Reservierungen durchgeführt wurden.

1.3.3 Task 3: Validierung

Alle Constraints der einzelnen Entitäten sollen verifiziert werden. Hierfür soll die Bean Validation API verwendet werden. Folgende Einschränkungen sollen überprüft werden:

- Zug und Strecke können nicht denselben Start- und Endbahnhof besitzen.
- Die eMail des Benutzers soll ein gängiges eMail-Pattern befolgen.
- Die Startzeit eines Sonderangebotes kann nicht in der Vergangenheit liegen.
- Der Name eines Bahnhofs darf nicht kürzer als zwei und nicht länger als 150 Zeichen sein. Sonderzeichen sind bis auf den Bindestrich zu unterbinden.

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen "überwiegend erfüllt"
  - Dokumentation und Beschreibung der angewendeten Schnittstelle
  - Task 1
  - Task 2
- Anforderungen ßur Gänze erfüllt"
  - Task 3
  - Ausreichende Testobjekte zur Validierung der Persistierung
  - Überprüfung der funktionalen Anforderungen mittels Regressionstests

2 Abgabe

Es ist ein Protokoll (TGM-HIT/latex-protocol) sowie der Link zum Github-Repository (als Kommentar) hier abzugeben. Die Durchführung der Übung ist mittels regelmäßigen Commits zu dokumentieren. Zum Abgabegespräch ist das Protokoll ausgedruckt vorzulegen (doppelseitig, beidseitig - an langer Kante).

3 Quellen

"The Java EE Tutorial - Persistence"; Oracle; online: https://docs.oracle.com/javaee/7/tutorial/partpersist.htmBNBPY
"HTML5 - A vocabulary and associated APIs for HTML and XHTML"; W3C; 17.12.2012; online: https://www.w3.org/TR/2
html5-20121217/forms.htmlvalid-e-mail-address "Hibernate ORM Documentation"; JBoss; online: http://hibernate.org/or
"JPA Westbahn"; Maven Project structure; online: https://github.com/TGM-HIT/syt4-jpa-westbahn "Hi-
bernate ORM 5.2.13 Final User Guide"; JBoss; 25.01.2018; online: https://docs.jboss.org/hibernate/orm/5.2/userguide/htm

4 Zeitaufwand

Geschätzt: 10 Stunden (inklusive Protokoll)
Aktuell:

5 Programm

5.1 Astah

Zum Beginn, wird das westbahn Beispiel von Prof. Borko verwendet. Dieses findet man hier. Das Repository muss geforked werden. Nun haben wir vollen Zugriff auf das Repo. Unter den Ordner /Design findet man ein Westbahn.asta. Dieses soll mit Astah geöffnet werden. In Astah kann man unter Tools -> Java -> Export Java, kann man den Quellcode generieren lassen. Außerdem befindet sich gradle und maven funktionsfähig am Repo.

5.2 Generieten Code anpassen

Der generierten Code muss in ein Ordner "Model"verschoben werden. Nun hat man 2 Packages:

- src/main/java/main
- src/main/java/model

In Model sollten sich nun die folgenden Klassen befinden:

- Bahnhof
- Benutzer
- Einzelticket
- Kreditkarte
- Maestro
- Praemienmeilen
- Preisstaffelung
- Reservierung
- Sonderangebot
- enum StatusInfo
- Strecke
- enum TicketOption
- interface Zahlung
- Zeitkarte
- enum ZeitkartenTyp
- Zug

Diese müssen aber noch angepasst werden. In IntelliJ kann man unter Code -> generate die Getter/Setter und den Konstruktor generieren lassen. Weiters müssen noch die Enums angepasst werden, da diese von Astah falsch generiert werden. Zum Beispiel bei TicketOptionen, da sieht der generierte Code so aus:

```
public enum TicketOption {

    ;

    public int FAHRRAD;

    public int GROSSGEPAECK;

}
```

Dieser muss angepasst werden, bis es so aussieht:

```
public enum TicketOption {

    FAHRRAD,

    GROSSGEPAECK;

}
```

Diese Änderung muss auch noch bei StatusInfo und ZeitkartenTyp durchgeführt werden.

5.3 Konfiguration

Unter /src/main/resources/META-INF befindet sich ein Config file, "persistence.xml". Dieses muss noch für die eigene Entwicklungsumgebung angepasst werden. Der User, das Password und die Datenbank müssen angegeben werden.

```
     <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
            http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="westbahn">
    <description> Hibernate JPA Configuration Example</description>
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url"
      ↪  value="jdbc:mysql://localhost:3306/westbahn?serverTimezone=UTC" />
      <property name="javax.persistence.jdbc.user" value="afreudenthaler" />
      <property name="javax.persistence.jdbc.password" value="Abc" />

      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>

      <property name="hibernate.show_sql" value="false" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
    </properties>

  </persistence-unit>

</persistence>
```

6 Umsetzung

6.1 Task 1

6.1.1 Klassen

Alle Klassen die abgespeichert werden, müssen mit den Tag @Entity makiert werden. Weiters sollen noch Attribute gesetzt werden, dafür gibt es ebenfalls eigene Tags. Folgende Annotationen werden verwendet:

- @Id: Every entity object in the database is uniquely identified (and can be retrieved from the database) by the combination of its type and its primary key. Primary key values are unique per entity class. Instances of different entity classes, however, may share the same primary key value.

  Only entity objects have primary keys. Instances of other persistable types are always stored as part of their containing entity objects and do not have their own separate identity.

- @GeneratedValue: Provides for the specification of generation strategies for the values of primary keys.

  The GeneratedValue annotation may be applied to a primary key property or field of an entity or mapped superclass in conjunction with the Id annotation. The use of the GeneratedValue annotation is only required to be supported for simple primary keys. Use of the GeneratedValue annotation is not supported for derived primary keys.

- @OneToOne: Defines a single-valued association to another entity that has one-to-one multiplicity. It is not normally necessary to specify the associated target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning side must use the mappedBy element of the OneToOne annotation to specify the relationship field or property of the owning side.

- @ManyToOne: Defines a single-valued association to another entity class that has many-to-one multiplicity. It is not normally necessary to specify the target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning OneToMany entity side must used the mappedBy element to specify the relationship field or property of the entity that is the owner of the relationship.

- @Column Is used to specify the mapped column for a persistent property or field. If no Column annotation is specified, the default values apply.

Source Strecke:

```
1  @Entity
2  public class Strecke {
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      private Long ID;
7
8      @NotNull
9      @ManyToOne
10     private Bahnhof start;
11
12     @ManyToOne
```

```
13      private Bahnhof bahnhof;
14
15      @NotNull
16      @ManyToOne
17      private Bahnhof ende;
18  }
```

6.2 InterFaces/Enum

Interfaces und Enums sollen nicht abgespeichert werden, da man sie nicht durch einen PrimaryKey eindeutig identifizieren kann.

6.3 XML-Mapping

Reservierung soll mit XML-Mapping abgespeichert werden. Dafür wird ein File reservierung.xml erstellt.

- <hibernate-mapping>...root-Element
  - <Class>-Element
  - <id> Für den Primary-Key
  - <property> für die Variablen
  - <component> Enum/Interfaces
  - <one-to-one>/<many-to-one> für die Annotationen

```
1   <?xml version='1.0' encoding='UTF-8'?>
2   <!DOCTYPE hibernate-mapping PUBLIC
3           "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4           "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5
6   <hibernate-mapping>
7       <class name="model.Reservierung" table ="reservierung">
8           <id name="ID" access="field">
9               <generator class="identity"></generator>
10          </id>
11          <property name="datum" type="java.util.Date" access="field" />
12          <property name="praemienMeilenBonus" type="int" access="field" />
13          <property name="preis" type="int" access="field" />
14          <component name="status" class="model.StatusInfo" access="field" />
15          <component name="zahlung" class="model.Zahlung" access="field" />
16          <one-to-one name="zug" class="model.Zug" access="field" />
17          <many-to-one name="strecke" class="model.Strecke" access="field" />
18          <many-to-one name="benutzer" class="model.Benutzer" access="field" />
19      </class>
20  </hibernate-mapping>
```

6.4 Main

In der Main-Klasse werden die Einträge erstellt. Dies erfolgt durch die Methode fillDB:

```
1            em.getTransaction().begin();
2        List<Bahnhof> bahnhoefe = new ArrayList<Bahnhof>();
3        List<Strecke> strecken = new ArrayList<Strecke>();
4        List<Zug> zuege = new ArrayList<Zug>();
5
6
7        Bahnhof shbf = new Bahnhof("SalzburgHbf", 20, 60, 120, true);
8        false);
9        false);
10       false);
11       Bahnhof wels = new Bahnhof("Wels-Zentrum", 102, 400, 250, true);
12       bahnhoefe.add(shbf);
13       bahnhoefe.add(wels);
14
15
16       Strecke wien_salzburg = new Strecke(wienhbf, amstetten, shbf);
17       Strecke wien_linz = new Strecke(wienhbf, amstetten, linz);
18       strecken.add(wien_salzburg);
19       strecken.add(wien_linz);
20
21
22       Zug wiener_linien = new Zug(at_time(5, 20), 450, 20, 10, wienhbf, shbf );
23       Zug rex = new Zug(at_time(8, 8), 100, 5, 0, linz, wels );
24       zuege.add(wiener_linien);
25       zuege.add(rex);
26
27       Zahlung maestro = new Maestro();
28
29
30       Ticket wochenkarte = new Zeitkarte(ZeitkartenTyp.WOCHENKARTE, new Date(),
         ↪  wien_linz, maestro);
31
32       Benutzer armin = new Benutzer("Armin", "Freudenthaler", "test@mail.at",
         ↪  "TestPW", "147", (long) 300, wochenkarte);
33
34       Reservierung r = new Reservierung(get_tomorrow(), 10, 100, StatusInfo.ONTIME,
         ↪  wiener_linien, wien_linz, armin, maestro);
35
36       for (Bahnhof b : bahnhoefe)
37           em.persist(b);
38
39       for (Strecke s: strecken)
40           em.persist(s);
41
42       for (Zug z: zuege)
43           em.persist(z);
44
45       em.persist(wochenkarte);
46       em.persist(armin);
47       em.persist(r);
```

```
48          em.flush();
49          em.getTransaction().commit();
```

Dabei werden Objekte erstellt. Diese werden dann mit .persist(Objekt) abgespeichert. Gestartet wird die Transaktion mit .begin() und beendet durch .commit().

6.5 Task 2

- name = "getAllReservationsForEMail", query = ßelect * from reservierung left outer join benutzer on reservierung.benutzer = benutzer.ID "+ "where benutzer.eMail = :eMail"
- name = "getAllUsersWithMonthTicket", query = SSELECT b FROM Benutzer b LEFT JOIN b.tickets t WHERE t.typ=1"
- name = "getAllTicketsWithoutReservation", query = SSELECT t FROM Ticket t "+ "LEFT JOIN Reservierung r ON r.strecke.ID=t.strecke.ID "+ "WHERE t.strecke.ID=:streckeID")

7 Aufwand

Vermuteter Zeitaufwand: 10 Stunden Tatsächlicher Zeitaufwand: 12 Stunden (inkl Protokoll)

8 Probleme

8.1 JDBC Connection Error:

Mit dem User afreudenthaler@localhost konnte sich der JDBC nicht mit dem MySQL Server verbinden.
Access denied for user 'afreudenthaler'@'localhost' (using password: YES)
Das Problem lag an den Rechten des Users. Durch den Befehl

```
GRANT ALL PRIVILEGES ON westbahn.* TO 'afreudenthaler'@'localhost';
```

wurde das Problem gelöst und man konnte eine Verbindung zum Server aufbauen.

8.2 JDBC Timezone

Der Server hatte die Falsche Zeit gesetzt: The server time zone value 'Mitteleuropäische Sommerzeit' is unrecognized Dies wurde durch das setzten der Serverzeit auf UTC gelöst.

```
<property name="javax.persistence.jdbc.url"
↪  value="jdbc:mysql://localhost:3306/westbahn?serverTimezone=UTC" />
```

8.3 SQLSyntax

Es war nicht möglich in die Datenbank zu schreiben, da unter persistence.xml der falsche hibernate.dialect angegeben war. Dieser musste auf MySQL5 gesetzt werden .

```
<property name="hibernate.dialect"
↪  value="org.hibernate.dialect.MySQL5Dialect"/>
```

8.4 Create

Es wurde keine neue Einträge erstellt. Dies lag an einem Falschen Wert in persistence.xml. Und zwar muss hbmddl.auto auf create gesetzt werden.

```
<property name="hibernate.hbm2ddl.auto" value="create" />
```

Abbildungsverzeichnis