tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

Systemtechnik Labor

4AHIT 2017/18

Komponentenbasierte Programmierung

Laborprotokoll

Marc Rousavy

22. April 2018

Bewertung:		Version:	1.1
Betreuer:	Michael Borko	Begonnen:	12. April 2018
		Beendet:	20. April 2018

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:

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.

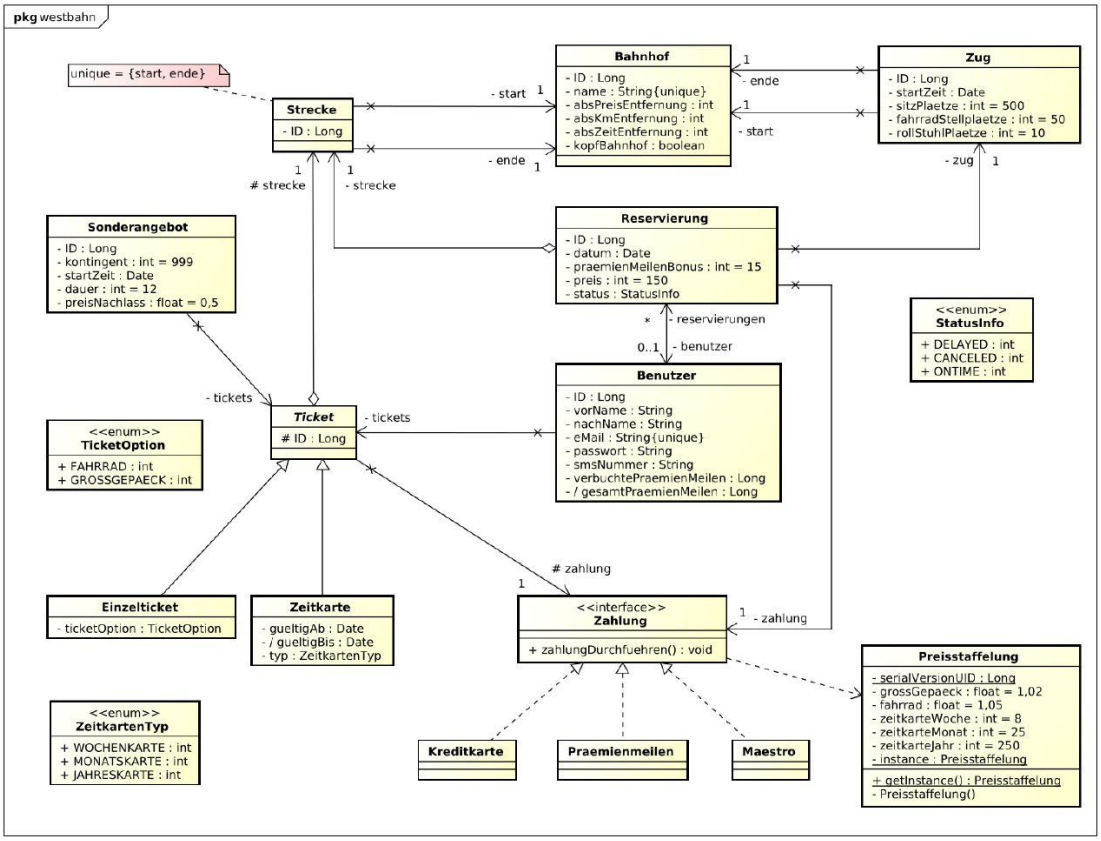

Abbildung 1: Die Westbahn Datenstruktur (UML)

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

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.

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:

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

Task 3 - Validierung Alle Constraints der einzelnen Entitäten sollen verifiziert werden. Hierfür soll die Bean Validation API verwendet werden. Folgende Einschränkungen sollen überprüft werden:

1. Zug und Strecke können nicht denselben Start- und Endbahnhof besitzen.
2. Die eMail des Benutzers soll ein gängiges eMail-Pattern befolgen.
3. Die Startzeit eines Sonderangebotes kann nicht in der Vergangenheit liegen.
4. 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

1.5 Quellen

"The Java EE Tutorial - Persistence"; Oracle; online: https://docs.oracle.com/javaee/7/tutorial/partpersist.htm#BNBPY

"HTML5 - A vocabulary and associated APIs for HTML and XHTML"; W3C; 17.12.2012; online: https://www.w3.org/TR/2012/CR-html5-20121217/forms.html#valid-e-mail-address

"Hibernate ORM Documentation"; JBoss; online: http://hibernate.org/orm/documentation/5.2/l

"Hibernate ORM 5.2.13 Final User Guide"; JBoss; 25.01.2018; online: https://docs.jboss.org/hibernate/orm/5.2/userg

2 Lösung

2.1 Vorbereitung

Als Projekt wird eine **Java EE 8** Application mit folgenden Frameworks erstellt:

- **Maven** - Package/Library Management System
- **Java Persistence API** - Annotation Declarations für Entities und das ORM System
- **Hibernate** - Die Implementierung von Java Persistence API (JPA)
- **Java Database Connectivity (JDBC)** - Java Datenbank Interface
- **log4j** - Java Logger API

IntelliJ lädt automatisch benötigte Libraries, andernfalls kann man dies manuell hinzufügen.
Maven wird bei dem Menüpunkt "Project Dependenciesäusgewählt.
Mittels Maven können nun alle Libraries hinzugefügt werden:

- MySQL:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>
```

- Hibernate (inklusive JPA):

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.16.Final</version>
</dependency>
```

- log4j:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- XML Bind:

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

Das **Maven** Project Object Model (POM) sieht dementsprechend folgendermaßen aus:

```xml
<!--pom.xml-->
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" >
    <modelVersion>4.0.0</modelVersion>

    <groupId>mrousavy</groupId>
    <artifactId>Westbahnhof</artifactId>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.2.16.Final</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
    </dependencies>
</project>
```

2.2 Entities

Entities sind einfache Klassen, bzw. Plain Old Java Object (POJO), welche durch die JPA auf die Datenbank *gemapped* werden.

Für die Westbahn, müssen alle Entities in dem UML (Siehe: Figur **??**) implementiert werden.

Je Entity muss ein POJO erstellt werden, welches eine von der JPA verwaltete Schnittstelle zwischen dem Java Code und der Datenbank repräsentiert.

Die Entities habe ich aus dem Astah UML Diagramm exportiert: **Tool -> Java -> Export Java**

Astah hat nun alle Klassen aus dem UML Diagramm exportiert, welche aber noch alle JPA **Annotations** erhalten werden.

Annotations sind Metadata informationen, welche zur Kompilierung (bei Ausnahmefällen: Auch zur Laufzeit) Klassen, Attributen oder Methoden hinzugefügt werden können.

Die wichtigste **Annotations** ist:

```
@Entity
public class Bahnhof {}
```

welche auf jedem *gemappten* POJO zu finden ist.

Außerdem sollte jede Entity eine eindeutige Identifikation haben, oft durch eine Ganzzahl mit dem namen `ID`.

Beispielsweise könnte so die ID von einem Bahnhof ausschauen:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long ID;
```

wobei die **Annotation** `@GeneratedValue` der JPA mitteilt, es handelt sich um ein generiertes Attribut welches im Idealfall nicht vom Benutzer modifiziert wird. Beispielsweise wäre dies eine automatisch mitzählende Identifikation.

Alle anderen Attribute bzw. **Properties** können zusätzliche **Annotations** erhalten, sind jedoch optional. Beispielsweise kann ein name auch einzigartig sein:

```
@Column(unique = true)
private String name;
```

Mit etwas Integrated Development Environment (IDE) Zauberei können auch getter und setter Methoden generiert werden, womit das POJO - und nun auch die Entity - nun so ausschaut:

```
package BusinessObjects;

import javax.persistence.*;
import java.io.Serializable;

@Entity
public class Bahnhof {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long ID;

    @Column(unique = true)
    private String name;

    private int absPreisEntfernung;

    private int absKmEntfernung;
```

```
18
19      private int absZeitEntfernung;
20
21      private boolean kopfBahnhof;
22
23      public long getID() {
24          return ID;
25      }
26
27      public void setID(long ID) {
28          this.ID = ID;
29      }
30
31      public String getName() {
32          return name;
33      }
34
35      public void setName(String name) {
36          this.name = name;
37      }
38
39      public int getAbsPreisEntfernung() {
40          return absPreisEntfernung;
41      }
42
43      public void setAbsPreisEntfernung(int absPreisEntfernung) {
44          this.absPreisEntfernung = absPreisEntfernung;
45      }
46
47      public int getAbsKmEntfernung() {
48          return absKmEntfernung;
49      }
50
51      public void setAbsKmEntfernung(int absKmEntfernung) {
52          this.absKmEntfernung = absKmEntfernung;
53      }
54
55      public int getAbsZeitEntfernung() {
56          return absZeitEntfernung;
57      }
58
59      public void setAbsZeitEntfernung(int absZeitEntfernung) {
60          this.absZeitEntfernung = absZeitEntfernung;
61      }
62
63      public boolean isKopfBahnhof() {
64          return kopfBahnhof;
65      }
66
```

```
67      public void setKopfBahnhof(boolean kopfBahnhof) {
68          this.kopfBahnhof = kopfBahnhof;
69      }
70  }
```

Dementsprechend müssen nun alle POJOs zu **Entities** gemacht werden.

Ein Sonderfall war die Entity Strecke (`Strecke.java`), da diese *unique constraints* benötigte.
Dazu gefunden habe ich die Dokumentation in den Java Docs: [1], welche zu folgendem Code für die Entity **Strecke** führte:

```
1   @Table(
2           uniqueConstraints = @UniqueConstraint(columnNames = {"start_id", "ende_id"})
3   )
```

Glossar

Entity „Entities sind einfache Klassen, bzw POJO, welche durch die JPA auf die Datenbank *gemapped* werden." [**wiki-entity**]. 8, 9, 11

Akronyme

IDE Integrated Development Environment. 9

JDBC Java Database Connectivity. 7

JPA Java Persistence API. 7–9

POJO Plain Old Java Object. 8, 9, 11

POM Project Object Model. 8

Literaturverzeichnis

[1] Oracle Corporation. *Java Docs - Annotation Type UniqueConstraint*. URL: https://docs.oracle.com/javaee/6/api/javax/persistence/UniqueConstraint.html (besucht am 10. 02. 2011).

Abbildungsverzeichnis