Aufgabenblatt 12: Applikationsprogrammierung 2 (bis Donnerstag, 29.01.2015)

Hinweis: um die *Studienleistung* für diese Vorlesung zu absolvieren, benötigen Sie 50% der Hausaufgabenpunkte aus diesen Übungen. Um das *Modul RDB1* erfolgreich abzuschließen, müssen Sie die Klausur am Ende des Semesters bestehen **und** die Studienleistung erfolgreich absolvieren. Die Übungen müssen stets **donnerstags vor der Vorlesung** abgegeben werden. Dies kann über unseren **Briefkasten** (Informatikzentrum zweiter Stock, gegenüber vom Fahrstuhl) oder zum **Start der Vorlesung** geschehen. Bitte versehen Sie ihre Abgaben stets mit ihrer **Matrikelnummer** und mit der **Nummer ihrer Übungsgruppe**. Die Lösungen dürfen auf Deutsch oder Englisch eingereicht werden. Benutzen Sie für die Lösungen stets ihre **eigenen Worte**.

Aufgabe 12.1 – Applikationsprogrammierung (25 Punkte)

Ziel dieser Aufgabe ist die Implementierung einiger Funktionalitäten, die die Kommunikation zwischen einer Java Applikation und einem DBMS steuern. Dafür wurde bereits etwas Quellcode in Java vorbeireitet, den ihr komplettieren sollt. Bitte ladet euch daher folgendes Archiv runter und entpackt es an einem beliebigen Ort auf eurem Rechner: **http://www.ifis.cs.tu-bs.de/webfm_send/1698**

Abgabe: Die Abgabe erfolgt über Email an die Mailadresse eures Hiwis. Benennt dafür den `src` Ordner, in eurem Projekt um in „`abgabe_<matno1>_<matno2>`“ und packt diesen Ordner in ein zip-Archiv mit demselben Namen. (Nach dem Entpacken soll der Ordner „`abgabe_<matno1>_<matno2>`“ in dem Ordner enthalten sein, in dem das Archiv entpackt wurde). Schickt dieses Archiv dann an euren Hiwi mit dem Betreff „[RDB Aufgabenblatt 12] Lösung von <name1>(<matno1>) und <name2>(<matno2>)“

Verwendetes Datenbankmanagementsystem: Als DBMS wird die leichtgewichtige Datenbank **SQLite** verwendet. Die Verwendung von SQLite hat den Vorteil, dass keinerlei Installation von Software nötig ist. Es muss lediglich die Jar `lib/sqlite-jdbc-3.8.7.jar` zum Classpath hinzugefügt werden. Die Datenbank wird dann einfach in einer Datei angelegt und über die Jar verwaltet.

Verwendetes Schema: Das Beispiel baut auf dem folgenden Schema auf:

Aufgabensetting: Für einen Onlineshop wurde ein Klassenmodell entworfen, das dem obigen ER Diagramm entspricht (siehe Paket `model` im Ordner `src`). Bisher ist es jedoch noch nicht möglich, die in der Applikation angelegten Objekte auch persistent zu speichern. Dies hat zur Folge, dass alle Kundendaten, Produkte und Warenkörbe verschwunden sind, sobald die Applikation beendet wird. Diese Schwachstelle soll in dieser Aufgabe behoben werden.

Daten Persistent speichern: Um die in der Applikation angelegten Objekte persistent zu speichern, wurde das Paket `persistence` angelegt. Dort enthalten ist eine abstrakte Klasse `PersistenceInterface<T>`. Diese Klasse definierten folgende abstrakte Methoden:

- `void _save(T)`: Speichert ein Objekt vom Typ `T` in der Datenbank
- `void _update(T)`: Aktualisiert die Werte eines Objektes vom Typ `T` in der Datenbank
- `T _fetch(Object)`: Gibt ein Objekt vom Typ `T` aus der Datenbank zurück. Das übergebene Objekt ist der Schlüssel der gesuchten Entität in der Datenbank.
- `List<T> _all()`: Gibt eine Liste mit allen in der Datenbank gespeicherten Objekten vom Typ `T` zurück
- `Object key(T)`: Gibt zu einem gegebenen `T` den zum Speichern und Auffinden dieses `T` in der Datenbank verwendeten Schlüssel zurück

Die Klasse definiert außerdem analog die Methoden `void save(T)`, `void update(T)`, `T fetch(Object)` und `List<T> all()` (die Methoden von oben ohne Unterstrich), die die oben implementierten Methoden verwenden und die gespeicherten oder geladenen Objekte im `PersistenzInterface` registrieren.

Main-Methode: Die Main-Methode befindet sich im Paket `main` in der Datei `Main.java`. Die Datei enthält außerdem einige nützliche Beispielmethoden, mit denen die Funktionalität des Persistenz-Interfaces demonstriert wird.

Aufgaben:

a) Implementieren Sie die Methode `getConnection()` in der Datei `DBHelper.java` im Paket `persistence`.
   - Denken Sie daran das Jar `lib/sqlite-jdbc-3.8.7.jar` dem Classpath hinzuzufügen
   - Die Klasse, die für den Treiber geladen werden muss, ist `org.sqlite.JDBC`
   - Die Connection-URL ist z.B. `jdbc:sqlite:test.db`, wenn die Datei, in der die Datenbank gespeichert werden soll, `test.db` heißen soll
   - Bei korrekter Implementierung sollte die `main`-Methode fehlerfrei durchlaufen

b) Implementieren Sie die Methode `createSchema(Connection)` in der Datei `DBHelper.java` im Paket `persistence`.
   - Die Methode soll die Tabellen erstellen, die in der Methode als Kommentar annotiert sind
   - Sind die Tabellen bereits vorhanden, muss keine besondere Behandlung des Fehlers vorgenommen werden
   - Um zu testen, ob die Tabellen richtig erstellt werden, kann der Kommentar „Task 1“ in der main-Methode auskommentiert werden

c) Implementieren Sie das `ShoppingBasketPersistenceInterface` in der Datei `ShoppingBasketPersistenceInterface.java` im Paket `persistence_impl`.
   - Die Klasse erbt `PersistenceInterface<ShoppingBasket>` und soll demnach die Methoden zum persistenten Speichern von Warenkörben definieren
   - Die durchzuführende Implementierung ist sehr ähnlich zu der vom `CustomerPersistenceInterface` und dem `ProductPersistenceInterface` im selben Paket
   - Beim Laden von Warenkörben ist es notwendig die im Warenkorb enthaltenen Produkte gemäß ihrer Produktnummern aufzulösen und die entsprechenden Produkte ebenfalls zu laden. Hierfür kann das `ProductPersistenceInterface` verwendet werden:
     - **Beispiel:** Hole Produkt mit Produktnummer 5: `ProductPersistenceInterface.getInstance(connection).fetch(5);`
   - Bei korrekter Implementierung sollten die Beispiele „Task 2“ - „Task 7“ sowohl einzeln ausgeführt als auch am Stück ausgeführt fehlerfrei funktionieren und die im Quelltext annotierte Ausgabe produzieren