Übungen EADJ

Die Übungen befinden sich in folgendem Bitbucket Repository:
https://bitbucket.org/jonasbandi/cas-eadj-2016

Klonen sie das Repository mit folgendeM Kommando:
`git clone https://bitbucket.org/jonasbandi/cas-eadj.git`

Oder Sie können die Sources über foldenden Link downloaden:
https://bitbucket.org/jonasbandi/cas-eadj-2016/downloads

Übung 1: POJO Testing

Vorbereitung

Führen Sie im Verzeichnis `99-Exercise/01-Java` den Maven build aus. Der Build sollte fehlerfrei durchlaufen:

```
mvn clean install
```

Business Logic Tests

Studieren Sie die Tests in `/domain/src/test/java/org/musicstore/domain`.
Überlegen Sie sich was genau getestet wird.

Tasks

- Schreiben Sie den Test in `OrderServiceTest` um, so dass keine Mocks mehr verwendet werden. Was sind die Vorteile, was sind die Nachteile?
- Erweitern Sie den `PriceCalculator` und den zugehörigen `PriceCalculatorTest` mit folgender Anforderung: Bestehende Kunden (identifiziert durch ihre Email) sollen einen grösseren Rabatt bekommen als neue Kunden. Gehen Sie Test-getrieben vor, indem Sie zuerst den Test schreiben und dann erst die Implementation anpassen.

Übung 2: Persistence Testing

Vorbereitung

Führen Sie im Verzeichnis `99-Exercise/02-Persistence` den Maven build aus. Der Build sollte fehlerfrei durchlaufen:

```
mvn clean install.
```

Persistence Tests

Studieren Sie die Tests in `persistence/src/test/java/org/musicstore/persistence/repositories`.
Überlegen Sie sich was genau getestet wird.

Starten Sie eine externe Datanbank mit dem Kommando.
Mac/Linux:

```
java -cp $M2_REPO/com/h2database/h2/1.4.184/h2*.jar org.h2.tools.Server
```

Windows:

```
java -cp %M2_REPO%/com/h2database/h2/1.4.184/h2*.jar org.h2.tools.Server
```

Geben sie im Web-Interface die folgende JDBC URL ein:
`jdbc:h2:tcp://localhost/~/musicstore`

Tasks

- Lassen Sie die Tests gegen die externe Datenbank laufen (ein/aus-kommentieren der entsprechenden Zeilen). Studieren Sie das zugehörige Setup in `persistence/src/test/resources/META-INF/persistence.xml`.
- Schreiben Sie einen Test für das `MusicOrderRepository`. Implementieren Sie im `MusicOrderRepository` die Methode `getOrdersByEmail`. Gehen Sie Test-getrieben vor, indem Sie zuerst den Test schreiben und dann erst die Implementation anpassen.
- Nachdem Sie in den Business Logic Tests gesehen haben wie man Collaborators mockt, versuchen Sie nun den Entity Manager in einem Repository Test zu mocken. Mach das Sinn?
- Erweitern Sie das Entitäten-Model mit der Entität "Genre". Ein Album soll einem Genre zugewiesen sein. Albums sollen nach Genre suchbar sein. Erweitern Sie dazu auch die Tests.

Übung 3: Arquillian

Vorbereitung

Führen Sie im Verzeichnis `03-Arquillian` den Maven build aus. Der Build sollte fehlerfrei durchlaufen.

```
mvn clean test
```

In-Container Tests mit Arquillian

Studieren Sie den Test

`src/test/java/org/musicstore/persistence/repositories/AlbumRepositoryTest.java`. Überlegen Sie sich was hier getestet wird.

Tasks

- Versuchen Sie den Test in einem richtigen Glassfish Server auszuführen. Dazu müssen Sie die Glassfish-Installation in dem File `src/test/resources/arquillian.xml` konfigurieren (Achtung: Ein Admin Passwort muss gesetzt sein, sie können dies setzen mit `asadmin change-admin-password --user admin`). Der Maven Build ist so aufgesetzt, dass dies über ein Profil gesteuert ist:

  ```
  mvn clean test -Parquillian-glassfish-remote
  ```

- Schreiben Sie ein "OrderService" EJB analog zu der Übung 1. Schreiben Sie einen Arquillian Test dazu. Was ist der Unterschied zu den Tests in Übung 1?

Übung 4 (Mittwoch Abend)

Versuchen Sie eine oder mehrere Test-Strategien in Ihrer Implementation des Bookstore umzusetzen:

- Reines Unit Testing der Business Logik
- Persistence Tests gegen eine in-memory Datenbank und/oder gegen eine dedizierte Datenbank.
- Integrations-Tests der Business Logik über Remote-Zugriff auf EJBs.
- Arquillian Tests
- UI-Tests mit einem UI-Automatisierung-Framewrork (Selenium, WebDriver, Canoo Web Test ...)