Using JPA

In this lab you will start using JPA.

Step 1 - Project Configuration and first test

Open a terminal and enter the following commands:

```
hg clone https://bitbucket.org/paulbakker/jpalabs
cd jpalabs
hg update start
```

Import the newly created project “jpalabs” in an IDE. The project already contains some classes and a persistence.xml. Open the persistence.xml and check the settings. Now open the Book class. It contains an id and a name, but is not an Entity yet. Now open the StandaloneJpaTest in the test folder. This test class already contains a test method that checks if a Book can be persisted. Try to run it, it should fail at this moment. The reason the test fails is because Book is not an Entity yet. Fix Book so that the test passes.

Step 2 - Finding books

Add a new test method to test retrieval of a book using the find method. The tables are recreated before every test, so you can be sure to have a clean database in every test. This does mean the table is empty! Insert some books to be able to find books. Your test could look as follows.

```
@Test
    public void testFind() throws Exception {
        createTestBooks();

        Book book = em.find(Book.class, 1L);
        assertNotNull(book);
        assertThat(book.getTitle(), is("Angels and demons"));
    }

    private void createTestBooks() {
        em.getTransaction().begin();
        em.persist(new Book("Angels and demons"));
        em.persist(new Book("Digital Fortress"));
        em.persist(new Book("The Da Vinci code"));
        em.persist(new Book("The Lost Symbol"));
        em.persist(new Book("Deception Point"));
        em.getTransaction().commit();

        em.close();
        em = emf.createEntityManager();
    }
```

Step 3 - Editing managed books
Create another test that test if managed books can be edited. Always use a new EntityManager and transaction to test for database changes!

Step 4 - Editing detached books
Retrieve a book using the find method and detach it by closing the EntityManager. Try if changes to the instance or not persisted to the database. Now use the merge method to synchronize the changes with the database.

Step 5 - Listing books
Write a simple query to select all books.

Step 6 - Mapping books
Add the following fields to Book:
-Date releaseDate (Date in the database)
-Enum Category
-String summary (should be a longtext in the datase)

Also make the title unique.

Add mapping configuration to make sure that the correct datatypes are used in the database.

Step 7 - Deleting books
Use the remove method to remove a book.

Step 8 - One to One Promotion
Create a new class Promotion. A promotion can be a temporarily lowered price for a limited amount of time. Add the following properties.
-String description
-BigDecimal newPrice
-Date beginDate
-Date endDate

Map a bi-directional relationship between Book and Promotion. Test if saving fetching from both sides work. Use cascading to be able to save a new book with a new promotion without persisting the promotion explicitly. Write tests to test bi-directional behavior and to test cascading remove.

Step 9 - One to Many reviews
Create a new class Review and map a bi-directional one-to-many relation with Book. Add the following properties to Review.
-String reviewerName
-Date reviewDate
-int rating
-String text

Test adding, cascading, lazy loading and bi-directional behavior. Also test cascading remove. Create a separate test to test a join fetch query for explicit eager loading.
Step 10 - Many to Many authors
Create a new class Author and configure a bi-directional Many-to-Many relation with Book with cascading persist. Test if persisting and retrieval works correctly.

Step 11 - Inheritance and ElementCollections
Create a new class EBook and a new enum EBookFormat. The EBookFormat represents formats such as EPUB and PDF. An EBook is a subclass of Book and contains a List of available EBookFormats. To map the List you'll need an @ElementCollection mapping. Test if EBooks can be persisted correctly.

Step 12 - Details embeddable
Create a new class Details with the following properties.
-int pages
-int isbn10
-int isbn13
-String language

Make Details @Embeddable and add a reference to it in Book. Make sure you initialize an empty details by default in Book, otherwise it's impossible to save a book without Details. Create a test to check if Details are persisted correctly on a Book.

Step 13 - Query for books written by more than one author
Write and test a JPQL query to retrieve books that are written by at least two authors. Don't forget to insert some test data first!

Step 14 - Query for the average rating for each Author
Write and test a JPQL query to retrieve the average rating for each author. The results should be returned as a value object Rating.

```
public class Rating {
    private String authorName;
    private double avgRating;

    public Rating(String authorName, double avgRating) {
        this.authorName = authorName;
        this.avgRating = avgRating;
    }

    public String getAuthorName() {
        return authorName;
    }

    public double getAvgRating() {
        return avgRating;
    }
}
```

Don't forget to add some authors and ratings before executing the query!

Step 15 - Criteria API
Write and test a method that accepts a title and author name filter. If both arguments are empty, all books should be returned, including books without an author. If both are non-empty, an AND filter should be used and if only one argument is non-empty only that field

should be filtered. The method should use the Type Safe Criteria API to build this dynamic query. Order results ascending by title.

Step 16 - Bean validation
Add bean validation to the Book class to make sure books always have a title. Write a test that expects a ConstraintViolationException.

Step 17 - Custom validator
Write a new bean validator annotation and implementation class to disallow certain words in a title. The annotation should be configurable:

```
@NotExplicit(filter = {"sex", "drugs", "rock & roll"})
private String title;
```