Lab 4 - ORM and JPA

Summary

The purpose of this assignment is to learn the basics of ORM and EntityManager operations, demonstrate through JUnit test cases, and to begin the design process for your final project.

Requirements

Documentation

Create a README file in your FP repository titled lab4readme.md.

Database Setup

Use your itmd4515 database and user from Lab 1.

Project Setup

Your uid-fp repository should already be setup, and you should continue pushing your commits into GitHub for this lab.

We will be working in this repository from now until the end of the semester. Please remember, I will be looking for multiple commits. Use a prefix of **Lab 4** on your commit messages, for example:

- Lab 4 - Initial Commit
- Lab 4 - Initial entity
- Lab 4 - Test cases

Project Requirements

1. First, and most importantly, think about what kind of business domain you would like to work with for the rest of the semester. There are many examples you could pick from, but you will do your best work if you are engaged in the entities and concepts that underly your code. If you are stuck and need examples, post a question to the discussion board. If you pick your domain now, then you will be able to continue building on this code for future labs and projects, thereby making your efforts cumulative. Some example ideas include:
    1. Library
    2. Education/Learning (like Canvas or Beacon)
    3. Sports
    4. Music
    5. Retail

    6. Hospitality
    7. Medical/Insurance
    8. Inventory or other ERP function
2. Refactor/Move your Lab 3 code into a separate package. I named mine Lab 3. We'll clean these up completely and remove them in a future lab.
3. Create a RESOURCE_LOCAL Persistence Unit named **itmd4515testPU** connecting to your **itmd4515** database using the **itmd4515** user. Use the persistence.xml configuration we discussed in class to drop-and-create the tables.
4. Add the following dependencies to your `pom.xml`:
    1. org.eclipse.persistence.jpa
    2. junit (latest non-beta version of junit 5)
    3. Bean Validation (as we did in Lab 2)
    4. mysql-connector-java (latest non-beta version of 8)

    Use appropriate scope for all your dependencies. Why are these dependencies required if we already have the javaee-api in our pom.xml?

    Make sure your group, artifact and version look like this. **When you use NetBeans to generate persistence related configuration or code files (such as I did with the Persistence Unit) NetBeans can add additional dependencies to your pom.xml for you. Clean it up and make sure yours looks like this:**

```
<dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>${jakartaee}</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.x.x</version>
</dependency>

<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>4.x.x</version>
</dependency>

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.x.x</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
```

```
        <version>8.0.1.Final</version>
    </dependency>

    <dependency>
        <groupId>org.glassfish.expressly</groupId>
        <artifactId>expressly</artifactId>
        <version>5.0.0</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate.validator</groupId>
        <artifactId>hibernate-validator-cdi</artifactId>
        <version>8.0.1.Final</version>
    </dependency>
```

5. Create an Entity as demonstrated in class. This should be a central and important concept to your business domain. ***Do not*** implement **User**, **Group** or **Role** (in the context of security). We will do that in a later lab. Focus on the **business** or **enterprise** aspects of your domain.
    1. Use an appropriate PK strategy, and use an appropriate data type for your PK (as per the options discussed in class)
    2. Include appropriate equals and hashCode methods for your PK strategy
    3. Include at least one temporal data type
    4. Include at least three different data types. There is no limit to the number of attributes you can include. Your attributes should be sufficient to represent your entity. Exercise good design and judgment.
    5. Include appropriate Constructors, accessors and mutators
    6. Include appropriate bean validation constraints based on your database types and sizes
    7. Include appropriate toString method for formatted output
6. Document the following:
    1. Paragraph that describes the business domain you have chosen to work with, and why.
    2. Write a second paragraph answering the following questions: There is only one entity required for Lab 4, but what other entities from your business domain can you think of? How might they relate to one another? You can answer this in narrative form, or you can answer it with a database diagram. One of your midterm questions will be very similar, about the design of your FP, so this is to help get you started early.
7. Create a JUnit test class as demonstrated in class

    For additional examples (beyond my in-class demonstration) you can refer to Goncalves sample code Chapter 06 - JPA Managing and Querying ex03.

    1. You must use `EntityManagerFactory` to create an `EntityManager`. You **may not** use JDBC on this lab.
    2. Make appropriate use of JUnit test fixtures as discussed in class, to obtain an EntityManager and also to stage consistent test data for your assertions
    3. Include test methods for CRUD functionality

        1. Create - em.persist
        2. Read - em.find and print to standard output
        3. Update - entity mutators (set methods) within a transaction
        4. Delete - em.remove
    4. Use at least one assertion in each test method to check pass/fail
    5. **You must implement all test cases for your entity class**
8. Document in your README by adding code block(s) containing the output of your Test class. Clearly identify each operation and discuss what is being tested.
9. Submit to Canvas or Beacon
    1. Right your **uid-fp** project and select "Clean"
    2. Go to your NetBeans Projects directory. Create a zip file of the **uid-fp** folder and submit it to the Canvas or Beacon assignment.

Graduate Students

Undergraduate students may do this for 5 points Extra Credit.

1. Include a second JUnit test class for bean validation. **Do not mix** JPA and Validation test cases, keep them within their respective test class.
2. Consider using abstraction to keep generic functionality and test fixtures at a more re-usable level in your test cases.
3. Include pass/fail (sunny-day/rainy-day) test cases for *each* of your validation constraints.