JPA Practical

Objectives

The objective of this practical session is to use JPA/Hibernate instead of JDBC to interact with the database

Reference Material

This practical session is based on the material covered in the *JPA with Hibernate* chapter. Additional information can be found in the JPA documentation.

Overview

In this practical you will replace the code in your `EmployeeControllerJDBC` with suitable JPA code, to use entity objects instead of writing SQL.

Practical: Using JPA

Setting up the project and connecting to the database

1. You can work on a previous project, or use starter code from the `exercises/student/JDBC` folder.
2. Create a Maven project, and edit the `pom.xml` file to include the dependencies:

```
<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>1.0.2.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.4.10.Final</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.20</version>
</dependency>
```

3. You will need an entity class, representing the `Employee` objects you want to get JPA to handle. Thus, create the class `Employee`. It will need:
   i. A no-argument constructor
   ii. Properties for `id`, `firstname`, `lastname` and `age`
   iii. Suitable annotations `@Entity`, `@Id` and maybe `@Table`, `@Column`

2. Create a new class `EmployeeControllerJPA` which implements the interface `EmployeeController`. Provide a no-argument constructor and the required methods.
3. Declare an instance variable of type `EntityManager`. This plays the role of a 'connection' and gives you access to the JPA API calls. Create an instance of this in your constructor (hint: use the `Persistence` class). The 'persistent unit name' should be '`MyPersistence`'.
4. Be sure to close the `EntityManager` in your `close()` method.
5. In the other methods, use `Employee` objects and JPA calls to implement the behaviour you require. Remember to use transactions where appropriate.
6. Create a file `persistence.xml`. In which folder in your project should this file be?
7. The contents of the file should be:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="MyPersistenceUnit"
                    transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.dialect"
            value="org.hibernate.dialect.MySQLInnoDBDialect"/>
      <property name="hibernate.connection.driver_class"
            value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.connection.username"
            value="root"/>
      <property name="hibernate.connection.password"
            value="password"/>
      <property name="hibernate.connection.url"
            value="jdbc:mysql://XX.XX.XX.XX:3306/employeedb"/>
    </properties>
  </persistence-unit>
</persistence>
```

where *XXXX* is the machine with MySql running.

8. Everything should now be in place to run and test your project. It should work as it did before, but note that now you have not used any explicit SQL.