Lab 09

[valid 2021-2022]

Persistence
Continue the application from lab 8, creating an object-oriented model and using JPA (Java Persistence API) in order to communicate with the relational database.

The main specifications of the application are:

Compulsory (1p)

- Create a *persistence unit* (use EclipseLink or Hibernate or other JPA implementation).
  Verify the presence of the *persistence.xml* file in your project. Make sure that the driver for EclipseLink or Hibernate was added to to your project classpath (or add it yourself).
- Define the entity classes for your model (at least one) and put them in a dedicated package. You may use the IDE support in order to generate entity classes from database tables.
- Create a *singleton* responsible with the management of an *EntityManagerFactory* object.
- Define *repository* clases for your entities (at least one). They must contain the following methods:
  - *create* - receives an entity and saves it into the database;
  - *findById* - returns an entity based on its primary key;
  - *findByName* - returns a list of entities that match a given name pattern. Use a *named query* in order to implement this method.
- Test your application.

Optional (2p)

- Add support for charts. A chart has a name, a creation date and an ordered list of movies.
- Create a generic *AbstractRepository* using *generics* in order to simplify the creation of the *repository* classes. You may take a look at the CrudRepository interface from Spring Framework.

- Implement both the JDBC and JPA implementations and use an *AbstractFactory* in order to create the DAO objects (the repositories).
- The application will use JDBC or JPA depending on a parameter given in an initialization file. (At least for one entity!)

Bonus (2p)

- We say that two movies are *related* if they have the same director (for example). Each day you want to see exactly two movies and you want to create the longest possible playlist that satisfies the following constraints:
    - each day you will watch two related movies;
    - any two movies from different days cannot be related;
- Implement an *efficient algorithm* (for a bonus+) or use one from a third-party library, like JGraphT.
- Test your algorithm for large subsets of movies from your database and describe the runtime performance in a suggestive manner.

Resources

- JPA Tutorial
- Java EE Tutorial: Persistence
- Java Persistence Performance