Iben Andersen
cph-ia62@cphbusiness.dk

Object Relational Mapping with JPA

General part

1) Explain the rationale behind the topic Object Relational Mapping and the Pros and Cons in using ORM

Object Relational Mapping (ORM) is a functionality which is used for converting data between an object and a relational database which are two incompatible systems. The functionality is mapping an object state to a database column and it is able to insert, update and deleting objects.

Some of the pros of ORM are that you are able to eliminate manual mapping work and that you often reduce the amount of code that needs to be written in order to convert data.

A con of using ORM can be that the code can get a bit abstract which can make the code obscure and hard to understand.

2) Explain the JPA strategy for handling Object Relational Mapping and important classes/annotations involved

You are able to use the Java Persistence API (JPA) that includes methods and classes to persist, store, data into databases.

An important annotation is @Entity. The @Entity annotation is used to specify that this class is representing an entity type. It is managed by the persistence.

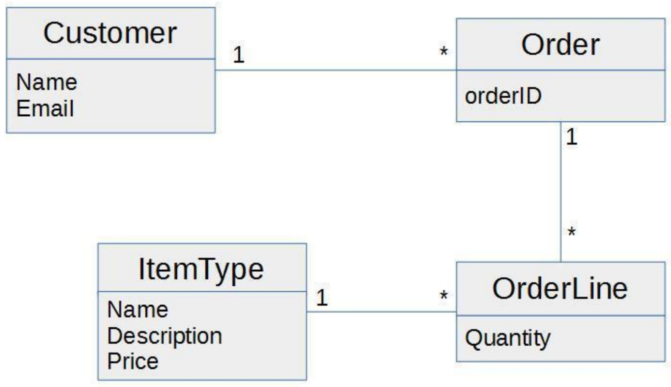
Other important annotations are the relation annotations @ManyToMany, @ManyToOne, @OneToMany and @OneToOne which are used to specify database relationships between entities.

3) Outline some of the fundamental differences in Database handling using plain JDBC versus JPA

A big difference between Database handling using JDBC and JPA is that when you use JDBC you make manual mappers in order to access and add data to a database. When using JPA you use annotations in mapping java objects which makes JPA easier to work with.

Iben Andersen
cph-ia62@cphbusiness.dk

Practical part



1) Examine and understand the diagram.

This model represents an initial model for a system that can handle Orders. Order refers to a Customer and a number of OrderLines. Each OrderLine has a quantity and it refers to an ItemType. The ItemType has a name, a description and a price. The price for each OrderLine is the quantity times the price. The total price for an Order is the sum of all its OrderLines.

You are able to see the relations between the entities. A Customer can have many Orders, but the Order can only belong to one Customer (One-To-Many relation). These two entities have a bidirectional relation. In this relationship every entity has a relationship field or refer the property to the other entity.

An ItemType can have many OrderLines, but the OrderLine can only belong to one ItemType (One-To-Many relation). These two entities have a bidirectional relation as well.

Order can refer the properties of OrderLine, but we do not it to work the other way, so this relation is only unidirectional. Order is in this case the owing side that specifies how to updates can be made in the database.

2) Create a Maven Java Application with NetBeans and use Object Relational Mapping (JPA) to implement the OO classes and the corresponding Database Tables.

NetBeans project ExamPrep_JPA is created and ORM is implemented.

Entities Customer, Order, ItemType and OrderLine are created and relations are made.

Database ExamPrep_JPA is created.

3) Create a Facade and implement as many of the methods below as you have time for, not necessarily in the given order:

Facade is implemented in the NetBeans project ExamPrep_JPA in the Source Package facades Facade.java.