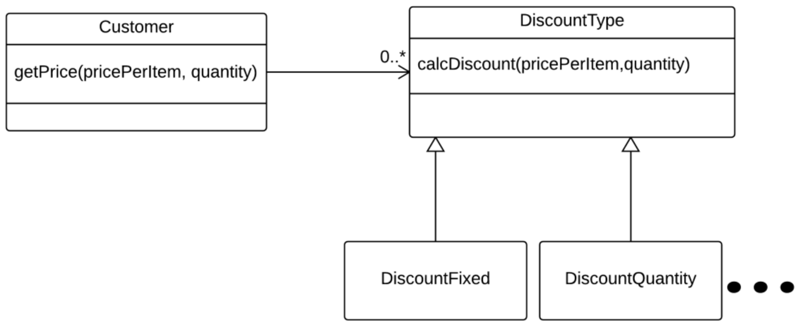
Exercise - JPA Entity Mappings -3

Mapping Inheritance

Imagine that we had different kinds of customers which would get a discount according to their type. The simple and often stupid solution would be to derive specialized Customers from the Customer class. Stupid because it won't allow a design where some customers actually could hold more than one type (get a better discount). This can be implemented using the design below.


Customer
getPrice(pricePerItem, quantity)
0..*
DiscountType
calcDiscount(pricePerItem,quantity)
DiscountFixed
DiscountQuantity
• • •

Before you start this exercise you must know the basics of JPA inheritance mapping using the default SINGLE_TABLE strategy (see the slides)

1) Implement the classes from the diagram above (all as Entity classes).

2) In the class `DiscountFixed` add the following code:

```
double discount = 0.1;  //10% on all
@Override
public double calcDiscount(double priceItem, int quantity) {
  return priceItem * discount *quantity;
}
```

3) In the class `DiscountQuantity` add the following code:

```
int quantityForDiscount = 3;
double discount = 0.2; //20% on all if quantity is 3 or more
@Override
public double calcDiscount(double priceItem, int quantity) {
    return quantity >=quantityForDiscount ? priceItem * quantity * discount : 0;
}
```

4) Run the project and investigate the generated table. Explain the content of each column (especially the DTYPE column) and how it relates to the object model.

5) What we have done so far is using the default Inheritance Strategy which is SINGLE_TABLE.

Add a `@Inheritance` annotation on top of the `DiscountType` class and select `InheritanceType.JOINED` for the strategy.

Regenerate the tables and explain the purpose/content of each table and each column (especially the DTYPE column) and how they relate to the object model.

Querying JPA with JPQL

This exercise uses the database introduced here as a starting point: http://www.mysqltutorial.org/mysql-sample-database.aspx

So the purpose of the exercise is twofold:

- Show how to create a JPA based design from an existing database
- Use this database and its data to get experience with JPQL (Java Persistence Query Language)

Before you start you should investigate the ER-diagram given in this link to an overview of the tables involved in the design.

Getting started:

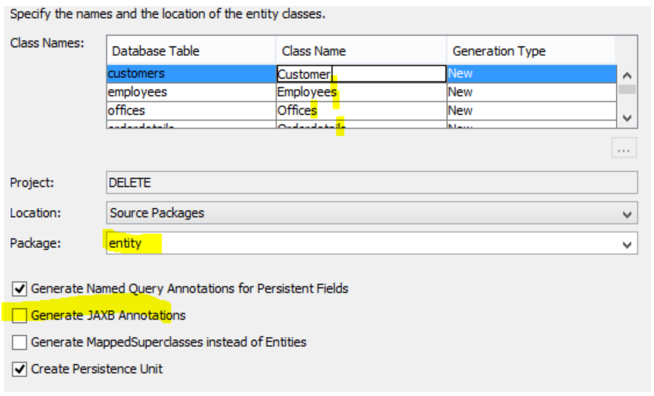
- Create a new plain java project "classicmodels".

Set up the Database

- Create a folder *scripts* and unzip and copy the script from *classicCarsEdited.zip* into the folder.
- Navigate to the services tab, select your local MySQL Server, right click and create a new Database "classicmodels".
- Go back to the script, right click and select "Run File". When prompted for the database to use, select the classicmodels database and execute the script.
- Verify that the tables has been created and populated with data.

Creating the Entity Classes

- Use the wizard "New Entity Classes From Database" to create Entity classes matching the tables in the classicCars database.
    - On the first wizard page, select the database and "Add all" to include all tables
    - On the second wizard page write ***entity*** for Package and deselect "Generate JAXB Annotations"
      The original designers of the database went for the plural form of naming tables (emplyee**s** and not employee). This is something we definitely not want to be reflected in our class names. Go to the Class Name column as sketched below, and rename all class names to their singular form[1].



Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
customers	Customer	New
employees	Employees	New
offices	Offices	New

Project: DELETE

Location: Source Packages

Package: entity

[✓] Generate Named Query Annotations for Persistent Fields

[ ] Generate JAXB Annotations

[ ] Generate MappedSuperclasses instead of Entities

[✓] Create Persistence Unit

    - On the last wizard, de-select the "Use Column Names in Relationships" since this typically ends up with some very unintuitive names

[1] Order is a reserved word (in JPQL) so rename this class to `ClassicOrder`

Tasks:

Explain the generated classes, especially the two (extra) classes generated, ending with PK.

Provide a single façade class, with the following methods:

Creating/Editing Entities:

- `createEmploye(..)` →Creates and return a new Employee[2]
- `updateCustomer(Customer cust)` Updates and returns the updated Customer

Querying

- `getEmployeCount()` → return total employees
- `getCustomerInCity(String city)` → Return all customers living in a given city (Barcelona = 1)
- `getEmployeMaxCustomers()` → Return the employees with most customers
- `getOrdersOnHold()` → Return all orders where status is "On Hold"
- `getOrdersOnHold(int customerNumber)` → Return all orders on hold for a given customer (try Customer 144)

[2] Just manually query the database, before you execute the statement to get the next free employee number. A new employee must have an office, assign the office with officeCode = "1" to all new Employees