*Technical University of Cluj-Napoca  
Faculty of Automation and Computers  
Department of Computer  
2nd Semester 2016-2017*

***Programming Techniques***

***Homework 3***

Student: Peter Ioan-Vlad

Group: 30421

**Problem specification**

**Specification**

Consider an order management application for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

1. Domain specific classes: Order, Customer and Product

2. Business Logic (warehouse-specific processing) classes: OrderProcessing, WarehouseAdministration, ClientAdministration

3. Presentation classes: GUI related classes

4. Data access classes: Database access related classes Other classes and packages can be added to implement the full functionality of the application.

**Requirements**

a. Analyze the proposed application, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Design, implement and test the application classes. Use javadoc for documenting classes.

c. Define, design and implement a system of utility programs (examples: reports for under-stock, totals, filters, etc.).

d. Design and implement a comprehensive demo driver for the order management application.

**Example of working**

The user has 3 options when he opens the application.

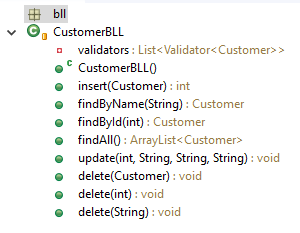
To manage the customers by clicking the “Customers” button the window Manage Customers appears and there the user can Delete, Edit and Add new customers. On the bottom of the window a table with the clients will appear.

By clicking the Products button a new window called manage products will appear. There the user has the option if Deleting, Editing and Adding a new product to the warehouse. On the bottom of the window a table with the products will appear showing the name, price and quantity.

In order to edit the Orders the user must click the Orders button.

**Classes Design**

**CustomerBL**



This class in from the business logic layer, DAO, package and imports java.util.ArrayList;

We use a list named validators that holds objects of type validator with an object of type customer as parameter. The validators are used to verify the email that the user gives to the application to be as a specific pattern.

Method insert has as parameter an object of type customer and goes through the list validators in order to validate each customer’s email and returns the id.

FindByName that has as parameter a string that will hold the name of the customer. A new object of type customer is created and this will receive the results of findByName from the class CustomerDAO from package DAO. If the customer does not exist the method will return null, so an NoSuchElementException will be thrown.

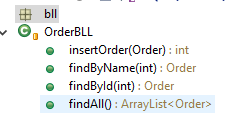
FindById has a parameter an int value that stores the id of the person that must be found. As in the findByName method a new object of type customer is created and this will receive the results of findById from the class CustomerDAO from package DAO. If the customer does not exist the method will return null, so an NoSuchElementException will be thrown.

The method findAll is used to get all the customers from the application. It creates an array list with objects of type clients where the result the result of the findAll method from the class CustomersDAO will be stored. In case there are no clients in the application the method will return null and NoSuchElementException will be thrown.

Method update receives as parameters all the information that a person has and creates a new object of type customer with it by using the method findById. After checking if the new object that was found has empty name, id or address the info is updated by calling the method edit from the CustomerDAO.

In order to delete a customer we have 3 option to do this. If we are given the id, the name or the email. For each of this case we have a separate method with the same name.

**OrderBLL**



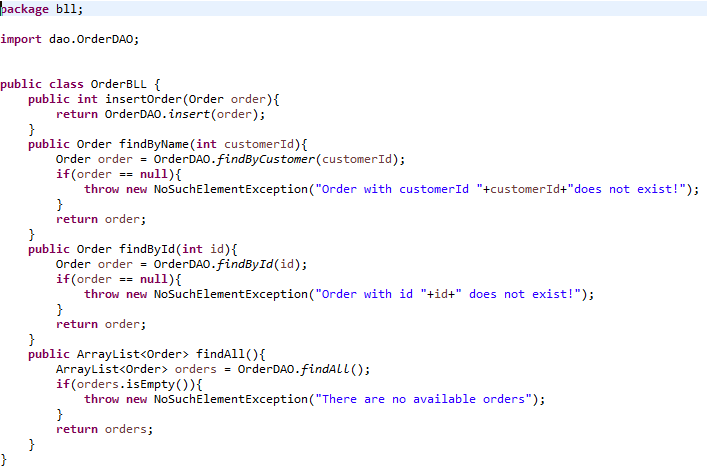
OrderBLL class is used to find orders by different given data and inserting a new order.

Method insertOrder has as parameter an object of type order that is inserted with the help of the insert method from class OrderDAO. As a result it gives the new order’s id in the data base.

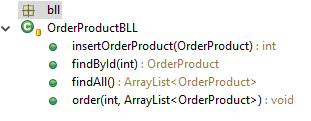
FindByName method is used to find an order based on the id of its client that is given as parameter. A new object of type Order is created that will hold the result of the findByCustomer with the parameter the one given at the beginning. If the returned value is null it means that the order was not found and a NoSuchElement Exception is thrown. The method returns the order.

FindById method is used to find an order based on its own id, the given parameter is an integer that represents the id of the to be found order. A new object of type order is fount and in it will be stored the result of the findById method from class OrderDAO. If the returned value is null it means that the order was not found and a NoSuchElement Exception is thrown. The method returns the order.

Method findAll return an arrayList with all the objects of type order that is in the database. The ArrayList is created and in it is stored the result of findAll method from the class OrderDao. If the returned value is null it means that the order was not found and a NoSuchElement Exception is thrown.



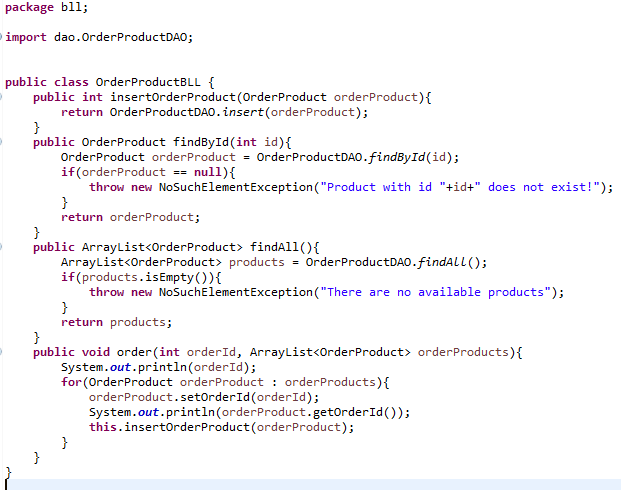
**OrderProductBLL**



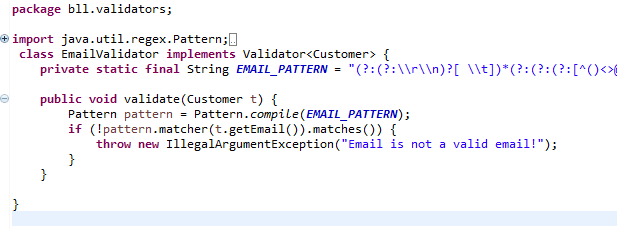
OrderProductBLL class is used when we want to rder a product. The methods in this class are insertOrderProduct that inserts a product in an ArrayList that can be imagined as the shopping cart, find by id that searches the product in the list and returns the id, find all that returns an ArrayList of all ordered products and order that puts a product into the cart.

findById is used to find an order product by its id . In order to do this it creates an object of type orederProduct and stores in it the result of the findById methid from the class OrderProductDAO. If the returned value is null it means that the order was not found and a NoSuchElement Exception is thrown.

findAll method is used to receive all the Ordered products in an ArrayList by using the method find all fom the class OrderProductDAO. If the returned value is null it means that the order was not found and a NoSuchElement Exception is thrown.

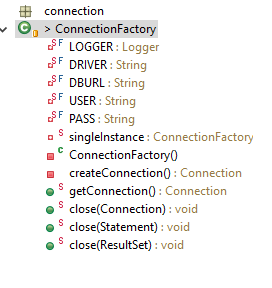


**EmailValidator**



This class from package bll.validators implements the interface Validator and it is used to check the pattern of the email when a user is introduced, if it is not the email pattern an exception will be thrown. This class imports regex,pattern. A regular expression defines a search pattern for strings. The abbreviation for regular expression is regex. The search pattern can be anything from a simple character, a fixed string or a complex expression containing special characters describing the pattern. The pattern defined by the regex may match one or several times or not at all for a given string.

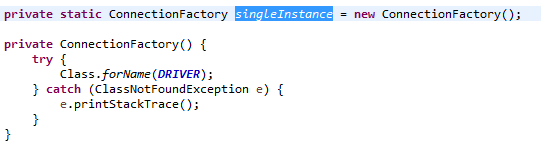
**Connection Factory**



A ConnectionFactory object encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider.

In this class we used a singletone apttern in order to always have only one connection. This is useful when exactly one object is needed to coordinate actions across the system as in our case.

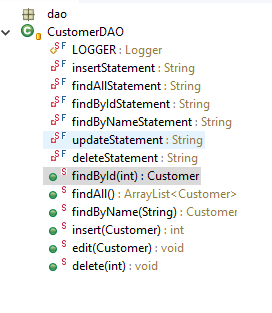
In order to create a new connection we used a singletone object named ConnectionFactory.



In the class createConnection of type Connection (singleton) an object of type Connection is created and initialized with null value. This object’s databse is able to provide information describing its table, it supports SQL grammar stores procedures etc. After this a connection is tried by using a DriverManager’s method getConnection. If the connection fails we throw a SQLException and display an appropriate message using the looger whith it’s method [**log**](https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html#log-java.util.logging.Level-java.lang.String-)([**Level**](https://docs.oracle.com/javase/8/docs/api/java/util/logging/Level.html) level, [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) msg) created at the beginning of this class.

**DAO (Data acces object) package**

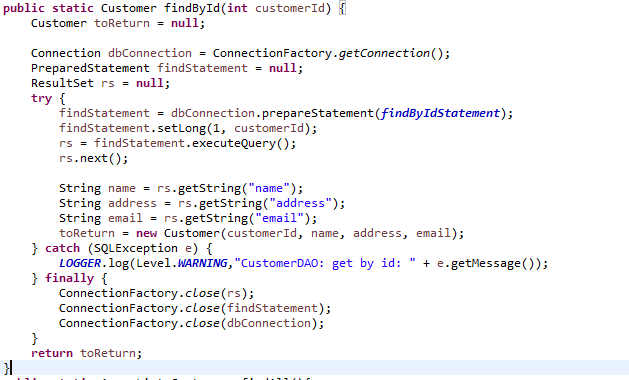
**CustomerDAO**



This class is part of the package dao that stands for data acces object, so this class will have methods for accessing information about customers.

At the beginiong of the class we have a logger with the name of the class obtained by using the clas Class and after that 6 strings that have stored in them SQL statements for insertion find all finding a client by its id, finding a client by it’s name, updating the info of a customer and one for ones deletion. The question marks from the string will be **replaced** with the info in order to execute the SQL instruction for the specific cuctomer.

**findById**



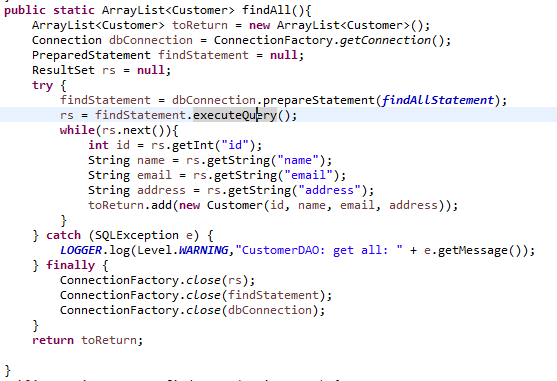
In class findById we have in integer as parameter, an object of type customer is created and initialized with null, in it will be stored the object to be found. A resultSet object called rs is created, it maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSetobject.

The method prepare statement is a subinterface of statement and it is used to execute a parametrized query that are the string declared at the beginning of the class. In the object PreparedStatemnt is stored an precopiled SQL statement, ini this case the one from string findAllStatemnt, and using the setLong it sets the customer ID to the given Java long value. After all this the result set variable stores the result of method executeQuery that Executes the SQL query in this PreparedStatement object and returns the ResultSet object generated by the query

After the execution of the sql query with the help of the next methof of the result set object we move to the next instruction. New string store the fields of a Customer and are given to the object of type cutomer that needs to be returned.

All this is puted in a try catch block and if eanythings goes wrong an appropriate message will be displayed. At the end we have a finally in which we close the connections and the last thing that is done is returning the object with the desired info.

**findAll**

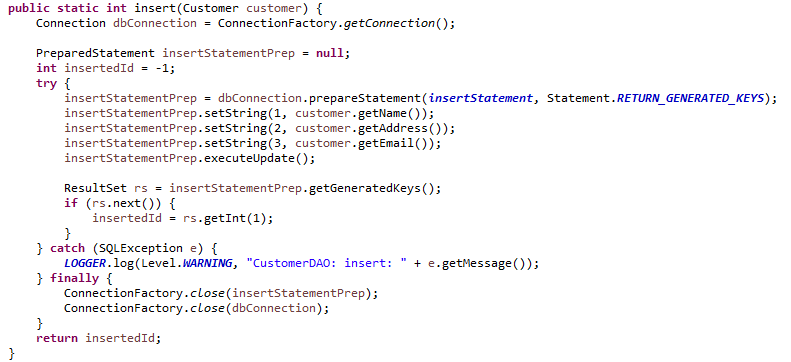


This class is quite similar to the the find by id but the difference is that this one but instead of returning an object of the to be found object it return an array list of objects of type cutomer. A result object is also created, in this we will store the result we will get after the execution of the query.

The process of searching in the database is putted in a try catch finally block. In the try block we put in the prepared statement object called find statement the skeleton of the query that will be used. In the result set object we will store the result of the execution. To go through all the objects we used the method .next() of the object of type result set, it will return 0 when there ar no object left so this was putted in while statement. Each time it finds an object we add a new object in the array list with the appropriate information. The id was stored in an int variable and the name, email and address in string variables.

If any of the operations did not succeded a SQL Exceptionis thrown and a logger is created. In the finally block the result set, find statement and database connection object are closed using the method .close() of the ConnnectionFactory. The array list called toReturn is returned.

**Insert**



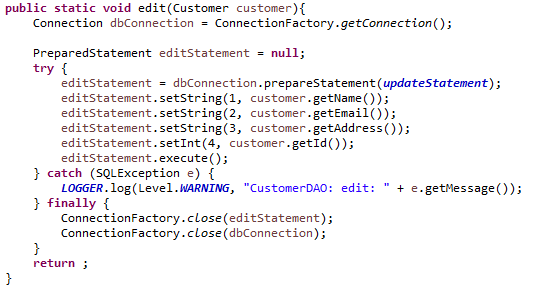
The insert method of class CustomerDAO is used to insert an object in the database. As parameter is an object of type customer is given, this is the one that we will insert.

At the beginning of the the class we create a new object of type connection is created, this will be used to establish the connection with the data base, a prepared statement is created, this will be used to store the statement that will insert the object in the db and integer variable called insertedID is created and it’s value is initialized with -1, this variable will be returned at the end of the method.

The operation to insert the given object in the database is putted in a try catch finally block. In the try block we put in the prepared statement object the query that will be used to insert an object in the db. After this the name address and email are inserted instead of the question marks that were puttet in sql query and the execute update is called in order to execute the insertion. At the end of the try block we put in the inserted id the id that the data base gave to the new inserted object.

In the catch block logger with level warning is created in case the execution in the try block did not succeded. In the finally block objects inserted statement and dbConncetion are closed using the close method of the connection factory.

**Edit**



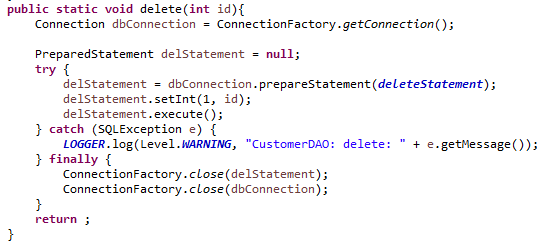
The edit method is used to update an existing object of type customer in the database. A new object of type connection is created called dbConnection and it is initialiezed with the help of the method get connection of the class connectionfactory. A new object of type prepared statement is created called editStatement an it is initialized null.

The process of insertion an object in to the database is putted in a try catch finally block, the method does not return anything, meaning that it is void so it will return nothing.

In the try block we put in the edit statement the prepared statement with query used for update statement. The we update each field that used to be a question mark with the name email, address and id. After they are updated we use the execute method to execute the sql query, the method is part of the interface of the editStament object.

In the catch block we have a logger with level warning if anything from the try block did not work correctly. In the finally block objects inserted statement and dbConncetion are closed using the close method of the connection factory.

**Edit**



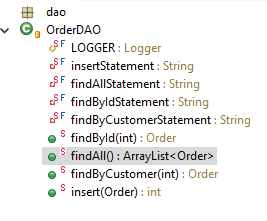
The delete method is used to delete an object of type customer from the database base on its id.

At the beginning an object of type connection is created in order to establish the connection to the database and another object of type prepared statement that is initialized null. The process of deleting a customer from the database is putted in a tru catch finally block .

In the try block we put in the prepared statement object the query that is used to delete an customer. We set the id with help of the setInt method and execute the deletion of the object.

In the catch block we have a logger of level warning in case anything goes wrong in the try block. . In the finally block objects inserted statement and dbConncetion are closed using the close method of the connection factory.

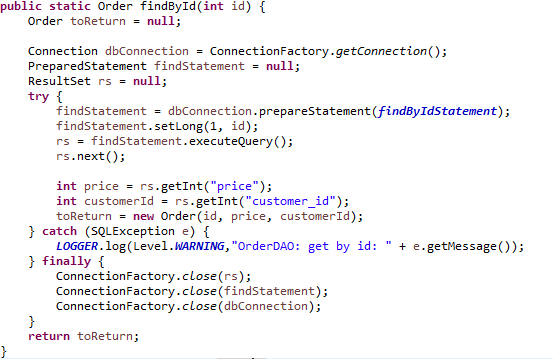
**ORDER Data Acces Object**



This class from the package dao is used to apply find(with different parammeters given) and insert methods.

In the beging we have a couple of strings that store the queryes that are used to to perform different actions on to the database. They have question marks in them that will be replaced with certain information like name id etc.

**findById**

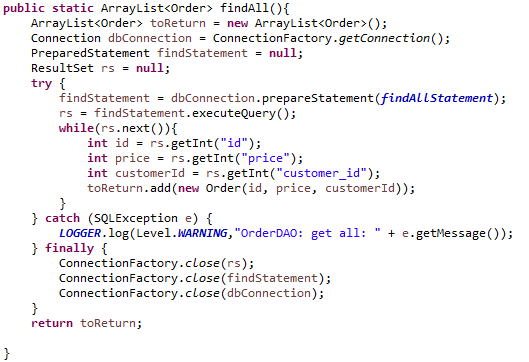


At the beginning of the class we create an object of type prder that will be returned at the end. In it will be stored the info of the order that is going to be searched by an id or let null if it is not found. An object of type connection is created that is used to establish the connection with the data base, a prepared statement that will store the query to perform the search in the db with the question marks and a result set that receives the result of the execution of the query.

The procces of searching is inside a try catch finally block. In the try block we put in the find statement the query and we update the question mark with the id with the help of the setLong method . The result set object get the exection of the query by using the executeQuery method and after that we go to the next column in the data base with the help of the .next method of the resul set interface.

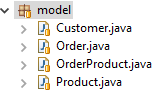
In the catch block we have a logger of level warning in case anything goes wrong in the try block. . In the finally block objects inserted statement and dbConncetion are closed using the close method of the connection factory.

**findAll**



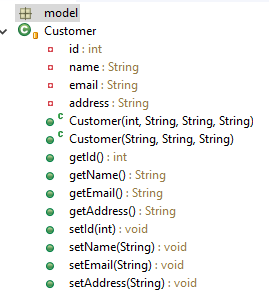
The find all method of the OrderDAO class is used to return all the orders in array list with ojects of type order. In the method an object of type coonection is created in order to establish the connection to the database and another object of type prepared statement that is initialized null. The process of deleting a customer from the database is putted in a tru catch finally block .

**Package Model**



In this package we have classes for Customer, Order, OprderProduct and product. In this classes we have the information each object has, constructors and getter&setters.

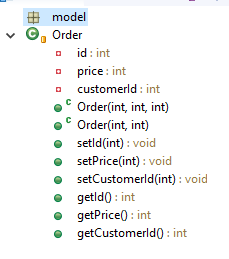
**Customer**



This Class is used to get the information and set it when we need it in other packages.

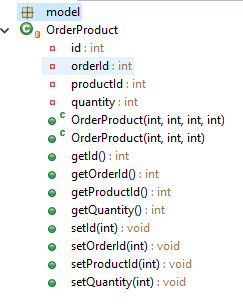
A customer has id, name, email and address. We have to constructors when we need one with the id and we don’t and the setters and getters for each field.

**Order**



This class is used to store, get and modify information about an order. It has an id, price and a customer id. We have 2 constructors, one for when we need the id and one when we don’t. After this there are setters and getters foar each field of the class.

**OrderProduct**



This class is used to store information about an order such as the quantity the id, the id of the customer that made it and the product id that it is in it.The class has 2 constructors, one when we do need the id and one when we do not. After that there are setters and getters for each field of the class.

**Conclusions**

Achieving such a program may be hard both in terms of algorithms, graphical structure. The best is to represent the customers and products as a tree type structure because this kind of structure makes it easier for some operations to be done: add, remove or search for an element from the structure.

For a better performance there should be implemented all cases where exceptions can occur and the application stops working due to an error made ​​by the user. Also, the division method should be done so it can calculate the reminder as well. Another thing that could be improved is the display so that it would be more elegant.