Technical University of Cluj-Napoca

Orders Management

-project documentation-

Pop Andrei

Group 30424

1. Objective

The objective of this project was to build an application which can manage orders from some customers. The application was designed such that a customer can see all the available products, their price and quantities for each of them and select one or more in order to be added into his order cart. The main objective was to simulate buying process that takes place online or on an installed app which communicates with a server and from which a customer can order different things. Also, another objective was to create the possibility for an administrator to manage the stock of products (update the price, change quantity etc). Also, the administrator has the rights to add, remove or update a new customer. In order to keep an evidence of the orders, their status can be updated only by the administrator. They can’t be changed or deleted by anyone because the information about a certain order might be needed at a certain point in time. Another objective was to create a template on which more functionalities can be added in the future. The application was designed such that the design will support new functionalities.

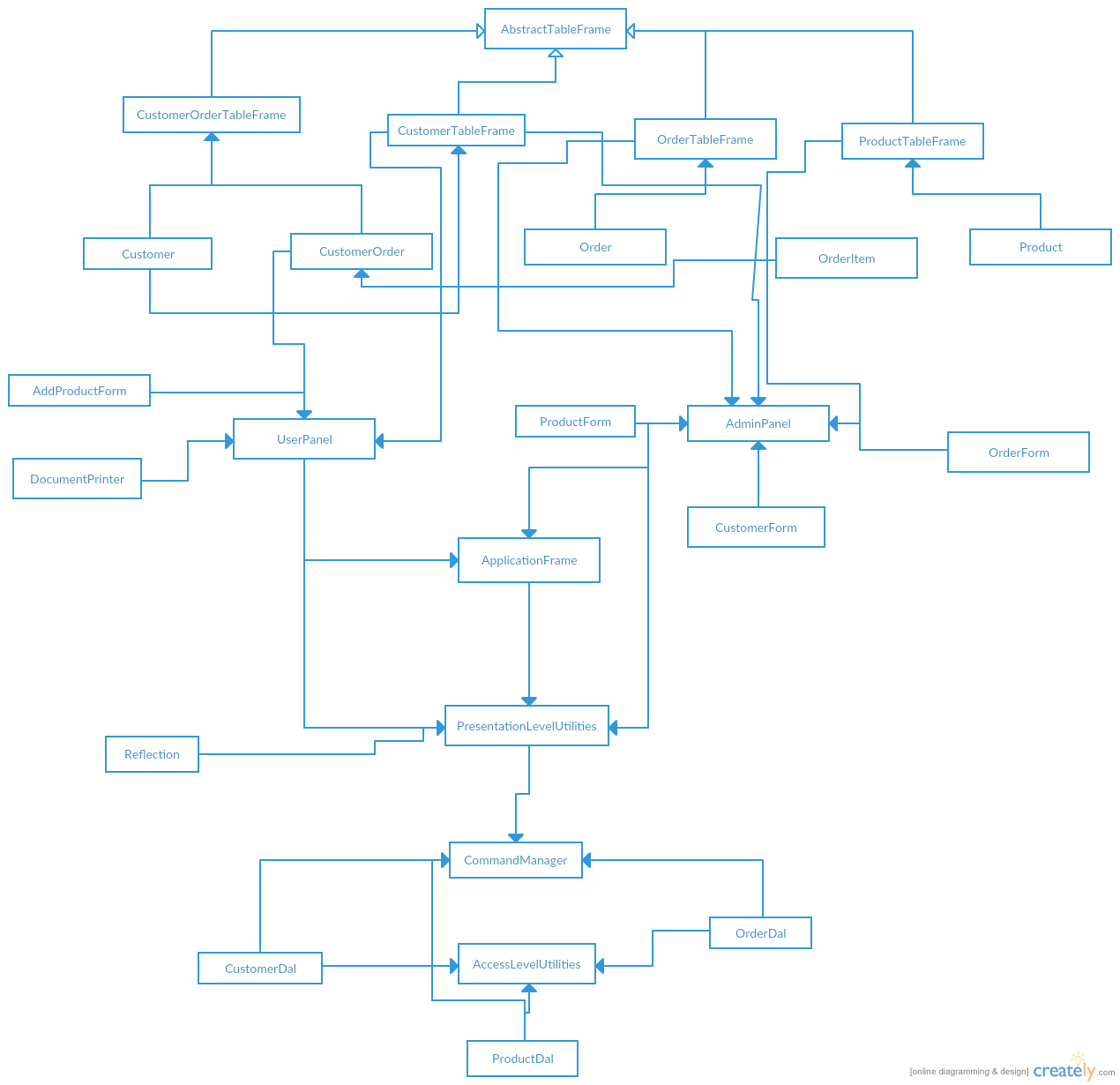
1. Problem analysis, modeling, scenarios, use cases

It was required to create an application which would manage the orders from the customers. In order to keep a clear evidence and to store new information, the application was developed on a relational database structure. The database contains all the necessarily information by the application. It contains information about the products, the customers, the orders that have been already submitted and the orders that are still opened and more products can be added. In order to communicate with the database, several layers were created. These layers are also good references in order to organize the needed classes. In order to make the communication with the database easier, each entity from the database, has a corresponding class in Java. So each time the user or the administrator are performing some operations, they also operate on the corresponding classes, not always directly with the database. The lowest layer of the application is the one that ensures the communication with the database. This layer is called ‘the access layer’ and it contains all the queries that will be executed in the database. Each of the entities has its specific queries, so for each entity, DAL class was created in order to store and execute when needed all the queries specific to a certain entity. The next level in the design of the application is the business logic layer. At this level, the business rules are checked in order to keep consistent data into the database. For example if a customer wants to buy a certain number of items of the same kind, there should be performed a check in order to find out if there are enough for the customer. The next level of the application is the presentation layer. At this level, the information is given to the user, and as a result of the user interaction with the application, new information is created. The information from the database is kept into JTables, in order to ease its displaying. There are six tables, three for each of the main panels: the user panel and the admin panel. On the user panel were added tables which display the information about the available products, the customers that can be selected for an order and the table which shows the order details such as the ordered products, their quantities and their price. On the admin panel, the customer table and the user table are the same, but the difference is that the user can modify their content by deleting, updating or inserting new information. Also, on the admin panel can be seen in a table all the orders that have been submitted by the users. An user can only select a product and add it to the chart. The administrator has all the rights to operate with the information, so he can insert, update or delete information. In case when the application is used by a regular user, after performing an operation, all the tables that store information are updated, so the user can also see the changes that were performed by the user. The same things happens when the administrator operates. All the tables are refreshed, so the user can see if there were added more products or customers or their information was updated. All the actions of the user are restricted such that the user can’t change the already existing information. A very important moment is when the user decides to submit the order containing the desired products. When the submit button is pressed, the information contained into the order area is saved into the database, and the user can’t see anymore the order and its details. The only one who can see the order details is the administrator. The administrator can also change the status of the order but not more than that. One of the important aspects is that the user can never add to its cart more products than there are available in the database. Due to this fact, when the user submits the order, the validation of the products has already been done, so only saving the information about the order is needed. The administrator should always add new valid information into the database, so it is supposed that the administrator is a specialized user. Unlike the administrator, the user doesn’t necessarily have to be an experimented user. The only things that are required are to understand how to add the products that he wants into the cart and how to insert the number of items that he wants of that kind. Besides that, the GUI is friendly at is displays if the user gives a bad input. The user and the administrator can’t operate at the same within the application. They have access only sequentially. In order to view the modifications done by the user on the admin panel, after each operation, all the tables from the user panel need to be refreshed. It works the same after each modification done by the administrator.

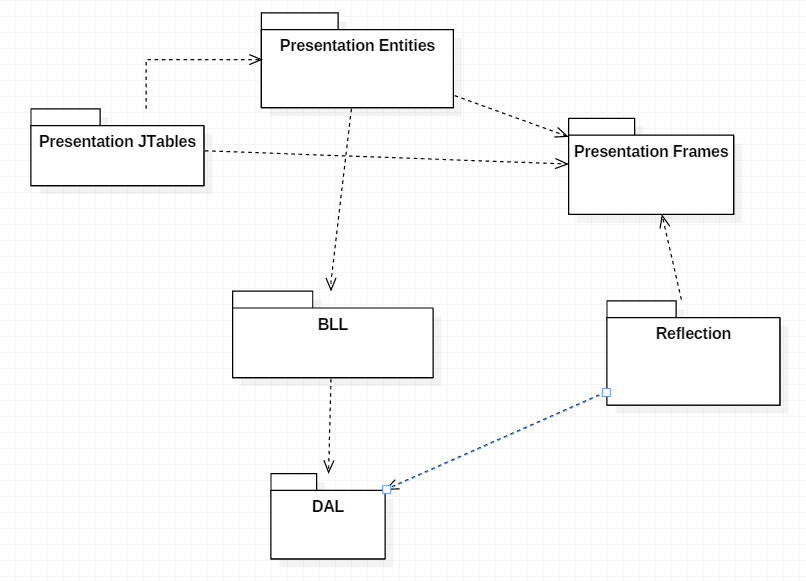
1. Design

3.1 UML Diagram

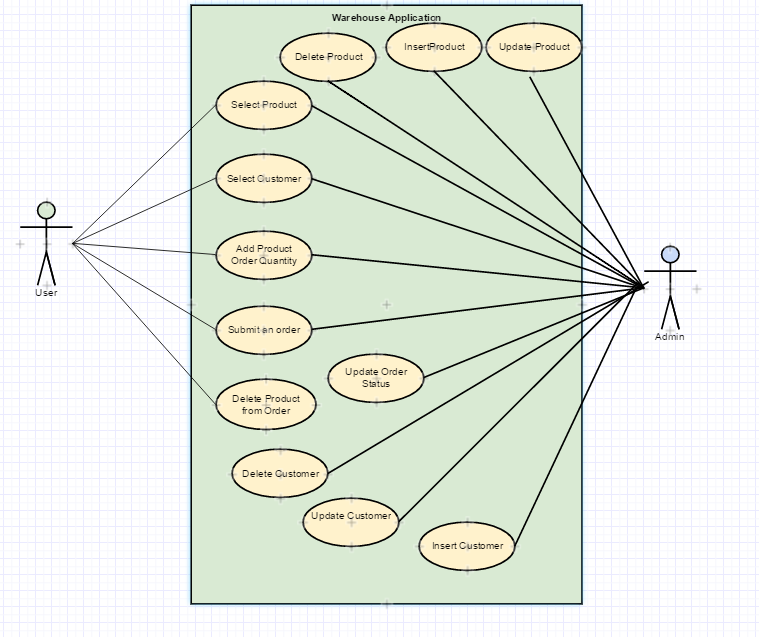
There are 27 classes, between which the dominant relationship is aggregation. There are some classes that extend from upper classes. The application was built in such a way to keep somehow the independence between the classes.



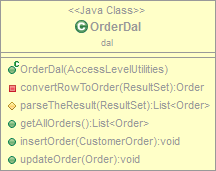
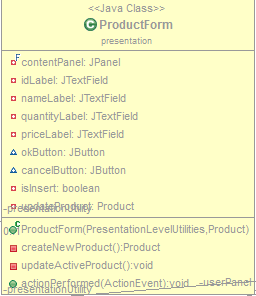
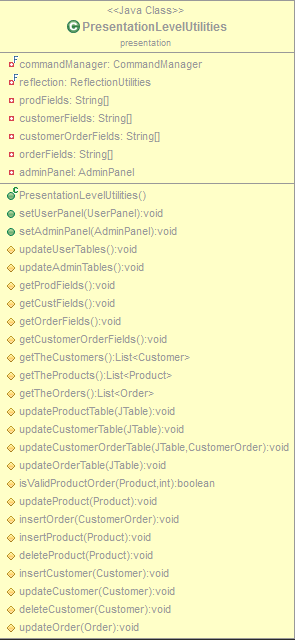
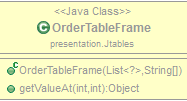
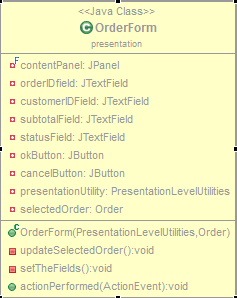
3.2 Package diagram

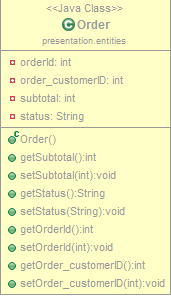


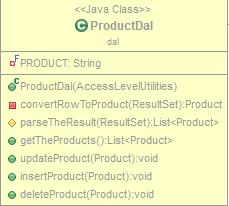
3.3 Use case diagram

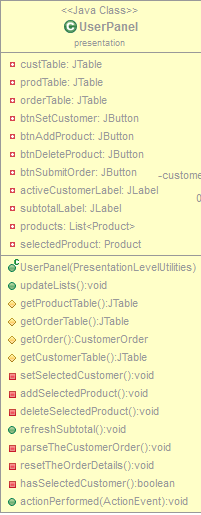
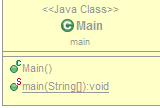
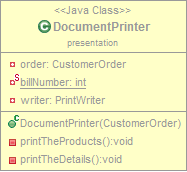
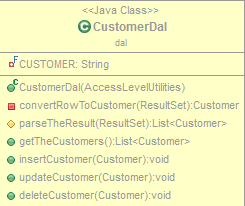
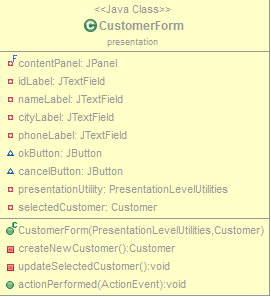
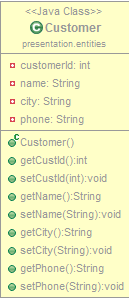
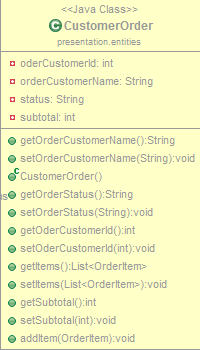


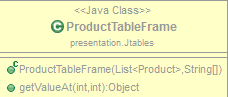
3.4 Class design

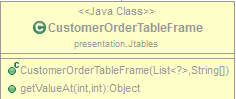
 

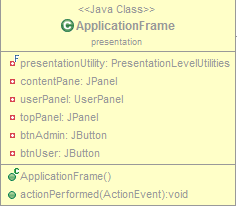
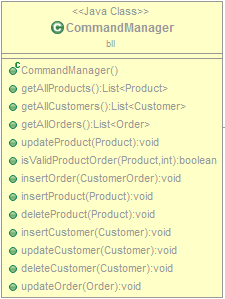
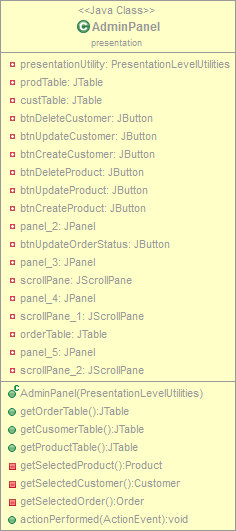
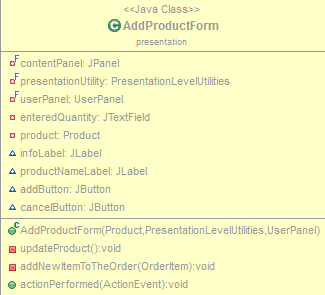
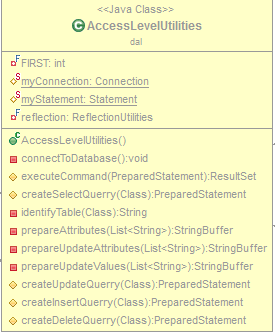
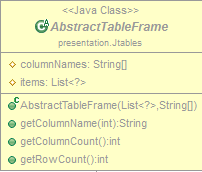












The classes are organized into seven packages. Each of the packages having a certain functionality. Some of the packages are grouped in layers. The objective was to have three main layers: presentation level, business logic and access level. Of course there are some extra packages that were created and which are not part of any of these three layers.

1. The presentation level

It contains three different packages, each of them grouping classes with a specific functionality. The packages contained by this level are:

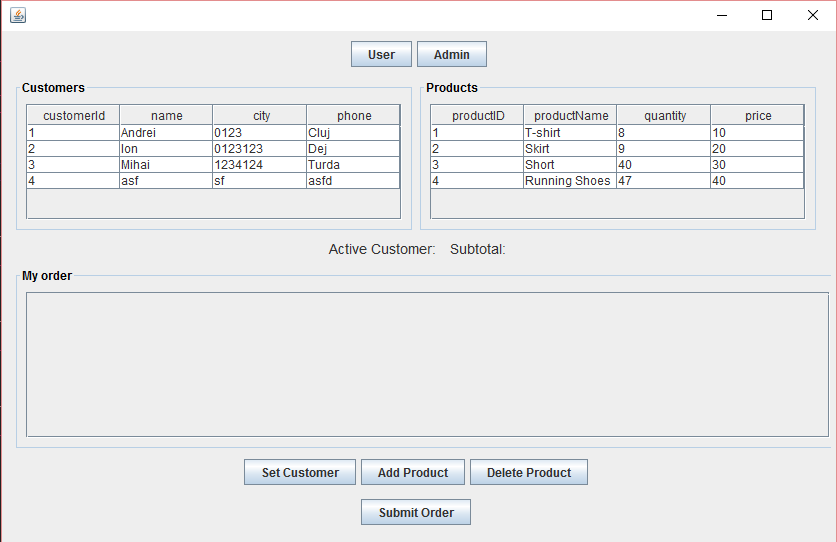
* **Presentation package**: - contains all the classes that compose the GUI. These classes implement JFrames, JDialogs etc and they are:
* AddProductForm: this class is implementing the frame for adding a new product. This frame contains some fields which are filled by the user when ordering a product.
* AdminPanel: contains all the data and all the components that should be seen by the administrator. The data is organized into three JTables on which some operations can be performed.
* ApplicationFrame: contains the UserPanel and the AdminPanel. This is the root frame on which everything is added directly or indirectly through other components.
* Customer Form: implements a JDialog that is used every time when the administrator wants to add or update a new customer. After filling the text fields of the form, the admin has the possibility to submit the changes to the database.
* DocumentPrinter: this class is used as a printer. When a user submits his order, a bill with all de buying details needs to be printed out. This class will generate a new file in the location of the project on the disk which contains all the information about the order.
* OrderForm: works pretty much as the CustomerForm or the ProductForm. The only difference is that the administrator can only update the status of an order. He can’t change the order details.
* PresentationLevelUtilities: this class is used in order to implement all the utilities required by the different components of the GUI. It contains all the specific functions which can be called by each of the GUI components. The most of the times, this class will pass almost all the requirements to another class from another package: CommandManager
* ProductForm: this class implements a JDialog which will pop-up every time the admin wants to add or update a new product. After filling the fields of the form, the information can be saved into the database by pressing the “add” button
* UserPanel: this panel contains all the components and the data that can be seen or accessed by the user. It contains three JTables, and a few other buttons that are needed in order to perform operations.
* **Presentation entities package: -** contains all the entities that store the information. These entities are a mirror of the database. Each existing entity in the database has a corresponding class in this package.
* Customer: the class that stores all the information about a customer that exists in the database. All the attributes correspond to some fields into the database.
* CustomerOrder: this class is an auxiliary class that was created in order to keep the information about the order of the customer. It contains a list of products which were ordered by a certain user who’s name was also stored in this class.
* Order: this class stores the information about the orders that have been previously submitted by the users. Its fields correspond to the the fields of the order table from the database.
* OrderItem: this class is also an auxiliary class that was created in order to ease the working with the customer orders. Each time the customer is adding another project tot his order, a new OrderItem is inserted into the CustomerOrder class. This class contains two attributes: product and quantity.
* Product: this class is the corresponding of the product class from the database. It contains the same fields as the product entity from the database.
* **Presentation JTables package: -** contains all the models for the tables that are to be added on the two panels of the application. Each of the tables will display some specific information. The role of each of the below models is to feed data to the tables.
* AbstractTableFrame: this class is an abstract class which contains some methods and data which is inherited by the subclasses. This class also extends the AbstractTableModel predefined class.
* CustomerOrderTableFrame: it is the specific model that is implemented by the CustomerOrderJTable in order to display all the details about the current order of a customer.
* CustomerTableFrame: this class implements the model that will be implemented by the Customer Table. The role of this class is to feed the necessary data to the Customer JTable.
* OrderTableFrame: it has the same functionality as described above. The only difference is that it is providing data for the OrderJTable.
* ProductTableFrame: the same functionality as the CustomerTableFrame but the difference is that it provides data for the Product JTable.
* **Main package:-** contains only the main class which will instantiate the ApplicationFrame in order to start the application’s interaction with the user.
* **Business logic layer package:**- contains all the methods and provides all the functionalities needed by the user interface classes. In this layer, the commands coming from the above package(the presentation package) are checked, and if they’re valid, they will be passes down to the next level, which is implemented by the Data access layer. It contains four classes:
* CustomerDal: this class will implement all the methods that perform operations in order to get, update or delete information about the customers which can be found into the database. It is the class which does the linking between the entity from the database and the application entity. Usually, all the classes from this package will create and execute specific sql queries in order to communicate with the databse.
* OrderDal: has the same functionality as described above. The only difference is that it does the linking between the database and the order entity. Also, all the queries will be specific to the order entity from the database.
* ProductDal: this class has the same functionality as CustomerDal but it realizes the linking between the Product entity and the product entity from the database.
* AccessLeveUtilities: it provides some utilities that are needed by all the entity specific classes. Usually this class will contain methods who will generate and execute queries that are required by the other classes.
* **Reflection package:-** contains only one class which implements some methods that are used when the reflection of a class is needed. In this application, the reflection is used when creating the JTables and when performing operations on the database. The methods from the Reflection class are used to get the fields of a class that was given as a parameters. The fields are returned as an array of strings.

3.4 Data structures

No special data structures have been used. All the information was kept into the databse, and when the information is needed from the database it is stored into Linked Lists. A special data structure that was created in order to ease the dealing with the customer orders is the OrderItem structure. It contains details about the items of the order, so each OrderItem object contains the ordered product and the quantity ordered.

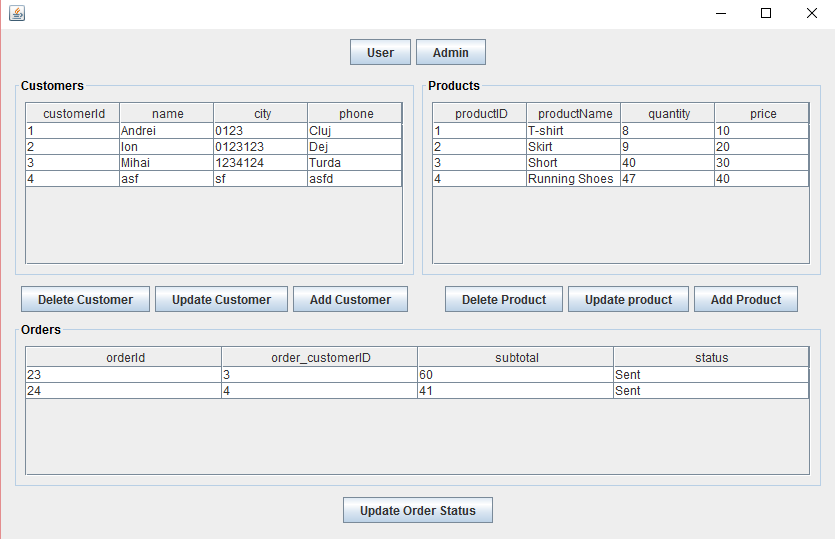
3.5 Graphical user interface

The graphical user interface is composed by two main panels which are added depending on the actions on the main frame. The user panel contains all the information a that can be used by a regular user who only wants to order some products. As it can be seen below, there are three tables, each of them containing some information. The My order table contains the details about the user’s order.



User view of the graphical user interface

The admin userPanel contains only information that can be seen and accessed by the user. The user has also some auxiliary operations that only him can perform. These operations are: adding, updating or removing an user ore adding updating or removing a customer. The administrator can also modify the status of the orders that have already been placed. The user can’t submit any orders because he can always access the user account.

****

s

Admin view of the graphical user interface

1. Future improvements

The application was created in such a way that more functionalities can be added in the future. For example, the administrator can’t modify an order once it was submitted. He can only modify the status of the order such that it will be known if that order has been sent or not. Another future improvement would be to add some functionalities for the administrator to see the order’s details and to modify these details. One of the most important improvement that can be made is working with usernames and passwords. This way the security will be increased and only known and selected user will be able to access the information about the products and be able to order. Also the administrator should have the rights to create user accounts such that only the user who have granted access will be able to use the application. The administrator should also have the possibility of creating another administrator account but only one administrator will be able to manage all the accounts(i.e to delete other administrator accounts). Another improvement would be to make the application to work on multiple synchronized threads such that, each user would have it’s own thread. A problem that can occur if we work this way, is that the data from the tables needs to be updated after each user operation. Also the data can become inconsistent if more users are modifying the same data at the same time. The multiple threading would make the application work much faster but there’s a lot of overhead because of the multiple thread which operate on the same data.

1. Conclusions – what I have learned

During the developing of this application I have learned how to store data in a database using Java. I have also learned how to design the code on layers such that the code developing would be more easy to perform. Also, due to this structure, more functionalities can be added without making changes in the packages structure or in the code classes. Another thing that I’ve learned is how to use JTables in order to represent data that is stored into the lists. In order to implement the JTables I have used the AbstractTableModel class and customize it for each table. When I’ve created the graphical user interface I have learned how to position elements by setting the distances between them in pixels.

The application can be developed further and adapted such that it can be used in multiple fields. For example it can be used in a sport online shop and also can be updated in order to be used by a library. The changes should be done only at the level of the fields in database and at the content level. It should always be customized such that the user can find very fast whatever he wants.

1. Bibliography

Books:

- Joshua Bloch, Effective Java (2nd Edition);

- Kathy Sierra, Bert Bates, Head First Java (2nd Edition), O'Reilly Media;

- Barry Burd, Java for Dummies (5th Edition), Wiley;a

Websites:

- http:// stackoverflow.com/

- https:// www.oracle.com/java/

-http:// www.oracle.com/technetwork/articles/javase/index-142890.html

- http:// www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

- http:// christoph-burmeister.eu/?p=1556

- http:// theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/

- http:// tutorials.jenkov.com /