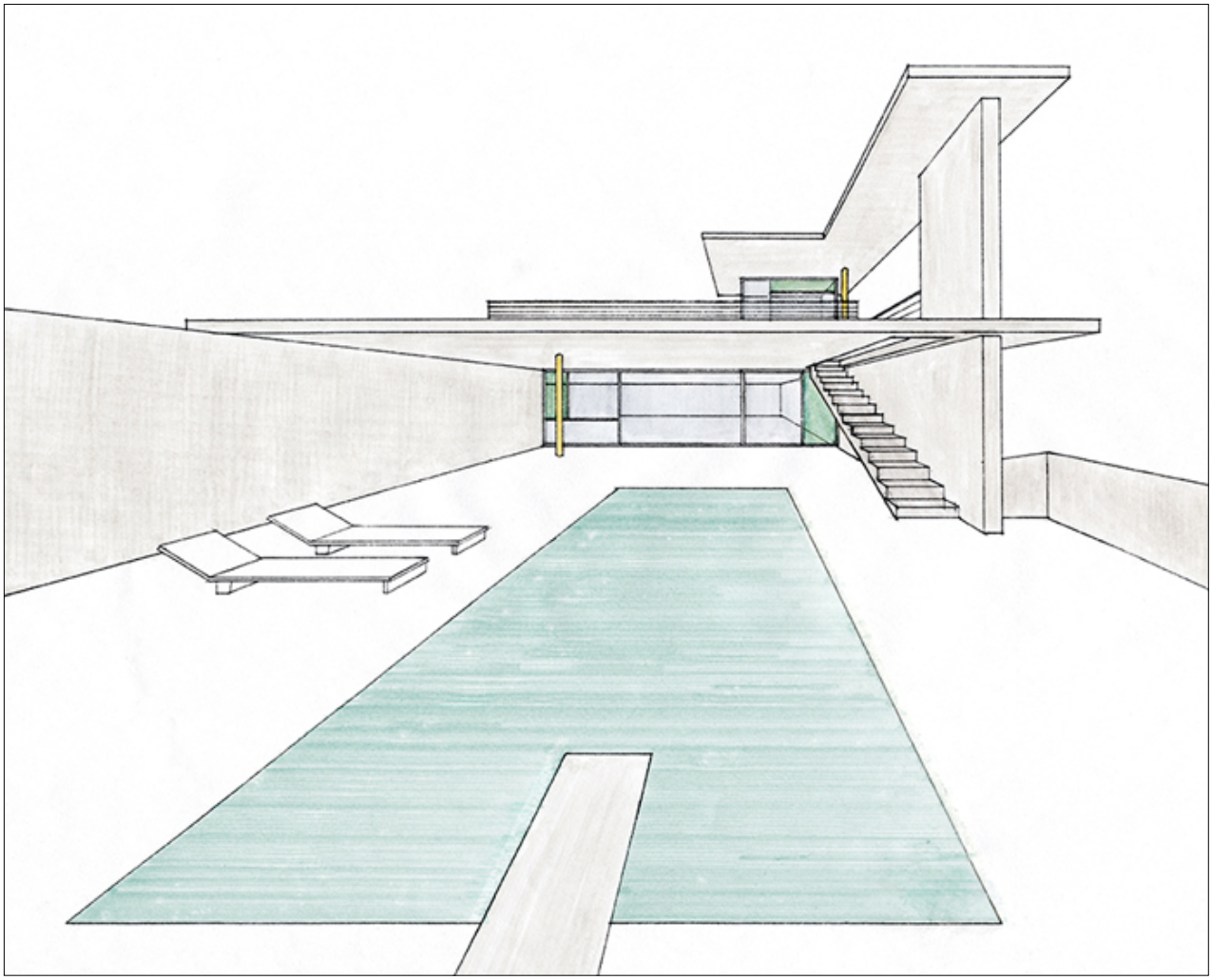
TECHNICAL UNIVERSITY OF CLUJ-NAPOCA



Warehouse Management using SQL

Prepared for: Dorin Moldovan

Prepared by: Alexandru Risa

15 April 2020

OBJECTIVE

Requirements

Consider an application **OrderManagement** for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application should be structured in packages using a layered architecture presented in the support material and should use (minimally) the following classes:

- **Model classes** - the data models of the application
- **Business Logic classes –** implement the application logic
- **Presentation classes –** implement the user input/output
- **Data access classes -** implement the access to the database

  The application should allow processing commands from a text file given as argument, perform the requested operations, save the data in the database, and generate reports in pdf format. Other classes and packages can be added to implement the full functionality of the application.

  Implement a parser to read commands in the Presentation layer (instead of the standard graphical user interface), and a pdf file generator to generate reports, according to the following examples:

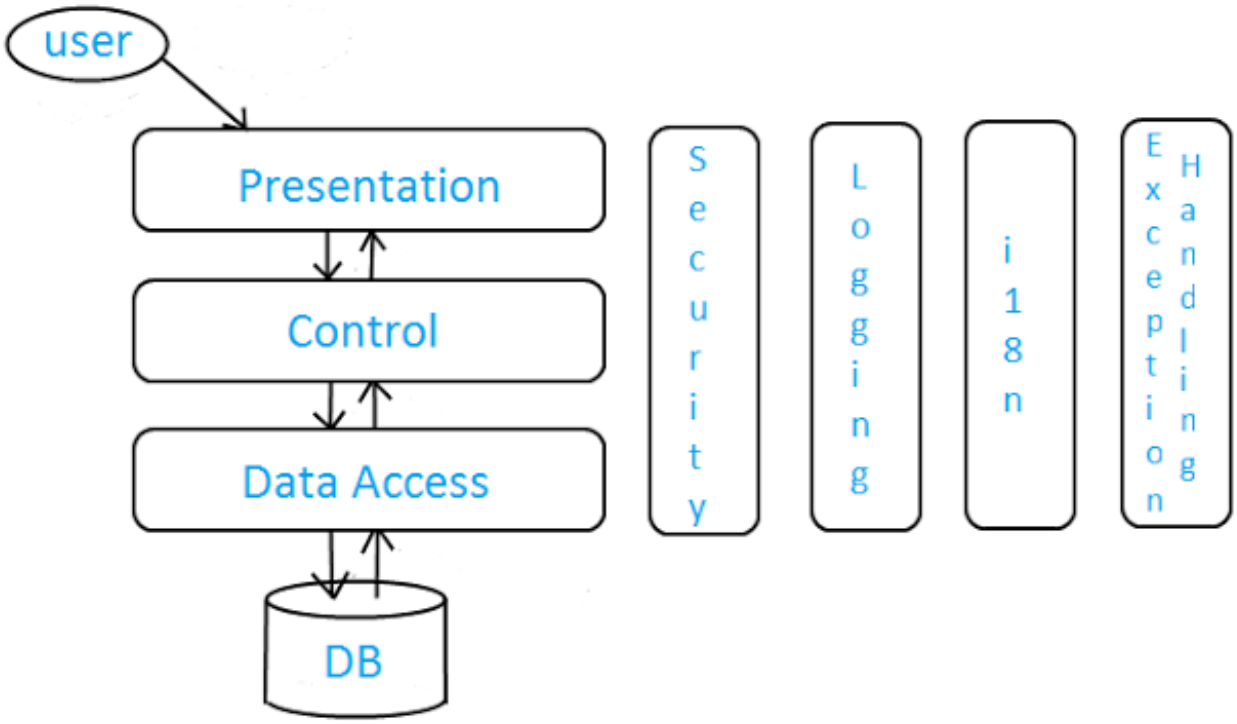
Command name	Command Syntax	Description
Add client to the database	Insert client: Ion Popescu, Bucuresti	Insert in the database a new client with name Ion Popescu and address Bucuresti
Delete Client from the database	Delete client: Ion Popescu	Delete from database the client with name Ion Popescu
Add product to the database	Insert product: apple, 20, 1	Add product apple with quantity 20 and price 1
Delete product from the database	Delete product: apple	Delete product apple from database
Create order for client	Order: Ion Popescu, apple, 5	Create order for Ion Popescu, with apple quantity 5. Also update the apple stock to 15. Generate a bill in pdf format with the order and total price of 5
Generate reports	Report client Report order Report product	Generate pdf reports with all clients/orders/products displayed in a tabular form. The reports should contain the information corresponding to the entity for which reports are asked (client, order or product) returned from the database by a SELECT * query, displayed in a table in a PDF file.

Design considerations

LAYERED APPROACH:

In a standard enterprise application which has a database and graphical UI (web or desktop), there are some typical layers which constitute the application. In this article we will mention those layers and give some instructions about them.

The general diagram is as below:



There are some important properties about that diagram:

There are vertical and horizontal layers. Vertical layers may be thought as general application service libraries which can parallely work on all horizontal layers and independent from each other. However, a horizontal layer's subject of interest is only its neighbour (top and bottom) horizontal layers, and they work sequentially from top-to-bottom and bottom-to-top. Lastly, user can only interact with topmost horizontal layer.

Horizontal layers:

- **Presentation**: Every enterprise application has a UI, in fact graphical UI. This UI can be web or desktop based, which doesn't matter. The rule is simple. UI takes user action and sends it to the controller. And at the end it shows result taken from controller to the user. UI can be implemented according to MVC, MVP, MVVM or another approach.

- **Control**: Handles business logic of the application. Takes info from user and sends it to DB layer (DAO or ORM framework) and vice versa. Abstraction type of controller may vary (a separate control and business layers for example) according to the application parameters or development patterns (MVC, MVP, MVVM, ...) but the main idea remains the same.

- **Data Access**: Database handling layer of the application. It may contain entity definitions, ORM framework or DB connection codes having SQL sentences, according to the abstraction decision. Its role is getting data from controller, performing data operation on database and sending results again to controller (if result exists). Database independence is a very important plus for this layer, which brings flexibility.

REFLECTION:

- Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection.

- Java Reflection is quite powerful and can be very useful. For instance, Java Reflection can be used to map properties in JSON files to getter / setter methods in Java objects, like **Jackson, GSON, Boon etc.** does. Or, Reflection can be used to map the column names of a **JDBC** ResultSet to getter / setter methods in a Java object.

CREATING PDFS:

- Creating a pdf with a use of the iText library is based on manipulating objects implementing Elements interface in Document (in version 5.5.10 there are 45 of those implementations).
- The smallest element which can be added to the document and used is called Chunk, which is basically a string with applied font.
- Additionally, Chunk‘s can be combined with other elements like Paragraphs, Section etc. resulting in nice looking documents.
- We might face a problem when we would like to add a table to our pdf. Luckily iText provides out-of-the-box such functionality.
- First what we need to do is to create a PdfTable object and in constructor provide a number of columns for our table. Now we can simply add new cell by calling

- Now we can simply add new cell by calling the addCell method on the newly created table object. iText will create table rows as long as all necessary cells are defined, what it means is that once you create a table with 3 columns and add 8 cells to it, only 2 rows with 3 cells in each will be displayed.
- In order to apply permission using iText library, we need to have already created pdf document. In our example, we will use our iTextHelloWorld.pdf file generated previously.

Implementation considerations

- Use the Java programming language
- Use the SQL driver jar to connect to database
- Use a scanner to loop read from stdin
- Use lists instead of arrays
- Use foreach instead of for(int i=0...)
- Implement classes with maximum 300 lines of code (except for the UI classes)
- Implement methods with maximum 30 lines of code
- Use the Java naming conventions (https://google.github.io/styleguide/javaguide.html)
- For the create order command:
  - Create a bill for each order as.pdf file
  - The product stock will be decremented after the order is finalised.
  - In case that there are not enough products, the order will not be created and the PDF document representing the bill will not be generated. In this case, instead of the bill, a PDF document will be generated in which an under-stock message will be written.
- Reports as .pdf files, generated for running the text file with commands
- Use **reflection techniques** to create a generic class that contains the methods for accessing the DB: create object, edit object, delete object and find object. The queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection.
- Database Structure (more than three tables)

Testing considerations

- Use the given commands for testing the application.

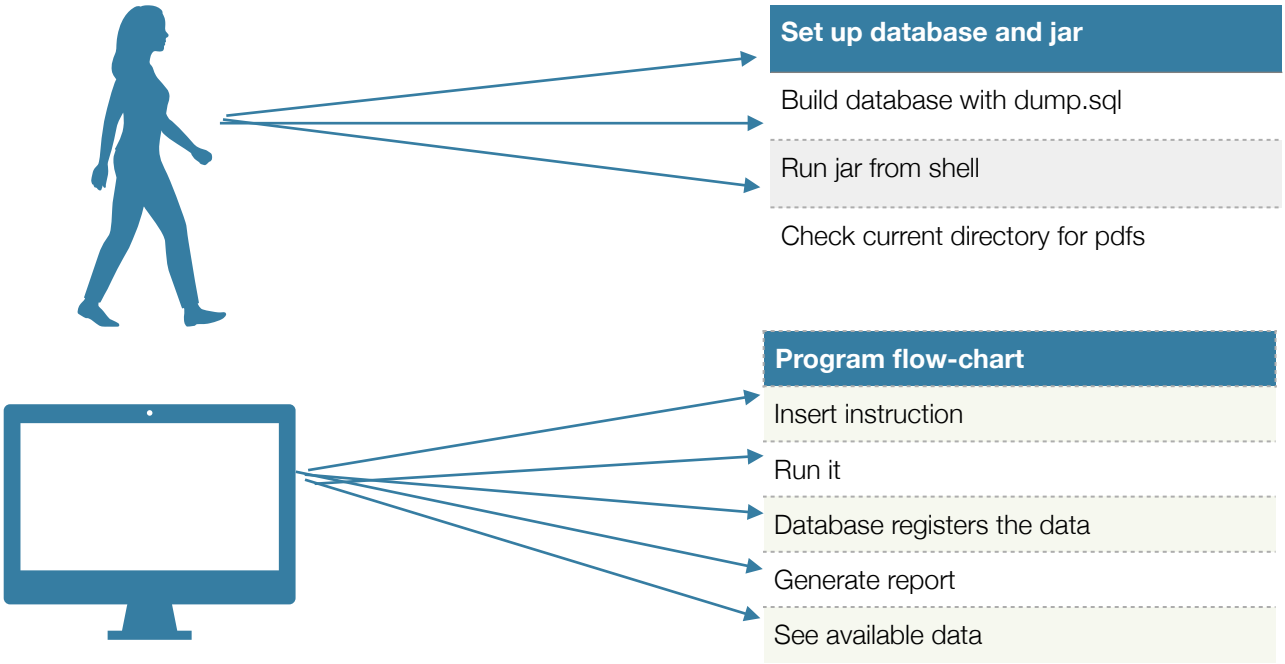
ANALYSIS

In order to achieve the desired program, one must recall the basics of working with databases, such as creating a database, initialising a database, updating, inserting, selecting, deleting, working with procedures and triggers.

In order to comply to the fundamental principles of OOP the program contains several classes within the several layers:

- Client, Product, OrderView
- ClientBL, ProductBL, OrderViewBL
- ClientDA, ProductDA, OrderViewDA
- Start

SCENARIOS



- One possible scenario is an invalid state of the input file, i.e. not respecting the given format or perhaps the input of an inexistent path;
- Another scenario would be to have no connections available, case in which we abort the operation altogether;
- Another scenario would be when there are no reports to generate due to the fact that the tables are empty;

- Perhaps as the database grows in data, it could become too large to be saved in a pdf and also have a use without being filtered;

PROGRAMMING DECISIONS

- The WarehouseManagement application is meant to hold all management a database;
- Each clients holds the following field: name and city;
- Each server holds an array list of the clients it is processing, the waiting time for the next client to be added and a sum of all waiting times for all clients processed;
- The run server runnable objects hold the server they run and two boolean variables meant to pause and stop the run() function of the thread;
- The Scheduler makes sure the Clients are added to the minimum waiting server and updates every server at every time unit(which is 200ms);
- The interval class aids the encapsulation process so that we can hold both the upper and lower limits of the arrival and service times in one object each;

IMPLEMENTATION

Product Class

Holds the data model for the product table in the database

Client Class

Holds the data model for the client table in the database

OrderView Class

Holds the data model for the order view table in the database

ProductBL Class

Adapts the ProductDA methods to the java logic

ClientBL Class

Adapts the ProductDA methods to the java logic

ProductDA Class

Contains the logic for managing the data within the database, regarding the Product class

ClientDA Class

Contains the logic for managing the data within the database, regarding the Client class

OrderViewDA Class

Contains the logic for managing the data within the database, regarding the OrderView class

AbstractDA Class

Contains the generic logic for managing the data within the database, regarding all classes;
Several methods are in here:

- Methods for creating the String instruction we want to execute: createSelectInstruction, createInsertInstruction, createUpdateInstruction, createDeleteInstruction;
- Methods for establishing the connections and executing the instructions: select, insert, delete, update, where select generates the required report and also returns the pdf object to be afterwards written to the hard drive;

Start Class

Contains the main class:

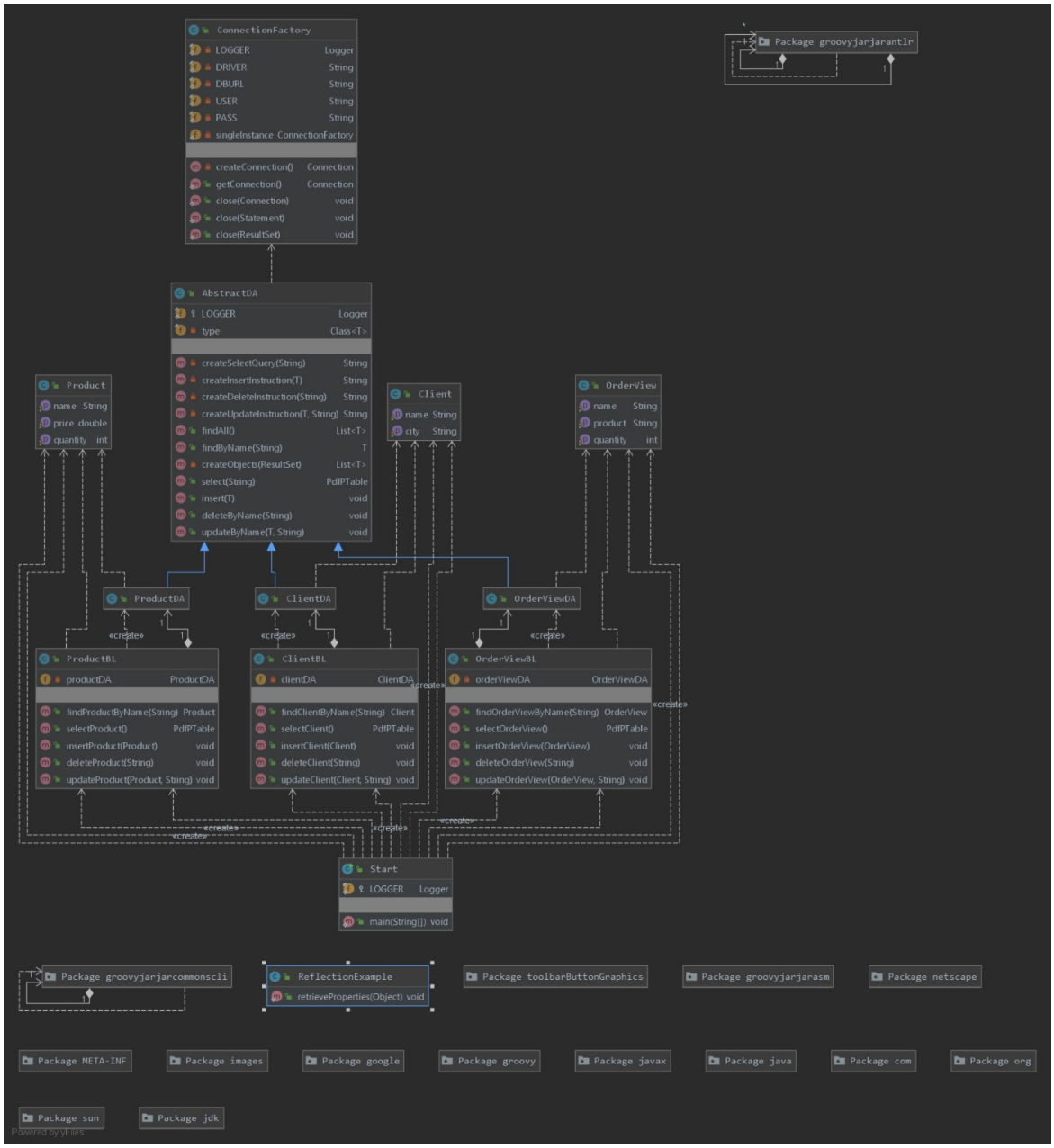
- It creates the pdfs for orders and reports using the iText library and outputs them to the hard drive;
- Loops until the end of file is reached and executes all instructions;

CONCLUSIONS

Conclusions include the fact that a well thought planning of the classes easies the work afterwords and saves time. With the increase of difficulty in projects, one must approach the problem with a smarted perspective and not just jump to writing code right away. Thus, starting with the UML diagrams gave me time to reflect upon the proper approach to the implementation of the desired program.

Perhaps the most interesting technique I've learnt out of this project is the use of the JDBC driver, which has proven itself to be amazingly useful and powerful. Working with java and sql is definitely something I wish I had known sooner. Also, working with Maven is always astonishing due to the convenient option of adding a vast number of plugins to your project just by adding their dependencies.

Improvements that could be made upon the project include a slight modification so that the dbms supports a larger variety of classes and tables, which could easily be achieved by altering the database structure in the function that converts the string user input to the respective polynomial object. Another could be perhaps going further into the mathematical side of computer science and implement an algorithm for the calculation of more efficient indexes in the MySQL database.



BIBLIOGRAPHY

1. http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/JDBC.pdf
2. https://stackoverflow.com
3. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
4. http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
5. https://dzone.com/articles/layers-standard-enterprise
6. https://www.baeldung.com/javadoc