**Commercial Trading Project**

**Design Patterns**

**Group Members:**

Luis Gustavo Rico Nani Castro - Student Number: 2015325

Marcionedes Cardoso De Abreu - Student Number: 2015254

Michael Faria - Student Number: 2015209



Table of Contents

[**Design Patterns and Data Structure** 3](#_Toc513144015)

[**Program Requirements** 5](#_Toc513144016)

[**Walkthrough** 6](#_Toc513144017)

[**Validation** 7](#_Toc513144018)

[**Outputs** 7](#_Toc513144019)

[**Functional Analysis** 9](#_Toc513144020)

[**Class Diagram** 11](#_Toc513144021)

[**General Report** 12](#_Toc513144022)

**Design Patterns and Data Structure**

* **Factory Pattern**

Factory design pattern is used when there is a super class with multiple sub-classes and based on input, we need to return one of the sub-classes.

Factory Pattern is used on the project to create Companies (A, B and C) depots and store them into ArrayLists. Further the ArrayLists will be used to retrieve variables (stockA, stockB, stockC, allowance, productCost, delivery) that are needed from the depots.

* **Singleton Pattern**

Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine. The singleton class must provide a global access point to get the instance of the class.

Singleton Pattern is implemented on the menu class because we don’t need to instantiate it more than one time on the program.

* **Facade Pattern**

Facade Pattern hides complexity of the system and provides an interface to the client. This pattern involves a single class which provides simplified methods required by the client and delegates calls to methods of existing system classes.

The facade pattern will be used on the project to hide the creation of depots. Therefore the client won’t be able to see all the complexities behind it. Another complexity that the CommercialTradeFacade class hides is the menu class which manages the trades.

* **Data Structure ( ArrayList)**

**ArrayList**isa resizable-array and is widely used because of the functionality and flexibility that offers. ArrayList can dynamically grow and shrink after addition and removal of elements. The ArrayList will be used on the project to store the depots (from companies A, B and C ) with all of their attributes.

**Program Requirements**

The program requirements of Commercial Trading Project provide an overview. The program allows 3 companies to trade among themselves.

* Each company produce only one type of product (Company A produces and supplies only “A’s”, Company B produces and supplies only “B’s” and Company C produces and supplies only “C’s”);
* Each company can only buy the other two companies’ products (i.e. Company A can buy B’s and C’s, Company B can buy A’s and C’s and Company C can buy A’s and B’s);
* Each company will have 100 depots;
* Each depot can hold the Minimum of 15 and Maximum of 50 of their own products (number will be generated randomly at runtime);
* Each depot can hold the Minimum of 3 and Maximum of 40 of other companies’ products (number will be generated randomly at runtime);
* The product price and delivery cost will be generated randomly (between €1 and €10) on every transaction;
* Each depot has an initial cash allowance (randomly generated between €50 and €100);
* Every depot must attempt to trade with every other depot (i.e. every depot “A” must attempt to trade with every depot “B” and every depot “C” and so on).

The user will have the ability to choose between two options:

* Choose company A company B or company C;
* Choose to trade autonomously.

**Walkthrough**

The following steps are an overview of how the program is going to work from the start until the end.

First Program execution will display a Message to the user*:*

“Welcome to Commercial Trading!”

“Please type your name”

After typing the name the user will be prompted two options:

1. **“Enter option 1 to choose a company”**

* Company A
* Company B
* Company C

After the user selected the desired company A, B or C, the program will make the trades automatically by checking if a trade is possible between depots.

1. **“Enter option 2 to trade autonomously”**

After the user selected the autonomously option the trade will happen normally but the output will be different as the user has not selected a company (A, B or C). The output will only display the profit and the loss of the companies.

**Validation**

To check weather a trade is possible or not the following parameters will be checked:

* If the value of the purchase leaves the depot with less than €50, it won’t complete the transaction;
* If a depot reaches its minimum native product (15), it won’t be able sell more products;
* If a depot reaches the maximum of other companies’ products (40), it won’t be able to buy more products.

**Outputs**

After the trades happen the following outputs will be shown:

* Case 1:

The user has chosen a company (A, B or C)

The first output will display detailed information on each user depot that traded for each transaction completed:

* Depot buying products;
* Depot they are buying from;
* Cost of products;
* Cost of delivery;
* Total cost of doing business.

The second output will display detailed information on the user’s company:

* Total trade for each company;
* Total amount of each product purchased;
* Total cost of purchase for products;
* Total cost of delivery for products.

The third output will display detailed information on finances:

* Profit and loss for each company;
* Company that spent the most and company that made the most.
* Case 2:

The user has choosen autonomously

The output will display:

* Profit and loss for each company.

**Functional Analysis**

Functional analysis is the systematic process of identifying, describing, and relating the functions a system must perform in order to to be successful. It does not address how these functions will be performed.

To the trade became achivable on the program a series of processes including creation of variables and different uses of data structures and methods in java must be implemented.

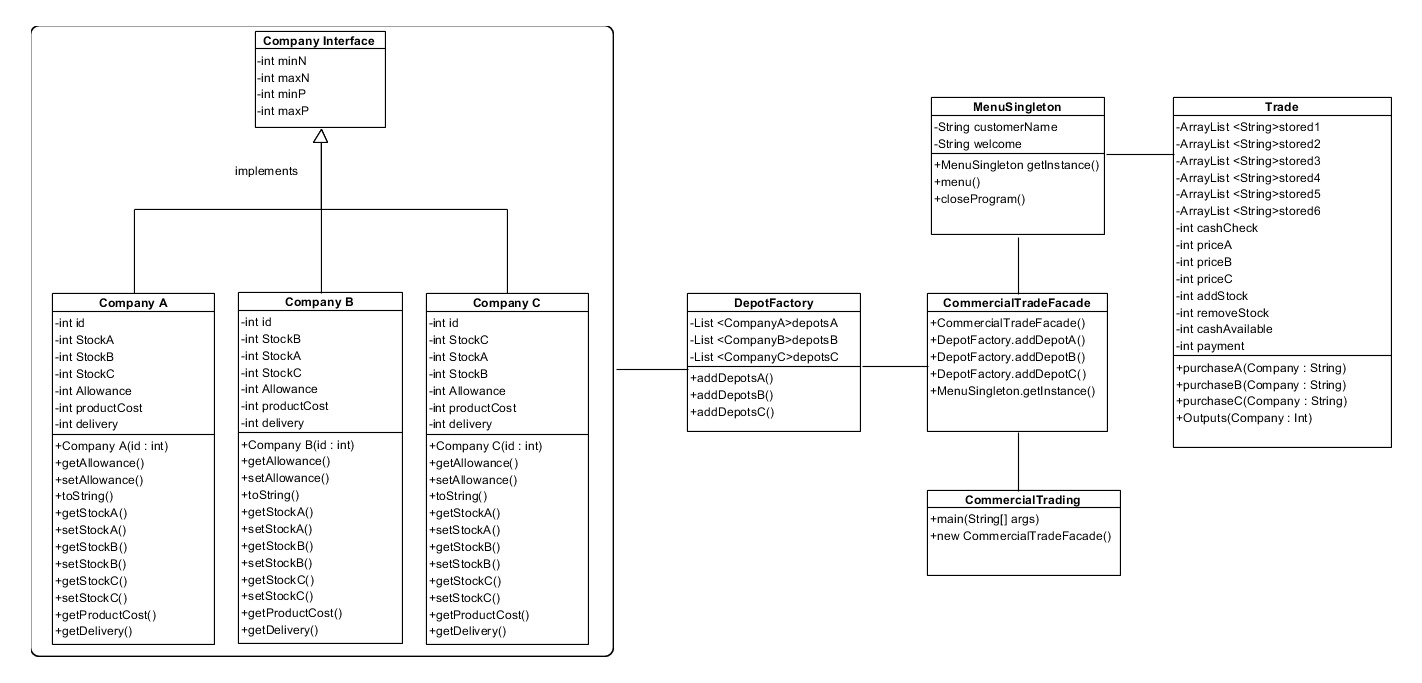
The start of creational process on the program takes Classes CompanyA, CompanyB and CompanyC implement Company Interface. Further DepotFactory Class instatiates a new Company (A, B and C) one hundred times each and stores into an ArrayList using a for loop. Every time it loops a new id is given to the new depot and the variables of each depot are generated randomly providing different values for variables.

Next step is to figure how the depots trade products among themselves. A Trade class is introduced to handle such task, an Algorithm must be created to make these trades feasible. The algorithm is based on a series of checks, first it checks if the company has enough cash to trade, secondly it will check if the company can store more products that they are trying to buy and for the last checks if the company they are buying from has enough products to sell. If the depot pass trough all the checks, it makes the trade until the depot runs out of money or stocks reaches its requirements.

Last step is printing the information of each sucesseful transation which occurs after the trade algorithms runs. The first output will display detailed information on each user depot that traded for each transaction completed. The second output will display detailed information on the user company that traded. The third output will display detailed information on finances.

**Figure 1 shows the paths to the result desired.**

**Figure 2 shows the checks passing through a funnel before the trade actually happens.****Class Diagram**



**General Report**

The Commercial Trading project had many phases and challenging tasks. At the beginning it was difficult gather all the requirements and implement design patterns on the program.

The project was built on the brainstorms and meetings. Each group member participated in all phases of the project, which include research, documentation, class diagrams and coding, smarten the development process. This approach was applied in order to make the project understandable in all spheres for each member, resulting in better results and enhance personal knowledge.

Our first idea was to implement the factory, prototype and singleton patterns. At time the trade among companies was a great challenge. At first we could not figure out how we would make the trades happen, it was truly challenging to develop the logic in our trade algorithm.

The project took a different path once we started coding and some changes were made from the previous submission:

* Prototype pattern was removed from the project as it was difficult to be implemented due to our lack of knowledge;
* Facade pattern and factory pattern implementation was slightly changed after a better understanding of the patterns;
* The first idea was to use singleton pattern in companies and then clone them with prototype pattern, after feedbacks we realized that the use of both is not feasible, therefore we kept the use of singleton pattern for the menu class;
* We changed the data Structure from Array to ArrayList because is a better alternative of traditional java arrays it has a set of methods to access elements and modify them, it is dynamically sized.

One of the things that made the final results achievable was the planning phase, it was crucial the use class diagram and functional analysis which was not encouraged on previous projects.

Unfortunately it was not possible to delivery all the features required, which can be seen in the output method.