Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Programming Techniques

Homework Assignment 4

Hashing techniques in Java

Student: Eric-Andrei Dumea

Group: 30423

Supervising Teacher: Cristina Bianca Pop PhD.

# Table of contents

1 . Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 . Problem analysis, modelling, scenarios, use-cases . . . . . . . . .3

2.1 Problem analysis, modelling . . . . . . . . . . . . . . . . . . . . . 3

2.2 Scenarios, use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3 . Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1 Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 Classes Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3 Packages & Interfaces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.4 User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

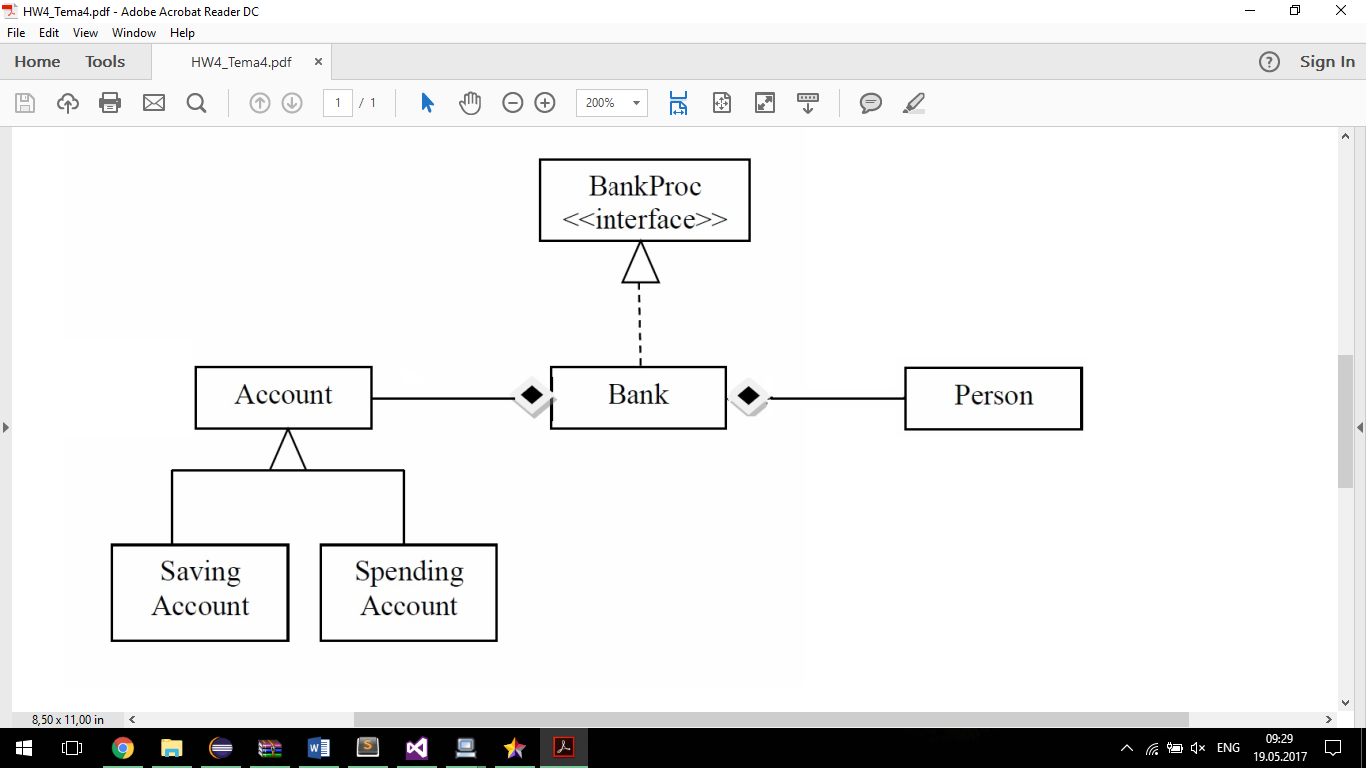
4. Implementation & testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

6. Conclusions, further development, self-taught knowledge . . . . . . . . . . 10

7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

# Problem specification

Considering the system of classes in the class diagram specified,

1. Define the interface BankProc (add/remove persons, add/remove holder associated accounts, read/write accounts data, report generators, etc). Specify the pre and post conditions for the interface methods. 2. Design and implement the classes Person, Account, SavingAccount and SpendingAccount. Other classes may be added as needed (give reasons for the new added classes). 3. An Observer DP will be defined and implemented. It will notify the account main holder about any account related operation. 4. Implement the class Bank using a predefined collection which uses a hashtable. The hashtable key will be generated based on the account main holder (ro. titularul contului). A person may act as main holder for many accounts. Use JTable to display Bank related information. 4.1 Define a method of type “well formed” for the class Bank. 4.2 Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions). 5. Design and implement a test driver for the system. 6. The account data for populating the Bank object will be loaded/saved from/to a file.

## Short description

Design patterns and advanced programming techniques are to be used in this project, along with Java Collection Framework (JCF) classes, such like HashMaps and ArrayLists

# Problem analysis, modelling, scenarios, use cases

## Problem analysis, modelling

For this application I used the paradigms of Object Oriented Programming in Java. Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. The main concepts of Object Oriented Programming are Encapsulation, Abstraction, Inheritance and Polymorphism.

*Inheritance* provides a powerful and natural mechanism for organizing and structuring your software. Classes inherit state and behavior from their superclasses, and you can derive one class from another using the simple syntax provided by the Java programming language.

*An interface* is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.

*A class* is a blueprint or prototype from which objects are created.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

In computing, a hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash collisions where the hash function generates the same index for more than one key. Such collisions must be accommodated in some way.Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Hashtables process workloads to create and update themselves, querying the data they contain and running applications against it.

Hashtables are often needed where big amounts of data are involved, and where data needs to be stored on the long-term, in secure and even encrypted sites and environments.

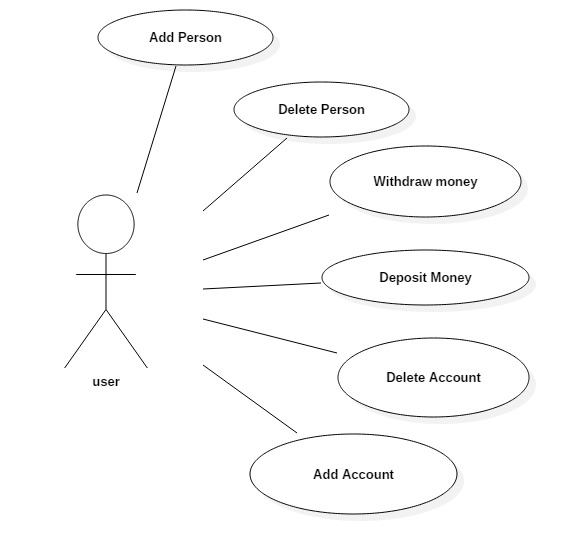
An application like this can be used by small or even medium retail companies, that need to keep track of their Persons, Accounts and orders. For larger companies, probably a large-scale solution should be more optimal, something like a dedicated framework, among with several secure servers.

Based on the above, the main concept of the assignment was the HashTable, along with the concepts involved, Persons, Accounts. Also, I tried using the layer – based implementation.

This application should simulate a bank, with a hashtable of Accounts, Persons and order details.

## Scenarios, use cases

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.



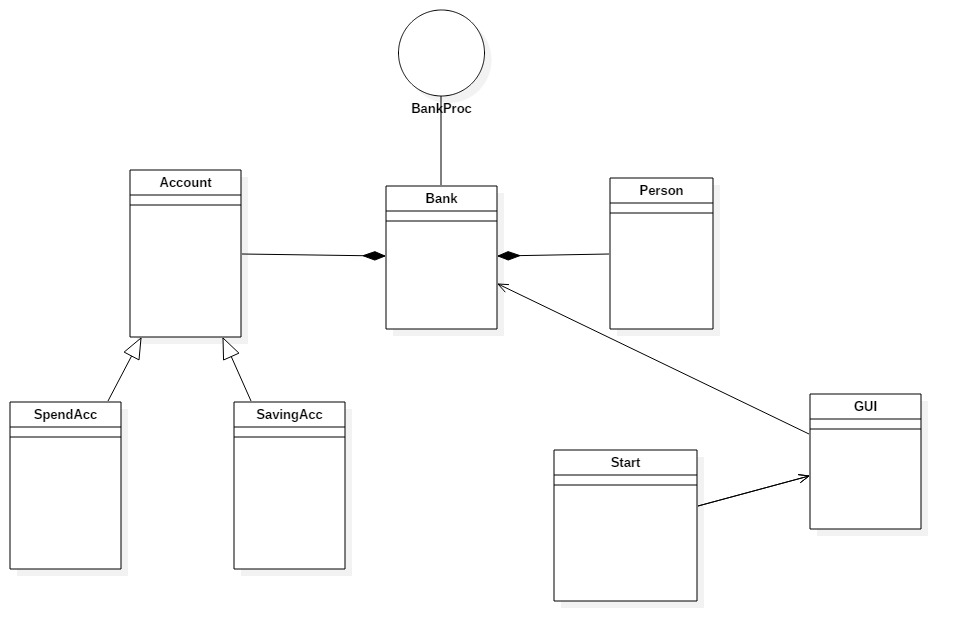
The program will have a basic input interface with buttons and some textboxes. Also, to visualize the data in the hashTable – based data base, some tables are provided, that are refreshed in real time. In the window related to the orders, Persons and Accounts are selected using comboboxes, just in order to be simpler.

# 3.Design

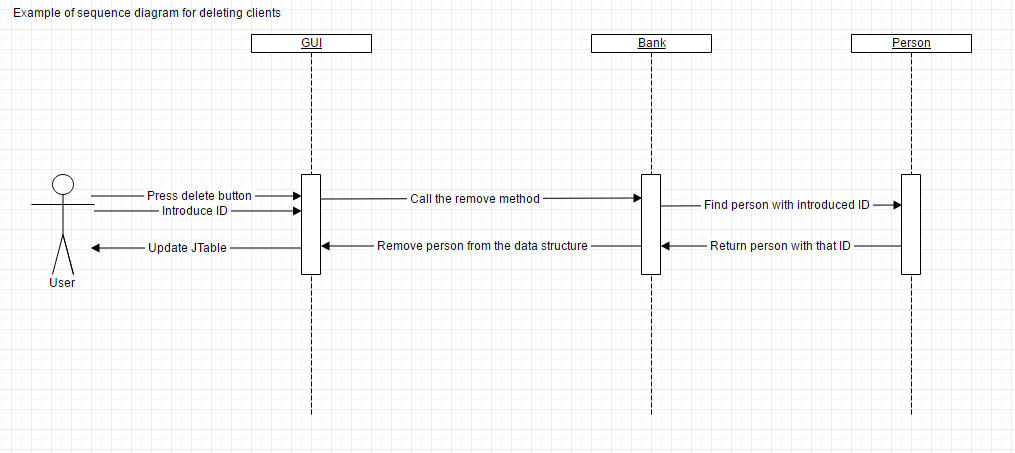
## 3.1 UML Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code

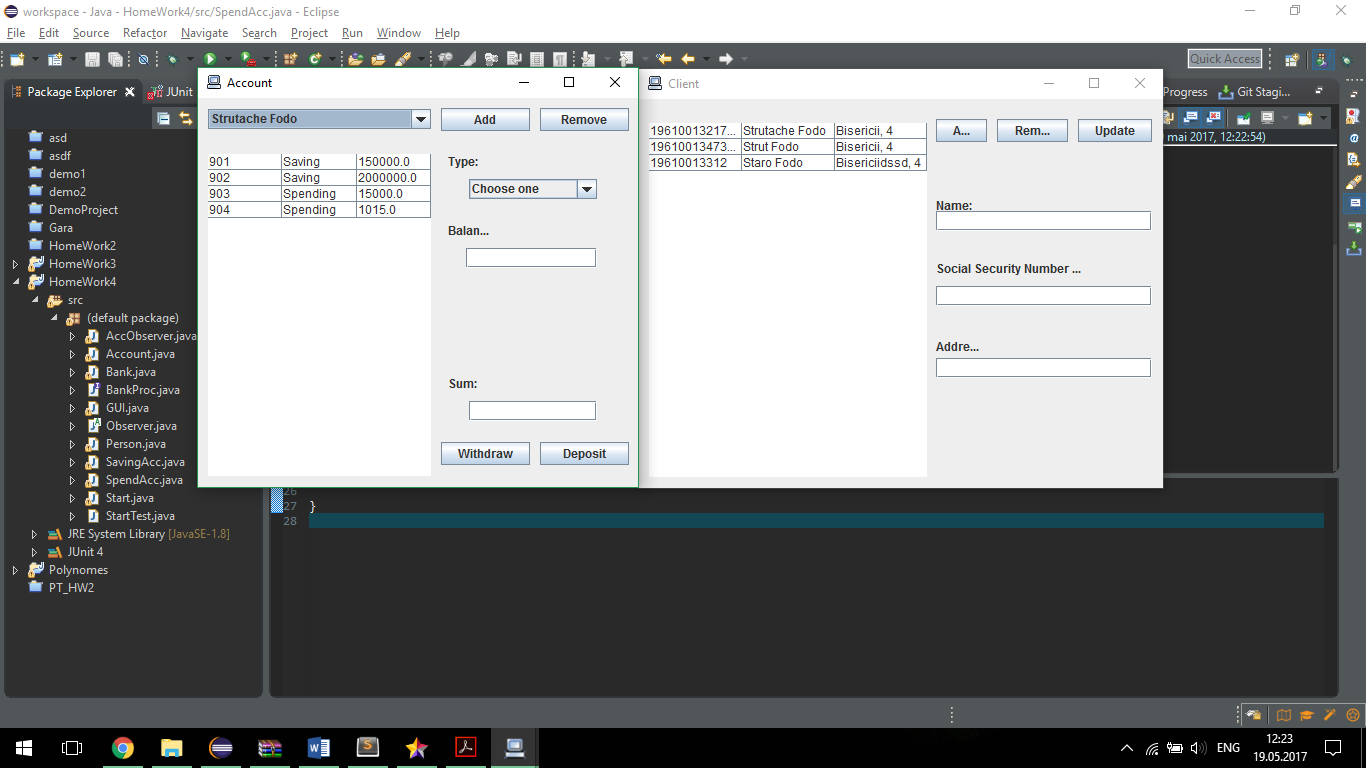


## 3.2 UML Sequence Diagram

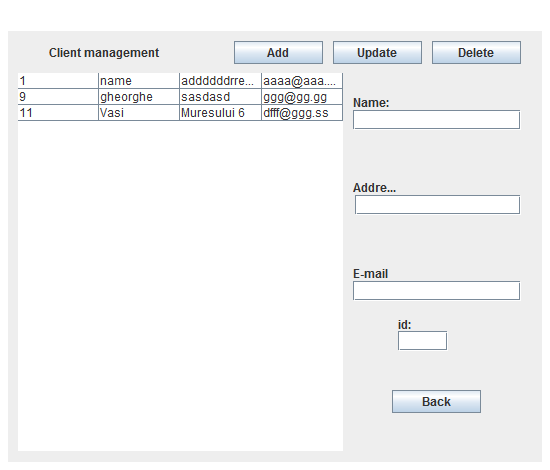


## 3.2 Classes design

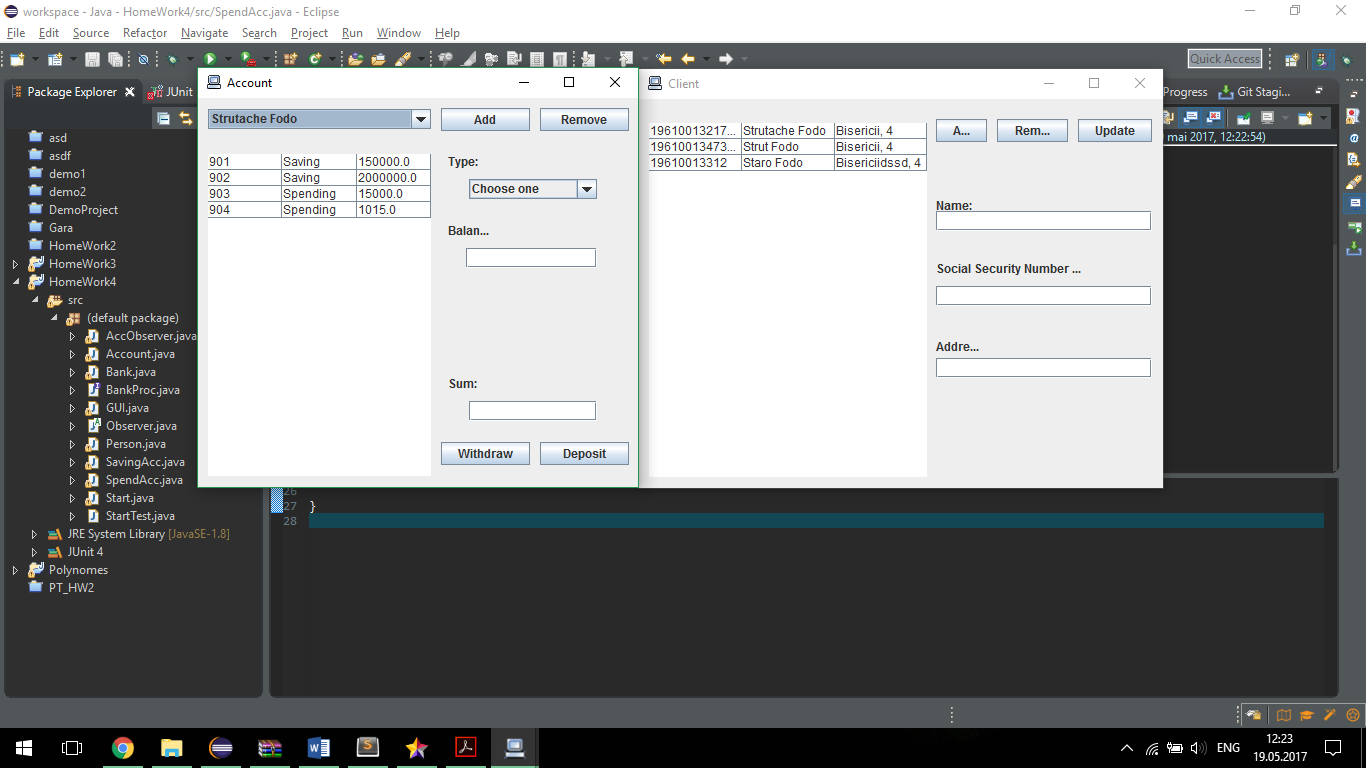
1.GUI Class

The “GUI” class implements the interaction with the user, also known as GUI or Graphical Interface Unit. The windows that pop up at runtime:

This is mainly the main menu of the application, where the user chooses the operation set he wants to access : the Person operations , Account The Person window:



The Account window:



This class contains many fields related to every component of the graphical unit interface, and methods to update the tables and the comboboxes.

The Start class contains the main method, *public static void main(String[] args),* which creates a new instance of the GUI class, thus calling the constructor, making the GUI – the main menu part of it, visible. The constructor builds up the main frame part, using the javax.swing framework. Here, in function of the button the user presses, a frame is opened – c for Person, o for order or p for Account. The GUI was built on a trial and error process, because I wanted to achieve a good positioning of the components.

The GUI class implements the *ActionListener* interface, thus it contains the *public void actionPerformed(ActionEvent e)* method. This method specifies the behaviour of the application at different stimuli (e.g. – Start button has been pressed).

Also, the Tester, as the name states, contains developer’s information, in the form of the *testing(. . .)* method. It contains things I needed to test the functionality of the app, before I implemented the graphical unit interface. (now commented)

2. Person class

This class is one of the classes that make the nucleus of the app. It contains fields related to name, address, email, and id. We have 2 constructors, one with and one without the id, namely **public** **Person**(**int** id, **String** name, **String** address, **String** email) and **public** **Person**(**String** name, **String** address, **String** email) , one without the id, the hashtable being implemented such, the id autoincrements. It has getters and setters for every field.

1. Account class

This is another essential class for the project, being the model of the Accounts. As the Person class, it contains two constructors : **public** **Account**(**int** id, **String** name, **int** amount, **double** price) and **public** **Account**( **String** name, **int** amount, **double** price) , again because of the design of the d base.

This class, also has got setters and getters.

1. Order class

This is another essential class for the project, being the model of the Accounts. As the Person class, it contains two constructors : **public** **Order**(**int** id, **String** cname, **String pname**, **int** amount) and **public** **Order**( **String** cname, **String pname**, **int** amount , again because of the design of the data base.

1. PersonDAO class

This class deals with the connection between the user interface and the hashtable itself, containing the SQL code and methods related to them. So, it contains the methods to add, find by id and name, delete and update Persons.

1. AccountDAO

This class deals with the connection between the user interface and the hashtable itself, containing the SQL code and methods related to them. So, it contains the methods to add, find by id and name, delete and update Accounts.

1. OrderDAO

This class deals with the connection between the user interface and the hashtable itself, containing the SQL code and methods related to them. So, it contains the methods to add, find by id and name, delete and update orders.

1. PersonBLL

This class behaves like an interface between the Data Acquisition Layer and the presentation layer. It contains methods similar as the DAO equivalent but without the SQL stuff and the data acquisition part. It belongs to the Business Layer. It also makes use of the validators.

1. AccountBLL

This class behaves like an interface between the Data Acquisition Layer and the presentation layer. It contains methods similar as the DAO equivalent but without the SQL stuff and the data acquisition part. It belongs to the Business Layer. It also makes use of the validators.

1. OrderBLL

This class behaves like an interface between the Data Acquisition Layer and the presentation layer. It contains methods similar as the DAO equivalent but without the SQL stuff and the data acquisition part. It belongs to the Business Layer. It also makes use of the validators.

1. Validators : EmailValid, StockValid, OrderValid

These classes implement the Validator interface, which checks for bad things in the input.

## 3.3 Packages and interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be bankd in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

I had to implement a Layered Architecture, such I had to use some packages

In this application, I used:

* javax.swing.\* , java.awt.\* and java.util.\* packages mainly for the GUI part
* bll – business layer package
* dao – data acquisition layer.
* Presentation - presentation layer
* Model – model layer, Person, Order, Account

Interfaces used: ActionListener implemented by the Graphical Unit Interface class ( GUI ), Validator.

## 3.4 User Interface

As I’ve previously mentioned, the user interface is pretty much basic and easily-understood, even by nonfamiliarized people. When running the application, a window will open and it will provide to the user the possibility of giving inputs to choose the operations suitable to their needs. This window is constructed in the GUI class using some predefined classes and instructions. This frame opens another frames, specific to the operations vised.

## 3.5 Algorithms

Choosing from all the methods implemented in this program, I consider the update methods from class GUI the most complex ones, as it control every move made by the application. It acts like a control unit for everything, doing most of the computations required.

***public******void******PersonsShow****(){*

*Connection* ***con*** *= (Connection)* ***ConnectionFactory****.getConnection();*

***try****{*

*List<Object>* ***obj*** *=* ***new*** *ArrayList<>();*

***Object******collumns****[] = {"id", "name", "address", "email"};*

***PreparedStatement******preparedStatement*** *= (****PreparedStatement****) con.prepareStatement("SELECT \* FROM Person");*

*ResultSet* ***rs*** *= preparedStatement.executeQuery();*

***while*** *(rs.next()){*

***int******cid*** *= rs.getInt("id");*

***String******name*** *= rs.getString("name");*

***String******address*** *= rs.getString("address");*

***String******email*** *= rs.getString("email");*

***System****.****out****.println(name+address+email);*

***Person******Person*** *=* ***new*** *Person (cid, name, address, email);*

*obj.add(Person);*

*}*

*ctable = createTable (obj, collumns);*

*ctable.setBounds(10, 42, 325, 378);*

*p1.add(ctable);*

*}*

***catch*** *(****SQLException******e****){*

***System****.****out****.println("nope, not today.");*

*}*

*}*

***public******void******AccountShow****(){*

*Connection* ***con*** *= (Connection)* ***ConnectionFactory****.getConnection();*

***try****{*

*List<Object>* ***obj*** *=* ***new*** *ArrayList<>();*

***Object******collumns****[] = {"id", "name", "address", "email"};*

***PreparedStatement******preparedStatement*** *= (****PreparedStatement****) con.prepareStatement("SELECT \* FROM Account");*

*ResultSet* ***rs*** *= preparedStatement.executeQuery();*

***while*** *(rs.next()){*

***int******cid*** *= rs.getInt("id");*

***String******name*** *= rs.getString("name");*

***double******price*** *=****Double****.parseDouble( rs.getString("price"));*

***int******amount*** *=* ***Integer****.parseInt( rs.getString("amount"));*

***System****.****out****.println(name+price+amount);*

***Account******prod*** *=* ***new*** *Account(cid, name, amount, price);*

*obj.add(prod);*

*}*

*ptable = createTable (obj, collumns);*

*//mc=(DefaultTableModel) ctable.getModel();*

*ptable.setBounds(24, 60, 360, 330);*

*pcontent.add(ptable);*

*}*

***catch*** *(****SQLException******e****){*

***System****.****out****.println("O dat fail 1!");*

}

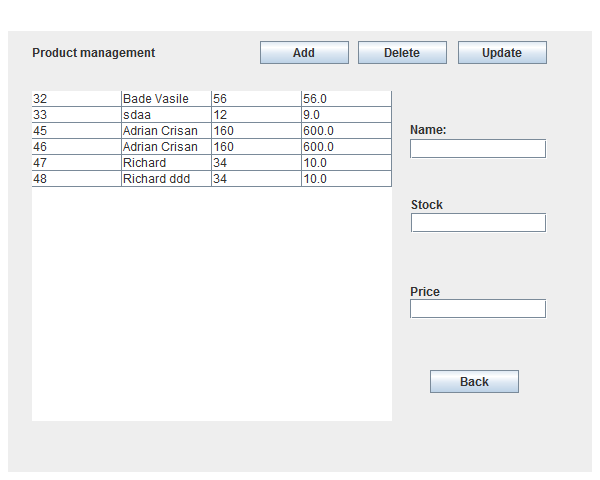
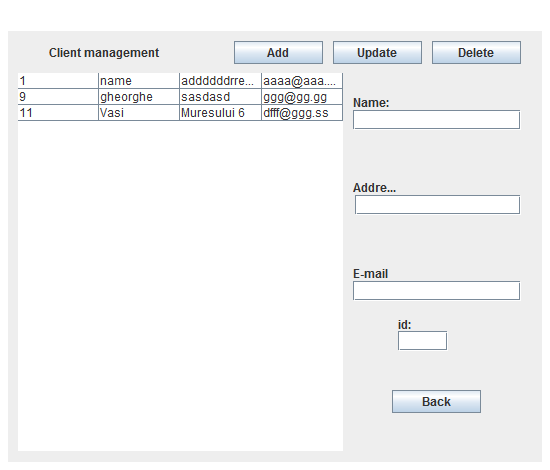
}

# 4. Implementation and testing

## Implementation

The implementation process began with the class diagram and some sketches of how the program should work. After considering the values needed as an input and the requirements of the program, as well as the final output, the actual implementation of the classes followed.

# 5. Results

The window of the Account or Person looks like this after run:

# 6. Conclusions, further development, gained knowledge

In the end, being able to implement something so dynamically and simulating such an often encountered situation of everyday life while combining everything with the concepts and rules of Object-oriented programming was a challenge with a self-rewarding result. As a further development, the first thing that should be a better exception system, especially at the input.

The problem specification presented itself as an interesting subject with many possibilities. Considering that it was the first program that used threads, the process of making was mostly based on a trial-error method and a lot of additional research which proved to be successful in the end.

Updating the graphical user interface in real time was also a challenge that needed extra attention and studying. Working with hashtables turned out to impose a lot of additional concepts and ways of implementation on top of the basic Object-oriented paradigms.

There is still a lot of space for improvements in order to make the program more efficient. Some of the algorithms could be optimized and maybe implemented in a different way. Some constrains can be imposed on the input, meaning that more exceptions can be handled with and there can be created more statistics for the simulation. Moreover, the animation can be done using some figures and more advanced graphic. The graphical user interface can be improved. There could be added some colors or given a different layout of the components in the frame.

This project/assignment taught me, as a first thing, to do a proper GUI using code only, to use Junit and to do a proper documentation (as in following a structure and using English extensively). Also, the main benefit of this project was the refresh on the Java coding skills.

# 7. Bibliography

* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW2_Tema2/Tema2_HW2_Barem.pdf>
* <https://stackoverflow.com> – for some quick answers to basic questions
* <http://users.utcluj.ro/~jim/OOPE> and the moodle Lab session guides of Mr. Joldos.
* <https://docs.oracle.com/javase/tutorial/java/concepts/>
* <https://en.wikipedia.org/wiki/Class_diagram>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW3_Tema3/Tema3_HW3.pdf>
* <http://courses.caveofprogramming.com/>
* <http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>
* <http://www.uml-diagrams.org/use-case-diagrams.html>
* [http://www.javaworld.com/article/2077157/data-storage/integrating-hashtables-with-java-via-jdbc.html](http://www.javaworld.com/article/2077157/data-storage/integrating-databases-with-java-via-jdbc.html)
* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW3_Tema3/HW3_Indications.pdf>
* <http://www.tutorialspoint.com/java/java_serialization.htm>
* <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
* <http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
* <http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>
* <http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise>