Bank

Programming Techniques – Fourth Project

Group: 30424

First\_Name: Daiana

Second\_Name: Spatacean

Teacher: Claudia Pop

Summary:

1**. Introduction**

1.1 Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2. **Description of the project**

2. 1 Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2. 2 Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3. **Projection**

3. 1 UML diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . .

3. 1. 1 Use-case diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . .

3. 1.2 Class diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . .

4**. Implementation** . . . . . . . .. . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . .

5. **Implementation** **and testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

6. **Further developments .** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

7. **Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

8. **References**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. **Intoduction**

A bank is a [financial institution](https://en.wikipedia.org/wiki/Financial_institution) that accepts [deposits](https://en.wikipedia.org/wiki/Deposit_account) from the public and creates credit. Lending activities can be performed either directly or indirectly through [capital markets](https://en.wikipedia.org/wiki/Capital_market). Due to their importance in the [financial system](https://en.wikipedia.org/wiki/Financial_system) and influence on national [economies](https://en.wikipedia.org/wiki/Economy), banks are [highly regulated](https://en.wikipedia.org/wiki/Bank_regulation) in most countries. Most nations have institutionalized a system known as [fractional reserve banking](https://en.wikipedia.org/wiki/Fractional_reserve_banking) under which banks hold liquid assets equal to only a portion of their current liabilities. In addition to other regulations intended to ensure liquidity, banks are generally subject to [minimum capital requirements](https://en.wikipedia.org/wiki/Minimum_capital_requirement) based on an international set of capital standards, known as the [Basel Accords](https://en.wikipedia.org/wiki/Basel_Accords).

Banking began with the first prototype banks of [merchants](https://en.wikipedia.org/wiki/Merchant) of the ancient world, which made [grain loans](https://en.wikipedia.org/wiki/Loan) to farmers and traders who carried goods between cities. This began around 2000 BC in [Assyria](https://en.wikipedia.org/wiki/Assyria) and [Babylonia](https://en.wikipedia.org/wiki/Babylonia). Later, in [ancient Greece](https://en.wikipedia.org/wiki/Ancient_Greece) and during the [Roman Empire](https://en.wikipedia.org/wiki/Roman_Empire), lenders based in temples made loans and added two important innovations: they accepted [deposits](https://en.wikipedia.org/wiki/Deposit_account) and [changed money](https://en.wikipedia.org/wiki/Bureau_de_change). Archaeology from this period in [ancient China](https://en.wikipedia.org/wiki/History_of_China#Ancient_China) and [India](https://en.wikipedia.org/wiki/History_of_India) also shows evidence of [money lending](https://en.wikipedia.org/wiki/Loan) activity.

**2.Description of the project**

The purpose of this assignment was to make some operations on the bank and the accounts that are in. The operations that can be performed are add account and person, delete account or person with all the accounts that he/ she has. And other operations like deposite money or withdraw money.

**3.Diagrams**

**3.1 Use-case diagram**

A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different [use cases](https://en.wikipedia.org/wiki/Use_case) in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

The user can be any person that wants to use the application and the interface must be pretty friendly.

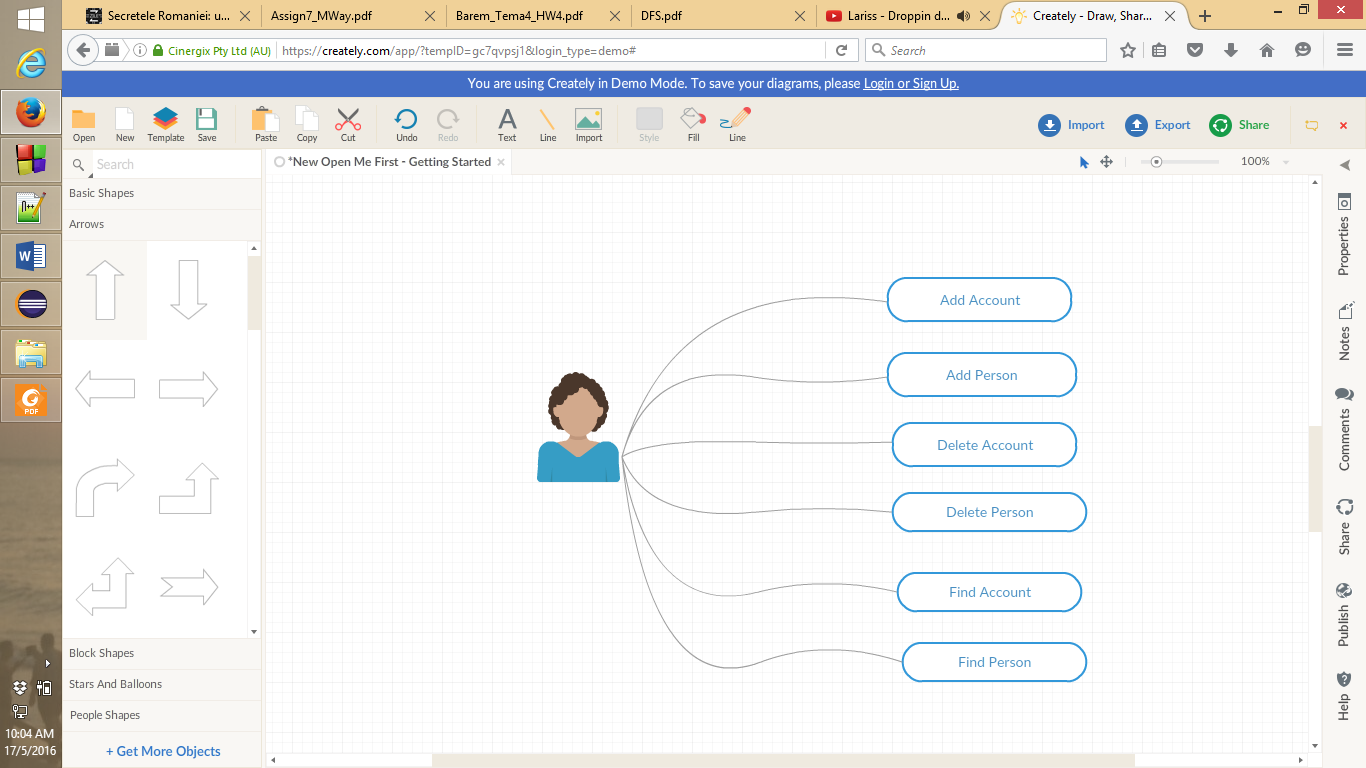


Fig. User- diagram

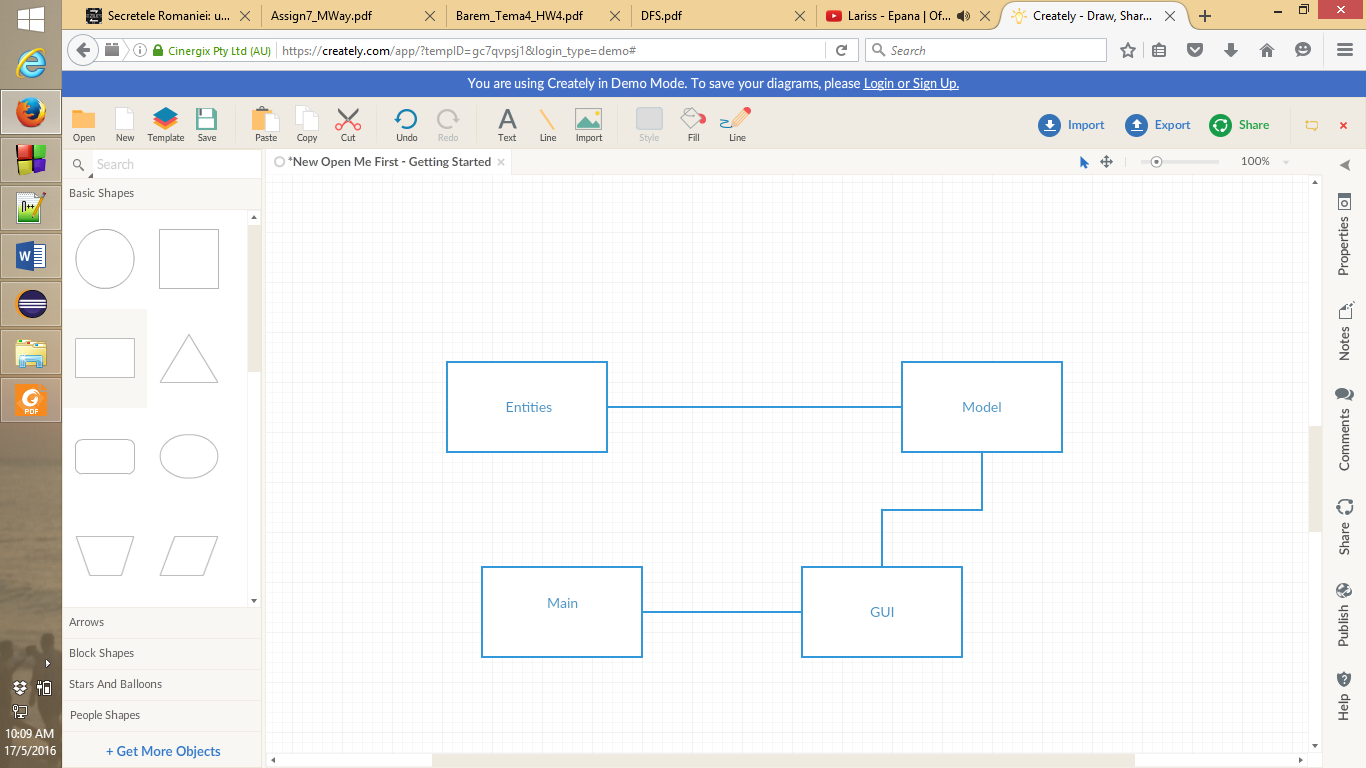
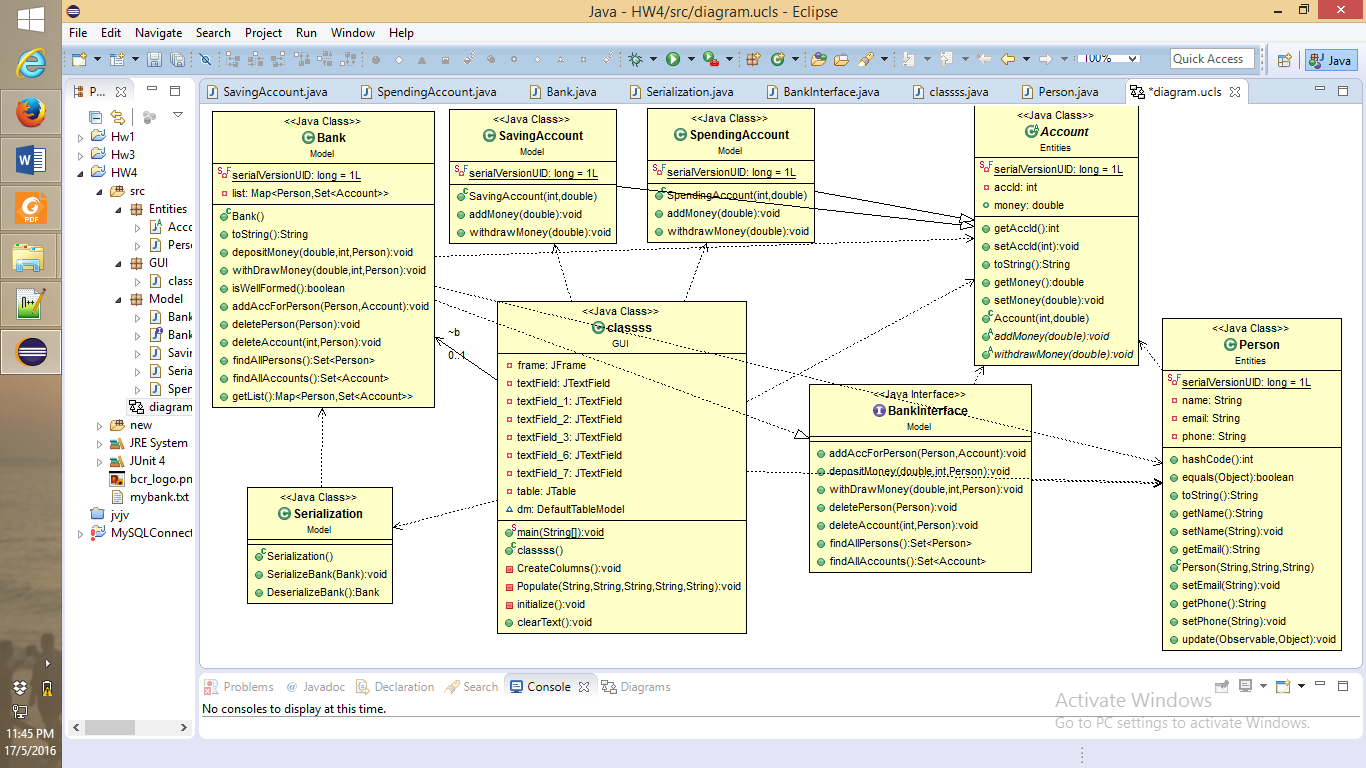


Fig. Package- diagram

**3.2 Class Diagram**



**4.Implementation**

This project is made following the OOP structure (classes, methods) and principles:

- **inheritance** when one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism

- **polymorphism** when one task is performed by different ways i.e. known as polymorphism. For example: to convert the customer differently, to draw something e.g. shape or rectangle etc.

- **abstraction** hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing

- **encapsulation** binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

Starting with the database:

* The table Customer contains the id of the customer which is a unique key, the name, the email address and the phone number
* The second table is called Product which also contains the id, name and quantity
* And the last one is about the Order

1. **Model entities**

* Class **PERSON**
* Contains some private variables which are unique for each person
* Here are some extra methods like: update(for the observer), hash code, equals, the last 2 are overridden and getters and setters

**public** **class** person implements Serializable, Observer {

**private** String name ;

**private** String email ;

**private** String phone ;

* Class **ACCOUNT**
* Contains some private variables which are unique for each account
* And two abstract methods void addModey and withdrawMoney

**public** **class** Acount extends Observable implements Serializable{

**private** **int** accId ;

**private** double money;

**2.GUI package**

* In this one is implemented the interface for the project
* Also here is a main window for the GUI
* Here I have buttons for the operations that can be done in my application like delete, add and so on
* I have a table JTable in which the informations from my serialized file appear
* And some text fields from which I get the information from the client that uses my application
* And also a logo for my bank

**3.Model package**

* Class **BANK**

1. **public** **class** Bank implements BankInterface, Serializable{

list = new HaspMap <Person, Set<Account>>()

}

1. **public** **void**  DepositeMoney()

{

* here is the class when the money are introduced in the bank

**}**

1. **public** **void**  withDrawMoney()

{

* when the money are used by the person that has the account to buy something

**}**

1. **public** **boolean** isWellFormed(){

**}**

1. **public** **void**  addAccForPerson()

{

* add a new account for a person

**}**

1. **public** **void**  deletePerson()

{

* delete the person and all the account that it has

**}**

1. **public** **void**  deleteAccount()

{

* delete specific account

**}**

1. **public** **Set<**Person> find AllPersons()

{

* find all the persons in the bank

**}**

1. **public** **Set<**Accounts> find AllAccounts()

{

* find all the accounts of a person

**}**

* Class **BANKINTERFACE**
* Same classes as the bank class because this class is an interface that is implemented by the bank class
* Postconditions and preconditions for each method

**Public**  **interface** BankInterface {

* Class **SAVINGACCOUNT**

**public** **class** SavingAccount implements Serializable extends Account{

}

* here are 2 methods add money and withdraw money which notifies through the observer that the money were added in a saving account or withdrawn from the account
* and here when I add money I have a commission and I will receive a message when the sum that I want to withdraw is grater then 300

**public** **void**  addMoney()

**public** **void**  withdrawMoney()

* Class **SPENDINGACCOUNT**

**public** **class** SavingAccount implements Serializable extends Account{

}

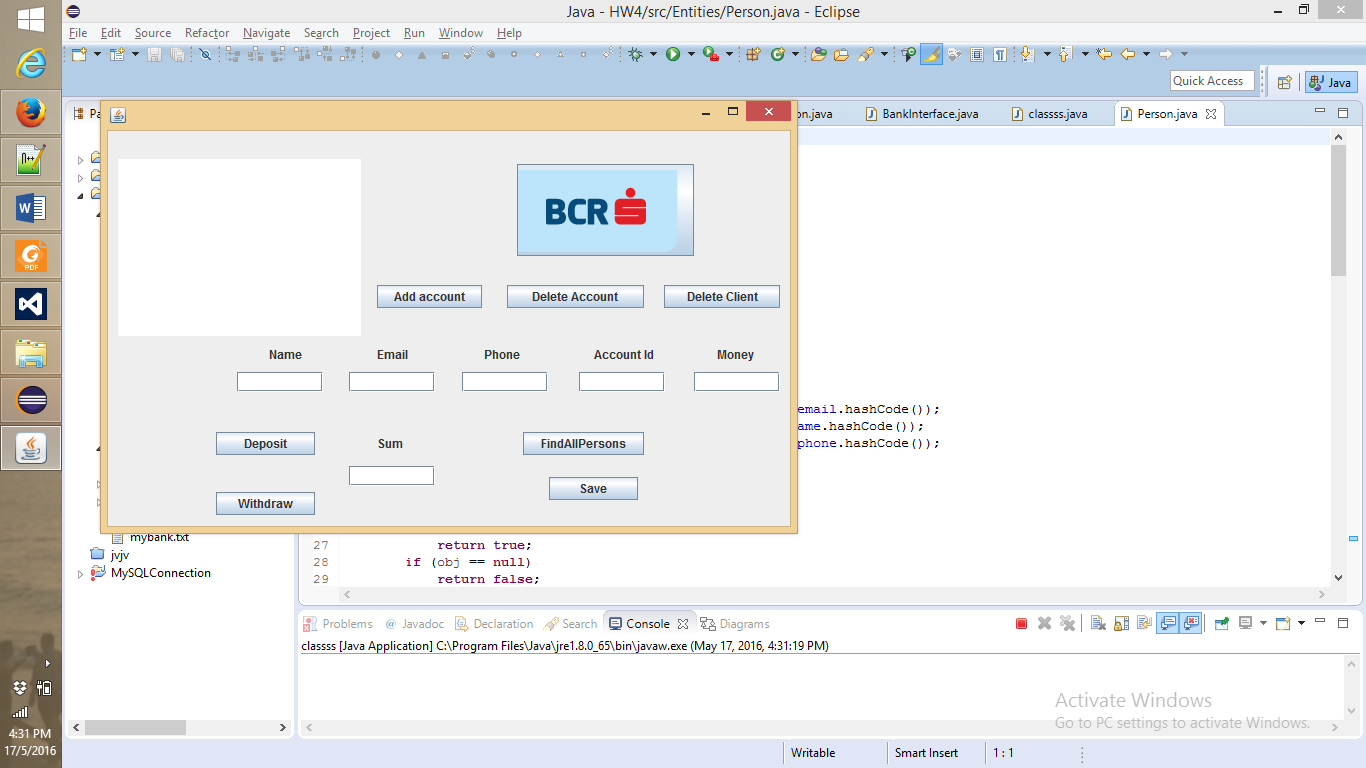
* Same as the saving account the same methods only the message tht will be printed is changed
* Class **SERIALIZATION**

**public** **void**  SerializeBank()

**public** **void** DeserializeBank ()

5. **Implementation** **and testing**

This is how the interface looks like:



* The **user** should:
* Enter the name, email , phone , accountId , money and add it to the bank and for delete only select the row and press delete
* Enter money and deposite it or withdraw it
* For the serialization the user should press the button Save and the dates will be keept in a binary file

Regarding the implementation process, I used as program Eclipse IDE. During the implementation of the project I made a lot of changes. First my program was into one package and then I split it. The classes had many methods. I also start the project not based on OOP principles. Before the interface was made I print the results in the console to see if they are correct and to know if I continue in that manner or I had to make some changes.

5. **Results**

I am pretty satisfied with my work and I think that the project is very easy to understand and the most important thing is that any type of user is able to use the interface and will have no problem with it. And in the end I think that I implemented all the petitions of the “client”.

6. **Further developments**

I think that I can make a very long list with the things that I want to improve related to my project with the bank :

* Starting from the “back” of the project: I like to make more operations with the money like make some credits
* Also the window can be modified because it doesn’t look so nice
* I also want to make reflection on the tables
* More OOP with more classes and method with less lines of code and not so many duplicate code in some
* I want to have a more structured code
* And for the interface I want to make it much pretty
* For the buttons if I will make them with colors and with another type of writing and for the toolbar I want to implement help and menu and so on
* It will be nice if the size of the window can be modified from the user

7. **Conclusions**

To conclude, I can say that this project meant hard work, a lot of new things learned, focusing, development and creativity. Even if I encountered a lot of problems, I was able to fix them after all, by searching on the internet or asking a colleague for advice. I think that my application satisfies the requirements and the users will have at their disposal all its functionalities. And in the end I found out that is very interesting to work with database in java because you make the connection and every thing that you modify in the database in any table it can be seen from the java program that you use.

8. **References**

* <https://en.wikipedia.org>
* <http://stackoverflow.com>
* <http://www.oracle.com/technetwork/articles/java/index-137868.html>
* <http://www.tutorialspoint.com/java/java_data_structures.htm>

And others!

.