Bookstore employees applicationn

Analysis and Design Document

Student: Alexe Simona Elena

**Group: 1st Year, CSC**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

1. **Objective**

The objective of this assignment is to allow students to become familiar with the Model View Controller architectural pattern and the Factory Method design pattern.

2.**Application Description**

Use Java API to design and implement an application for the employees of a bookstore. The application should have two types of users (a regular user represented by the book store employee and an administrator user) which have to provide a username and a password in order to use the application.

The regular user can perform the following operations:

- Search books by genre, title, author.

- Sell books.

The administrator can perform the following operations:

- CRUD on books (book information: title, author, genre, quantity, and price).

- CRUD on regular users’ information (clientId, name, address, birthday, preffered genres)

- Generate two types of reports files, one in pdf format and one in csv format, with the books out of stock.

1. **Application Constraints**

The data will be stored in a database. Use the Layers architectural pattern to organize your

application. Use a domain logic pattern (transaction script or domain model) / a data source hybrid

pattern (table module, active record) and a data source pure pattern (table data gateway, row data

gateway, data mapper) most suitable for the application.

1. **Requirements**

- Create the analysis and design document.

- Implement and test the application.

1. **Deliverables**

* Analysis and design document.
* Implementation source files.
* Readme file that describes the installation process of the application and how to use it:
* how to install your application on a clean computer
* how to access your application and with what users
* images with all use cases and their scenarios implemented

# Functional Requirements

The regular user can perform the following operations:

- View all accounts (account information: identification number, type, amount of money,

date of creation).

- Transfer money between accounts.

- Add another account for the same client ID ( especially for another currency)

The administrator user can perform the following operations:

- CRUD on clients information (name, identity card number, personal numerical code,

address, etc.).

- Create/update/delete/view clients' accounts.

# Non-functional Requirements

This application has a fast time response, so the users can see the informations they need. They can also perform the operations needed quickly.

The design is very simple, made in a way that anyone can use the application without specific training before using. Next to all of the fields, which have to be completed, is a description(where needed) so that the users won’t have problems regarding the content they need to provide. The information is also displayed in a simple and logical way, making the application very simple to use.

2. Use-Case Model

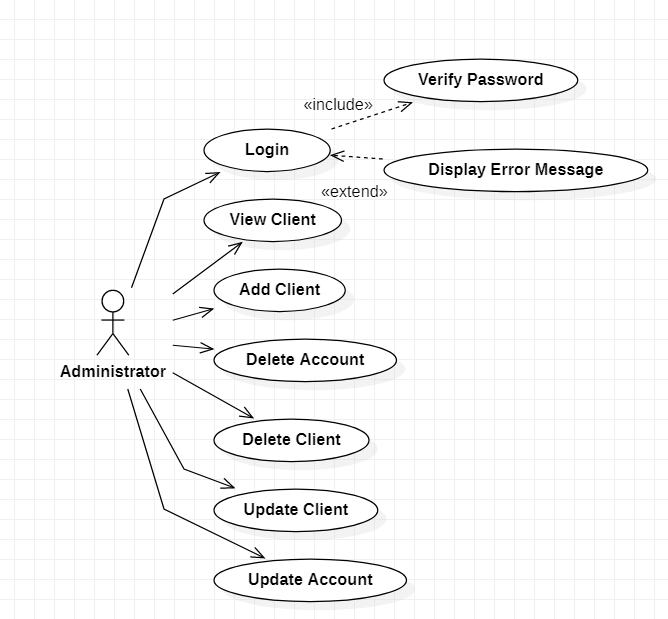
*Use case:* The goal is to represent the functionalities of a mini banking application

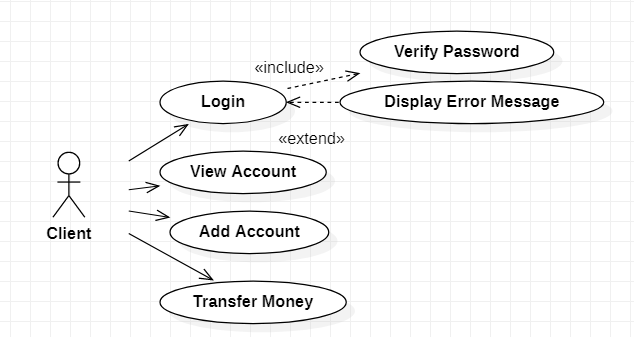
*Level:* User-goal level

*Primary actor:* The client

*Main success scenario:* The client has the following functionalities: login, view his account informations, transfer money and add another account for another currency

*Extensions:* If the client provides a valid username and password, he can successfully gain the functionalities described before, if not, an error message will appear.





3. System Architectural Design

3.1 Architectural Pattern Description

The present application is using an architectural pattern called “The n-tier architecture”. This is also called “A multi-layer architecture” because the software is engineered to have the processing, data management, and presentation functions physically and logically separated (1. The n-tier architecture). It is suitable to support enterprise level client-server applications (like this one) by providing solutions to scalability, security, fault tolerance, reusability, and maintainability. Also, it helps developers to create flexible and reusable applications (2. The n-tier architecture).

The most widespread use of "multi-tier architecture" refers to three-tier architecture, and in this application they are called “Data Soruce Logic Layer”, “Domain Logic Layer” and “Presentation Layer (3. The n-tier architecture).

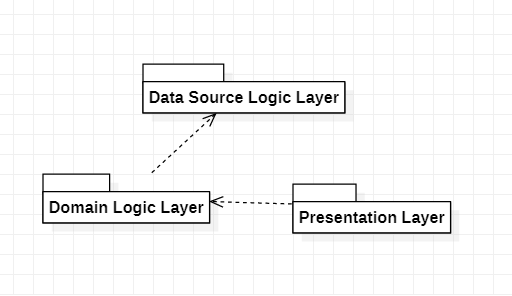
The first one, “Data Source Layer” communicates with the database (created using MySQL Workbench). “Domain Logic Layer” coordinates the application, processes commands, makes logical decisions and evaluations and performs calculations. “Presentation Layer” represents the user interface, the main function of this translating tasks and results in a way that the user can understand.

In this application, “Data Source Logic Layer” contains 4 classes: DataBaseConnection, LoginMapper, AccountMapper and ClientMapper.

The “Domain Logic Layer” contains 6 classes: Login, Account, Client, LoginLogic, AccountLogic and ClientLogic.

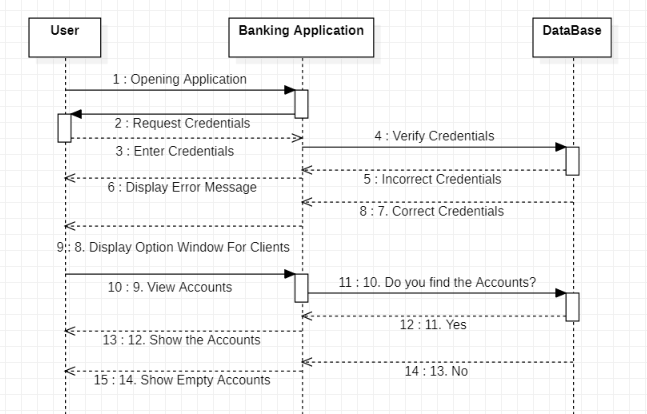
The “Presentation Layer” contains 10 JFrames: AccountViewAdmin, AccountView, AdministratorGUI, ClientGUI, InsertAccount, InsertClient, MoneyTransfer, UpdateAccount,UpdateClient and UserLogin. Each of these frames use methods from 3 classes of the Domain Logic Layer (ClientLogic, AccountLogic and LoginLogic). The Domain Logic Layer classes use information from the 3 Mappers, which are stored in the Data Source Logic Layer.

3.2 Diagrams

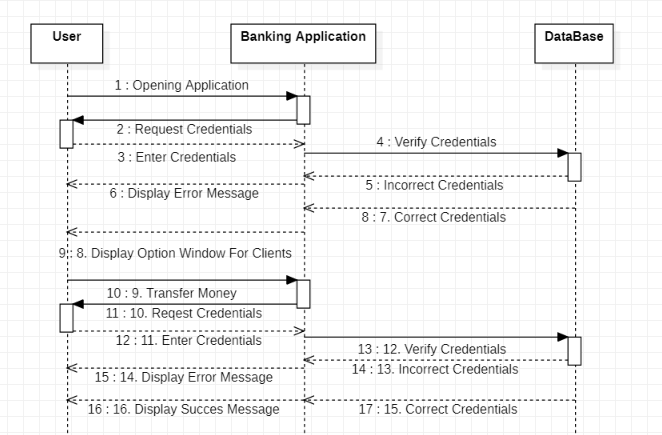


4. UML Sequence Diagrams

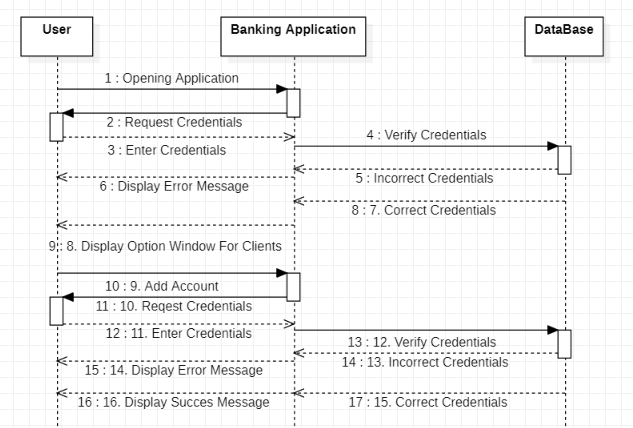
1. Login



1. Transfer Money



1. Add Account



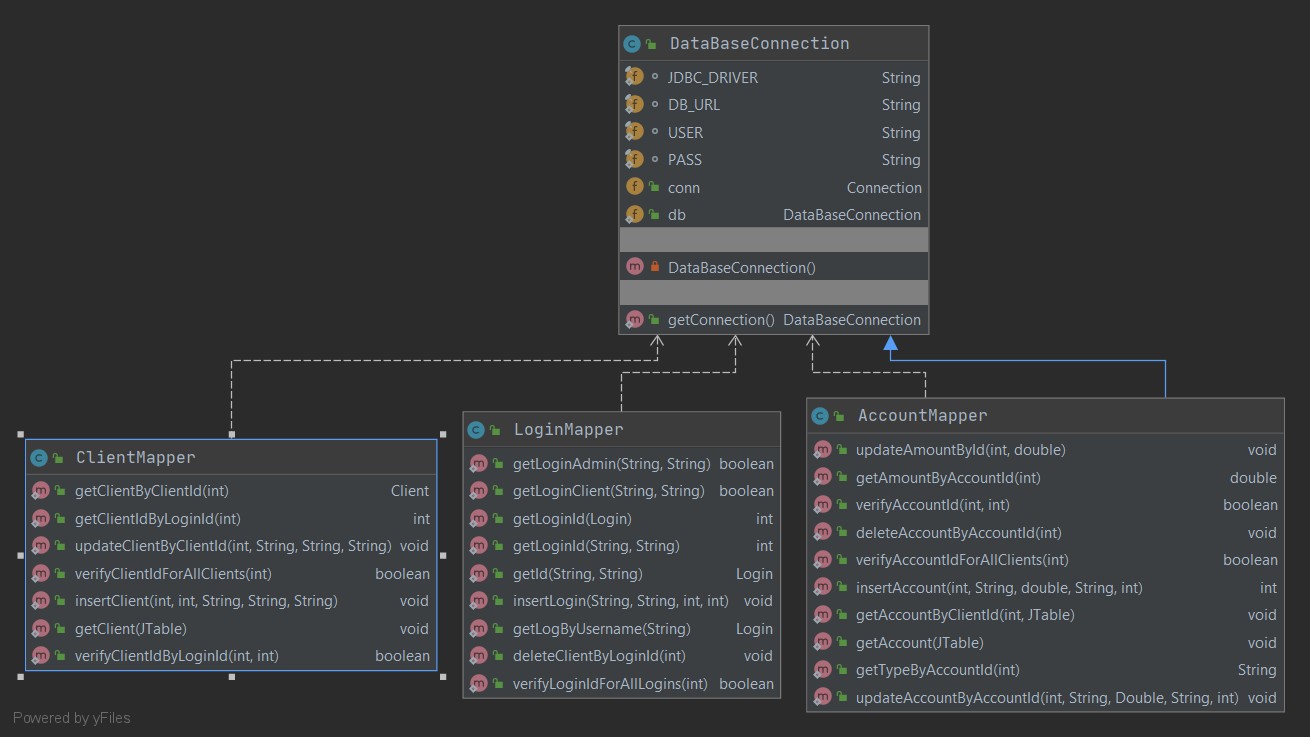
5. Class Design

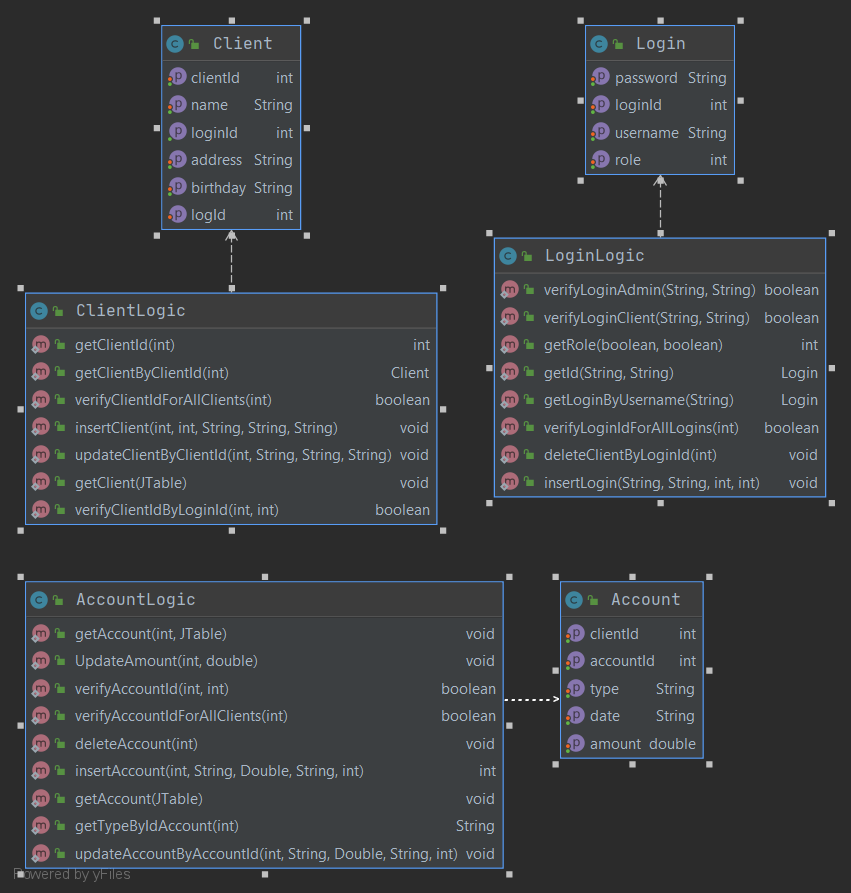
* 1. **Design Patterns Description**

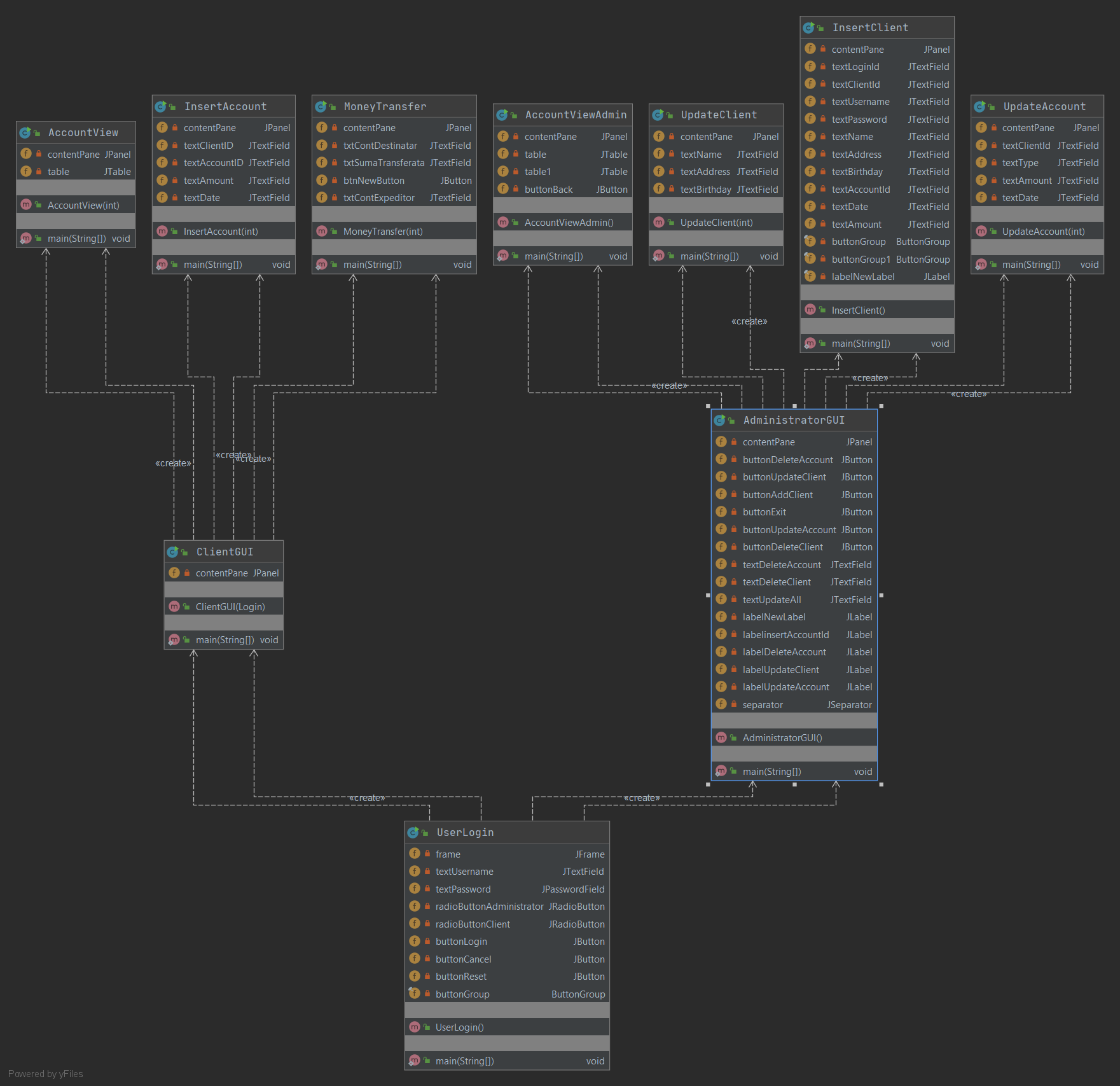
For this application it was used *Singleton Pattern for DataBaseConnection*: it was used because we need to have only one instance of our class. The DataBaseConnection is shared by multiple objects and it would be redundant to create a separate DataBaseConnections for each object (4. Singleton Design Pattern).

Along with the pattern described before, *Domain Model was used for Domain\_Logic\_Layer classes*: it incorporates both behavior and data, and the behavior responsibility is finely defined between classes.

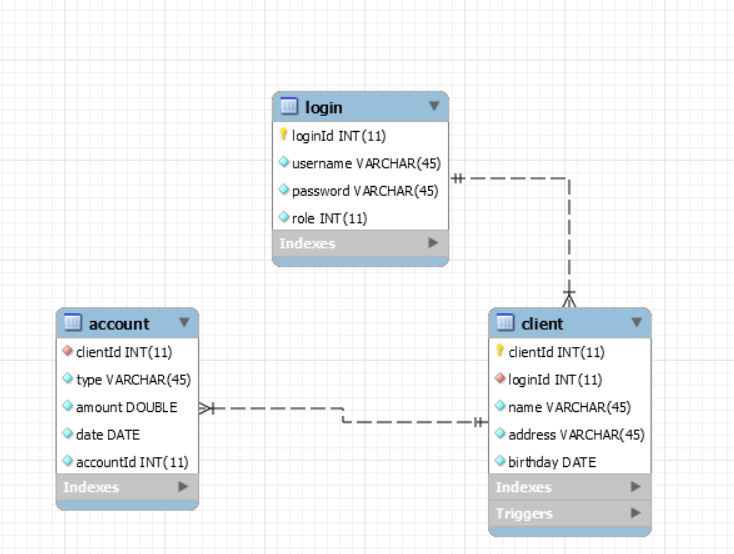
* 1. **UML Class Diagram**







1. Data Model

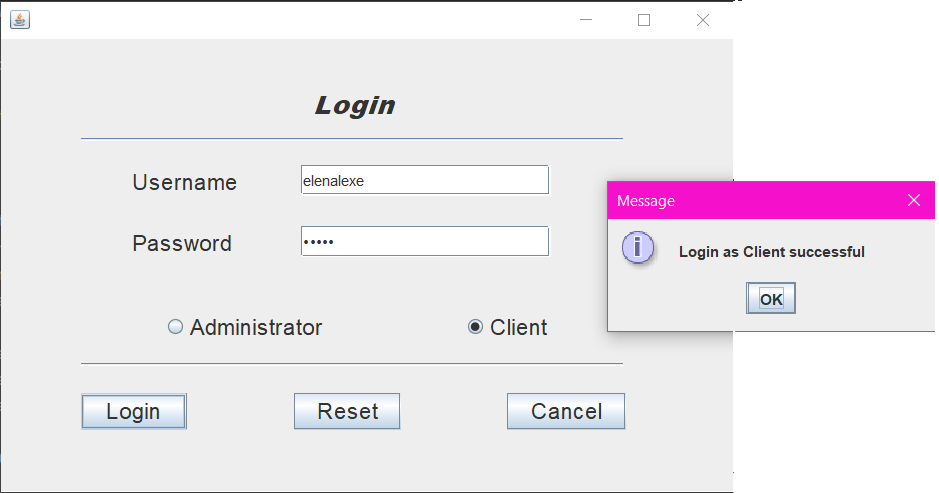
**

The application project has three tables in the Database: the Login table, the Client table and the Account table. The Login table contains 4 columns: loginId (which is also the primary key of the table), username, password and role. The Client table contains six columns: clientId (which is also the primary key of the table), loginId, name, address and birthday. The Account table contains five columns: clientId, type, amount, date and accountId (which is also the primary key of the table).

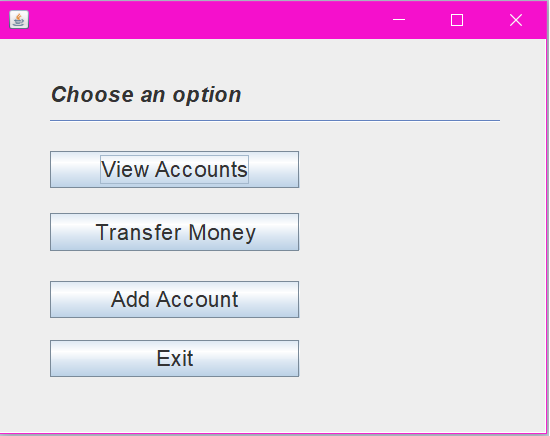
7. System Testing

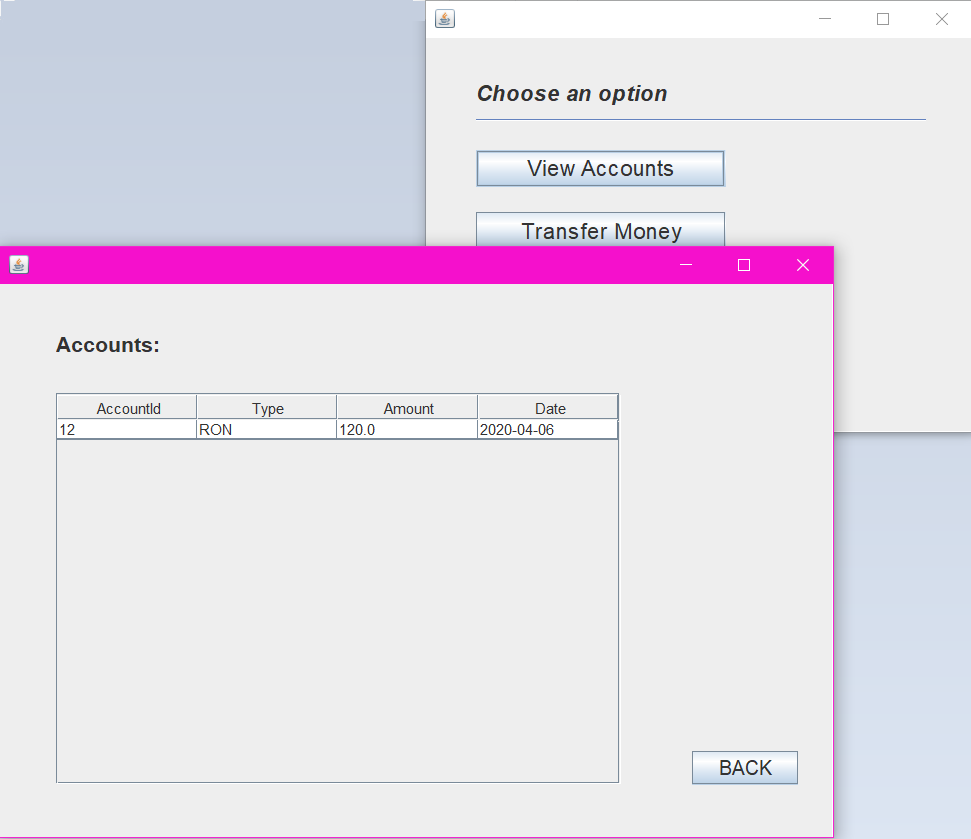
**Scenario 1: A client wants to send money from his account to another one**

* 1. Successful scenario:
     1. Login

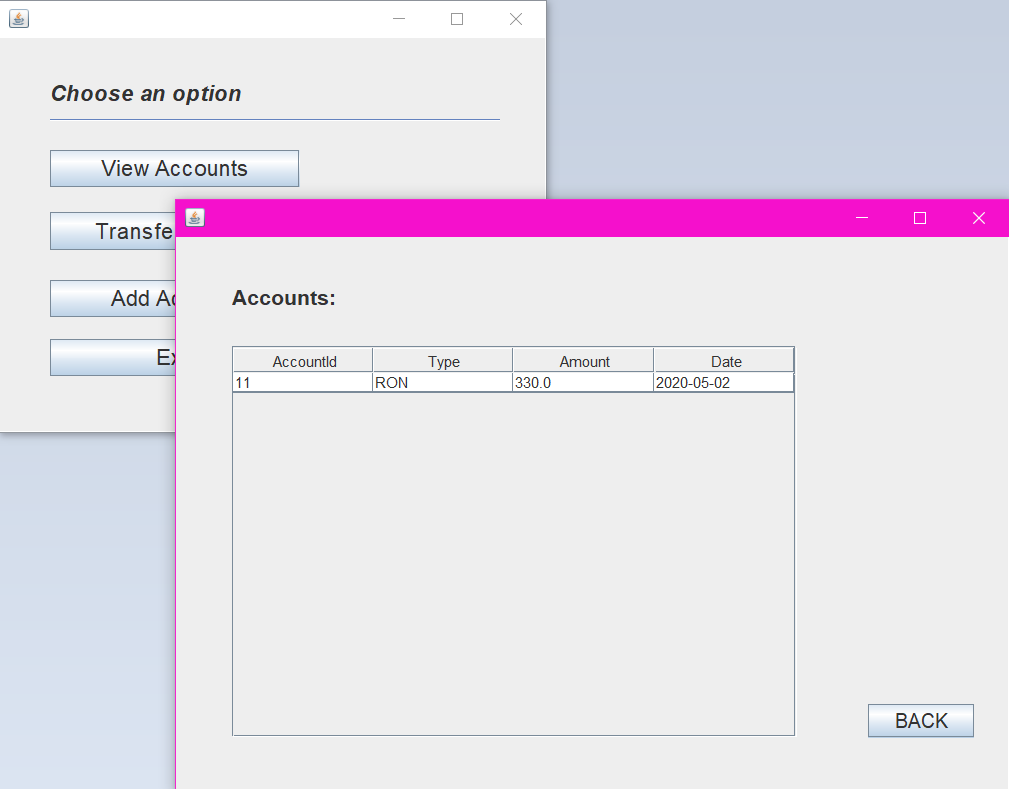


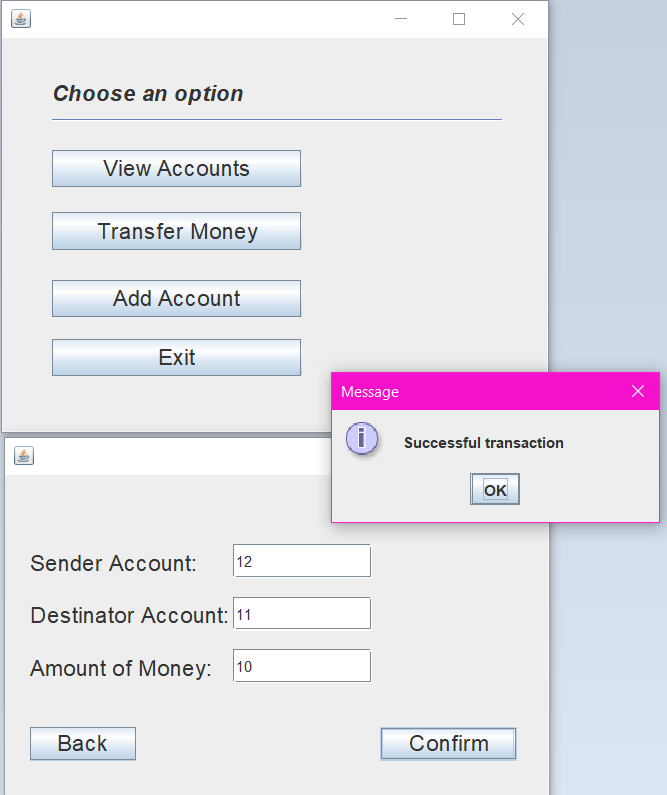
* + 1. View Account

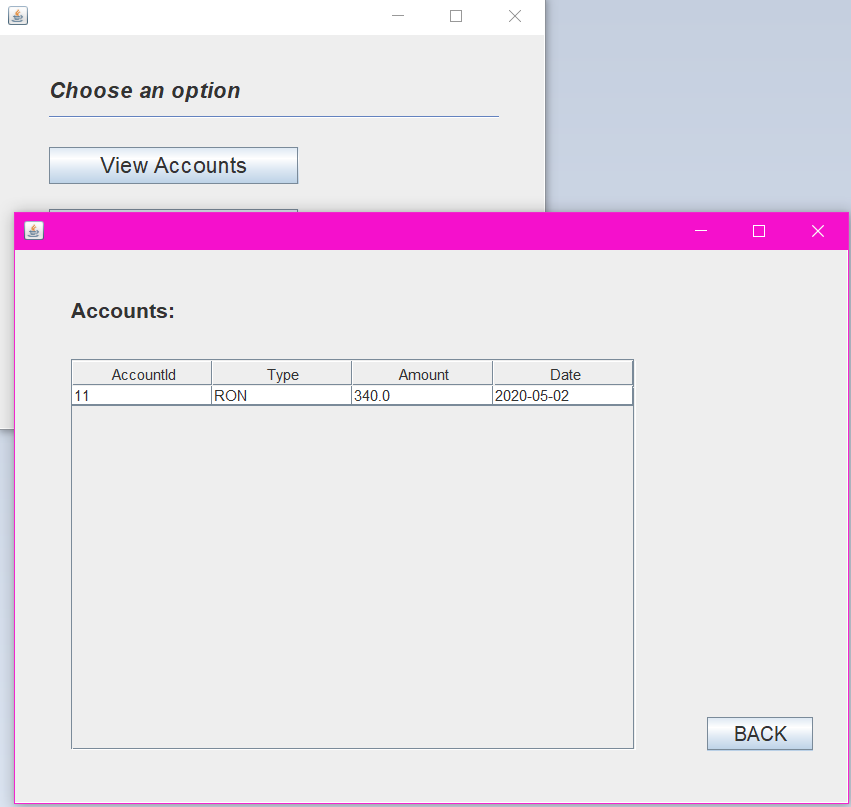


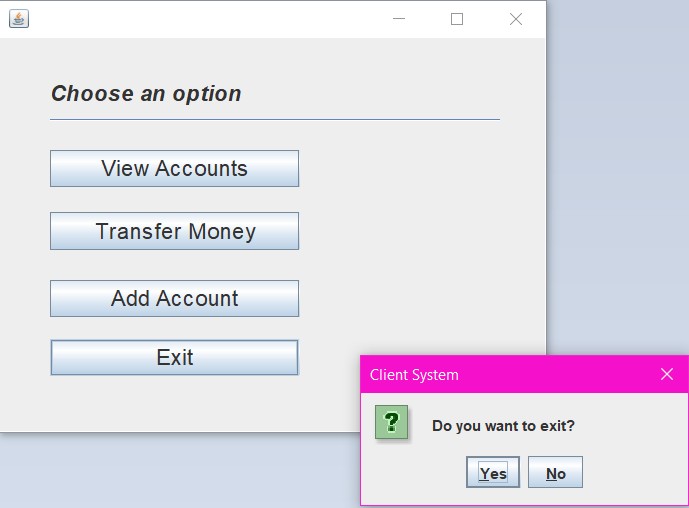


1.1.3 Transfer Money



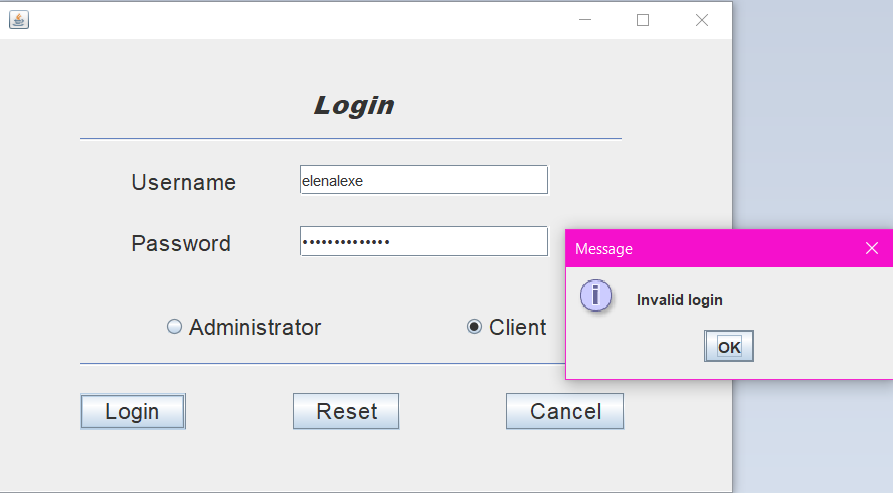




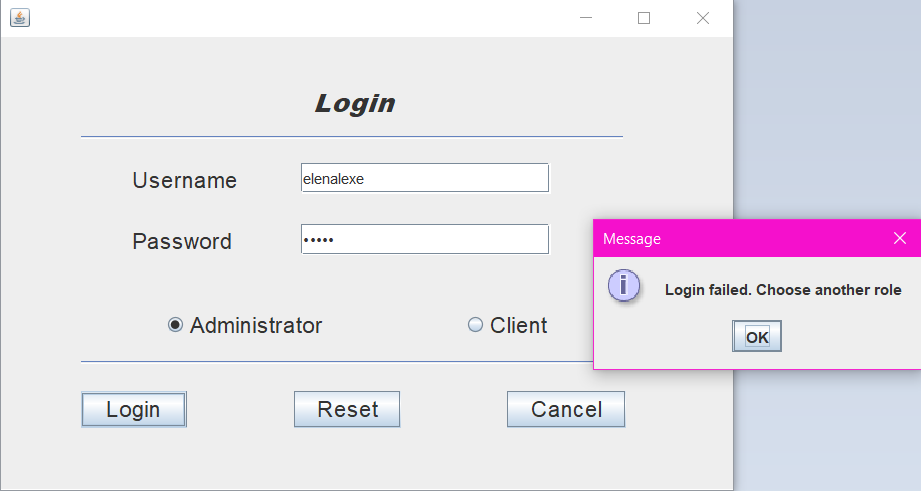


* 1. Unsuccsesful scenarios:

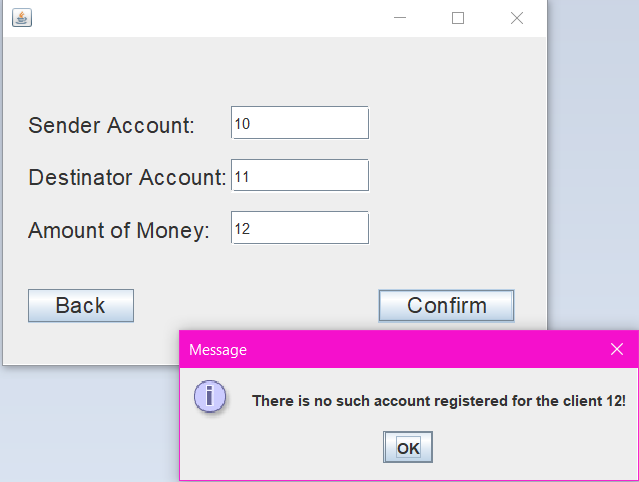
1.2.1Wrong username/password



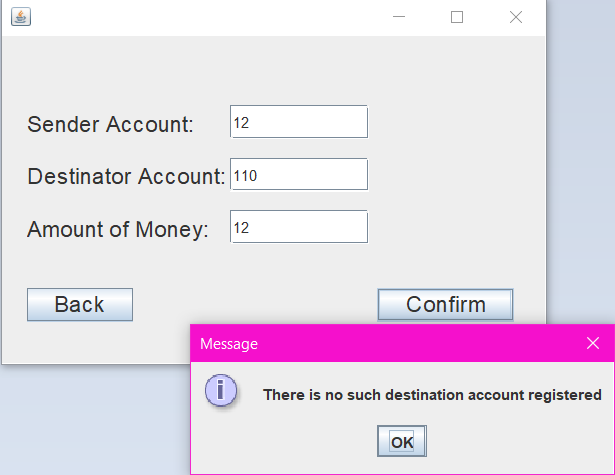
1.2.2 Wrong role



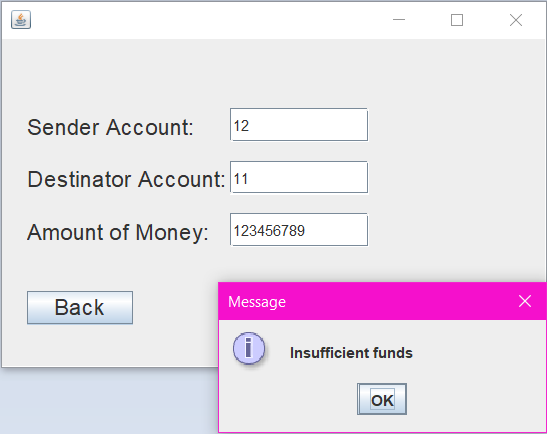
1.2.3 The client tries to send money from an account which is not his/hers



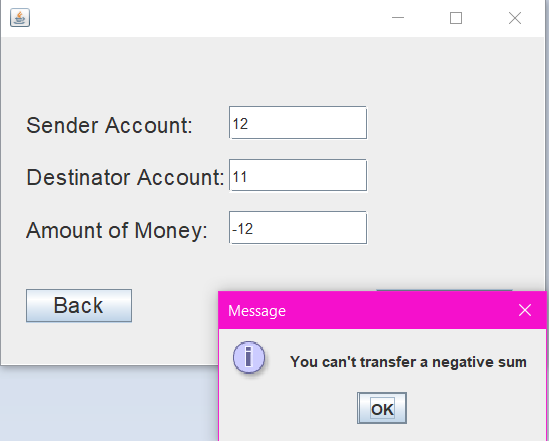
1.2.4 The client tries to send money to an account which is not existing



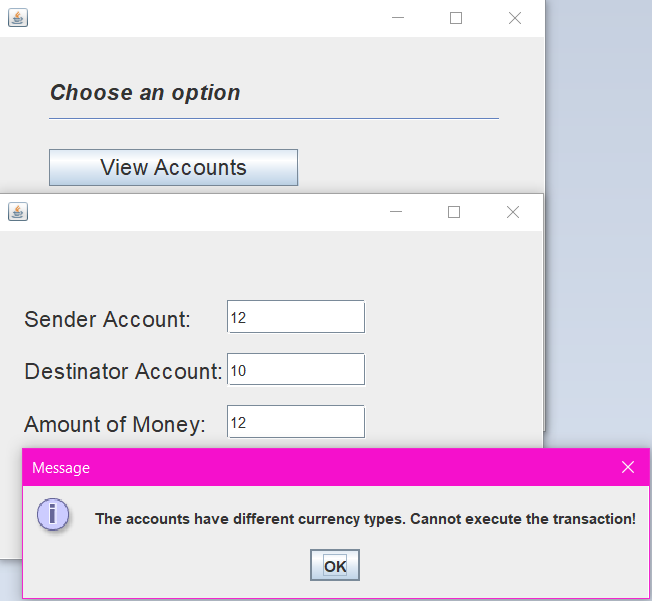
1.2.5 The client tries to send an amount bigger than his fonds



1.2.6 The client tries to send a negative amount of money



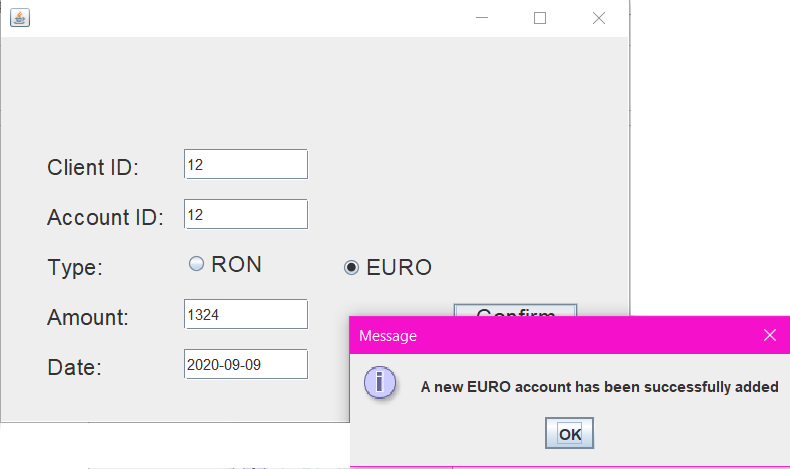
1.2.7 The client tries to send money to another account which has another currency selected



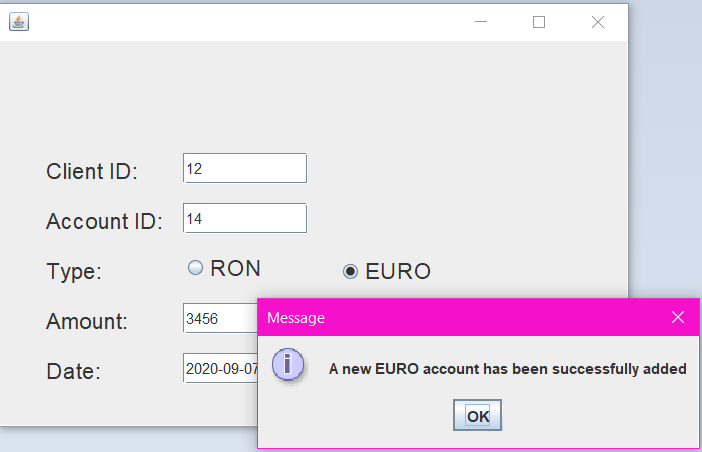
**Scenario 2: Adding another account**

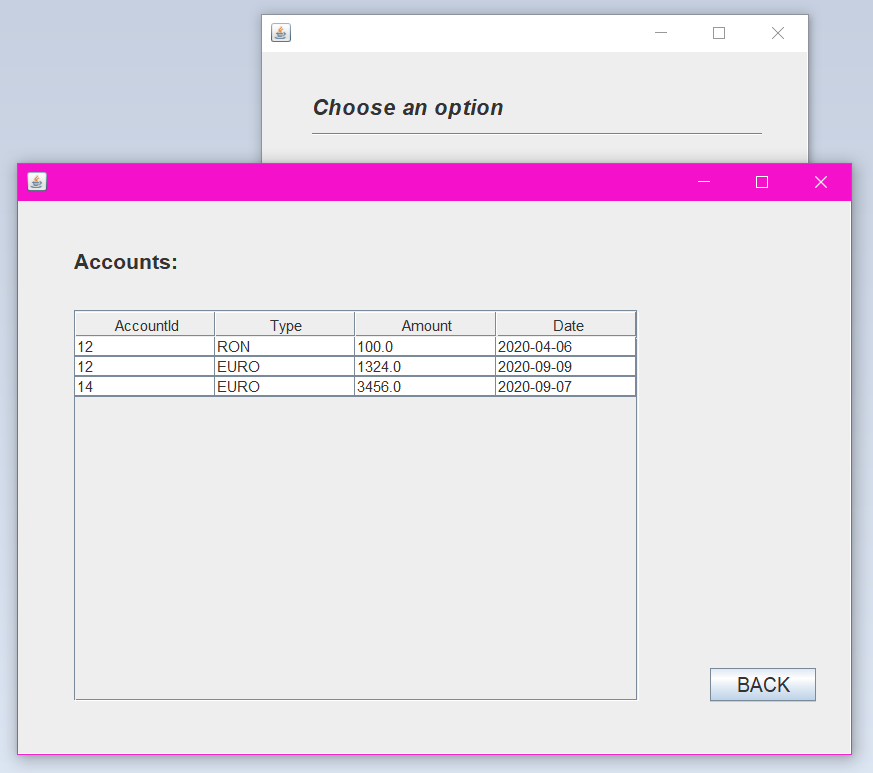
2.1: Succsessful scenarios

2.1.1: Creating an account with the same number, different currency



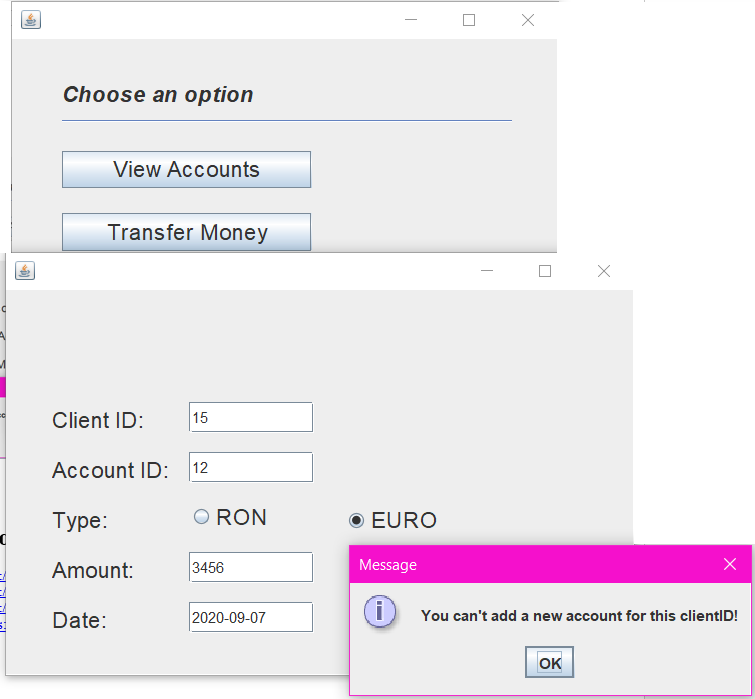
2.1.2 Creating an account with a different number, different currency





2.2 Unsuccesful scenarios

2.2.1 Wrong client id



8. Bibliography

1. The n-tier architecture

<https://stackify.com/n-tier-architecture/>

1. The n-tier achitecture

<https://www.guru99.com/n-tier-architecture-system-concepts-tips.html>

1. The n-tier architecture

<https://stackoverflow.com/questions/312187/what-is-n-tier-architecture>

1. Singleton Design Pattern

<https://www.geeksforgeeks.org/singleton-design-pattern/>