Bank management system

UTCN CTI ENGLISH

Muresan Daniel, 30422

Contents

1. **Goal of the application**

Banks are one of the most important part of the society. All good banks have good management systems is order to operate as faster as possible. For the clients this is a very important aspect, so a bank with an optimized and efficient management system will be more attractive to the people than one with a poor system.

This application is meant to simulate such a management system for the clients and the accounts of a bank, in order to efficiently keep track and manage the banking business. This application will store the clients and the accounts in table and save them to a serialized file from where them can be accessed any time from the application.

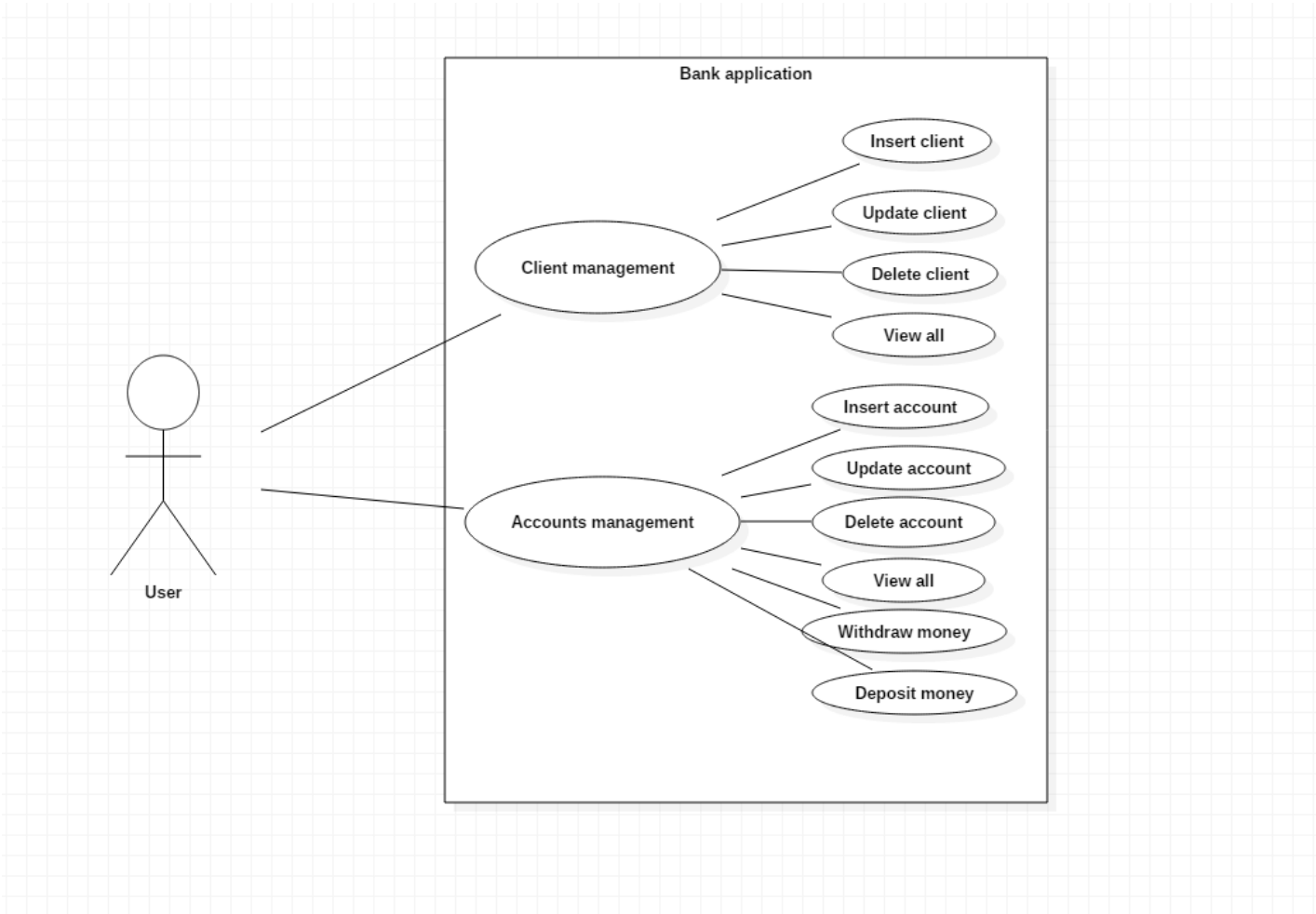
The management system will give the possibility to manage separately the clients and accounts respectively. Every client may have multiple accounts associated. The application will also have an easy-understandable graphic user interface with intuitive buttons, labels and text fields. All those will make it easy to use for any user regardless the experience and the knowledge in this domain.

2. **Problem analysis**

Depending on the data it wants to manage, the application must be able to receive some information about a customer or about an account and then give the possibility for the user to choose an operation to apply on that specific data like adding into the table, updating some entries from the table, deleting some other entries form the table or getting all the entries from a specific table or withdraw and deposit in the case of accounts.

The assumption that I made for this application are the following ones: every client and every account respectively are uniquely identified by an ID which can't have duplicates and the uses must introduce the data for every client and every account with respect to the unique ID which cannot be modified. Because of this assumption I will use only the ID for inserting, deleting and updating the entries for the table.

The use cases of the application are presented in the next use case diagram.



- **Use case: Insert client**
- **Primary actor: User**
- **Main success scenario:**

    - The user presses the button for managing clients.
    - The user introduces data about a new client and presses the button for insert.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application takes the previous data from the serialized file.
    - The application checks if the entry is not already in the table and then introduces it into the table.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**
    - The user presses the button for managing clients.
    - The user introduces data about a new client and presses the button for insert.
    - The application founds some invalid data or some fields that are not filled.
    - The application provides an error message.
    - The sequence goes back to the point where the user introduces data about a client.

- **Use case: Delete client**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing clients.
    - The user selects a row from the table and presses the button for delete.
    - The application takes the previous data from the serialized file.

    - The application checks if the entry is in the table and then deletes it from the table.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**
    - The user presses the button for managing clients.
    - The user does not select a row from the table and presses the button for delete.
    - The application provides an error message.
    - The sequence goes back to the point where the user selects a row from the table and then presses the delete button.

- **Use case: Update client**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing clients.
    - The user selects a row from the table, introduces updated data about the selected row and presses the button for update.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application takes the previous data from the serialized file.
    - The application checks if the entry is in the table and then updates it in the table with data from the field.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**

    - The user presses the button for managing clients.
    - The user does not select a row from the table or does not complete all fields correctly and presses the button for update.
    - The application provides an error message.
    - The sequence goes back to the point where the user selects a row from the table and introduces data in the field and then presses the update button.

- **Use case: View all clients**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing clients.
    - The user presses the button for view all.
    - The application takes the previous data from the serialized file.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.

- **Use case: Insert account**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing accounts.
    - The user introduces data about a new account and presses the button for insert.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application takes the previous data from the serialized file.

    - The application checks if the entry is not already in the table and then introduces it into the table.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**
    - The user presses the button for managing accounts.
    - The user introduces data about a new account and presses the button for insert.
    - The application founds some invalid data or some fields that are not filled.
    - The application provides an error message.
    - The sequence goes back to the point where the user introduces data about an account.

- **Use case: Delete account**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing accounts.
    - The user selects a row from the table and presses the button for delete.
    - The application takes the previous data from the serialized file.
    - The application checks if the entry is in the table and then deletes it from the table.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**
    - The user presses the button for managing accounts.

    - The user does not select a row from the table and presses the button for delete.
    - The application provides an error message.
    - The sequence goes back to the point where the user selects a row from the table and then presses the delete button.

- **Use case: Update account**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing accounts.
    - The user selects a row from the table, introduces updated data about the selected row and presses the button for update.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application takes the previous data from the serialized file.
    - The application checks if the entry is in the table and then updates it in the table with data from the field.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.
- **Alternate case scenario:**
    - The user presses the button for managing accounts.
    - The user does not select a row from the table or does not complete all fields correctly and presses the button for update.
    - The application provides an error message.

    - The sequence goes back to the point where the user selects a row from the table and introduces data in the fields and then presses the update button.

- **Use case: View all accounts**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing accounts.
    - The user presses the button for view all.
    - The application takes the previous data from the serialized file.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.

- **Use case: Deposit money**
- **Primary actor: User**
- **Main success scenario:**
    - The user presses the button for managing accounts.
    - The user introduces the sum and then presses the button for deposit.
    - The application takes the previous data from the serialized file.
    - The application provides a success message if there were no errors during the process.
    - The application updates the serialized file.

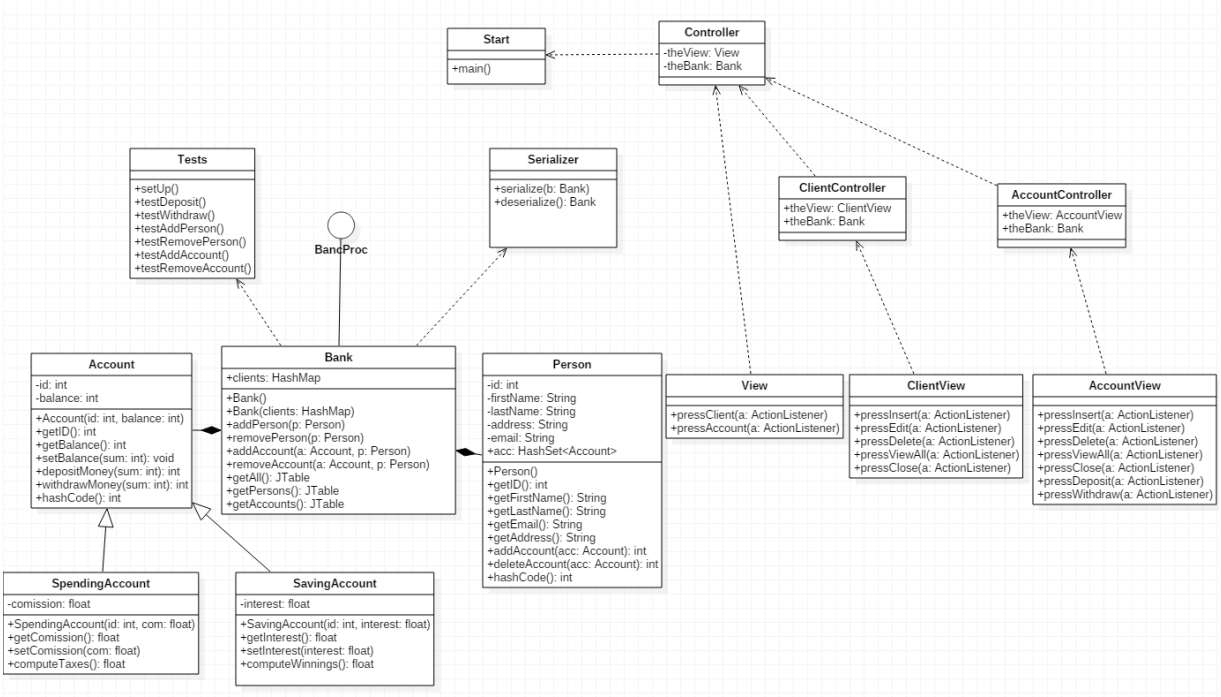
- **Use case: Withdraw money**
- **Primary actor: User**

- **Main success scenario:**
  - The user presses the button for managing accounts.
  - The user introduces the sum and then presses the button for withdraw.
  - The application takes the previous data from the serialized file.
  - The application provides a success message if there were no errors during the process.
  - The application updates the serialized file.

3. **Projecting**

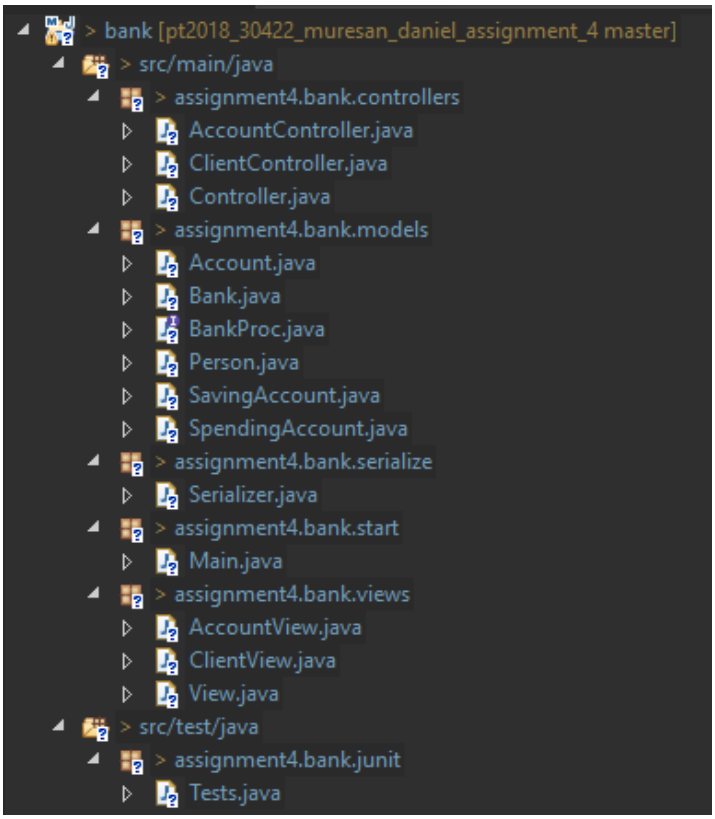
The classes I used for developing this application are the following ones: Client, Account, SpendingAccount , SavingAccount , Bank and the interfaceBankProc as models; Serializer for data serialization; View, ClientView and AccountView as views; Controller, ClientController and AccountController as controllers; Tests class for JUnit testing and the class Start used for launching the application.

All the classes and the relationships between them are displayed in the next class diagram.



I used a model view controller pattern with two extra packages for organizing the packages. I used the next packages as follows: assignment4.bank.models, assignment4.bank.views, assignment4.bank.controllers, assignment4.bank.serialize, assignment4.bank.junit and assignment4.bank.start.

The structure of packages is presented in the next image.



4. **Implementation**

Person class

Is the class that represents the clients for the bank. The class has few attributes: clientID, firstName, lastName, mail, access and a hash set with accounts. The class has a constructor with all attributes given as parameters, getters, setters and methods for adding and removing accounts from the hash set. It also has a method for calculating the hash code with respect to the id.

Account class

Is the class that represents the accounts from the bank. The class has two attributes: id and balance. It has a constructor with the attributes given as parameters, getters and setters, methods for deposit and withdraw money

and a method for calculating the hash code with respect to the id. It has two subclasses: SavingAccount and SpendingAccount.

SavingAccount class

Is a subclass of the class Account it has an extra attribute for interest. It has a constructor and a method for calculating the winnings.

SpendingAccount class

Is a subclass of the class Account it has an extra attribute for commission. It has a constructor and a method for calculating the taxes when withdrawing money.

BankProc interface

Is the interface which defines methods for adding a removing clients and accounts. It also defines methods for getting data in a table from the hash map where the data is stored.

Bank class

Is the class where all clients and accounts are stored and managed. It has an attribute called clients which is a hash map with persons as keys and hash sets with the accounts of the persons as values. There are implemented all methods from the BankProc interface.

Serializer class

Is the class where the serialization of the data is made. There are two static methods: one for introducing the data in the serialized file and one for retrieving data from the serialized file.

View class

Is a component of the graphical user interface and it gives the possibility for the user to choose between managing clients and managing accounts.

ClientView class

Is another component of the graphical user interface and lets the user to handle the clients from the hash map with the help of a table.

AccountView class

Is the last component of the graphical user interface and lets the user to handle the accounts from the hash map with the help of a table.

Controller class

Is the class that manages the functionality of the instances of the View class.

ClientController class

Is the class that manages the functionality of the instances of the ClientView class.

AccountController class

Is the class that manages the functionality of the instances of the AccountView class.

Start class

Is the class which has the method main() used to launch the application by creating an instance of the Controller class.

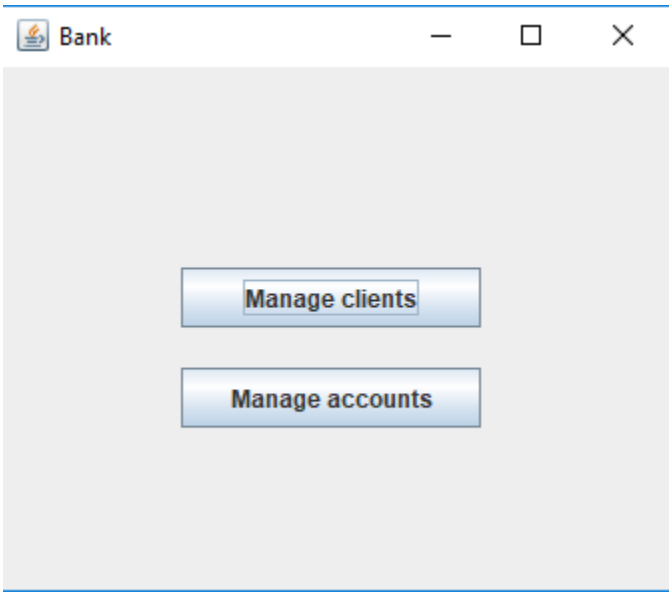
5. **Testing**

For testing I created a separate class where I used the JUnit framework to create few methods for testing the most used methods in the app.So there I have the following methods: testDepositMoney(), testWithdrawMoney(), testAddPerson(), testRemovePerson(), testAddAccount(), testRemoveAccount().

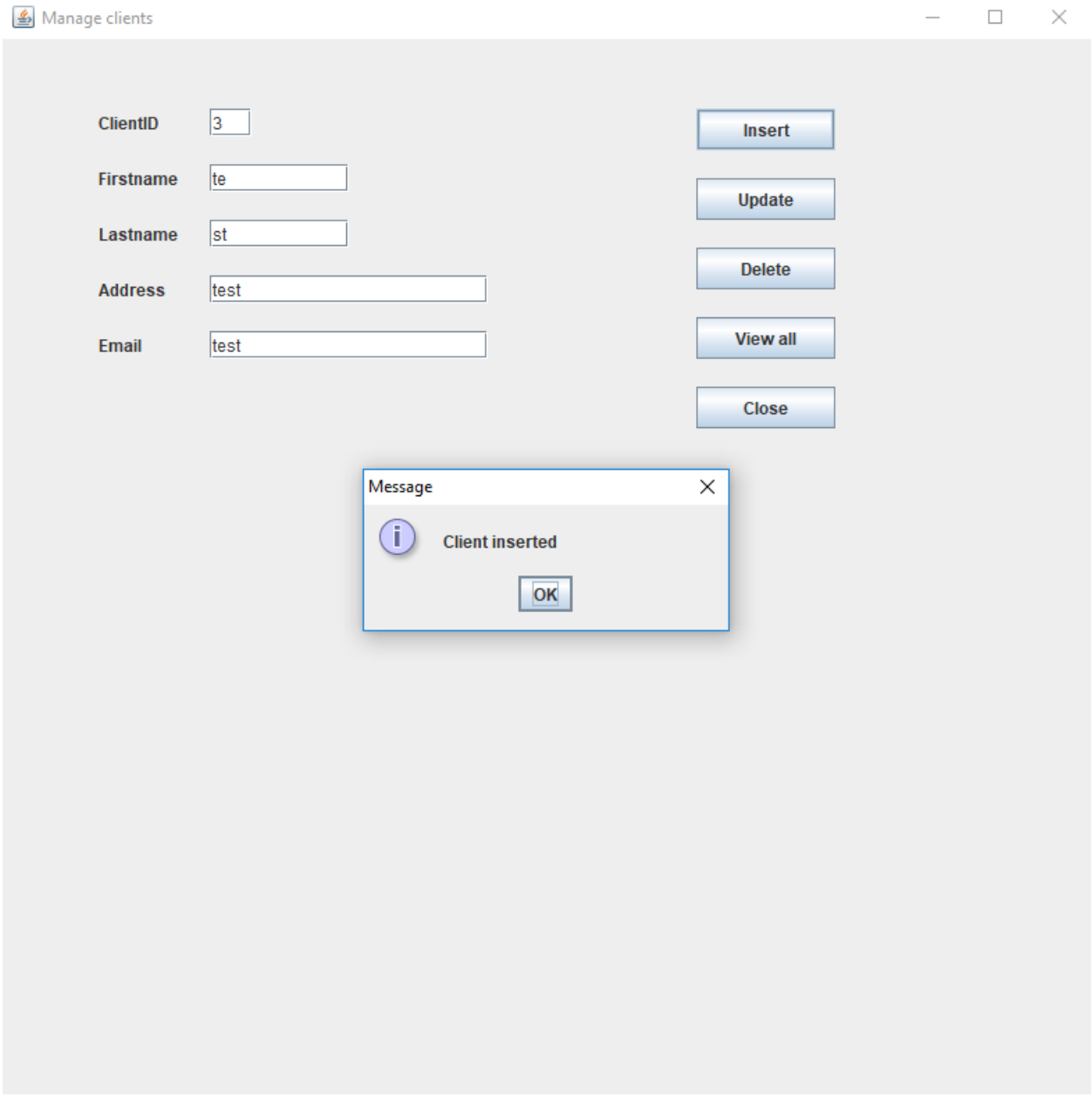
6. **Results**

In this section I will present a few screenshots with the final application.

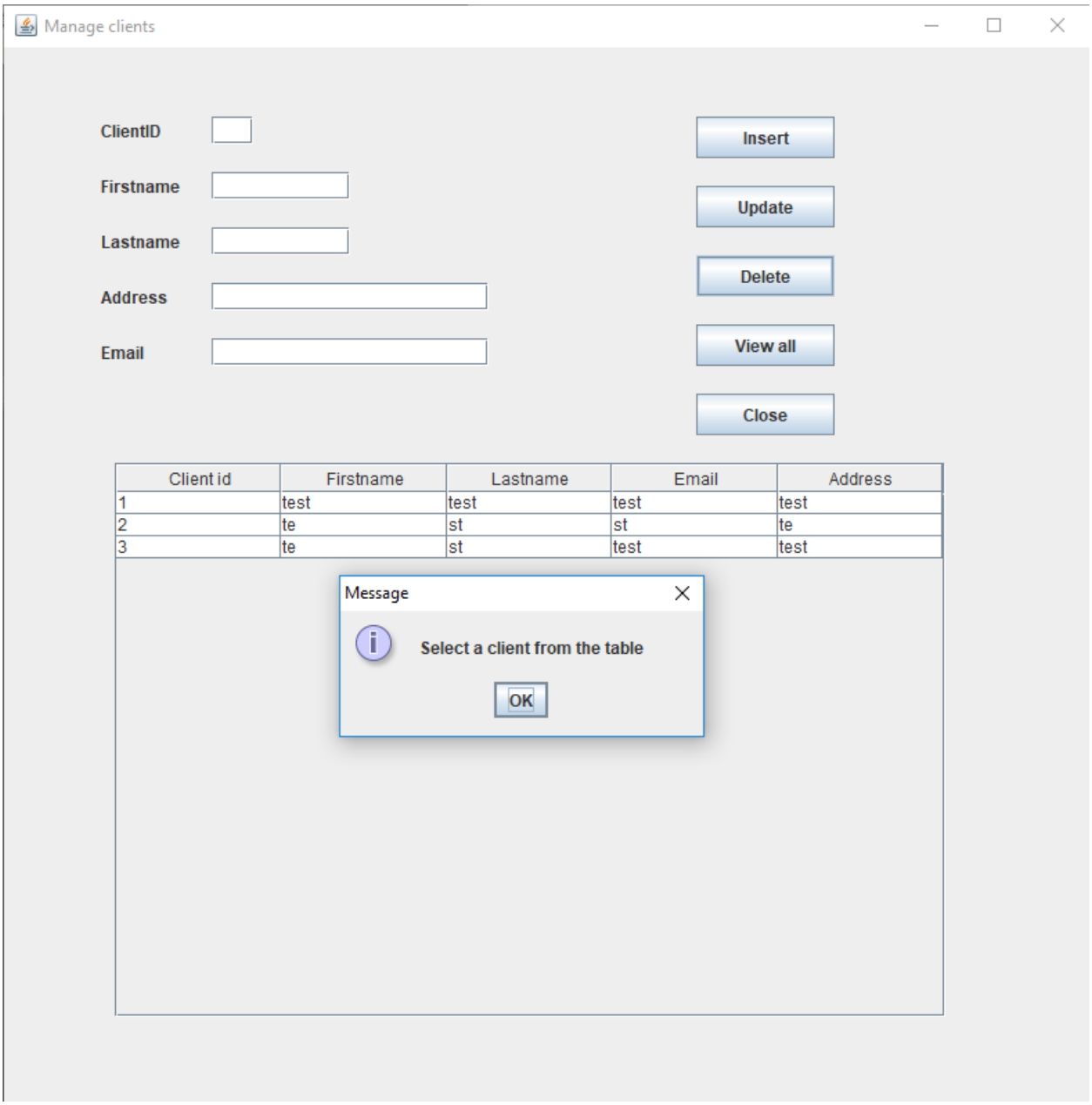
The first image is with the frame the user sees when the application is launched.



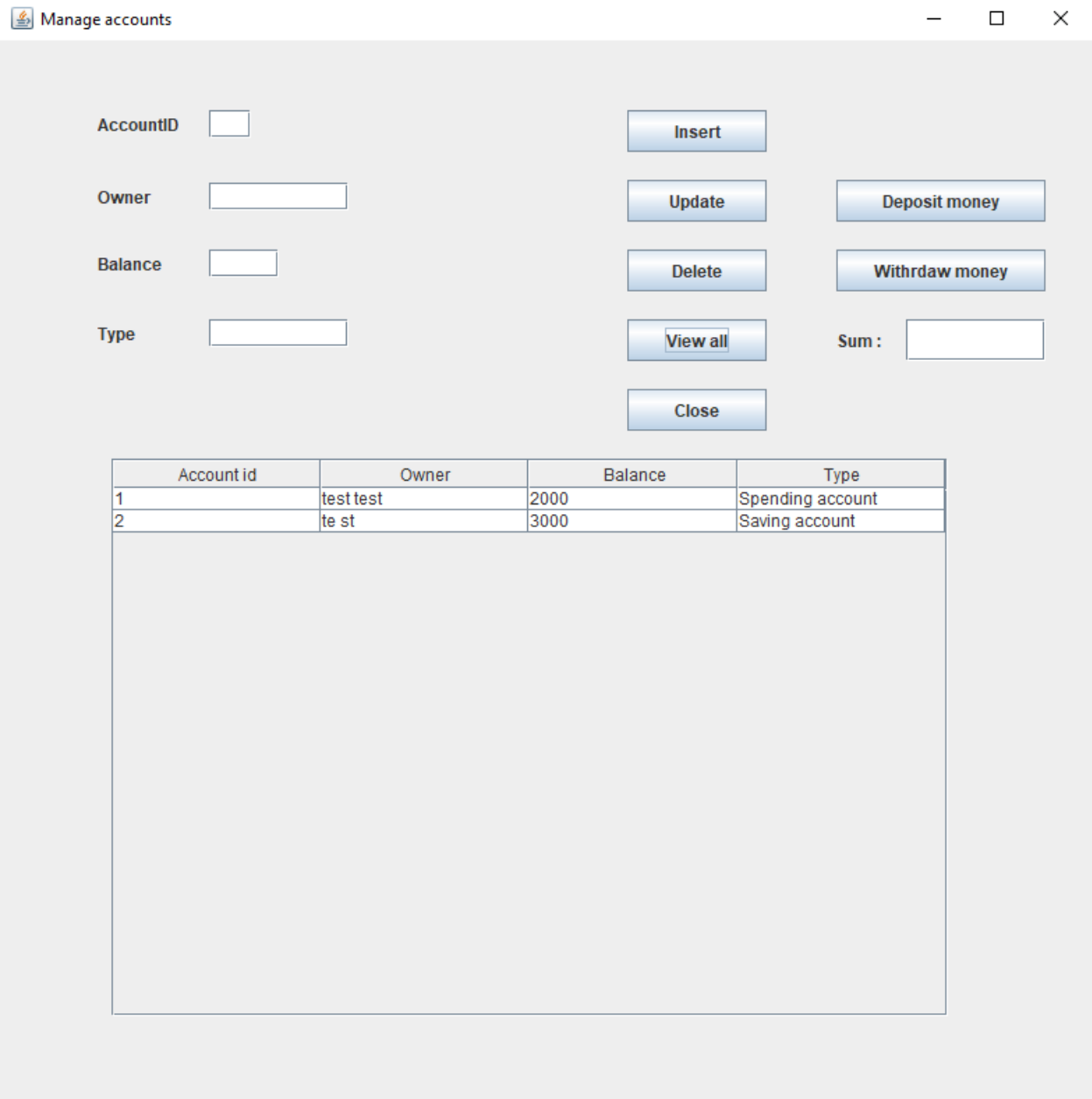
The next image is the result of the operation insert for a client.



The next image is the result of a wrong delete operation for a client.



The last image is the view all operation for accounts



7. **Conclusions**

By developing this application I learnt to use serialization, hashing and working with hash sets and hash maps.

8. **Bibliography**

http://www.tutorialspoint.com/java/java_serialization.htm

https://www.youtube.com/watch?v=Zt4g6HiFNxo

https://dzone.com/articles/java-hashing

https://www.youtube.com/watch?v=c3RVW3KGIIE