*Technical University of Cluj-Napoca*

*Faculty of Automation and Computers*

*Department if Computer Science*

*2nd Semester 2015-2016*

*Programming Techniques*

*Homework 4*

Student: Hitter András

Group: 30424

Contents

[What is a 'Bank' 3](#_Toc451252457)

[What is a 'Flexible Spending Account - FSA' 3](#_Toc451252458)

[1. Example of working 5](#_Toc451252459)

[2. Design 6](#_Toc451252460)

[2.1. Relational Diagram 6](#_Toc451252461)

[3.3 Packages and Interfaces 10](#_Toc451252462)

[3. Using and testing the application 13](#_Toc451252463)

[4. Conclusions 15](#_Toc451252464)

[5. References 16](#_Toc451252465)

Abstract

## What is a 'Bank'

A bank is a financial institution licensed as a receiver of deposits. There are two types of banks: commercial/retail banks and investment banks. In most countries, banks are regulated by the national government or central bank.

**What is a 'Savings Account'**

A savings account is a deposit account held at a bank or other financial institution that provides principal security and a modest interest rate. Depending on the specific type of savings account, the account holder may not be able to write checks from the account (without incurring extra fees or expenses) and the account is likely to have a limited number of free transfers/transactions. Savings account funds are considered one of the most liquid investments outside of demand accounts and cash. In contrast to savings accounts, checking accounts allow you to write checks and use electronic debit to access your funds inside the account. Savings accounts are generally for money that you don't intend to use for daily expenses. To open a savings account, simply go down to your local bank with proper identification and ask to open an account.

## What is a 'Flexible Spending Account - FSA'

A Flexible Spending Account (FSA) is a type of savings account available in the United States that provides the account holder with specific tax advantages. Set up by an employer for an employee, the account allows employees to contribute a portion of their regular earningsto pay for qualified expenses, such as medical expenses or dependent care expenses.

Homework 3

**Objective**

Design by Contract Programming Techniques

**Description**

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. Define 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. Implement a test driver for the system.

6. The account data for populating the Bank object will be loaded/saved from/to a file.

# Example of working

Assumed the entered values are correct (for example 3/50/1/5) the program takes these values and processes this way:

* The **Bank** is the class which creates new **Accounts** and **Persons**. Each Server gets one thread. The servers are organized in a List, so we can check on all of them at the same time. It also controls the clients, which means that it does not let for a Server to have more than 10 (in this case) clients at the same time.
* The **Simulator** creates a new Scheduler and generates the processing times for each “client” and calls the Display method
* The SimulatorFrame it simply checks the number of servers and visualizes the processes and clients as it is necessary.
* The Handler Class controls what happens when we push the START or STOP button and reads the user inputs.

# Design

## Relational Diagram

To resolve the problem, I’ve chosen to use eight different classes, so that the design of application would be easier.

* Account
* Bank
* BankProc
* GUI
* Person
* SavingAccount
* SpendingAccount
* Start

To get a better view related to the attributes of each class, there are below the UML diagrams for each class. Thus, we can see every class with objects and their methods.

D:\Letöltések\Banking.png

Class Design

1. *GUI*: **public** **class** GUI **extends** JFrame. This class is designed to create a graphical user interface so the application would be easier to use. We consider the Interface class as a subclass of the predefined class "JFrame" so we can use objects of type "button", "frame" or "panel".

In this class there is only a constructor and a createNewAccount method, by wich is realisied the creation of a container. To this container there are added more panels: inputData, panel. The previous one is responsible for containing -many- JTextFields in which the user may enter the parameters. The last one is responsible for containing buttons, for activating different methods of the program: start, stop. The program will not accept the command start again after the stop was pressed. In case of not filling all fields correctly an error message appears and says: Fill all the fields!!!.

The attributes if the SimulatorFrame Class:

To achieve the desired SimulatorFrame I needed many attributes of different types:

* A JFrame: it is created by extending the JFrame
* Panels:
  + Private panel adminPanel – contains all the text boxes necessary to maintain the communication with the user
  + Private panel - contains all the buttons necessary which by the user interacting with the program can understand what he or she wants and execute the corresponding operation(s).
* 2 Buttons for executing the desired operations by the user:
  + Public static JButton back;
  + Public static JButton adminLogin;
  + Public static JButton ClientLogin;
* Textfields:
  + Public static JTextField title;
  + Public static JTextField ctitle;
  + Public static JTextField min;
  + Public static JTextField max;
* Constructor:
  + Public SimulatorFrame() {…} – the initializations, coloring, placing(layout) are done here.

The following elements will be created: frame, textfields, buttons, labels.

* Creating a new HandlerClass object:
  + HandlerClass handler = new HandlerClass();

Every button is added to and ActionListener: start.addActionListener(handler);

stop.addActionListener(handler);

1. *HandlerClass* class: public class HandlerClass implements ActionListener;

This class was created to keep track of the user activity, and do operations corresponding to the user will. This class implements ActionListener class: public class HandlerClass implements ActionListener;

This class contains:

* banc object( attributes ):
* client object
* ( attributes ) pin code
* The actionPerformed(ActionEvent): void method.

This method always listens to the buttons, textfields. If the user enters something this method reacts and performs operations predefined for the case that a particular button (or enter in a textfield) is pressed.

Also checks the correctness of the input. It will give instructions if an incorrect input occurs.

1. *Account* class

This class represents a bank account from real life! This class contains the abstract of all the operations which can be done using a real life bank account.

In this class there is:

CONSTRUCTOR which sets all the necessary information about the account to

**public** **abstract** **void** addMoney(**double** sum);

**public** **abstract** **boolean** withdrawMoney(**double** sum);

**public** **abstract** **int** getAccID();

**public** **abstract** banking.Person getClient();

**public** **abstract** **double** getMoney();

**public** **abstract** **int** getAccType();

**public** **abstract** String getPIN();

@Override

**public** **boolean** equals(Object o) {

}

@Override

**public** **int** hashCode() {

}

@Override

**public** **abstract** String toString();

This method displays the tasks, so we see what is happening right now.

1. *Person* class

This class is responsible for modelling a real Person aka Customer at the bank.

In the run method takes some tasks. In the addTask method adds factually the tasks and in the getTasks method returns the tasks.

1. *SavingAccount and SpendingAccount* classes

The first one in the run method adds tasks to a server and displays them until it reached the finish time or the STOP button is pressed.

The second one displays the servers and clients for the Simulator class.

## Packages and Interfaces

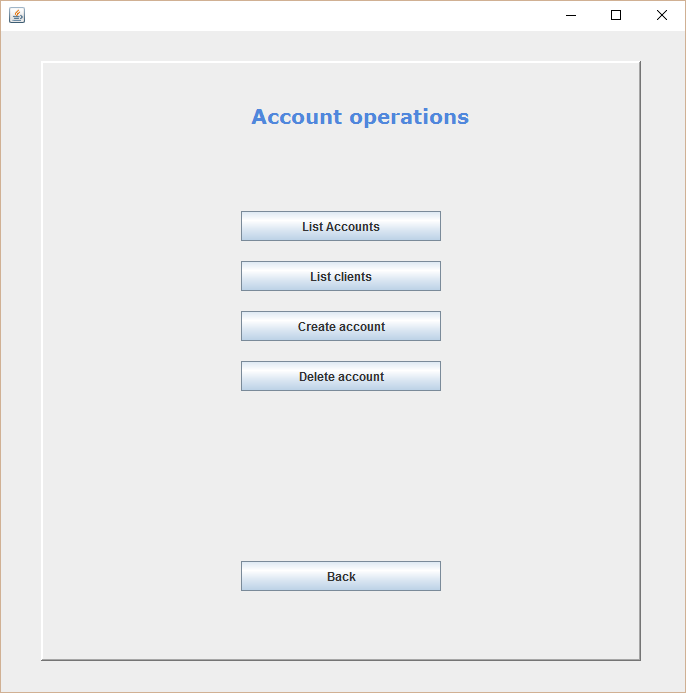
A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. A package provides a unique namespace for the types it contains. Classes in the same package can access each other's package-access members.

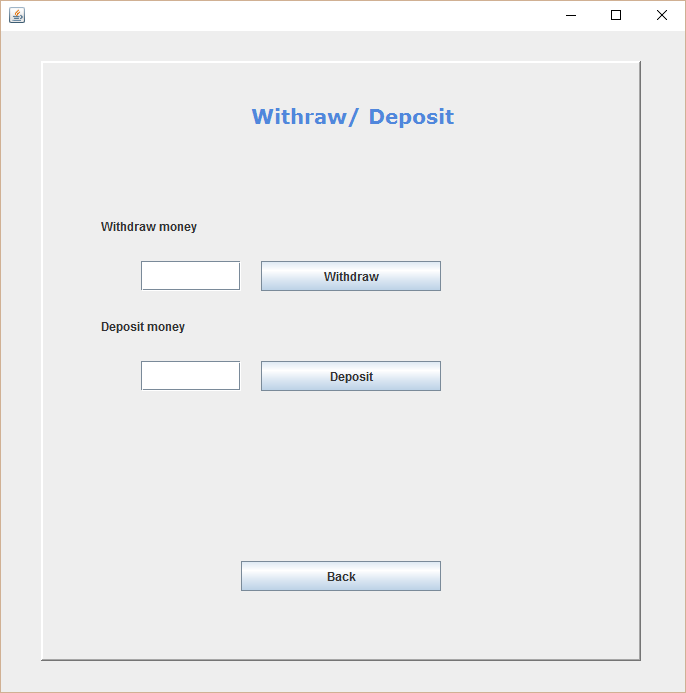
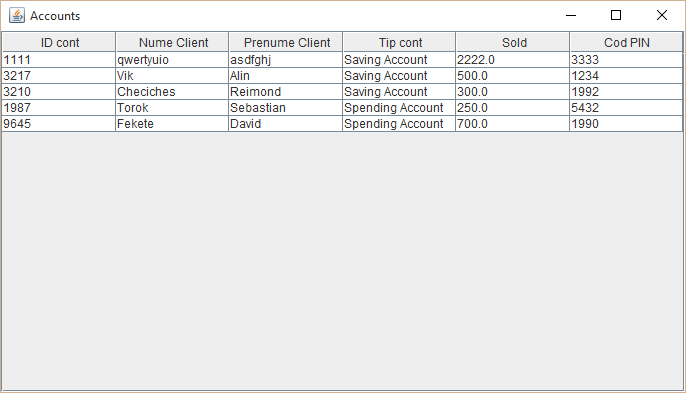
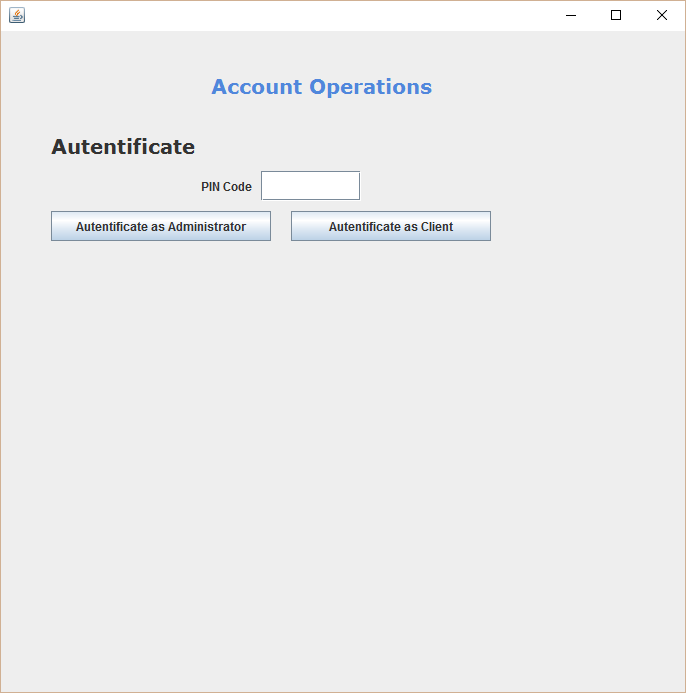
A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks.

For this application the following packages are imported, each of them having a certain role for the proper working of the application. We import them in the Gui Class (most of them relate to the user interface properties):

* import java.awt: Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.
  + java.awt.BorderLayout: A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.
  + java.awt.Color: The Color class is used encapsulate colors in the default RGB color space or colors in arbitrary color spaces identified by a [ColorSpace](http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/color/ColorSpace.html).
  + java.awt.Dimension: This encapsulates the width and height of a component (in integer precision) in a single object.
  + java.awt.GridLayout: The GridLayout class is a layout manager that lays out a container's components in a rectangular grid for a better view of all the buttons and textfields which are added to the main panel.
* import java.awt.event
  + java.awt.event.ActionEvent;
  + java.awt.event.ActionListener;
* import javax.swing: Typical Swing applications do processing in response to an event generated from a user gesture. For example, clicking on a JButton notifies all ActionListeners added to the JButton. That’s why we use this package for creating the user interface Gui.
  + javax.swing.JButton;
  + javax.swing.JFrame; javax.swing.JLabel; javax.swing.JPanel;

## Using and testing the application

In order to use the application open Homework3. This will open a window which generates the Gui class. Thus the user can enter the desired values and selecting start or stop buttons. 



# Conclusions

Another design pattern learned!

It was a really good practice, which taught me many things about this kind of programming.

# References

[*http://stackoverflow.com/*](http://stackoverflow.com/)

[*http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html*](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html)

*Teacher(s)*