Implementing and simulating a bank

Technical University of Cluj-Napoca

Student: Natalia Carina Sîrbu

Computer Science, 2nd year, group 30425

May 10, 2017

Table of Contents

[1. Introduction 3](#_Toc482368603)

[2. Class description 3](#_Toc482368604)

[2.1. Model Package 3](#_Toc482368605)

[2.1.1. Account Class 3](#_Toc482368606)

[2.1.2. Saving Account Class 3](#_Toc482368607)

[2.1.3. Spending Account Class 4](#_Toc482368608)

[2.1.4. Person Class 4](#_Toc482368609)

[2.1.5. Serialization Handler 4](#_Toc482368610)

[2.1.6. Observer and Observable class 4](#_Toc482368611)

[2.1.7. Bank Proc Interface 5](#_Toc482368612)

[2.2. Graphical User Interface Package 5](#_Toc482368613)

[3. Class diagram 5](#_Toc482368614)

[4. Bibliography 6](#_Toc482368615)

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# **Introduction**

In order to implement this application, two new concepts need to be explained: Observer Design Pattern and Design by Contact.

The Design Pattern technique, also known as the Observer Pattern, is a software design pattern that uses an object (called subject), which has a list of its dependents, called observers. All these observers will be notified when a changed happens in the observable object. The notification will be possible by calling one of the methods from the Observable class.

The Design by Contact technique is a software correctness methodology. It uses preconditions and postconditions to document (or programmatically assert) the change in state caused by a piece of a program. Classes should specify their invariants: what is true before and after executing any public method. Methods should specify their pre- and post-conditions: what must be true before and what must be true after their execution, respectively. Pre-conditions may be weakened by subclasses and post-conditions may be strengthened.

# **Class description**

## Model Package

### Account Class

This class is an abstract one and it implements two interfaces: Serializable and Observable. Three instance variables are declared as protected (visible only in the subclasses and in the superclass’s package; even if the subclasses are in a different package than the superclass, the variable are still visible). The first one is *protected String id*, that represents the id of the account which will be generated randomly using UUID.*randomUUID* ().toString (). The second one is *protected int sum* which represents the sum from the account. The third one is an instance of the Observer class *protected Observer observer*.

* *public void addObserver (Observer o)* – this is a method inherited from the Observable class that adds an observer to the account.
* *public void deleteObserver (Observer o)* – this is another method inherited from the Observable class that deletes an observer.
* *public void notifyObserver (int oldSum, int currentSum, String sort) ­*– this method will call the update method on the observer. The update method is in the Observer interface.

The getters and setters of the variable instances are declared in this class and also the *toString ()* method is overridden.

### Saving Account Class

This class extends the Account class and it has only one private instance variable of type long named *startTime*. This is used to compute the interest. The particularity of this class that the holder cannot deposit any money and he or she can perform only one withdrawal.

* *public void computeInterest (long time)* – this method will return the interest computed by the following formula: 5% of the sum each second.
* *public void withdraw ()* – the name is suggestive to the functionality of the method: to the sum to be withdrew will be added the interest computed by calling the *computeInterest* method. This will also notify the obsever.

### Spending Account Class

This class extends the Account class and it also inherits all its method.

* *public void deposit (int s)* – this method will just add the value given as parameter to the existing sum from the account. It will also notify the observer by calling the *notifyObserver*.
* *public void withdraw (int s) ­*– this method will just subtract from the existing balance the value given as parameter and it will notify the observer.

### Person Class

This class implements two interfaces: Serializable and Observer. Three instance variables are declared as private: *private String ssn* (this is the social security number that is unique for every person in the bank and in this application will be generated randomly using UUID.*randomUUID* (). toString () ), *private String firstName* and *private String lastName.* The getters and setters are declared here also.

* *public boolean equals (Object o)* – this overrides the initial *equals* method. In this case, it verifies if two objects of type *Person* are equal.
* *public int hashCode ()* – this method will generate an unique hash code for each person using the hash code of the person’s first name multiplied by the hash code of the person’s last name.
* *public void update (Obseravble o, String sort, int oldSum, int currentSum)* – this method overrides the method from the class *Observer*. The holder will receive a different message depending on the *sort*: if it equals to “withdraw” a particular message will be sent and if it equals to the “deposit” another particular message will be sent.

### Serialization Handler

Serialization is the conversion of an object to a series of bytes, so that the object can be easily saved to persistent storage or streamed across a communication link. The byte stream can then be deserialized - converted into a replica of the original object. This class consists in two methods:

* *public void writeObject (Object o)* – this method will write the data into a file. The file is created at the beginning of the method. Moreover, in a try and catch block, the action will be performed.
* *Public Object readObject ()* - this method will read data from the file that was already created. The block of code is also written a try and catch block.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

### Observer and Observable class

These two classes are to support the Design Patters technique. The **Observable** is actually an interface that has only three methods (because only these three were needed for this application; one more reason why I have implemented the interface by myself): *public void addObserver (Observer o), public void deleteObserver (), public void notifyObserver (int oldSum, int currentSum, String sort).* All these methods will be implemented in the **Account** class*.* The **Observer** interface has only one method – *public void update (Observable o, String sort, int oldSum, int currentSum)­ –* that will be implemented in the **Person** class.

### Bank Proc Interface

This is an interface that will be implemented by the **Bank** class. This interface contains only the declaration of the basic actions that the bank will perform: add and remove a person, add and remove an account.

### Bank

In this class, the methods for adding a person, removing a person, adding a new account to an existing holder and removing an existing account.

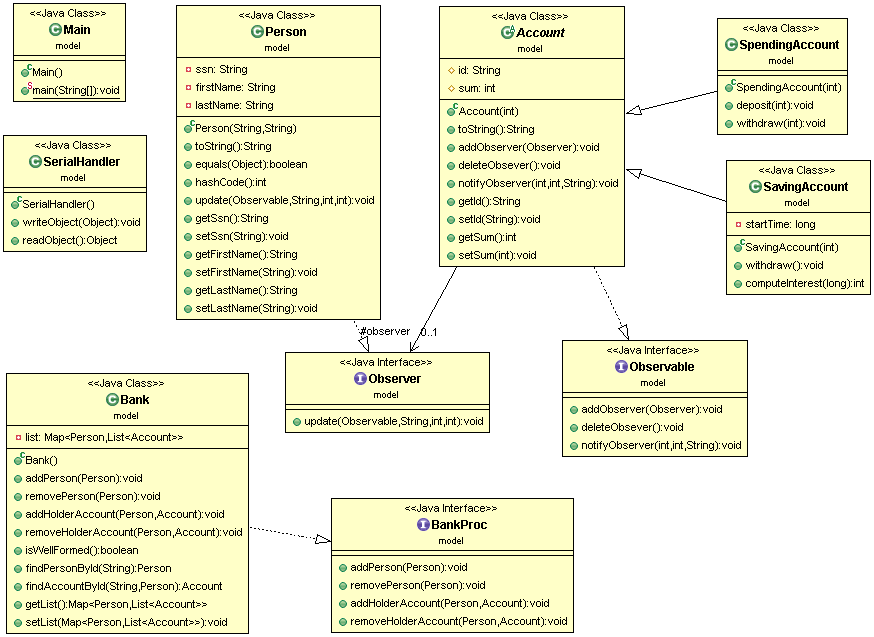
The is wellFormed() method verifies if there is the keys of the hash table is empty and then returns false.

*Public Person findPersonById (String id) ­* - this gets the id of the person and returns the person object.

*Public Account findAccountById (String id, Person p)* – this receives as a parameter the id of the account and a person and returns the account matching those information.

## Graphical User Interface Package

# **Class diagram**

This class diagram was designed using UML, i.e. Unified Modeling Language.

# **Bibliography**

<http://wiki.c2.com/?DesignByContract>

<https://en.wikipedia.org/wiki/Observer_pattern>

<http://stackoverflow.com/questions/447898/what-is-object-serialization>