Bank application

Student: Coca Sergiu

**Group: 30233**

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

Use JAVA/C# API to design and implement an application for the front desk employees of a bank. The application should have two types of users (a regular user represented by the front desk employee and an administrator user) which have to provide a username and a password in order to use the application.

# Functional Requirements

**The regular user** can perform the following operations:

- Add/update/view client information (name, identity card number, personal numerical code, address, etc.).

- Create/update/delete/view client account (account information: identification number, type, amount of money, date of creation).

- Transfer money between accounts.

- Process utilities bills.

**The administrator user** can perform the following operations:

- CRUD on employees’ information.

- Generate reports for a particular period containing the activities performed by an employee.

# Non-functional Requirements

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

All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

2. Use-Case Model

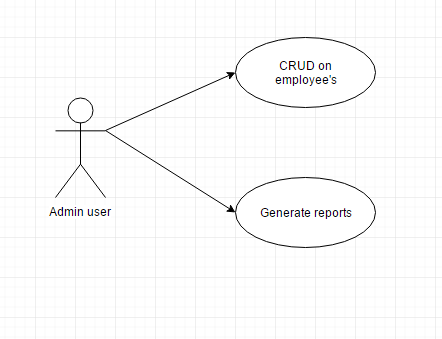
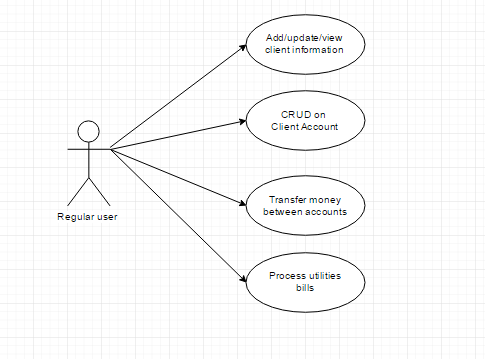
*Use case: Log in*

*Level: user-goal level*

*Primary actor: Regular user*

*Main success scenario:* provide Username and Password and press the Log in button

*Extensions:* the actor doesn’t have an account so he ask the administrator to take care of it

3. System Architectural Design

**3.1 Architectural Pattern Description**

An architectural pattern is a general, reusable solution to a commonly occurring problem in [software architecture](https://en.wikipedia.org/wiki/Software_architecture) within a given context. Architectural patterns are similar to [software design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) but have a broader scope.

The architectural patterns address various issues in [software engineering](https://en.wikipedia.org/wiki/Software_engineering), such as [computer hardware](https://en.wikipedia.org/wiki/Computer_hardware) performance limitations, [high availability](https://en.wikipedia.org/wiki/High_availability) and minimization of a [business risk](https://en.wikipedia.org/wiki/Business_risk).

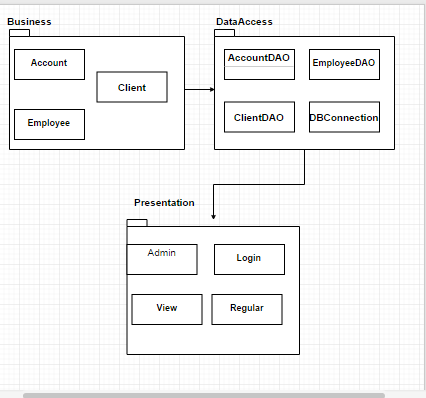
**Layer**

In object-oriented design, a layer is a group of classes that have the same set of link-time module dependencies to other modules. In other words, a layer is a group of reusable components that are reusable in similar circumstances. In programming languages, the layer distinction is often expressed as “import” dependencies between software modules.

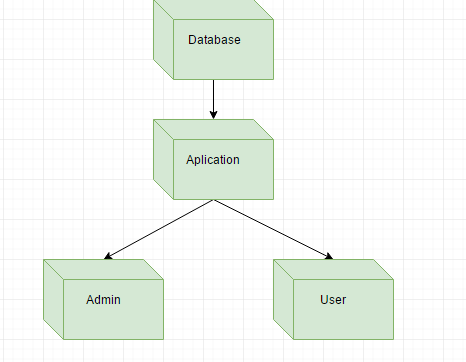
Layers are often arranged in a tree-form hierarchy, with dependency relationship as links between the layers. Dependency relationship between layers are often either inheritance, composition or aggregation relationship, but other kinds of dependencies can also be used.

**3.2 Diagrams**

**Package Diagram**

****

**Deployment diagram**

****

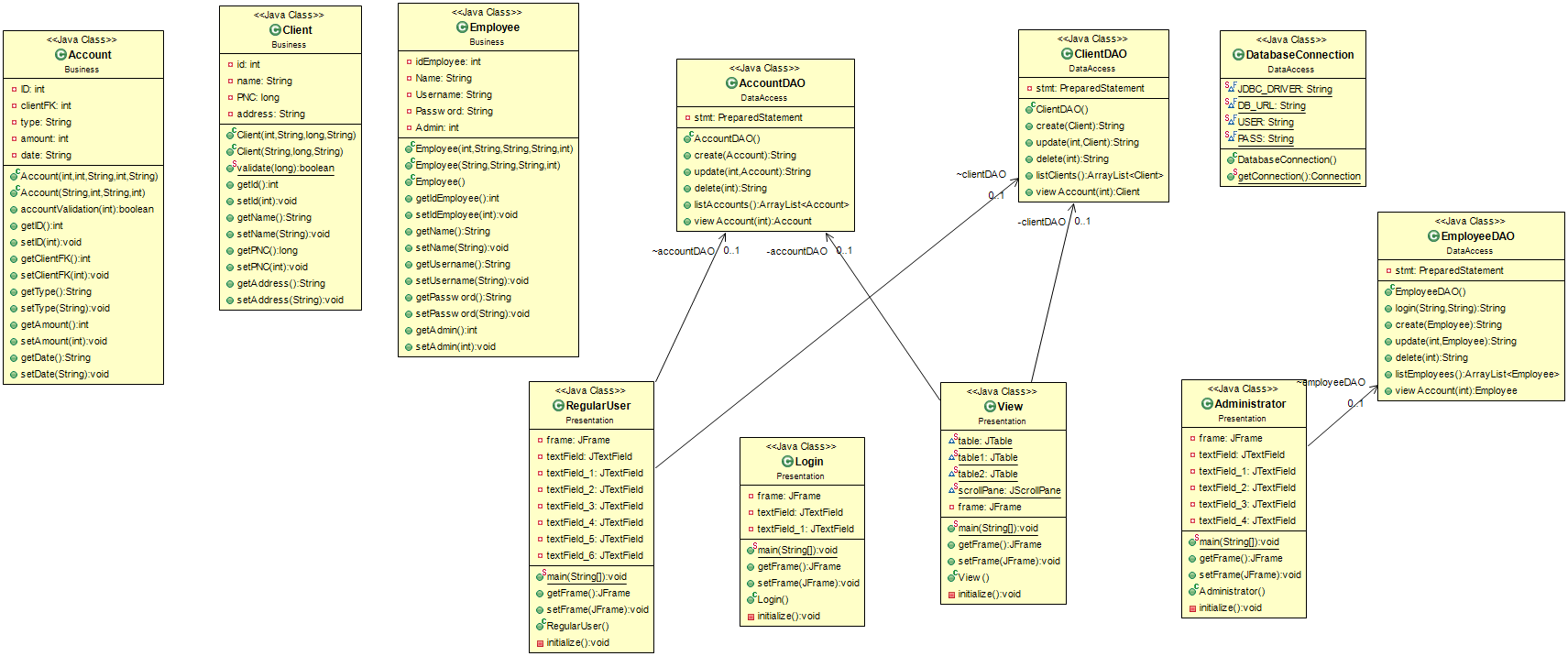
4. Class Design

**4.1 Design Patterns Description**

**Data Mapper** is a layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself. It is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.

**Transaction Script** organizes all this logic primarily as a single procedure, making calls directly to the database or through a thin database wrapper. Each transaction will have its own Transaction Script, although common subtasks can be broken into subprocedures.

**5.2 UML Class Diagram**

****

6. Data Model

In my database I used four tables: Clients, Accounts, Employees, Transaction.

7. System Testing

I implemented two methods for Account validation(amount can’t be less than 0)and Client validation(the PNC should have 13 digits).

8. Bibliography

Martin Fowler et. al, Patterns of Enterprise Application Architecture

http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html