

## Universitatea Tehnică Cluj-Napoca

## Facultatea de Automatică şi Calculatoare

## Secţia: Calculatoare, engleză

## Programming Techniques

## ~ Bank ~

Student: Ban Bogdan Marian

Group id: 30425, second year

Professor: Salomie Ioan

Table of Contents

1. Task Objectives. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 03

2. Problem analysis, modelling, scenarios, use cases. . . . . . . . . . . . . . . . . .03

3. Projection (UML diagrams, data structures, class projections, . . . . . . . .08

Interface, relations, packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .09

Sequence ). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .09

4. Implementation and testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

5. Results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

6. Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

7. Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

1. Task Objectives

**Objective**

Design by Contract Programming Techniques

**Description**

Consider the system of classes in the class diagram below.



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.

1. Problem analysis, modelling, scenarios, use cases

After creating a project that resembles the diagram above , we need to understand the concepts of pre , post conditions , assertion , Serilizable.

The Design by Contract (DBC) software development technique ensures high-quality software by guaranteeing that every component of a system lives up to its expectations. As a developer using DBC, you specify component contracts as part of the component's interface. The contract specifies what that component expects of clients and what clients can expect of it.

Bertrand Meyer developed DBC as part of his Eiffel programming language. Regardless of its origin, DBC is a valuable design technique for all programming languages, including Java.

Central to DBC is the notion of an assertion -- a Boolean expression about the state of a software system. At runtime we evaluate the assertions at specific checkpoints during the system's execution. In a valid software system, all assertions evaluate to true. In other words, if any assertion evaluates to false, we consider the software system invalid or broken.

DBC's central notion somewhat relates to the #assert macro in C and C++ programming language. However DBC takes assertions a zillion levels further.

In DBC, we identify three different kinds of expressions:

Preconditions

Postconditions

Invariants

Let's examine each in more detail.

Preconditions

Preconditions specify conditions that must hold before a method can execute. As such, they are evaluated just before a method executes. Preconditions involve the system state and the arguments passed into the method.

Preconditions specify obligations that a client of a software component must meet before it may invoke a particular method of the component. If a precondition fails, a bug is in a software component's client.

Postconditions

In contrast, postconditions specify conditions that must hold after a method completes. Consequently, postconditions are executed after a method completes. Postconditions involve the old system state, the new system state, the method arguments, and the method's return value.

Postconditions specify guarantees that a software component makes to its clients. If a postcondition is violated, the software component has a bug.

Invariants

An invariant specifies a condition that must hold anytime a client could invoke an object's method. Invariants are defined as part of a class definition. In practice, invariants are evaluated anytime before and after a method on any class instance executes. A violation of an invariant may indicate a bug in either the client or the software component.

Assertions, inheritance, and interfaces

All assertions specified for a class and its methods apply to all subclasses as well. You can also specify assertions for interfaces. As such, all assertions of an interface must hold for all classes that implement the interface.

iContract -- DBC with Java

So far, we have talked about DBC in general. You probably have some idea by now what I am talking about, but if you are new to DBC, things might still be a bit foggy.

In this section, things will become more concrete. iContract, developed by Reto Kamer, adds constructs to Java that allow you to specify the DBC assertions we talked about earlier.

iContract basics

iContract is a preprocessor for Java. To use it, you first process your Java code with iContract, producing a set of decorated Java files. Then you compile the decorated Java code as usual with the Java compiler.

All iContract directives in Java code reside in class and method comments, just like Javadoc directives. In this way, iContract ensures complete backwards-compatibility with existing Java code, and you can always directly compile your Java code without the iContract assertions.

In a typical program lifecycle, you would move your system from a development environment into a test environment, then into a production environment. In the development environment, you would instrument your code with iContract assertions and run it. That way you can catch newly introduced bugs early on. In the test environment you may still want to keep the bulk of the assertions enabled, but you should take them out of performance-critical classes. Sometimes it even makes sense to keep some assertions enabled in a production environment, but only in classes that definitely are in no way critical to your system's performance. iContract allows you to explicitly select the classes that you want to instrument with assertions.

Serialization

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.

Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

The ObjectOutputStream class contains many write methods for writing various data types, but one method in particular stands out:

public final void writeObject(Object x) throws IOException

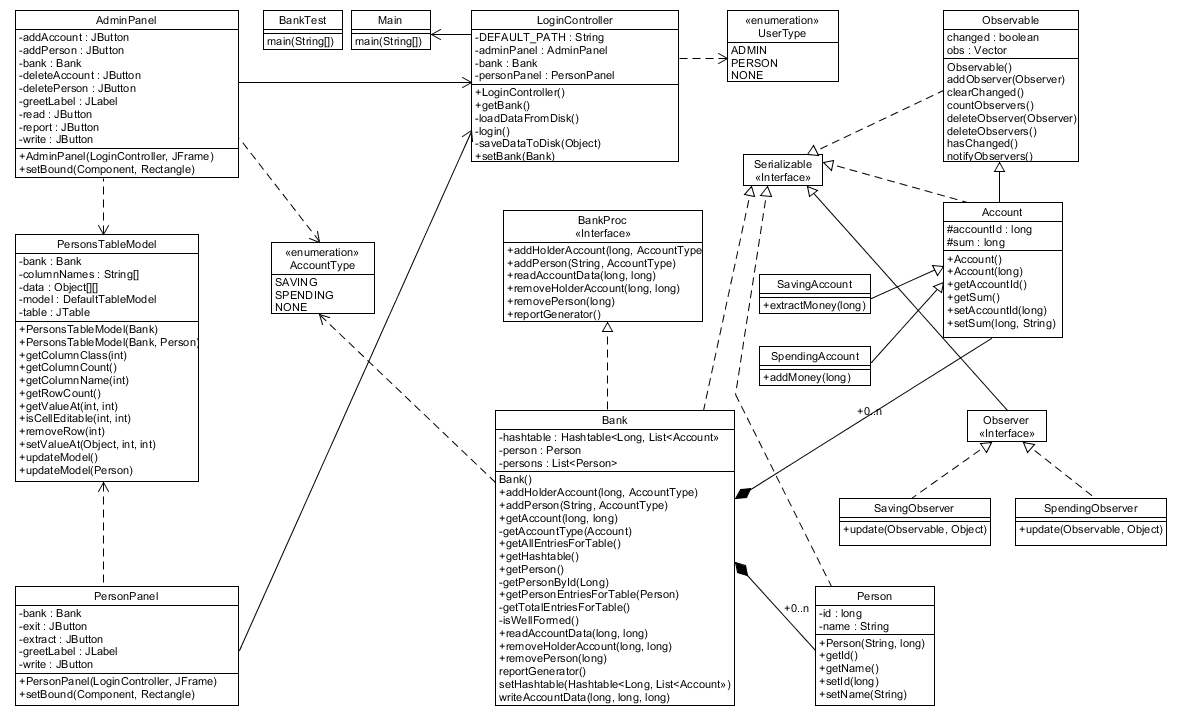
The above method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains the following method for deserializing an object:

public final Object readObject() throws IOException, ClassNotFoundException

This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.

3.Projection

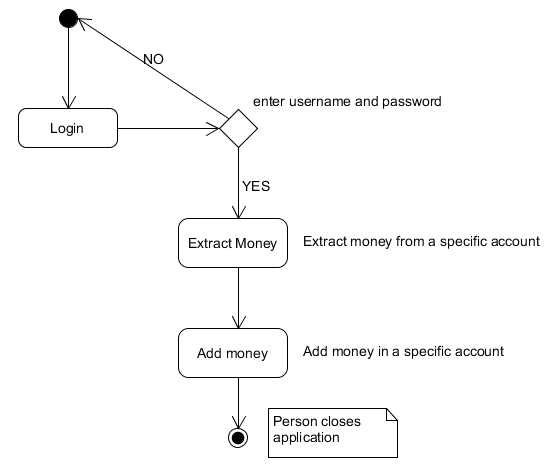
UML Class Diagram



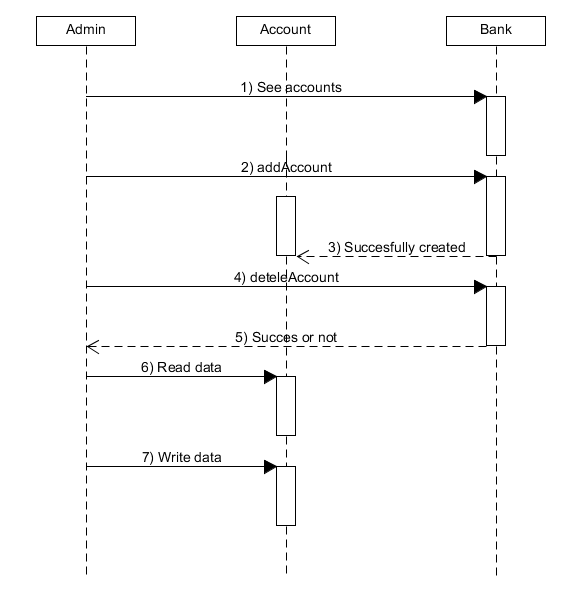
Use case Diagram

Admin

Activity diagram



Sequence diagram



A sequence diagram of the objects Admin,Account and Bank.

Packages

The classes chosen for this project are specific and each one of them handles an important part of the program.

1. Implementation and testing

As seen from the diagrams the implementation took several interfaces and manipulating the objects in a certain way,

5. Results

The results can be seen from the application. As for this we obtained an easy application that creates a bank with accounts.

6. Conclusions

Very good opportunity to learn about Design by contract.

7. Bibliography

http://www.javaworld.com/article/2074956/learn-java/icontract--design-by-contract-in-java.html

http://www.tutorialspoint.com/java/java\_serialization.htm