Innopolis University | Software Architecture | Spring 2016

A UML Class Diagram for a Banking Application

Néstor Cataño | **Homework – Week 05** | Due: 22/Feb/2016

Rudimentary Description

LLoyds Bank Group has opened a programming contest for Innopolis (INO) students to design and implement an Object-Oriented (O-O) Java application for the management of bank accounts for professors and students. LLoyds offers two types of bank accounts, *saving* and *checking* accounts. Both types of accounts allow users to deposit and withdraw money.

Saving Accounts. Saving accounts for professors are created with an initial balance of 200 Euros. Saving accounts for students are created with an initial balance of 100 Euros. Professors and students have an overdraft protection of 200 Euros and 100 Euros respectively. If a debit operation might result in an overdraft beyond that protection, then the bank will not permit the operation. Users are not permitted to debit from an overdrawn saving or checking account. A fixed daily interest rate for saving accounts is set when they're opened.

Checking Accounts. Checking accounts are created with an initial balance of 0 Euros for both professors and students. Checking accounts do not enjoy an overdraft protection. A user can credit her as many times as she wishes. A user cannot debit her account more than two consecutive times. Thus, to debit a checking account a third consecutive time, the user needs to credit her account first. Checking accounts do not earn any interest.

Currency. LLoyds bank group manages bank accounts in Euros and Dollars only.

Logged Accounts. Saving and Checking Accounts keep a log of the transactions made by users. Logs keep information about the type of the transaction and the amount of money transferred to or from bank accounts.

What to Hand In?

1. (**10 marks**) A class diagram capturing the software requirements of the Lloyds banking application. It should include classes, attributes and methods. You must use StarUML to write your class diagram. Instructions to download and use StarUML can be reached at `http://staruml.io/`.
2. OCL specifications for the constraints below. The OCL specifications should be written as comments to your class diagram.
   (a) (**5 marks**) Invariant for class `CheckingAccount`. The balance of a checking account is always non-negative since checking accounts don't have an overdraft protection.
   (b) (**5 marks**) Pre- and Post-conditions for `CheckingAccount::debit(amount: double)`. A user cannot debit her own checking account more than two consecutive times.

(c) (**5 marks**) Pre- and Post-conditions for `CheckingAccount::credit(amount: double)`. Crediting a checking account will reset the number of consecutive debits associated to that account.

(d) (**5 marks**) Pre- and Post-conditions for `SavingAccount::debit(amount: double)`. If a debit operation might result in an overdraft beyond the account protection, then the bank will not permit the operation. Users are not permitted to debit from an overdrawn saving account.

(e) (**5 marks**) Invariant for class `Customer`. A customer has exactly two different bank accounts.

3. (**10 marks**) An implementation of the class diagram that enforces the OCL constraints that your wrote. Your implementation can be written in Java or Eiffel. If your implementation is written in Java, then the OCL specifications must be written as Java comments. For instance, you could write a method pre-condition as `/** <b> pre: xxx </b> */`, and a method postcondition as `/** <b> pos: xxx </b> */`. Method pre- and post-conditions are written immediately before method declarations. Invariants should be written as class comments, e.g., `/** <b> inv: xxx </b> */`, placed next to the Java class attribute declarations. If your implementation is written in Eiffel, the OCL specifications are already part of your Eiffel implementation.