## Final Practical Exam

# 420-D05-SU

# Programming in a graphical environment - C#.Net

Duration: 4h00

Name: Date:

Group:

**Instructions:**

**Documents are permitted.**

**Please clearly write your name on the exam.**

**Any cheating or plagiarism will result in a grade of zero (0%).**

**A project that does not compile will receive a grade of zero (0%).**

**No communication is allowed with anyone.**

**The use of cellphones is prohibited.**

**Push your work in BitBucket as a regular exercise. Please, ensure to put your project in a Visual Studio solution folder.**

# Bank account manager - (16 points)

You are required to create an application for managing bank accounts.

This must be a multi-layer application (Model, Data Access Layer, Business Logic Layer, and View).

The view is composed of two windows. The first window allows you to log in with an account number and a password. The second allows you to manage the account.

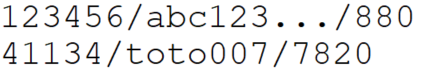
Each time the client logs out, the file is updated with the new balance of the account.

1. The accounts are stored in a text file (accounts.txt). (**0.5 points**)

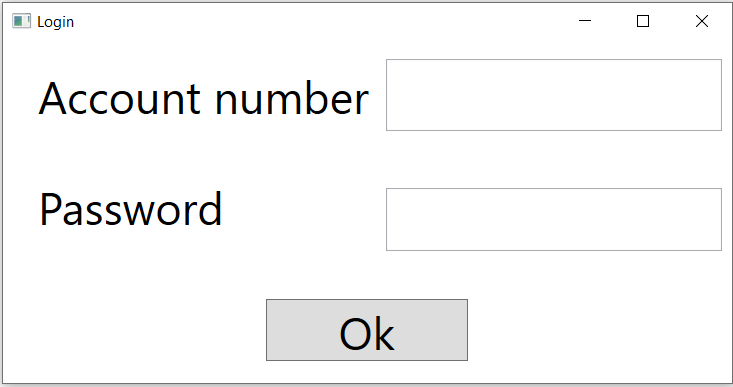
Each line represents an account. The format of a line is as follow:



The following file represents two bank accounts. Account numbers are 123456 and 41134, passwords are abc123… and toto007. Their balances are 880$ and 7820$.

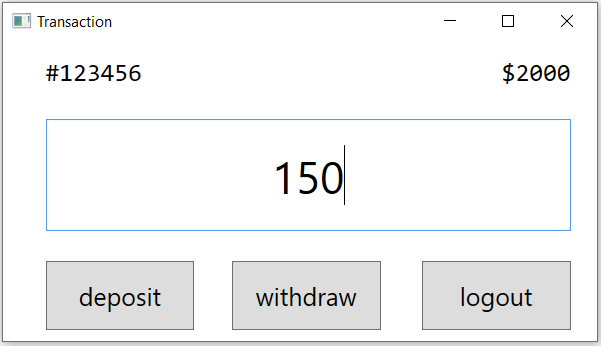


1. Create a multi-layer application (Model, DAL, BLL and View). (**1 point**)
   1. Set the correct references between the 4 projects (*1 point*)
2. Model layer (**3.5 points**)
   1. Create an **Account** class containing the 3 information of an account. (*1 point*)
   2. An event (or an instance of delegate) to be able to notify some listeners when the balance changes. (*0.5 points*)
   3. Two methods to allow deposit and withdraw money. (*1 point*)
   4. When the balance changed, the account notifies the listeners. (*1 point*)
3. BLL (**2.5 points**)
   1. Create a static class **AccountManager**. (*0.5 points*)
   2. One method to look for an account given its number and the password. (*0.5 point*)
   3. One method to save a given account. (*0.5 point*)
   4. Two methods to allow deposit and withdraw money. (*1 point*)
4. DAL (**2.5points**)
   1. Create a static class **AccountService** for manipulating the text file. (*0.5 points*)
   2. One method to retrieve an account given its number and the password. (*1 point*)
   3. One method to save a given account. (*1 point*)
5. View layer: class **LoginWindow** (**3 points**)
   1. At application launch, the following login window should appear. (*1 point*)



* 1. When the **Ok** button is clicked, the account number and the password are searched in the accounts text file. (*0.5 points*)
  2. If the account number is found and the password is correct, a transaction window for managing this account would appear and the login window would close. (*1 point*)
  3. Otherwise, a MessageBox saying “Inexisting account or wrong password” appears. (*0.5 points*)

1. View layer: class **TransactionWindow** (**3 points**)



Current balance

Bank account number

Specified amount for the transaction

* 1. When the **deposit/withdraw** button is clicked, the corresponding transaction is done on the account. In this window, current balance is updated via Observer pattern (see below). (*2 points*)
  2. When the **logout** button is clicked, the new balance of this account is saved. Then, this transaction window will close and a new login window should appear. (*1 point*)

# Observer pattern

In this application, the instance of **TransactionWindow** is listening to the instance of **Account**.

When the account changes its balance, the view is notified.

In its handler, the view shows the new balance of the current account.

To achieve this, you can either create your own delegate type or use one of the .NET framework.

# Reminder

Only the DAL contains file manipulation statements.

Only the View contains user application input or output.

The view cannot directly talk to the DAL, it needs to pass through the BLL.

# Bonus (1 point)

* Check for bad format on input data. (*0.5 points*)
* Warn user if he tries to withdraw more than the current balance on the account. (*0.5 points*)