SE361: OBJECT ORIENTED DESIGN

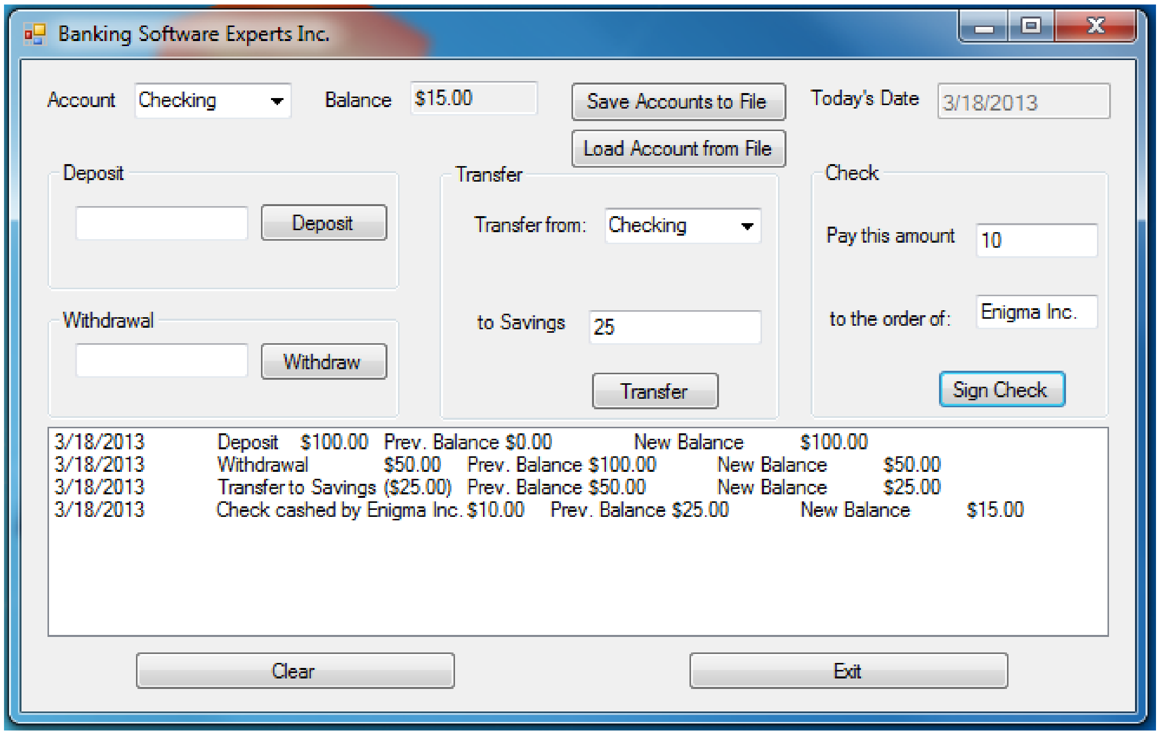
# Assignment

## Banking Transactions Application

For this assignment, you will develop a Windows Application to maintain a person’s Savings and Checking accounts.

The application should keep track of and display the balances in both savings and checking accounts, and maintain a list of transactions (deposits, withdrawals, fund transfers, and check clearings) separately for each account. The two lists of transactions should be stored in sequential text files so that they will persist between program sessions. The user should also be able to retrieve and display the contents of the text files in the Transactions list box.

Consider the GUI form shown below. The two dropdown combo boxes should each contain the items *Checking* and *Savings*. Each of the four group boxes corresponds to a type of transaction. When *Savings* is selected in the Account combo box, the Check group box should disappear. The user makes a transaction by typing data into the text boxes of a group box and pressing the button. The items appearing in the Transactions list box should correspond to the type of account that has been selected. The caption of the second label in the Transfer group box should toggle between "to Checking" and "to Savings" depending on the item selected in the "Transfer from" combo box. If a transaction cannot be carried out, a message such as "Insufficient Funds" should be displayed. Finally, separate button needs to be created on the GUI form so that the user can retrieve and display the contents of the text files in the Transactions list box.



Account: Checking
Balance: $15.00
Today's Date: 3/18/2013
Transfer from: Checking
to Savings: 25
Pay this amount: 10
to the order of: Enigma Inc.

```
3/18/2013      Deposit   $100.00  Prev. Balance $0.00          New Balance      $100.00
3/18/2013      Withdrawal         $50.00   Prev. Balance $100.00       New Balance      $50.00
3/18/2013      Transfer to Savings ($25.00)  Prev. Balance $50.00        New Balance      $25.00
3/18/2013      Check cashed by Enigma Inc. $10.00    Prev. Balance $25.00        New Balance      $15.00
```

**OO DESIGN**: Your OO design should include two entity classes: Transaction and Account. In addition, all necessary boundary and control classes need to be created. The class Transaction should have properties such as transaction name, amount, date and whether it is a credit (deposit) or debit (withdrawal/check). The class Account, which will have both a checking account and a savings account as instances, should use a collection of transaction objects. In addition, it should have properties such as name (Checking or Savings) and balance. It should have methods to carry out a transaction (if possible), to display the list of transactions, and to save/load the set of transactions into or from a sequential text file. (**Hint:** a simple way to display the date on the Window form is to use the statement: `lblDate.Text=Today`)

**UNIT & INTEGRATION TESTING**: For each class of your application create a separate test driver class to check for incorrect data entered (e.g. nonnumeric answers, attempt to check the answer on empty number values, etc.) and print the appropriate error messages. After each class has been tested separately (i.e. unit testing) you will integrate and test all your classes together (i.e. integration testing) by creating the final main driver form class as shown in the GUI above.

**DELIVERABLES**: This assignment will be completed into two separate stages: completion of the OO design first, followed by the implementation. On the first due-date posted on our website, you will present and defend your OOD to your peers in class. By the second due date, you will demonstrate your implementation in class. Also, upload a zipped file with all your work on our website including the following:

1) Your completed program with meaningful comments in the code
2) A detailed UML class diagram that includes all classes with their member variables (data) and methods (operations). Also, show all relationships between classes (e.g. association, aggregation, composition, inheritance) as well as their multiplicity.
3) A complete TOE (Task, Object, Event) table that describes the GUI design of your application.
4) Interaction diagrams for three use-cases: “Deposit”, “Withdraw” and “Transfer”
5) A copy of the program’s output (use the *PrintScreen* button on your keyboard to save a copy of the screen) showing different execution scenarios. **It is important to demonstrate all possible normal (i.e. valid input data) use case scenarios and all possible exceptional (i.e. invalid input data) scenarios.**