**CST8253 Web Programming II**

Lab 5

# Objective

1. Apply Object Oriented Analysis and Design, And Object Oriented Programming

# Due Date

See Canvas posting for the due date. To earn 5 points, you are required:

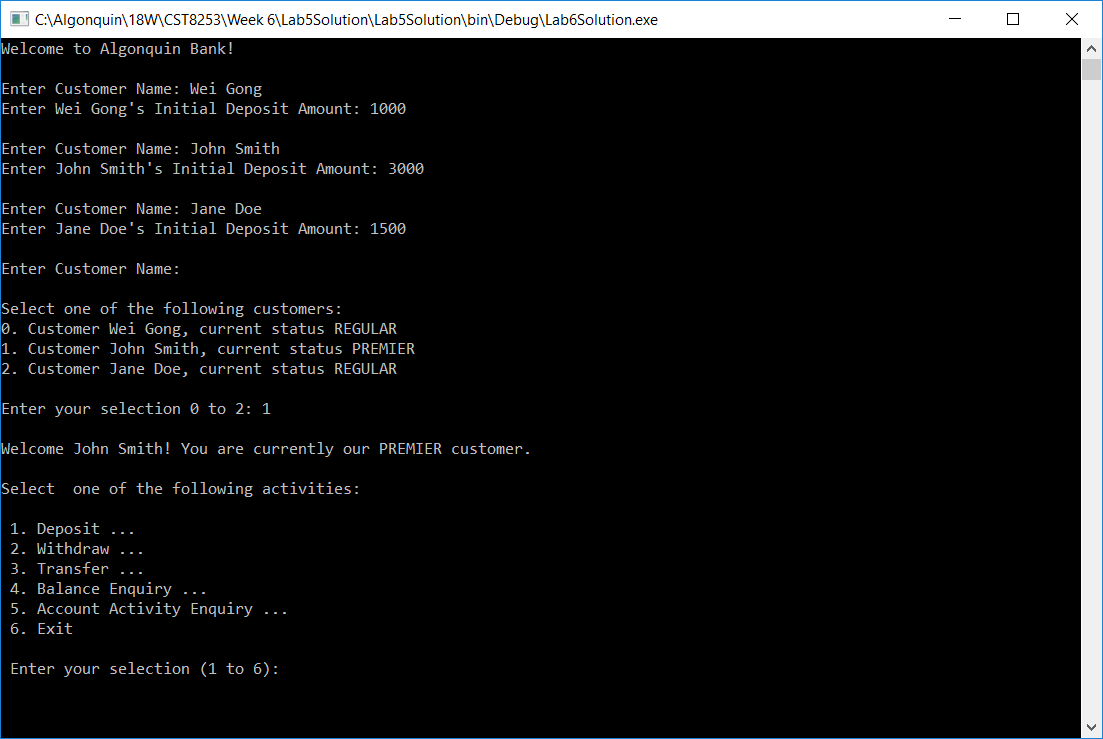
1. Complete the lab as required.
2. Zip the project folder (the folder containing all project files) and the solution file (**.sln** file). Submit the zipped file to Canvas
3. Demo your lab work during the following week’s lab session.

# Requirements

Create a C# console application for a banking system. The application should have the following functionality:

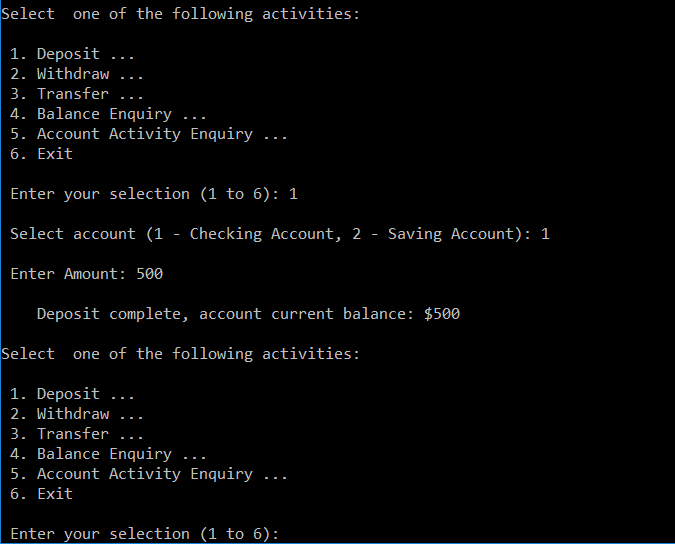
On start, the application will run according to the following steps:

1. Prompts the user to a name and an initial deposit amount.
2. Creates a Customer with the given name.
   1. The customer will have two accounts, a Checking Account with a zero balance and a Saving Account with a balance of the initial deposit amount.
   2. If his/her Saving Account’s balance is below $2000.0, the customer is in REGULAR status.
   3. If his/her Saving Account’s balance is equal to or more than $2000.0, the customer is in PRIMIER status.
3. Repeats step 1 – 2 to create another customer until the user enters blank at step 1.
4. Lists all the customers and their status if the user enters blank when prompted for customer’s name
5. Prompts the user to select one customer to perform banking functions.

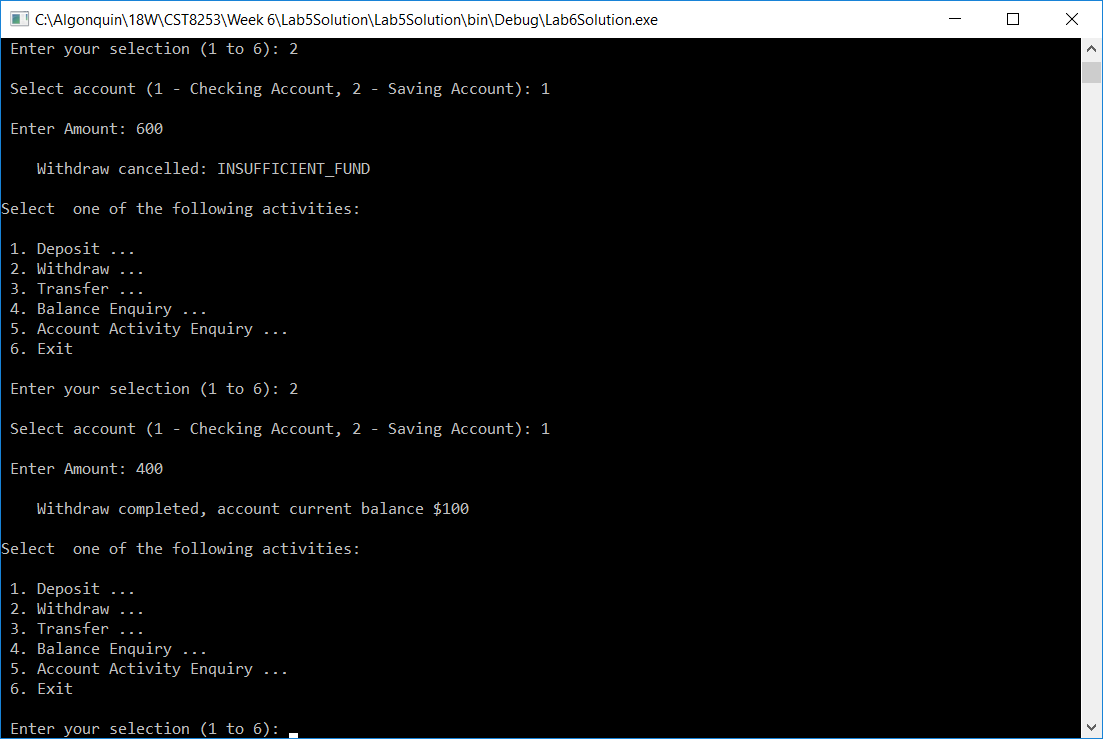


1. After selecting a customer, the user can perform the following activities to his/her two accounts:

* Deposit – the user can select an account and specify an amount to deposit into the selected account.
  1. If the deposit to the Saving Account resulted in a balance equal to or greater than $2000, the customer becomes a PRIMIER customer.

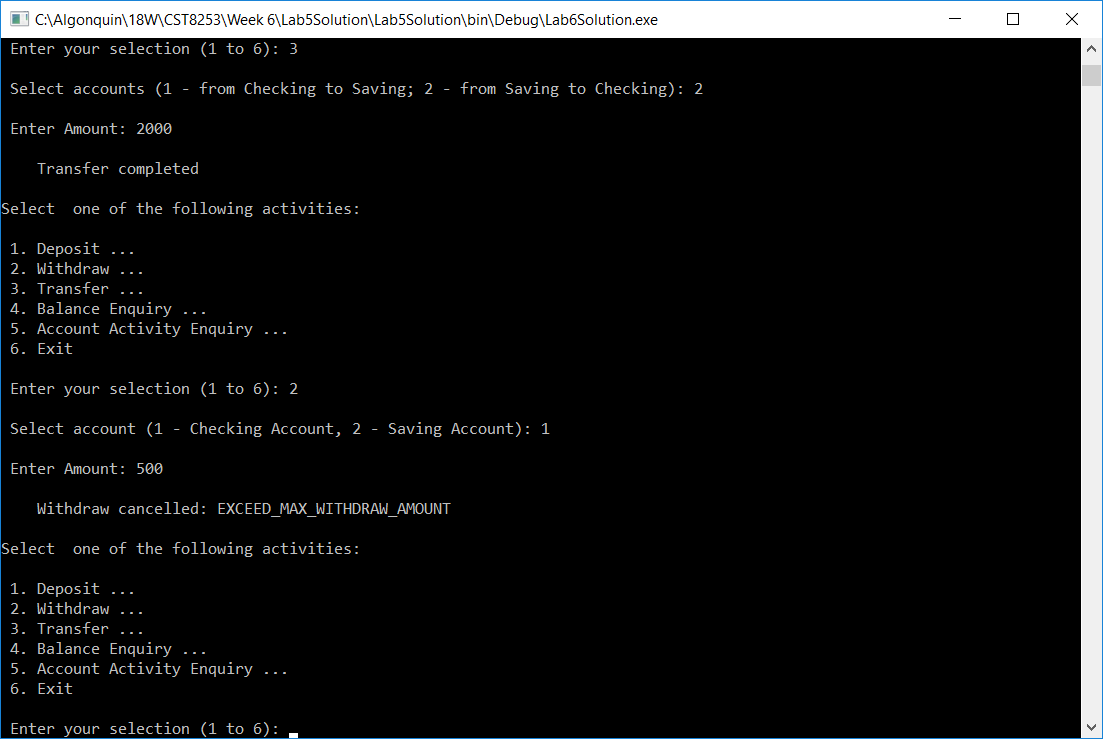


1. Withdraw – the user can select an account and specify an amount to withdraw from the selected account.
   1. The withdraw amount can never be greater than the current balance of the account.
   2. For a REGULAR customer, he/she can only withdraw $300 maximum from his/her Checking account even if the current balance is greater than $300. There is no such limit for PRIMIER customers.
   3. For REGULAR customers, each withdraw from the Saving Account will incur a $10 penalty. There is no penalty to PRIMIER customer
   4. If the withdraw from the Saving Account resulted in a balance below $2000, the customer becomes a REGULAR customer.



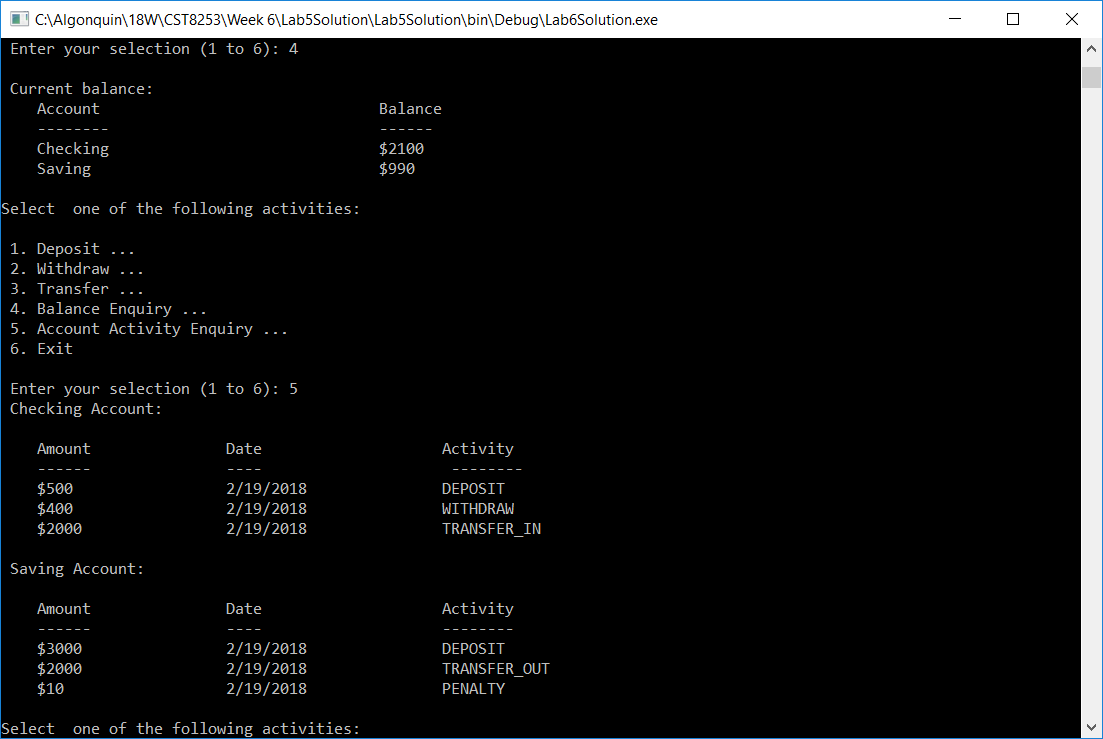
Note: In above example, the customer is in PRIMIER status, so he has no limit and can withdraw $400 from his checking account.

1. Transfer – the user can transfer a specified amount from one account (withdraw) to another account (deposit).
2. Transfer-out amount can never be greater than the current balance.
3. When transfer-out from Checking account, the maximum withdraw limit does not apply.
4. If a transfer-out from the Saving Account resulted in a balance below $2000, the customer becomes a REGULAR customer.
5. If a transfer-in to Account resulted in a balance equal to or greater than $2000, the customer becomes a PRIMIER customer.

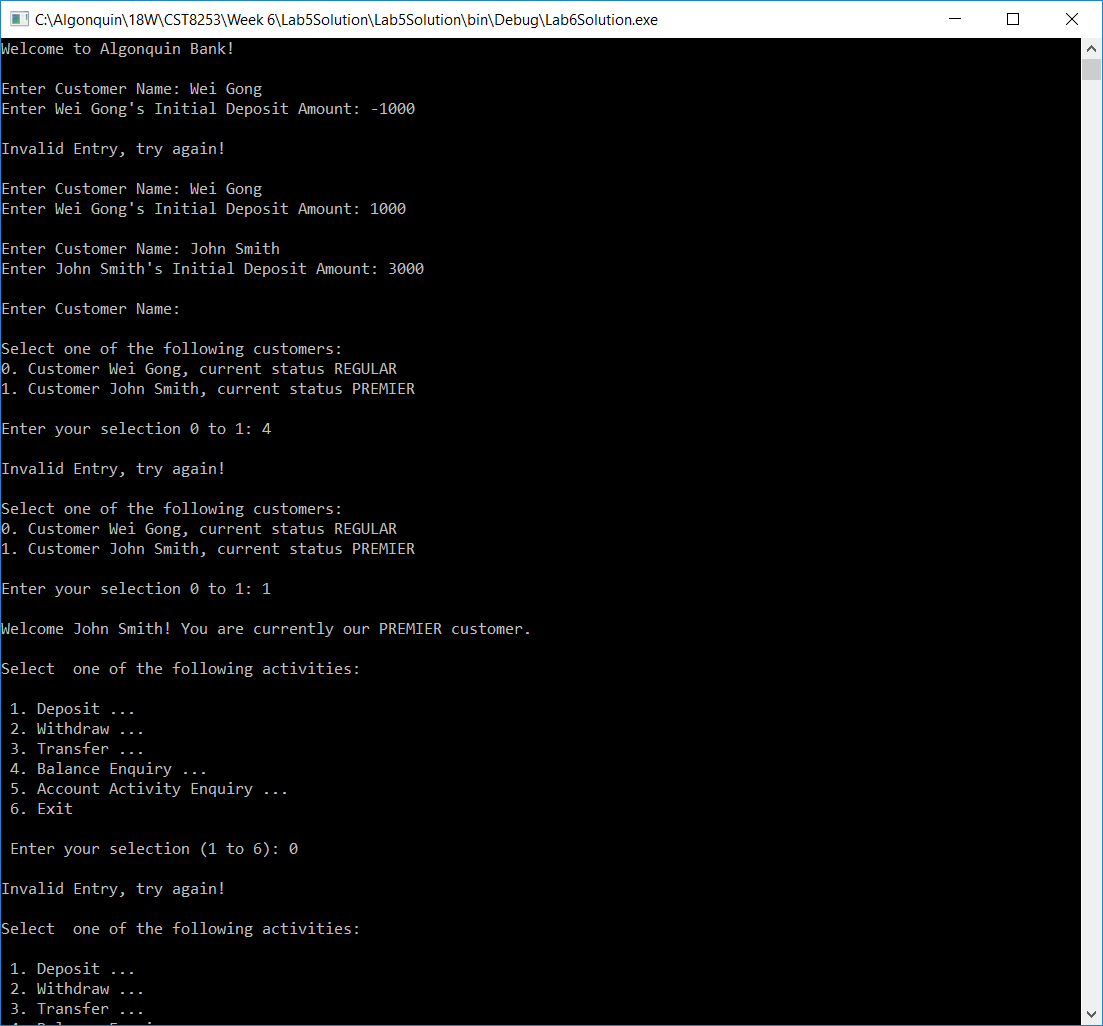


Note: In above example, after transferring $2000 from the Saving Account to the Checking Account, the customer becomes a REGULAR customer. The $500 withdraw from the Checking Account is therefore cancelled even through the Checking Account has a balance of $2100.

1. Balance Enquiry – list the current balance of the both accounts.
2. Activity Enquiry – list the both accounts’ transaction activities, transaction dates and transaction amounts.

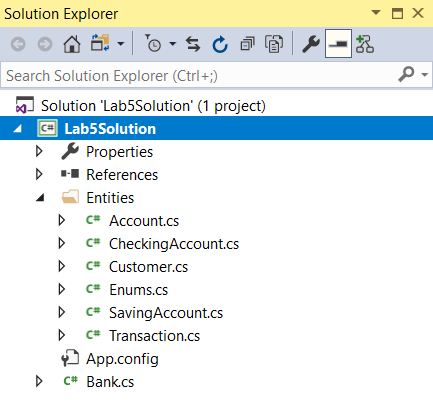


1. At any point, if the user entered an invalid value, the application will prompt the user for try again.



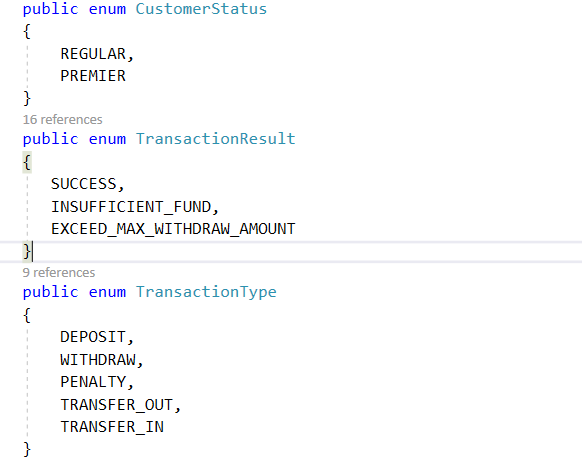
# Implementation Notes

1. Your Lab 5 Solution should have the following structure:



To better organize files in the project, you can create folders by right click on the project and select “**Add > New Folder**”.

1. You should create a C# file named **Enums.cs** which contains the definitions of following enumerations:



1. You should create a **Customer** class at minimum having the following public properties. You may define more properties.

* Name – string type, the name of the customer.
* Status – **CustomerStatus** type, the status of the customer
* Checking – **CheckingAccount** type, the customer’s checking account
* Saving – **SavingAccount** type, the customer’s saving account.

1. The **customer** class should have a constructor taking one string type parameter as customer’s name.
2. You should create an **Account** class at minimum having the following properties:

* Owner – Customer type; the owner of this account.
* Balance – double type; the balance of the account
* TransactionHistory – List of **Transaction** type; maintain all transactions performed to this account.

1. The **Account** class should have a constructor take one **Customer** type parameter to initialize its Owner property.
2. At minimum, the Account class should have the following methods:

* Deposit – Take a Transaction type parameter to increase the balance by the transaction amount. Upon completion successfully, add the transaction to the TransactionHistory.
* Withdraw – Take a Transaction type parameter to decrease the balance by the transaction amount when the transaction amount is equal to or less than the account’s balance. Upon completion successfully, add the transaction to the TransactionHistory.

If the transaction amount is greater than the account’s balance, the transaction should be cancelled and the method returns

TransactionResult.INSUFFICIENT\_FUND

1. You should create a **CheckingAccount** class inheriting from the **Account** class.
2. At minimum, the **CheckingAccount** should have a static property:

* MaxWithdrawAmount – double type, initialize to 300.0

1. **CheckingAccount** should have a constructor take one **Customer** type parameter to pass to its base class’ constructor.
2. The **CheckingAccount** should override its base class’ Withdraw method to check for REGULAR customers if the withdraw amount is greater than the MaxWithdrawAmount.
3. You should create a **SavingAccount** class inheriting from the **Account** class
4. At minimum, the **SavingAccount** should have following static property:

* PrimierAmount – double type, initialize to 2000.0
* WithdrawPenaltyAmount – double type, initialize to 10.0

1. **SavingAccount** should have a constructor take one **Customer** type parameter to pass to its base class’ constructor.
2. The **SavingAccount** should override its base class’ Deposit and Withdraw method to update the customer’s status based on the account’s updated balance.

The Withdraw should also apply the penalty, if the customer’s status is REGULAR

1. You should create a **Transaction** class having the following properties:

* Amount – double type, withdraw or deposit amount
* Type – TransactionType type, the type of the transaction
* TransactionDate – DateTime type, the date and the time of the transaction.

1. The **Transaction** class should have a constructor taking a double type parameter to initialize its Amount property and a TransctionType type parameter to initialize its Type property. Within the constractor the TransactionDate property is initialize to current date and time: **DateTime.Now**

Select one of the following activities

If I enter, it takes me to transfer

PENALTY