

CIS605 Test 3

Fall 2015

Name: _____

____ / 56 = ____ / 100

Instructions

This test will cover Chapters 5 & 10 – on the topics of Selection and Custom Events. You, of course, still have to know how to do the other things you've been learning all semester, but this test will *focus* on these topics.

This is a take-home test, with a limited amount of time allowed. It is open-book, open-note, open-Internet, closed-neighbor. You may not obtain any form of help from another human in answering the questions on this test. Only you, the person who is receiving a grade for the test and the course may complete the work for the test.

You may work on this any time between 10:45am on Thursday, Nov 19, 2015, and just before midnight (11:59pm) on Tuesday, Nov 24, 2015, all Mountain Time. You may take up to 3 hours of contiguous time to work on the test between these two times. In other words, once you open and start the test the clock starts running and the 3 hours start counting down. You cannot work on the test for a little while, go away (and have the clock stop counting down), and come back and continue. Once you begin, you have a maximum of 3 hours from that moment. If you use more than 3 hours to get everything completed and submitted, you will not receive credit for the test. However, you must have everything completed and submitted in Canvas before the specified final due date/time. Make sure that you submit well before this closing time so that any clock differences on the Canvas system and your clock do not cause you grief and keep you from being able to make your submission. All parts of the test will automatically disappear from view at the closing time, and you will not be able to make a submission after that. Once you "submit" it you cannot make any additional changes or make additional submissions. Thus, make sure you are really done before choosing to "submit".

By submitting your answers for a grade you agree that you have done the following:

1. You, the person receiving a grade for this course, are the only one who has worked on and submitted your answers for, this test.
2. During the test you have neither received any help from, nor given any help to, other people regarding the test.
3. Before the test you have not communicated regarding the test with anyone who has already taken the test, and after the test you have not communicated regarding the test with anyone who has not yet taken the test.
4. You have not used any unauthorized resources during the test.

For this test you will have solely programming questions. Follow all standard naming conventions prescribed for this semester, use the provided Class Template, and put code in proper Region(s). Use the standard approaches and techniques that have been taught and illustrated in lecture and assignments.

You will submit everything for this test via Canvas: 1) your answer to the Honor Code question, 2) your answers to the conceptual questions, 3) your attached ZIP file with your complete Visual Studio / VB.Net Solution / Project. The VB zip file will be attached to, and submitted as part of, the Canvas Quiz (test).

Questions/Tasks

Honor Pledge

This must be completed in order to be eligible for any points on the test.

- I have not given, received, or used any unauthorized assistance, nor will I give, receive, or use any unauthorized assistance. (You must answer True to this in order to receive any credit for this test.)

Programming Questions (56 pts)

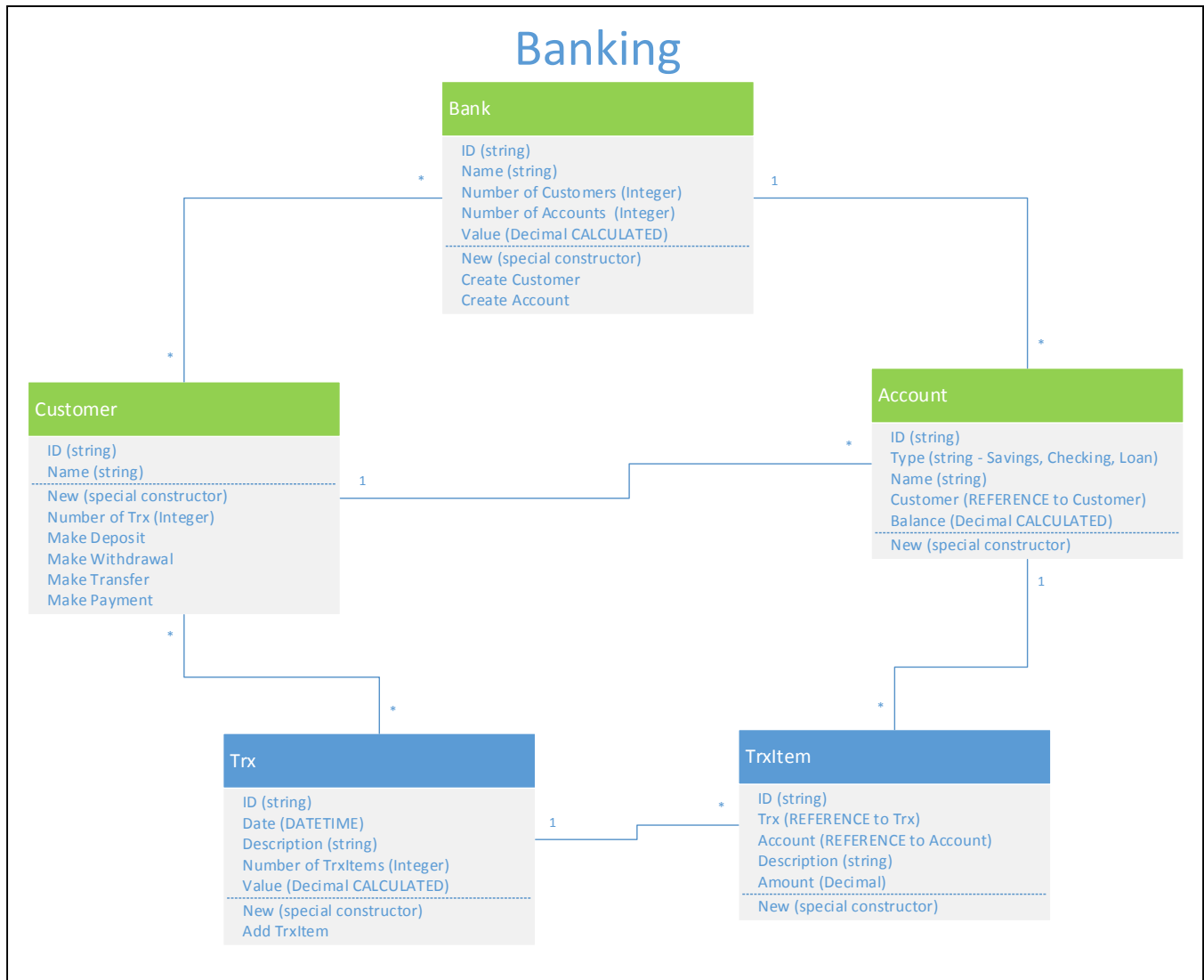
In your VB.Net Project for this test, write VB.Net code that answers each of the following questions. **The Class Template must be used, and our standard naming conventions followed, as well as our standard approach to creating and using custom events. You do not have to write standard comments in the code as would be required for a normal programming assignment.**

Attach your ZIPped Solution subdirectory tree to this question when you are finished.

The following programming questions all reference the Banking domain, as summarized in the UML diagram provided below. You will build on what we did in the previous test. Before starting this test you should have prepared (per the provided instructions) by getting your copy of the previous test ready for use in this test. You will change and build on that code. You will create VB.Net business logic classes for these, and implement the indicated attributes and behaviors (functionality) for each class. You will then write code in the main Form to demonstrate through some hard-coded tests that your business logic classes work as intended and that you know how to interact with the business logic methods from the main form. Your run should produce the exact same output as my sample below shows.

Class Diagram

(For your convenience, below is the UML Class Diagram that was used in Test2 and will continue to be used in Test3.)



Test Data

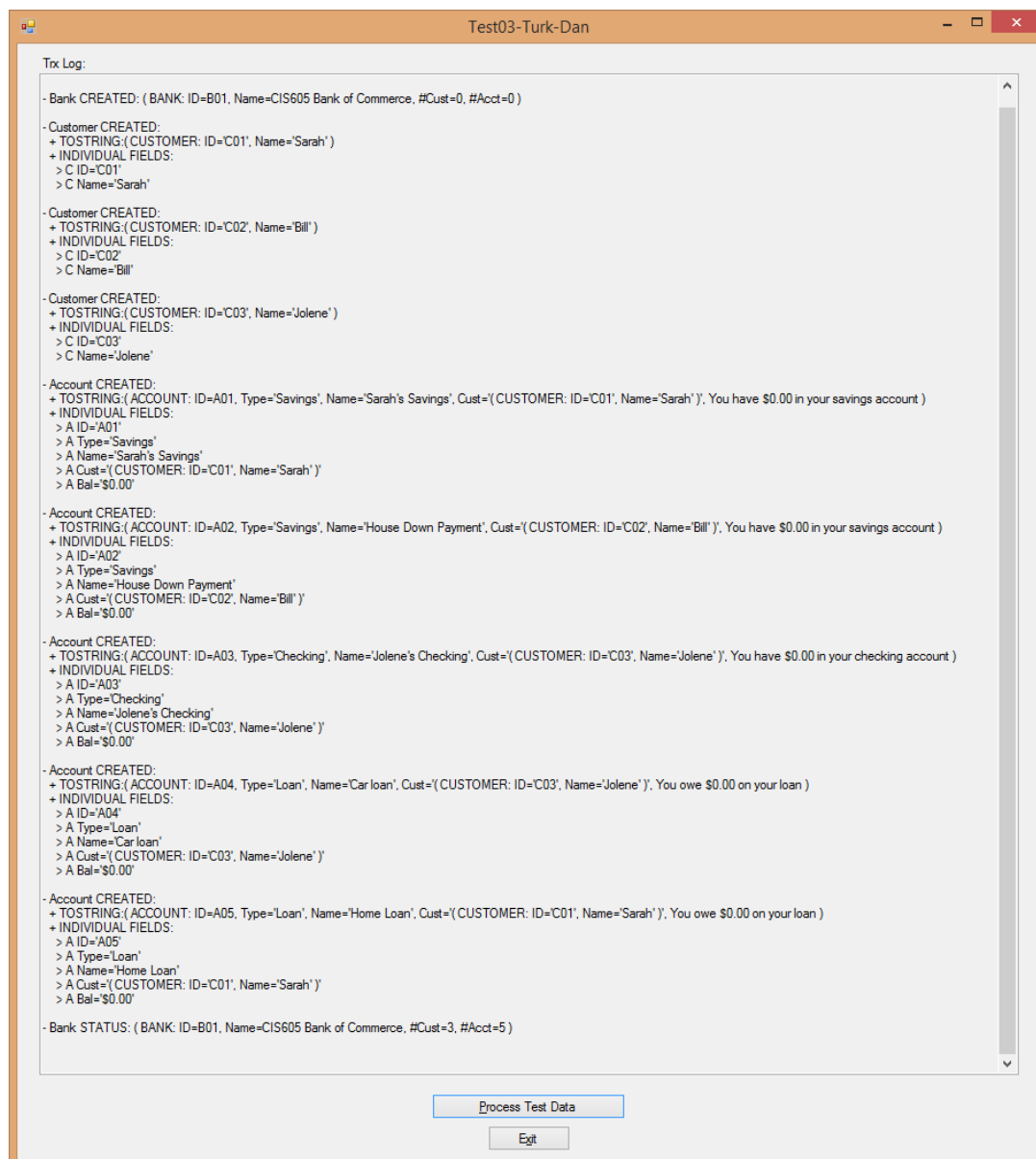
(Use this where you hard-code the test data.)

1. Bank
 - a. B01, CIS605 Bank of Commerce
2. Three Customers
 - a. C01, Sarah
 - b. C02, Bill

- c. C03, Jolene
- 3. Five Accounts
 - a. A01, Savings, "Sarah's Savings", <reference to Customer C01>
 - b. A02, Savings, "House Down Payment", <reference to Customer C02>
 - c. A03, Checking, "Jolene's Checking", <reference to Customer C03>
 - d. A04, Loan, "Car loan", <reference to Customer C03>
 - e. A05, Loan, "Home loan", <reference to Customer C01>

Output

(When finished, your running program should produce output in the Transaction Log that look like this.)



```
Test03-Turk-Dan

Txn Log:

- Bank CREATED: ( BANK: ID=B01, Name=CIS605 Bank of Commerce, #Cust=0, #Acct=0 )

- Customer CREATED:
+ TOSTRING: ( CUSTOMER: ID='C01', Name='Sarah' )
+ INDIVIDUAL FIELDS:
> C ID='C01'
> C Name='Sarah'

- Customer CREATED:
+ TOSTRING: ( CUSTOMER: ID='C02', Name='Bill' )
+ INDIVIDUAL FIELDS:
> C ID='C02'
> C Name='Bill'

- Customer CREATED:
+ TOSTRING: ( CUSTOMER: ID='C03', Name='Jolene' )
+ INDIVIDUAL FIELDS:
> C ID='C03'
> C Name='Jolene'

- Account CREATED:
+ TOSTRING: ( ACCOUNT: ID=A01, Type='Savings', Name='Sarah's Savings', Cust=( CUSTOMER: ID='C01', Name='Sarah' ), You have $0.00 in your savings account )
+ INDIVIDUAL FIELDS:
> A ID='A01'
> A Type='Savings'
> A Name='Sarah's Savings'
> A Cust=( CUSTOMER: ID='C01', Name='Sarah' )
> A Bal='$0.00'

- Account CREATED:
+ TOSTRING: ( ACCOUNT: ID=A02, Type='Savings', Name='House Down Payment', Cust=( CUSTOMER: ID='C02', Name='Bill' ), You have $0.00 in your savings account )
+ INDIVIDUAL FIELDS:
> A ID='A02'
> A Type='Savings'
> A Name='House Down Payment'
> A Cust=( CUSTOMER: ID='C02', Name='Bill' )
> A Bal='$0.00'

- Account CREATED:
+ TOSTRING: ( ACCOUNT: ID=A03, Type='Checking', Name='Jolene's Checking', Cust=( CUSTOMER: ID='C03', Name='Jolene' ), You have $0.00 in your checking account )
+ INDIVIDUAL FIELDS:
> A ID='A03'
> A Type='Checking'
> A Name='Jolene's Checking'
> A Cust=( CUSTOMER: ID='C03', Name='Jolene' )
> A Bal='$0.00'

- Account CREATED:
+ TOSTRING: ( ACCOUNT: ID=A04, Type='Loan', Name='Car loan', Cust=( CUSTOMER: ID='C03', Name='Jolene' ), You owe $0.00 on your loan )
+ INDIVIDUAL FIELDS:
> A ID='A04'
> A Type='Loan'
> A Name='Car loan'
> A Cust=( CUSTOMER: ID='C03', Name='Jolene' )
> A Bal='$0.00'

- Account CREATED:
+ TOSTRING: ( ACCOUNT: ID=A05, Type='Loan', Name='Home Loan', Cust=( CUSTOMER: ID='C01', Name='Sarah' ), You owe $0.00 on your loan )
+ INDIVIDUAL FIELDS:
> A ID='A05'
> A Type='Loan'
> A Name='Home Loan'
> A Cust=( CUSTOMER: ID='C01', Name='Sarah' )
> A Bal='$0.00'

- Bank STATUS: ( BANK: ID=B01, Name=CIS605 Bank of Commerce, #Cust=3, #Acct=5 )

[Process Test Data]
[Exit]
```

Programming Questions

1. Clean up from Test2.
 - a. (2 pts) If you are using code from Test2, and have not already done so, clean it up so that it does not have comments about Test2 anywhere in it and remove all code from within the ProcessTestData method on the main form.
2. (8 pts) Add two EventArgs Classes, one each to be used by the _createCustomer() and _createAccount() methods in the Bank class.
3. Update the Bank Class.

NOTE: Do not create any additional attributes.

 - a. (2 pts) Define the events to be raised by the _createCustomer() and _createAccount() methods.
 - b. (8 pts) Update your _createCustomer() and _createAccount() methods to raise events with appropriate corresponding names. As in Test2, these methods should be Function procedures, should accept and use parameters for all attributes of each respective type of object, create the object within each _create() method, and return the customer and account created, respectively, in these methods, based on the attribute parameters passed in.
2. Update the Account Class.
 - a. (2 pts) Add a set of public/private methods that provides the balance for the account. For instance, if my checking account had \$500 in it, the balance() method would return 500. If my loan account was for \$1000, the balance() method would return 1000. (At present this will always be zero, since we have no logic for storing/calculating other values.)
 - b. (6 pts) Add 3 sets of public/private methods, that determine whether the Account is a Savings, Checking, or Loan account. The public versions of these should be called isChecking(), isSavings(), and isLoan() respectively, and should return a True/False value depending on what they are checking and what type of Account the object is.
 - c. (6 pts) Update the _toString() method to include information telling what the balance is.
 - i. Ex: If the account is a checking account, it should say "You have \$X,XXX.XX in your checking account", where XXXX.XX is the account's current balance.
 - ii. Ex: If the account is a savings account, it should say "You have \$X,XXX.XX in your savings account", where XXXX.XX is the account's current balance.
 - iii. Ex: If the account is a loan account, it should say "You owe \$X,XXX.XX on your loan", where XXXX.XX is the account's current balance.
4. Update the Main Form.
 - a. (2 pts) Make a variable called mTheBank able to hear events raised in the Bank class, and create a single standard property procedure for managing this variable.

- b. (2 pts) In your `_initializeBusinessLogic()` method create a new bank object using the variable declared that will listen for events and using the property procedure that manages it.
- c. (2 pts) In your `_initializeUserInterface()` method, display the `ToString()` status of and information about the Bank object before any “transactions” are processed.
- d. (8 pts) Create two Event Procedures, one to respond to the event raised by the `Bank.createCustomer()` method and one to respond to the event raised by the `Bank.createAccount()` method. These event procedures will display the information illustrated in the screenshot above on the Transaction log first using the object’s `ToString()` method and second using the individual public property procedures to display each piece of information individually. See the screenshot for details of how this should look. Your running code should produce output that looks identical to that shown in this screenshot.
- e. In your `ProcessTestData` method, in the order described below, do the following:
 - i. If not already done, clear out anything that might be remaining in this method from `Test2`. (See above.)
 - ii. (2 pts) Define variables for 8 pieces of data described early in this document.
 - 1. Name the Customer variables `cust1`, `cust2`, and `cust3`.
 - 2. Name the Account variables `acct1`, `acct2`, `acct3`, `acct4`, and `acct5`.
 - iii. (2 pts) Create customers using their relevant “create” methods.
 - iv. (2 pts) Create accounts using their relevant “create” methods.
 - v. (2 pts) Display the `ToString()` information about the Bank object after all 8 of the above “create” transactions have been processed.