SIT771 Object Oriented Development

(Trimester 1, 2014)

Assignment 2 (150 marks, weight 15%)

Due Date: Monday, 5 May, 2014 (11:59pm AEST)

Introduction

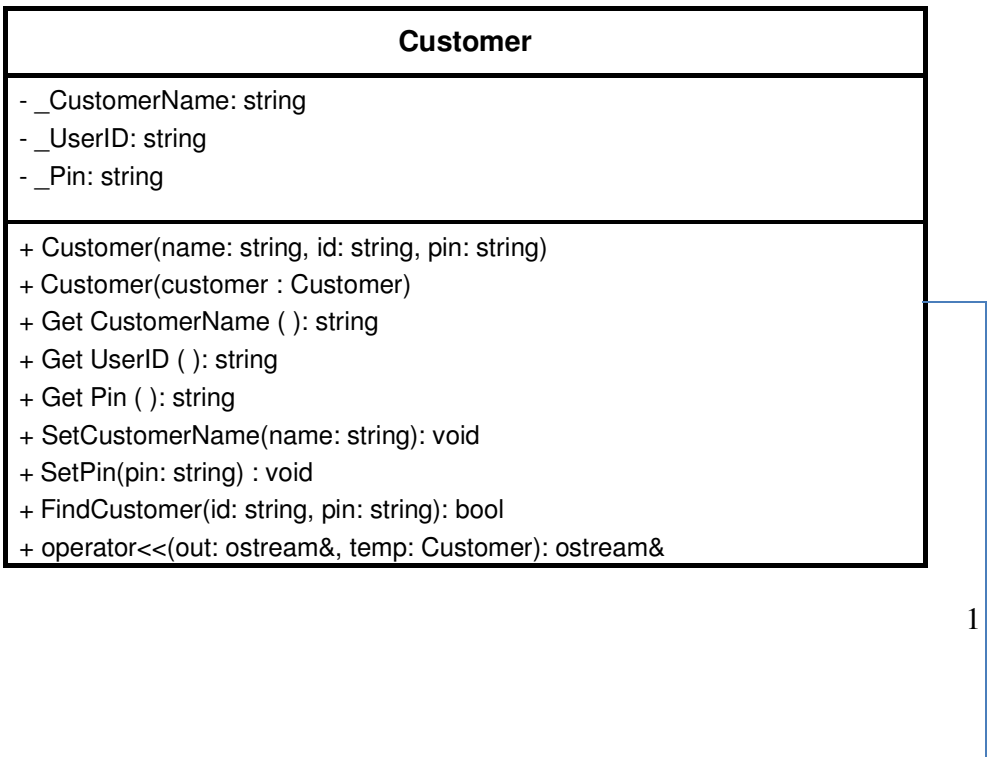
This assignment intends to get you familiar with Object Oriented programming. You are required to re-develop *Customer, Transaction* and *Account* structure from the assignment 1 as a reusable class, and re-develop the program to track transactions of a bank account using the *Account* class. The solution should have the following features:

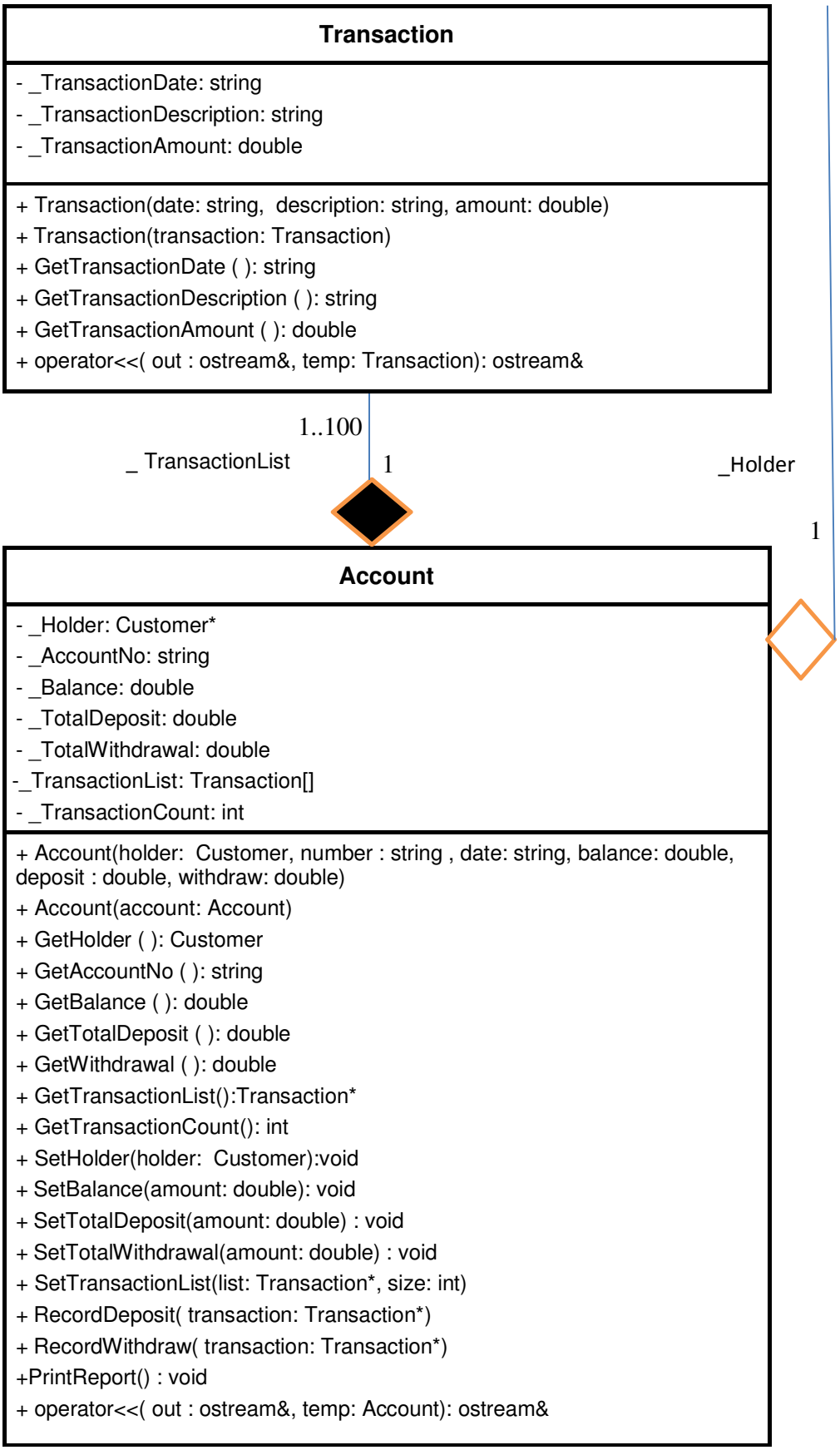
- Implements the Customer class information;
- Implements the Transaction class information;
- Implements the Account class information; and
- Uses the provided main to call the above class methods and displays summary information for the bank account.

The detailed requirements are as follows.

Requirements

In the solution, you are required to implement and use a **Customer** class which is shown in the following UML diagram:




Transaction
- _TransactionDate: string
- _TransactionDescription: string
- _TransactionAmount: double
+ Transaction(date: string, description: string, amount: double)
+ Transaction(transaction: Transaction)
+ GetTransactionDate ( ): string
+ GetTransactionDescription ( ): string
+ GetTransactionAmount ( ): double
+ operator<<(out : ostream&, temp: Transaction): ostream&
1..100
_ TransactionList
1
_Holder
1
Account
- _Holder: Customer*
- _AccountNo: string
- _Balance: double
- _TotalDeposit: double
- _TotalWithdrawal: double
-_TransactionList: Transaction[]
- _TransactionCount: int
+ Account(holder: Customer, number : string , date: string, balance: double, deposit : double, withdraw: double)
+ Account(account: Account)
+ GetHolder ( ): Customer
+ GetAccountNo ( ): string
+ GetBalance ( ): double
+ GetTotalDeposit ( ): double
+ GetWithdrawal ( ): double
+ GetTransactionList():Transaction*
+ GetTransactionCount(): int
+ SetHolder(holder: Customer):void
+ SetBalance(amount: double): void
+ SetTotalDeposit(amount: double) : void
+ SetTotalWithdrawal(amount: double) : void
+ SetTransactionList(list: Transaction*, size: int)
+ RecordDeposit(transaction: Transaction*)
+ RecordWithdraw(transaction: Transaction*)
+PrintReport() : void
+ operator<<(out : ostream&, temp: Account): ostream&

Provided main program:

```
int main()
{
    Customer* Mary = new Customer("Mary Jones", "235718", "5074");
    Customer* John = new Customer("John Smith", "375864", "3251");

    Account* MaryAccount = new Account(*Mary, "06-3121-10212357", "01/03/2014", 100);
    Account* JohnAccount = new Account(*John, "06-3121-10213758", "10/03/2014");

    MaryAccount->RecordWithdraw(new Transaction("01/03/2014", "ATM Withdrawal", 50) );
    MaryAccount->RecordDeposit(new Transaction("02/03/2014", "Deposit", 90) );
    MaryAccount->RecordWithdraw(new Transaction("04/03/2014", "ATM Withdrawal", 150) );
    MaryAccount->RecordDeposit(new Transaction("05/03/2014", "Deposit", 20) );
    MaryAccount->RecordWithdraw(new Transaction("05/03/2014", "Withdraw", 100) );
    MaryAccount->RecordWithdraw(new Transaction("05/03/2014", "Withdraw", 50) );

    JohnAccount->RecordDeposit(new Transaction("11/03/2014", "Deposit", 20) );
    JohnAccount->RecordDeposit(new Transaction("12/03/2014", "Deposit", 80) );
    JohnAccount->RecordWithdraw(new Transaction("12/03/2014", "Withdraw", 50) );

    MaryAccount->PrintReport();
    JohnAccount->PrintReport();

    return 0;
}
```

Required Output:

```
ERROR! invalid withdrawal value

ACCOUNT SUMMARY :: 06-3121-10212357 - Mary Jones
        Total Deposits:              $     110.00 DR
        Total Withdrawals:           $     200.00 CR
        Final Balance:               $      10.00 DR

Date       Description          Deposit    withdrawal     Balance
 ============================================================================
01/03/2014 Opening Balance    $   100.00                 $     100.00
01/03/2014 ATM withdrawal                  $     50.00  $      50.00
02/03/2014 Deposit            $    90.00                 $     140.00
05/03/2014 Deposit            $    20.00                 $     160.00
05/03/2014 withdraw                        $    100.00  $      60.00
05/03/2014 withdraw                        $     50.00  $      10.00



ACCOUNT SUMMARY :: 06-3121-10213758 - John Smith
        Total Deposits:              $     100.00 DR
        Total Withdrawals:           $      50.00 CR
        Final Balance:               $      50.00 DR

Date       Description          Deposit    withdrawal     Balance
 ============================================================================
10/03/2014 Opening Balance    $     0.00                 $       0.00
11/03/2014 Deposit            $    20.00                 $      20.00
12/03/2014 Deposit            $    80.00                 $     100.00
12/03/2014 withdraw                        $     50.00  $      50.00
```

Your program is required to perform the following tasks:

1. Customer constructor has name which sets to blank default value, id is set to "000001" and pin is set to "0000" default value.
2. Transaction constructor has date that is set to "01/01/2014" for default value, description starts with "Opening Balance" and amount is default zero.
3. Account constructor has holder which is set to Customer default constructor object, number is set to "06-3121-1021-0001" , first transaction date is "01/01/2014", balance is zero, total deposit is zero, and the total withdraw is zero. The 1st item on Transaction List array is the transaction create in parameter date, the description is "Opening Balance" and the balance is from the parameter value. Therefore, the Transaction Count should be start at 1.
4. SetHolder() needs to destructor/release the current object before use the deep copy to create a new object from the parameter.
5. SetTransactionList() needs a loop to assign all parameter transaction list to the Account Transaction List and Update the Transaction Count to the size parameter. This method will be used in the Account copy constructor.
6. PrintReport() should display the account details and all the Transaction List records on the array according the output format provided.
7. All three classes should have the operator<< overload for the ostream type (check lecture Journey example)

Submissions

1. Two files for the class *Customer.h, Transaction.h* and *Account.h* which contains the declaration of the class, and *three .cpp files* which contains the definition of the class (i.e. executable code);
2. One file that contains a *main()* method, performing the above tasks by invoking methods in all above three classes.
3. Completed electronic assignment cover sheet.

Additional Requirements and Notes

1. All code files must contain a comment at the top with the following information:
   - Your name and student ID number;
   - Which assignment the code is written for;
   - A general comment describing the code contained within that file;
   - Any reference materials that were used in building your solution.
2. All code should be adequately commented such that another programmer can understand your code quickly. In particular,
   - you should describe any attributes by indicating what they are used for and how they should be manipulated;
   - any methods should be described by indicating their purpose and a description of their functionality (how they work), and
   - each parameter and return value should be described.
3. The Faculty electronic assignment cover sheet is available on CloudDeakin and must be completed and submitted with your assignment.
4. Any text or code adapted from any source must be clearly labelled and referenced. You should clearly indicate the start and end of any such text/code.
5. All assignments must be submitted through CloudDeakin. Assignments will not be accepted through any other manner without prior approval. Students should note that this means that email and paper based submissions will ordinarily be rejected.
6. Submissions received after the due date will be penalized at a rate of 10% per day. Assignments that are submitted ***three*** days after the due date will NOT be assessed.
7. To apply for an assignment extension, the assignment extension form (found on CloudDeakin) must be completed and submitted to the lecturer responsible for your assessment prior to the due date. Applications on or after the due date will only be considered where students can demonstrate that it was impractical to apply otherwise.
8. Your attention is drawn to the **Plagiarism Statement** (available in the folder Academic Honesty on CloudDeakin). Anyone using cut-and-paste or copying other people's work will be easily detected; the outcome is a disciplinary committee hearing.

Assessment Guide

1. Declaration of the Customer class (i.e. *CustomerAccount.h*)	/35
2. Declaration of the Transaction class (i.e. *TransactionAccount.h*)	/35
3. Declaration of the Account class (i.e. *Account.h*)	/50
4. Comments describing the program code. *Comments are required for attributes, methods, parameters etc.*	/15
5. Program performance. *Whether the code can be compiled, required tasks can be performed correctly?*	/15
Total	/150

.