

# SIT771 Object Oriented Development

(Trimester 1, 2014)

## Assignment 1

(150 marks, weight 15%)

Due Date: Monday, 7 April, 2014 (11:59pm AEST).

### Introduction

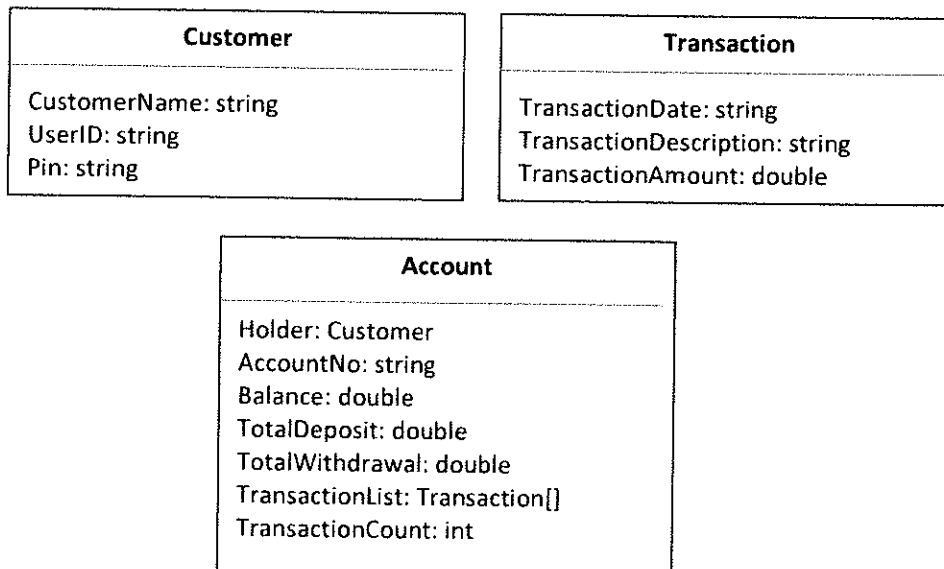
This assignment intends to get you familiar with Object Oriented programming. You are required to develop the *BankAccount* structure for the assignment 1, and develop sub-functions will be called by the provided main program to track transactions of a bank account using the *BankAccount* structure. The solution should have the following features:

- Declares three structures for Customer, Transaction and Account information;
- Implements function to Create Customer, Transaction and Account object;
- Records the valid Deposit transaction information;
- Records the valid Withdrawal transaction method; and
- Create function to Display summary balance information for the bank account.

The detailed requirements are as follows.

### Requirements

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



- The **Customer** structure (defined in a header file Customer.h) containing three members: Customer Name, User ID and Pin.

- A function **CreateCustomer()** that creates a Customer. It has three parameters to initialise each member of the structure (Customer Name, User ID, and PinNumber). All parameters should have default parameter blank values are to be specified. Prototype:

*Customer\* CreateCustomer(const string& name, const string& id, const string& pin)*

- The **Transaction** structure (defined in a header file Transaction.h) containing three members: Transaction Date, Description and Amount. The Amount can't be negative value.

- A function **CreateTransaction()** that creates a Transaction. It has three parameters to initialise each member of the structure (Transaction Date, Description and Amount). All parameters should have default parameter values are to be specified. The Date sets to "01/01/2014", the Description is blank and amount is zero. Prototype:

*Transaction\* CreateTransaction(const string& date, const string& description, const double& amount)*

- The **Account** structure (defined in a header file Account.h) containing seven members: Account Holder, Number, Balance, Total Deposit, Total Withdrawal, Transaction List array of maximum 100 records and Transaction count on the existing transaction records.

- A function **CreateAccount()** that creates an Account. It has six parameters to initialise each appropriate member of the structure. The date parameter use to create the first transaction for the Transaction List array, the description is Opening balance from the balance parameter. Therefore, the Transaction count should be set to one after the Account is created. All parameters from the third one should have default parameter values are to be specified. The Date sets to "01/01/2014", the balance, deposit and withdraw set to zero value. Prototype:

*Account\* CreateAccount(const Customer& holder, const string& number, const string& date, const double& balance, const double& deposit, const double& withdraw)*

- A function **RecordDeposit()** that store the Transaction parameter into the Transaction List array of the parameter Account and increasing the Transaction count by one. Make sure to check the transaction parameter must have the amount higher than zero. Otherwise, display an error message. There is no requiring for the default parameter values.

*void RecordDeposit(Account\* account, Transaction\* transaction)*

- A function **RecordWithdraw()** that store the Transaction parameter into the Transaction List array of the parameter Account and increasing the Transaction count by one. Make sure to check the transaction parameter must have the amount higher than zero and less than or equal to the parameter account balance. Otherwise, display an error message. There is no requiring for the default parameter values.

*void RecordWithdraw(Account\* account, Transaction\* transaction)*

- A function **PrintReport()** that accepts an Account pointer parameter to display the summary balance account and the list of Transaction record on the array to displays as the provided output format. Prototype:

*void PrintReport(Account\* account)*

### Provided Code - main()

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

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

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

    RecordDeposit(JohnAccount, CreateTransaction("11/03/2014", "Deposit", 20) );
    RecordDeposit(JohnAccount, CreateTransaction("12/03/2014", "Deposit", 80) );
    RecordWithdraw(JohnAccount, CreateTransaction("12/03/2014", "Withdraw", 50) );

    PrintReport(MaryAccount);
    PrintReport(JohnAccount);

    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	Withdrawal		\$ 100.00	\$ 60.00
05/03/2014	Withdrawal		\$ 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	Withdrawal		\$ 50.00	\$ 50.00

## Submissions

1. Three files for the *Customer*, *Transaction* and *Account.h* which contains the declaration of the structure, function implementation which contains the definition of the header (i.e. executable code);
2. One file that contains a *main()* method, performing the above tasks by invoking methods in the structures of the *Bank Account*.
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 structure Customer (i.e. <i>Customer.h</i> )	/12
2. Declaration of the structure Transaction (i.e. <i>Transaction.h</i> )	/12
3. Declaration of the structure Account (i.e. <i>Account.h</i> )	/15
2. Implement functions	
- <i>CreateCustomer()</i>	/09
- <i>Create Transaction ()</i>	/09
- <i>Create Account ()</i>	/15
- <i>RecordDeposit()</i> .	/10
- <i>RecordWithdraw()</i> .	/15
- <i>PrintReport()</i> .	/23
3. Comments describing the program code. <i>Comments are required for attributes, methods, parameters etc.</i>	/15
4. Program performance. <i>Whether the code can be compiled, required tasks can be performed correctly?</i>	/15
<b>Total</b>	/150

# SIT771 Object Oriented Development

## (Trimester 1, 2014)

### Assignment 2

(150 marks, weight 15%)

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

### Introduction

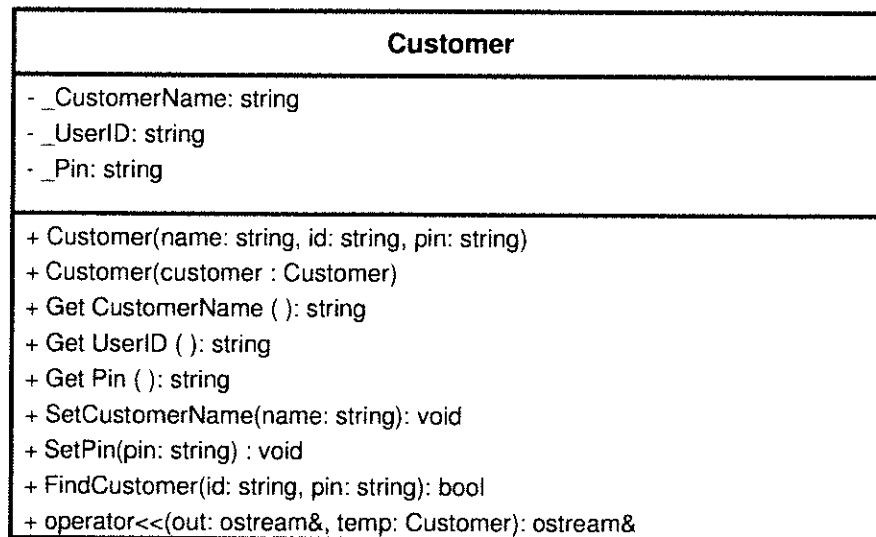
This assignment intends to get you familiar with Object Oriented programming. You are required to re-develop *Customer*, *Transaction* and *Account* structure from the assignment 1 as a reusable class, and re-develop the program to track transactions of a bank account using the *Account* class. The solution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

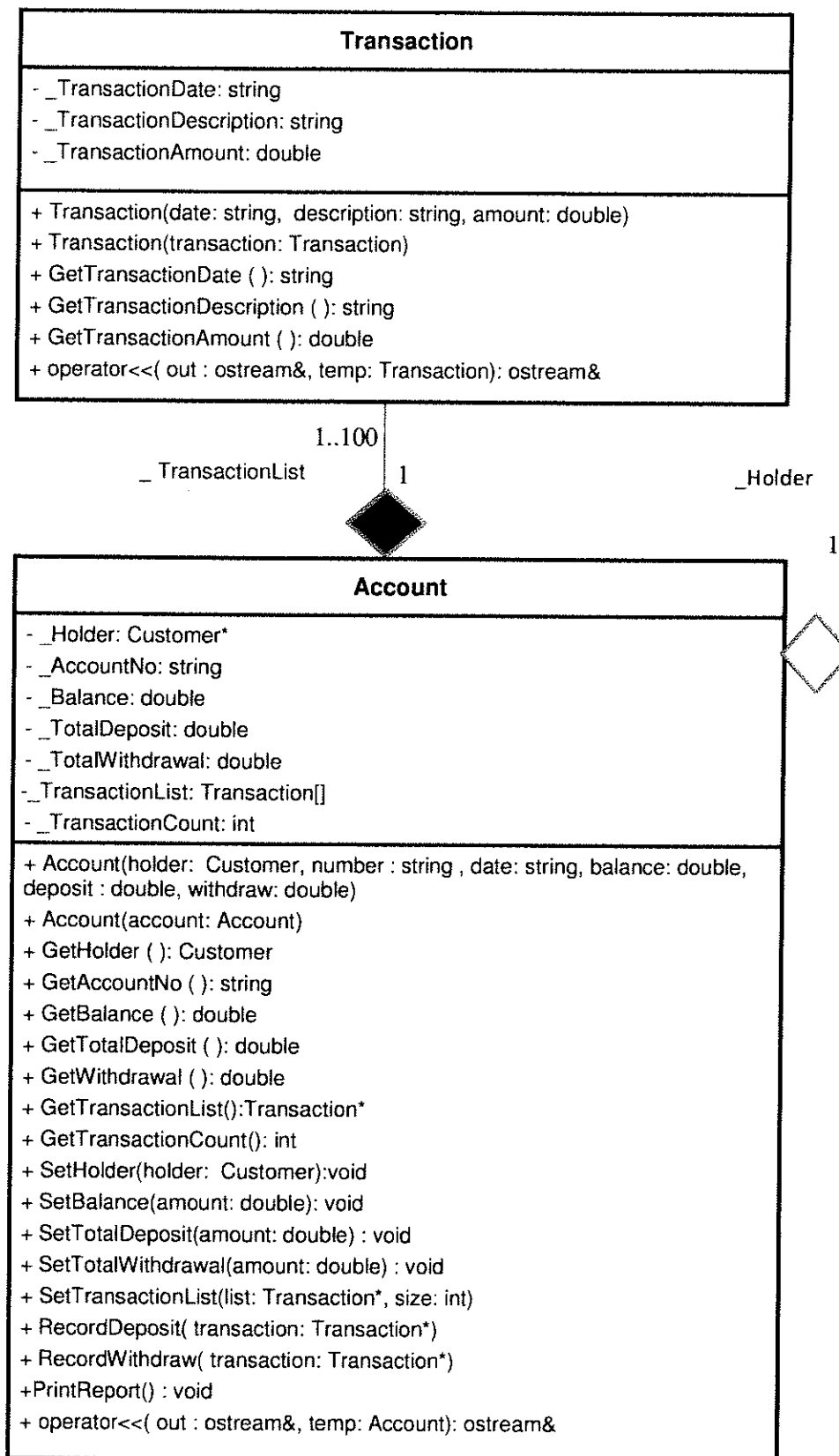
- Implements the Customer class information;
- Implements the Transaction class information;
- Implements the Account class information; and
- Uses the provided main to call the above class methods and displays summary information for the bank account.

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:





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