

# Final Project

**Automatic Teller Simulator (25%)**

## DUE DATE

---

Part 1- session 7

Part 2- session 11

Part 3- session 15

## OBJECTIVE

The main objectives of this project are to:

- Interpret specifications and analysis performed
- Design a solution based on the requirements and specifications
- Design the logic required for an event-driven solution
- Create a user interface leveraging UWP / WPF components
- Translate design documents and algorithms into source code
- Read from and write to a file/ database / or other storage medium
- Implement data validation and error handling.
- Use debugging tools, and error-handling techniques
- Validate the solution with test data
- Integrate the knowledge acquired thus far
- Have fun while programming with Visual C#.NET

## DESCRIPTION

---

Project is divided into 3 parts and requires 20-30 hours to complete this project including off-campus homework hours.

## REQUIRED MATERIAL

You will need the following material to complete this project:

- Microsoft Visual Studio.NET (specifically C#.NET)
- Visio (Standard or Professional version)
- Microsoft Visual Studio.NET documentation or any other reference material suggested or provided by your instructor
- A blank high-density diskette (to give back when the project is completed)

Your instructor may provide you with sample of a correctly functioning application model. Your application need not look exactly like the sample. Any good solution is acceptable if produced within the time allowed. Your ATM must have at least three Windows forms (login, main, and supervisor). You will also receive the banking data files required for the solution. (txt files or a database)

After successfully completing your studies, you have landed the job of your dreams with a consulting firm. Today is your first day on the job as a junior programmer. Your immediate supervisor gives you the responsibility to develop an application prototype that simulates the operation of an automatic teller machine (ATM). The client is running an older version and would like it updated.

Your supervisor is expecting you to deliver a functioning prototype in four days. This is your opportunity to showcase your creative talents, and your analysis, design, coding, and testing abilities.

This project will leverage some of the design features that you completed as part of the Programming Logic and Design project. During your first course you were asked to complete the logic and design of the Automatic Teller Simulator. Now that you have completed the C# programming courses, it is time for you to program the functional aspect of the ATS. In this project, you will expand upon what you designed in the Programming Logic and Design project into a functional simulator by adding the proper C# code to provide the desired functionality.

### Key Functions

The main functions of the ATM simulator may be summarized as follows:

- Before performing any transaction, the user must enter his or her name and PIN (personal identification number) on an input screen. Since the operation of this input screen should simulate the normal operation of an ATM, the PIN should not appear on the screen.
- In addition to the message which appears after every unsuccessful attempt, if after three tries the PIN matching the name has not been entered, the application should display a message requesting the user to try using the ATM again later. The names and PINs of users must be validated using data contained in the **Customers** data source having the following structure:
  - name (String)
  - PIN (String – 4 characters)

Note: For the purposes of this project, the data source can be either of two types, a SQL database whereby you will require to connect your application to the database tables and access the content accordingly using LINQ. Alternatively, you can use the provided CustomerInfo.txt text file as a source of the data. In this case you will need to use the handling files and streams information found in the content for Session 7 of your Student Learning Guide. It is your choice on how you store and access the data.

- Once access has been authorized, the main form of the application should allow the user to carry out one of the following transactions:
  - Deposit

- Withdrawal
- Transfer
- Bill payment
- 
- When a user performs an operation, the application should first ask if it should be done using a chequing or savings account, and then ask for the transaction amount. Below is the structure of the Accounts.txt sequential file ( database) in which account balances are stored:
  - account type (1 character)
  - PIN (String – 4 characters)
  - account number (String – 5 characters)
  - account balance (single)

Note: if you are using the Database setup instead of the text files, please note that the Accounts database has the same fields and structure as the Accounts text file.

- For a deposit, the user must enter the amount and, if required, be able to select the account type to be credited. The chequing account is the default for this transaction. There is no maximum amount.
- For a withdrawal, the user must enter the amount and, if required, be able to select the account type to be debited. The chequing account is the default for this transaction subject to a maximum of \$1,000. If the user enters an amount greater than \$1000, the system must present a message indicating that \$1000 is the maximum per withdrawal. The ATM accepts only transactions for which the amount entered is a multiple of \$10. There is no maximum amount (apart from the user's account balance) in which case the system must also inform the user if the account balance is less than the transaction amount. In addition, the ATM Machine starts its day with a full \$20,000 in funds. Each withdrawal must also be validated against the amount left in the machine. If the withdrawal amount is greater than the amount left in the machine, the system must inform the user that the machine has insufficient funds (not the user account)
- For a transfer, the user must enter the amount and the type of transfer (from chequing to savings, or savings to chequing). This transaction is subject to a maximum \$10,000 (or the user's account balance). The system must also validate that the account balance from which the amount is being transferred has sufficient funds to cover the transaction amount.
- For a bill payment, which is done from a chequing account only, the user must enter the amount of the transaction. The chequing account is debited by the same amount. In addition, a \$1.25 fee is charged to the chequing account. The maximum per transaction is \$10,000 (or the user's chequing account balance). Note that if the \$10,000 maximum bill payment is made, then the actual amount removed from the account is \$10,001.25
- The application must check the account balance before doing a transaction. Any transaction that would result in a negative balance must be rejected.
- The balance of the account affected by a transaction should be updated and displayed after each

transaction.

- The user should be able to do as many transactions as he or she would like to do before leaving the ATM.
- A warning message should inform the user that the ATM can no longer carry out withdrawals when there is no money available. When a withdrawal transaction event occurs for an amount greater than the balance remaining in the ATM, the ATM should advise the user that they can change the transaction amount to the amount still available in the ATM.
- Each time the application starts, the ATM checks DailyBalances.txt to see if it contains a record with today's date. If not, then this is the first time the ATM is used today, and it automatically sets the ATM's balance to \$20,000 and adds a record to DailyBalances.txt Below is the structure of the DailyBalances data source (database table or text file, your choice) in which the ATM's daily balances are stored:
  - Date (DateTime)
  - ATM balance (single)

The following functions concern the ATM's functioning with respect to the system administrator and the internal mechanisms of the ATM, not the user.

- The system administrator, as any other user, must enter his or her name and PIN (personal identification number) on the same input screen. The system administrator may perform only system transactions (he or she has no personal account).
- Once access has been authorized, a special menu is displayed. This menu offers the following options:
  - pay interest
  - refill the ATM with money
  - take the ATM out of service
  - print the accounts report
- The supervisor can cause interest to be paid to all savings accounts at the rate of 1% (new balance = current balance + (current balance \* 0.01)).
- The system administrator re-fills the ATM in batches of \$5,000. There should not be more than \$20,000 available in the ATM.
- The supervisor can take the ATM out of service, which ends the program (in theory, only the supervisor can end the program - when a regular user exits, it should return to the Welcome screen for another user to login).
- The accounts report displays all chequing and savings account information for all customers in a textbox.

Note: If all the key functions mentioned above are met, a maximum mark of 100% may be given.

### Error Management: Test the ATM Prototype application

1. You should test as you build to avoid unnecessary delay.
2. Make sure you add appropriate data input validation, and error-handling (HINT: every time you open a file, connect to a database or accept user input, you should provide error-handling)

### CLASS DESIGN:

Since you have not yet learned about object-oriented design, a basic class design is provided for you. This is just a sample (you may not need all of the classes, properties and methods listed below, but it is a nicely detailed list). You may find it useful to quickly diagram the classes. You will need to determine the data type and scope of each class member: public, private, etc.

NOTE: all classes will require customized constructors.

If you have time and want to design your own classes, you may do so provided they meet the specifications, you meet the deadline and you have your instructor's permission.

### Class Library:

#### Account

Properties: pinNumber, accountNumber, accountBalance, maximumWithdrawal, maximumTransferAmount.

Methods: withdraw (amount); deposit(amount); transferOut(amount); transferIn(amount).

You will need the following to return values: Get Pin, Get Account Number, and Get Balance.

#### Chequing

Inherits from Account.

Property: maximumBillAmount

Constant Properties: billFee

Methods: PayBill (amount) returns Boolean.

#### Savings

Inherits from Account.

Constant Properties: interestRate.

Methods: PayInterest()

#### Bank

Inherits from Account.

Constant Properties: maximumTopUp, refillAmount

Methods: refillATM().

#### chequingAccounts

This class needs to add a chequing account, and to return a chequing account (using the index or key depending upon type of collection used).

### **savingsAccounts**

This class needs to add a savings account, and to return a savings account (using the index or key depending upon type of collection used).

### **DailyBalance**

Properties: atmDate, atmBalance

Constant Property: minimumAmount

Methods: UpdateDailyBalance() to update atmBalance when supervisor fills up ATM.

### **DailyBalances**

This class needs to add a daily balance, and to return a daily balance (using the index or key depending upon type of collection used).

### **Customer**

Properties: name, PINNumber

Methods: May need to return strings with GetName and GetPinNumber.

### **Customers**

This class needs to add a customer account, and to return a customer (using the index or key depending upon type of collection used).

### **ATMManager**

Properties: bank, customers, chequingAccounts, savingsAccounts, currentAccountBalance, dailyBalances.

Methods:

ValidateUser(name, pin) returns Boolean

WithdrawChequing(pin, amount) returns Boolean

WithdrawSavings(pin, amount) returns Boolean

DepositChequing(pin, amount) returns Boolean

DepositSavings(pin, amount) returns Boolean

PayBillPayment(pin, amount) returns Boolean

TransferFunds(pin, amount, accountType) returns Boolean.

- The account type determines from which account to transfer out, and which account to transfer in

DisplayAccountBalance() returns currentAccountBalance

ReadCustomers() returns Boolean

ReadAccounts() returns Boolean

- Do not read the first character (C or S) found in the Accounts.txt into the object instances. Use it to determine to which collection the account should be added (chequingAccounts or savingAccounts)

WriteAccounts() returns Boolean

- Write the bank, all chequingAccounts, and all savingsAccounts to Accounts.txt.
- Add a first character (C or S) to the beginning of each account string before writing to Accounts.txt.

ReadDailyBalances() returns Boolean

CheckDailyBalances() returns Boolean

- Checks DailyBalances.txt for today's date.

AddDailyBalance()

- Adds new record for today's date and default ATM balance.

WriteDailyBalances()

- Writes all records to DailyBalances.txt

### Client Application:

The properties and methods are largely determined by the interface design. This assumes only three forms. You may use more.

#### Login form

Properties: atmManager, pin, amount, and a Boolean accountsRead variable.

Methods: ReadCustomers().

Event Handlers: Submit: If the user is found, it sets the next form, for example, frmMain to visible and activates it.

#### Main form

Properties: user, pin, counter. Also need an instance of any forms launched from the login form, for example frmMain.

Methods: InputNumber(string number) validates the amount entered by the user – see Tips and Hints below.

Event Handlers:

Load: Reads accounts if not read before.

Close: Sets visible property.

Transaction radio buttons: Each determines if is checked. If true, it controls which account radio button is checked if a default is required.

Keypad 'buttons': Each label sends a value to InputNumber.

Submit: (pin number and amount). Writes account data to the Accounts.txt file. Display information to the user.

#### Supervisor form

Properties: need an instance of any forms launched from the login form, for example frmSupervisor.

Methods: ListAccounts – see Tips and Hints below.

Event Handlers:

Load: Reads accounts if not read before.

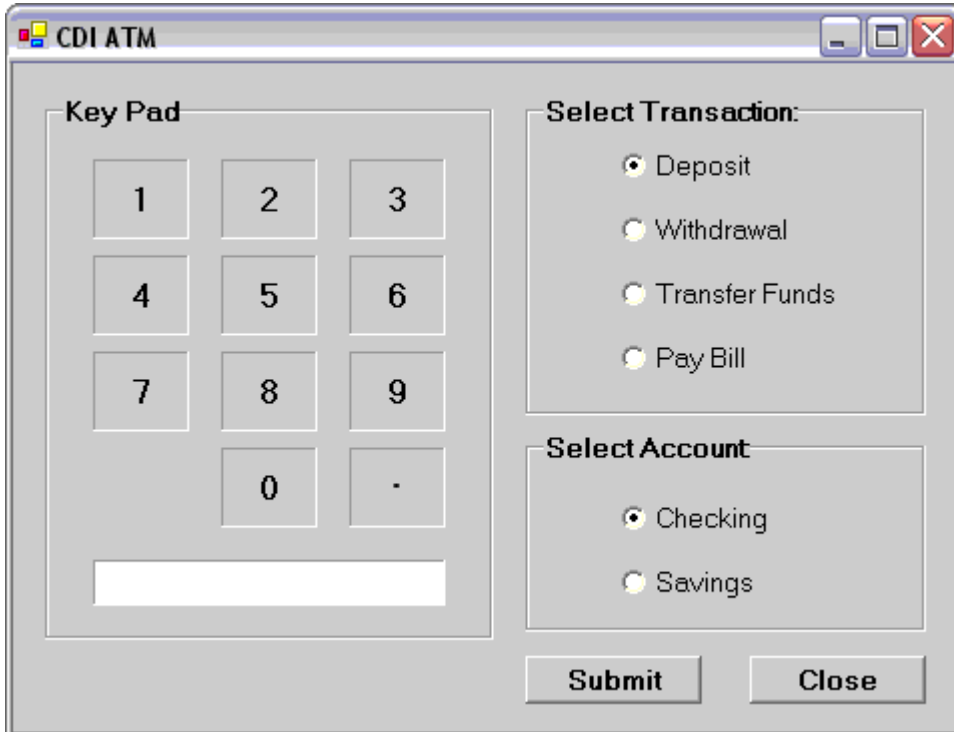
Close: Sets visible property.

Open, Exit, View Accounts, Print.

Pay Interest, Refill ATM.

**TRICKS AND HINTS**

- If the logic is the same for more than one method, you can copy and edit as required.
- You can create the application using only three forms: a login form, a main form, and a supervisor form (you can also have a welcome form). The appropriate form becomes visible if the user's name and PIN are found in Customers (text file or data table) The main form might look something like the following:



Note that the form above does not include any notices to the user about limits or additional fees. These notices must appear somewhere for example in the form or in message boxes, or a combination.

Each click on a 'key' sends the equivalent string value to an InputNumber method. This method appends each number string and displays the resulting string in the text box below. You should make sure the user cannot enter two decimals (HINT: some buttons can be enabled/disabled as certain buttons are clicked).

Submit would call the class method as determined by the radio buttons selected.

- Make sure you draft the class library before you modify or create any Windows forms. It is much easier to add or remove controls in a form, than it is to write classes to fit a form.
- You will need to convert data from one to another especially when reading and writing to files, or



displaying information in message boxes.

- Much of the code will be the same for each class. You can save time by copying and editing code. Be careful when editing.
- Make sure the Windows forms can access the information in all of the forms. To do this, create a form-level instance for the main and for the supervisor form in the login form. For the main form: assign to the main form's instance, any variables you need to access in both forms. For example, each form would need a PIN variable. You get the pin in the login form to validate the user. You need to use the same PIN value in the main form to find the user's account therefore you assign the user's pin to the PIN variable in the main form before you switch to the main form. To switch to the main form from the login form, set the main form's Visible and Activate properties. To return to the login form from the main form, set Visible to false in a Close button (when the main form is visible, the login form is as well unless you control it with its visible property as well). For the supervisor form: assign any required variables in login in the same manner as the main form.
- You should only have one atmManager property declared in your program, in your login form. You should not re-declare it in your other forms. It should be declared as public so that the other forms can access it.
- Use a textbox to display the accounts report in the supervisor form.
- Create the client so that you can test the classes as you define them. A working application that is 75% complete will earn higher marks than a non-working application that is 100% complete, or one that is more than 3 days late.
- Validate input data as much as possible.
- Don't forget to comment your code.

## INSTRUCTIONS

---

### SCHEDULE

Very important: Do not wait until the end of Session 11 to start working on your project. It is important to start thinking through the possible solutions as soon as possible. It is recommended that you begin looking at the design pieces of this project at the latest Session 5. This will allow you to begin your preparation early enough and revisiting it as you acquire additional skills and knowledge. Adopting this strategy will allow you to save the time in Sessions 12 to 15 to complete the actual final coding of the methods and forms. If you keep the full project to the last 4 sessions, you risk not having enough time.

The following schedule is just a guideline. It is not set in stone. The sooner you complete the analysis

and design portion; the sooner you can start the actual coding.

### PART 1 ANALYSIS AND DESIGN - DUE SESSION 7

Sessions 5-7: Analysis and Design of the work to be done

- Analyze the project specifications, and text files.
- Compare the sample application with the specifications.
- Note the similarities and differences between transactions and the two accounts.
- Create a data dictionary, a relationship diagram and a DFD for your event-driven ATS application.

#### SUBMISSION INSTRUCTIONS

Work must be submitted in the correct file type and be properly labelled as per the College naming convention:

NAME\_COURSE\_ASSIGNMENT. E.g. XuXiaLing\_FM50D\_A01.

Format of your documents: PDF or jpeg/png (in case of images)

#### SUBMIT

- all relevant documentation including diagrams, data dictionary and a DFD in one zipped folder named as NAME\_COURSE\_ASSIGNMENT\_design

### PART 2 BUILD LOGIC - DUE SESSION 11

Sessions 8-11: Logic Required by the Processes

- Identify the classes (things) and their properties and methods (processes). Note the methods that can be inherited and any that can be overridden.
  - Using UML is not an outcome of this course. Use the class design information listed previously as your reference.
- Analyze the processes to be performed in the application.
- Create flowcharts or pseudocode (at least one instance of each) using Visio to describe the processes within the application.

#### SUBMISSION INSTRUCTIONS

Work must be submitted in the correct file type and be properly labelled as per the College naming convention:

NAME\_COURSE\_ASSIGNMENT. E.g. XuXiaLing\_FM50D\_A01.

Format of your documents: PDF or jpeg/png (in case of images)

### SUBMIT

- all relevant documentation including class methods relationships, flowcharts, pseudocode in one zipped folder named as NAME\_COURSE\_ASSIGNMENT\_logic
- all extra documentation in separate zipped folder named as NAME\_COURSE\_ASSIGNMENT\_logic\_extra

## PART 3 IMPLEMENTATION - DUE SESSION 15

### Sessions 12 to 15: Apply Analysis and Design to Coding the Application

- Apply your problem solving skills
- Define the classes beginning with the base class. Encapsulate the base, derived and collection classes in a single class that will act as the main interface for the library.
- Create the client application to test your class as you progress.
- Add appropriate data input validation and error-handling
- When coding your application, you may need to revise the logic (iterative development). Make sure you update your design documentation.
- Validate the solution using test cases.
- Add any missing input validation or error-handling.
- Update the documentation.
- Submit the project.

### SUBMISSION INSTRUCTIONS

Work must be submitted in the correct file type and be properly labelled as per the College naming convention:

NAME\_COURSE\_ASSIGNMENT. E.g. XuXiaLing\_FM50D\_A01.

Format of your documents: PDF or jpeg/png (in case of images)

SUBMIT: All of them in single zipped folder as NAME\_COURSE\_ASSIGNMENT\_final\_project

- all relevant and update documentation related to designs in one folder named as NAME\_COURSE\_ASSIGNMENT\_design
- all relevant and update documentation related to logic in one folder named as NAME\_COURSE\_ASSIGNMENT\_logic
- all code should be placed in one folder named as NAME\_COURSE\_ASSIGNMENT\_code
- A project report containing: a title page with your name, the submission date and your instructor's name, a table of contents, the project specifications, explanations of any additions that you made (where applicable), list of features that is completely working, partially working or not done should be submitted in pdf format named as NAME\_COURSE\_ASSIGNMENT\_project\_documentation

### SUBMISSION INSTRUCTIONS

---

Refer to the submission provided beneath each of the three project components above. You will submit each component of the final project in the relevant submission page.

We strongly recommend that you verify, using anti-virus software, that your diskette contains no viruses.

#### Penalties

- A penalty of 5% will be deducted from your mark for each day your project is late.
- Any project submitted more than 3 calendar days late will receive a maximum mark of 60%.
- Any project containing a virus must be resubmitted and will receive a maximum mark of 60%.

### EXTRA FUNCTIONALITY

#### IF YOU HAVE TIME (THIS IS OPTIONAL ONLY AND NOT A REQUIREMENT)

- The details of each transaction should be recorded in the Transactions.txt sequential file, as follows:
  - o transaction number (Integer)
  - o transaction date (String)
  - o PIN (String)
  - o user account number (String – 5 characters)
  - o user name (String)
  - o account type (Char – ‘C’ for chequing, ‘S’ for savings)
  - o transaction code (Char – ‘D’ for deposit, ‘W’ for withdrawal, ‘T’ for transfer or ‘B’ for bill payment)
  - o transaction amount (single)
  - o account balance (single)
  - o balance available at ATM (single)
- After each transaction, the system displays a transaction record detailing the last transaction made to the user.
- The following functions concern the ATM's functioning with respect to the system administrator and the internal mechanisms of the ATM, not the user.
  - o Add to the special menu:
    - daily transactions report
  - o When the system administrator puts in more money in batches of \$5,000, it will appear as a transaction. For transaction codes use “F” for the first automatic fill of the day and “R” for system administrator refills.
  - o The system administrator should be able to request a listing of all the transactions. This list should show all the information contained in the transaction file.

### GRADING CRITERIA

---

Project Value: 25%

Grading Criteria	Grading
<b>Part 1 - Design</b> <b>Analysis and Design (data dictionary, revised documents) - DFD,</b> Relationship diagrams and any other design document	15
<b>Part2 - Logic</b> <b>Flowcharts, pseudocode, classes layout structure - Revised logic required by</b> the processes/methods (pseudocode and flowcharts)	20

<b>Part3 -Code</b> <b>Project documentation and updated documentation</b> <b>User manual</b> <b>Code-</b> Functioning library with source code based on analysis, design, logic and classes for each of the transaction types, Functioning client interface and source code based on analysis, design, logic and library <b>Input validation and error handling using controls and code</b>	  5 5 25 25 5
<b>TOTAL</b>	<b>/100</b>