1Course Title	Phase 1 Integration Project
Publication Number	PP1E v1-1 Project 2006-0104
Project Title	Automatic Teller Simulator
Number of Pages	13 (plus cover page)
Released	
Last Revised	January 4, 2006

Integration Project – Phase 1

Project 1: Automatic Teller Simulator

INTRODUCTION

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 six days. This is your opportunity to showcase your creative talents, and your analysis, design, coding, and testing abilities.

OBJECTIVES

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
- Translate design documents and algorithms into source code
- Read from and write to a file
- 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

TIME REQUIRED

You will require 30 hours to complete this project.

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 will provide you with a sample of the current version 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, supervisor). You will also receive the banking data files required for the solution.

To run the demo: The supervisor's login is Korben Dallas, and the password is D001. The customer login and password is any listed in the Pins.dat file except Korben Dallas for example Jerry Cann, C001.

SPECIFICATIONS

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.txt text file having the following structure:

  - name (String)
  - PIN (String – 4 characters)

- 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 in which account balances are stored:

  - account type (1 character)
  - PIN (String – 4 characters)
  - account number (String – 5 characters)
  - account balance (single)

- 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.

- 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. The ATM accepts only transactions for which the amount entered is a multiple of $10. There is no daily maximum amount (apart from the user's account balance).

- For a transfer, the user must enter the amount and the type of transfer (from chequing to savings, or vice versa). This transaction is subject to a maximum $100,000. The system must allow only a transfer from checking to savings, or from savings to checkings.
- 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.
- 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 they can charge the transaction amount to the amount still available in the user's account.
- When the application is initiated at start of each day, the bank's balance money is automatically refilled with up to $5,000 for a maximum of $20,000.

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).
- The supervisor can cause interest to be paid to all savings accounts at the rate of 1% (Balance * rate/365/100).
- 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 system administrator puts in more money in batches of $5,000. There should not be more than $20,000 available in the ATM.

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

REQUIREMENTS:

Model the ATM requirements:

1. Refer to the Class design to determine the things (data entities). Create a data dictionary containing all data entities (you can create a class diagram if you wish). For each entity, list its data elements and the data elements attributes.

    a. data element name, eg. Customer pin

    b. data element's real name or alias

    c. data type

    d. length e.g. How many characters or numbers

    e. range of values e.g. 0 – 100

    Make sure you read through all of the specifications.

2. Draw relationship diagram. Make sure your chart includes minimum and maximum cardinalities.

3. Draw a data flow diagram. All of the data stores needed by the system and all of the main processes identified in the accompanying documentation should be indicated on this diagram. Label the data flows. Any external entity that interacts with the system should be indicated. While all of the main processes described above need to be included in the DFD, please pay particular attention to the data flows flowing in and out of the processes to print the accounts sales report.

Design the Program Logic

You will create structure charts, flowcharts and/or pseudocode to model the logic for the ATM system.

1. Flowcharts and Pseudocode

    Model an algorithm for each of the processes described above using a flowchart or pseudocode (you must provide more than one example of each technique in your project).

    You will be able to copy and modify to save some time. Be very careful when using copy and edit that you make all required changes.

Interpret the Program Logic to Write Source Code

1. Refer to your analysis and design work, and the class design provided to create a class library for ATM.

2. Create a client ATM of at least three Windows forms to use the class library. Refer to the logic you designed.

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: everytime you open a file, 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. If you have time and want to design your classes, you may do so provided they meet the specifications, you meet the deadline and you have your instructor's permission. You will need to decide the data type and scope of each class member: public, private, etc. NOTE: all classes require customized constructors.

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, Get Balance.

Checking

Inherits from Account.

Constant Properties: billFee, maximumBillAmount

Methods: PayBill (amount).

Savings

Inherits from Account.

Constant Properties: interestRate.

Methods: PayInterest()

Bank

Inherits from Account.

Constant Properties: maximumTopUp, refillAmount

Methods: refillATM().

checkingAccounts

This class needs to add a checking account, and to return a checking account (using the index).

savingsAccounts

This class needs to add a savings account, and to return a savings account (using the index).

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).

ATMManager

Properties: bank, customers, checkingAccounts, savingsAccounts, currentAccountBalance

Methods:

ValidateUser(name,pin) returns Boolean

WithdrawChecking(pin, amount) returns Boolean

WithdrawSavings(pin, amount) returns Boolean

DepositChecking(pin, amount) returns Boolean

DepositSavings(pin, amount) returns Boolean

PayBill(pin, amount) returns Boolean

TransferFunds(pin, amount, accountType)

- 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 (B, C or S) found in the Accounts.txt into the object instances. Use it to determine to which collection the account should be added (Bank, checkingAccounts, savingAccounts)

WriteAccounts() returns Boolean

- write the bank, all checkingAccounts, and all savingsAccounts to Accounts.txt.

- add a first character (B, C or S) to the beginning of each account string before writing to Accounts.txt.

Client Application:

The properties and methods are largely determined by the interface design. This assumes only two 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: atmManager, 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: atmManager. Also 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.

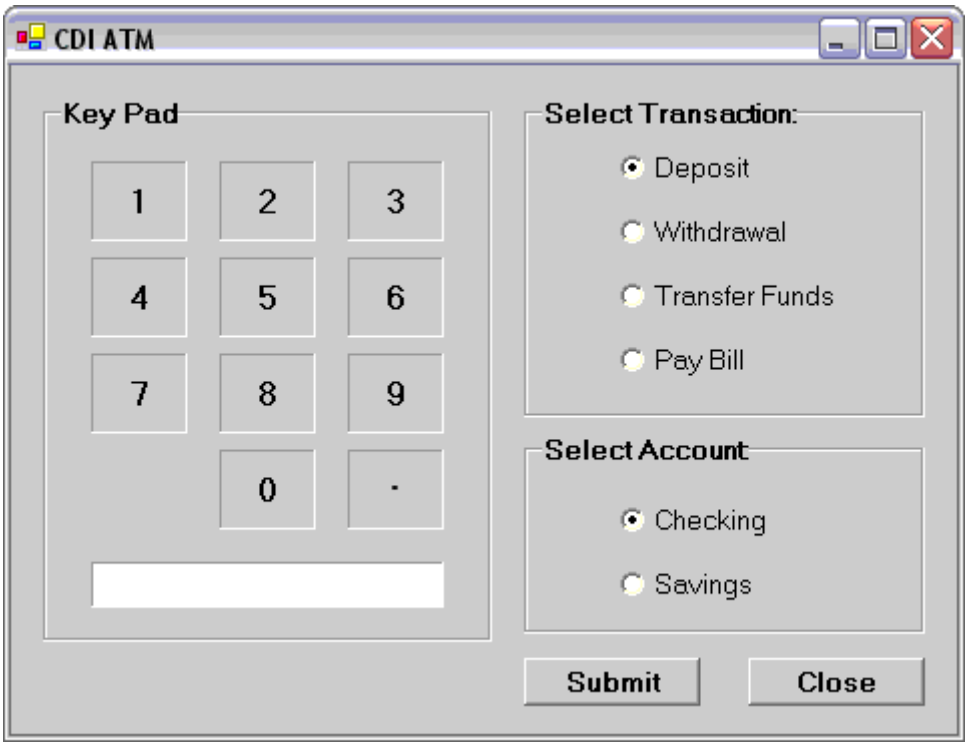
Close: Sets visible property.

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 two forms: a login form, and a main form which becomes visible if the user's name and PIN are found in customers. 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: use a character array and the string split method.)

  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.

- Use a textbox to display 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.

SCHEDULE

Day 1: Analysis and Design of the work to be done

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

Day 2: Logic Required by the Processes

- Analyze the processes to be performed in the application.
- Chart or diagram the classes (things) and their properties and methods (processes). Identify the methods that can be inherited and any that can be overridden.

  * Using UML is not an outcome of this course. The class design information is for your reference.
- Create a flowchart or pseudocode (or both) using Visio to describe the processes within the application.

Days 3 to 5: 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.

Day 6: Solution validation and documentation update

- Validate the solution using test cases.
- Add any missing input validation or error-handling.
- Update the documentation.
- Submit the project.

IF YOU HAVE TIME

- The details of each transaction should be recorded in the Transactions.txt sequential file, as follows:
    - ♦ transaction number (Integer)
    - ♦ transaction date (String)
    - ♦ PIN (String)
    - ♦ user account number (String – 5 characters)
    - ♦ user name (String)
    - ♦ account type (Char – ‘C’ for chequing, ‘S’ for savings, or ‘B’ for bank)
    - ♦ transaction code (Char – ‘D’ for deposit, ‘W’ for withdrawal, ‘T’ for transfer or ‘B’ for bill payment)
    - ♦ transaction amount (single)
    - ♦ account balance (single)
    - ♦ 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.

- Add to the special menu:
    - ♦ daily transactions report

- 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.
- 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.

MARKING SCHEME

Evaluation Elements	% of mark
Analysis and Design (data dictionary or class diagram, relationship diagram, DFD)	15
Logic required by the processes/methods (pseudocode and flowcharts)	20
Functioning library with source code based on analysis, design, logic and classes	25
Functioning client interface and source code based on analysis, design, logic and library	20
Documentation preparation and update	5
Input validation and error handling using controls and code	15
TOTAL	**100**

WHAT TO SUBMIT

Your project must include the following:

- A diskette containing all source code, as well as all other files required for the proper functioning of your application.
- 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
  - all relevant documentation including diagrams, flowcharts, pseudocode
  - the user manual (instruction guide for your instructor)
  - explanations of any additions that you made (where applicable)

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%.