Christian J. Hughes
CIS 501 - David Schmidt
03/12/15

USE CASE REALIZATIONS:

User Interface 1 (Teller):

Summary: "Interactions with the teller: The teller is a person who uses the computerized Teller on a desktop computer. You have some freedom in defining the appearance of the GUI:"

1. *"A customer visits a teller and opens an new customer account by supplying name and address. (We trust the teller to check IDs, etc.) The customer receives a new customer-number."*

-The human teller presses the "Create new Account" button. The state advances.
-Buttons to select which type of account the user would like to open become visible (new customer account or new bank account).
-The human teller selects the "New Customer Account" button. The state advances.
-The Status Label will change to "Please enter the customer's name." A text box below the label will become active, allowing the teller to enter the customer's name.
-If the input is invalid, the state will repeat, and the form will ask for a valid name.
-Else, this information is passed to the controller, where a new customer object is created (and who's name property is now filled). The state advances.
-The Status Label will change to "Please enter an address." A text box below the label will become active, allowing the user to enter the customer's address.
-The controller will not advance repeat state if the input is invalid (should an account already exist with the same name and credentials).
-Else, this information is passed to the controller and customer object (who's address property is now filled).
-The controller requests a new Customer Number from the Account Database.
-This Customer Number is assigned to the customer object, which now contains all relevant information (address, name, Customer Number).
-The Customer object is added to the Customer Database. The state advances.
-The Customer Number is printed to the Status Label for customer use. The "Begin New Session" button is now active.

2. *“A customer who is registered with the bank visits the teller and opens a new account by supplying customer-number. The customer receives a new account number.”*

-The human teller presses the “Create new Account” button. The state advances.
-Buttons to select which type of account the user would like to open become visible (new customer account or new bank account).
-The human teller selects the “New Bank Account” button. The state advances.
-The Status Label will change to “Please enter the customer’s number.” A text box below the label will become active, allowing the teller to enter the customer’s number.
-This information is passed to the controller, which will either report the number is invalid OR it will create a new account.
-If the controller claims the input was invalid, the state will repeat, and the form will prompt the user to enter a valid customer number.
-Else the input was a valid customer number: The controller searches the Customer Database for the customers name. A new Account is created using the given information. The account is added to the Account Database. The state advances.
-The Status Label will print the customer’s new account number. The “Begin New Session” button is now active.

3. *“A customer with a bank-account number visits a teller and deposits money into the account. The account's balance is updated and the customer gets a receipt.”*

-The human teller presses the “Work With an Existing Account” button. The state advances.
-Buttons to select what action the user would like to make with the account are now visible (Deposit, Withdraw, or Balance Inquiry).
-The human teller selects the “Deposit” button. The state advances.
-The Status Label will change to “Please enter the customer’s bank account number.” A text box below the label will become active, allowing the teller to enter the customer’s bank account number.
-This information is passed to the controller, which will either report the number is invalid (either it is not a number, or it is not in the Account Database) OR it will prompt the user for a withdrawal amount. The state will be adjusted accordingly.
-If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid bank account number.

–Else, the input was a valid bank account number: (The controller searched the Account Database for the account) The state is updated.

–The Status Label will change to “Please enter the amount the customer wishes to deposit.” A text box below the label will become active, allowing the teller to enter the customer’s deposit amount.

–This information is passed to the controller (which consults the Account Database), which will either report the amount is invalid OR it will deposit will be added to the account. The state will be adjusted accordingly.

–If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid deposit amount.

–Else the input was a valid deposit amount: A receipt form is created and shown displaying the customer’s Account Number, and updated balance. The “Begin New Session” button is now active.

4. *“A customer with a bank-account number visits a teller and asks for the account's balance. The customer sees/is told the balance.”*

–The human teller presses the “Work With an Existing Account” button. The state advances.

–Buttons to select what action the user would like to make with the account are now visible (Deposit, Withdraw, or Balance Inquiry).

–The human teller selects the “Balance Inquiry” button. The state advances.

–The Status Label will change to “Please enter the customer’s bank account number.” A text box below the label will become active, allowing the teller to enter the customer’s bank account number.

–This information is passed to the controller, which will either report the number is invalid (either it is not a number, or it is not in the Account Database) OR it will return the current balance of the associated account. The state will be adjusted accordingly.

–If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid bank account number.

–Else the input was a valid bank account number: (The controller searched the Account Database for the account) The state is updated.

–The Status Label will now display the balance of the customer’s account. The “Begin New Session” button is now active.

5. *“A customer with a bank-account number visits a teller and does a withdrawal. The customer receives cash and the account's balance is updated.”*

-The human teller presses the “Modify an Existing Account” button. The state advances.

-Buttons to select what action the user would like to make with the account are now visible (Deposit, Withdraw, or Balance Inquiry).

-The human teller selects the “Withdraw” button. The state advances.

-The Status Label will change to “Please enter the customer’s bank account number.” A text box below the label will become active, allowing the teller to enter the customer’s bank account number.

-This information is passed to the controller, which will either report the number is invalid (either it is not a number, or it is not in the Account Database) OR it will prompt the user for a withdrawal amount. The state will be adjusted accordingly.

-If the controller claims the input was invalid, the state repeats, the form will prompt the user to enter a valid bank account number.

-Else the input was a valid bank account number: (The controller searched the Account Database for the account) The state is updated.

-The Status Label will change to “Please enter the amount the customer wishes to withdraw.” A text box below the label will become active, allowing the teller to enter the customer’s withdrawal amount.

-This information is passed to the controller (which consults the Account Database), which will either report the amount is invalid (either it is not a number, or the user cannot withdraw the desired amount) OR it will be withdrawn from the account. The state will be adjusted accordingly.

-If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid deposit amount.

-Else the input was a valid withdrawal amount: The controller returns the new state as well as the updated balance of the account. The TellerForm is updated to reflect this new information. The “Begin New Session” button is now active.

6. “*A customer with a bank-account number visits a teller and closes the account.”*

-The human teller presses the “Close an Account” button. The state advances.

-Buttons to select what kind of account the user would like

to close are now visible (Close Customer Account, or Close Bank Account).

-The human teller selects the “Close Bank Account” button. The state advances.

-The Status Label will change to “Please enter the customer’s bank account number.” A text box below the label will become active, allowing the teller to enter the customer’s bank account number.

-This information is passed to the controller, which will either report the number is invalid (either it is not a number, or it is not in the Account Database) OR it will close the given bank account. The state will be adjusted accordingly.

-If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid bank account number.

-Else the Account number was found in the Account Database: The controller consults the database for the account balance. The full balance is withdrawn, and a receipt with the full withdrawal amount is displayed to the customer. The account is then deleted from the Account Database, removed from the Customers list of accounts, and deleted. The “Begin New Session” button is now active.

7. “*A customer with a bank-account number visits a teller and closes the customer account.”*

-The human teller presses the “Close an Account” button. The state advances.

-Buttons to select what kind of account the user would like to close are now visible (Close Customer Account, or Close Bank Account).

-The human teller selects the “Close Customer Account” button. The state advances.

-The Status Label will change to “Please enter the customer’s bank account number.” A text box below the label will become active, allowing the teller to enter the customer’s bank account number.

-This information is passed to the controller, which will either report the number is invalid (either it is not a number, or it is not in the Customer Account Database) OR it will close the given customer account. The state will be adjusted accordingly.

-If the controller claims the input was invalid, the state repeats, and the form will prompt the user to enter a valid bank account number.

-Else the Account number was found in the Account Database: The controller consults the Customer Database for all

of accounts owned by the customer. The full balance of each bank account is withdrawn, and a receipt with the full withdrawal amount for each bank account is displayed to the customer. The accounts are then deleted from the Account Database, removed from the Customers list of accounts, and deleted. The Customers Account is then deleted. The "Begin New Session" button is now active.

User Interface 2 (ATM):

Summary: "Interactions with an ATM: The ATM is a Form that looks and behaves like the ones you've seen on the street: you enter your credentials using a key pad (buttons), choose from a menu of actions, and complete the transaction:"

1. *"A customer with a bank-account number uses an ATM to log in and obtain the account's balance. The customer sees the balance."*

-The display menu initially says "Hello! To log in: Please enter the account number of the bank account you wish to access."
-The customer enters their account number on the keypad (containing number 0-9), and presses the enter key. A text box below the menu populates with the users input as they type.
-The AtmPresenter looks through the Account Database to see whether or not the account exists.
-If the account does not exist, the state repeats, and the menu updates (saying that the user entered an invalid bank account number).
-Else, the account does exist. The state advances.
-The display menu updates giving the user two options: "Press (1) for a balance inquiry. Press (2) to make a withdrawal."
-The customer presses the "1" button. The presenter retrieves the account balance, and displays it in the display menu. The state advances
-Below the balance, the display menu has two more options: "Press (1) to make a withdrawal. Press (2) to log out."
-Pressing the "1" button will lead the customer to the below use case. Pressing the "2" button will move the ATM to the LoggedOut state.

2. *"A customer with a bank-account number uses an ATM to log in and withdraw cash. The customer receives cash (we won't implement the cash!) and the account's balance is updated."*

-The display menu initially says "Hello! To log in: Please enter the account number of the bank account you wish to access."

-The customer enters their account number on the keypad (containing number 0-9), and presses the enter key. A text box below the menu populates with the users input as they type.

-The AtmPresenter looks through the Account Database to see whether or not the account exists.

-If the account does not exist, the state repeats, and the menu updates (saying that the user entered an invalid bank account number).

-Else, the account does exist. The state advances.

-The display menu updates giving the customer two options: “Press (1) for a balance inquiry. Press (2) to make a withdrawal.”

-The customer presses the “2” button. The state is updated. The display menu now displays the following: “Please enter the amount that you would like to withdraw in the from “00.00.”

-The Customer enters the withdraw amount using the keypad, and presses the enter key.

-If the input is invalid (or the customer attempts to withdraw more money than is present in the account), the state repeats, and the menu updates (saying that the user entered an invalid currency amount).

-Else, the input is valid. The display menu displays the amount withdrawn, and AtmPresenter contacts the model to update the balance of the account. The updated account balance is displayed.

-In addition to the above withdrawal information, the display menu will list the following as the only additional option: “Press (2) to log out.”

-Pressing the “2” button will move the ATM to the LoggedOut state.

COMPONENT EXTRACTION:

User Interface 1 (Teller):

Teller *View*:

Components:

-*TellerForm Class* - The view for the system, containing input and output form components.

-“*Create new account” button* - Enables *“New Customer Account” button*, and *“New Bank Account” button*. Updates State.

-*“Work With an Existing Account” button* - The *Status Label* prompts the user for a bank account number. If it is a valid number, Enables *“Deposit” button*, “*Withdraw” button*, and *“Balance Inquiry” button*. Updates State.

-*“Close an Account” button* - Enables *“Close Customer Account” button*, and *“Close Bank Account” button*. Update State.

-*“Begin New Session” button* - Upon clicking, the state is reset back to start. Is activated once a set of operations is complete.

-*Status Label* - To tell the user what kind of input is required/information on

current state (based on System state).

-*Static Title Label* - Just a banner with the name of the banking system.

-*Input Text Box* - A text box for the teller to enter customer/account information.

-*Status field* - Keeps track of the current state of the system, which allows the form to determine which buttons to have enabled. Implements the *TellerStatus* enumeration listed in Teller->Controller->Components.

-*ReceiptForm Class* - Pops up whenever the customer requires a receipt.

-*Static Receipt Label*

-*Account Number Label*

-*Updated Balance Label*

Methods:

-*TellerForm Class*

-Click event methods for all of the above buttons. They will implement delegate methods passed in by the controller.

-*void updateForm(TellerStatus state)*

-*ReceiptForm Class*

-*ReceiptForm(int AccountNumber, double balance)* - Constructor gets the necessary information for printing a receipt for the customer.

Teller *Controller:*

Components:

-*TellerController Class* - The controller for the Teller form. It interacts with the databases, and passes the necessary information to the form. It is connected to the form via delegates.

-*List<ObserverState> registry* - registers updates to the form, and tells it to repaint itself.

-*Customer currentCustomer*

-*Account currentAccount*

-*TellerStatus state*

-*TellerStatus Enumeration* - Keeps track of the state of the controller.

-*Start*

-*ClosingAnAccount*

-*AttemptClosingCustomerAccount*

-*ClosedCustomerAccount*

-*AttemptClosingBankAccount*

-*ClosedBankAccount*

-*CreatingNewAccount*

-*AttemptCreatingNewCustomerAccount*

-*NeedValidAddress*

-*CreatedNewCustomerAccount*

-*AttemptCreatingNewBankAccount*

-*CreatedNewBankAccount*

-*WorkingWith AnExistingAccount*

-*Depositing*

-*NeedValidDepositAmount*

-*DepositSuccess*

-Withdrawing
-NeedValidWithdrawalAmount
-WithdrawalSuccess
-BalanceInquiry
-DisplayingBalanceInquiry

Methods:
-TellerController Class
-TellerController(CustomerDatabase custDatabase, AccountRecordsDatabase acctDatabase) - Constructor that retains handles to the two models.
-TellerStatus handleCloseBankAccount(int accountNumber)
-TellerStatus handleCloseCustomerAccount(int accountNumber)
-Tuple<TellerStatus, int> handleNewCustomerAccount(string name, string address) - Returns Customer Number (so that it can be displayed to the user on the form). Adds the customer to the *CustomerDatabase*. Status ONLY advances if successfully created.
-Tuple<TellerStatus, int> handleNewBankAccount(int customerNumber) - Returns new account number (after adding to the AccountDatabase), as well as the updated status IF the account is created successfully. Else, the status remains the same, and the user is asked to supply a valid customer number.
-Tuple<TellerStatus, double> handleDeposit(int accountNumber, double depositAmount) - Returns the new status, as well as the new balance of the account.
-Tuple<TellerStatus, double> handleWithdraw(int accountNumber) - Returns the new status, as well as the new balance of the bank account.
-Tuple<TellerStatus, double> handleBalanceInquiry(int accountNumber) - Returns the new status, as well as the current balance of the bank account.
-private double getAccountBalance(double amount)
-private double completeWithdrawal(double amount)
-private double completeDeposit(double withdrawal)
-void printReceipt(int accountNumber, double balance) - "Prints" (shows form) receipt for the customer.
-void register(ObserverState observe)
-void updateViews(List<observer> reg)

Teller *Model:*
Components:
-*Customer Class* - Contains the fields necessary for a single customer.
-string customerName
-string customerAddress
-int customerNumber
-int[] accountNumbers
-*CustomerDatabase Class* - Database containing all of the customers.
-Dictionary<int, Customer> - A Dictionary of Customers indexed by customer numbers.
-int latestCustomerNumber
-*Account Class* - Contains the fields for constructing a single account.
-int accountNumber

-int customerNumber
-double accountBalance
-*AccountRecordsDatabase Class* - Database containing all of the accounts
-*Dictionary<int, Account>* - A Dictionary of Accounts indexed by account numbers.
-*int latestAccountNumber*

Methods:
-*Customer Class*
-string toString()
-void removeBankAccount(int accountNumber)
-*CustomerDB Class : CustomerDatabase*
-Customer lookUpCustomer(int customerNumber)
-bool addNewCustomer(Customer newCustomer)
-bool removeCustomer(int customerNumber)
-bool removeBankAccountFromCustomer(int accountNumber)
-int returnNewCustomerNumber()
-*Account Class*
-bool deposit(double amountToDeposit) - Returns true if the amount was successfully deposited.
-bool withdraw(double amountToWithdraw) - Returns true if the amount was successfully withdrawn.
-*double returnBalance()*
-*AccountDB Class : AccountDatabase*
-int lookUpAccountReturnCustomerNumber(int accountNumber) - returns customer number.
-double depositIntoAccount(int accountNumber, double depositAmount) - returns the account balance after the deposit.
-double withdrawFromAccount(int accountNumber, double withdrawAmount) - returns the account balance after the withdrawal.
-double returnAccountBalance(int accountNumber) - returns the current account balance.
-void addNewAccount(Account newAccount)
-void removeAccount(int accountNumber)
-int returnNewAccountNumber()

User Interface 2 (ATM):
ATM *View*:
Components:
-AtmForm Class - The form that handles all ATM input and Output.
-"1" Button
-"2" Button
-"3" Button
-"4" Button
-"5" Button
-"6" Button
-"7" Button

-"8" Button
-"9" Button
-"0" Button
-"Enter" Button
-"Cancel" Button
-Display Menu Label
-Customer Input Textbox
-Static Title Label
-AtmStatus state

Methods:

-Click event methods for all of the above buttons. They will implement delegate methods passed in by the controller.

-*void updateForm(AtmStatus state)*

ATM *Controller:*

Components:

-AtmPresenter Class

-AtmStatus field

-AtmStatus Enumeration

-Start
-LoggedIn
-ChoseBalanceInquiry
-ChoseWithdrawalNeedAmount
-CompletedWithdrawal
-LoggedOut

Methods:

-AtmPresenter Class

-Status handleStart(int accountNumber)

-Tuple<Status, double> handleLoggedIn(string buttonPressed) - Returns the new Status, as well as the account balance, should the balance inquiry button be pressed.

-Status handleChoseBalanceInquiry(string buttonPressed)

-Tuple<Status, double> handleChoseWithdrawalNeedAmount(double withdrawalAmount)

-Status handleCompletedWithdrawal(string buttonPressed)

-private double checkBalance()

-private double completeWithdrawal(double amount)

ATM *Model:*

Components:

-Same model components as the Teller.

Methods:

-Same model methods as the Teller.

^ADDITIONAL TWEAKS MADE TO THE ABOVE COMPONENT EXTRACTIONS IN THE CLASS DIAGRAM^