Juan Miguel Brocca

Advanced programming  Whitecliffe

assignment 2 - bank App

**A description of the class, listing attributes and methods (with their signatures)**

There are two classes:

Accounts and

Customers

**Accounts Class:**

Attributes:

* Id (int): An identifier for the account.
* Balance (double): The current balance of the account.
* Overdraft (double): The overdraft limit for the account.
* Fee (double): The fee charged for a failed transaction.
* Interest (double): The interest rate for the account.

Methods:

* Withdraw(double Amount) (abstract): A method that must be implemented by any derived class. This method is used to withdraw a certain amount from the account. If the withdrawal is successful, the balance will be reduced by the withdrawal amount. If the withdrawal fails (i.e., the balance is not sufficient), a FailedWithdrawalException will be thrown.
* Deposit(double Amount) (abstract): A method that must be implemented by any derived class. This method is used to deposit a certain amount into the account. If the deposit is successful, the balance will be increased by the deposit amount.
* CalculateInterest() (abstract): A method that must be implemented by any derived class. This method is used to calculate the interest earned on the account balance.

The ***Account*** class is an abstract class, which means that it cannot be instantiated directly. Instead, it serves as a base class for other classes (such as Omni, Everyday, and Investment) that inherit from it and implement its abstract methods. These derived classes can be instantiated, and they inherit the attributes and methods of the Account class.

**Customers Class:**

The Customer class is an abstract class, meaning that it cannot be instantiated and serves only as a base class for other classes. It has the following attributes:

* Id: an integer representing the customer's identification number.
* Name: a string representing the customer's name.
* Contact\_details: a string representing the customer's contact details.
* Fee: a double representing the fee charged to the customer. It is in public access for testing purposes.
* Accounts: a list of Account objects representing the accounts owned by the customer.

The Staff class inherits from Customer and represents the bank staff. It has the following constructor:

* Staff(int Id, string Name, string Contact\_details): initializes the Id, Name, and Contact\_details attributes with the given values, and sets the Accounts attribute to an empty list. It also sets the Fee attribute to 50.0.

The Staff class also overrides the ToString method to return the name followed by "Staff".

The Client class also inherits from Customer and represents a bank client. It has the following constructor:

* Client(int Id, string Name, string Contact\_details): initializes the Id, Name, and Contact\_details attributes with the given values, and sets the Accounts attribute to an empty list. It also sets the Fee attribute to 0.0.

The Client class also overrides the ToString method to return the name followed by "Customer".

***The Controller Classes***

**The Customer Controller Classes**

The CustomersController class is responsible for managing the list of customers in the bank application. It has the following attributes and methods:

Attributes:

* customers: A list of Customer objects.

Methods:

* CustomersController(): Constructor that initializes the customers list.
* getList(): Returns the customers list.
* AddCustomer(Customer customer): Adds a Customer object to the customers list.
* RemoveCustomer(Customer customer): Removes a Customer object from the customers list and returns a boolean indicating whether the removal was successful.
* getAccounFromCustomer(Customer customer): Returns a list of Account objects associated with a particular Customer.
* PopulateList(ListBox listbox): Clears the listbox and populates it with the names of all Customer objects in the customers list.

**The Accounts Controller Classes**

The AccountsController class manages the accounts of a specific Customer. The attributes and methods (with their signatures) are:

* private Customer customer: the customer whose accounts will be managed
* public AccountsController(Customer customer): constructor that sets the customer attribute
* public void RegisterAccount(Account account): adds an Account to the customer's account list
* public bool UnregisterAccount(Account account): removes an Account from the customer's account list. Returns false if the account has a balance greater than 0 and cannot be closed.
* public void ListAccounts(ListBox listbox): lists the customer's accounts in a ListBox control
* public void DepositIntoAccount(Account account, double amount): deposits a double amount into an Account. Displays an error message if account is null.
* public void WithdrawlFromAccount(Account account, double amount): withdraws a double amount from an Account. Displays an error message if account is null.

**Explanation of how this design will meet the requirement of managing customer information for potentially different user interfaces.**

The design presented in this code is suited for managing customer information for different user interfaces. This is because it separates the concerns of managing the data and interacting with the user, which is a good practice in software engineering.

The Customer class defines the properties that are common to all customers, such as their ID, name, contact details, fee, and a list of accounts they own. The Staff and Client classes inherit from the Customer class and provide specific implementations for staff and client customers. This approach allows for easy extension of the system if other types of customers are added in the future.

The CustomersController class is responsible for managing the list of customers. It provides methods to add and remove customers from the list, as well as to retrieve a list of customers and the accounts associated with a particular customer. This class is independent of the user interface and can be used by different interfaces, such as a web application or a desktop application.

The AccountsController class is responsible for managing the accounts associated with a customer. It provides methods to register and unregister accounts, as well as to deposit and withdraw money from accounts. This class is also independent of the user interface and can be used by different interfaces.

In terms of the user interface, the ListBox control is used to display the list of customers and accounts. The PopulateList method of the CustomersController class is called to populate the ListBox with the list of customers. The ListAccounts method of the AccountsController class is called to populate the ListBox with the accounts associated with a selected customer.

This design allows for different user interfaces to be created, such as a web application or a desktop application, which can use the same data management classes. This separation of concerns makes the code easier to maintain and modify, as changes made to the user interface will not impact the underlying data management classes.