Test 02

Centennial College

COMP 123 – Programming II

TEST 2

Overview

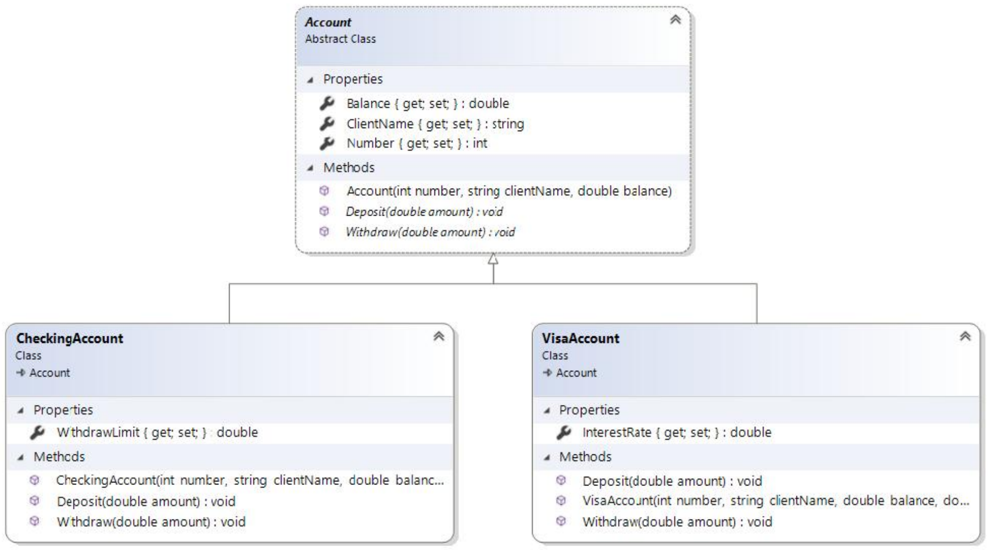
You will be creating a Windows Application that will allow bank employees to create new accounts for clients. The account can be a Checking Account or a Visa Account. After the account gets created it is added to a List of accounts.

Details

STEP 1 Project

1 - Create a Windows Application .NET Framework project and solution called "**AccountsApp**".

STEP 2 Implement the following 3 classes in the project:


Account
Abstract Class
Properties
Balance { get; set; } : double
ClientName { get; set; } : string
Number { get; set; } : int
Methods
Account(int number, string clientName, double balance)
Deposit(double amount) : void
Withdraw(double amount) : void
CheckingAccount
Class
Account
Properties
WithdrawLimit { get; set; } : double
Methods
CheckingAccount(int number, string clientName, double balanc...
Deposit(double amount) : void
Withdraw(double amount) : void
VisaAccount
Class
Account
Properties
InterestRate { get; set; } : double
Methods
Deposit(double amount) : void
VisaAccount(int number, string clientName, double balance, do...
Withdraw(double amount) : void

Account Class

3 Marks 2.1 - This class is an *Abstract* class used to define general information about accounts. It has two derived classes: **CheckingAccount** and **VisaAccount**

2.2 - `Number` – this integer property represents the Account Number

2.3 - `Balance` – this double property represents the Account Balance.

2.4 - `ClientName` – this string property represents the name of the client who's the owner of the account.

5 Marks 2.5 - `public Account(int number, string clientName, double balance)` – This constructor takes 3 arguments and populates the properties of the Account;

3 Marks 2.6 - `public abstract void Deposit(double amount)` – This is an abstract method.

3 Marks 2.7 - `public abstract void Withdraw(double amount)` – This is an abstract method.

CheckingAccount Class

3 Marks 2.8 - This class is derived from the abstract **Account** class.

2.9 - `WithdrawLimit` – this double property indicates the maximum daily amount the client can withdraw from the account.

5 Marks 2.10 - `public CheckingAccount(int number, string clientName, double balance, double withdrawLimit)` – This constructor takes 4 arguments and populates the properties of the CheckingAccount;

5 Marks 2.11 - `public override void Deposit(double amount)` – This is a concrete method. It should add the amount to the Account Balance and deduct a 30¢ (thirty cents) transaction fee from the Account Balance as well. The 30¢ (thirty cents) fee should be coded as a class **constant**.

5 Marks 2.12 - `public override void Withdraw(double amount)` – This is a concrete method. It should deduct the amount from the Account Balance and deduct a 30¢ (thirty cents) transaction fee from the Account Balance as well. The 30¢ (thirty cents) fee should be coded as a class **constant**, the same used for the Deposit method.

VisaAccount Class

3 Marks 2.13 - This class is derived from the abstract **Account** class.

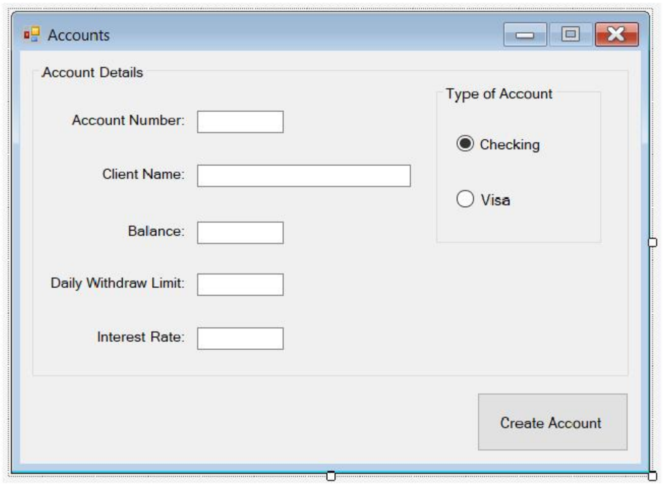
2.14 - `InterestRate` – this double property indicates the monthly interest rate the account pays to the client.

5 Marks 2.15 - `public VisaAccount(int number, string clientName, double balance, double interestRate)` – This constructor takes 4 arguments and populates the properties of the VisaAccount;

5 Marks 2.16 - `public override void Deposit(double amount)` – This is a concrete method. It should add the amount to the Account Balance and deduct a $1.5 transaction fee from the Account Balance as well. The $1.5 fee should be coded as a class **constant**.

5 Marks 2.17 - `public override void Withdraw(double amount)` – This is a concrete method. It should deduct the amount from the Account Balance and deduct a $1.5 transaction fee from the Account Balance as well. The $1.5 fee should be coded as a class **constant**, the same used for the Deposit method.

STEP 3 Create the following GUI:



Details about how the GUI should work:

- 5 Marks - 3.1 - Provide meaningful names to the GUI controls.
- 5 Marks - 3.2 - Declare and create a private field called **accounts** in the **Form** class of type **List<Account>**. This list will store the **Account** object once it gets created.
- 6 Marks - 3.3 - If **"Checking"** radio button is selected, the **Daily Withdraw Limit** should be enabled and the **Interest Rate** should be disabled because Checking accounts only have Daily Withdraw Limit, not Interest Rate;
- 6 Marks - 3.4 - If **"Visa"** radio button is selected, the **Daily Withdraw Limit** should be disabled and the **Interest Rate** should be enabled because Visa accounts only have Interest Rate, not Daily Withdraw Limit;
- When the user clicks on **"Create Account"** the following needs to happen:
  - 10 Marks 3.5 - A **new account** will be created, and the information filled in the GUI will be used to set the account properties;
  - 6 Marks 3.6 - Use a **try..catch** block to handle any invalid input, like when the user enters text in the Balance or on another numeric field – Show a message to let the user knows about the problem;
  - 5 Marks 3.7 - Add the newly created account to the **accounts** List. This List should store the list of all accounts getting created;
  - 5 Marks 3.8 - Clear all textboxes and set the selected type of account to "**Checking";**
  - 2 Marks 3.9 - Use **MessageBox.Show()** to show the total number of accounts stored so far in the **accounts** List;

STEP 4 **BONUS** – **<u>This is optional</u>**, and it is considered Bonus 8 if you choose to do it

4 – Create a search feature in the GUI that will allow the user to enter an Account Number and the code will search for the account in the account List and if found, it will populate the GUI with the account information that has been found.

STEP 5 **UPLOAD INSTRUCTIONS:**

Once you are done with your project, zip it, name it with your username (for instance **301055221.zip**) and upload to eCentennial under Assessments / Assignments / Test 02.

IMPORTANT: The following points will also be considered when grading:

1 – Make sure your project is compiling and running. Tests with code that is not running will get **<u>20 Marks</u>** deduction automatically.

2 – Apply the naming conventions for variables, methods objects and classes:
- *variable names, parameters and fields* – use camelCasing
- *classes, methods, properties, enumerations* – use PascalCasing
- *constants:* SNAKE_UPPERCASE

Note: You might be required to demonstrate and answer questions about the test one-on-one with the instructor.