CSCI 1250 Lab 9 – BankAccount Class

You are asked to create a Bank Account class. Please use the UML Diagram, attached, and the description below to help create the class requested. Do not deviate from the UML Diagram. Create a class to test the BankAccount class. This class would create an object or objects of the BankAccount class and test out the code.

A bank account can be of the following types:

- A savings account. Savings accounts must always have a $5.00 minimum balance in the account and must be opened with at least $5.00. Savings accounts receive 2.4% interest. If a customer does not open the account with $5.00, then it will resort to a checking account. Additionally, a savings account withdrawal cannot exceed $500.

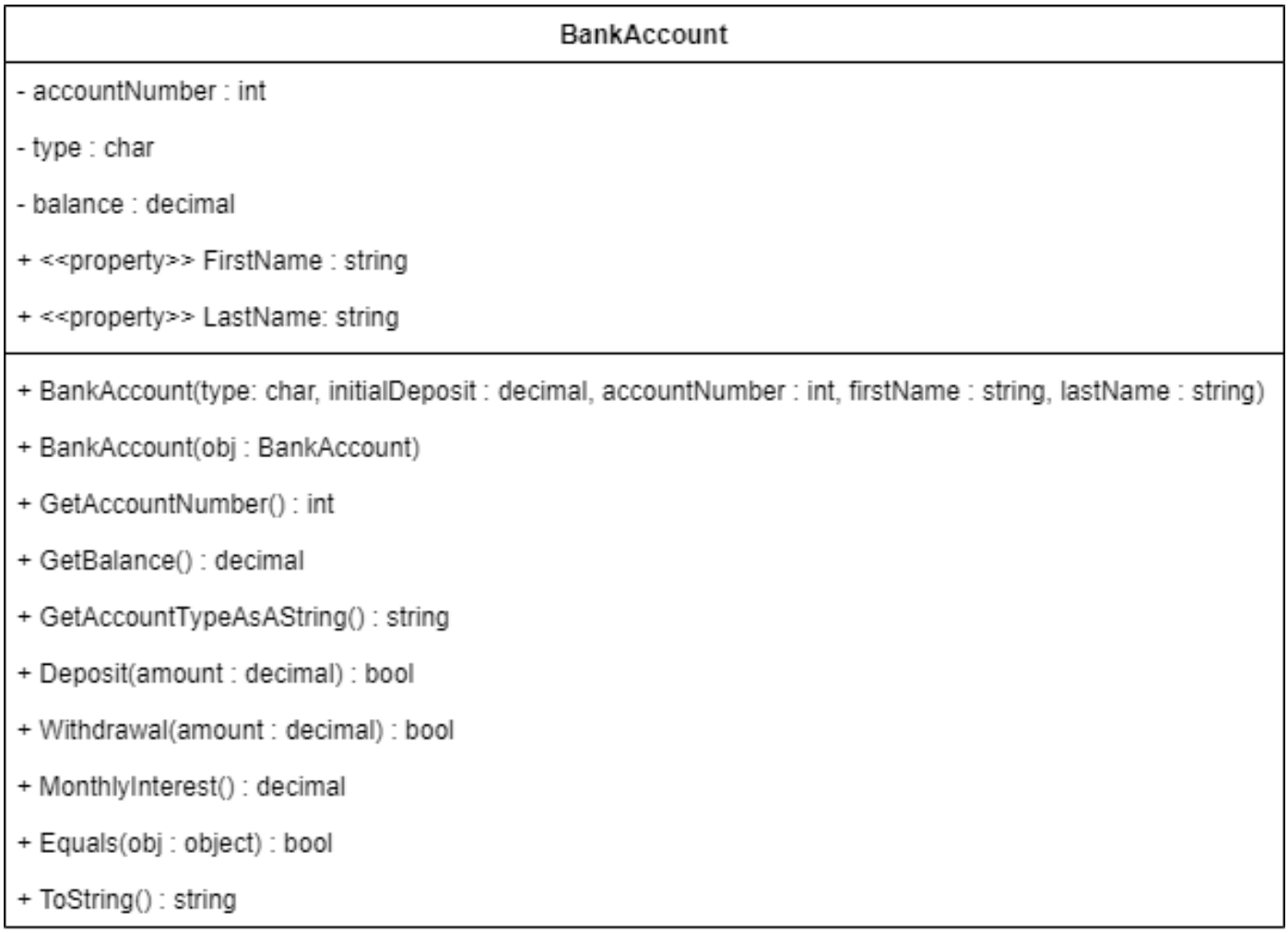
- A checking account. Checking accounts do not need to keep a minimum balance in the account, but only receive .5% interest. Checking accounts do not have any restrictions.

- Money Market Account (MMA) must always have a $1000 minimum balance in the account and must be opened with at least $1000. MMA receives 8% interest. If a customer does not open the account with $1000, then it will resort to a checking account. Additionally, a money market account withdrawal cannot exceed $250.

Other Important information and hints:

- Withdrawals cannot exceed the amount the customer has available, or the account restrictions, or if the withdraw will make them go below the account requirements. (e.g., savings must be $5.00 min.)
- The ***MonthlyInterest()*** will add simple interest to the ***balance*** based on the current ***balance***. Interest varies based on the account ***type.*** Additionally, this method should return the amount of interest added to the balance.
- ***Deposit()*** and ***Withdrawal()*** return a Boolean. If something goes wrong, return a false. An example of something going wrong could be if you try to deposit a negative number. Think of things that might go wrong and check for it. **VALIDATION (or making sure everything is working) IS IMPORTANT!**
- Once an account is given a type and a number, it can never change. (That is why we do not have a **SetAccountType() or SetAccountNumber().**)
- ***GetAccountTypeAsAString()*** does not return a character, it returns a String representation of what the ***type*** attribute means.
- Accounts are equal if the account number, type, and balance are the same.
- The ***ToString()*** should look like this given the objects information:

Name: Mathew Desjardins
Account Number: 1
Account Type: Money Market Account
Current Balance: $25.90
Current Monthly Interest Rate: 8.00%


BankAccount
- accountNumber : int
- type : char
- balance : decimal
+ <<property>> FirstName : string
+ <<property>> LastName: string
+ BankAccount(type: char, initialDeposit : decimal, accountNumber : int, firstName : string, lastName : string)
+ BankAccount(obj : BankAccount)
+ GetAccountNumber() : int
+ GetBalance() : decimal
+ GetAccountTypeAsAString() : string
+ Deposit(amount : decimal) : bool
+ Withdrawal(amount : decimal) : bool
+ MonthlyInterest() : decimal
+ Equals(obj : object) : bool
+ ToString() : string

In Program.cs – test the following functionalities.

1. Prompt the user for an initial deposit and store it in a variable
    a. Make the deposit such an amount that the account type is MMA
2. Prompt the user for their first and last name, and store them in variables
3. Determine the account type (based on the initial deposit entered by the user) – this will be a char
    a. Make sure you assign the appropriate character representing the account type
4. Create a new bank account object and assign its location to a reference variable
    a. Pass in the values obtained from the user to the parameterized constructor
5. Deposit a negative amount – should see an error message
6. Withdraw a negative amount – should see an error message
7. Withdraw an amount greater than the account's balance – should see an error message
8. Withdraw a valid amount – display the account using ToString
9. Deposit a valid amount – display the account using ToString
10. Calculate the account's new balance after 4 months – display the account using ToString
11. Create a new account object using the copy constructor of BankAccount – Display using ToString
12. Change the last name property of the new account to a different value
13. Check if the two account objects are equal – should be true after the last name change
14. From the new account object, withdraw a valid amount
15. Check if the two account objects are equal using Equals – should be false after the withdrawal