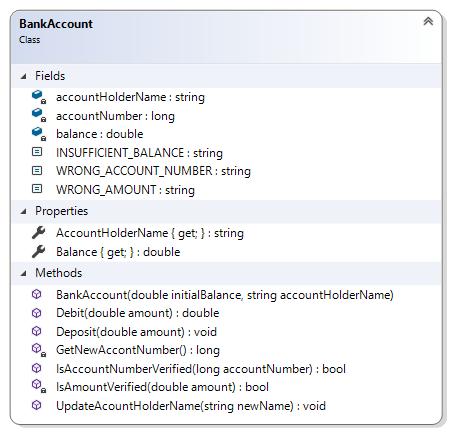
**Testing Project**

In this project you are required to implement the following C#.Net class according to the requirements and design given below. The class simulates typical banking operations.

You are also required to implement C# test scripts and design unit test cases and methods to achieve 100% code and decision coverage and apply proper test generation techniques where applicable. Your tests must ensure the given class diagram is implemented correctly, and method signatures are fully met.



This class simulates a typical bank account with the following attributes, operations, and properties. Please ensure the consistency of their accessibility with the given class diagram (private or public):

**Attributes and constants:**

private double balance;

private String accountHolderName;

private long accountNumber;

public const String WRONG\_ACCOUNT\_NUMBER = "Wrong account number generated";

public const String WRONG\_AMOUNT = "Cannot withdraw or deposit zero or negative amount.";

public const String INSUFFICIENT\_BALANCE = "Insufficient balance";

**Properties:**

1. *AccountHolderName*: that returns the value of *accountHolderName*
2. *Balance*: that returns the value of *balance*

**Methods:**

1. *BankAccount():* this is the constructor of the class that receives the initial balance and account holder name and assigns them to the associated class attributes.

It also calls *GetNewAccontNumber()* to receive a new account number. Nevertheless, before assigning this new account number to *accountNumber*, an accountNumber validation is performed via *IsAccountNumberVerified()* to ensure the newly generated account number follows the bank policies given below.

1. *GetNewAccontNumber():* This private method generates a random number and returns it as account number. The account number follows these policies:
2. Account number must be 8 digits
3. Account number must start with 32

Any account number must follow the above policies. No verification of account number is performed inside this method.

1. *IsAccountNumberVerified():* a public method that receives an account number and ensures the aforementioned account number policies are followed. If so, it returns *true*, otherwise an [*ArgumentException*](https://msdn.microsoft.com/en-us/library/system.argumentexception(v=vs.110).aspx) is thrown with the message provided by WRONG\_ACCOUNT\_NUMBER constant and *false* is returned.
2. *UpdateAcountHolderName():* A public method that receives the new account holder name and updates *accountHolderName* attribute.
3. *IsAmountVerified():* it is a private method called when debit or credit operations are requested. It receives the transaction amount and checks if the requested amount is not zero or negative and returns true or false accordingly.
4. *Deposit():* it is a public method that updates *balance* with the deposited amount. However, the method checks beforehand for any invalid amount via *IsAmountVerified()* and throws an [*ArgumentOutOfRangeException*](https://msdn.microsoft.com/en-us/library/dsyf5bb2(v=vs.110).aspx) with *WRONG\_AMOUNT* message if the verification fails.
5. *Debit():* it is a public method that updates *balance* with the debited amount and returns the debited amount. Nevertheless, before updating *balance*, the following two checks need to be performed:
6. Check for wrong amount via *IsAmountVerified()* and throw [*ArgumentOutOfRangeException*](https://msdn.microsoft.com/en-us/library/dsyf5bb2(v=vs.110).aspx)if necessary.
7. Check for insufficient balance and throw[*ArgumentOutOfRangeException*](https://msdn.microsoft.com/en-us/library/dsyf5bb2(v=vs.110).aspx)with *INSUFFICIENT\_BALANCE* message if necessary.

**Testing**

The test schedule is provided in the following table. You need to design a test method for each of the following tests. Each test method identifier must start with *ID\_num\_ -* for example, *ID\_15\_TestMethodDebit\_BVAInValidAmount()*. If designed correctly, 100% code and decision coverage will be achieved.

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Test Subject | Test Type | Description |
|  | *IsAccountNumberVerified()* | BVA[[1]](#footnote-1) Valid | A valid account number is generated (min) |
|  | *IsAccountNumberVerified()* | BVA Valid | A valid account number is generated (min+1) |
|  | *IsAccountNumberVerified()* | BVA Valid | A valid account number is generated (max-1) |
|  | *IsAccountNumberVerified()* | BVA Valid | A valid account number is generated (max) |
|  | *IsAccountNumberVerified()* | Invalid | An invalid account number that invalidates account number policy *a* |
|  | *IsAccountNumberVerified()* | Invalid | An invalid account number that invalidates account number policy *b* |
|  | *IsAccountNumberVerified()* | BVA Invalid | An invalid BVA test case for lower end of account number |
|  | *IsAccountNumberVerified()* | BVA Invalid | An invalid BVA test case for upper end of account number |
|  | *UpdateAcountHolderName()* | Valid | A valid test case that verifies *accountHolderName* is property updated |
|  | *Deposit()* | BVA Valid | Verify a valid deposit operation and check the balance afterwards (min) |
|  | *Deposit()* | BVA Valid | Verify a valid deposit operation and check the balance afterwards (min+1) |
|  | *Deposit()* | BVA Invalid | Perform a deposit with zero amount |
|  | *Deposit()* | BVA Invalid | Perform a deposit with negative amount |
|  | *Debit()* | Valid | Verify a valid debit operation and check the balance and debited amount afterwards |
|  | *Debit()* | BVA Invalid | Perform a debit with zero amount |
|  | *Debit()* | BVA Invalid | Perform a debit with a negative amount |
|  | *Debit()* | InValid | Perform a debit that triggers insufficient balance |

**Assessment conditions**

* Individual assignment
* Open book
* No time limits (please refer to the due date)

1. [↑](#footnote-ref-1)