Overview

For this task, you will pick up from the Bank Account program and add transaction classes which will perform the deposit, withdraw and transfer operations on the bank account.

Submission Details

Submit the following files to OnTrack.

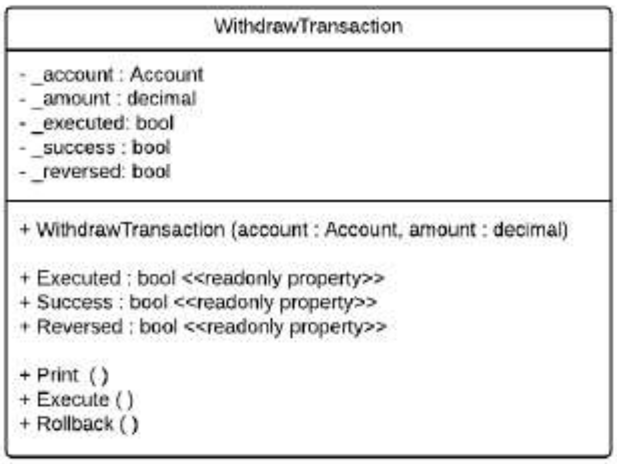
- Your program code (*Program.cs*, *Account.cs*, *WithdrawTransaction.cs*, *DepositTransaction.cs*, *TransferTransaction.cs*)
- A screen shot of your program running

Instructions

In this task, you're going to be created 3 new classes: WithdrawTransaction, DepositTransaction, and TransferTransaction. Objects of these classes can then be used to retain a history of the transactions that have occurred. We will add this transaction history in the next task, for the moment we need to create these classes and get them working.

To start off, let's add a WithdrawTransaction class, which will be able to withdraw money from an account.

1. Create a new C# file named `WithdrawTransaction.cs` , in it, add the WithdrawTransaction class which meets the following UML diagram.


WithdrawTransaction
- _account : Account
- _amount : decimal
- _executed: bool
- _success : bool
- _reversed: bool
+ WithdrawTransaction (account : Account, amount : decimal)
+ Executed : bool <<readonly property>>
+ Success : bool <<readonly property>>
+ Reversed : bool <<readonly property>>
+ Print ()
+ Execute ( )
+ Rollback ( )

   - The `_account` field stores the Account object that we are going to withdraw from.
   - The `_amount` field stores how much we want to withdraw.
   - The `_executed` field remembers if the withdraw has been executed. This can be used in `Execute` to ensure that the transaction is only performed once. Externally this is accessible via the `Executed` property.

- The `_success` field remembers if the withdraw was successful, which can be accessed from the read only `Success` property.
- The `_reversed` field remembers if the withdraw was reversed (via the `Rollback` method). This is accessed from the read only `Reversed` property.
- `Print` is used to output the transaction's details.
- `Execute` is used to get the transaction to perform its task. In this case, it will do the withdrawal and remember if this succeeded.
- `Rollback` performs a reversal of the execute if the transaction was successful and has not already been reversed. In this case it deposits the funds back into the account.

The code for part of this is below...

```
public class WithdrawTransaction
{
    private Account _account;
    private decimal _amount;
    private bool _executed = false;
    private bool _success = false;
    //TODO: add reversed

    public bool Success
    {
        get
        {
            return _success;
        }
    }

    //TODO: Need reversed and executed property

    public WithdrawTransaction(Account account, decimal amount)
    {
        _account = account;
        _amount = amount;
    }

    public void Execute()
    {
        if ( _executed )
        {
            throw new Exception("Cannot execute this transaction as it has al
        }

        _executed = true;
        _success = _account.Withdraw(_amount);
    }

    public void Rollback()
    {
        //TODO: Throw an exception if the transaction has not been executed
        //TODO: Throw an exception if the transaction has been reversed
        //TODO: Remember that we have reversed
        //TODO: Then rollback by reversing actions from execute
    }

    public void Print()
    {
        //TODO: print here
    }
}
```

2. Implement the `Rollback` method using the notes in the above code. It will reverse the action performed in `Execute`, but should only run if the transaction has been executed and has not been reversed already.

3. Have a go at implementing the `Print` functionality - it should print to the terminal whether or not

the `WithdrawTransaction` was successful, how much was withdrawn from the account, and a note if it was reversed.

You should be able to get some of this code from the `DoWithdraw` method of your *Program*.
