SIT771 Object Oriented Development

Pass Task 4.2: Bank Transactions.

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 Doubtfire.

- Your program code (*Program.cs*, *Account.cs*, *Withdraw.cs*, *Deposit.cs*, *transfer.cs*)
- A screen shot of your program running

Instructions

In this task, you're going to be created 3 new classes: Withdraw, Deposit and Transfer. 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 Withdraw class, which will be able to withdraw money from an account.

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

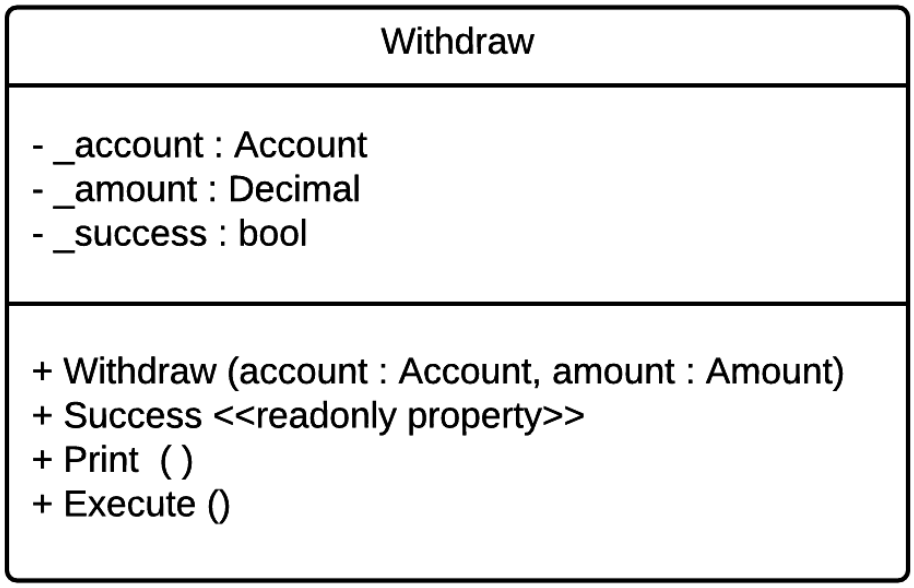

Figure: withdraw uml

- 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 `_success` field remembers if the withdraw was successful, which can be accessed from the read only `Success` 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.

The code for part of this is below...

```
public class Withdraw
{
    private Account _account;
    private decimal _amount;
    private bool _success = false;

    public bool Success
    {
        get
        {
            return _success;
        }
    }

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

    public void Execute()
    {
        _success = _account.Withdraw(_amount);
    }

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

2. Have a go at implementing the Print functionality - it should print to the console whether or not the Withdraw was successful, and how much was withdrawn from the account.

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

3. Switch back to **Program.cs** and change the `DoWithdraw` method to use your new class. It will need to perform the following steps:

   - Ask the user how much to withdraw
   - Create a new `Withdraw` transaction object, for the indicated account and amount.

- Ask the transaction to `Execute`
- Ask the transaction to `Print`

4. Compile and run the program to make sure that it works correctly.

5. Add a Deposit file and class yourself. It should be modeled off the `Withdraw` class you've just created.

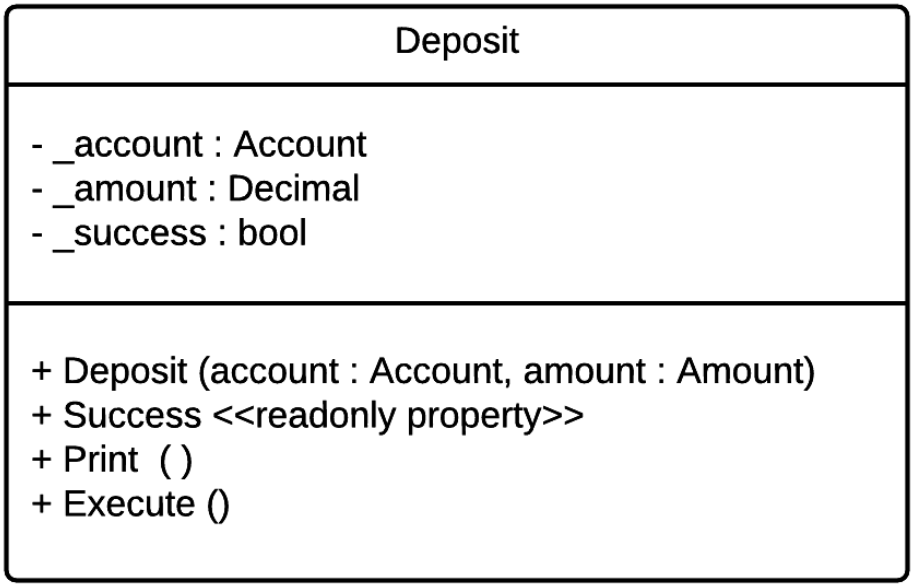

Figure: account transfer class UML

6. Also change `DoDeposit` in the *Program* to make use of the `Deposit` class.

7. When you have the code, compile and run to make sure everything still works.

8. Add a Transfer file and class. The transfer class is a lot like the withdraw and deposit, however it will require two accounts (the from and to account), and internally it can create both a `Deposit` and a `Withdraw` object. Here is the UML diagram for it:

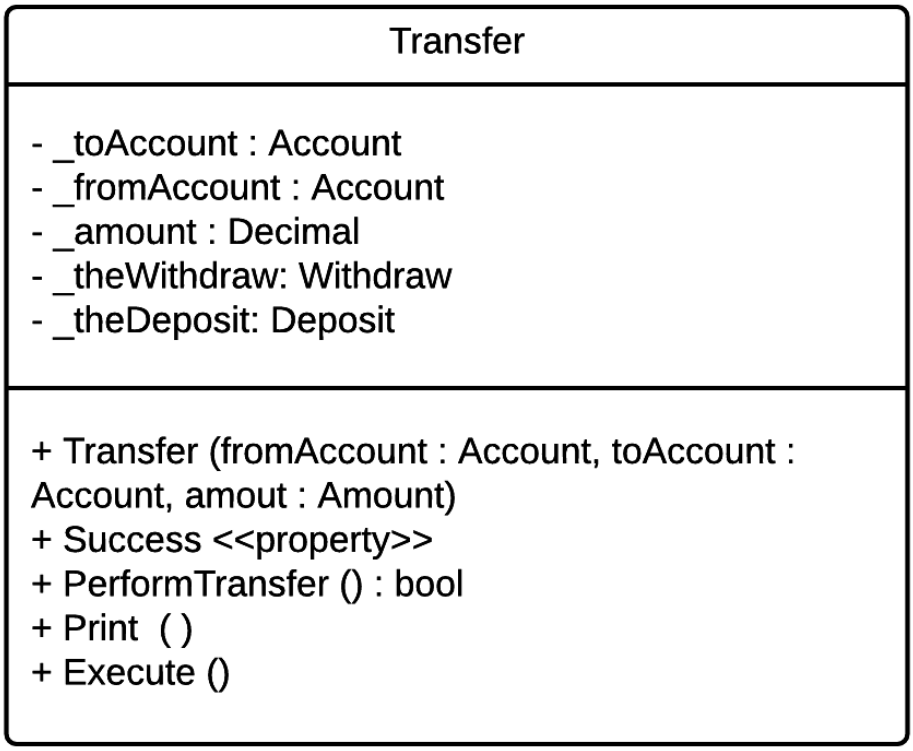

Figure: account transfer class UML

In the constructor, create both the `Deposit` and `Withdraw` objects, and store them in their respective fields. The Transfer object can then get these objects to do the basic work for it. This will avoid code duplication, which is a good thing.

The `PerformTransfer` method will withdraw and deposit the `_amount` value from the two account's balances - however you will need to ensure that the withdraw was successful before you can deposit into the other account. Here's some pseudocode for what this method could look like:

- `Execute` the withdraw
- if that was successful
- `Execute` the Deposit
- if that was unsuccessful
  - Deposit the `amount` back into the `_fromAccount`.

    Use the account directly for this.

Also attached is a flowchart of this method:

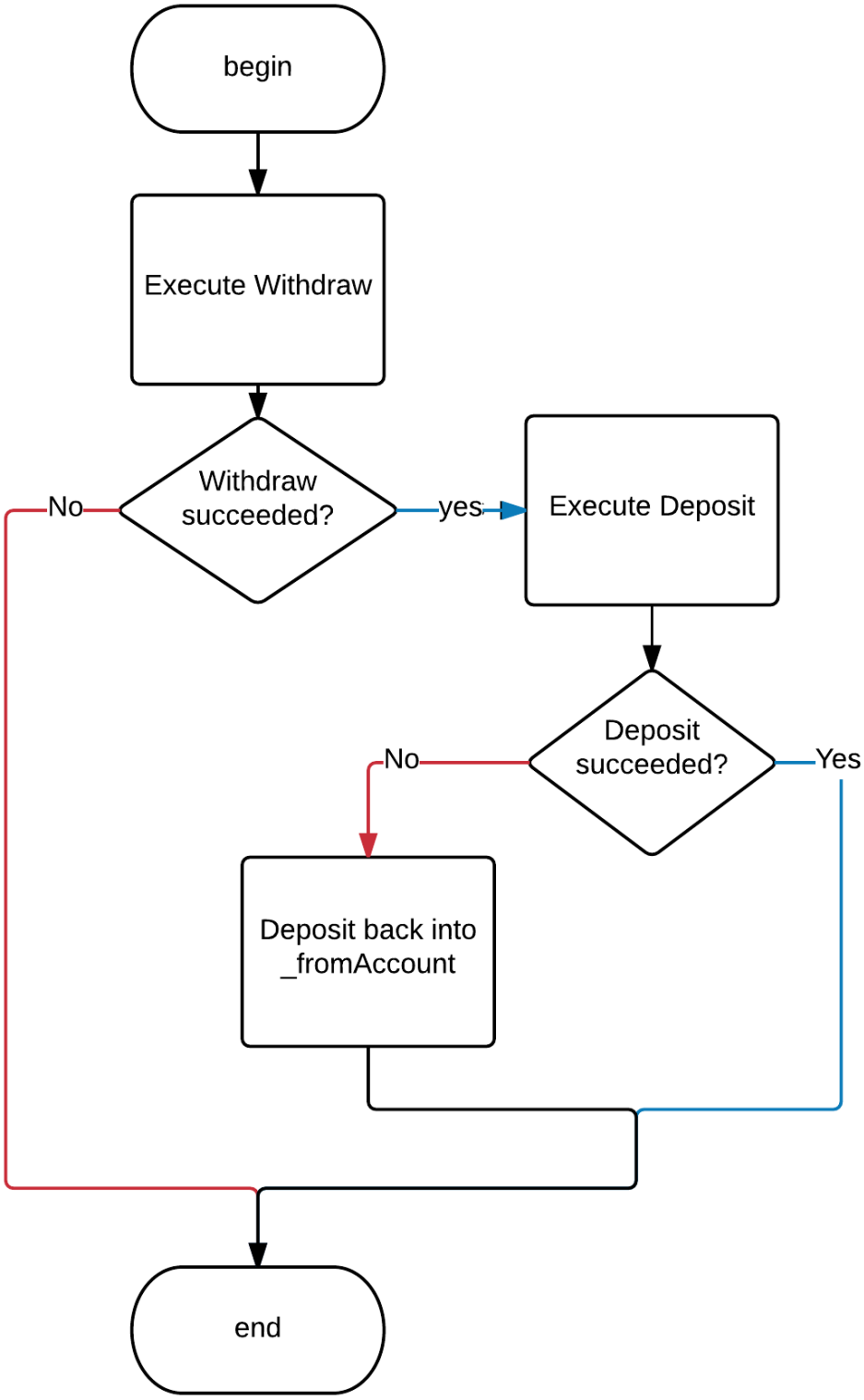

Figure: transfer flowchart

`Success` can be calculated for the `Transfer` : it is true when both of its Deposit **and** its Withdraw transactions succeeded. Otherwise it is false.

When Printed, print a summary line eg "Transferred $x from Jake's Account to My Account". Then get both the withdraw and deposit transactions to print their details. Indent these by asking `Console` to `Write` some spaces before you ask each transaction to print.

9. Switch back to **Program.cs** and update to include options to transfer between accounts.

Have your program transfer from Jake's account to your account.

10. Run the program and deposit, withdraw, transfer and print, create a screenshot and upload alongside code files to Doubtfire!

Remember to backup your work, and keep copies of everything you submit to Doubtfire.