14 Methods and Members

May 12, 2022

1 Methods and Members

Here's your bank account so far! It doesn't do much right now, only prints out the owner and balance. It doesn't even have an account number yet. You'll work on a transaction class, which has been added as an empty class for you.

> Run each code chunk below and see what gets printed. This is what you ended up with last time.

[2]:
```
public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance { get; }

    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        this.Balance = initialBalance;
    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

[3]:
```
var account = new BankAccount("Kendra", 1000);
Console.WriteLine($"Account{account.Number} was created for {account.Owner}␣
 ↪with {account.Balance} dollars");
```

```
Account was created for Kendra with 1000 dollars
```

1.1 #1: Account Number

You need a starting number, that you can base the new account numbers off of, to insure that all the accounts are unique. Below is the code for this number "seed". What does it mean?

- **Private**: This means that no client can see this number. It's internal, part of the internal workings of the code.
- **Static**: This mean the number is universal amongst all individual accounts. If one account changes it, then that number is updated for all the other accounts. This is how you can make it a great way to make sure the account numbers are all unique! Once an bank account uses it for it's bank number, it can add one to the account seed, and the next new bank account has a new number.

  Copy the code below and paste it in the `// Properties` section of the `BankAccount` class.

```
private static int accountNumberSeed = 1234567890;
```

Copy this next bit of code and add it to the constructor.

```
this.Number = accountNumberSeed.ToString();
accountNumberSeed++;
```

Run this code and see what happens!

[4]:
```
public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance { get; }
    //(Paste first bit here!)

    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        this.Balance = initialBalance;
        //(Paste second part here!)

    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

[5]:
```
var account = new BankAccount("Kendra", 1000);
Console.WriteLine($"Account {account.Number} was created for {account.Owner}␣
  ↪with {account.Balance} dollars");
```

```
Account  was created for Kendra with 1000 dollars
```

1.2 #2: Transaction Properties

The next part you need is a balance! One way you could do this is just keep a running tab. However, another way to do it is to create a history of transactions. To do that, you're going to make a little transaction class, that records one transaction.

> paste the properties below into the class `Transaction`

```
public decimal Amount { get; }
public DateTime Date { get; }
public string Notes { get; }
```

[6]:
```
public class Transaction
{
    // Properties (Paste here!)

    // Constructor
}
```

1.3 #3: Transaction Constructor

Next, you need to add the constructor to the class!

> Add the following code to the `Transaction` class, under constructor.

```
public Transaction(decimal amount, DateTime date, string note)
{
    this.Amount = amount;
    this.Date = date;
    this.Notes = note;
}
```

[7]:
```
public class Transaction
{
    // Properties
    public decimal Amount { get; }
    public DateTime Date { get; }
    public string Notes { get; }

    // Constructor (Paste here!)
}
```

1.4 #4: Update BankAccount to match

Now that you have a transaction class, you can use that in our bank account. First, you need to make a list of transactions.

Copy the following code into the Properties section.

```
private List<Transaction> allTransactions = new List<Transaction>();
```

[8]:
```
using System.Collections.Generic;

public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance{ get;}
    private static int accountNumberSeed = 1234567890;
    //(Paste here!)

    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        this.Balance = initialBalance;
        this.Number = accountNumberSeed.ToString();
        accountNumberSeed++;

    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

1.5 #5: Updating Balance

Now that you have a list of transactions that you can use, you need to attach `Balance` to that. What you want to do is, whenever someone wants to get balance, the code checks the list of transactions and tallies it all up, before returning the answer. You can do this by attaching some instructions to the `get` in Balance!

In `BankAccount`, replace `public decimal Balance { get;}` with the following code:

```
public decimal Balance
```

```
{
    get
    {
        decimal balance = 0;
        foreach (var item in allTransactions)
        {
            balance += item.Amount;
        }

        return balance;
    }
}
```

[9]:
```
public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance { get; }  // replace this line!
    private static int accountNumberSeed = 1234567890;
    private List<Transaction> allTransactions = new List<Transaction>();

    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        this.Balance = initialBalance;
        this.Number = accountNumberSeed.ToString();
        accountNumberSeed++;
    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

1.6 #6: Fixing errors

You may have noticed a red squiggly line under `this.Balance`. There's a new error you created! Because whenever you're getting Balance, it goes through a process of summing up the list of transactions, you can't just say that `Balance` is initial balance. You won't fix this entirely in this module, but you can make the code usable for now.

Remove the line `this.Balance = initialBalance`.

[11]:
```
public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance
    {
        get
        {
            decimal balance = 0;
            foreach (var item in allTransactions)
            {
                balance += item.Amount;
            }

            return balance;
        }
    }
    private static int accountNumberSeed = 1234567890;
    private List<Transaction> allTransactions = new List<Transaction>();


    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        //this.Balance = initialBalance; //delete this line
        this.Number = accountNumberSeed.ToString();
        accountNumberSeed++;
    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

2 Review: Where did Initial Balance go?

Here's our final code for this module below. There's a problem though! You no longer have an initial balance and have 0 money! Since you tied up your balance with transactions, you're gonna need to be able to make deposits and withdrawals to put money in the bank. You'll learn that in the next module!

> Run the code cells below.
>
> Try making your own transaction methods before the next module! Where are you getting stuck? What do you need to learn?

[12]:
```
public class Transaction
{
    // Properties (#2)
    public decimal Amount { get; }
    public DateTime Date { get; }
    public string Notes
    {
        get;

    }

    // Constructor (#3)
    public Transaction(decimal amount, DateTime date, string note)
    {
        this.Amount = amount;
        this.Date = date;
        this.Notes = note;
    }
}
```

[13]:
```
using System.Collections.Generic;

public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance  //(#5)
    {
        get

        {
            decimal balance = 0;
            foreach (var item in allTransactions)
            {
                balance += item.Amount;
            }

            return balance;
        }


    }
    private static int accountNumberSeed = 1234567890; //(#1)
    private List<Transaction> allTransactions = new List<Transaction>(); //(#4)


    // Constructor
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        //(#6: deleted "this.Balance = initialBalance;")
        this.Number = accountNumberSeed.ToString(); //(#1)
        accountNumberSeed++; //(#1)

    }

    // Functions
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

    public void MakeWithdrawal(decimal amount, DateTime date, string note)
    {
    }
}
```

[14]:
```
var account = new BankAccount("Kendra", 1000);
Console.WriteLine($"Account {account.Number} was created for {account.Owner}␣
 ↪with {account.Balance} dollars");
```

```
Account 1234567890 was created for Kendra with 0 dollars
```

[ ]:
```
```