13 Objects and Classes

May 12, 2022

1 Objects and Classes

1.1 Object Oriented Programming

Objects are a way to mimic the real world in coding. If you take the concept of a person, they can have a name, address, height, all of these properties that change from person to person. Object oriented coding packages that type of information, so that you can easily make a person with all those details. There are lots of stuff you can do with objects, but for now you can start with the basics.

1.2 Making a Bank

Over the next few modules, you'll make a bank account object with these attributes:

It has a 10-digit number that uniquely identifies the bank account.

It has a string that stores the name or names of the owners.

The balance can be retrieved.

It accepts deposits.

It accepts withdrawals.

The initial balance must be positive.

Withdrawals cannot result in a negative balance.

You can categorize these goals:

- **Properties**: details about the object (how much money it has, the name of the account).
- **Actions**: things the object can do (accept deposits and withdrawals).
- **Rules**: guidelines for the object so that it doesn't try to do impossible things (make sure that the account can never go negative).

1.3 Make it yourself

Below is a blank `BankAccount` object that you're going to create. You'll add in code step by step.

1.4 #1: Properties

Properties are a nice little list of values each object holds. `get` and `set`: sometimes you want to only want a user to see a variable but not change it. Other times you want the user to be able to change a variable. `get` lets you see the variable, `set` lets you change it. (right?)

Copy the code below and paste it into the BankAccount object under //Properties

```
public string Number { get; }
public string Owner { get; set; }
public decimal Balance { get; }
```

[3]:
```
public class BankAccount
{
    // Properties (paste under here)

    // Constructor

    // Functions
}
```

1.5 #2: Constructor

This method is what creates a specific instance of an object. Making a `BankAccount` class, like you're doing now, is like making a template for all bank accounts. It's not a singular individual account. The constructor is what will make a singular account, with all the persons actual details. You give the constructor all the details you want for a specific account, and it assigns the details to the new object's properties.

`this` is a styling choice. It makes explicit that the variable "Owner" is the variable of that specific instance. In the future, you'll have two instances of an object to interact, and `this` will become a bit more explicitly helpful. You can also write `Owner` instead of `this.Owner` if you want!

You're taking the variables `name` and `initialBalance` and creating a bank account that contains these variables.

Copy paste the constructor into the `BankAccount` below, under `//Constructor`

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

[4]:
```
public class BankAccount
{
    // Properties
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance { get; }

    // Constructor (Paste here!)

    // Functions
}
```

1.6 #3: Making an Instance

Now that you have the code written out, see what happens if you make a `BankAccount`!

> Run the two code cells code below to create a specific bank account. Does it do what you expected?
>
> Change the code to make a bank account for yourself. How much money do you want in your bank account?

[5]:
```
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
}
```

[6]:
```
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
```

1.7 What about the account.Number?

You may have noticed that the code didn't print out anything for `account.Number`. That's okay! You haven't put anything in it yet. You'll learn about it in the next module.

1.8 #4: Functions

Functions exist to do actions with an object or change the object variables. These two functions will make a deposit (add money) and make a withdrawal (take out money). You'll be adding stuff in the methods later, but for now you just want to add the empty versions.

> copy the functions below and add them to `BankAccount` under `//Functions`

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

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

```
{
}
```

[7]:
```
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 (paste here!)
}
```

2 Review

Here's the version of `BankAccount` you end up with in this module. You'll be adding more to it in the next module, but why don't you try out stuff, just to see what you need to learn?

> Can you add a 10-digit code? What would your object need to know to make sure the code was unique?
>
> Try out adding to the deposit function! What do you want it to do?
>
> How might you check that the initial balance is positive?

[8]:
```
public class BankAccount
{
    // Variables (#1)
    public string Number { get; }
    public string Owner { get; set; }
    public decimal Balance { get; }

    // Constructor (#2)
    public BankAccount(string name, decimal initialBalance)
    {
        this.Owner = name;
        this.Balance = initialBalance;
    }

    // Functions (#4)
    public void MakeDeposit(decimal amount, DateTime date, string note)
    {
    }

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

[9]:
```
//Make an instance (#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
```

[ ]:
```
```