SIT771 Object Oriented Development

Pass Task 2.2: The Account Class

Overview

This is the start of a series of tasks in which you will develop a small program. These tasks are designed to help you explore the concepts being covered, and to practice your programming skills.

For this task, you will write the start of this program and implement one of its core classes.

Submission Details

Submit the following files to OnTrack.

- Your program code (*Program.cs* and *Account.cs*)
- A screen shot of your program running

You want to focus on the use of classes and objects within the program, and the ability to capture knowledge and behaviour within the program's objects.

Instructions

You will be creating a Banking program. The bank program will have an account, and the account will be able to withdraw and deposit money. It will also be able to print its balance and other details to the terminal.

When the program runs, an `Account` object is created, and money is deposited and withdrawn into the account.

To get this started we will build the code for the Account class, and give it the knowledge and methods it needs to be able to withdraw, deposit and print its balance.

1. Use skm and dotnet to start a new project for **BankProgram**.

```
skm dotnet new console
skm dotnet restore
```

> **Tip:**
>
> If setup correctly, you can also type `code .` in the terminal to open the current directory in Visual Studio Code!

Account Class First Design

Before we can start writing the code we need to understand what we are trying to build (analyse the requirements) and then come up with a plan to get something working quickly.

For this program we will be modeling a standard **Bank Account**. Think about the kinds of things you would associate with a Bank Account, and these can be used to design an `Account` class. What things will an Account need to **know** and what will it need to be able to **do**?

I have a design in mind, but have a quick think before progressing to the next page.

Ok, hopefully you had some good ideas. The way I see this, we are going to need a **Program** which creates **Account** objects. The Program will then perform some operations on the Account (such as withdraw and deposit), and also ask the account to print its balance.

When building software, you want to work in small steps. So rather than trying to get *all* of this working in one go let's focus on getting part of the account working.

Here is the UML Class diagram that shows what we need to build in this iteration (another name for step):

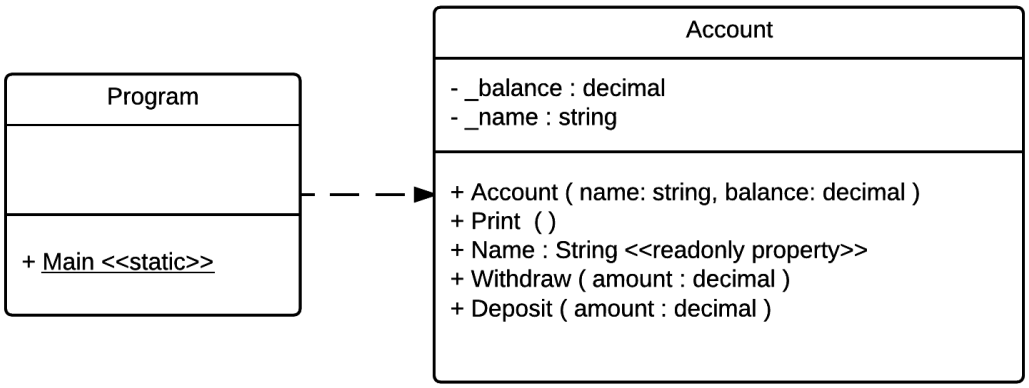

Figure: Iteration 1, showing the Program and Account classes

1. In the project, create a new file called **Account.cs**.

   We will use this file to contain all of the code for the Account class. For anything beyond a small program, it is best to separate the code into multiple files.

2. Create a new `Account` class.

   ```
   using System;

   public class Account
   {

   }
   ```

3. Within the account class block, add the one `private decimal` field for the `_balance`.

   ```
     private decimal _balance;
   ```

   Reminder: a code block is the lines of code between a pair of braces: `{}`.

4. Have a go at adding the `private string` field `_name` yourself.

   Let's have a go at compiling and running this program, it won't do much right now, but it's a good time to make sure we have no syntax errors.

   Execute `skm dotnet run` in the terminal to compile and run the program, you shouldn't see any errors.

Now that our class has "knowledge", let's add some of the functionality (methods) to our class.

5. The first method we're going to add is a *constructor*.

```
  public Account(string name, decimal startingBalance)
  {
    _name = name;
    _balance = startingBalance;
  }
```

   This will be used when we create an Account object to initialise the name and balance of the account.

6. Also inside of Account class, add the `Deposit` method:

```
public void Deposit(decimal amountToAdd)
{
  _balance = _balance + amountToAdd;
}
```

7. Have a go at adding the `Withdraw` method yourself!

8. We can now also add in a `Name` read only property to give others the ability to check the name of the Account. A read only property is one that only as a `get` accessor. Here is the code for getting the name of the account.

```
public string Name
{
    get { return _name; }
}
```

9. Finally, have a go at implementing the `Print` method. The print method will use `Console.WriteLine()` to print the contents of _balance and _name.

   At this point, the *Account* class is finished, compile and run your program again to make sure there are no errors.

10. Now for the exciting part, it's time to create an object of the Account type, and use our new found functionality in our program!

    Inside of our program, create a new Account object which we can refer to from the `account` variable:

```
public class Program
{
    public static void Main()
    {
        Account account = new Account("Jakes Account", 200000);
        //... code below continues from here
    }
}
```

11. Lets print the account, deposit some funds, then print the account again. You can do this using:

    ```
    account.Print();
    account.Deposit(100);
    account.Print();
    ```

    Compile and run this and check that your output is correct.

12. Add a call to withdraw some funds from the account, and then print the balance again.

    Compile and run this and check that your output is correct.

13. Now, create a second account object. Use your name and whatever starting balance you want :)

    Perform a few deposits and withdraws in the `Main` method, and print the balance of the account as you go!

We now have a working Accounts program!

Take a screenshot of your programming running *(make sure it includes a few deposits, withdrawals and prints on the two accounts)* then backup your work and submit the required material to OnTrack.

Task Discussion

For this task you need to discuss at least the following with your tutor:

- How classes are used to define objects.
- How methods, fields, and properties all work together when you create a class
- How fields give knowledge to each object created from the class
- How methods give capabilities to each object created from the class