# Lab: Defining Classes

Problems for exercises and homework for the ["CSharp DB Advanced" course @ Software University](https://softuni.bg/trainings/1741/databases-advanced-entity-framework-october-2017).

# Part I: Defining Classes

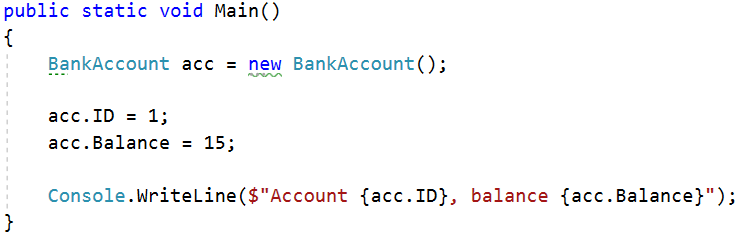
## Define Bank Account Class

Create a class named BankAccount.

The class should have public properties for:

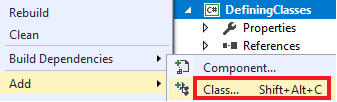
* ID: int
* Balance: decimal

You should be able to use the class like this:



### Solution

Create a **new class** and ensure **proper naming**

****

## Methods

Create a class BankAccount (you can use class from previous task)

The class should have properties for:

* ID: int
* Balance: decimal

And methods for:

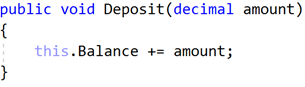
* Deposit(decimal amount): void
* Withdraw(decimal amount): void

You should be able to use the class like this:

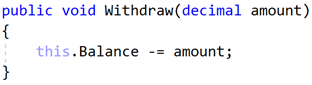
|  |
| --- |
| static void Main()  {  BankAccount acc = new BankAccount();  acc.ID = 1;  acc.Deposit(15);  acc.Withdraw(5);  Console.WriteLine($"Account {acc.ID}, balance = {acc.Balance:f2}.");  } |

### Solution

Create a method Deposit(decimal amount)



Create a method Withdraw(double amount)



## Test Client

Create a test client that tests your BankAccount class.

Support the **following commands**:

* **Create {Id}**
* **Deposit {Id} {Amount}**
* **Withdraw {Id} {Amount}**
* **Print {Id}**
* **End**

If you try to create an account with existing Id, print **"Account already exists".**

If you try to perform an operation on **non-existing account** with existing Id, print **"****Account does not exist"**.

If you try to withdraw an amount larger than the balance, print **"****Insufficient balance"**.

The Print command should print **"Account ID {id}, balance = {balance}"**. Round the balance to the second digit after the decimal separator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Create 1  Create 1  Deposit 1 20  Withdraw 1 30  Withdraw 1 10  Print 1  End | Account already exists  Insufficient balance  Account ID1, balance 10.00 |
| Create 1  Deposit 2 20  Withdraw 2 30  Print 2  End | Account does not exist  Account does not exist  Account does not exist |

### Solution

Create a Dictionary<int, BankAccount> to store existing accounts

Create the input loop



Check the **type of command** and **execute** accordingly (***optional:*** *you can create a separate method for each command*)

Implement the Create command



Implement the rest of the commands following the same logic

## Define Person Class

Create a **Person** class.

The class should have **properties** for:

* Name: **string**
* Age: **int**
* Accounts: List<BankAccount>

The class should have **constructors**:

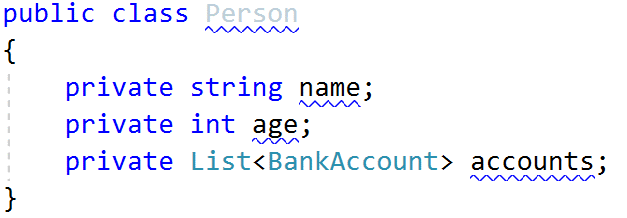
* Person(string name, int age)
* Person(string name, int age, List<BankAccount> accounts)

The class should also have **public methods** for:

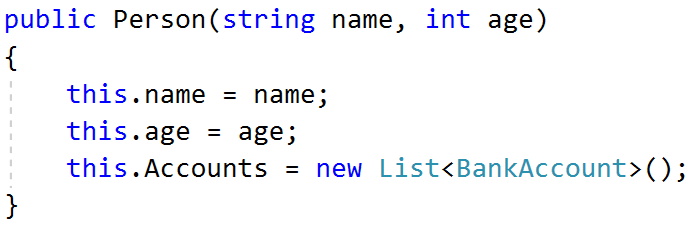
* GetBalance(): decimal

### Solution

Create the class as usual



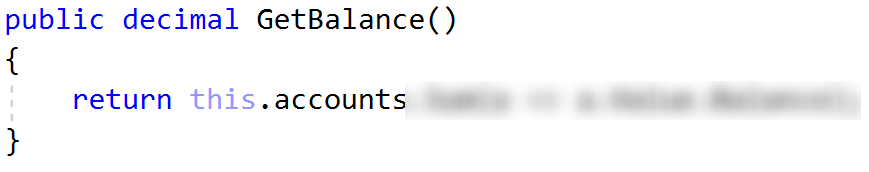
Create a constructor with two parameters



Create a constructor with three parameters



Create method GetBalance()



***Optional:*** You can take advantage of **constructor chaining**

