# Lab: Defining Classes

Problems for the ["C# OOP Basics" course @ SoftUni](https://softuni.bg/courses/csharp-oop-basics)".

You can check your solutions here: <https://judge.softuni.bg/Contests/674/Defining-Classes-Lab>

## Bank Account

**NOTE**: You need a StartUp class with the namespace BankAccount.

Create a **class** named BankAccount.

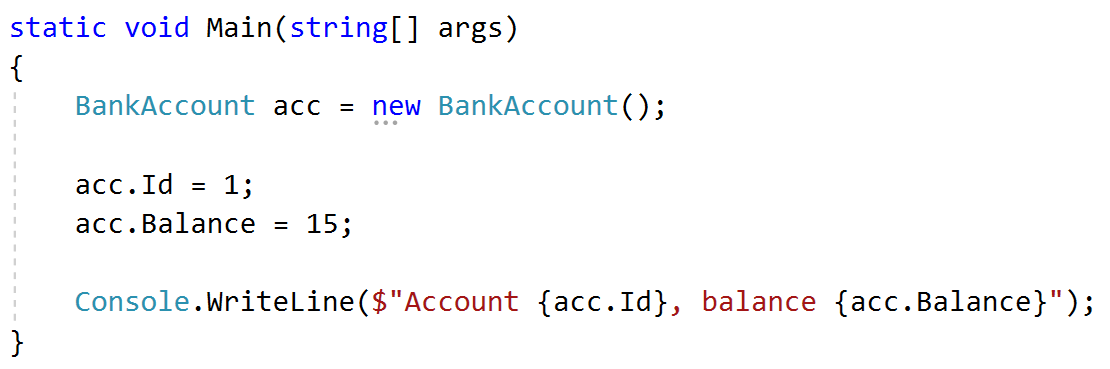
The class should have **private** **fields** for:

* id: int
* balance: decimal

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

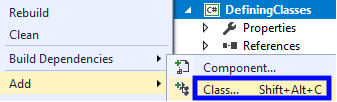
* Id: int
* Balance: decimal

You should be able to use the class like this:



### Solution

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



## Bank Account Methods

**NOTE**: You need a StartUp class with the namespace BankAccount.

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

The class should have private fields for:

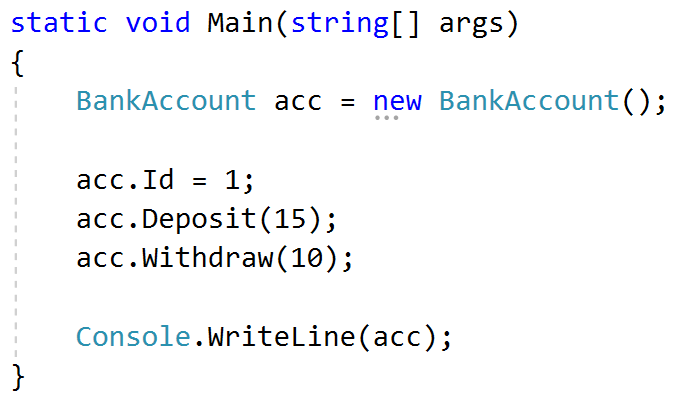
* id: int
* balance: decimal

The class should also have properties for:

* Id: int
* Balance: decimal
* Deposit(decimal amount): void
* Withdraw(decimal amount): void

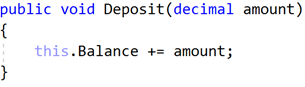
Override the method ToString().

You should be able to use the class like this:

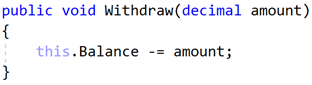


### Solution

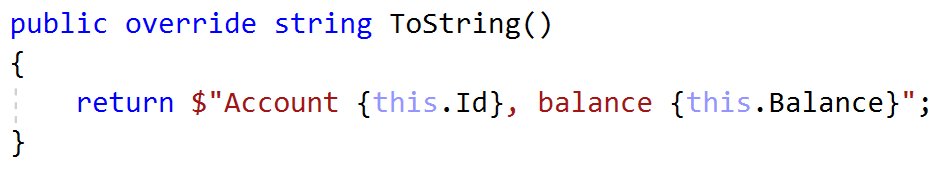
Create a method Deposit(decimal amount)



Create a method Withdraw(decimal amount)



**Override** the method ToString()



## 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 an existing Id, print **"Account already exists".**

If you try to perform an operation on a **non-existing account**, 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 |
| 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.

## Person Class

**NOTE**: You need a **StartUp** class with the namespace **BankAccount**.

Create a **Person** class.

The class should have **private fields** for:

* name: **string**
* age: **int**
* accounts: List<BankAccount>

The class should have **constructors**:

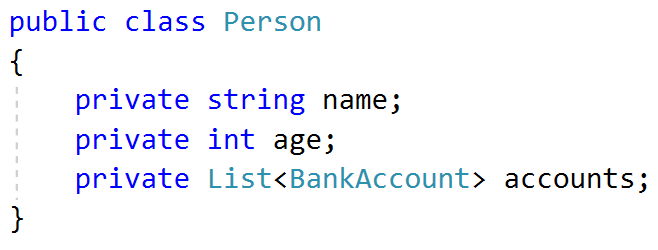
* Person(string name, int age)
* Person(string name, int age, List<BankAccount> accounts)

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

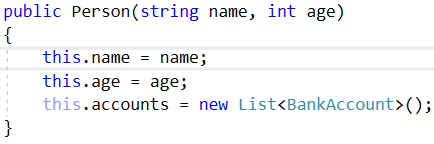
* GetBalance(): decimal

### Solution

Create the class as usual:



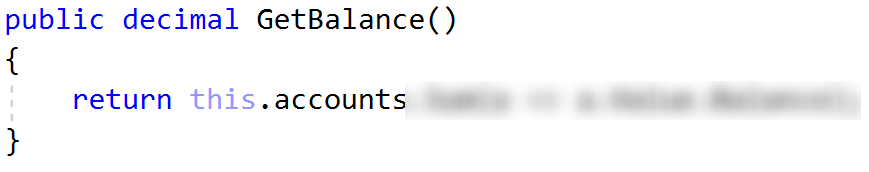
Create a constructor with two parameters:



Create a constructor with three parameters:



Create method GetBalance():



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

