Упражнения: Методи

Problem 1. Дефиниране на клас Person

Създайте клас **Person**.

Класът трябва да има:

- name: String - поле
- age: int - поле
- **Name: String - свойство**
- **Age: int - свойство**

Използвайте класа в Main по следния начин:

```
public static void Main(string[] args)
{
    Person firstPerson = new Person();
    firstPerson.Name = "Гошо";
    firstPerson.Age = 15;

    firstPerson.IntroduceYourself();
}
```

Problem 2. Добавяне на методи към класа Банкова сметка

Създайте клас **BankAccount** (или използвайте вече създадения клас)

Класът трябва да има private полета за:

- id: int
- balance: double

Класът трябва да има и следните свойства и методи:

- **ID: int**
- **Balance: double**
- **Deposit(Double amount): void**
- **Withdraw(Double amount): void**

Предефинирайте и метода **ToString()**.

Трябва да можете да използвате класа по този начин:

```
public static void Main()
{
    BankAccount acc = new BankAccount();

    acc.ID = 1;
    acc.Deposit(15);
    acc.Withdraw(5);

    Console.WriteLine(acc.ToString());
}
```

Решение

Създайте метод **Deposit(double amount)**

```
public void Deposit(double amount)
{
    this.balance += amount;
}
```

Създайте метод **Withdraw(double amount)**

```
public void Withdraw(double amount)
{
    this.balance -= amount;
}
```

Предефинирайте метода **toString()**

```
public override string ToString()
{
    return $"Account {this.id}, balance {this.balance}";
}
```

Problem 3. Човекът и неговите пари

Създайте клас **Person**.

Той трябва да има полета за:

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

Класът трябва да има метод, който изчислява всички пари, които притежава човека от сметките си:

- **GetBalance(): double**

Решение

Използвайте по-горния клас и му добавете възможност за пазене на списък от банкови сметки

```
public class Person
{
    private string name;
    private int age;
    private List<BankAccount> accounts;
}
```

Създайте метод **GetBalance()**

```
public double GetBalance()
{
    return this.accounts
}
```

Problem 4. Най-стария член на фамилията

Създайте клас **Person** с полета **name** и **age**. Създайте клас **Family**. Този клас трябва да има **списък от хора**, метод за добавяне на членове (**void AddMember(Person member)**) и метод, връщащ най-стария член на фамилията (**Person GetOldestMember()**). Напишете програма, която прочита името и възрастта на **N** души и **ги добавя към фамилията**. После **отпечатва името** и **възрастта** на най-стария ѝ член.

Бележки

Добавете в main метода следния код преди вашия. Ако сте дефинирали коректно класа, тестът би трябвало да мине успешно.

```
MethodInfo oldestMemberMethod = typeof(Family).GetMethod("GetOldestMember");
MethodInfo addMemberMethod = typeof(Family).GetMethod("AddMember");
if(oldestMemberMethod == null || addMemberMethod == null)
{
   throw new Exception();
}
```

Примери

Вход	Изход
3 Pesho 3 Gosho 4 Annie 5	Annie 5

Вход	Изход
5 Steve 10 Christopher 15 Annie 4 Ivan 35 Maria 34	Ivan 35

Problem 5. Статистическо проучване

С помощта на класа **Person** и класа **People** (съдържащ private списък от обекти от тип Person) напишете програма, която прочита от конзолата **N** реда с лична информация за хора и после извежда имената на всички, които са на **възраст над 30** години, **сортирани в азбучен ред**.

Бележки

Добавете методи в класа People за добавянето, сортирането и извеждането на хората.

Примери

Вход	Изход
3 Pesho 12 Stamat 31 Ivan 48	Ivan - 48 Stamat - 31
5 Nikolai 33 Yordan 88 Tosho 22 Lyubo 44 Stanislav 11	Lyubo - 44 Nikolai - 33 Yordan - 88

Problem 6. Разликата в дни между две дати

Създайте клас **DateModifier,** който пресмята разликата в дни между две дати. Той трябва да съдържа метод, приемащ **два низови параметъра, указващи дати** в текстов формат и **изчислява** разликата в дни между тях.

Примери

Вход	Изход
1992 05 31 2016 06 17	8783
2016 05 31 2016 04 19	42

Problem 7. Тестов Клиент

Създайте тестов клиент, който използва класа **BankAccount,** направен в задача 2.

Трябва да поддържате следните операции, подавани като входни данни от конзолата:

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

Създайте методи към Program.cs за всяка от командите. Имайте в предвид и следната допълнителна обработка на данните:

- Ако се опитате да създадете сметка със съществуващо Id, изведете **"Account already exists".**
- Ако се опитате да извършите операция върху несъществуваща сметка, изведете **"Account does not exist"**.
- Ако се опитате да изтеглите сума, която е по-голяма от баланса, изведете **"Insufficient balance"**.
- Print командата, трябва да изведе **"Account ID{id}, balance {balance}"**. Закръглете баланса до втория знак след запетаята.

Примери

Вход	Изход
Create 1 Create 2 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	Account does not exist Account does not exist Account does not exist

End	

Решение

Използвайте **`Dictionary<int, BankAccount>`** за да пазите сметките

Направете си цикъла за приемане на команда

```
var cmdArgs = command.Split();

var cmdType = cmdArgs[0];
switch (cmdType)
{
    case "Create":
        Create(cmdArgs, accounts);
        break;
    case "Deposit":
        Deposit(cmdArgs, accounts);
        break;
    case "Withdraw":
        Withdraw(cmdArgs, accounts);
        break;
    case "Print":
        Print(cmdArgs, accounts);
        break;
}
```

Създайте методи към Program.cs, за всяка от командите.

- Create – проверявате дали в речника има ключ с такова id – ако няма, създавате сметката.

```
private static void Create(string[] cmdArgs, Dictionary<int, BankAccount> accounts)
{
    var id = int.Parse(cmdArgs[1]);
    if (accounts.ContainsKey(id))
    {
        Console.WriteLine("Account already exists");
    }
    else
    {
        var acc = new BankAccount();
        acc.ID = id;
        accounts.Add(id, acc);
    }
}
```

Имплементирайте останалите команди работейки с подобна логика.

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).