# Упражнения: Дефиниране на класове

## Problem 1. Валидна личност

Дефинирайте прост клас Person, притежаващ следните полета: **first name**, **last name** и **age**. **Валидирайте** данните в setter-ите на свойствата и **хвърлете** подходящи **изключения** ако са въведени невалидни данни.

### Step 1. Създайте клас Person

Създайте проект за това упражнение и добавете клас Person в отделен .cs файл. Класът трябва да съдържа следните полета: **first name (string), last name (string) and age (int)**.

Всички полета са **задължителни**, което означава, че ви трябва един конструктор, който приема и трите като параметри. Например:

```
namespace ExceptionHandling_Exercises
{
    public class Person
    {
        private string firstName;
        private string lastName;
        private int age;

        public Person(string firstName, string lastName, int age)
        {
            // TODO: add properties and validate data
        }

        //TODO: add properties
    }
}
```

### Step 2. Добавете свойства и валидирайте данните

Добавете свойство за всяко от полетата. Извършете валидиране в техните **setter**-и, за да запазите състоянието на обекта Person валидно.

Свойствата **first** и **last name** не може да бъдат **null** или **празни** низове. За да проверите това, използвайте метода **string.IsNullOrEmpty()**.

Свойството **age** трябва да е в диапазона **[0 … 120]**.

Ако са въведени невалидни данни, **хвърлете** подходящи изключения с достатъчно описателни **съобщения**. Например ако бъде въведено празно име, едно подходящо изключение би било **ArgumentNullException**. Ако възрастта е отрицателна или твърде голяма, подходящото изключение ще е **ArgumentOutOfRangeException**.

Пример за валидирането на **first name** (last name е аналогично):

```
public string FirstName
{
    get
    {
        return this.firstName;
    }

    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new ArgumentNullException(
                "value",
                "The first name cannot be null or empty.");
        }

        this.firstName = value;
    }
}
```

Пример за валидирането на **age**:

```
public int Age
{
    get
    {
        return this.age;
    }

    set
    {
        if (value < 0 || 120 < value)
        {
            throw new ArgumentOutOfRangeException(
                "value",
                "Age should be in the range [0 ... 120].");
        }

        this.age = value;
    }
}
```

Сега вече конструкторът би трябвало да използва свойствата вместо да модифицира директно частните полета:

```
public Person(string firstName, string lastName, int age)
{
    this.FirstName = firstName;
    this.LastName = lastName;
    this.Age = age;
}
```

## Step 3. Тествайте класът Person

Във вашата главна програма, проверете дали вашият клас се държи както се очаква. Създайте няколко обекта от тип Person – един с **валидни данни**, един с **празно first name**, един с **null за last name**, един с **отрицателна**

**възраст** и един с **age > 120**. Проверете дали изпълнението на кода води до грешки, когато се въведат невалидни данни. Тествайте невалидните случаи един по един чрез коментиране на редовете на другите невалидни случаи (защото вашата програма ще спре изпълнението си още при първата грешка).

```
public static void Main()
{
    Person pesho = new Person("Pesho", "Peshev", 24);

    Person noName = new Person(string.Empty, "Goshev", 31);
    Person noLastName = new Person("Ivan", null, 63);
    Person negativeAge = new Person("Stoyan", "Kolev", -1);
    Person tooOldForThisProgram = new Person("Iskren", "Ivanov", 121);
}
```

## Step 4. Добавете Try-Catch блокове

За да предотвратите сриването на програмата, обградете невалидните редове в **try-catch** блокове. Добра практика е да поставите различни catch блокове за различните типове грешки, които очаквате, че командите може да предизвикат. Отпечатайте **съобщението** на изключението от catch блока.

Пример (невалидно **име**):

```
try
{
    Person noName = new Person(string.Empty, "Goshev", 31);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}

// Result in console:
// Exception thrown: The first name cannot be null or empty.
// Parameter name: value
```

Пример (невалидна **възраст**):

```
try
{
    Person negativeAge = new Person("Stoyan", "Kolev", -1);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}

// Result in console:
// Exception thrown: Age should be in the range [0 ... 120].
// Parameter name: value
```

## 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).