# Exercise: Exception Handling

This document defines the **exercise assignments** for the ["C# OOP Basics" course @ Software University](https://softuni.bg/trainings/2084/csharp-oop-basics-october-2018).

## Square Root

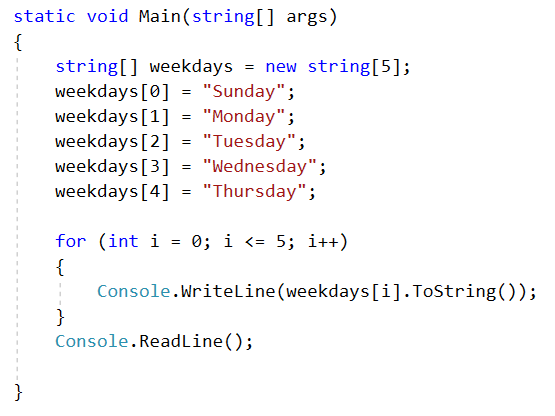
Write a program that reads an integer **number** and **calculates** and **prints** its **square** **root**. If the number is invalid or negative, print "Invalid number". In all cases finally print "Good bye". Use **try-catch-finally**.

## Enter Numbers

Write a method **ReadNumber(int start, int end)** that enters an integer number in a given range [**start…end**]. If an **invalid number** or a **non-number** text is entered, the method should **throw an exception**. Using this method write a program that enters **10 numbers**: **a1, a2, … a10, such that 1 < a1 < … < a10 < 100**. If the user enters an invalid number, make the user enter all of them again.

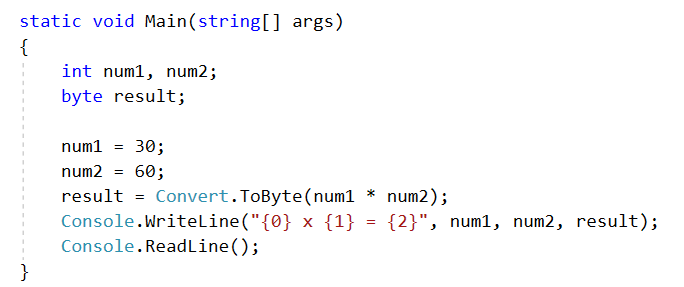
## Fixing

This program is throwing an IndexOutOfRangeException. Using your skills, fix this problem using a try catch block.



## Fixing Vol2

The given program is throwing an OverflowException. Fix it.



## Convert.ToDouble

The static methods in the static class System.Convert can convert values from one type to another type. Check the documentation of System.Convert.ToDouble. There are several **overloads** of this **method**. Which **exceptions** can occur by converting from **string** to **double**? Write a program which triggers these exceptions.

Finally, supply **handlers** for the exceptions. The **handlers** should **report** the **problem** on a standard output, rethrow the exception and then continue.

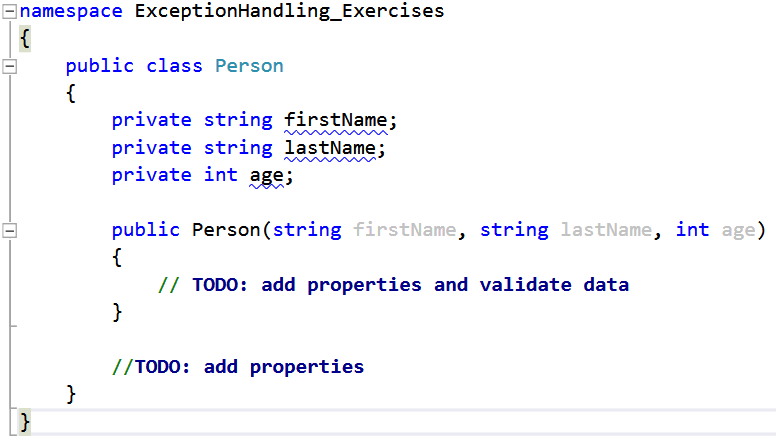
## Valid Person

Define a simple class **Person,** which has the following fields: **first name**, **last name** and **age**. **Validate** the data in the properties’ **setters**, **throw** appropriate **exceptions** in case invalid data is entered.

### Step 1. Create a Class Person

Create a project for this exercise and add a class **Person** in a separate .cs file. The class should contain the following fields: **first name (string), last name (string) and age (int)**.

All fields are **required**, meaning you should have one constructor accepting all three as **parameters**. For example:



### Step 2. Add Properties and Validate the Data

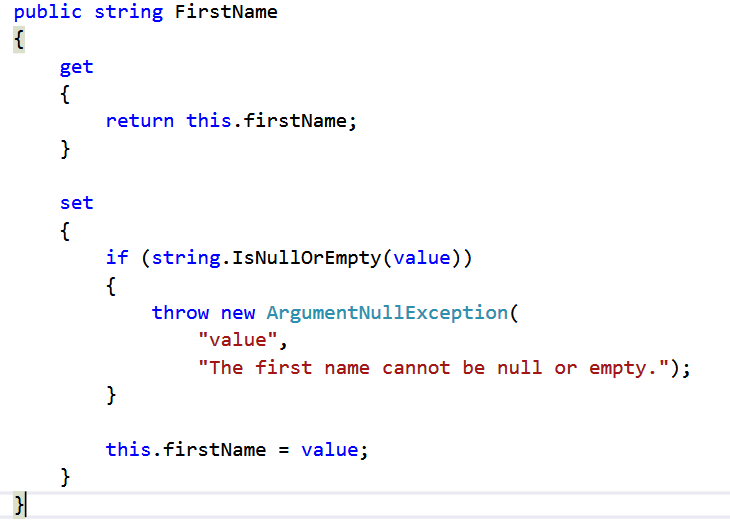
Add a **property** for each of the fields. Perform validations in their **setters** to keep the state of the **Person** objects correct.

The **first** and **last name** cannot be **null** or **empty** strings. To check this, use the **string.IsNullOrEmpty()** method.

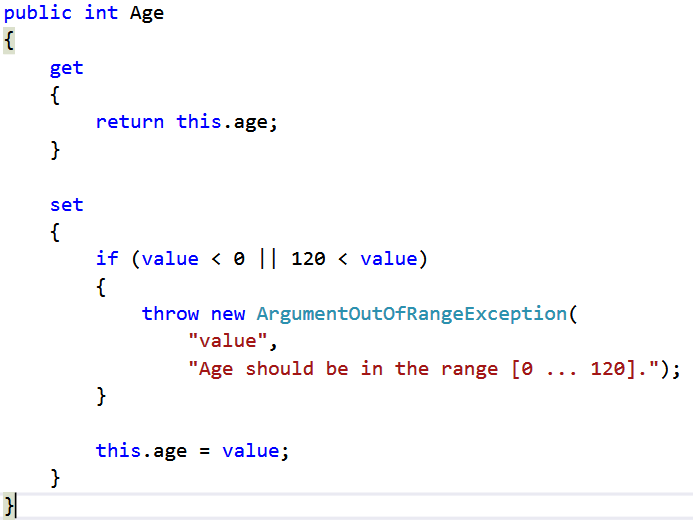
The **age** must be in the range **[0 … 120]**.

If invalid data is entered, **throw** appropriate exceptions with descriptive **messages**. E.g., if an empty name is entered, an appropriate exception may be **ArgumentNullException**. If the age is negative or too big, an appropriate exception would be **ArgumentOutOfRangeException**.

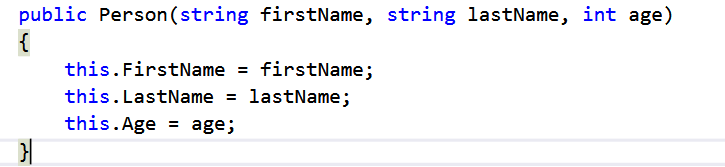
Example for validating the **first name** (last name is analagous):



Example for validating the **age**:

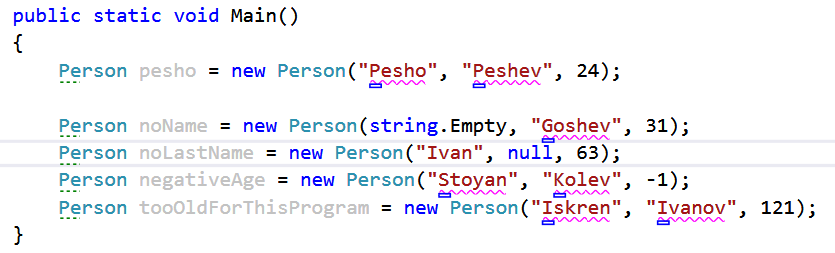


Now the constructor should make use of the properties instead of modifying the private fields directly:



### Step 3. Test the Person Class

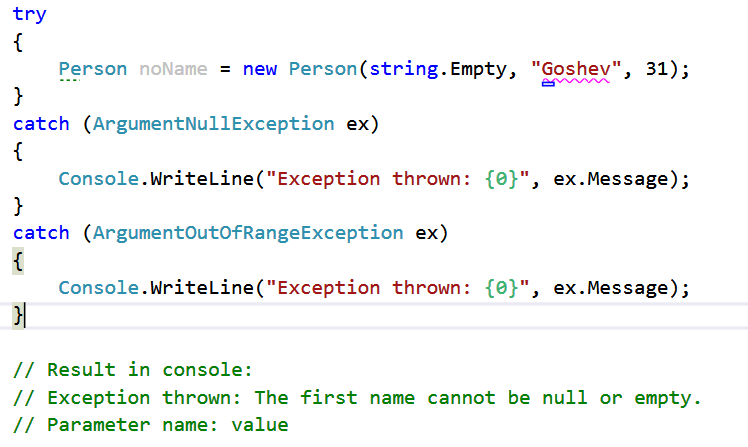
In your main program, test whether your class behaves properly. Create several objects of type Person – one with **valid data**, one with an **empty first name**, one with **null as last name**, one with **negative age** and one **with age > 120**. Check whether executing the code results in errors, when bad data is provided. Test the invalid cases one by one by commenting out the other invalid lines of code (your program will stop executing when the first error is encountered).



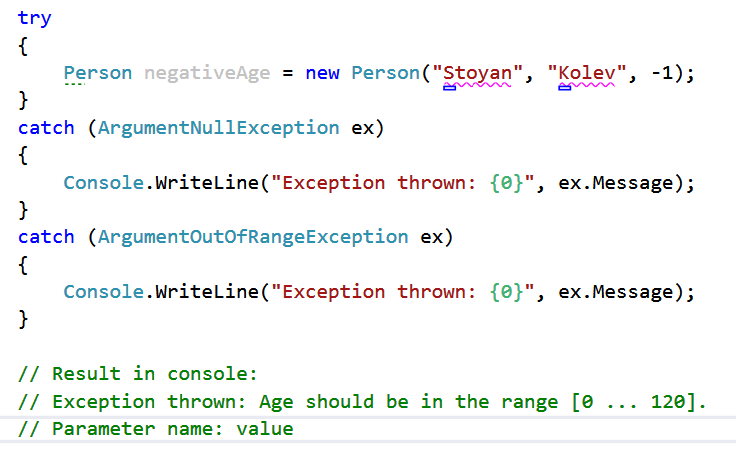
### Step 4. Add Try-Catch Blocks

To prevent the program from blowing up, surround the invalid lines in **try-catch** blocks. It’s a good practice to put different catch blocks for the different types of errors you anticipate the operation might throw. Print the **message** of the exception in the catch block.

Example (invalid **name**):



Example (invalid **age**):



## Custom Exception

Create InvalidPersonNameException class in the previous problem, which does not allow any special character or numeric value in a name of any of the students. To do that create Student class with Name and Email properties. When trying to create student with name "Gin4o", throw your custom Exception class.