Exercises: Implementing Stack

Problems for exercises and homework for the "Programming Advanced- OOP Basics" course from the official "Applied Programmer" curriculum.

You can check your solutions here: https://judge.softuni.bg/Contests/2876/Implementing-Stack

Create a new project named **`ImplementCustomStack`**, rename class **`Program.cs`** to **`StartUp.cs`** and add new **public** class **`CustomStack.cs`** .

Overview

The custom stack will have similar functionality to the C# stack and we will make it work **only** with **integers**. Later on, we will learn how to implement it in a way that will allow us to work with any types. It will have the following functionality:

- **`void Push(int element)`** – Adds the given element to the stack
- **`int Pop()`** – Removes the last added element
- **`int Peek()`** – Returns the last element in the stack without removing it
- **`void ForEach(Action<int> action)`** – Goes through each of the elements in the stack

Feel free to implement your own functionality or to write the methods by yourself.

Implement the CustomStack Class

Details about the Structure

The implementation of a custom stack is much easier, mostly because you can only execute actions over the last index of the collection, plus you can iterate through the collection. You should be able to create it entirely on your own. The first thing you can do is have a clear vision of how you want your structure to work under the provided public functionality. For example:

- It should hold a **sequence of items in an array**.
- The structure should have a **capacity** that **grows twice** when it is filled, **always starting at 4**.

The **`CustomStack`** class should have the **properties** listed below:

- **`int[] items`** - An array, which will hold all of our elements.
- **`int Count`** – This property will return the count of items in the **collection.**
- **`const int InitialCapacity`** – this constant's value will be the initial capacity of the internal array.

Implementation

In the created class **`CustomStack`** add a private constant field named **`InitialCapacity`** and set the value to **4.** This field is used to declare the **initial capacity** of the **internal** array. We already know that it's not a good practice to have **magic numbers** in your code. Afterwards, we are going to declare our internal array and a field for the count of elements in our collection.

```
public class CustomStack
{
    private const int initialCapacity = 4;
    private int[] items;
    private int count;
```

Of course, you have to initialize the collection. Also, set the count variable to 0. As we already know, this can be done inside the constructor of the class:

```
public CustomStack()
{
    this.count = 0;
    this.items = new int[initialCapacity];
}
```

Next, you have to add a public property **Count** that holds the value of the **count** field. This way, you will be able to get the count of items in the collection from other classes.

```
public int Count
{
    get
    {
        return this.count;
    }
}
```

Now you can proceed to the implementation of the methods, which your **CustomStack** is going to have. All of the functionalites described in the description are very easy to implement, so we strongly recommend for you to try to do it on your own. If you have any difficulties, you can help yourself with the code snippets below.

1. Method Push(int Element)

This method adds an element to the end of the collection, just like the C# Stack **Push()** method does . This is a very easy task. Here is the code you can use, if you meet any difficutlies:

```
public void Push(int element)
{
    if(this.items.Length == this.count)
    {
        [illegible]
    }
    this.items[this.count] = element;
    count++;
}
```

2. Method Pop()

The **Pop()** method returns the last element form the collection and removes it. The implementation is easier than the implementation of the **RemoveAt(int index)** and **Remove()** methods of the **CustomList**. Try to implement it on your own. Afterwards, you can look at this implementation:

```
public int Pop()
{
    if(this.items.Length == 0)
    {
        throw new InvalidOperationException("CustomStack is empty");
    }
    [illegible]
}
```

3. Method Peek()

The **Peek()** method has the same functionality as the C# Stack – it returns the last element from the collection, but it **doesn't remove** it. The only thing we need to consider is that you can't get an element from an empty collection, so you must make sure you have the proper **validation**. For sure, you will be able to implement it on your own.

4. Method ForEach(Action<object> Action)

This method goes through every element from the collection and executes the given action. The implementation is very easy, but it requires some additional knowledge, so here is what you can do:

```
public void ForEach(Action<object> action)
{
    [redacted]
}
```

You can add any kind of functionalities to your **CustomStack** and afterwards you can test how it works in your **Main()** method in your **StartUp** class.

static void Main()	Output

`var stack = new CustomStack();`
`stack.Push(2);`
`stack.Push(3);`
`stack.Push(4);`
`Console.WriteLine(stack.Count);`
`Console.WriteLine(stack.Pop());`
`Console.WriteLine(stack.Peek());`
`stack.ForEach(x => Console.WriteLine(x));`
	3
	4
	3
	2
	3