

## Week 1 assignments

When creating the program code, you must apply the following basic principles:

- create a separate project for each assignment;
- use name 'assignment1', 'assignment2', etcetera for the projects;
- create one solution for each week containing the projects for that week;
- make sure the output of your programs are the same as the given screenshots;

## CodeGrade auto checks

Make sure all CodeGrade auto checks pass (10/10) for your assignments. The manual check will be done by the practical teacher.

Auto checks assignment 1-<sup>10</sup>/<sub>10</sub> **AT**

Auto checks assignment 2-<sup>10</sup>/<sub>10</sub> **AT**

Manual check

Automatic checks for assignment 1		
0	10	
	10	
	100	%

Submit

6.67

20 / 30

✖

⌵

## Assignment 1

In this assignment we will implement a Stack. A stack is a data structure in which items can be stored, the last stored item will be the first item to be retrieved from the stack (*and the first stored item will be the last item to be retrieved*). This processing order is known as LIFO (Last-In First-Out).

An example of stack usage is the stack-memory in computer programs; for every method call, the parameters and the local variables (of the method) are stored in the stack. As soon as the method has done its work, these stored items are removed from the stack. Here the same rule applies: items of the last method call are removed first.

We want to be able to perform the following Stack operations (*no matter how the stack is implemented*):

1. `void Push(int value)` → pushes a new item (int) onto the stack; throw an exception when the stack is full
2. `int Pop()` → returns the last pushed item (int) on the stack (and removes this item from the stack); throw an exception if the stack is empty
3. `bool Contains(int value)` → returns true if the stack contains a specific item (int), false otherwise
4. `int Count { get; }` → returns the number of items on the stack
5. `bool IsEmpty { get; }` → returns true if the stack is empty, false otherwise

Create an interface (IStack) with these 5 methods/properties. Create a class (`ArrayStack`) that implements this IStack interface, using an array for storing the items. Use a constructor parameter for the maximum number of stack items. Use the program below to test your Stack implementation. You have to implement method `CheckValues` yourself.

```
void Start()
```

```
{
    IStack myStack = new ArrayStack(50);
    AddValues(myStack);
    CheckValues(myStack);
    ProcessValues(myStack);
}

void AddValues(IStack stack)
{
    Random rnd = new Random();
    for (int i = 0; i < 5; i++)
    {
        int value = rnd.Next(100);
        stack.Push(value);
        Console.WriteLine($"pushed {value,2},
                           new count: {stack.Count}");
    }
}

void CheckValues(IStack stack) { ... }

void ProcessValues(IStack stack)
{
    while (!stack.IsEmpty)
    {
        int value = stack.Pop();
        Console.WriteLine($"popped {value}, new count: {stack.Count}");
    }
}
```

```

C:\Users\gerwin...
pushed 70, new count: 1
pushed 97, new count: 2
pushed 45, new count: 3
pushed  5, new count: 4
pushed 69, new count: 5

Enter a number: 45
stack contains value 45
Enter a number: 46
stack does not contain value 46
Enter a number: 0

popped 69, new count: 4
popped  5, new count: 3
popped 45, new count: 2
popped 97, new count: 1
popped 70, new count: 0
  
```

## Assignment 2

Implement the next interfaces (each in a separate file):

```
public interface IPencil {
    bool CanWrite { get; } // determines if the pencil can still write
    void Write(string message); // writes characters of the message
    void AfterSharpening(); // the pencil is made 'new' (so it can write 'max' again)
}

public interface IPencilSharpener
{
    void Sharpen(IPencil pencil);
}
```

Create a class "Pencil" that implements interface "IPencil", and create a class "PencilSharpener" that implements interface "IPencilSharpener".

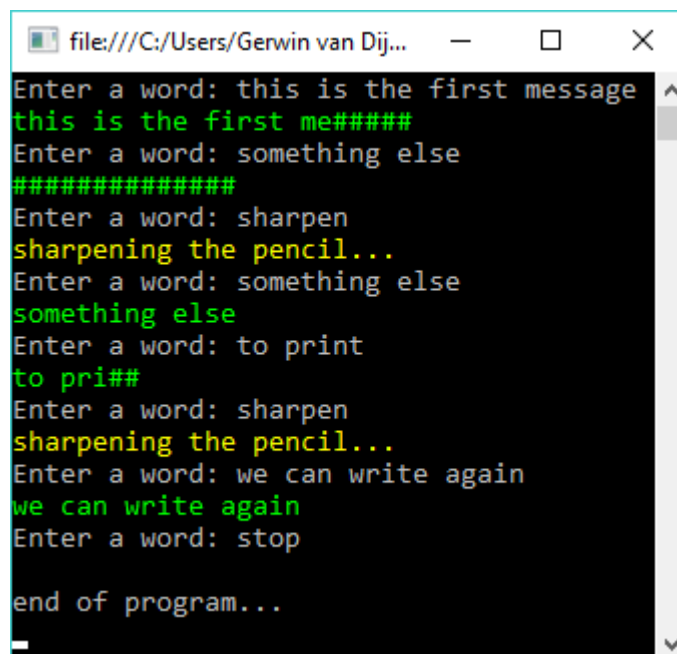
Add the following member fields to class Pencil:

```
private int maxToWrite; // number of characters to write with a sharpened pencil
private int nrOfCharsWritten; // number of written characters
```

The pencil will not be able to write the complete message if it's too long. After writing a certain number of characters, the pencil will become dull and each remaining character will be written as '#'.

Write a small program that reads messages until the user enters the word "stop". Each entered message should be written by the (created) pencil. If the user enters the word "sharpen", then the pencil must be sharpened by the (created) pencil-sharpener.

Output of the program could be like:



```
file:///C:/Users/Gerwin van Dij...
Enter a word: this is the first message
this is the first me#####
Enter a word: something else
#####
Enter a word: sharpen
sharpening the pencil...
Enter a word: something else
something else
Enter a word: to print
to pri##
Enter a word: sharpen
sharpening the pencil...
Enter a word: we can write again
we can write again
Enter a word: stop
end of program...
```