SOM kick off exercise

September 6, 2019

Introduction

In this first exercise you will deal with *abstract classes*. Unfortunately, this topic is not covered by the introductory course "Mobiel programmeren". However, a good understanding of this concept is essential for grasping the core ideas of object oriented design. The essence of the software design challenge is applying *abstraction* at the right place and time. Although running the risk of getting too philosophical, we might even state that *abstraction* is at the core of practicing science.

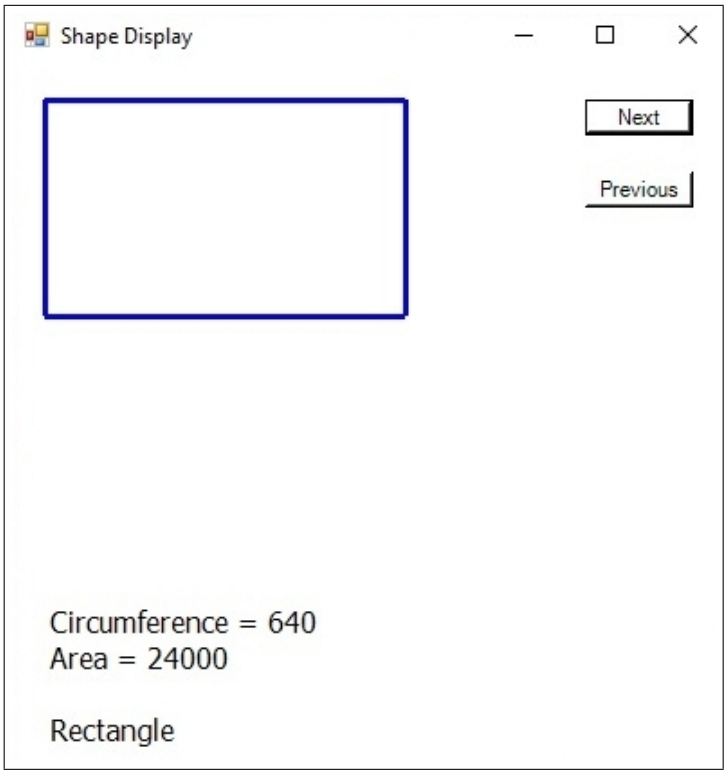
Yes, this exercise will require *some* programming, but do not get worried too much. The amount of coding required is minimal. The meaning of *required* should be taken very strictly: writing too much code is regarded to be a negative quality. Go for minimality (in coding) and for optimality (by thinking).

You will start with a running piece of code. We think that this code reflects a certain level of design quality. You will have to realize some extensions of this code. (One could state that the software design problem is to deal with modifications before you know them.) Your first task is to understand how this piece of software has been structured. This does not mean that you have to understand all the details. It means that you have to understand those parts that you need to realize the modifications. So do not get troubled by irrelevant details.

Start with watching the clip on abstract classes, if necessary. Next identify the spots in the example code where you have to add the extensions. If you think you have this clear, discuss your approach with one of the teaching assistants. She/he may give you some hints.

Step 1

The current program displays a series of shapes with a description: the area and the name of the figure. Your task is to add the circumference of the figure.



For those of you who do not have sweet memories of Euclides and Pythagoras, use this table.

shape	properties	area	circumference
rectangle	width w, height h	wh	$2w + 2h$
triangle	basis b, height h	$bh/2$	$2\sqrt{(b^2/4 + h^2)} + b$
circle	radius r	πr^2	$2\pi r$
square	side s	s^2	$4s$

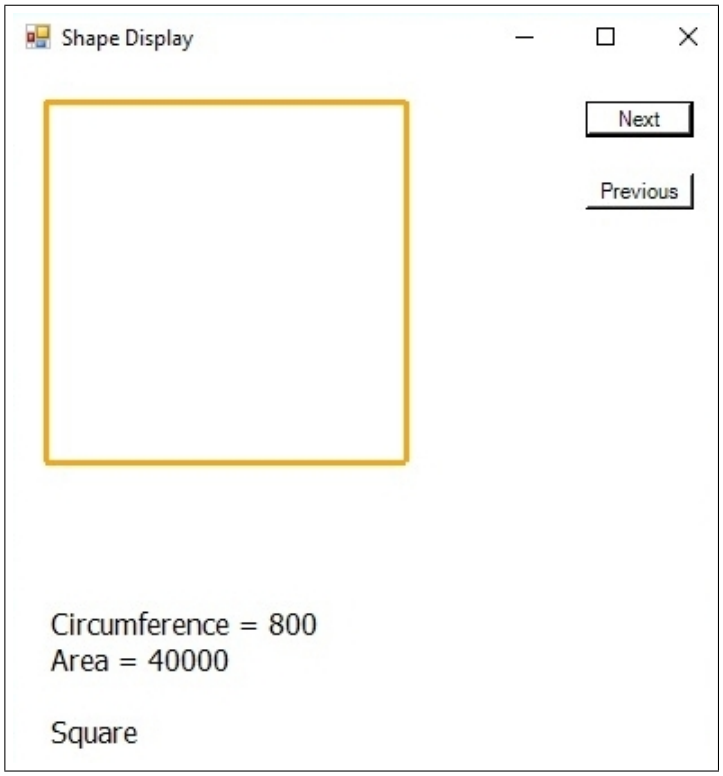
The static class Math provides you with constant PI and function Sqrt. For example, this C#-expression calculates the most complex expression of the table: the circumference of a triangle.

```
2 * Math.Sqrt(Basis * Basis / 4 + Height * Height) + Basis
```

If you succeeded to implement this extension, show it to the teaching assistant. We will register a pass mark. You may work together in small groups.

Step 2: for the brave of heart

Extend the result of the previous step to add squares. Squares have only one describing property: the *side*.



In the method `InitTestCollection()`, a command for adding an example square is given in a comment line. Your program should be able to deal with this command after removing the comment symbols. Don't mess with the parameters.
This extension can/should be realized by adding only three lines of code (not including brackets)!

Appendix

In the clip about abstract classes, we do not cover a specific issue related to inheritance. Suppose we have a class A that is *not* abstract, so it has a constructor method. If we want to create a subclass B of A that should initiate a new object the same way as A does, but with some specific extras because it is a B, this is the way to do it. Construction of a new B object guarantees the initialization of both `myAProperty` and `mySpecialBProperty` by using the `base` keyword.

```
class A
{
    public int myAProperty;

    public A (int initval)
    {
        myAProperty = initval;
    }
}

class B : A
{
    public int mySpecialBProperty;

    public B (int initvalA, int initvalB): base(initvalA)
    {
        mySpecialBProperty = initvalB;
    }
}

...

A myA = new A(3);
B myB = new B(3, 42);
```

So we avoid copying the initialization code of `myAProperty` in the constructor method for B. This looks not very spectacular in this toy example, but imagine that the constructor method for A is far more complex and prone to modifications. We should never copy/paste code .