

# CSCI 1301 – Lab 07

September 4, 2019

## 1 Using a Pre-Defined Class

This lab will guide you in your first manipulation of a programmer-defined class. We will use the example shown in class. The last part is challenging, we will give a possible solution to it in class, but make sure you try to solve it by yourself beforehand.

### 1.1 Manipulating Two .cs Files at a Time

1. Download the Rectangle<sup>1</sup> project, extract it, and open it with VS. Note that in the “Solution Explorer”, there are two `cs` files listed: `Program.cs` and `Rectangle.cs`.
2. In the Solution Explorer, double-click on `Rectangle.cs`, and note how close it is from what was presented during the lecture.
3. In the Solution Explorer, double-click on `Program.cs`, and observe it.
4. Compile and execute the code.
5. Now, do the following:
  - Introduce a syntactical error in `Program.cs` (e.g., remove a `;`), and try to build the solution: what do you observe? Restore the program to its previous state, using `CTRL + z` to “undo” your operation.
  - Introduce a syntactical error in `Rectangle.cs` (e.g., remove a `;`), and try to build the solution: what do you observe? Undo the modification using `CTRL + z`.
  - Add `length = 12;` in the main method of `Program.cs` and try to build the solution: what do you observe? Undo the modification using `CTRL + z`.

### 1.2 Enriching Program.cs

Edit the `Main` method of `Program.cs` by adding at its end statements that perform the following:

1. Create a second `Rectangle` object and set its `length` to 3 and its `width` to 3.
2. Create a third `Rectangle` object, and ask the user to specify its `length` and `width`. Display the area of this rectangle at the screen.
3. Create a fourth `Rectangle` object, do not specify its `length` or `width`, and display them at the screen. What do you observe?

In the last part, you may notice that the `length` and the `width` of the newly created object were assigned default values. To know more about this, refer to <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/default-values-table>.

---

<sup>1</sup>Rectangle.zip

## 1.3 Editing `Rectangle.cs`

Edit `Rectangle.cs`:

1. Rename uniformly `lengthParameter` to `lengthP` in the `SetLength` method (that is, replace the two occurrences). You can use the symbol refactoring<sup>2</sup> of C# to do so. Compile and run your program, what do you observe?
2. Some people use the convention of prefixing instance variables with `_` (the underscore character), `m` (for “member”), or even `m_`. You will always find someone furiously advocating for one particular convention, the truth is that if you’re not forced to use one, you should pick whichever suits you best. Still, just to use it at least once, rename uniformly `width` into `m_width` and see how it feels. Compile and run your program, what do you observe? Either undo this modification, or rename `length` into `m_length` (you have to be consistent!).
3. Change the name of one of the accessor method in `Rectangle.cs` without changing it in `Program.cs`. Compile and run your program, what do you observe? Undo your modification.

## 1.4 Enriching `Rectangle.cs`

By taking inspiration from the `ComputeArea()` method, write three new methods:

1. A method that returns the perimeter of the calling object.
2. A method that double the length and the width of the calling object.
3. A method that swap the length and the width of the calling object.

For each method, pick a (valid) name, think about the return type and the parameters, and write the body of the method carefully. After compilation succeed, call that method in `Program.cs` and see if it has the expected behaviour.

---

<sup>2</sup><https://docs.microsoft.com/en-us/visualstudio/ide/reference/rename?view=vs-2017>