# Precise Rectangle and Circle Class

https://csci-1301.github.io/about#authors

May 27, 2021 (08:06:27 PM)

## Contents



## 1 Writing Your Own PreciseRectangle Class

In this exercise, you will create your own first class instead of using and expanding one that was written for you. The idea is to take inspiration from the class you already know (`Rectangle`) to create a new class, called `PreciseRectangle`, that will manipulate rectangles whose width and length are floating-point values, instead of integers (as in `Rectangle`).

This should be a fairly straightforward exercise, that mostly re-enforce what you should know already (except for the UML part, and for how to create a class in Visual Studio).

### 1.1 Conception

Draw the UML diagram of this class: it should have two attributes, of type `double`, and eight methods:

- two setters, two getters (i.e., one for each attribute),
- one method to compute the area of a precise rectangle,
- one method to compute the perimeter of a precise rectangle,
- one method to swap the length and the width of a precise rectangle,
- one method to multiply the length and width of a precise rectangle by an ratio given in argument as an integer.

## 1.2 Implementation

To implement your class in your IDE, you are given two methods below: you can edit the pre-existing project, or start "fresh". It is recommended to pick the one you feel the most comfortable with (you will get an opportunity to start "fresh" in the next problem in any case).

### 1.2.1 Edit the Pre-Existing Project

1. Re-download the "Rectangle" project[1], extract it in a folder, and open it with your IDE.
2. Within your IDE, re-name the project to "PreciseRectangle", and rename the "Rectangle.cs" file to "PreciseRectangle.cs"
3. In the "PreciseRectangle.cs" file, replace `class Rectangle` with `class PreciseRectangle`.
4. Comment out the body of the `Main` method in "Program.cs".
5. Your program should compile as it is, but you have to edit `PreciseRectangle.cs` to now store the `width` and the `length` with `double`, and to propagate this change accordingly. What should be the return type of `GetWidth`, for instance?
6. Declare and manipulate precise rectangles (i.e., with floating-point values for the width and the length) in the `Main` method, and make sure they behave as expected (can you compute the area, for instance?).
7. Add the missing methods (`ComputePerimeter`, `Swap`, `MultiplyRectangle`).

### 1.2.2 Starting From Scratch

1. Create a new project in your IDE, name it "PreciseRectangle".
2. In the Solution Explorer, right-click on "PreciseRectangle", then on "Add..." and select "Class". Then, select "Class", write "PreciseRectangle.cs" as the name of the file, and click on "Add".
3. You are now supposed to have two ".cs" files opened and displayed in the Solution Explorer: "Program.cs" and "PreciseRectangle.cs".
4. Implement the `PreciseRectangle` class according to your UML diagram. Don't forget about the `ComputePerimeter`, `Swap`, and `MultiplyRectangle` methods.
5. Declare and manipulate rectangles with floating-point values for the width and the length in the `Main` method, and make sure they behave as expected (can you compute the area, for instance?).

# 2 Writing A Circle Class

This time, you will have to start your project "from scratch" and shouldn't try to edit a previous program.

## 2.1 Foundations

1. Create a new project in your IDE, name it "Circle".
2. In the Solution Explorer, right-click on "Circle", then on "Add..." and select "Class". Then, select "Class", write "Circle.cs" as the name of the file, and click on "Add".
3. You are now supposed to have two `.cs` files opened and displayed in the Solution Explorer: `Program.cs` and `Circle.cs`.
4. Declare one single instance variable in `Circle.cs`, of type `double` and named `radius`. Write a `set` and a `get` method for this instance variable.
5. In `Program.cs`, write statements that create a new `Circle` object and set its radius to 2.3. Display its radius on the screen using the method you defined previously.

[1] ../Rectangle/Rectangle_Solution.zip

### 2.2 Extending the Class

1. In C#, `Math.PI` is a `double` holding an approximation of $\pi$. In the `Main` method of `Program.cs`, write a statement that displays its value on the screen. It should be 3.14159265358979.
2. Now, edit this statement and use the format specifier `N`, to display the value of $\pi$ rounded to 3.14.
3. In the `Circle.cs` file, add two methods:

   a) A method that returns the circumference of the circle that calls it (i.e., $2\pi$ times the radius),
   b) A method that returns the area of the circle that calls it (i.e., $\pi$ times the radius squared).

4. Test those two methods in your `Main` program, by displaying on the screen the area and the circumference of the object you created in the previous exercise.
5. Use the format specifier `N` to round the circumference.

## 3 Pushing Further (Optional)

The following are two independent tasks, to widen your understanding of this class, and to prepare you for the next labs.

1. Class diagrams (the ones we will be using) are just a special case of UML diagram. Have a look at https://en.wikipedia.org/wiki/Unified_Modeling_Language#Diagrams. In which category are class diagrams: behaviour, or structure diagram? Have a look at https://en.wikipedia.org/wiki/Activity_diagram and try to read the example of activity diagram for a guided brainstorming process.
2. Now that you know more about naming conventions, have a look at [microsoft's naming guideline](https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines(, and particularly at

- the documentation on general naming conventions[2],
- and the documentation on capitalization conventions[3].

[2] https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/general-naming-conventions
[3] https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/capitalization-conventions