Contents

Constructors and `ToString`

This lab serves multiple goals:

- To make sure that you understand the basics of class implementation and usage,
- To practice writing and using different constructors,
- To understand the role played by `ToString` and how to call this method,
- To understand that the information stored in the attributes and its visualization are two different things.

Adding Constructors and `ToString` to An Existing Class

As a warm-up, you will practice writing constructors and `ToString` methods by adding them to a class you have already implemented and used.

A Constructor for Rectangle

1. Open the "Rectangle" project you created in the "Rectangle Class" lab[1]. You can use the "enriched" class that was shared previously[2] if you prefer.
2. In "Rectangle.cs", add a constructor to `Rectangle` that takes two arguments, a length and a width, and uses them to initialize the `length` and `width` attributes.
3. Within your `Main` method, you will notice that your `new Rectangle()` instantiation statements are now highlighted as errors. This is because `Rectangle` no longer has a zero-argument (or "no-args") constructor. Change each instantiation statement to call your new

[1] ./labs/Rectangle
[2] ./code/projects/Enriched_Rectangle.zip

two-arguments constructor. Use the initial length and width values that you previously used in `SetLength()` and `SetWidth()`.

4. Compile and execute your program and ensure your Rectangles still behave as expected.

A `ToString` Method

In the `Main` method of your program, you should have one or more statements that display the length and width of a Rectangle. For example, you may have used one when testing if your `Swap` method worked as intended; an example is shown below:

```
Console.WriteLine($"The rectangle's length is {myRectangle.GetLength()}" +
            $", its width is {myRectangle.GetWidth()}.");
myRectangle.Swap();
Console.WriteLine($"The rectangle's length is {myRectangle.GetLength()}" +
            $", its width is {myRectangle.GetWidth()}.)";
```

We will add a `ToString` method to `Rectangle` to make it easier to display this information repeatedly.

1. In "Rectangle.cs", add the following method to the `Rectangle` class:

   ```
   public override string ToString()
   {
      return $"Rectangle with length {length} and width {width}.\n";
   }
   ```

2. Within your `Main` method, find a `WriteLine` statement that displays the length and width of a `Rectangle`, and replace the calls to `GetLength` and `GetWidth` with a single call to `ToString`. For example, you can replace the statement

   ```
   Console.WriteLine($"The rectangle's length is {myRectangle.GetLength()}" +
           $", its width is {myRectangle.GetWidth()}.");
   ```

   with

   ```
   Console.WriteLine($"My rectangle: {myRectangle.ToString()}");
   ```

3. Compile and execute your program. What do you observe about the new `WriteLine` statements?

A Room Class

Now, we will create a new `Room` class. You will be writing the attributes and methods, including a constructor and a `ToString` method.

Initial Set-Up

Create a `Room` class with three attributes. They will hold:

1. the name of the room,
2. the length of the room, and
3. the width of the room.

Name the attributes the way you want and pick appropriate data types. We will want to be able to store the length and the width of rooms (expressed in meters) using floating point numbers.

Create six methods:

- three methods to set the value of each attribute ("setters")
- three methods to get the value of each attribute ("getters")

To test your `Room` class, you will need to edit your `Main` method. You will need to create a `Room` object. Set the three attributes by asking the user for the rectangle's name, length, and width. Then display the name of the `Room` object on the screen.

Adding Methods

Now, add two methods:

- A constructor that takes three arguments and uses them to initialize the length, width, and name of the room
- A method that returns the area of the room in square meters

Test them before moving on.

A `ToString` Method

This final part will guide you in writing your own `ToString` method.

1. To understand the need for such a method, start by trying to display an object "directly." In your `Main` method, create a `Room` object called `myKitchen` and write:

   ```
   Console.WriteLine(myKitchen);
   ```

   Compile and execute your program. Is the information displayed on the screen what you expected? Is it useful?

2. Add the following to your `Room` class:

   ```
   public override string ToString()
   {
         return "The name of the room is...";
   }
   ```

3. Test this method by adding the following to your `Main` method:

   ```
   Console.WriteLine(myKitchen.ToString());
   ```

4. Remove `.ToString()` from the previous statement and execute your program again. Did something change?

5. "Expand" this method by having it return a more meaningful string. This string should also contain the name of the room and its dimensions in meters. Use format specifiers to make it look nice!

Pushing Further (Optional)

This last part is here to help you understand that the data stored in attributes is not necessarily the data displayed by the `ToString` method.

Internationalization of the Room Class

Suppose that we want to accommodate users who are more familiar with feet. Note that *we do not want to change the meaning of our width and length attributes, which are still supposed to hold dimensions in meters*, but we want to create methods that perform the conversions for us. Remembering that

- 1 meter = 3.28084 feet,
- 1 foot = 0.3048 meter,

add four methods to your class:

- A method that returns the width of the room in feet,
- A method that returns the length of the room in feet,
- A method that returns the area of the room in square feet,

Then edit your `ToString` method to display the dimensions of the rooms in both meters and feet.