Constructors and ToString

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

September 29, 2021 (01:28:26 PM)

Contents

1 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 written.

1.1 A Constructor for PreciseRectangle

1. Open the "PreciseRectangle" project you created in the "PreciseRectangle and Circle" lab.
2. In "PreciseRectangle.cs", add a constructor to `PreciseRectangle` 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 PreciseRectangle()` instantiation statements are now highlighted as errors, because `PreciseRectangle` no longer has a zero-argument constructor. Change each instantiation statement to call your new two-argument constructor, using the initial length and width values that you previously used in `SetLength()` and `SetWidth()`.
4. Compile and run your program and make sure your PreciseRectangles still behave as expected.

1.2 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 PreciseRectangle, e.g. to test the result of your `Swap` method, like this:

```
Console.WriteLine($"My rectangle has length {myRectangle.GetLength()} and width
 ↪  {myRectangle.GetWidth()}");
myRectangle.Swap();
Console.WriteLine($"My rectangle has length {myRectangle.GetLength()} and width
 ↪  {myRectangle.GetWidth()}");
```

We will add a `ToString` method to `PreciseRectangle` to make it easier to write these statements.

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

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

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

   ```
   Console.WriteLine($"My rectangle has length {myRectangle.GetLength()} and width
    ↪  {myRectangle.GetWidth()}");
   ```

   with

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

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

2 A Room Class

Now, we will create a `Room` class "from scratch," including constructors and a ToString method.

2.1 Initial Set-Up

Create a `Room` class, with three attributes: one to hold the name of the room, one for the length of the room, and one for the width of the room. Name the attributes the way you want, and pick appropriate datatypes, knowing that we want to be able to store the length and the width of rooms (expressed in meters) using floating point numbers.

Create 6 methods:

- A method to set the value of each attribute ("setters")
- A method to get the value of each attribute ("getters")

To test your `Room` class, edit your main method to create a `Room` object and ask the user for its name, length and width, and then display on the screen the name of the Room object that was created.

2.2 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.

2.3 Internationalization of the Room Class

Suppose that we want to accommodate users who are more familiar with feet. Note that *we don't 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,
- A constructor that takes two arguments, a length and a width in feet, and create an object with the corresponding measures in meters. In this constructor, you should pick a sensible default value for the name of the room, since it will not be supplied in an argument.

Try to write these methods using constants for the conversion factors, and test them before moving on.

2.4 A ToString Method

Finally, create a `ToString` method. 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?

Add the following to your `Room` class:

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

1. Test this method by adding

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

   to your main method.

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

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