Step 4 – Properties

* Explain why we have properties
* Construct a properties
* Explain Getters
  + Demonstrate Creating one, and then what happens if we remove it
* Explain Setters
  + Demonstrate creation, then what happens when we have one and when we don’t
* Remove public from our fields

Tutorial:

When we set our fields to **public** we give complete access over them. The users can change the values to anything they want. Sometimes we want to restrict the type of information we want users have access to or change. And some fields they don’t need access to at all, as they are only used by the class. To give the user proper access to fields we use **Properties**.

In our Course.cs and beneath our commented out **Properties** type the following code

public string Name {

get {

return \_name;

}

set {

\_name = value;

]

}

Let’s work through this code. Starting at the top we have our public access modifier again. Followed by a return type of string. **This is the same exactly type as the field**. Then we give it a name. Here we are creating a property for \_name, so we do Name, upper case, no underscore. Properties don’t have parenthese in c#, so we just have curly braces.

Inside our **property** we have some new syntax. We see a **getter** and **setter**. Get and set are specific keywords in c#.

**Get** is used to give access to the user so they can display it from our code. It’s lower case **get** followed by curly braces. Inside of our curly braces we place **return \_name**. That returns the information from our \_name field.

**Set** gives the user the ability to change the associated values. **set** is lower case followed by curly braces. Inside we place **\_name = value**; Notice that value turns dark blue. That’s because it’s another c# keyword. In this case value represents whatever the user is trying to assign to the \_name field;

Variable name = value;  
course1.Name = “Programming 122”;

Now go back up to the top of our Course class and remove “public” from the front of our name field.

string \_name;

And go back to program.cs

You should see an error where we try to display our course name.

Console.WriteLine($"Course Name: {course1.\_name}");

That’s because we’ve made that field **private**, or restricted access to that name, **\_name**, to only the class. But now we can use our **Property** we created. Change our error to course1.Name.

Console.WriteLine($"Course Name: {course1.Name}");

We should now be able to display our information like normal.

Now before our Console.WriteLine() where we display the name, let’s assign a new value.

course1.Name = “Programming 252”;

And run the code again. You should see the name display Programming 252. That’s because we just assigned a new value using our setter.

To get a better idea of what getters and setters are doing, go back to Course.cs.

And comment out the getter.

And now go back to program.cs. Notice that we get an error on our Console.WriteLine(). This is because we removed the getter, preventing the user from accessing that information. Go back, uncomment the getter, and now comment out the setter, and go back to program.cs.

Now you see that we have an error where we tried to change the value. It’s **Read-Only** access. That’s the power of getters and setters. They give strict access to only the information we want.

Now go back to Course.cs.

1. Uncomment the setter in Name
2. Create properties for CourseNumber, Teacher, and Student
3. Remove public from all the fields.
4. Fix your code in Program.cs so it’s displaying information properly.

Great. You now understand

* How to create properties.
* How to strict access to fields
* How to create getters and setters
* Why we would want to create getters and setters
* How to use our properties.

In our next step were going back to program.cs and going to create a list of Courses. And create a method to display them.