Step Two – Fields:

Create and explain fields

* Explain why we consider two fields
* User place holder for teacher and strings
* Make them public ( Save access modifies for later )
* Create a Course instance in Program ( explain instance )
* Have change name, teacher, and student ( explain dot modifier )
* Have display name, teacher, and student

Tutorial – Step Two:

When thinking about our course class we want to think about variables that might be associated with it. This doesn’t have to be comprehensive or even complete as we start our class, as we can always change our code later. But we do want to think out some requirements

Let’s think about what a course may have. We know it needs a name. And it may have a course number that’s related to the school database. Also, a course will have a teacher and students. For now we are going to create 4 string fields to hold this information. Put the world public in front of all the fields.

**Get Screen shot of creating the fields**

string \_name;  
string \_courseNumber;  
string \_teacher;  
string \_student;

When creating field names, make them descriptive and start them with an underscore. This helps identify them as fields.

Logically a teacher and students are more than just a string, but we will come back later to change that. For now we have a good starting point.

* Switch To Program.cs

Now lets create an **INSTANCE**  of our new course class.

Inside main in Program.cs declare a Course variable, similar to how we create arrays and lists.

Course course1 = new Course();

This creates a new **Instance** of a course. If a class is a like a blueprint, the instance is like an individual product made from the blueprint. Everytime you create a new instance of an object it will have it’s own fields, but they we be separate from other instances of the same object.

Now let’s assign some values to a course1 fields.

To access fields, and in the future properties and methods, in a object you use the dot operator. It’s just a period. You’ve already been doing this when you do Console.WriteLine() or rand.Next()

We will use Console.WriteLine() to display information from our instance.

Notice: If you don’t see your field names appear, make sure that you have the word **public** in front of them in your **Course** **class**.

Console.WriteLine($”Course Name: {course1.\_name}”);  
Console.WriteLine($”Course Number: {course1.\_courseNumber}”);  
Console.WriteLine($”Course Teacher: {course1.\_teacher}”);  
Console.WriteLine($”Course Student: {course1.\_student}”);

Now run your code

You should see a result of

Course Name:   
Course Number:   
Course Teacher:   
Course Student:

So, what happened? Well our code is working properly, but when we created our **Course instance** we never assigned values to the fields. Now above our Console.WriteLines(), lets assign our values.

course1.\_name = “Programming 122”;  
course1.\_courseNumber = “CSI\_122\_2”;  
course1.\_teacher = “William Cram”;  
course1.\_student = “Hannah Angel”;

Notice we assign values the same way we do with strings, that’s because we declared all of our fields as strings.

Now let’s display that information again, run your code.

Now run your code

The result should show.

Course Name: Programming 122  
Course Number: CSI-122-2  
Course Teacher: William Cram  
Course Student: Hannah Angel

As we see, all this information is now related by this single instance. The class object lets us group information together in a logical manner.

Excellent. You’ve now

1. **Created a Course class**
2. **Create fields in the Course class**
3. **Create a new instance of Course**
4. **Assigned values to it’s fields**
5. **Displayed information from it’s fields**

In our next step we will look at **Class Constructors** and how to limit or permit the information needed when creating the **instance** of a new object.