Step Three – Constructor:

* Explain the purpose of a constructor
* Naming a constructors
* Explain the default constructor
* Explain creating our own constructor and how it gets rid of the default constructor
* Calling your constructor

Tutorial

Return back to our **Course** **class** and lets take a look at creating a **Class Constructor**.

In our program.cs you’ll remember that we created a new instance of an object with

Course course1 = new Course();

When we did that, new Course() was calling a class constructor. This was saying we creating a new **instance** of this object in memory. This was actually a **default** constructor. That’s a constructor that takes no arguments. This constructor is automatically created if no other constructors are written for the class.

As we say when we ran our program the first time before we **assigned values** we had no information to display. Logically we would never want to have a record of a course without at least having a class name and number. So we are now going to create a constructor in our **Course class** to prevent users from creating a **Course instance** with no data.

Underneath our commented our constructor write

public Course() {

}

Let’s take a look at this. We notice that a constructor isn’t too different from a method, but there are some key differences.

First we start with public, because our code in program.cs needs access to this Constructor. That’s because public is an “access modifier”. This allows other pieces of our code access one another. We will go into more detail on this later. For now we start with public.

Next notice we do not have a return type. Constructors do not have returns type.

Then we give it a name. The name **has to be the same name as the class**. Here our class name is Course. If you try to give it a different name, you will get an error.

After that we have some parenethese with nothing in between and a set of curly braces.

This is a basic constructor. This is exactly what a default constructor is. We allow the user to create an object with it, but don’t actually take any values for our fields. Take a quick look at our program.cs and notice that everything is behaving like before. We’re about to change that.

Return back to Course.cs and back to the constructor we created. We know we don’t want the user to be able to create an **instance with no values** so we are going to change our constructor

In the parentheses, create two parameters, one for each field that we want to make sure are assigned values when an instance is created. For use these are \_name and courseNumber

string name and string courseNumber.

public Course(string name, string courseNumber) {

}

Notice how I game them the same names as the two fields we want to have filled in. But I didn’t do the underscore. That’s to make it easy to read, but not confuse the fields and the parameter names.

This is a start, as we’ve created a constructor that takes two arguments. But now we need to make sure the values passed in are assigned to our fields.

Inside the constructor we are going to connect our fields and aguments.

public Course(string name, string courseNumber) {

\_name = name;

\_courseNumber = courseNumber;

}

With our new code we assign our fields, the variables that start with an underscore, with our arguments, the variables being passed in.

Lets go back to our program.cs

* Back to program.cs

You should now see an error where we call **new Course()**. That’s because we have changed our constructor to take 2 arugments, name and course number. Since we created a constructor we no longer have a default constructor, or one that doesn’t take any arguments. Hence the error.

Lets fix our code by passing in two arguments into our new constructor.

Console course1 = new Course(“Programming 122”, “CSI\_122\_2”);

Our error should have gone away.

Now remove all the lines of code where we assign values

// Delete These

course1.\_name = "Programming 122";

course1.\_courseNumber = "CSI\_122\_2";

course1.\_teacher = "William Cram";

course1.\_student = "Hannah Angel";

Now run our code.

You should see

Course Name: Programming 122  
Course Number: CSI-122-2  
Course Teacher:   
Course Student:

The first two lines should be filled out because we passed our arguments into the constructor.

The last two are empty because we deleted our code that **assigned values** to teacher and student. While we could edit our constructor to take these arguments too, it’s not needed as logically we can have a class before have a teacher or students in it.

Excellent. You’ve now

1. **Created your own constructor that takes required information for our instance**
2. **Called that constructor and passed in the proper arguments**
3. **Displayed information from it’s fields**

So far we’ve been using our fields to change and display information, but this is very bad coding practice. We are going to refactor our code and switch to **Properties**.