Step 6 – Create the Teacher class

* Create fields,
  + First Name - string
  + Last Name – string
  + Teacher Class – SchoolClass ( explain object in object ) // Hold off till later
* Create a constructor
  + Must take a first and last name ( a teacher can be an employee without a class )
* Create properties
* Create a Teacher object
* Display Teacher first and last name
* Add a class to the teacher
* Display teacher and class information
* Create a teacher list
* Add two more teachers
* Add classes from Class list

Tutorial:

To get started, let’s add a class like we did our first one, and name it Teacher.

Right click on one of the top two objects in the solution explorer. Add -> Class -> Teacher.

namespace Tutorial\_Classes

{

internal class Teacher

{

// fields

// constructor

// properties

// method

} // class

} // namespace

Notice I added the comments to help me stay organized when creating my classes.

Now we can follow the same steps we did for the Course class,

* Create two string fields for first and last name.
* Create a constructor to take a first and last name
* Create the properties to access the first and last name.

// fields

string \_firstName;

string \_lastName;

// constructor

public Teacher(string firstName, string lastName)

{

\_firstName = firstName;

\_lastName = lastName;

} // Teacher

// properties

public string FirstName

{

get

{

return \_firstName;

}

set

{

\_firstName = value;

}

} // FirstName

public string LastName

{

get

{

return \_lastName;

}

set

{

\_lastName = value;

}

} // LastName

This should be very similar to how we created the Course class. As your experience grows your classes will become far more complex, but for now this will suit our needs.

Go back to Program.cs

Following these steps, let’s display our information about our teacher.

1. Create a global List<Teacher>, call it teachers.
2. In main create a new instance of teacher
   1. Pass one of your teachers names as the argument to the constructor
3. Add that instance of teacher to your list.
4. Display the information to the screen.
   1. Ex. Teacher Name: William Cram
5. Add two more teachers.
6. Now create a method that displays all the teachers.

This whole process is very similar to what we did for courses. Feel free to jump back to those steps if you need a refresher.

Before we move on, we want to now associate a Teacher with a class. But how can we do that, since we just assigned a string for teacher in our Course class.

Our fields can be more than just primitive types, they can be instance objects too.

Return back to the Course.cs class, were about to refactor our code.

Back to Course.cs

In the course class, change the string type associated with \_teacher, to Teacher. The new class we just made.

internal class Course

{

// Fields

string \_name;

string \_courseNumber;

Teacher \_teacher; // This is now a teacher object

string \_student;

// Constructor

Now we can attach an instance of Teacher to our Course. So as our Teacher object has more or less data, we can access all of it via our class.

You will find some red marks in your Course class, especially in the Properties. Just change the return type from string, to Teacher.

public Teacher Teacher

{

get

{

return \_teacher;

}

set

{

\_teacher = value;

}

} // Teacher

Now lets go back into our program, and assign a teacher to a class.

Back to Program.cs

Let’s tweak our code a little bit. By now we should have a list of teachers and courses. We are going to do two things.

1. Assign a teacher to a course. Of if you want, teachers to every course.
2. Edit our DisplayCourses to display the teacher name.

To start off simple, we are going to attach the first teacher in our list, to the first class in our list.

In main, after your code where you’ve added teachers and courses, add this code.

courses[0].Teacher = teachers[0];

This assigns the first teacher in our list to the first class spot for teacher. Notice how we access .Teacher to assign the value.

Now we need to tweak our DisplayCourses()

Change

Console.WriteLine($"Course Teacher: {courses[i].Teacher}");

To

Console.WriteLine($"Course Teacher: {courses[i].Teacher.FirstName} {courses[i].Teacher.LastName}");

Because we are now accessing the Teacher object INSIDE of our courses object. So we can get the FirstName and LastName fields.

Run your code.

Text

Description automatically generated

If your project crashed, that’s alright. We expected this. Our first course printed perfectly, teacher name and all. But then it crashed on the second course. What happened?

Unlike primitive data types, instance objects don’t have a default type. They are just **null**. That means they don’t exist in memory at all. This isn’t even equidistant to zero, this is nothing. And when the computer ran into a null that it wasn’t prepared for, it threw an error. So let’s change our code to prepare for that.

Go back to Course.cs.

Inside of our constructor we are going to add another line of code.

public Course(string name, string courseNumber)

{

\_name = name;

\_courseNumber = courseNumber;

\_teacher = new Teacher("None", "Assigned");

} // Course

At the bottom we add

\_teacher = new Teacher(“None”, “Assigned”);

But we don’t pass in a teacher argument. What we are doing is creating a default Teacher object when the constructor is called. If we assign a teacher later on in Program.cs, it over rides this one. Otherwise it creates a Teacher instance and assigns it to this class.

Rerun your code and check the results.

Text

Description automatically generated

Now because we’ve created a place holder Teacher instance, it prints out the info from there. And since we passed in “None” and “Assigned” as the arguments, that’s what it displayed. This is not optimal, but it’ll do in a pinch.

In this step you’ve

* Created a new class, Teacher.
* You created fields, a constructor, and methods.
* You added a List<Teacher> to your Program and added 3 teachers.
* Then you refactored your **Course.cs** code to take a teacher instance.
* You also tweaked the constructor and property of course.
* Then you assigned a teacher to a class.
* Afterwards you altered your DisplayCourses() to properly display the first and last name of the teacher assigned to the class.

That’s a lot of work! Well done. You’re well on your way to having a working app. In the next step we are creating our last class. The student class.