Step 5 – Create a list of Classes in Program

* Create a single Course object
* Display the information from the object
* Create a list of Course objects;
* Add that original class, display the information using the list.
* Create two more classes, add them to the list.
* Create a method that can be used to display the classes.

Tutorial:

Now that we have our Course class built, lets tweak our program a bit so we can work with a list of Courses instead of just one course.

In **Program.cs** our code currently looks like

Course course1 = new Course("Programming 122", "CSI\_122\_2");

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

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

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

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

Which works for one course, but schools have hundreds. Usually we keep track of them in a database, but for now we are going to mimic this with a list.

At the top of Program.cs, outside of Main, declare a new List<Course>

static List<Course> courses = new List<Course>();

We declare it in the global scope, outside of a method block, so everything in our class can have access to it.

Now in main, lets alter our code so that we add our original Course object to our list, and then read from the list.

Underneath our course1, we are going to add that variable to our courses list.

static List<Course> courses = new List<Course>();

static void Main(string[] args)

{

Course course1 = new Course("Programming 122", "CSI\_122\_2");

courses.Add(course1);  
 …

Remember to add an item to a list, you need to call **.add** on the list name, here it’s courses. Then pass in your course1 object.

Adding the course to your list places it at the end of the list. Since this is the first item added, then we can test it by access the element at index 0, or the first item in a list / array.

Change your Console.WriteLine() to use the list, instead of the course1 object.

Console.WriteLine($"Course Name: {courses[0].Name}");

Console.WriteLine($"Course Number: {courses[0].CourseNumber}");

Console.WriteLine($"Course Teacher: {courses[0].Teacher}");

Console.WriteLine($"Course Student: {courses[0].Student}");

Run your code and observe the result. It should look the same as before.

The only difference is that we used the list and index to access the instance of the information, instead of just the single variable. Because of this we can now easily add more courses and interact with them.

Let’s add two more classes to our list.

course1 = new Course("Programming 120", "CSI\_120\_2");

courses.Add(course1);

Under where we add the first course, we are going to reuse our **course1** variable, instead of declaring a new one. By using the **new** keyword, we create a new instance of the **Course** object. This doesn’t replace the old information. In fact you will see we can still access our original information since we added our original instance to the List<Course>.

courses[0].Name will display “Programming 122”. After we reassign course1, course[0].Name still display “Programming 122”. While course.Name will display “Programming 120” because we created a **new** **instance.** Now we add that to the list.

Course course1 = new Course("Programming 122", "CSI\_122\_2");

courses.Add(course1);

course1 = new Course("Programming 120", "CSI\_120\_2");

courses.Add(course1);

This adds the newly created instance to our list, at the second index, 1. So now you can access the info using courses[1].Name, etc… Feel free to test it out.

Let’s add a third class, but avoid declaring a new variable altogether.

courses.Add(new Course("Programming 252", "CSI\_252\_1"));

Underneath where we add our other two courses to the list, we add another. But instead of using course1 and reassigning a value, we create the **instance** directly in our .Add().

Note: When you create an instance, all the data is stored in the **heap**, and when it’s assigned to a variable name, that name is just an address to the location in the heap.

Store in stack stored in heap

Course course 1 = new Course(“Programming 310”, “CSI\_310”);

When you add it to a list, the list name and index ACTS like a variable name. So creating a **new** instance and passing it into the list just assigns an index as the address to the **instance** on the heap.

So passing new Course into .Add is the same as

Courses[2] = new Course(“Programming 310”, “CSI\_310”);

All three of these techniques work to add objects to a list, choose whichever you feel comfortable with.

Course course1 = new Course("Programming 122", "CSI\_122\_2");

courses.Add(course1);

course1 = new Course("Programming 120", "CSI\_120\_2");

courses.Add(course1);

courses.Add(new Course("Programming 252", "CSI\_252\_1"));

Now that we have added 3 courses to our list, it’s starting to get difficult to display information about each one. So lets create a method that we can easily call to display our classes when we want.

First, create a new method outside of the Main code block and call it DisplayCourses()

public static void DisplayCourses()

{

} // DisplayCourses

Now let’s take our Console.WriteLine() from our main, cut them, and place them inside of our method.

public static void DisplayCourses()

{

Console.WriteLine($"Course Name: {courses[0].Name}");

Console.WriteLine($"Course Number: {courses[0].CourseNumber}");

Console.WriteLine($"Course Teacher: {courses[0].Teacher}");

Console.WriteLine($"Course Student: {courses[0].Student}");

} // DisplayCourses

Right now if we called it, this will still work! Since it is access our courses list which is global. But this will only work for a single class, courses[0]. Let’s tweak it to display all of our classes.

Using a for loop, let’s have this display all of our classes.

public static void DisplayCourses()

{

for (int i = 0; i < courses.Count; i++)

{

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

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

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

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

Console.WriteLine();

}

} // DisplayCourses

With a standard for loop, I place all our Console.WriteLine() inside of it. I make sure to loop through courses.Count. Remember, that list’s use .Count, instead of arrays.length. I also changed the 0 in the brackets to i.

I also added an empty Console.WriteLine() to add an extra space between our course display. Now call your DisplayCourses() in the main and check out the results.

Text

Description automatically generated

This is looking good! Now whenever you add a course to the list, it will automatically be displayed with our new method.

Great. You now understand

* How to create a list to hold our new Course
* Multiple ways to add a course to our list
* How to display one object from a list
* Create a method that we can use to display every object from our list.