**INT422 – Lecture 1-2**

Hands-on with Visual Studio. Writing a C# class. Working with a controller and views. Begin work on Assignment 1.

**Typical agenda for Lab sessions**

On your timetable, this course is a four-hour course. The Thursday (section A) or Friday (sections B) session, in a computer-lab room, is part of the course. It is not optional, and you cannot skip it. In each lab session, there will be in-class grading which will make 10% of the assignment mark.

During our lab sessions, we will typically have a mix of discussion/lecture, and hands-on activity. The hands-on activity may help you learn the application of a concept or technique, and will also typically include some time where you get started on the weekly assignment.

Also, the lab sessions give you an ideal opportunity to get help from your professor. We can help then, or during the “student help time” usually in the same lab room or in the open lab. We are not available at 3:00am to help you by email/online, so take advantage of the scheduled class times and office hours.

**Textbook coverage**

Chapter 2 – Controllers – skim the content

Chapter 3 – Views – skim the content

**Code examples**

For this course, the code examples will be published weekly, on the course website.

During the course’s lifetime, about thirty code examples will be published.

As you have observed, each ASP.NET web app or web service project includes a sizable “packages” folder. Each code example does not include the “packages” folder, as a time-saving measure. Each “packages” folder is about 50MB in size, so twenty code examples would require about 1GB storage. The contents of the “packages” folder does not compress much.

Therefore, if you have downloaded and opened a project in Visual Studio, do a build/compile using Ctrl+Shift+B (Build>Build Slution), or start/run the project using Ctrl+F5 (Debug>Start Without Debugging) before you begin working with it. These build/compile and start/run actions will fetch the packages it needs from the web.

All code examples are designed to be usable immediately and error-free. Each should compile and execute correctly.

If you have any problems with a code example, please contact your professor.

.

**Console applications for prototyping and testing**

A new ASP.NET MVC web app or web service project has many files and folders, and “moving parts”. The task of quickly prototyping or testing an idea can be burdensome. You have to work with a controller, a view, and perhaps other classes (HttpApplication, “startup”, or Manager classes).

How can you quickly prototype or test an idea?

With a Console Application.

File > New > Project, then in the Visual C# group, choose “**Console** Application”.

Run a console app with Ctrl+F5 (Start Without Debugging). It will open a text-based window.

The Console.WriteLine() static method will “write” or send text content to standard output.

Use Console.Read() (or ReadLine()) to get user input. Or, just code the data you need in variables.

The three Week 1 code examples are console apps. Enjoy.

**Collections**

Recently, you were introduced to the job of writing a class to model a real-world object.

Now, it’s time to learn about collections.

When we work with data in our web apps, we are going to do lots of work with objects, and collections of objects.

In lecture 1, there is a sample class *Manufacture* that have a navigation property *Products* with data type *ICollection<Product>*. However, the *Products* property is initialized to an object of *List<Product>*. (why?). Both *ICollection* and *List* are examples of using collection.

The following is from the MSDN Library [Collections and Data Structures](http://msdn.microsoft.com/en-us/library/7y3x785f.aspx) document:

❝Closely related data can be handled more efficiently when grouped together into a collection. Instead of writing separate code to handle each individual object, you can use the same code to process all the elements of a collection.

To manage a collection, you can … add, remove, and modify either individual elements or a range of elements in the collection.

Here’s some more information, from the MSDN Library:

[Commonly Used Collection Types](http://msdn.microsoft.com/en-us/library/0ytkdh4s.aspx) (we will use ICollection<T>, List<T>, and IEnumerable<T>)

[When To Use Generic Collections](http://msdn.microsoft.com/en-us/library/ms172181.aspx)

[Introduction to Generics](http://msdn.microsoft.com/en-us/library/0x6a29h6.aspx) (first paragraph only)

❝Note: In C++, you are familiar with templates. In C#, this concept is known as generics. Same idea, with some differences.

[List<T> Class](http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx) reference

A collection can be created and used anywhere in your code. We also see collections as data types for some properties in data classes.

When we declare a collection property in a data class, we use one of two collection types:

For objects that browser users interact with, we use the data type IEnumerable<T>.

For objects that are defined by the persistent store model, we use the data type ICollection<T>.

The **T** is a type name **placeholder**, which will be replaced by the actual type of the object in the collection (e.g. IEnumerable<Person>, IEnumerable<Vehicle>, IEnumerable<Product>, etc.).

Reference information is described in the [IEnumerable<T>](https://msdn.microsoft.com/en-us/library/9eekhta0%28v=vs.110%29.aspx) and [ICollection<T>](http://msdn.microsoft.com/en-us/library/92t2ye13%28v=vs.110%29.aspx) documents. The “Remarks” and “Examples” sections of these document will help you learn.

When you add a collection property, you MUST do one other task: In the default constructor, initialize the property by using **a concrete type**.

Both IEnumerable<T> and ICollection<T> are [interfaces](http://msdn.microsoft.com/en-us/library/ms173156.aspx). You can tell, because .NET Framework interfaces have an upper-case letter “I” prefix. In C++ programming, you learned about an abstract class with pure virtual methods. In C# and the .NET Framework, the counterpart is an interface.

A best practice tells us to use an interface as the data type when declaring properties in a class. However, just as in C++, we cannot initialize an interface. Instead, we must use a concrete type that implements/inherits the interface. (Why don’t we just declare the property’s data type as the concrete type? Well, it’s not a best practice. You’ll understand more later.)

So, in a data class that has a collection property (probably named “Products”, of data type “Product”), we write the following code in the default constructor:

|  |  |
| --- | --- |
|  | **Products = new List<Product>();** |

**More about the Razor View Engine**

The textbook has good coverage of this topic in chapter 3, “Views”.

The Razor View Engine is used during request processing to generate an HTML response. The view is the source code file that contains markup and code expressions.

* The markup language is HTML
* The code expression language is C#

Begin a code expression with the ‘at’ sign, @. Then use C#. Razor will transition back to markup at the end of the code expression (in most situations).

In a view, the “Model” property is the strongly-typed model object that is passed in from the controller:

❝*The first line of code in the view should declare the model object’s type…****@model*** *IEnumerable<MyProject.Controllers.PersonBase>*

*Then, in the markup, it is dereferenced with “****Model****”.*

*Notice the difference – in the declaration, “model” begins with a lower-case letter. However, the “Model” property begins with an upper-case letter.*

The “ViewBag” property provides access to other data that is passed in from the controller. It is dereferenced with “@ViewBag”. You should try to avoid its frequent use.

A multi-line code expression is placed in a code block: **@{ }**

Use parentheses to remove ambiguity: @( )

**Introduction to “scaffolding”**

This is a feature of the framework that’s well-loved by developers.

Question – what web app programming task is annoying?

Answer – for most people, its programming pages that perform common data operations – get-all, get-one, create, update, and delete.

[Scaffolding](http://www.wikiwand.com/en/Scaffold_%28programming%29) helps address this task. We can build controllers easier, and views easier. Although some of the results may not make a web designer happy, they enable us to get results fast, and build a base that can be extended.

The ASP.NET MVC designers didn’t invent this approach. It was a well-established feature by the time they got started in 2009. Every major web app framework does this now.

**Work on Assignment 1**

Your professor will guide students as we get started on Assignment 1.

The assignment specifications document is much longer than it will be for later assignments. Why? We want to include details and screenshots to help you get started in a productive and informed manner. We also use it to teach you some things about ASP.NET MVC web apps, as you work on the assignment. Have fun.

Before you leave the room at the end of the time slot, ensure that your professor has checked your work, for the in-class grading part of the assignment.