BTI420 - Web Programming on Windows

Lecture 1 Part 2

Collections

- When we work with data in our web apps, we are going to do lots of work with ***objects***, and ***collections of objects***.
- The following is from the MSDN Library Collections and Data Structures 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 (we will use **ICollection<T>**, **List<T>**, and **IEnumerable<T>**)
  - When To Use Generic Collections
  - Introduction to Generics (first paragraph only)
  - List<T> Class reference

Collections

- 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>
- Reference information is described in the IEnumerable<T> and ICollection<T> documents. The "Remarks" and "Examples" sections of these document will help you learn.

Collections

- 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. 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>();
```

Lab Activities

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

The End