String Interpolation

Inserts a substring into a string literal. Example:

string substr0 = “ABCDE”;

int n0 = 17;

string sLiteral = “Alphabet xx piece.”;

To replace “n” with n0, and “xx” with “ABCDE”, the old way is

String sNewLiteral 1 = n0.ToString() + “Alphabet ” + substr0 + “ piece.”

Using String Interpolation --

String sNewLiteral2 = $”{n0} Alphabet {substr0} piece.”;

to get the same result.

Property in Class (Get and Set)

For example “Book” is a class that has “Name” as one of its properties.

public class Book

{

public string Name;

. . . other fields

. . . constructor and methods

}

Instead of allowing an invoking module unmanaged access to “Name”, it is preferable to write functions that control access.

public class Book

{

public string Name

{

get; // the invoking module can use book.Name to get the name of the book

private set; // methods in class Book can set the value of Name, but invoking

} // modules can not do so.

. . . other fields

. . . constructor and methods

}

To make “set” public, but include validation code, use more verbose syntax such as

private string name

public string Name

{

get

{

return name;

}

set

{

if (string.IsNullOrEmpty(value))

throw new ArgumentNullException("Value cannot be null");

if (value.Length > 128)

throw new ArgumentOutOfRangeException("Value cannot be longer than 128

characters");

name = value;

}

}

readonly and const property in Class

When you give a field in a class the “readonly” property, this allows the field to be set to a value only via the class’s constructor. A field with the “const’’ property does not allow the field to be modified, even by a constructor. “const” can be used inside a method; it is typically used to provide a meaningful name to a literal. When “const” is used outside the context of methods, it applies to the entire class.

Delegates and events in Classes

The instructor suggested that delegates and events will probably not be used much in Core Web projects. In other contexts they might be used. Refer to the last 5 topics under [Building Types], specifically [Introducing Events and Delegates], [Defining a Delegate], [Using Mult-icast Delegates], [Defining an Event], and [Subscribing to an Event].

In addition to events, delegates are used for call-back (where a function is passed as an argument to a method, and the method invokes the function). A common example of a call-back occurs with a “sort” method; when “sort” is invoked, a comparison call-back function is provided as an argument; the comparison call-back takes 2 arguments – x and y; it returns -1 if x < y, 0 if x = y, and 1 if x > y.

A recent improvement in C# are “Lambda Expressions”. Refer to <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions>

Also see “Delegates, Anonymous Methods and Lambda Expressions” <https://itnext.io/delegates-anonymous-methods-and-lambda-expressions-5ea4e56bbd05>

Polymorphism

Polymorphism deals with the keywords “virtual”, “abstract” and “interface”. I did not skip this subject, but I did not absorb it carefully. The sections that deal with this are in [Object Oriented Programming].[Setting Up a Scenario], [Object Oriented Programming].[Defining an Abstract Class], [Object Oriented Programming].[Defining an Interface], and [Object Oriented Programming].[Writing Grades to a File]. There is only a very low probability that I will need to use Polymorphism in the code to be developed for Current Pixel. But if the need arises, I could perhaps return to the sections cited above to learn them more carefully.