WEB524

WEB PROGRAMMING ON WINDOWS

WEEK 1 – LECTURE 2

INTRODUCTION TO THE .NET FRAMEWORK

Seneca

Resources

Much of this lecture is pulled from the various resources listed below:

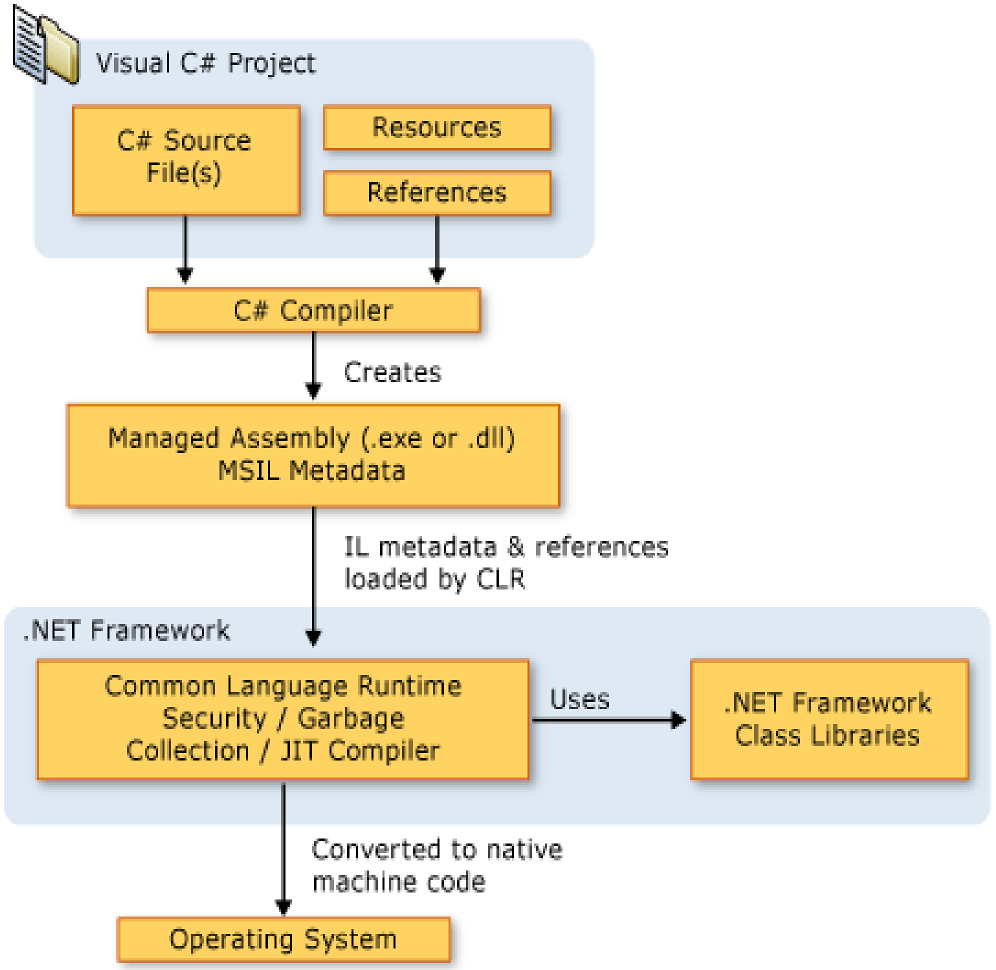
- Classes and Structs
- Introduction to the C# Language and the .NET Framework
- C# for C++ Developers
- C# for Java Developers
- Introduction to Generics
- Collections and Data Structures
- Commonly Used Collection Types
- Interfaces

C#

- Elegant and type-safe object-oriented language.
- Uses the curly-brace syntax similar to C, C++ and Java.
- Features:
  - Nullable value types – even primitive types
  - Enumerations – list of named constants
  - Delegates – type safe event notification
  - Lambda expressions – e.g. inline functions
  - Generic methods and types – used often in collections
  - Attributes – declarative metadata (used often in MVC)
  - Inline XML documentation
  - Language-Integrated Query (LINQ) – built in query capabilities
  - No separate header files and no requirement to sort methods/types

.NET Framework

- C# programs run on the .NET Framework.
- C# is compiled into an intermediate language (IL) – .exe or .dll file
- Common Language Runtime (CLR) performs Just-In-Time (JIT) compilation to convert code to native machine instructions.
- CLR provides: garbage collection, exception handling, resource management.
- Code executed by the CLR is called "Managed Code" as opposed to "Unmanaged Code" which is compiled into native machine language that targets a specific system.


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

.NET Framework

- Intermediate code from C# can interact with other .NET languages like Visual Basic, Visual C++, plus many more.
- A single assembly can contain multiple modules written in different .NET languages.
- Includes an extensive library of thousands of classes organized into namespaces for performing file operations, string manipulation, XML parsing, and more.

Classes

- The class is the most fundamental container, or building block, for your code.
- You will use classes to define:
  - data objects (i.e. classes that are things)
  - objects with behaviour (i.e. classes that do things)
  - both data and behaviour.
- A web app contains a lot of classes! Controllers and models are both classes.
- Code for a class is placed within a namespace:
  - Used for organization of classes
  - Used to control the scope in larger projects

Classes

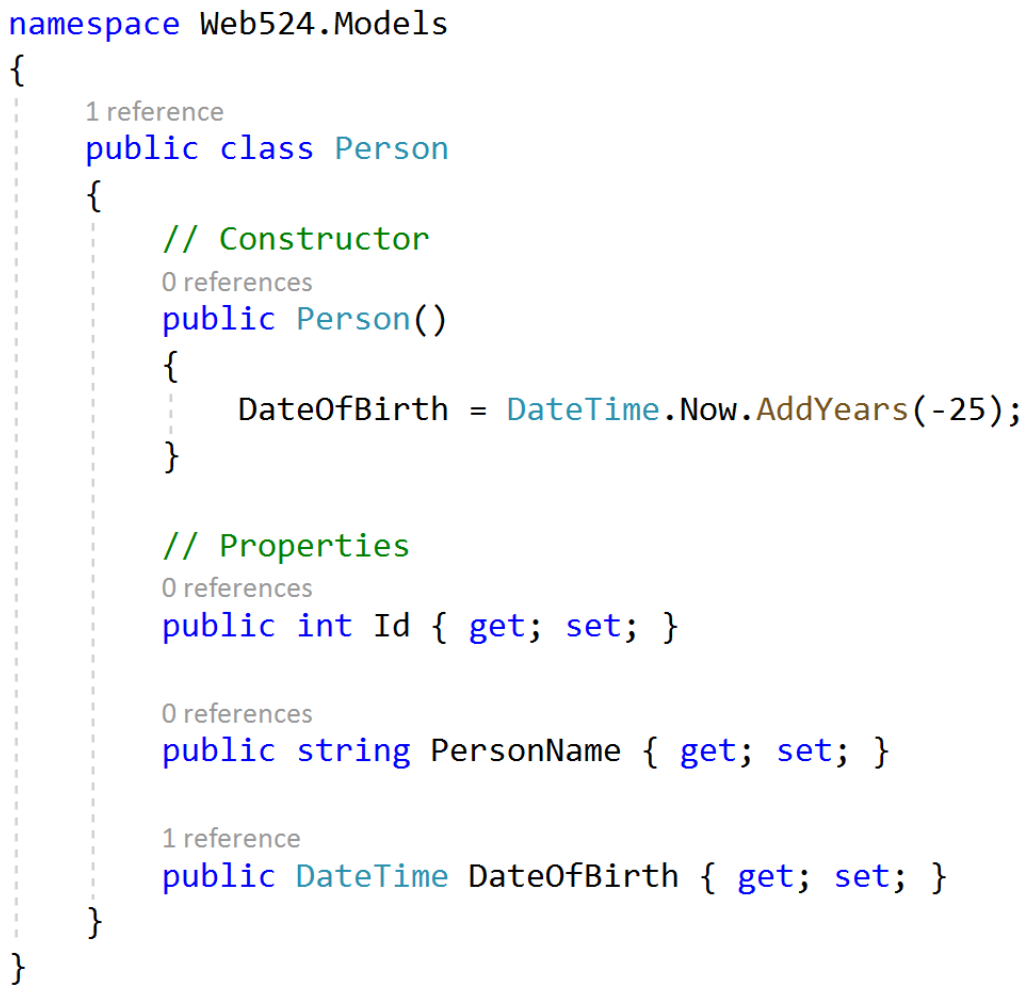
- A class will include members.

- Common members include:
    - Private fields, not visible outside the class
    - Constructors, for creating a new instance of the class
    - Properties, for holding publicly-available data
    - Methods, public or private, for behaviour
    - There are a few other kinds of members that can be added.

- Try not to nest a class inside another class. It is better to place the two classes side-by-side within a namespace.

Modelling Entities with Classes

- A model class needs properties, maybe one or more methods, and one or more constructors. It should always have a default constructor.
- Omitting the constructor will assume a public empty constructor.
- Assume that you want to model a “Person”. Here is some sample code:



```
namespace Web524.Models
{
    1 reference
    public class Person
    {
        // Constructor
        0 references
        public Person()
        {
            DateOfBirth = DateTime.Now.AddYears(-25);
        }

        // Properties
        0 references
        public int Id { get; set; }

        0 references
        public string PersonName { get; set; }

        1 reference
        public DateTime DateOfBirth { get; set; }
    }
}
```

Generics

- In C++ you are familiar with **templates**, in C# this concept is known as **generics**.
- Generic classes and methods combine reusability, type safety and efficiency.
- Usually used with collections and methods that operate on them.
- Several generic collections contained in System.Collections.Generic namespace including lists, dictionaries and queues.
- Benefits:
  - Type safety – no need to use ArrayList any longer
  - Performance – no need to box and unbox objects when iterating collections
  - Better compile-time type checking
- Example List<T>, read as "List of T" where T is the type of object.
- More to come…

Collections

- Closely related data can be handled more efficiently when stored 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.
- Why not use an array? You can, it depends on the use case.
- Collections offer an easy way to add, remove and modify individual elements or a range of elements.
- Capacity and Count properties (not always available).
- Built-in collections include: hash tables, queues, stacks, bags, dictionaries, and lists.

Interfaces

- In C++ you learned about an **abstract class with pure virtual methods**, in C# the counterpart is an interface.
- An interface contains definitions for a group of related functionalities that a class can implement.
- In .NET Framework, interface names start with an uppercase "I".
- You can use an interface as the data type when declaring properties in a class but like C++ you cannot initialize an interface. You must use a concrete type that implements/inherits the interface.
- A class can implement more than one interface.
- Both IEnumerable<T> and ICollection<T> are interfaces.

IEnumerable and ICollection

- A collection can be created and used anywhere in your code.
- A collection can be used as a data type for properties in classes.
- For objects that the users interact with, we can use the data type IEnumerable<T>.
- For objects that are defined by the persistent store model, we can use the data type ICollection<T>.
- The T is a type name placeholder which will be replaced by the actual type of object in the collection. For example, IEnumerable<Person> would contain only Person objects.
- Check out the “Remarks” and “Examples” section of the IEnumerable and ICollection resources above.
- When you add a collection property, you should remember to initialize the property in the constructor.

String ↔ Number Conversions

- The String class is a **reference type** and is **immutable** however the editor, compiler, and runtime make it easy to work with strings.
- Declaring a string is simple and does not require the “new” keyword (unlike the declarations for other reference types):
```
string myName = “John”;
```
- Although a string is immutable after it has been created, the compiler and runtime enable you to change an existing string’s value by using a natural and comfortable syntax. Using the myName instance from above, let’s change it:
```
myName = “John Smith”;
```
- Numbers in the .NET Framework are typically value (primitive) types. Many numeric types are available, but we typically use int and double in C#.

ToString

- In a web app, browser data is transmitted as strings, so it is quite clear why conversion between strings and numbers is so critical.
- Therefore, if you are "round-tripping" numbers, as data, to and from a web page, you must convert them to and from strings.
- You can "convert" a number to a string with the number's "ToString()" method:
  ```
  string foo = 123.ToString();
  ```
- Some number types have ToString() overloads, which permit you to specify a preset (common) or custom number format string.

Convert & TryParse

- If a string contains or holds a numeric value, there is a **System.Convert** class, which has methods that can help with conversion. For example:
  ```
  int bar = Convert.ToInt32("123");
  ```
- This is an unsafe conversion because it's possible that the string cannot be converted. If that's the situation, this statement will cause an exception (a runtime error).
- A better approach is to use the **Int32.TryParse()** method:
  ```
  int foo;
  bool result = int.TryParse("123", out foo);
  ```
- The return result of TryParse() is a bool. If the conversion is successful the result is true and the variable foo holds the converted string-to-number. If conversion failed, false is returned and foo is zero. If conversion fails, an exception is not thrown.
- There is a **TryParse()** method that you can use for most value types, like Double.

Null Strings

- To check if a string is null, then use `== null`.
- To check if a string is empty, then use `.Length == 0`.
- To check if a string is null or empty, then use `string.IsNullOrEmpty()`.
- To check if a string is null, empty, or contains white-space, then use `string.IsNullOrWhiteSpace()`.