# Lesson 28 - More on OOP I

## Nested Classes

C# supports nested classes: a class that is a member of another class.

```
public class Car
{
    string name;
    public Car(string nm)
    {
        name = nm;
        Motor m = new Motor();
    }
    public class Motor
    {
        // some code
    }
}
```

The `Motor` class is nested in the `Car` class and can be used similar to other members of the class. A nested class acts as a member of the class, so it can have the same access modifiers as other members (public, private, protected).

> Just as in real life, objects can contain other objects. For example, a car, which has its own attributes (color, brand, etc.) contains a motor, which as a separate object, has its own attributes (volume, horsepower, etc.). Here, the Car class can have a nested Motor class as one of its members.

## Namespaces

Whenever we make a program, (lets not consider the new top-level-statement .Net 6 syntax), the program has the following structure:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ObjectOrientedProgramming
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Note, that our whole program is inside a `namespace`. So, what are namespaces?

**Namespaces** declare a scope that contains a set of related objects. You can use a namespace to organize code elements. You can define your own namespaces and use them in your program. The `using` keyword states that the program is using a given namespace. For example, we are using the System namespace in our programs, which is where the class Console is defined:

```
using System;

Console.WriteLine("Hello World!");
```

Without the `using` statement, we would have to specify the namespace wherever it is used:

```
System.Console.WriteLine("Hello World!");
```

> The .NET Framework uses namespaces to organize its many classes. `System` is one example of a .NET Framework namespace. Declaring your own namespaces can help you group your class and method names in larger programming projects.

## Structs

A `struct` type is a value type that is typically used to encapsulate small groups of related variables, such as the coordinates of a rectangle or the characteristics of an item in an inventory. The following example shows a simple struct declaration:

```
struct Book
{
    public string title;
    public double price;
    public string author;
}
```

Structs share most of the same syntax as classes, but are more limited than classes. Unlike classes, structs can be instantiated without using a new operator.

```
Book b;
b.title = "Test";
b.price = 5.99;
b.author = "David";

Console.WriteLine(b.title);
```

> Structs do not support inheritance and cannot contain virtual methods.

Structs can contain methods, properties, indexers, and so on. Structs cannot contain default constructors (a constructor without parameters), but they can have constructors that take parameters. In that case the `new` keyword is used to instantiate a struct object, similar to class objects.

```
struct Point
{
    public int x;
    public int y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

```
Point p = new(10, 15);
Console.WriteLine(p.x);
```

## Structs vs Classes

In general, classes are used to model more complex behavior, or data, that is intended to be modified after a class object is created. Structs are best suited for small data structures that contain primarily data that is not intended to be modified after the struct is created. Consider defining a struct instead of a class if you are trying to represent a simple set of data.

> All standard C# types (int, double, bool, char, etc.) are actually structs.

```
struct Cuboid
{
    public int length;
    public int width;
    public int height;

    public Cuboid(int l, int w, int h)
    {
```

```
        length = l;
        width = w;
        height = h;
    }

    public int Volume()
        => length * width * height;

    public int Perimeter()
        => 4 * (length + width + height);
}
```