L3-4: Classes

SESSION	Sessions 3-4
CLASS MODE	LECTURE PRACTICAL
SLIDES	
Completed	☐
Date	@October 25, 2022

Scope

💡 Defines the **visibility** of a variable, method or object/class.

- Visibility means accessing, using, changing, etc. During runtime (when the program runs)
- Language dependent

Block level scope

```
internal class Program
{
    private static void Main(string[] args)
    {
        for (int i = 0; i < 10; i++){
            Console.WriteLine(i)
            int k = i*2
        }
    }
}
```

> Whatever you declare inside a block (e.g. inside a for loop) it will ‘die’ the moment the block is closed. In this previous example i and k are declared inside the for loop and die when it ends.

```
internal class Program
{
    private static void Main(string[] args)
    {
        {
            int k=1;
        }
    }
}
```

> You can use { } to define a block as well. In this case, you can’t access the variable k outside the block { }.

Nesting blocks

```
internal class Program
{
    private static void Main(string[] args)
    {
        for(int i=0; i<10;i++){
            int k = i*2;
            for (int j=100; j> 10; j--){
                Console.WriteLine(x);
            }
            //The variable x won't be visible here (outside the loop)
        }
    }
}
```

Method level

```
internal class Program
{
    private static void Main(string[] args)
    {
        string message = "Hello World";
        int i = 10;
    }

    private static void AnotherMethod()
    {
        Console.WriteLine(message) //will produce error
        int i = 11;
    }
}
```

Can we access the variable “message” in the AnotherMethod method?

I think not, but **ASK CARLOS AGAIN!!!** I tried it in VisualStudioCode and looks like it doesn't even succeed in building the program, compiling i guess.

Class level

We'll see classes more in depth now…

Class

💡 **Class** is a template definition for variables and methods (and therefore also data structure)

💡 **Object** is an instance of a class and contains data

How we declare a class:

```
internal class MathLecture
{
    public int numberOne;
    private int numberTwo;
    public int GetNumber(){
        int j = 19;
        return j *numberOne;
    }
}
```

How we use a class:

💡 Here, it is important how we declare a class. We must highlight that it is the same as calling a reference type, these are arrays objects… Let me elaborate. when you declare an array for example, you first type the **name of the datatype (int[ ])**, then you give it a **name** (my array, for example), then the **= sign**, then **new** (to indicate that it is a new reference type variable we are declaring, then the computer will allocate memory for this specific type) but we have to specify how much memory it has to allocate, that is why we lastly put int[10]. Here we are telling the computer we want a new integer array of size 10.

→ The same goes for classes, first the name of the datatype (in this case it is a class that we have created, so we decide the name of the data type in some sense) **MathLecture**, then the name we want to give it inside our Main Code, **lecture, = sign,** then **new** and lastly the size we want to give it **MathLecture().** It might seem confusing because we are not setting a size, but that depends on the class we created.

```
internal class Program
{
    private static void Main(string[] args){
        MathLecture lecture = new MathLecture();
        lecture.numberOne = 10;
```

```
        //lecture.numberTwo = 12; #Here we can't access numberTwo, it is a private variable in MathLecture
        Console.WriteLine(lecture.GetNumber());
    }
}
```

▼ Creating our first class

```
// We create a class for storing data of a car
internal class Car
{
    public string Brand;
    public string NumberPlate;
    public int YearBuilt;
    public string Location;
    public string SetLocation(string newLocation){
        Location = newLocation;
        return Location;
    }
    public void WriteConsole(){
        Console.WriteLine("The brand of the car is: " + Brand);
        Console.WriteLine("The number plate of the car is: " + NumberPlate);
        Console.WriteLine("The location of the car is: " + Location);
    }
}
```

```
  internal class Program
{
    private static void Main(string[] args){
        Car mycar = new Car();
        mycar.Brand = "Prosche";
        mycar.NumberPlate = "1834DBC";
        mycar.Location = "Spain";
        mycar.YearBuilt = 2017;

        mycar.WriteConsole();

        mycar.SetLocation("Madrid");

        mycar.WriteConsole();
    }
}
```

Object

Value Type

Contains an instance of the type

- When using in methods, assignment etc, the value is copied (remember, int assignments)
- Built-in:
  - Integer, float

- bool
- char

Reference Type

Contains reference to an instance of the type.

- When using in methods, assignment etc. the reference us copied (remember, array “copy”)
- e.g. array, objects created from classes.

It is sooooo hard to explain the difference between reference and value. The definitions he gives are pretty shitty.
→ The value type, stores a value itself. Be it 1, 2, 35, ”, ‘c’, true. These types include integers (int), characters or booleans.
→ The reference type, on the other side also stores a value, but this value is a pointer, you can see pointers as the address of the location in the memory of certain value.

For example, if I create a REFERENCE TYPE called M**yHouse**, that variable stores the address of my house (that is the pointer) but if you call MyHouse in the code, internally, the computer goes to that specific address, but in that address, we have everything in my house, sofas, me, my dad, the kitchen table, a bed, toilet…

NOW, for example, an array is a REFERENCE TYPE, **int[ ]{1,2,3,35,9,2345}**, but the value stored in that specific variable is the pointer!! the address!!! in the world of computers, memory addresses are something like: **0x23f874ab47.** So when I call my array, the computer goes to the address **0x23f874ab47** of the memory, but in that address we have the array [**1,2,3,35,9,2345].**