C# programming language. The beginning

Course

Programming Languages

Semester 2, FIIT

Mayer S.F.

Mikhalkovich S.S

LECTURE # 1

- Lecture 1. A program structure and main constructions

A History

- The C# programming language was born in 1999 at Microsoft. The original name was COOL (C-like Object-Oriented Language)
- It was created by Anders Hejlsberg
- Strongly-typed
- Object-oriented

Microsoft programming is based on a number of different languages

- C#
- VB
- C++
- F#
- XAML
- HTML
- JavaScript

- Visual Studio is the IDE used for Microsoft Development Environment
  - Integrated Development Environment
- Visual Studio is used to build various types of applications
  - .NET
  - Cloud
  - Desktop
  - Phone
  - Tablet & PC
  - Web

  - Documentation MSDN: https://docs.microsoft.com/en-us/dotnet/csharp/

Visual Studio

- To reposition various windows
- To pin or unpin
- To pull down
- We can dock it whenever we want
- We can grab windows and drag them, drop them
- Left click
- Window – reset window layout
- Tools - options

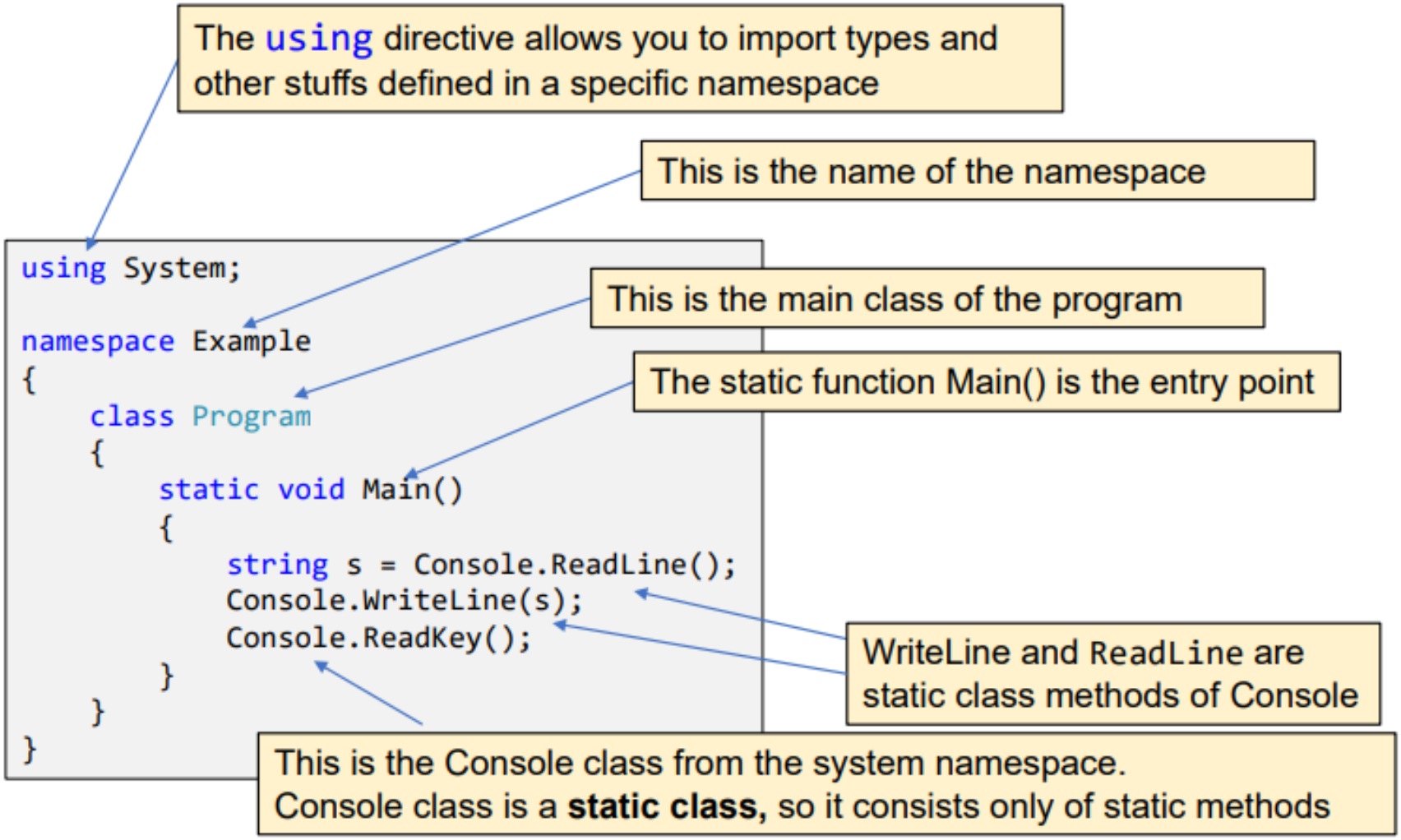
A program structure and main constructions

Lection # 1

Hierarchy

- Namespace
  - Classes
    - Methods

The example of a simple C# program



Without using directive

We must specialize namespace explicitly:

```
namespace ConsoleApp5
{
    class Program
    {
        static void Main()
        {
            var a = int.Parse(System.Console.ReadLine());
            var b = int.Parse(System.Console.ReadLine());
            System.Console.WriteLine(
              $"{System.Math.Sqrt(a * b)}"
            );
        }
    }
}
```

WriteLine and `ReadLine` – static class methods of Console. Console class consists of only from static methods and is a **static class**

using static directive

We can call static methods of a static class without explicit classname if we statically import these functions by directive `using static` <full classname>:

```
using static System.Console;
using static System.Math;

namespace ConsoleApp5
{
    class Program
    {
        static void Main()
        {
            var a = int.Parse(ReadLine());
            var b = int.Parse(ReadLine());
            WriteLine($"{Sqrt(a * b)}");
        }
    }
}
```

Math.Sqrt

The example of a simple C# program Using a static directive

You can call static methods of a static class without an explicit classname if you statically import these functions by directive

```
using static <full classname>;
```

```
using System;

namespace Example
{
    class Program
    {
        static void Main()
        {
            string s = Console.ReadLine();
            Console.WriteLine(s);
            Console.ReadKey();
        }
    }
}
```

```
using static System.Console;

namespace Example
{
    class Program
    {
        static void Main()
        {
            string s = ReadLine();
            WriteLine(s);
            ReadKey();
        }
    }
}
```

Standard namespaces

```
System
System.Collections.Generic (Collection classes)
System.Linq (extension methods for sequences)
System.IO
System.Text
System.Numerics (BigInteger)
System.Reflection
System.Diagnostics.Debug (Assert)
System.Text.RegularExpressions
```

Collection classes

There are the following classes in System.Collections.Generic:

```
List<T>
HashSet<T>
SortedSet<T>
Dictionary<Key,Value>
SortedDictionary<Key,Value>
```

Uninitialized local variables

Uninitialized local variable in C# - is an error!

```
using System;

class Program
{
    static void Main()
    {
        int a;
        Console.WriteLine(a);
    }
}
```

Error CS0165

Standard types and type casts

Standard types

```
int
double

char
string

void
byte
bool    (true,false)
```

Implicit Casting and Explicit Casting

Implicit Casting done automatically when passing a smaller size type to a larger size type.

```
char -> int -> long -> float -> double

int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double
```

Explicit Casting must be done manually by placing the type in parentheses in front of the value.

```
double -> float -> long -> int -> char

double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int
```

sample:

```
double d = 2.6; int i = (int)d; // 2
char c = (char)i;
int j = c; //    implicitly
string s = 'c'; // error! implicitly is not allowed!
// it is allowed s = "" + ‘c’
```

Type Conversion Methods

It is also possible to convert data types explicitly by using built-in methods:

- Convert.ToBoolean,
- Convert.ToDouble,
- Convert.ToString,
- Convert.ToInt32 (int),
- Convert.ToInt64 (long).

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

var a = Convert.ToString(myInt);    // convert int to string a = "10"
var b = Convert.ToDouble(myInt);   // convert int to double  b = 10.0
var c = Convert.ToInt32(myDouble); // convert double to int  c = 5
var d = Convert.ToString(myBool);  // convert bool to string d = "True"
```

Tuples

Tuples have a type System.ValueTuple<…>

```
(int i, int j) = (2, 5);
(i, j) = (j, i); // (5, 2)

(int, int) t = (2, 3);
Console.WriteLine($"{t.Item1} {t.Item2}");

(int x, int y) pt = (2, 3); // Tuples with named fields
Console.WriteLine($"{pt.x} {pt.y}");
```

Basic operations

```
* / % + - =
7 % 3          // 7 mod 3
7 / 3          // 7 div 3
x = y = z;
```

Arithmetic operators are left-associative operators. They are evaluated in order from left to right. For example, a + b - c is evaluated as (a + b) - c.

The assignment operator is right-associative operator. For example, x = y = z is evaluated as x = (y = z).

```
var n = 1/2;           // int 0
var x = (double)1/2; //  double 0.5
var x = 1.0/2;         //  double 0.5
var x = 1/2.0;         //  double 0.5
var x = 1.0/2.0;       //  double 0.5

> < >= <= == !=
```

For the ==, <, >, <=, and >= operators, if any of the operands is not a number (Double.NaN), the result of operation is false. That means that the NaN value is neither greater than, less than, nor equal to any other double (or float) value, including NaN.

```
string s1, s2;
s1.CompareTo(s2)<0 (==0, >0) // s1<s2 is an error!
```

Basic operations

```
i<<3  // i shl 3
i>>2  // i shr 2

& | ^ ~              // and, or, xor, not

b = a++;             // t = a; a++; b = t;
b = --a;             // a--; b = a;

&& || ^ !            // and, or, xor, not
(i<0 || i==2)        // или
(i>=2 && i!=3)       // и
!(i>2)               // не

min = a < b? a : b;

if (a < b) min = a; else min = b;
```

These are operations with bytes for operands of the integral numeric types.

These are conditional logical operators for logical operands.

The ternary conditional operator

Standard mathematical functions

```
using System;
Math.Abs(x)
Math.Sqrt(x)
Math.Pow(a, b)
Math.Min(a, b)
Math.Max(a, b)
Math.Sin(x)
Math.Cos(x)
Math.Exp(x)
Math.Log(x)
```

```
using System;
Math.Round(2.6)=3.0
Math.Round(2.5)=2.0
Math.Round(3.5)=4.0
Math.Floor(2.6)=2.0
Math.Ceiling(2.4)=3.0
```

Random number generation

```
Random r = new Random();
var i = r.Next(2, 5);
var r = r.NextDouble();
```

User Input

```
The Console.ReadLine() method returns a string.

So you must convert any type explicitly,
by using one of the Convert.To methods.
```

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

Lecture task

Lesson #1, task 2

All the rest of Lesson #1 students have to do themselves

https://labs-org.ru/c-sharp1-eng/