C# programming language
The beginning

FIIT, Semester 2

Programming Languages

Mikhalkovich S.S.
Abramyan A.V.

The example of a simple C# program

The `using` directive allows you to import types and other stuffs defined in a specific namespace

This is the name of the namespace

```
using System;

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

This is the main class of the program

The static function Main() is the entry point

WriteLine and `ReadLine` are static class methods of Console

This is the Console class from the system namespace.
Console class is a **static class,** so it consists only of static methods

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();
        }
    }
}
```

Nested namespaces

Namespaces can be nested.

```
namespace My
{
  class MyClass
  {
    void M()
    {
      var mc0 = new MyClass();
      var mc1 = new Your.Company.Project.MyClass();
    }
  }
}

namespace Your
{
  namespace Company
  {
    namespace Project
    {
      public class MyClass { }
    }
  }
}
```

Nested namespaces

Namespaces can be nested. You can define nested namespaces using dot notation:

```
namespace My
{
  class MyClass
  {
    void M()
    {
      var mc0 = new MyClass();
      var mc1 = new Your.Company.Project.MyClass();
    }
  }
}

namespace Your
{
  namespace Company
  {
    namespace Project
    {
      public class MyClass { }
    }
  }
}
```

```
namespace My
{
  class MyClass
  {
    void M()
    {
      var mc0 = new MyClass();
      var mc1 = new Your.Company.Project.MyClass();
    }
  }
}

namespace Your.Company.Project
{
    public class MyClass { }
}
```

Creating an alias with the "using" directive

The `using` directive can be used to define type aliases.

```
namespace My
{
    using YourProject = Your.Company.Project; // namespace alias
    using YourClass = Your.Company.Project.MyClass; // type alias
    class MyClass
    {
        void M()
        {
            var mc1 = new YourProject.MyClass();
            var mc2 = new YourClass();
        }
    }
}

namespace Your.Company.Project
{
    public class MyClass { }
}
```

Reopening of a namespace

Namespace can be reopened, even System Namespace!

```
namespace System
{
    class MyClass
    {
        void M()
        {
        }
    }
}
```

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)
```

Type casts

```
double d = 2.6;
int i = (int)d; // 2

char c = (char)i;
int j = c; // OK

string s = 'c’;       // Error!
string s = "" + 'c'; // OK
```

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!
```