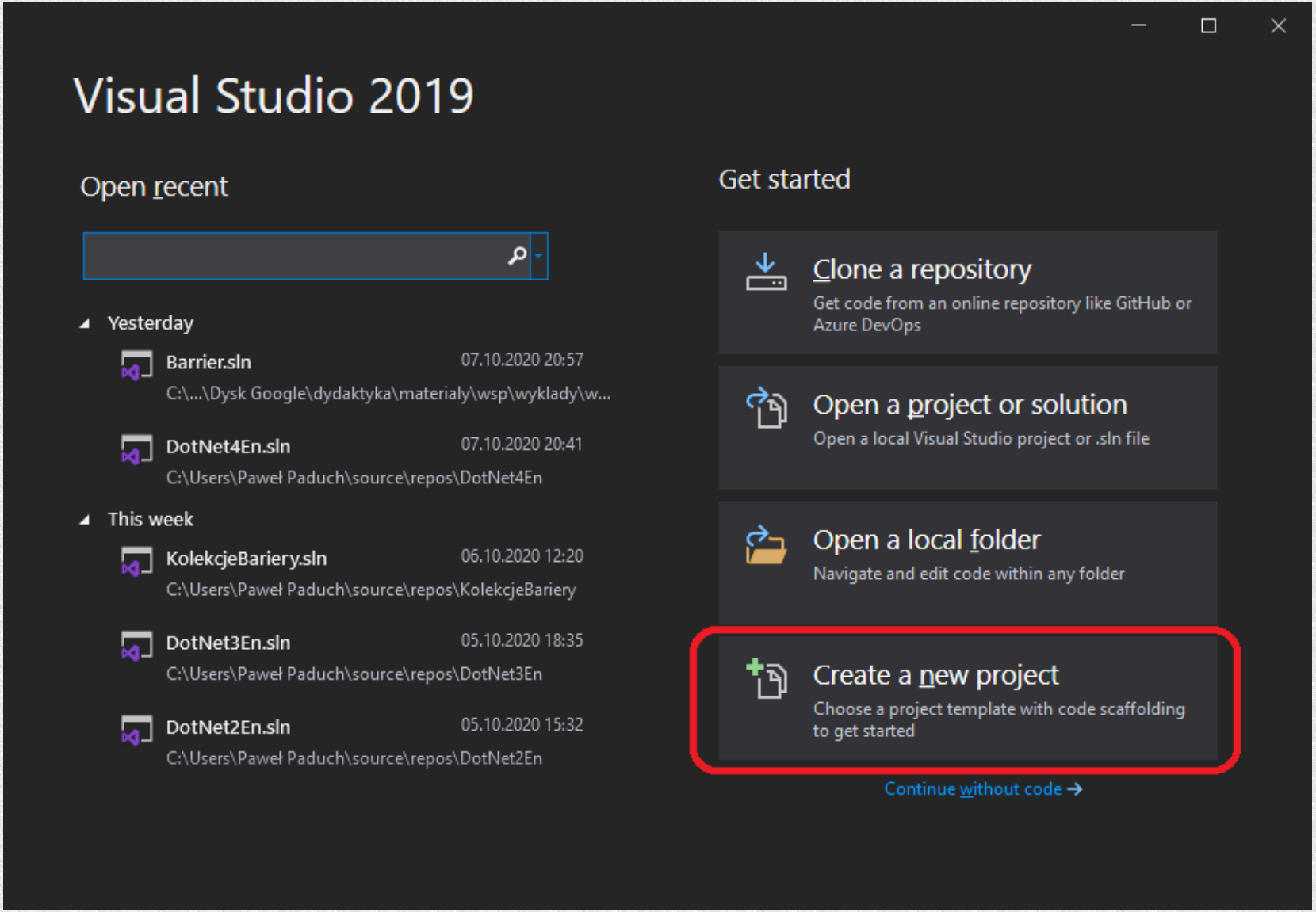
Concurrent Programming

C#

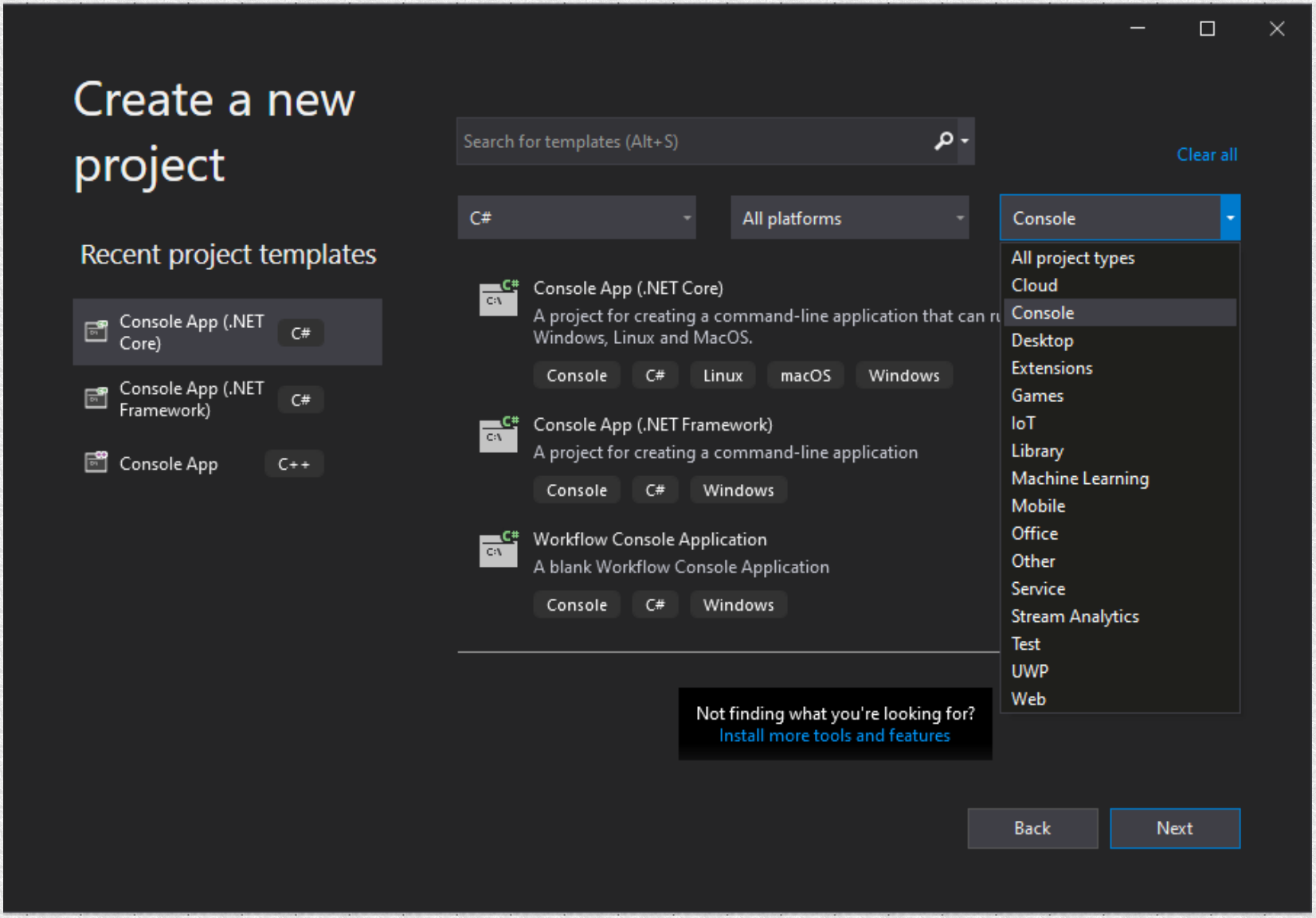
Bibliography

- https://www.microsoft.com/en-us/download/developer-tools.aspx
- http://www.albahari.com/threading/
- https://docs.microsoft.com/dotnet/standard/async
- https://docs.microsoft.com/en-us/dotnet/csharp/
- Programowanie równoległe i asynchroniczne w C# - Warczak,Matulewski, Pałaszek, Sybilski, Borycki, Dziubak. (helion 2014)
- Pro C# 8 with .NET Core 3 : Foundational Principles and Practices in Programming - Andrew Troelsen, Phil Japikse

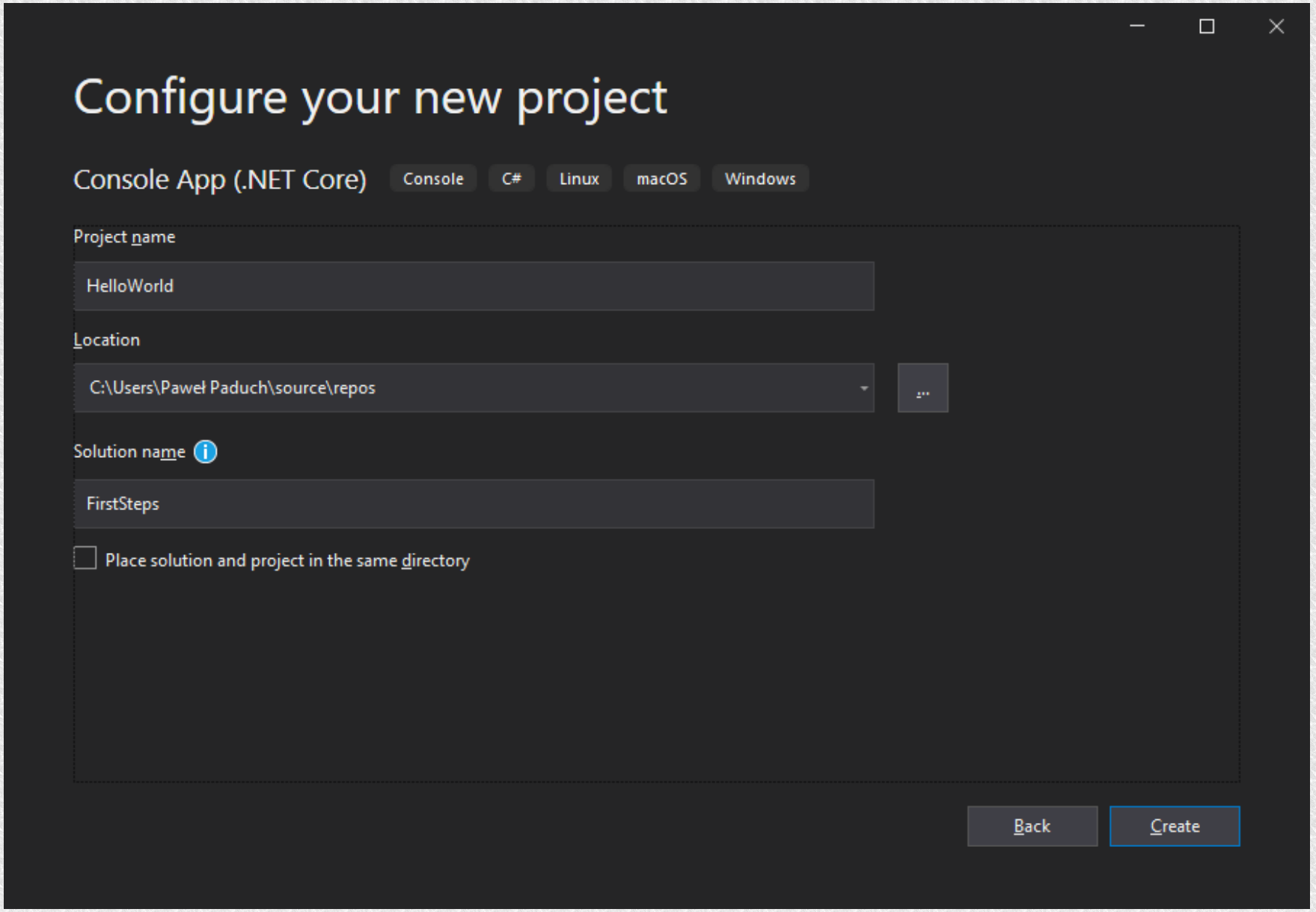
First steps


Visual Studio 2019
Open recent
Yesterday
Barrier.sln
07.10.2020 20:57
C:\...\Dysk Google\dydaktyka\materialy\wsp\wyklady\w...
DotNet4En.sln
07.10.2020 20:41
C:\Users\Paweł Paduch\source\repos\DotNet4En
This week
KolekcjeBariery.sln
06.10.2020 12:20
C:\Users\Paweł Paduch\source\repos\KolekcjeBariery
DotNet3En.sln
05.10.2020 18:35
C:\Users\Paweł Paduch\source\repos\DotNet3En
DotNet2En.sln
05.10.2020 15:32
C:\Users\Paweł Paduch\source\repos\DotNet2En
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

First steps

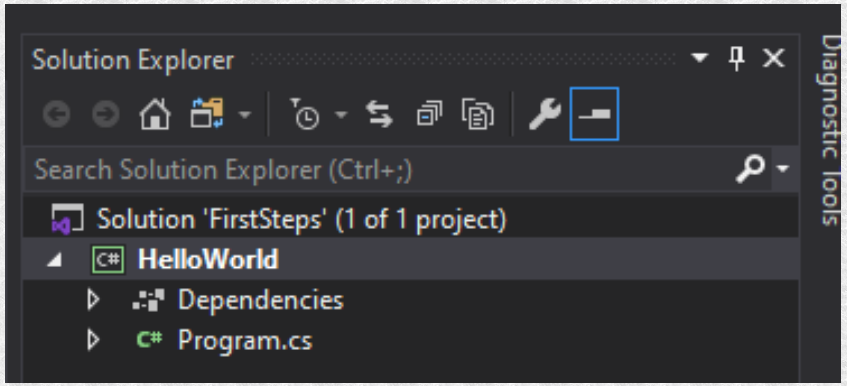

Create a new project
Recent project templates
Console App (.NET Core)
C#
Console App (.NET Framework)
C#
Console App
C++
Search for templates (Alt+S)
Clear all
C#
All platforms
Console
All project types
Cloud
Console
Desktop
Extensions
Games
IoT
Library
Machine Learning
Mobile
Office
Other
Service
Stream Analytics
Test
UWP
Web
Console App (.NET Core)
A project for creating a command-line application that can ru
Windows, Linux and MacOS.
Console
C#
Linux
macOS
Windows
Console App (.NET Framework)
A project for creating a command-line application
Console
C#
Windows
Workflow Console Application
A blank Workflow Console Application
Console
C#
Windows
Not finding what you're looking for?
Install more tools and features
Back
Next

First steps


Configure your new project
Console App (.NET Core)
Console
C#
Linux
macOS
Windows
Project name
HelloWorld
Location
C:\Users\Paweł Paduch\source\repos
...
Solution name
FirstSteps
Place solution and project in the same directory
Back
Create

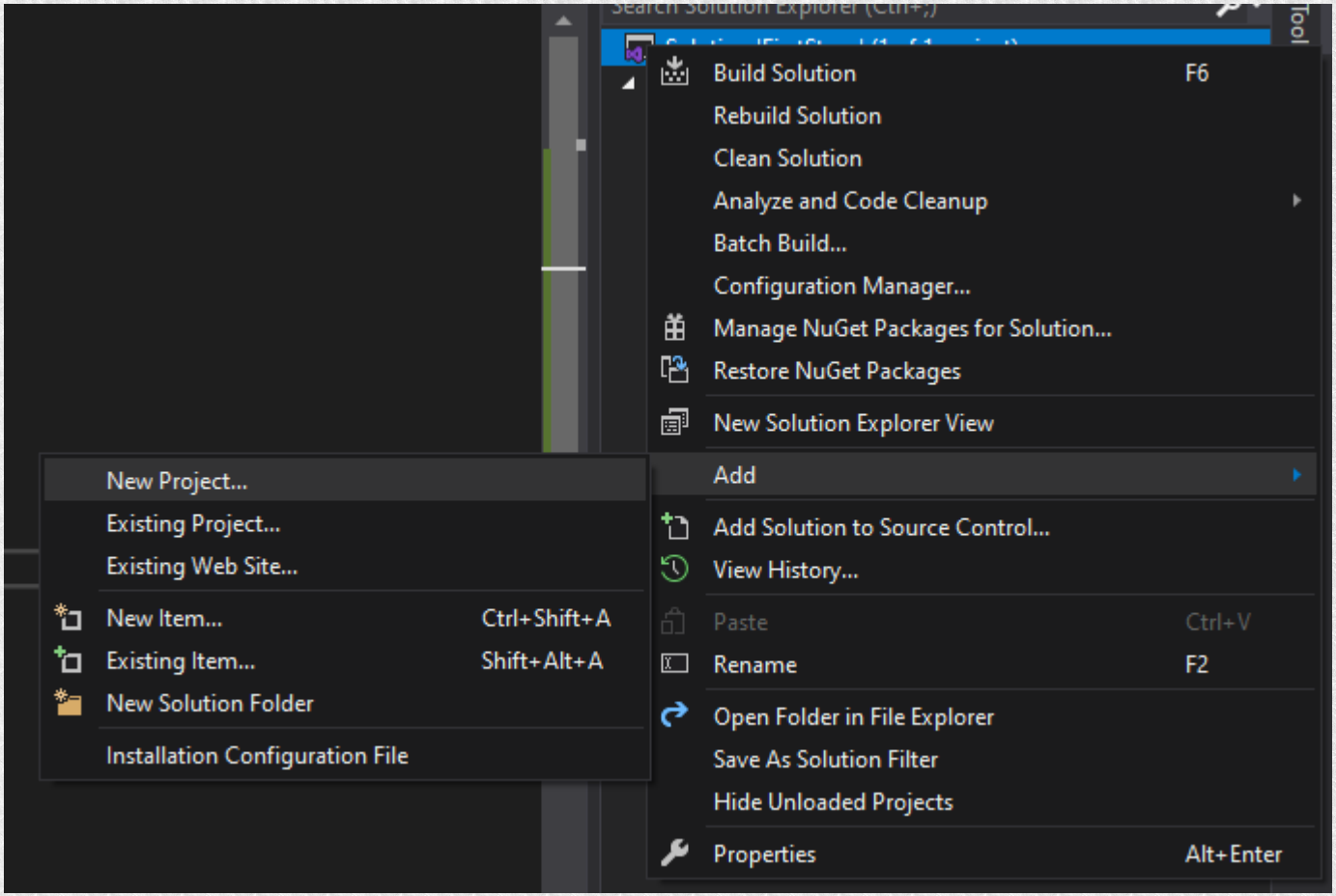
First steps

- **Solution** is main container of the **projects**
- In this example we have solution FirstSteps with one project HelloWorld
- The project contains various elements of the project like: Directories, source files, references, etc.



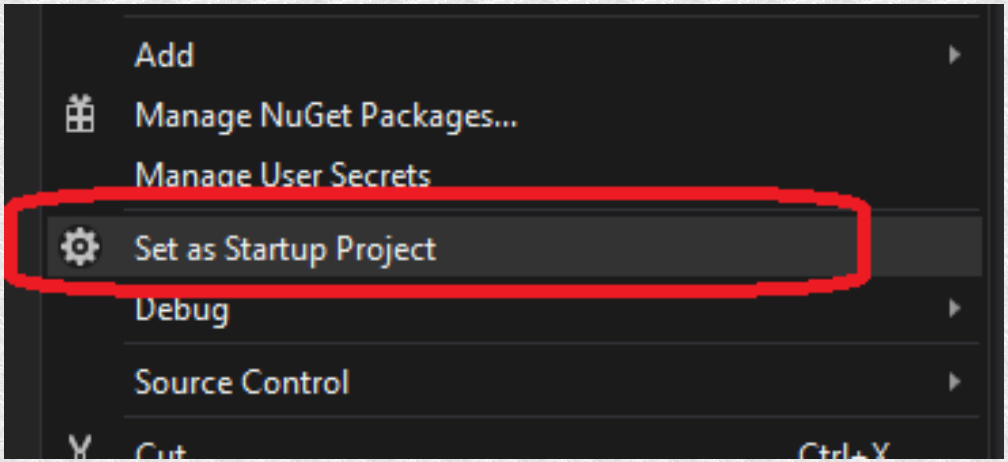
First steps

- Many elements has context menu. For example to add a new project to the solution.

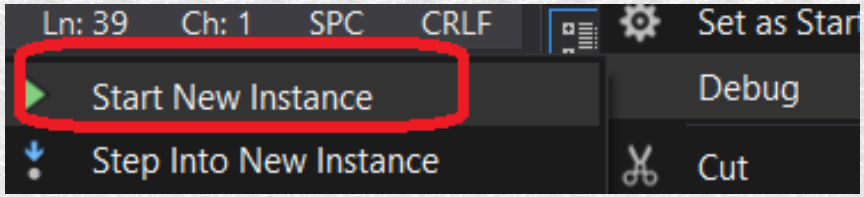


First steps

- To compile and run project just press F5 (in debug mode) or ctrl+F5 in release mode.
- If you have more project you can choose default project to run with context menu of the project.

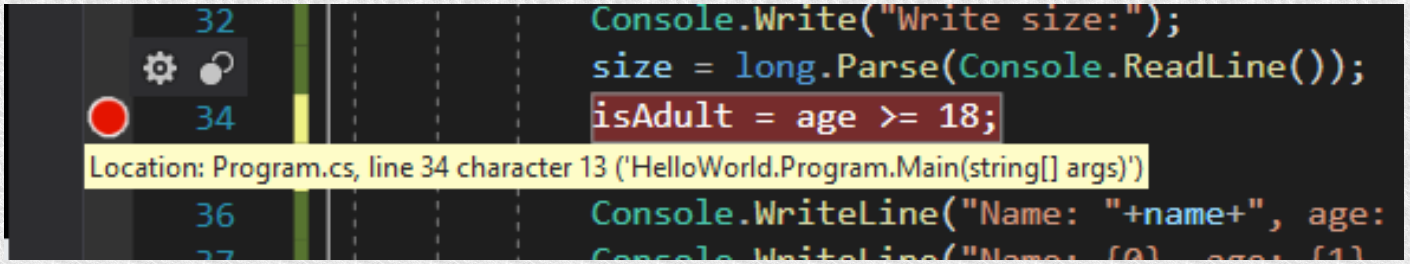


- You can also choose from menu Debug->Start New Instance

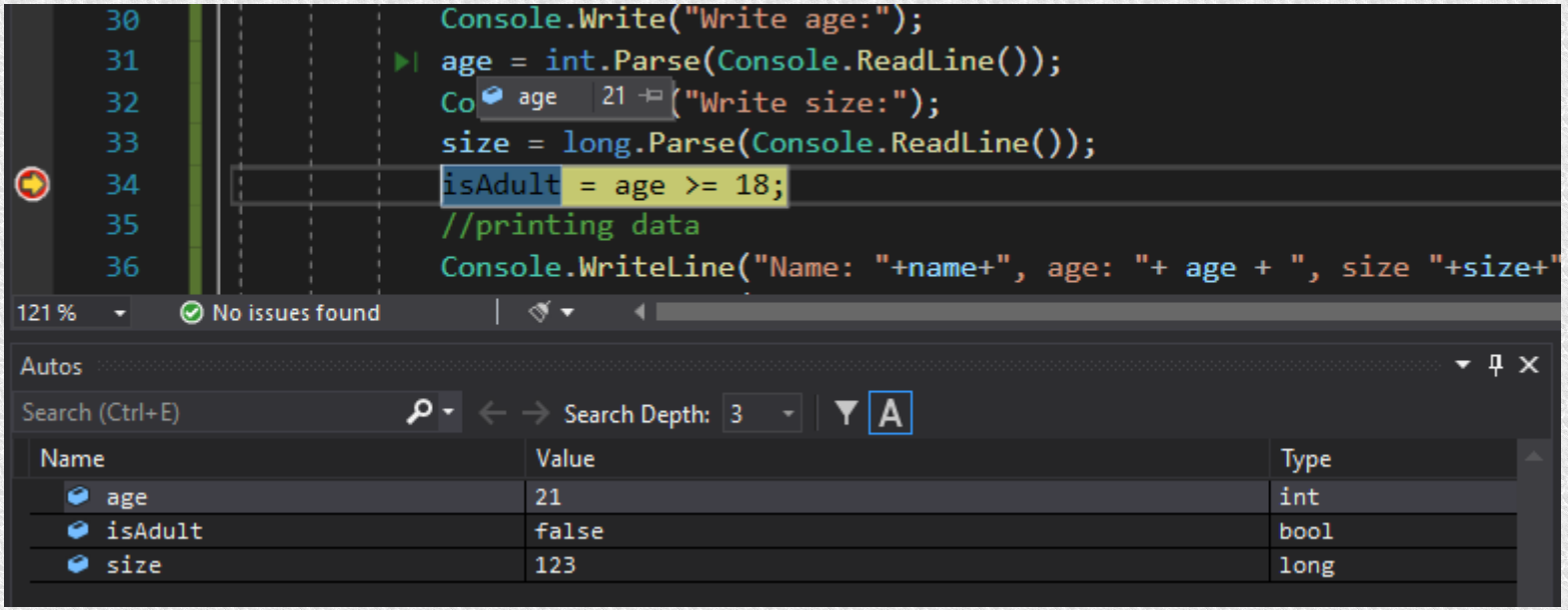


First steps - debug

- In debug mode you can use breakpoints, conditional breakpoints
- You can put it by cliking left margin of code:



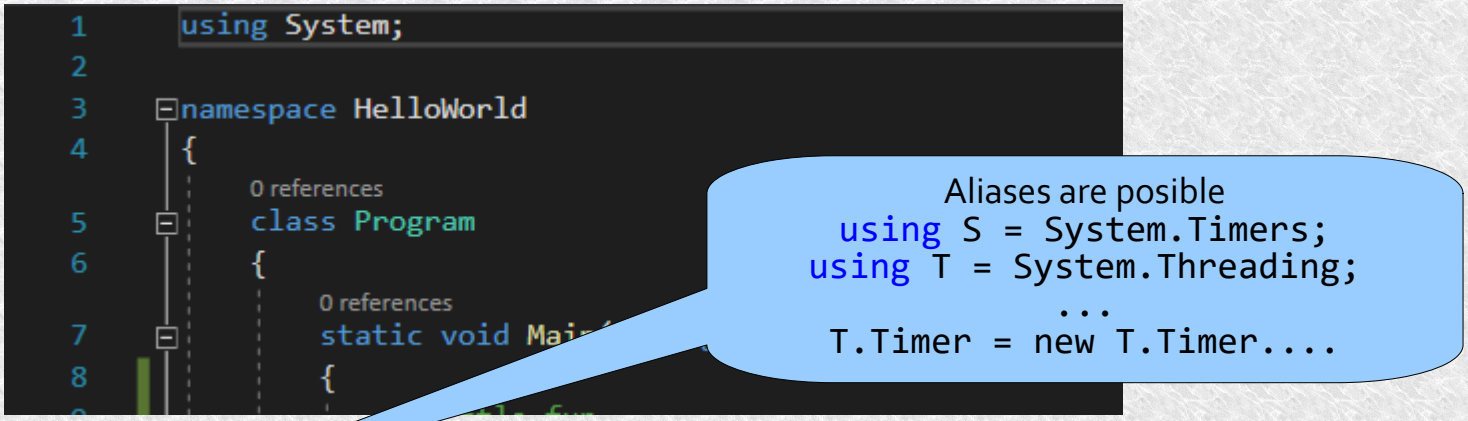
- You can watch variables



- You can run program in steps (F10, F11, shift+F11), and many more

First steps - console

- Console project is basic application where user interface based on inputting and outputting text.



- Code consists **using** part. What namespaces will be used.
- **namespace** block
- classes
- and methods. As in **C** there is one Main method

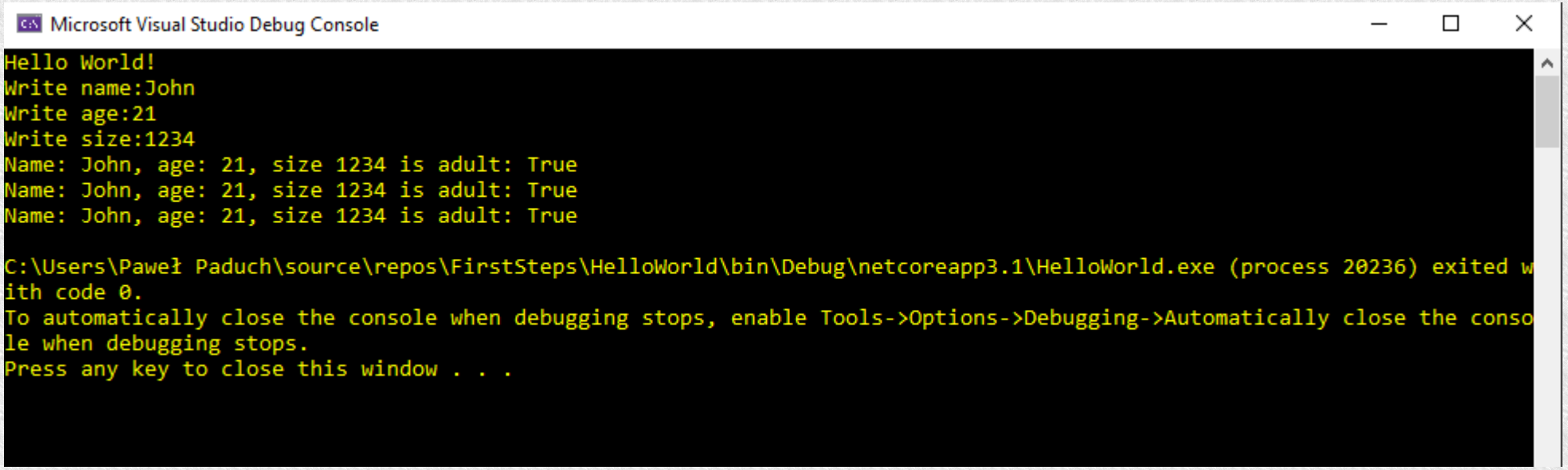
First steps - console

```
//Some music :)
Console.Beep(500, 100);
Console.Beep(1000, 100);
Console.Beep(2000, 100);
Console.Beep(4000, 100);

// Colors
Console.BackgroundColor = ConsoleColor.Black;
Console.ForegroundColor = ConsoleColor.Yellow;

// Size
Console.WindowHeight = 30;
Console.WindowWidth = 120;

//Printing something
Console.WriteLine("Hello World!");
```

First steps - console

```
//declaring variable
string name;
int age;
long size;
bool isAdult;

//getting data from user
Console.Write("Write name:");
name = Console.ReadLine();
Console.Write("Write age:");
age = int.Parse(Console.ReadLine());
Console.Write("Write size:");
size = long.Parse(Console.ReadLine());
isAdult = age >= 18;

//printing data
Console.WriteLine("Name: "+name+", age: "+ age + ", size "+size+" is
adult: "+isAdult);
Console.WriteLine("Name: {0}, age: {1}, size {2} is adult:
{3}",name,age,size,isAdult);
Console.WriteLine($"Name: {name}, age: {age}, size {size} is adult:
{isAdult}");
```

First steps - console



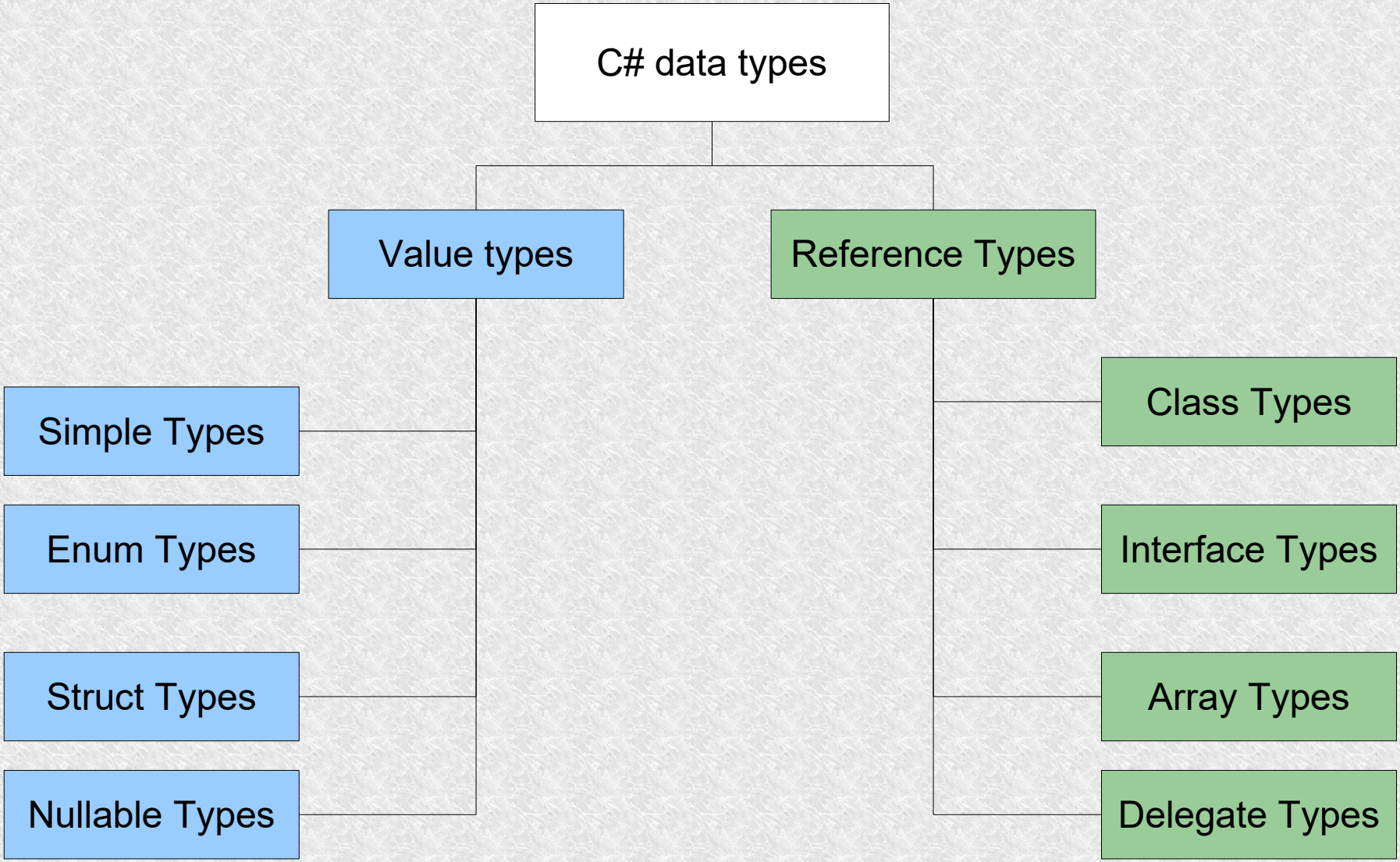
Basics - keywords

abstract as base bool
break byte case catch
char checked class const
continue decimal default delegate
do double else enum
event explicit extern false
finally fixed float for
foreach goto if implicit
in int interface internal
is lock long namespace
new null object operator
out override params private
protected public readonly ref
return sbyte sealed short
sizeof stackalloc static string
struct switch this throw
true try typeof uint
ulong unchecked unsafe ushort
using virtual void volatile
while

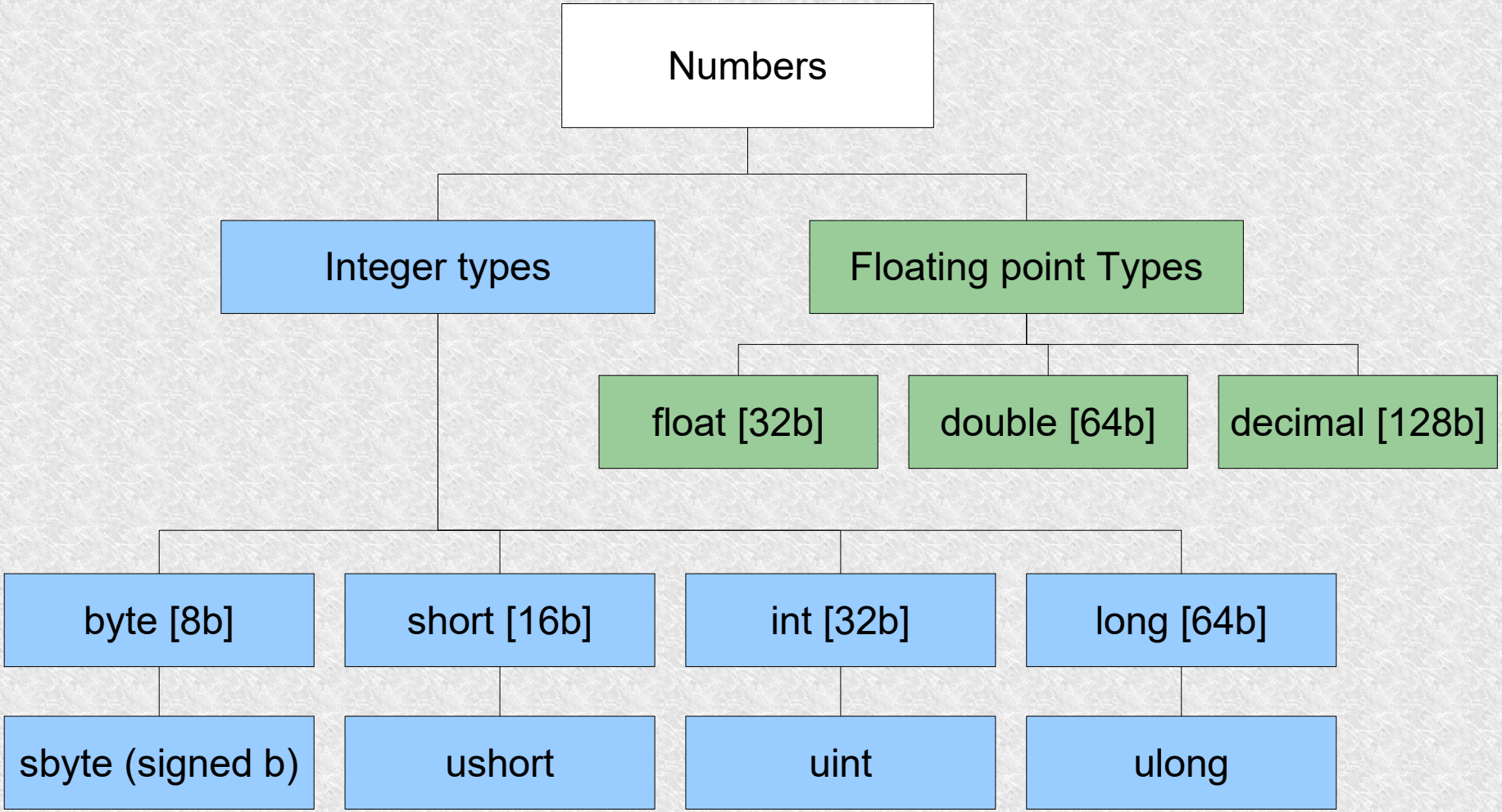
add alias ascending
async await by
descending dynamic equals
from get global
group into join
let nameof notnull
on orderby partial (type)
partial (method) remove select
set unmanaged (generic type constraint)
value
var when (filter condition) where (generic type constraint)
where (query clause) yield

For more information please visit:
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/

First steps - data types



Basics - numbers



Basics - numbers

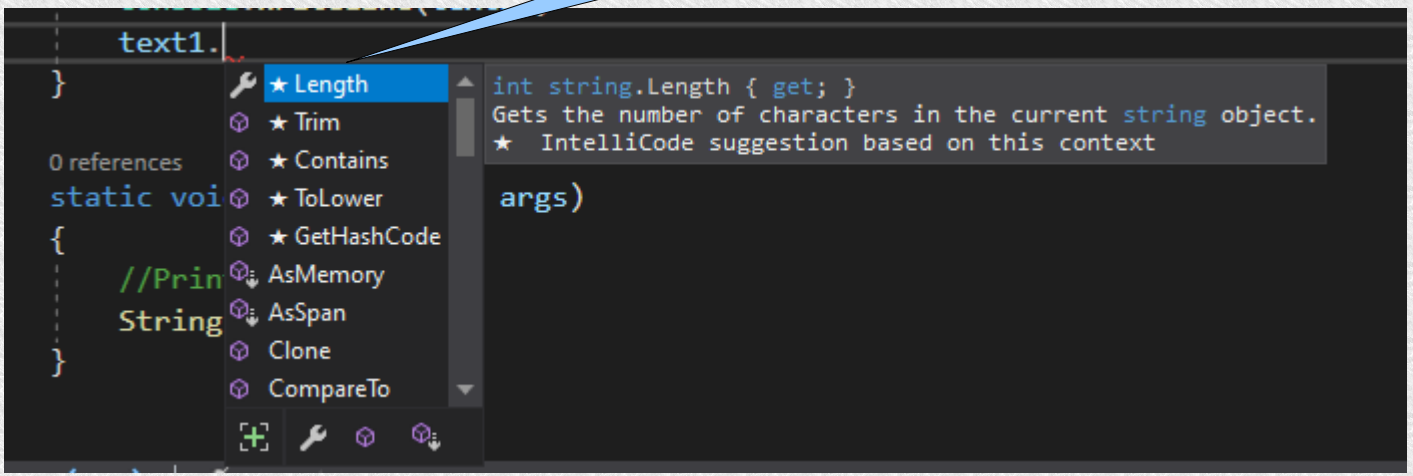
```
Console.WriteLine($"sbyte {sbyte.MinValue}..{sbyte.MaxValue}");
Console.WriteLine($"byte {byte.MinValue}..{byte.MaxValue}");
Console.WriteLine($"short {short.MinValue}..{short.MaxValue}");
Console.WriteLine($"ushort {ushort.MinValue}..{ushort.MaxValue}");
Console.WriteLine($"int {int.MinValue}..{int.MaxValue}");
Console.WriteLine($"uint {uint.MinValue}..{uint.MaxValue}");
Console.WriteLine($"long {long.MinValue}..{long.MaxValue}");
Console.WriteLine($"ulong {ulong.MinValue}..{ulong.MaxValue}");
Console.WriteLine($"float {float.MinValue}..{float.MaxValue}");
Console.WriteLine($"double {double.MinValue}..{double.MaxValue}");
Console.WriteLine($"decimal {decimal.MinValue}..{decimal.MaxValue}");
```

sbyte -128..127
byte 0..255
short -32768..32767
ushort 0..65535
int -2147483648..2147483647
uint 0..4294967295
long -9223372036854775808..9223372036854775807
ulong 0..18446744073709551615
float -3,4028235E+38..3,4028235E+38
double -1,7976931348623157E+308..1,7976931348623157E+308
decimal -79228162514264337593543950335..79228162514264337593543950335

Basics - Strings

```
string c = "S";
string w = "Word";
string t = "This is a sentence.";

//string as array
char[] chars = { 's', 't', 'r', 'i', 'n', 'g' };
string str = new string(chars);
foreach (char chr in str)
    {
    Console.WriteLine(chr);
    }
Console.WriteLine(str);
string text1 = "This is a \"string\"";
string text2 = @"c:\tmp\dir";
string text3 = "c:\\tmp\\dir2";
Console.WriteLine(text1);
Console.WriteLine(text2);
Console.WriteLine(text3);
```

Use intellisense to find methods and properties



Basics - Datetimes

```
//assigns default value 01/01/0001 00:00:00
DateTime dt1 = new DateTime();
//assigns year, month, day
DateTime dt2 = new DateTime(2020, 10, 08);
//assigns year, month, day, hour, min, seconds
DateTime dt3 = new DateTime(2020, 10, 08, 19, 10, 20);
//assigns year, month, day, hour, min, seconds, UTC timezone
DateTime dt4 = new DateTime(2020, 10, 08, 19, 10, 20, DateTimeKind.Utc);

long currentTicks = DateTime.Now.Ticks;
// Ticks are 100 - nanosecond intervals that have elapsed since
// January 1, 0001, at 00:00:00.000 in the Gregorian calendar.
DateTime dt = new DateTime(currentTicks);
long minticks = DateTime.MinValue.Ticks;
long maxticks = DateTime.MaxValue.Ticks;
Console.WriteLine($"current ticks {currentTicks} current Date: {dt} min
ticks: {minticks} max ticks: {maxticks}");

DateTime dt5 = new DateTime(2020, 12, 31);
DateTime dt6 = new DateTime(2020, 12, 30,12,12,12);
TimeSpan ts = new TimeSpan(25, 20, 55);
DateTime newDate = dt5.Add(ts);
Console.WriteLine(newDate);
TimeSpan ts2 = dt5-dt6;
Console.WriteLine(ts2);
```

Basics - Datetimes

```
Console.WriteLine("Date String Current Culture: " + dt.ToString("d"));
Console.WriteLine("MM/dd/yyyy Format: " + dt.ToString("MM/dd/yyyy"));
Console.WriteLine("dddd, dd MMMM yyyy Format: " + dt.ToString("dddd, dd MMMM yyyy"));
Console.WriteLine("MM/dd/yyyy h:mm tt Format: " + dt.ToString("MM/dd/yyyy h:mm tt"));
Console.WriteLine("MMMM dd Format:" + dt.ToString("MMMM dd"));
Console.WriteLine("HH:mm:ss Format: " + dt.ToString("HH:mm:ss"));
Console.WriteLine("hh:mm tt Format: " + dt.ToString("hh:mm tt"));
Console.WriteLine("Short Date String: " + dt.ToShortDateString());
Console.WriteLine("Long Date String: " + dt.ToLongDateString());
Console.WriteLine("Short Time String: " + dt.ToShortTimeString());
Console.WriteLine("Long Time String: " + dt.ToLongTimeString());
```

```
var str = "2020-12-23 14:35:00";
DateTime dts;
var success = DateTime.TryParse(str, out dts);
if (success)
    Console.WriteLine($"Date from string: {dts}");
else
    Console.WriteLine($"{str} is not a valid date string");
```

Basics

Implicitly-Typed Variable

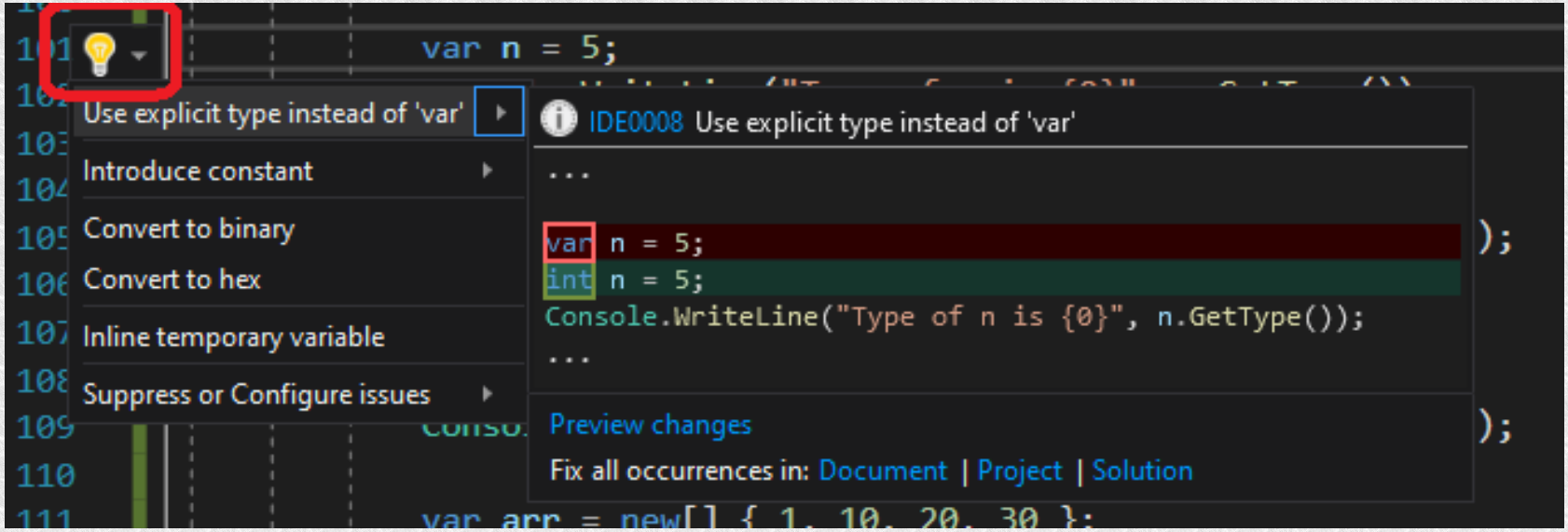
- Explicitly typed variables need specific type name (string, int, short, Object etc.)
- From c# 3.0 we can use Implicitly-Typed Variable. **var**

```
var isOk = true;
Console.WriteLine("Type of isOk is {0}", isOk.GetType());

var str = "String sample";
Console.WriteLine("Type of str is {0}", str.GetType());

var n = 5;
Console.WriteLine("Type of n is {0}", n.GetType());

var dbl = 100.50d;
Console.WriteLine("Type of dbl is {0}", dbl.GetType());


var ano = new { name = "Steve" };
Console.WriteLine("Type of ano is {0}", ano.GetType());

var arr = new[] { 1, 10, 20, 30 };
Console.WriteLine("Type of arr is {0}", arr.GetType());

var file = new FileInfo("MyFile");
Console.WriteLine("Type of file is {0}", file.GetType());
```

Basics

Implicitly-Typed Variable

- Implicitly typed variables can be converted in visual studio



Basics

Dynamic Variable

- C# 4 introduces a new type, **dynamic**
- It is checked in run time not during compilation

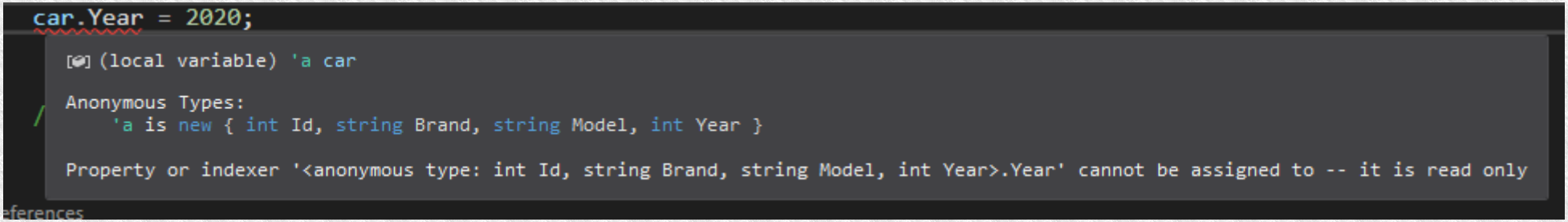
```
dynamic myVariable = 1;
Console.WriteLine($"MyVariable: {myVariable}");

myVariable = "Other type";
Console.WriteLine($"MyVariable: {myVariable}");

myVariable = DateTime.Now;
Console.WriteLine($"MyVariable: {myVariable}");
```

Basics

Anonymous types

```
var car = new { Id = 1, Brand = "Mazda", Model = "6", Year = 2019 };
Console.WriteLine($"car id: { car.Id} Brand: { car.Brand} Model: { car.Model}
Year: { car.Year}");
```

Anonymous types are read only



Basics

Arrays and Lists

```
int[] numbers = new int[5] { 2, 4, 6, 8, 10 };
string[] colors = new string[3] { "red", "green", "blue" };
Console.WriteLine($"numbers[0] = {numbers[0]}");
Console.WriteLine($"numbers[1] = {numbers[1]}");
Console.WriteLine($"colors[0] = {colors[0]}");
Console.WriteLine($"colors[1] = {colors[1]}");
colors[0] = "black";
Console.WriteLine($"colors[0] = {colors[0]}");
List<string> names = new List<string>();
names.Add("John");
names.Add("Tom");
names.Add("Anna");
Console.WriteLine($"{names[0]} {names[1]} {names[2]}");
```

Basics

Multidimensional arrays

```
int[,] array2d; // 2D array
int[,,] array3d; // 3D array
array2d = new int[3, 2]{
                        {1, 2},
                        {3, 4},
                        {5, 6}
};
for (int i = 0; i < 3; i++)
    {
    Console.WriteLine($"{ array2d[i, 0]} { array2d[i, 1]}");
    }
array3d = new int[2, 2, 2]{
                            { {1, 2}, {3, 4} },
                            { {5, 6}, {7, 8} }
};
for (int i = 0; i < 2; i++)
    {
    for (int j = 0; j < 2; j++)
        {
        Console.WriteLine($"{ array3d[i, j, 0]} { array3d[i, j, 1]}");
        }
    }
```

Basics

Jagged arrays

- A jagged array is an array of array.
- Jagged arrays store arrays instead of literal values.
- A jagged array is initialized with two square brackets [][]. The first bracket specifies the size of an array, and the second bracket specifies the dimensions of the array which is going to be stored.

```
int[][] array1 = new int[2][]; //array of two single-dimensional arrays
int[][,] array2 = new int[3][,]; // array of three two-dimensional arrays

array1[0] = new int[4] { 1, 2, 3, 4 };
array1[1] = new int[5] { 5, 6, 7, 8, 9 };
foreach (int[] arr in array1)
    {
    foreach (var item in arr)
        {
        Console.WriteLine( item );
        }
    }
for (int i = 0; i < 2; i++)
    {
    for (int j = 0; j < array1[i].Length; j++)
        {
        Console.WriteLine($"array1[{i}][{j}] = {array1[i][j]}");
        }
    }
```

Basics

Jagged arrays

- Example Array of arrays of arrays.

```
int[][][] arrayOfarraysOfarrays = new int[2][][]
            {
            new int[2][]
                {
                new int[3] { 1, 2, 3},
                new int[2] { 4, 5}
                },
            new int[1][]
                {
                new int[3] { 7, 8, 9}
                }
            };

Console.WriteLine(arrayOfarraysOfarrays[0][0][0]); //1
Console.WriteLine(arrayOfarraysOfarrays[1][0][1]); //8
```

Basics

if-else

```
int x = 5, y = 10;

if (x < y)
{
    Console.WriteLine("x is less than y");
}

if (x > y)
{
    Console.WriteLine("x is greater than y");
}
else
{
    Console.WriteLine("x is less or equal to y");
}
//Ternary Operator ?:
var greater = x > y ? x : y;
Console.WriteLine($"Greater number is {greater}");
```

Basics

switch case

```
Console.Write("Give number from 1 to 3: ");
int i = int.Parse(Console.ReadLine());

switch (i)
    {
    case 1:
       Console.WriteLine($"You have choose one");
       break;
    case 2:
       Console.WriteLine($"You have choose two");
       break;
    case 3:
       Console.WriteLine($"You have choose three");
       break;
    default:
       Console.WriteLine($"You have choose wrong number {i}");
       break;
    }
```

Basics

for foreach

```
string[] colors = new string[3] { "red", "green", "blue" };
List<string> names = new List<string>();
names.Add("John");
names.Add("Tom");
names.Add("Anna");

for (int i = 0; i < colors.Length; i++)
    {
    Console.WriteLine(colors[i]);
    }

foreach (var item in names)
    {
    Console.WriteLine(item);
    }
```

Basics

while loops

```
int x = 0, y = 1;
while (x < 3)
    {
    x++;
    while (y < 5)
       {
       Console.WriteLine($"x = {x}  y = {y}");
       y++;
       }
     y = 1;
     }

int i = 0;
do
    {
    Console.WriteLine(++i);
    }
while (i < 5);
```

Basics

Enum

```
enum Status
    {
        Stopped,
        Running,
        Finished,
        Unknown
    }

enum Level
    {
        VeryLow = 0,
        Low = 1,
        Normal = 2,
        High = 3,
        VeryHigh = 4
    }
```

```
Console.WriteLine("Choose level from 0 to 4");
Level level = (Level)int.Parse(Console.ReadLine());
switch (level)
    {
    case Level.VeryLow:
        Console.WriteLine("You have choosen Very Low level");
        break;
    case Level.Low:
        Console.WriteLine("You have choosen Low level");
        break;
    case Level.Normal:
        Console.WriteLine("You have choosen Normal level");
        break;
    case Level.High:
        Console.WriteLine("You have choosen High level");
        break;
    case Level.VeryHigh:
        Console.WriteLine("You have choosen Very High level");
        break;
    default:
        Console.WriteLine("You have choosen wrong level");
        break;
    }
```

Basics
Structure

- Structure is a value type, scalar type, it is kept on the stack.
- When the class is small, it is better to use a structure.
- Constructor must be with all values
- (cannot be the default or incomplete).
- In C #, simple types are structures.
- Structures do not support inheritance.

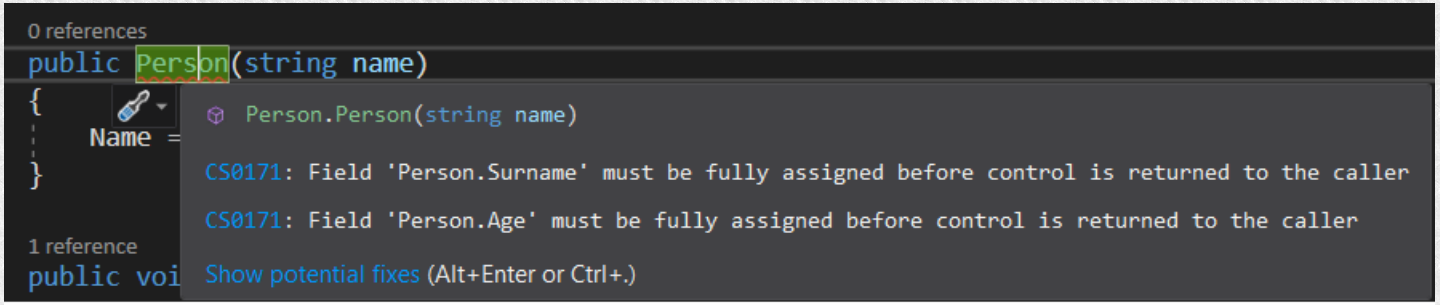
Basics

Structure

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Basic
{
    struct Person
    {
        public string Name;
        public string Surname;
        public int Age;
        public Person(string name,string surname, int age)
        {
            Name = name;
            Surname = surname;
            Age = age;
        }
        public void SetPerson(string name, string surname, int age)
        {
           Name = name;
           Surname = surname;
            Age = age;
        }
    }
}
```

Basics
Structure

- We cannot define a default constructor in the structure
- You can only define a constructor with all fields otherwise we will get an error:



- A possible workaround is to inherit from the default constructor:

```
public Person(string name) : this()
  {
      Name = name;
  }
```

Basics

Structure

```
Person person = new Person();
person.Age = 18;
person.Name = "Johny";
person.Surname = "Walker";
Console.WriteLine($"{person.Name} {person.Surname} {person.Age}");

Person person2 = new Person("Jack", "Daniels", 21);
Console.WriteLine($"{person2.Name} {person2.Surname} {person2.Age}");

Person person3 = new Person();
person3.SetPerson("Captain", "Morgan", 25);
Console.WriteLine($"{person3.Name} {person3.Surname} {person3.Age}");
```

Podstawy
Class

- The class is a reference type
- Her place is on the heap
- When we have large structures it is better to use a class

Basics

Class

```
//access modifier = public
//class name = Vector
public class Vector
    {
    public string Name = "vector"; //field
    //property
    private double x;
    public double X
        {
            get { return x; }
            set { x = value; }
        }
    //auto implemented property
    public double Y { get; set; }
    public double Z { get; set; }
    public Vector() //default constructor
        {
            x = 0.0; Y = 0.0;  Z = 0.0;
        }
    public Vector(double x,double y, double z)
        {
            this.x = x; this.Y = y;  this.Z = z;
        }
    public void Print() //method/function
        {
            Console.WriteLine($"[{x},{Y},{Z}]");
        }
}
```

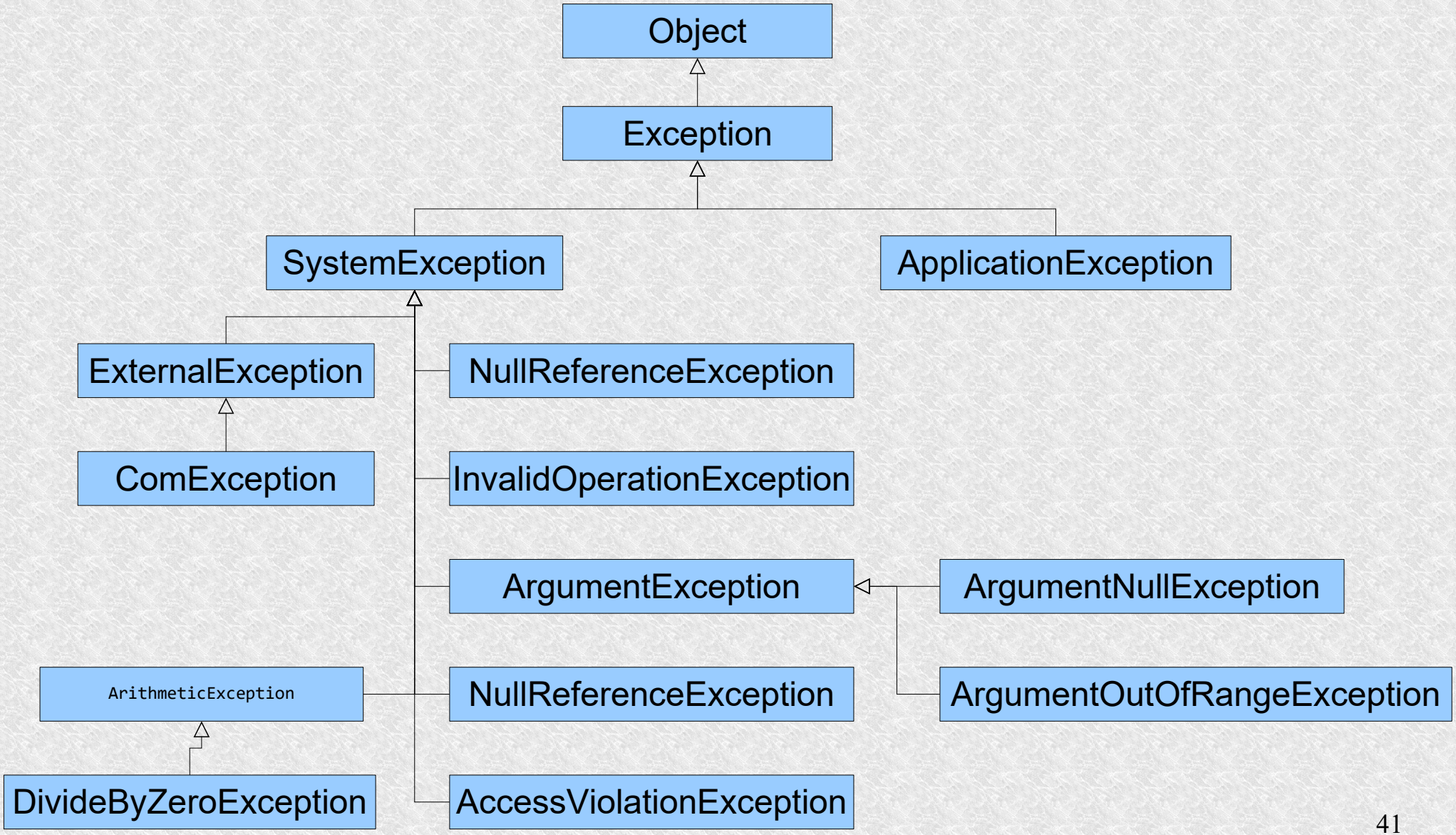
Basics

Class

```
Vector vec1 = new Vector();
vec1.X = 1.0;
vec1.Y = 2.0;
vec1.Z = -2.5;

Vector vec2 = new Vector() { X = -0.5, Y = 0, Z = 3.0 };
Vector vec3 = new Vector(4.2, 2.2, 3.1);
vec1.Print();
vec2.Print();
vec3.Print();
```

Basics

Exceptions



Basics

Exceptions

```
try
    {
    //int[] array = new int[2];
    //array[3] = 11; // IndexOutOfRangeException
    //int a = int.Parse("a"); //FormatException
    //int zero = 0;
    //var b = 2 / zero;
    throw new ApplicationException("OMG! We have exception!!!");
    }
catch (DivideByZeroException zeroEx)
    {
    Console.WriteLine($"type: {zeroEx.GetType()} message: {zeroEx.Message}");
    }
catch (IndexOutOfRangeException iex)
    {
    Console.WriteLine($"type: {iex.GetType()} message: {iex.Message}");
    }
catch (Exception ex)
    {
    Console.WriteLine($"type: {ex.GetType()} message: {ex.Message}");
    }
finally
    {
    Console.WriteLine("Finally block");
    }
```

More examples
Generic Type

- When something (method, interface, class etc.) is generic it means there is no defined exact type.
- Generic type is declared by <T>

```
public class Vector<T>
    {
        public T X { get; set; }
        public T Y { get; set; }
        public T Z { get; set; }

        public Vector()
        { }
        public Vector(T x, T y, T z)
            {
            X=x;
            Y=y;
            Z=z;
            }
        public void Print() //method/function
        {
            Console.WriteLine($"[{X}][{Y}][{Z}]");
        }
    }
```

```
Vector<int> iVector = new
Vector<int>(1,2,3);
Vector<double> dVector = new
Vector<double>(2.4, 1.2, 4.5);
iVector.Print();
dVector.Print();
```

More examples
Delegate

- The delegate is a reference type data type that defines the method signature
- Signature means what type is returned and what parameters are passing in.
- In c it was pointer to the function without typization.

```
delegate void DelExample(string message);

static DelExample del1 = new DelExample(MethodPrint);
// or
static DelExample del2 = MethodPrint;
// or anonymous using lambda expression
static DelExample del3 = (string msg) => Console.WriteLine(msg);

// target method
static void MethodPrint(string message)
    {
     Console.WriteLine(message);
     }

static void MethodPrint2(string message)
    {
     Console.WriteLine("This is the message: " + message);
    }
```

More examples

Delegate

```
static void Main(string[] args)
{
    del1.Invoke("Delegate 1 says - Hello World!");
     del2("Delegate 2 says - Hello World!");
     del3("Delegate 3 says - Hello World!");
     DelExample del4 = MethodPrint2;
     del4("Delegate 4 says - Hello World!");

    //delegates can be combined
    del1 += del2 + del3 + del4;
    del1("All together");
    del1 -= del4;
    del1("Almost all");
}
```

More examples

Built-in generic delegate

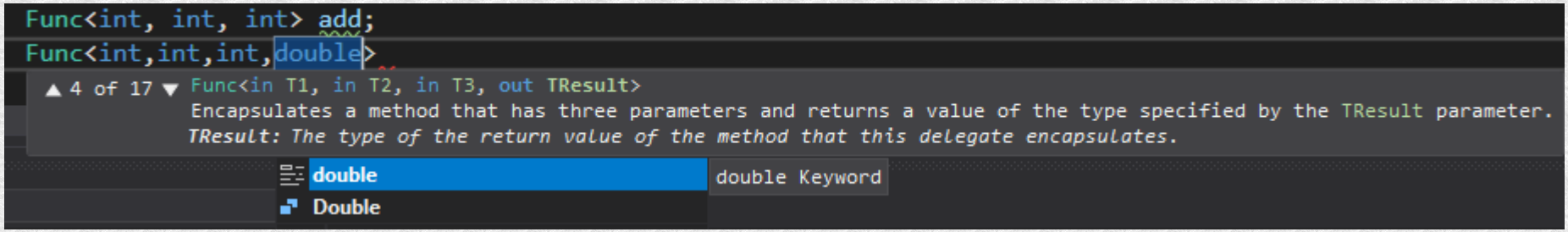
C# includes built-in generic delegate types

- Function
- Action
- Predicate

So there is no need to define custom delegates manually in many cases

- **Function** can have zero or more (up to 16) input parameters and has one return parameter,
  so Func delegate type must return a value
- Func delegate does not allow ref and out parameters
- Func delegate type can be used with an anonymous method or lambda expression
- Signature of function is following:

public delegate Tresult Funct<in T1, in T2,..., out Tresult>(T1 arg1, T2 arg2,...)



More examples

Function

```
static int Lenght(string str)
    {
    return str.Length;
    }

static string Join(string str1, string str2)
    {
    return str1 + str2;
    }


Func<string, int> len = Lenght;
int l = len("Hello World");
Console.WriteLine($"lenght is: {l}");

Func<string, string, string> join = Join;
string strJoined = join("Hello ", "World!");
Console.WriteLine(strJoined);
```

More examples

Action

Action is like a Function delegate but return no parameter

```
static void PrintVertically(Vector<int> v)
      {
          Console.WriteLine($"[{v.X}]");
          Console.WriteLine($"[{v.Y}]");
          Console.WriteLine($"[{v.Z}]");
      }
```

```
Vector<int> iVec = new Vector<int>(1, 2, 3);
Action<Vector<int>> print1 = PrintVertically; //the simplest
Action<Vector<int>> print2 = new Action<Vector<int>>(PrintVertically); //new
Action<Vector<int>> print3 = delegate (Vector<int> v) //anonymous
       {
       Console.WriteLine($"[{v.X}]");
       Console.WriteLine($"[{v.Y}]");
       Console.WriteLine($"[{v.Z}]");
       };
Action<Vector<int>> print4 = v => {  //lambda
       Console.WriteLine($"[{v.X}]");
       Console.WriteLine($"[{v.Y}]");
       Console.WriteLine($"[{v.Z}]");
       };
print1(iVec);
print2(iVec);
print3(iVec);
print4(iVec);
```

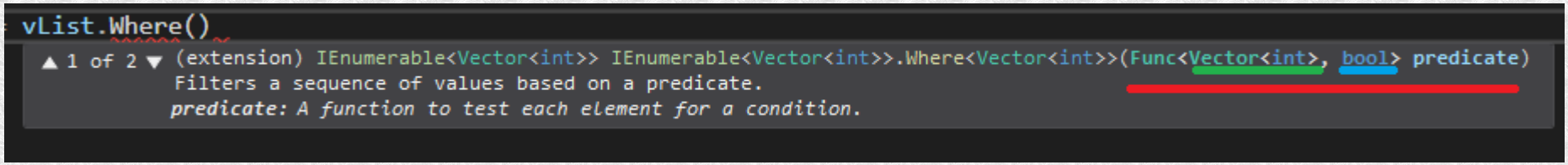
More examples

Predicate

- Predicate is like a Function delegate but return bool and has only one parameter
- As other delegate types, Predicate can also be used with any method, anonymous method, or lambda expression

```
List<Vector<int>> vList = new List<Vector<int>>();
Random rand = new Random();
for (int i = 0; i < 10; i++)
    {
    vList.Add(new Vector<int>(rand.Next(20), rand.Next(20), rand.Next(20)));
    }

foreach (var v in vList) v.Print();

var smallVectors = vList.Where(v => v.X < 10 && v.Y < 10 && v.Z < 10);
Console.WriteLine($"I found {smallVectors.Count()} small vectors");

foreach (var v in smallVectors)  v.Print();
```

Intellisense suggest when predicate should be



More examples

Extension Methods

- When we want add some method to pre compiled class, we can use Extension Methods

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Advanced
{
    public static class Extensions
    {
        public static bool IsNullOrEmpty(this string str)
        {
            if (str == null) return true;
            if (str.Length == 0) return true;
            return false;
        }
    }
}
```

```
string s1 = "aa";
string s2 = "";
string s3 = null;
Console.WriteLine("s1 is null or empty: " + s1.IsNullOrEmpty());
Console.WriteLine("s2 is null or empty: " + s2.IsNullOrEmpty());
Console.WriteLine("s3 is null or empty: " + s3.IsNullOrEmpty());
```

More Info

- For more info visit https://docs.microsoft.com/en-us/dotnet/csharp/
- Read books about programming in .net
- Find some tutorials
- And practise, practise, practise

Thank You