# C# basics

Introduction to Computer Programming

Maciej Piernik

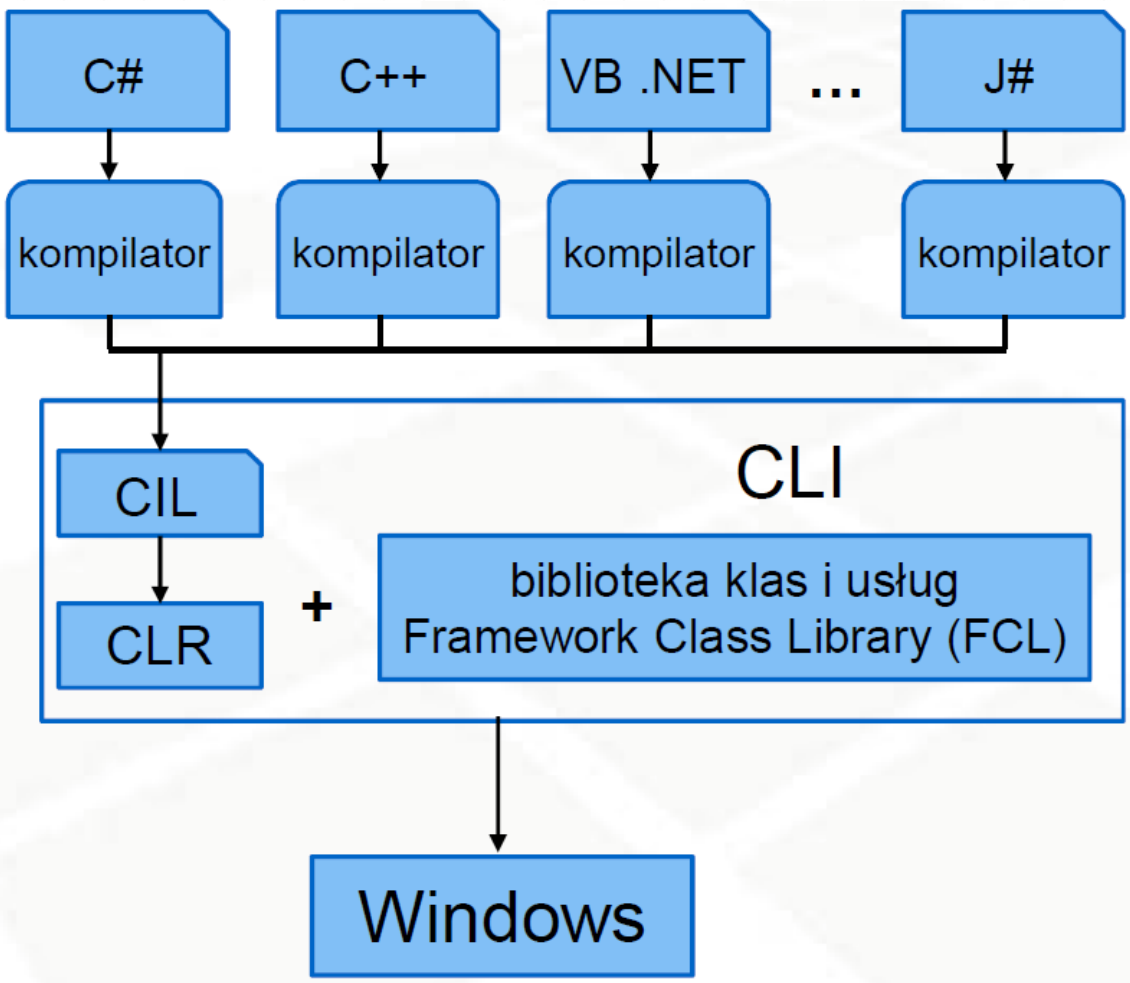
# Review of Lecture 2

- Variable
- Basic types
- Basic operators
- Conditional statements
- Loops

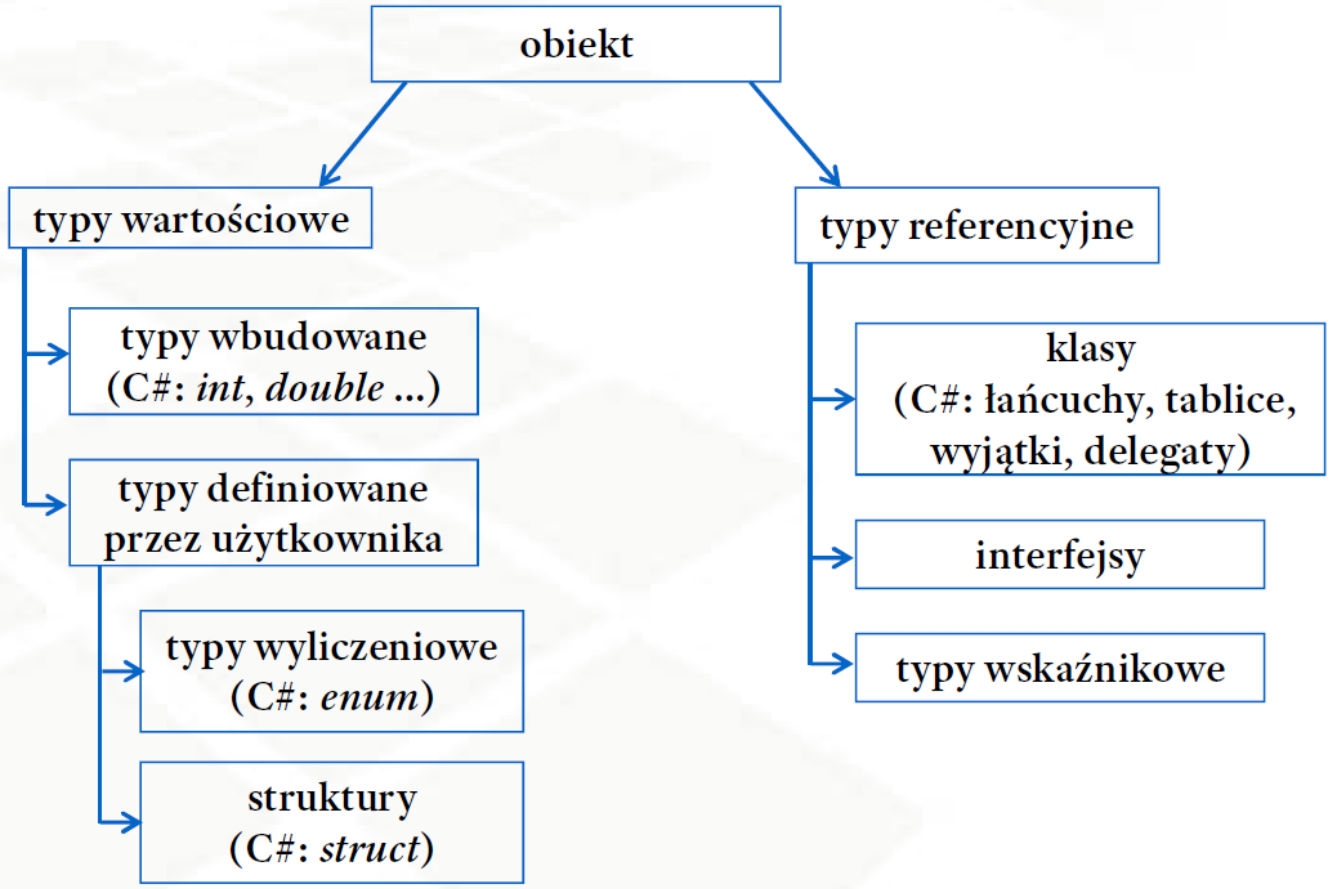
# Outline

- .NET Framework
- Memory management
- Built-in types and literals
- Enumerations
- Altering the flow of loops

# .NET Framework architecture



# Commont Type System



# Memory management

- Memory areas
  - generated code area
  - static data area
  - stack
  - pile
- Memory allocation methods
  - static
  - automatic
  - dynamic

# Value and reference types

- Value types
  - simple variables, enumerations, structures
  - stored on a stack
- Reference types
  - text, arrays, classes
  - stored on a pile

# Dynamic memory allocation

- Allocation
  - 'new' operator
- Release
  - automatic
  - Garbage Collector
  - GC.Collect()

# Integer types

| Type | Range |
|---|---|
| byte | 1B, 0 : $2^8$ |
| sbyte | 1B, $-2^7$ : $2^7-1$ |
| short | 2B, $-2^{15}$ : $2^{15}-1$ |
| ushort | 2B, 0 : $2^{16}$ |
| int | 4B, $-2^{31}$ : $2^{31}-1$ |
| uint | 4B, 0 : $2^{32}$ |
| long | 8B, $-2^{63}$ : $2^{63}-1$ |
| ulong | 8B, 0 : $2^{64}$ |

# Integer literals

- No suffix – sufficiently large type out of int, uint, long, ulong, e.g., 23
- Suffix u – sufficiently large type out of uint, ulong, e.g., 23u
- Suffix l – sufficiently large type out of long, ulong, e.g., 23L
- Suffix ul – ulong, e.g., 23ul

# Variable initialization

```
int x;
Console.WriteLine(x);
```

```
// Initialization with an arbitrary value
int x = 0;

// Initialization using a constructor
int y = new int();

// Initialization with a default value
int z = default(int);
```

# Floating point types

| Type | Range |
|---|---|
| float | 4B, 7 precise digits from range: $\pm 1.5*10^{-45}$ : $\pm 3.4*10^{38}$ |
| double | 8B, 15 precise digits from range: $\pm 5.0*10^{-324}$ : $\pm 1.7*10^{308}$ |

- Literals
  - No suffix or d suffix – double, e.g., 2.3
  - Suffix f – float, e.g., 2.3f
  - Exponential notation allowed, e.g., 2e3d

# Other types

| Type | Description |
| --- | --- |
| char | 2B, Unicode character |
| decimal | 16B, fixed point, 28 or 29 digits precision |
| string | text |
| BigInteger | arbitrality large integer number |
| bool | logical type (true/false) |

# Text literals

- Regular
- Verbatim
- Interpolation

```
string s1 = "This is a regular text. Path: \\home\\docs";

string s2 = @"This is a text,
    in which every
                        character matters verbatim.
Path \home\docs";

int i = 0;
string s3 = $"This is an interpolated text and the value of i is {i}";
```

# Enumeration

- A set of named integers
- By default, first value equals 0

```
enum Directions { Up, Right, Down, Left }

static void Main(string[] args)
{
    Directions k = Directions.Right;
    k = 2;

    Console.Write("Choose a direction: ");
    k = (Directions)Enum.Parse(typeof(Directions), Console.ReadLine());

    k++;

    Console.WriteLine(k);
}
```

# Break and continue

- break – immediately breaks the loop
- continue – immediately skips to the next iteration of the loop

```
while (warunek)
{
    //...
    break;
    //...
}
```

```
while (warunek)
{
    //...
    continue;
    //...
}
```

# ;

- Each instructin in C# ends with ;
- Watch out for empty instructions!

```
for (int i = 0; i < 10; i++) ;
{
    Console.WriteLine("How many times will you see this question?");
}
```

# Summary

- .NET Framework
- Memory management
- Built-in types and literals
- Enumerations
- Altering the flow of loops