Design of Software Systems

Introduction to C#

Assoc. Prof. Dr. Marco Kuhrmann,
Maximilian Irlbeck
Elena Markoska

Agenda

- Introduction to C#
- Language design
- Language features and syntax

C# - General information!

– C# was influenced by

- ▪▪ Java
- ▪▪ C/C++
- ▪▪ Pascal/Delphi
- ▪▪ Haskell
- ▪▪ Modula-3
- ▪▪ Visual Basic

– C# supports the following IDEs:

- ▪▪ **Microsoft Visual Studio** 2003, 2005, 2008, 2010, 2012, 2013
- ▪▪ SharpDevelop
- ▪▪ MonoDevelop
- ▪▪ XNA Game Studio
- ▪▪ C#-Builder
- ▪▪ Baltie

LANGUAGE DESIGN AND TYPE SYSTEM

Design of Software Systems – Introduction to C#

Types in C#!

- C# is a strong-typed, object-oriented programming language!

- Two kinds of types: !
  - Reference Types (`class`, `interface`, `delegate`, arrays, …)!
  - Value Types (`struct`, `enum`, base types, …)!

- Every type is per se a child class of `object`, i.e. all types implement the standard method from `object` (e.g., `ToString()`)!

- **Note**: for reference types exists a special type: `null` – the “empty” reference

The small difference: reference- and value types

Value types

A value type is stored on the **stack**. Hence, an explicit instantiation to create an instance (as a variable) is not necessary. An assignment of a variable always creates a **copy of the value**; if variables of a value type are compared, the **identity of the value** is tested.

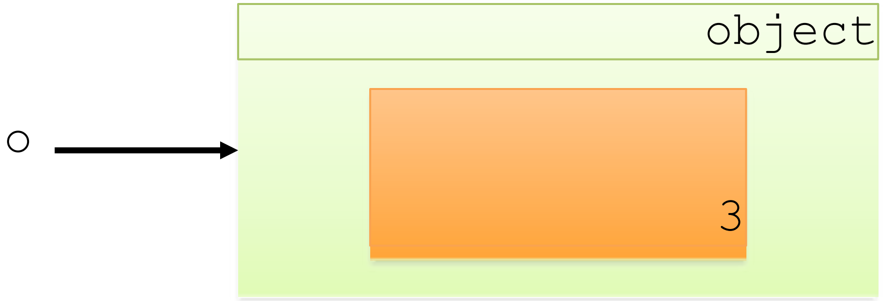
Reference types!

A reference type is stored on the **heap**, the stack only contains a reference to the respective memory address in the heap. To use an instance (create a variable), it is required to instantiate a reference type. An assignment only creates a **copy of the pointer**; if variables are compared, the **identity of the pointers** is tested.!

Boxing/Unboxing

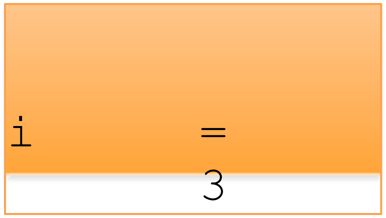
Value types can be stored in “real” objects (reference types): casting

```
object o = (object)3;!
```



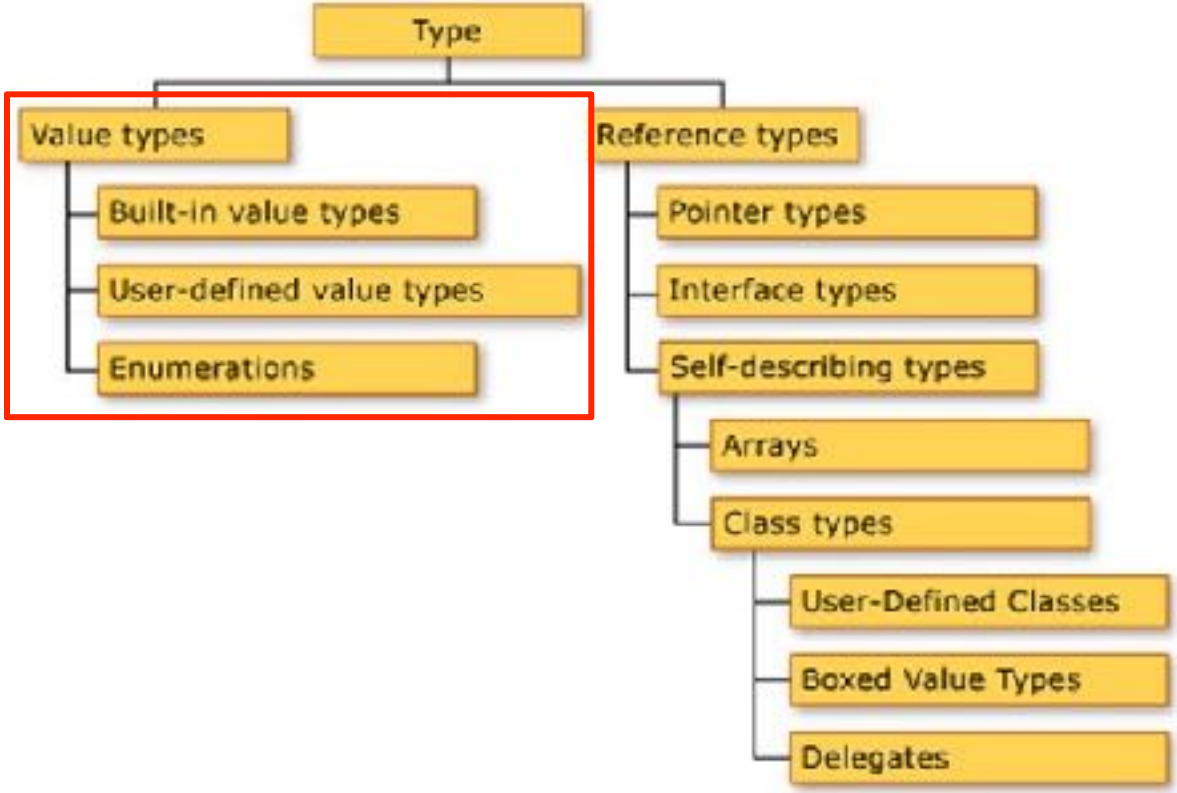
Boxing

```
int i = (int)o;
```



Unboxing

Value Types!


Type
Value types
Built-in value types
User-defined value types
Enumerations
Reference types
Pointer types
Interface types
Self-describing types
Arrays
Class types
User-Defined Classes
Boxed Value Types
Delegates

Value types: base types

Data Type	Range	Size in Bits	.NET Runtime type
bool	true oder false	1	System.Boolean
byte	0 … 255	8	System.Byte
sbyte	-128 … 127	8	System.SByte
char	0 … 65535	16	System.Char
short	-2^{15} … 2^{15} -1	16	System.Int16
ushort	0 … 65535	16	System.UInt16
int	-2^{31} … 2^{31} -1	32	System.Int32
uint	-32.768 … 32.767	32	System.UInt32
float	1,4 x 10^{-45} … 3,4 x 10^{38}	32	System.Single
ulong	0 … 2^{64} -1	64	System.UInt64
long	-2^{63} … 2^{63} -1	64	System.Int64
double	5,0 x 10^{-324} … 1,7 x 10^{308}	64	System.Double
decimal	±1,0 x 10^{-28} … ±7,9 x 10^{28}	128	System.Decimal

Base types – examples…!

```
uint u = 6;
int i = -10;
byte b = 0x01;         // Hexadecimal
float f = 4.0F;
double d = 0.5D;
double d2 = 0.5F;

char c1 = 'Z';         // Character literal
char c2 = '\x0058';    // Hexadecimal
char c3 = (char)88;    // Cast from integral type
char c4 = '\u0058';    // Unicode
char c5 = '\t';        // Special character

decimal d = 440.5m;    // m for Money!
```

Value types – enumerations…!

```csharp
enum DaysOfWeek : byte // any integer type works (e.g. long)
{
   Monday = 1, // standard: 0-indexed, now 1-indexed
   Tuesday,       // = 2
   Wednesday,     // = 3
   Thursday,      // = 4
   Friday,        // = 5
   Saturday,      // = 6
   Sunday         // = 7

};

DaysOfWeek today = DaysOfWeek.Thursday;
int today = (int)today; // today = 4
```

Value types – structures…!

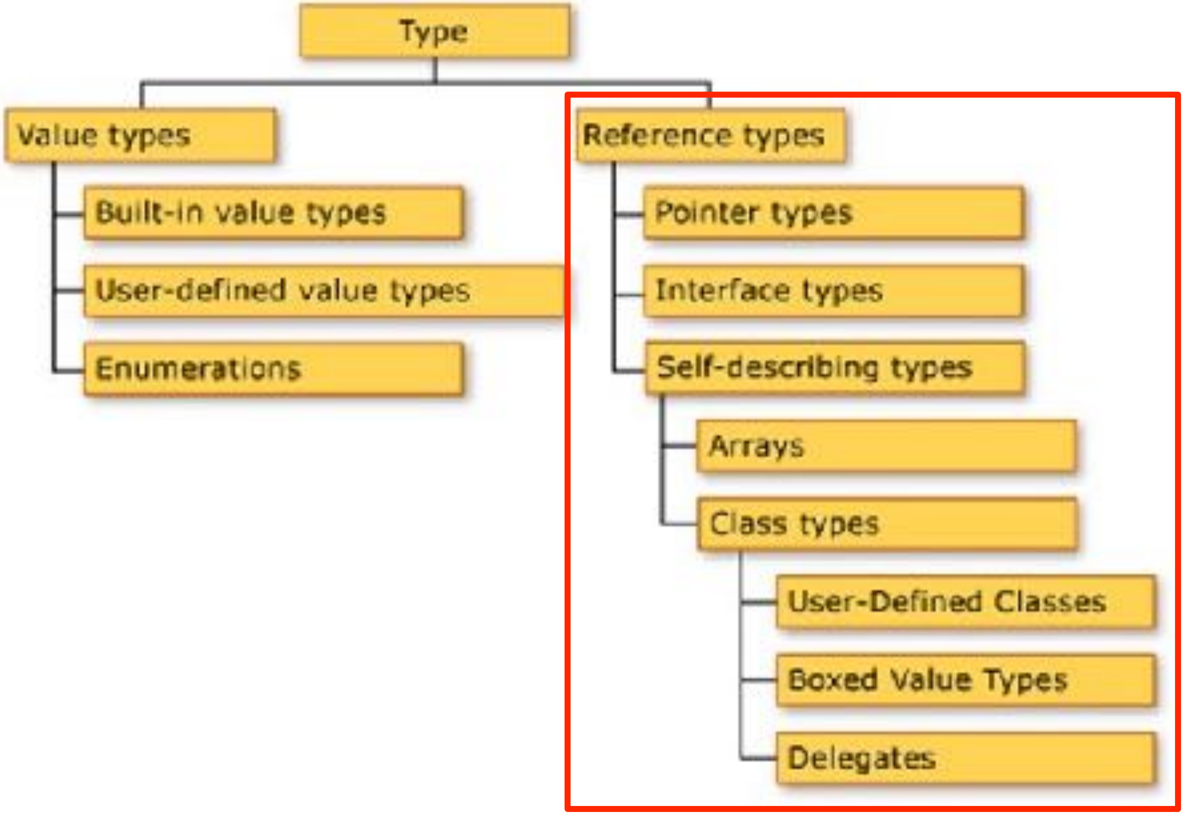
```
public struct Circle {
public double radius;
public double centerX;
public double centerY; }
```

– Structs can also implement:!

- Constructors, Methods, Operators, Events!
- Constants, Fields (variables), Properties, Indexers!
- Nested types!

– Structs can also implement an **interface**, but they cannot inherit from another struct. That is, member of a struct cannot declared **protected**.!

Reference Types!



Types: Arrays!

```
int[] myInts = new int[10];
int[] myInts = new int[10]{0,0,0,0,0,0,0,0,0,0};
```

32 Bit

Value	0	0	0	0	0	0	0	0	0	0
Index	0	1	2	3	4	5	6	7	8	9

```
myInts[2] = 255;
```

Value	0	0	255	0	0	0	0	0	0	0
Index	0	1	2	3	4	5	6	7	8	9

```
myInts[7] = myInts[2] - 1;
```

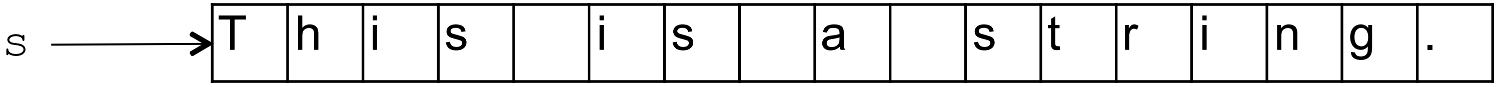
Value	0	0	255	0	0	0	0	254	0	0
Index	0	1	2	3	4	5	6	7	8	9

Types: Strings!

Strings are…!

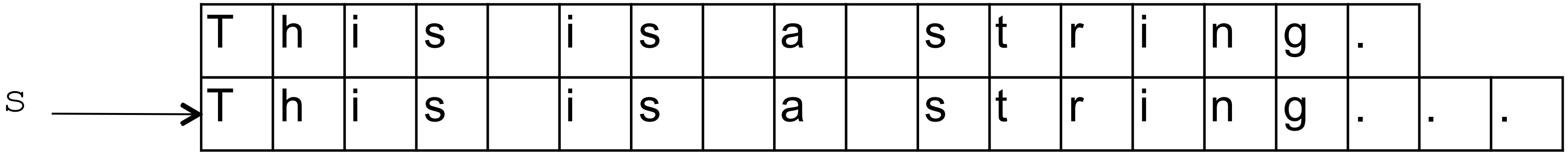
- “Special” reference types!
- Strings are composed of Unicode characters, and are “constant”

```
string s = "This is a string."
```



s → T h i s   i s   a   s t r i n g .

```
s  += ".."
```



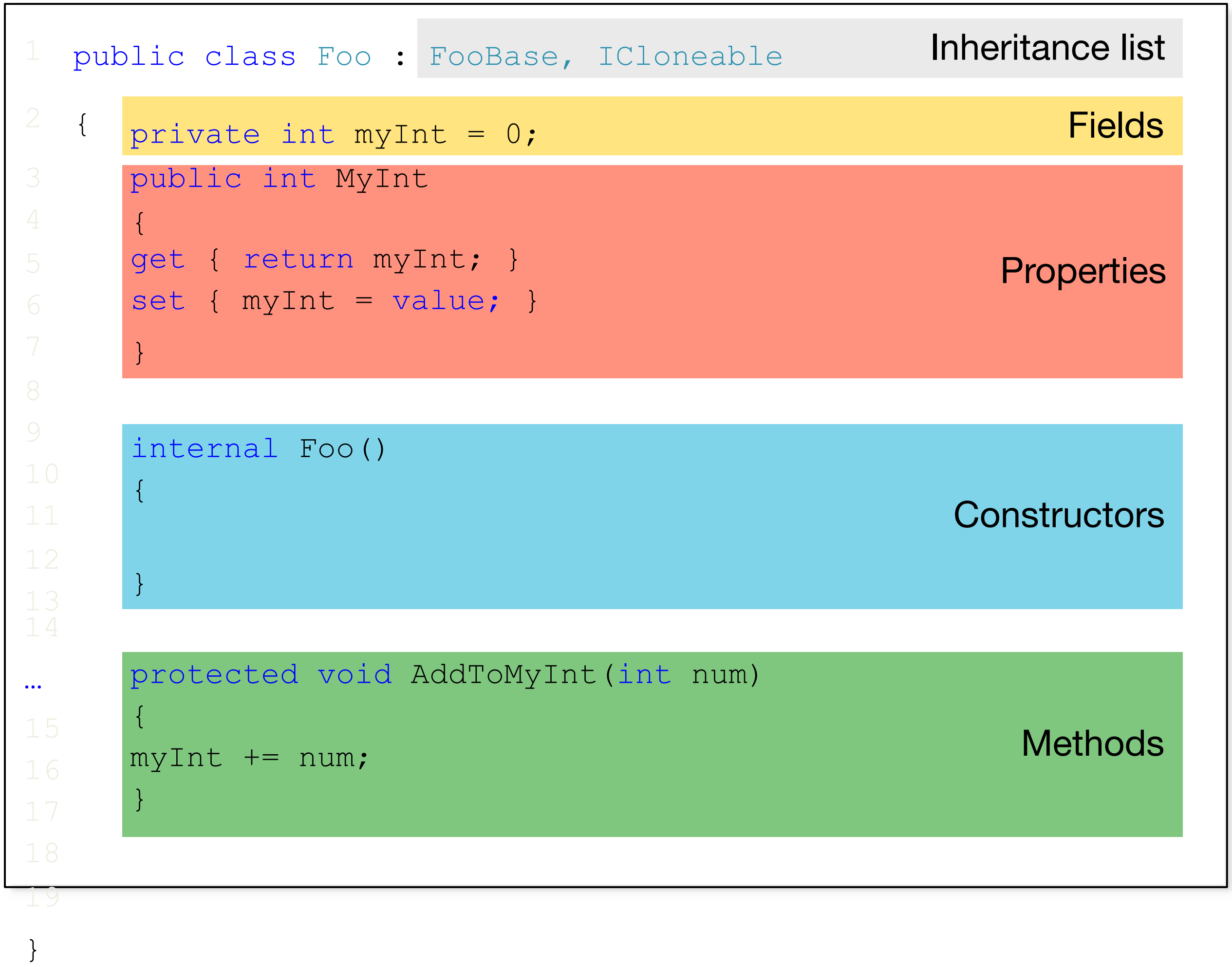
T h i s   i s   a   s t r i n g .

s → T h i s   i s   a   s t r i n g . . .

- Equivalence (== and !=) of Strings **<u>is not</u>** evaluated via reference identity, but by comparing the values of the strings!

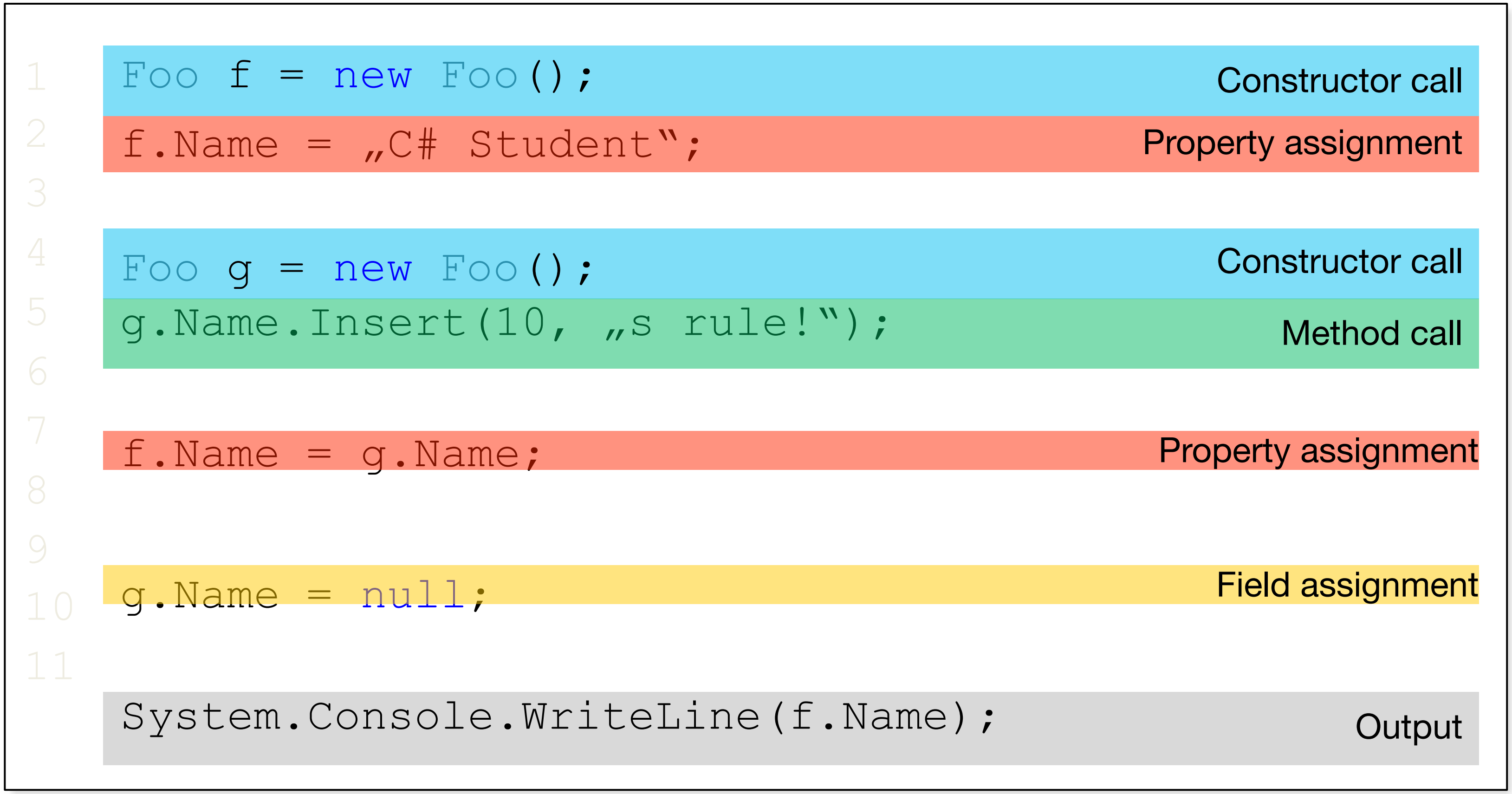
- **Note:** difference to Java; implemented using operator overloading!

Types: Classes!



```
public class Foo : FooBase, ICloneable
 {
   private int myInt = 0;
   public int MyInt
   {
   get { return myInt; }
   set { myInt = value; }
   }

   internal Foo()
   {


   }

   protected void AddToMyInt(int num)
   {
   myInt += num;
   }

}
```

Constructing and using objects!



```
Foo f = new Foo();
f.Name = „C# Student“;

Foo g = new Foo();
g.Name.Insert(10, „s rule!“);

f.Name = g.Name;

g.Name = null;

System.Console.WriteLine(f.Name);
```

Constructor call

Property assignment

Constructor call

Method call

Property assignment

Field assignment

Output

C# language conventions for identifiers!

Pascal Case (e.g., BackColor)!

- Classes!
- Interfaces!
- Events!
- Structures!
- Properties!
- Enumerations!
- Enumeration values!

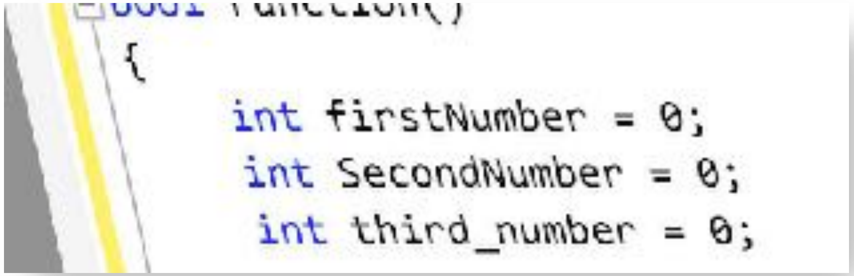
Camel Case (e.g., backColor)!

- Fields!
- Parameters!
- Local variables!

Upper Case (e.g., BACKCOLOR)!

- Constants !
- Identifiers with <= 2 letters

(mostly used for namespaces, e.g., System.IO)!



```
bool Function()
{
        int firstNumber = 0;
         int SecondNumber = 0;
          int third_number = 0;
```

Code organization – Namespaces !

All code is organized in **Namespaces**: A **namespace** is a hierarchical **logical** structure of code and libraries. A namespace is qualified and accessed using the Scope Operator (“.”).

Note: Namespace ≠ Assembly!

Example:

▨ = Namespace

```
System.DateTime

System.Xml.XmlDocument

System.Xml.Serialization.XmlSerializer
```

Import/access a namespace with the keyword `using`.

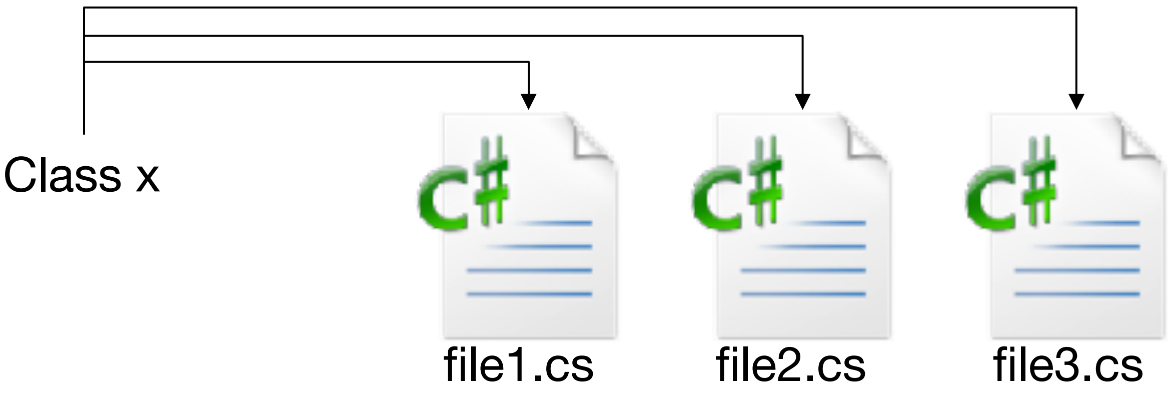
Code organization – files !

- Code is stored in *.cs files.

- Conventions (Note: it’s not a physical restriction)
    - One file per class/structure
    - Class name = file name
    - File system structure = namespace structure

- Special case:

using the keyword `partial`, classes can be split across several files.



Class x

file1.cs

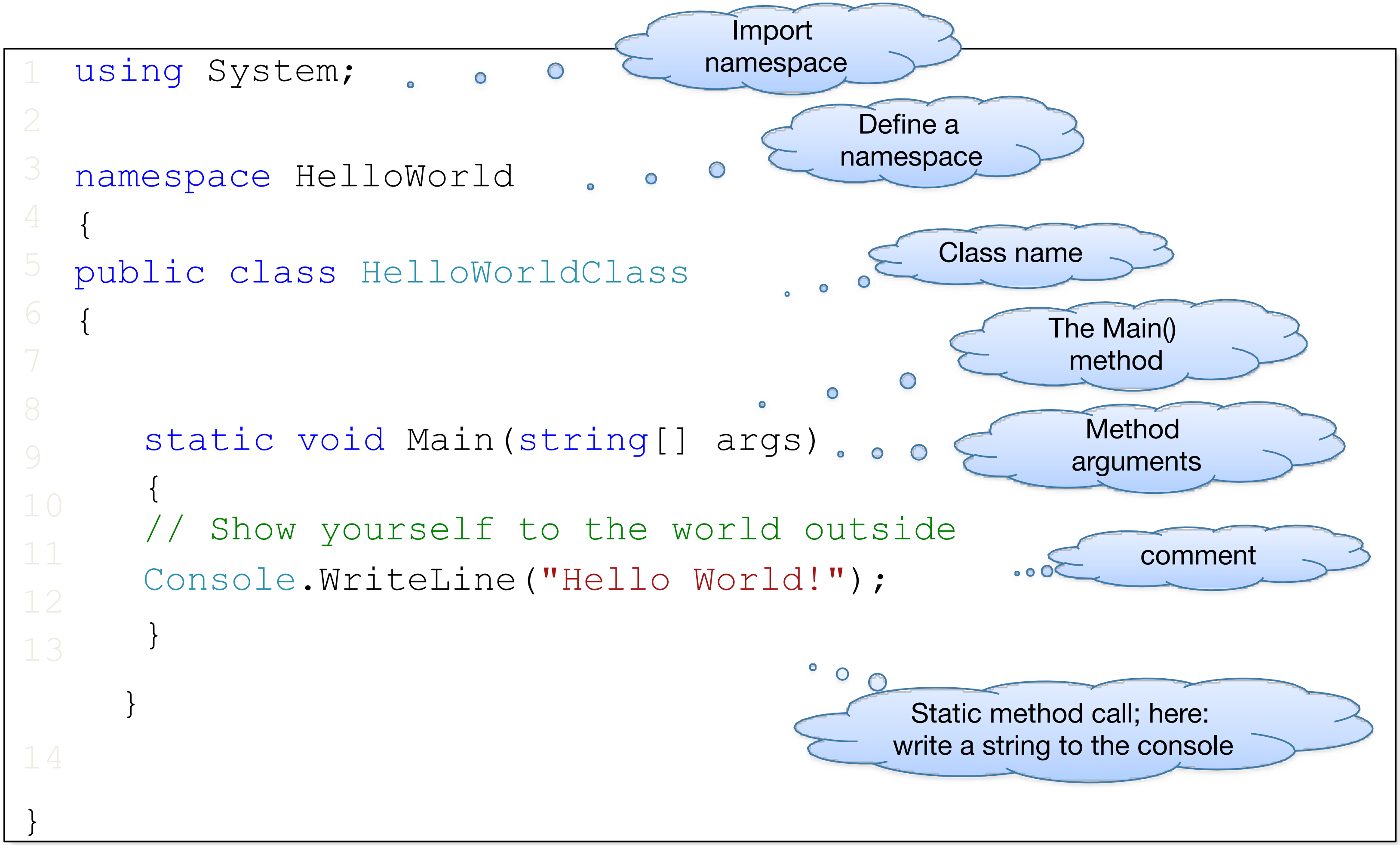
file2.cs

file3.cs

LANGUAGE FEATURES AND SYNTAX!

Design of Software Systems – Introduction to C#!

Hello World in C#



```
  using System;

  namespace HelloWorld
  {
  public class HelloWorldClass
  {


     static void Main(string[] args)
     {
     // Show yourself to the world outside
     Console.WriteLine("Hello World!");
     }

    }

}
```

Import namespace

Define a namespace

Class name

The Main() method

Method arguments

comment

Static method call; here: write a string to the console

Accessibility levels in C# - visibility

Level	Visibility/accessibility
`public`	Globally visible
`internal`	Visible in the same assembly
`protected`	Visible in the inheriting class
`internal protected`	Visible in the inheriting class in the same assembly
`private`	Visible in the class only

Methods

```
/// <summary> Documentation </summary>
<visibility> <return type> <Name>(<Parameterliste>)
{
// implementation
}
```

For example...

```
public static string Sort(string input)
{
char[] arr = input.ToCharArray();
Array.Sort(arr);
return new string(arr);
}
```

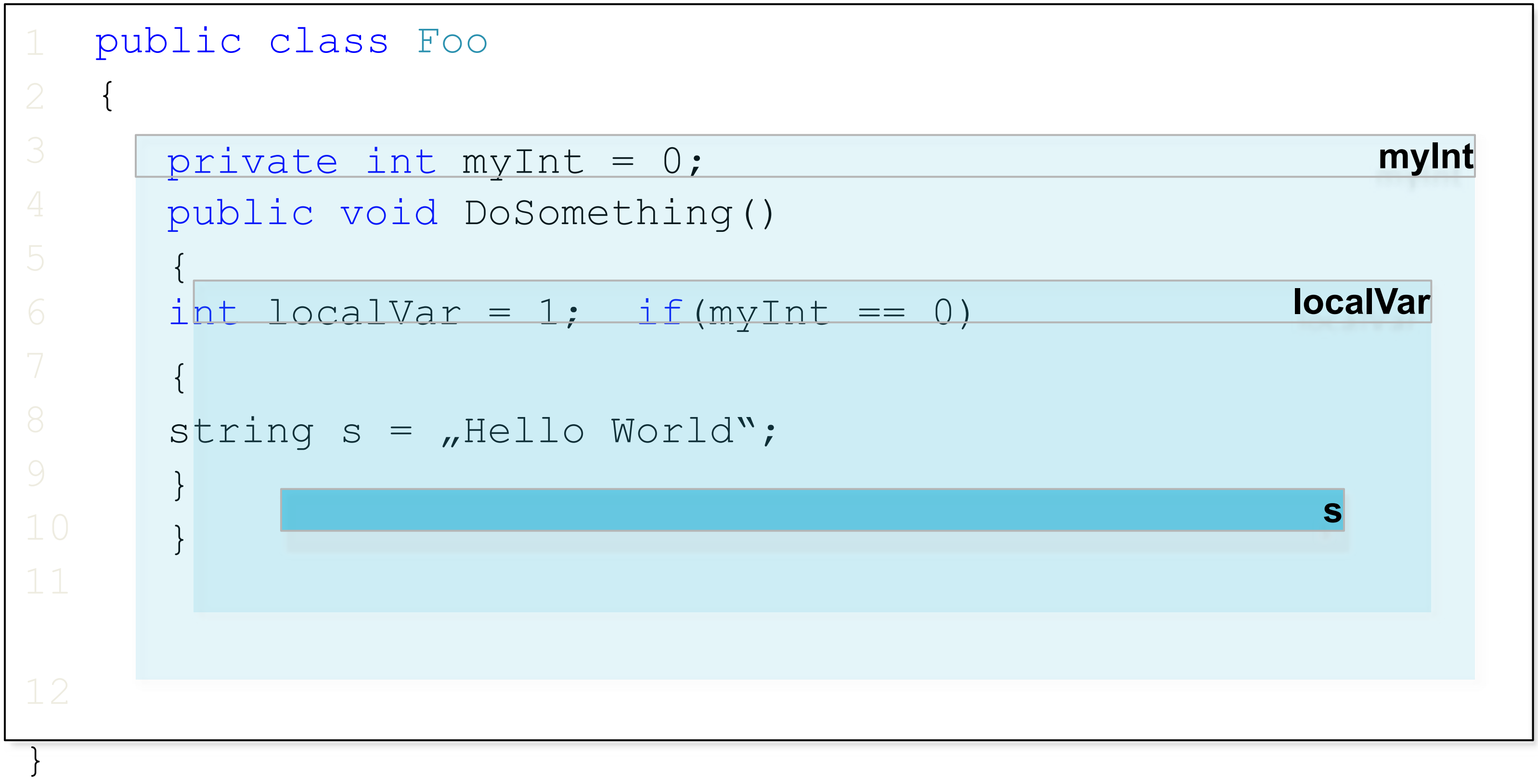
```
internal static void CaesarCipher(char[]

charArray)   {
 for (var i = 0; i < cArray.Length; i++)
 { cArray[i] = (char)(cArray[i] + key); } // Maybe wrap? ;-)

}
```

Methods: example – find the maximum

```
public int Max(int first, int second)
{
if (first >= second)
{
return first;
}
else
{
    return second;
}
}
```

```
public int MaxFunctional(int first, int second)
{
    return first >= second ? first : second;
}
```

Scopes



```
public class Foo
{
    private int myInt = 0;
    public void DoSomething()
    {
    int localVar = 1;   if(myInt == 0)
    {
    string s = „Hello World‟;
    }
    }

}
```

Delegates

- Refer to methods (so-called method pointer)
- Have a defined signature
- In C# used for, e.g., events and asynchronous method calls

```
delegate string MyStringDelegate(string input);
```

```
public string LowerFunction(string inputStr)
{
return inputStr.ToLower();
}


public void OtherFunction(string someString)
{
MyStringDelegate lowerDelegate = LowerFunction;
lowerDelegate(someString); // Calls LowerFunction(someString)
}
```

Generics

Generic classes:

```
class GenericClass<T>     where T        :
IComparable
{
T      member; // T will implement IComparable
…
}
```

Generic methods:

```
U      GenericMethod<T,U>(T input)      where T
       :      new()
{
T      tVal   =      new    T(); // Cool, I'm able to
create T!
…
}
```

Interfaces in C#

```
public interface INumberable // Just an example interface.
{
int Number { get; set;}
string NumberToString();
}
public class NumerableImplementation : INumberable
{
private int number = 0;
public int Number
{
get { return number ; }
set { number = value; }
}
public string NumberToString()
{
        return number.ToString();
}
}
```

Conditional execution and branching

```
bool lazy = true;

bool bored = true;

if (lazy)
{
// I‘m lazy
}
else if (bored)
{
// I‘m not lazy     but bored
}
else
{
// I‘m not lazy & not bored
}
```

```
  DaysOfWeek day =
  DaysOfWeek.Wednesday;
// Typical week of a student
  switch (day)
{
     case DaysOfWeek.Saturday:
     case DaysOfWeek.Sunday:
Sleep(); Eat(); Chill(); Sleep();
        break;
     default:
Sleep();
        GoToLecturesAndSleep();
        EatAtCafeteria();
        GoToLecturesAndSleep();
        ChillOrParty();
        Sleep();  break;
        }
```

Loops

```
while (!feelingLikeParty)
{
// The Party Loop.
DrinkABeer();
}
```

```
do
{
    /* always drink at least
       one beer */
    DrinkABeer();
}
while (!feelingLikeParty);
```

`break` exit the whole loop

`continue` skip one cycle of the loops

```
for (int i = 0; i < 100; i++)
{
// Write a hundred times...
Console.WriteLine(„I will not throw paper
airplanes in class.");
}
```

```
List<string> seminarList =
new List<string>(){„Timm“, „Dominik“, ..};

foreach (string name in seminarList)
{
// Say hello to everyone!
Console.WriteLine(„Hello {0}.“ , name);
}
```

Operators in C#

- **Comparison**

  `==` ! ! !tests equivalence;

  `!=` ! ! !tests non-equivalence

  `<, >, <=,` !test if a relation is fulfilled

  `>=`

- **Logical operators!**

`!`	*NOT* (unary negation)
`&&`	conditional *AND*
`||`	conditional *OR*

- **Type checking and type casts!**

`is`, `typeof`	type compatibility, type exploration
`as`	Cast (checked)
`(<Typ>)`	Cast (unchecked, in case of an error, an exception is thrown)!

Operators in C#

- **Arithmetic operators**

+, -, *, /	basic maths
++, --	(pre- or post-) increment, decrement
%	modulo operation

- **Access- and assignment operators**

[<Index>]	access an indexed data structure
=	(value) assignment
+=, -=, *=,… ″	combined (value) assignment

- **Bit operators**

<<, >>	bit shifting
&, |, !, ^	bit-wise logical AND, OR, NOT, XOR

Operator overloading

Syntax:

```
public static <result type> operator <Operator> (<Operand1>,
<Operand2>)
```

Examples:

```
public static bool operator >=(GeometricObject geoObj1, GeometricObject
geoObj2)
{
if(geoObj1.GetArea() >= geoObj2.GetArea())  return true;
return false;
}

// Special case: Indexer Properties  public T this[int i] {
get { return arr[i]; }  set { arr[i] = value; }
}
```

Attributes in C#

Syntax:

```
[AttributeClass(arguments)]
public void Method()
```

Examples:

```
[WebService(Namespace="http://codeproject.com/webservices/",
Description="This is a demonstration WebService.")]  public class
WebService1 : System.Web.Services.WebService
{
[WebMethod]
public string HelloWorld()
{
return "Hello World";
}
}
```

Tips for the practical route to C#!

The best way to learn a programming language is **using** it

Online training material for free:

- http://msdn.microsoft.com/en-us/library/aa288436(v=vs.71).aspx!
- http://www.csharp-station.com/Tutorial/CSharp/
- http://www.introprogramming.info/english-intro-csharp-book/videos/
- http://simple.wikipedia.org/wiki/C_Sharp_(programming_language)
- http://channel9.msdn.com/Series/C-Sharp-Fundamentals-Development-for-Absolute-Beginners
- http://www.tutorialspoint.com/csharp/index.htm

Happy coding!!!