www.IT-Visions.de®
Dr. Holger Schwichtenberg

.NET and C#

Dr. Joachim Fuchs

About Dr.-Ing. Joachim Fuchs

- University studies: Electrical Engineering RWTH Aachen
- Doctorate at WZL in the field of automation and robotics
- Author of books by Addison Wesley and MS Press
- Author of articles in the dotnetpro magazine and other
- Lecturer for .NET, web applications, desktop applications
- j.fuchs@it-visions.de
- www.fuechse-online.de (German only)

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Agenda

- Chapter 1: Basics .NET and C#
- Chapter 2: C#-Syntax and basics
- Chapter 3: Data types (structures, classes, references)
- Chapter 4: Collections
- Chapter 5: Interfaces
- Chapter 6: Delegates, Lambdas und LINQ
- Chapter 7: Inheritance
- Chapter 8: Exception handling
- Chapter 9: Strings

Chapter 1: Basics .NET and C#

Dr. Joachim Fuchs
j.fuchs@it-visions.de

Overview

- What is .NET?
- What can you do with .NET?
- History: CLR, Framework, Versions
- Programming languages
- Tools, Installation
- Hello World, Console applications

What is .NET?

- Development platform for (almost) all kinds of applications
- .NET-Framework
  - Runtime environment (Common Language Runtime, kurz CLR)
  - Class libraries (for most common purposes ☺)
    - Base Class Library
      - Basistypes, Diagnostics, Collections, Globalization, IO usw.
    - Libraries for web- and desktop-applications, database access, network, office applications etc.
- available for different plattforms
  - multiple windows versions (since Windows NT4)
  - using Mono also usable on Unix, Linux, Mac OS X

What can you do with .NET? (examples)

- Web applications and services
  - ASP.NET (MVC), Web-API
  - Cloud-Applications (Microsoft Azure)
  - Windows Communication Foundation (WCF)
- Desktop applications
  - Windows Forms
  - Windows Presentation Foundation (WPF)
- Mobile applications
  - Universal Windows Platform (UWP)
- Databases
  - ADO.NET, Entity Framework
- Console applications
- Windows PowerShell

Versions

2001	.NET 1.0	CLR 1.0	C# 1.0	>= NT4
2003	.NET 1.1	CLR 1.1		
2005	.NET 2.0	CLR 2.0	C# 2.0	>= Windows 2000
2006/7	.NET 3.0			>=Windows XP
–	=.NET 2.0			
–	+ WPF + WCF + WF + Cardspace			
2007/8	.NET 3.5	CLR 2.0	C# 3.0	LINQ
2008	.NET 3.5 SP1			
2010	.NET 4.0	CLR 4.0	C# 4.0	
2012	.NET 4.5	CLR 4.0	C# 5.0	>=Vista
2015	.NET 4.6 / Core 5	CLR 4.0	C# 6.0	
2017	.NET 4.7 / Core 2.0		C# 7.x	
2019/20	.NET 4.8 / Core 3.1		C# 8.0	
current	.NET 5		C# 9.0	

www.IT-Visions.de®
Dr. Holger Schwichtenberg

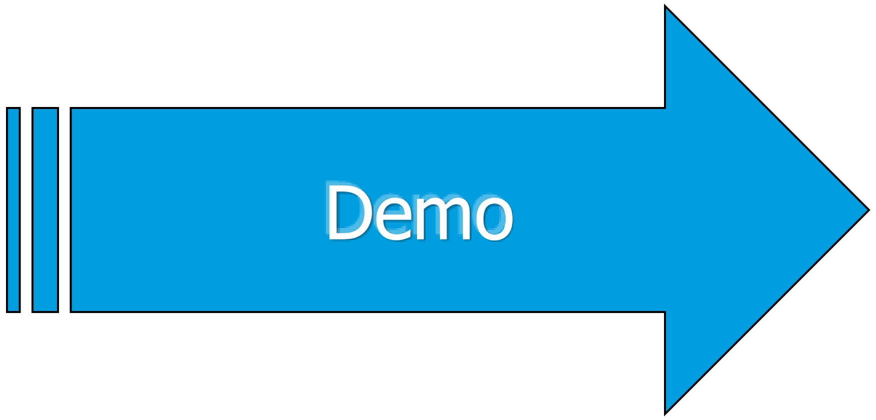
Programming Languages (Selection)

- Microsoft
  - C#
  - VB.NET
  - F#
  - C++
- Others
  - Pascal
  - Python
  - Ruby
  - Eiffel
  - Fortran
  - Cobol

Tools, installation

- Visual Studio 2019
  - free
    - Community 2019
  - chargable
    - Professional 2019
    - Enterprise 2019
- .NET Framework without Visual Studio
- Downloads
  - https://www.visualstudio.com/de-de/downloads/download-visual-studio-vs.aspx

Hello world, console application



www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 2: C#-Syntax and basics

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Agenda

- Comments
- Local variables
- Operators
- Control statements
- Methods
- Overloading methods
- Variable parameter lists
- Optional parameters
- Passing parameters by Reference/Value

Comments

- Create comments:

  // and /* */

```
int i = 0; // annotated statement

// etc.

/* Multi-line comments are possible,
 * but are rarely used,
 * da sie durch einzeilige Kommentare in Verbindung
 * because they can be replaced by single-line
 * comments in conjunction with Visual Studio
 * Support */
```

Comments

- Documentation comments (XML-Comments)
  - Comment public types and their members
  - Intellisense support
  - HTML documentation using additional tools like
    - Sandcastle
      https://sandcastle.codeplex.com/

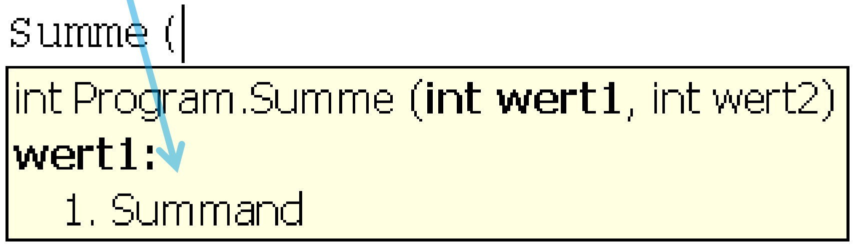
    - NDOC
      http://sourceforge.net/projects/ndoc3/

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Comments

- Documentation Comments (XML-Comments)

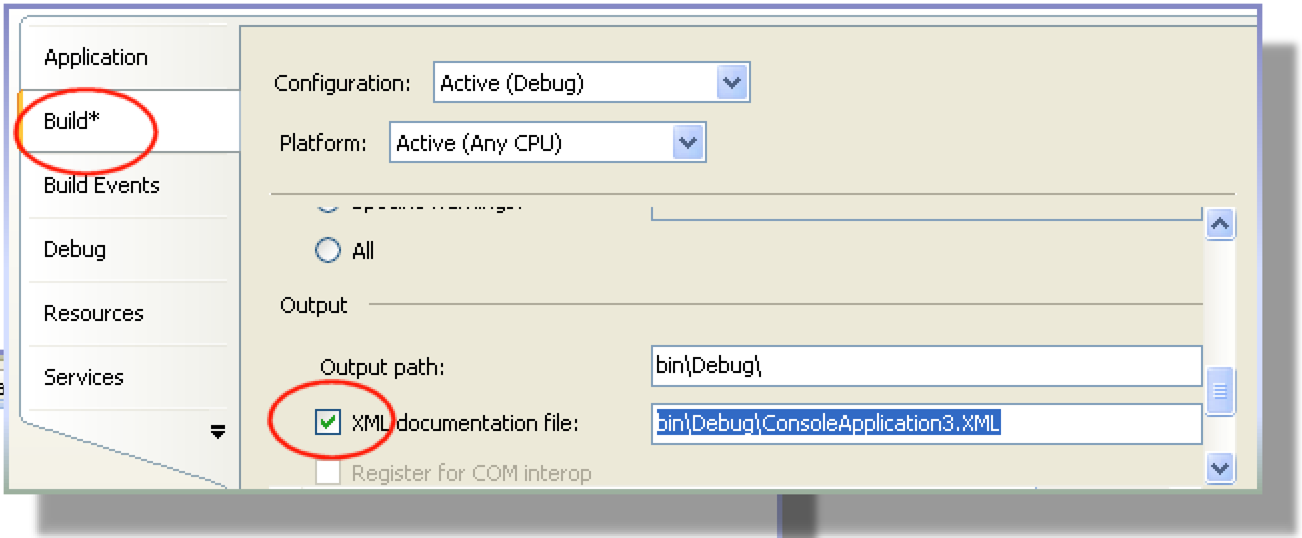
```
/// <summary>
/// Berechnen der Summe
/// </summary>
/// <param name="wert1">1. Summand</param>
/// <param name="wert2">2. Summand</param>
/// <returns>Ergebnis laut Summenformel</returns>
public int Summe(int wert1, int wert2)
{
  return wert1 + wert2;
}
```



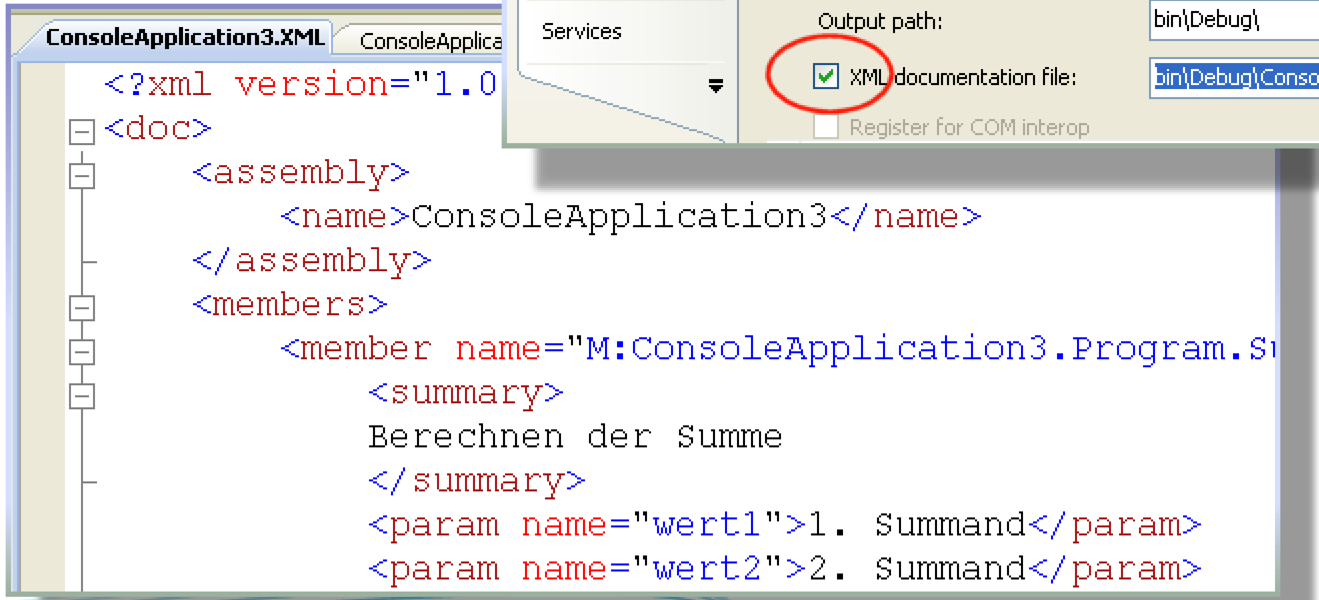
```
Summe (|
int Program.Summe (int wert1, int wert2)
wert1:
   1. Summand
```

Comments

- Documentation Comments (XML-Comments)



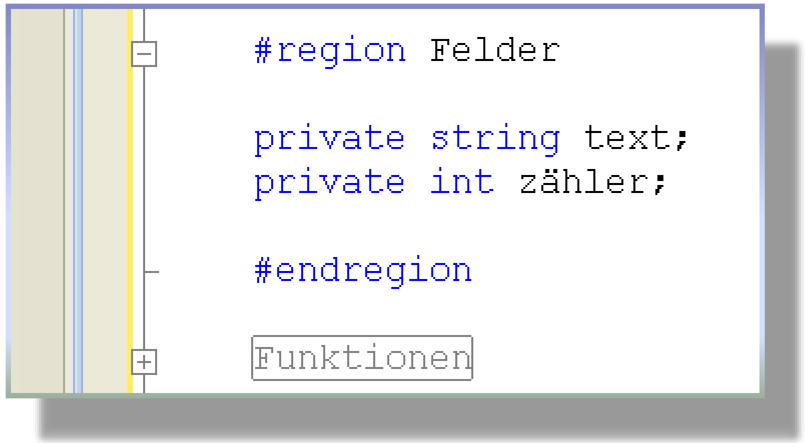
Configuration: Active (Debug)
Platform: Active (Any CPU)
Output path: bin\Debug\
[✓] XML documentation file: bin\Debug\ConsoleApplication3.XML



```
<?xml version="1.0
<doc>
    <assembly>
        <name>ConsoleApplication3</name>
    </assembly>
    <members>
        <member name="M:ConsoleApplication3.Program.S
            <summary>
            Berechnen der Summe
            </summary>
            <param name="wert1">1. Summand</param>
            <param name="wert2">2. Summand</param>
```

Comments

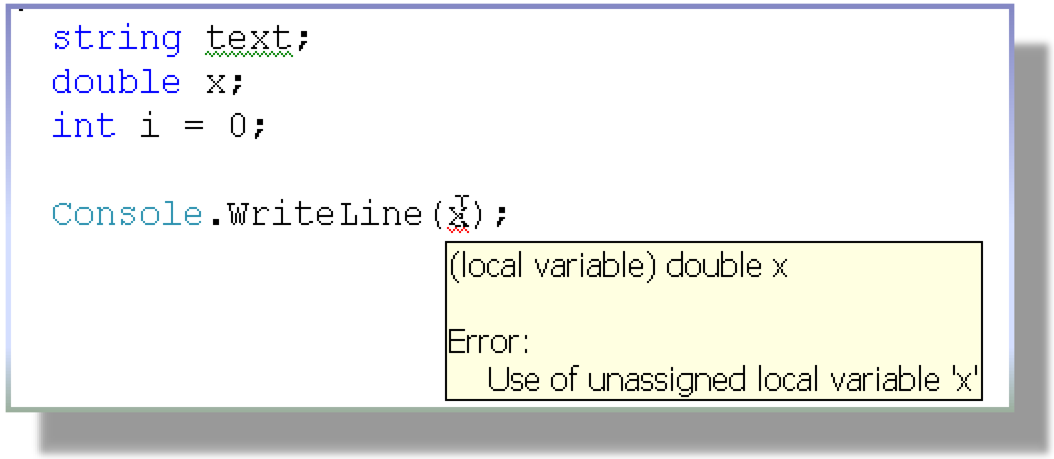
- #region - #endregion
  - Condense blocks
  - Fold and unfold regions in editor



```
#region Felder

private string text;
private int zähler;

#endregion

Funktionen
```

Local variables

- Local variables
  - Definition
  - Value assignment, usage
  - Variables must be initialized before first usage



```
string text;
double x;
int i = 0;

Console.WriteLine(x);
```

```
(local variable) double x

Error:
    Use of unassigned local variable 'x'
```

Operators (selection)

- Arithmetic: +, -, *, /, %
- Comparison: ==, !=, <, >, <=, >=
- Assignment, combinations: =, +=, -=, *= …
- Increment, decrement: ++, --
- Bit-operations: &, |, ~, ^, <<, >>
- Boolean operators: &&, ||, !
- Ternary operator: ?

Control statements

- if / else
- while, do … while
  - Head or foot controlled loops
- for
- foreach
- switch case

Exiting methods or blocks

- break
  - ends a loop
  - leaves a switch block
- return
  - ends a method

Methods

- Definition, call
- Overloading
  - Same name
  - Different parameter list (types, count)
  - Return type doesn‘t matter
  - Compiler must be able to bind to correct method by name and parameter list

Variable parameter lists

- params-keyword
  - Variable number of parameters
  - Only the last can be decorated with params

Optional parameters

- Available since C# 4.0
- Definition of default values
- Multiple syntax variations for passing parameters
  - classic (all parameters in correct sequence)
  - named parameters (*Parametername: theValue*)
  - sequence of named parameters doesn‘t matter
  - parameters can be omitted if a default value is declared
  - non optional parametes must be declared before optional parameters

Passing parameters by Reference/Value

- Default: byValue
- ref: pass an initialized variable
- out: pass a non initialized variable
- ref / out must be used in pairs in definition and in call

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 3: Data types (structures, classes, references)

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Value types (structures)

- standard value types in framework (selection)

C#	Bytes	System.
bool		Boolean
int	4	Int32
long	8	Int64
float	4	Single
double	8	Double
decimal	12	Decimal
char	2	Char
uint	4	UInt32

Value types (structures)

- Complete copy by assignments
- Should only contain other value types
- Storage depends on context
- Keep small (<= 16 bytes)
- Programming a parameterless constructor impossible
- No inheritance possible

Namespaces

```
namespace Firma
{
  public class Klasse1...

  namespace Bereich
  {
    namespace Produkt1
    {
      public class Klasse2 ...
    }
  }

  namespace Abteilung.Unterabteilung.Basics
  {
    public class Klasse3...
  }
}
```

Firma.Klasse1

Firma.Bereich.Produkt1.Klasse2

Firma.Abteilung.Unterabteilung.Basics.Klasse3

Classes (reference types)

- class – object - reference
  - Namespaces
    - for each file type, the namespace must be either fully specified
    - or listed by *using* at the beginning of the file
    - *using* binds exactly one namespace
    - the full qualification must be made with ambiguities
    - *using* does not bind any libraries! It only represents a syntactic simplification!

Classes (reference types)

- class – object - reference
  - Fields
    - Describe the state of an object
    - Will usually be declared with *private* in order to protect them from outside changes
    - Will always be initialized

Classes (reference types)

- class – object - reference
  - Constructors
  - Instantiation (new)

Classes (reference types)

- class – object - reference
  - objects are instances of classes
  - a reference is needed to access an object
  - a reference points to an existing object or has the value null
  - if two references are equal, they point to the same object
  - the operators == and != compare references by default but can be overloaded (see String-class)

Classes (reference types)

- class – object - reference
  - Objects have
    - a status (described by fields)
    - a behavior (described by methods and properties)
    - an identity (described by their references)

Classes (reference types)

- class – object - reference
  - this-Referenz
    - refers inside of instance methods to the object for which the method was invoked
    - can be inserted automatically by the compiler, provided that the member name in the current context is unique

Classes (reference types)

- class – object - reference
  - visibility

C#	is known
private	only inside the class
public	everywhere
protected (different in Java!)	only inside inheritance chain
internal	inside assembly as public, outside as private
protected internal	inside assembly as public, outside as protected

Attention! private etc. does not provide any security! It is used for structuring only.

Classes (reference types)

- class – object - reference
  - Properties (get, set)
    - look like fields from the outside
    - encapsulate two methods internally
    - set {} can be omitted -> ReadOnly property
    - get {} can be omitted -> WriteOnly property
    - multiple support in Visual Studio

Classes (reference types)

- class – object - reference
  - properties (get, set)
  - many mechnisms of the framework depend on public properties
    - auto properties: simplified syntax with hidden field {get;set;}

Classes (reference types)

- class – object - reference
  - properties (get, set)
    - the visibility of the get- or set- accessor can be limited additionally
    - C# 6:
      - initializer for auto properties
      - get-only auto properties

Classes (reference types)

- class – object - reference
  - methods and constructors
    - can be overloaded on same level
  - methods can also be overloaded across inheritance levels

Classes (reference types)

- class – object - reference
  - instance members versus **class members** (static)
    - belong to class, not to objects
    - can be referenced from the outside only by the class name
  - static fields can be initialized
  - static methods or properties do not have a *this*-reference

Classes (reference types)

- class – object - reference
  - static constructors
    - only one per class
    - must not have modifiers or parametes
    - will be executed once prior to any acces to class
    - will be executed after initialization of static fields

Classes (reference types)

- class – object - reference
  - live time of objects, Garbage Collection
    - you can only instantiate objects (new)
    - there is no way to delete an object
    - if the program does not reference an object any longer, the memory managemet system may remove it silently from the heap
    - memory managemet runs asynchronous and ist activity cannot be predicted
    - do not try to interfere with the garbage collection

Classes (reference types)

- class – object - reference
  - live time of objects, Garbage Collection
    - destructors (finalizer) can be programmed, but their call is not predictable
    - memory management is adjusted exclusively to the managed heap. External resources are NOT considered (see dispose-pattern)

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Type casts and checks

- Typecasts
- is-Operator
- as-Operator

Type casts and checks

- Typecasts
  - implicit oder explicit
  - always checked at compile time and at runtime
  - lead to an exception if types do not match

Type casts and checks

- is-operator
  - allows the type checking at runtime
- as-operator
  - combines typecast and *is* operator
  - does not throw exceptions

??-operator

- checks if first operand is null
- result is:
  - 1. operand, if not null
  - 2. operand otherwise

```
Person found= ...
Person person = found ?? new Person { ...};
```

Boxing

- occurs when an object reference is assigned a value type
- boxed values are stored on the heap and are read-only
- repeated boxing same value results in multiple objects of different identity
- unboxing requires a typecast to the type of boxed value

Generic data types

- language extension since C# 2.0
- like templates in C++
- allow type independent programming
- one or more type variables allowed
- for each type variable constrains can be defined
- generic classes are meta classes and cannot be instantiated

Generic data types

- Constraints
  - base class
  - interfaces
  - class (type must be reference type)
  - struct (type must be value type)
  - new() (type must contain a public default constructor)

Generic data types

- Nullable<T>
  - adds null reference for value types
  - allows query whether value exists or not
- Abbreviation:
  - ?

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 4: Collections

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Collections

- Arrays
  - describe a coherent, linear addressable memory range of values of the same type
  - can be single or multi-dimensional
  - are reference types, so they always live on the heap
  - arrays must be instantiated
  - after instantiation the size is unchangeable
  - array elements are always initialized (default values)

Collections

- Arrays
  - declaration and usage
  - single and multiple dimensions

Collections

- Arrays
  - instance methods and properties

Name	Meaning
Length	# of all elements
Rank	# of dimensions
GetLength(dimension)	# of elements for a dimension
[] – Indexer	acces of elements
GetUpperBound(dimension)	max index
CopyTo (overloaded)	copy data

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Collections

- Arrays
  - static methods of Array class

Name	Bedeutung
Sort()	sort
IndexOf()	linear search
BinarySearch()	binary search. expects sorded data
Clear()	set elements to their default values
Reverse()	reverse order
Copy (überladen)	copy data

Collections

- object based collections
    - ArrayList
    - Hashtable
    - Queue
    - SortedList
  - used for all data types
  - no type safety at compile time
  - all data accesses use object references!
  - when dealing with values types -> Boxing!

Collections

- object based collections
  - ArrayList
    - internally stores an array of object references
    - manages the internal array automatically
    - Add elements by *Add* () or *Insert* ()
    - supports methods for searching and sorting

Collections

- object based collections
  - ArrayList
    - implements IEnumerable -> foreach…
    - access by indexer (type cast required)

Collections

- object based collections
  - other collection types

Name	Funktion
Hashtable	Key-/Value- list
Queue	FIFO- list
Stack	LIFO- list
SortedList	sorted list

Collections

- Generic collection types (C# 2.0)
    - List
    - Dictionary
    - more
  - typified definition
  - operate internally with the specified data types
  - no boxing
  - Type safety at compile time

Collections

- Generic collection types (C# 2.0)
  - List<T>
    - generic variant of ArrayList
  - Dictionary<TKey, TValue>
    - generic variant of Hashtable

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 5: Interfaces

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Interfaces

- Definition
  - only declarations of
    - methods
    - properties
    - events
  - no implementations
  - no accessibility modifiers
  - interfaces can inherit from each other

Interfaces

- Definition
- Implementation
  - implicit
    - as public members
  - explicit
    - by interface name

Interfaces

- Examples in .NET framework
  - IComparable
  - IComparer
  - IEnumerable
  - IList
  - IDisposable

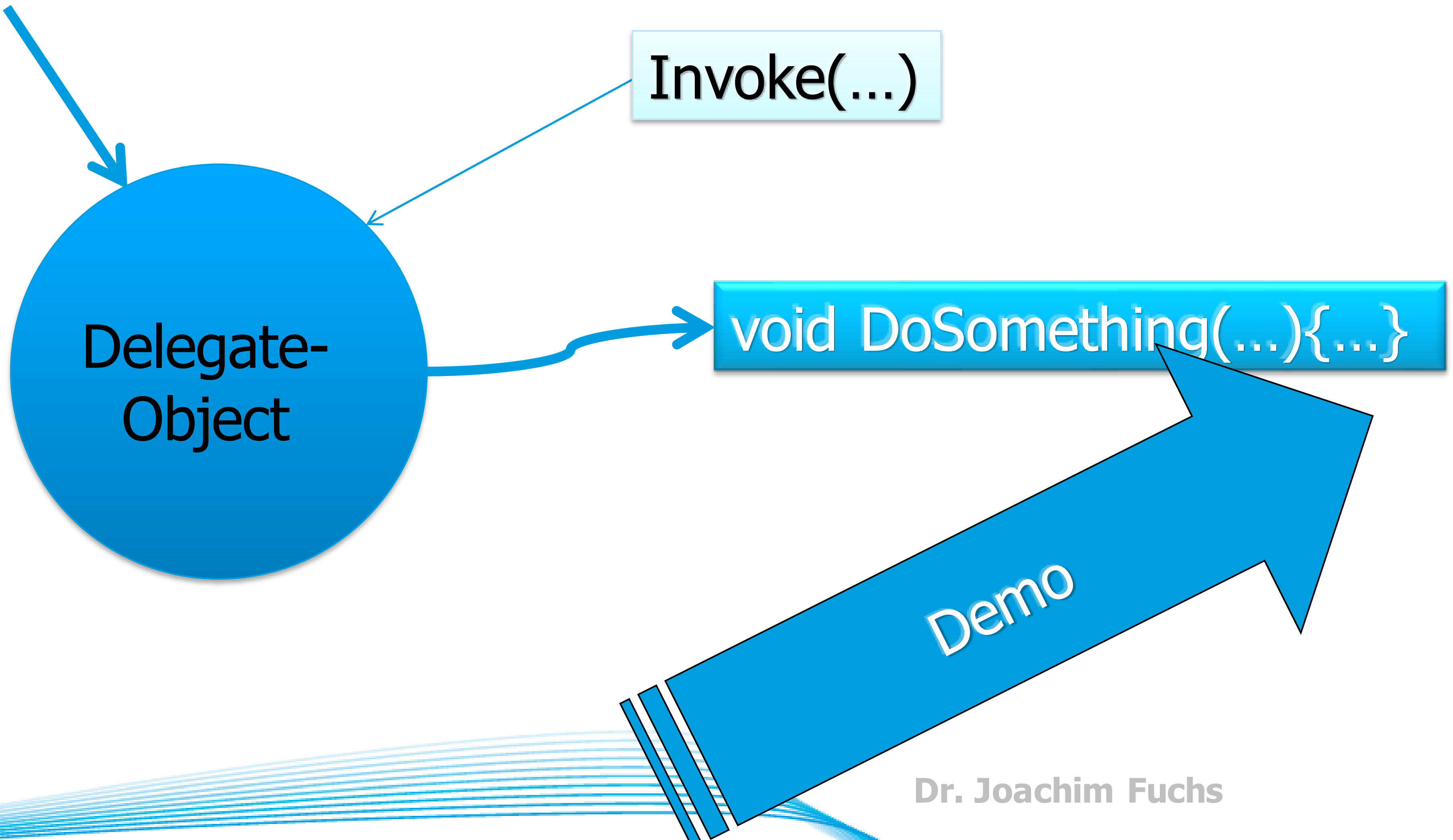
www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 6: Delegates, Lambdas and LINQ

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Delegates



www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 7: Inheritance

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Inheritance

- each class has exactly one base class
- C# does not support multiple inheritance
- base class is implicitly System.Object
- methods are NOT automatically virtually
- non-virtual methods can be hidden (magic word *new*)
- virtual methods can be overridden (magic word *override*)
- using sealed subsequent inheritance (classes) or overrides (methods) can be prevented

Inheritance

- abstract classes
  - must be marked with abstract
  - can not be instantiated
  - derived classes must implement all abstract members or even be marked with abstract
- abstract methods and properties
  - must also be labeled abstract
  - do not have an implementation

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 8: Exception handling

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Exceptions

- Meaning and purpose
  - intercepting otherwise not retrievable error situations
  - should be, as the name suggests, only be the exception
  - handover of error conditions that can not be resolved at the current level
- syntax:
  try {…}
  catch (…){}
  …
  finally {}

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Chapter 9: Strings

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Strings

- Storage
  - strings are reference types!
  - internally coded with Unicode (2 bytes per char)
  - operatores ==, != compare contents!
  - difference between null-reference and empty string
  - strings are immutable (can not be changed)
  - functions and operators create new objects

Strings

- instance methods and properties

Method / property	Meaning
Length	# of characters
IndexOf, IndexOfAny	linear search
StartsWith, EndsWith	compare beginning or end
Substring	substring
ToLower, ToUpper	lower / upper case
Trim, TrimEnd, TrimStart	remove characters at beginning or end
Insert, Remove, Replace	

Strings

- static methods and operators of class String

Method / property	meaning
+, Concat	concat strings
==, !=	comparison
Format	formatting
Join	join array elements to one string
IsNullOrEmpty	check if reference is null or points to an empty string
literals:	
@"c:\A\B\x.txt"	avoid \ escape sequences

Strings

- StringBuilder
  - uses predefined buffer
  - avoids multiple object instantiation when working with string operations

www.IT-Visions.de®
Dr. Holger Schwichtenberg

String interpolation (C# 6)

- Simplifies String.Format
  - define parameters inline
  - escape-Sequence for braces: {{ or }}

```
string s1 = $"Length: {m1.LengthInMeter:0.00}m";
  -> Length: 100,00m
string s2 = $"Employer: {firstEmployer.Name}, Age:
                  {firstEmployer.Age:000} years";
  -> Employer:          Peter Smith, Age: 045 years
string json = $"{{age:\"{firstEmployer.Age}\",
                        name:\"{firstEmployer.Name}\"}}";
  -> {age:"45", name:"Peter Smith"}
```