

Die .NET-Technologie

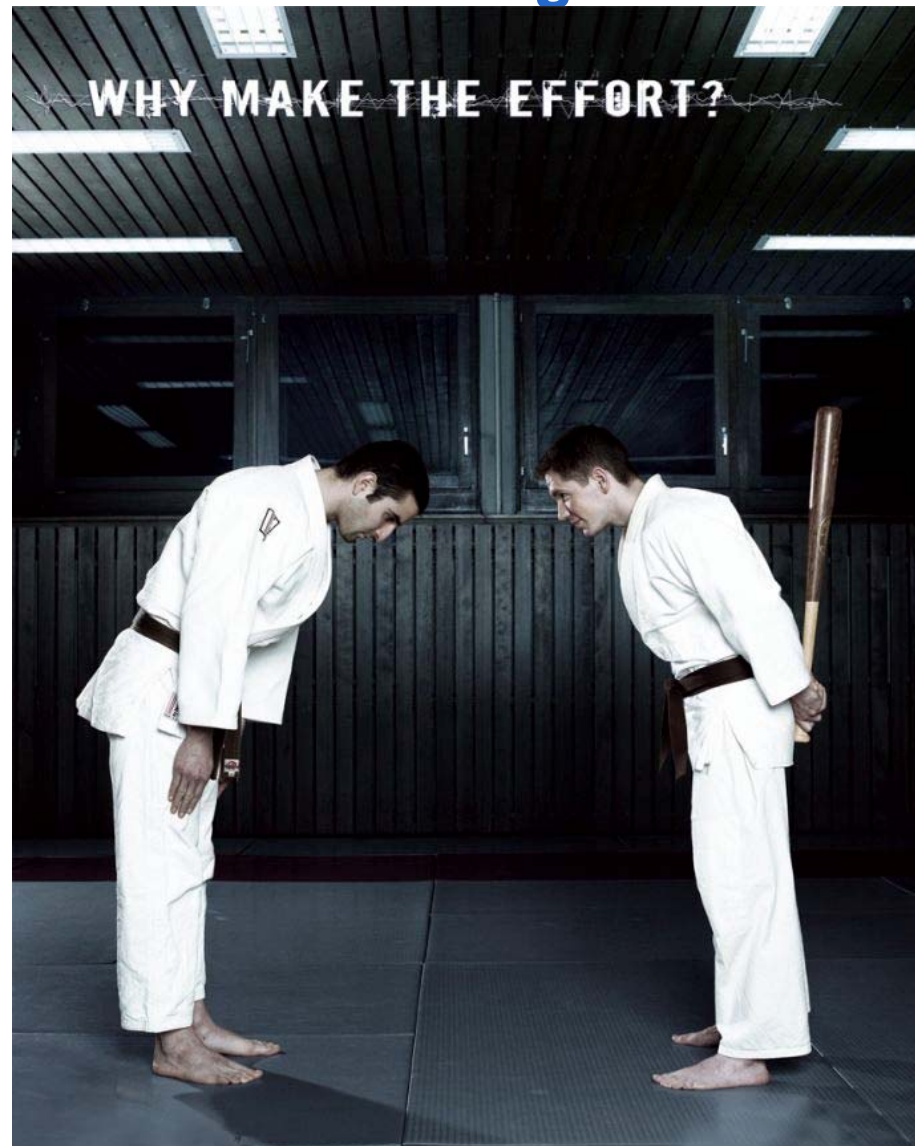
■ Karl Rege

<http://www.zhaw.ch/~rege>

Teil der Folien mit freundlicher Genehmigung von
Prof. Dr. Hanspeter Mössenböck, Dr. Wolfgang Beer, Dr. Herbert
Prähofer (Uni Linz)



Motivation für diese Vorlesung



Motivation für diese Vorlesung

- Man wird im Informatiker Berufsleben fast unausweichlich mit dieser Technologie in Kontakt kommen (ausser man wechselt den Beruf).
- Entwickler, die beide Welten (Java/Open-Source, Microsoft) kennen, haben bessere Berufsaussichten.
- Integration verschiedener Systeme wird eine wichtige/die wichtigste kommende Herausforderungen für SW-Ingenieure sein.
- In .NET sind einige Konzepte umgesetzt, die man bisher im Studium noch nicht kennengelernt hat; es ist auch ein Beispiel von gutem Engineering.

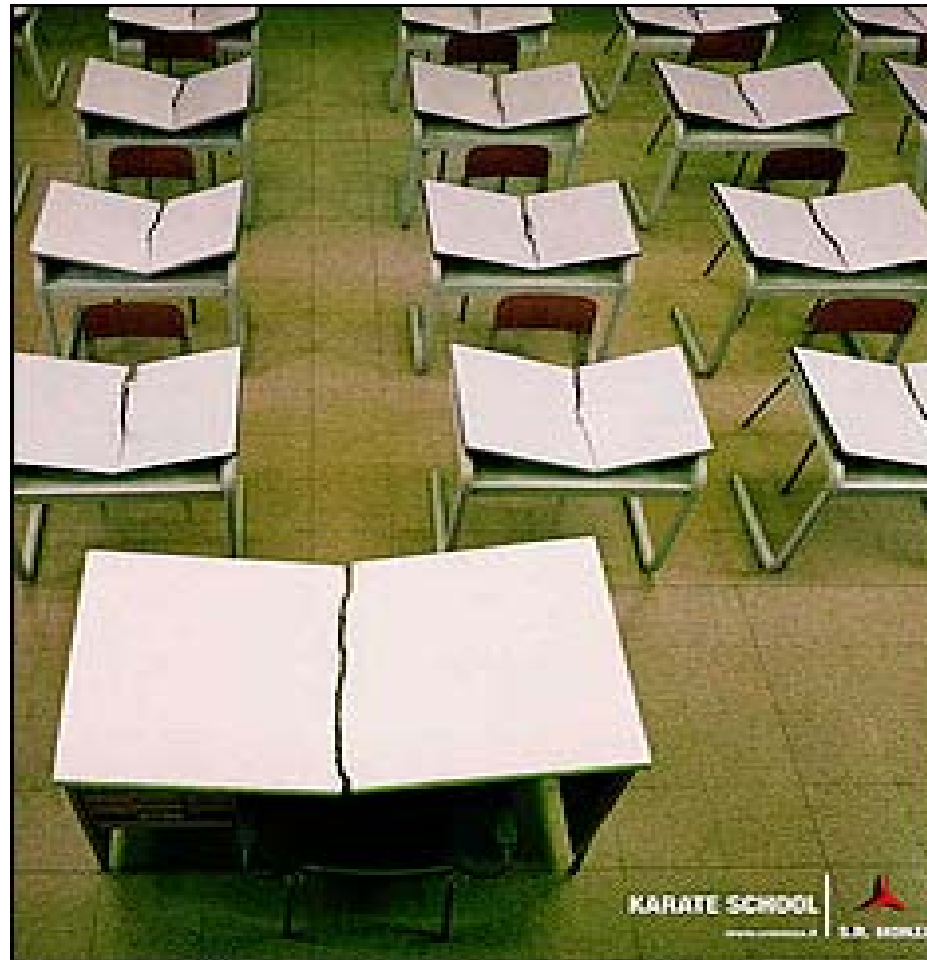
Pro Microsoft

- Microsoft stellt die **beste Software der Welt** her.
- Sie handelt **absolut selbstlos** und will nur das Beste für die Welt.
- C#/.NET ist bei Microsoft die zentrale Plattform für die Anwendungs-entwicklung

Pro Java/FOSS/Apple

- M\$ ist das **Evil Empire** der Softwarebranche.
- M\$ versucht den Software-Markt zu **monopolisieren**.
- Lerne deinen Kontrahenden kennen.
- Man kann sich von M\$ "inspirieren" lassen und gute Ideen in die Open-Source Welt hinüber nehmen

Aufbau der Vorlesung



Aufbau der Vorlesung

■ Semesterplanung (ohne Gewähr)

1	Einführung, Die Sprache C# Teil 1 Einführung, Symbole, Typen, Ausdrücke, Deklarationen, Anweisungen, Ein-/Ausgabe
2	Die Sprache C# Teil 2 Klassen und Structs, Vererbung, Interfaces
3	Die Sprache C# Teil 3 Delegates und Events, Ausnahmen, Namensräume,
4	Die Sprache C# Teil 4 Attribute, automatisch generierte Kommentare
5	C# Teil 5, Die .NET Architektur Virtuelle Maschine, CTS, CLS, CIL, Metadaten, Assemblies und Module, VES, Sicherheit, Zeiger
6	Klassenbibliothek Teil 1 Collections, Ein-/Ausgabe, Threading
7	Klassenbibliothek Teil 2 Netzwerkcommunication, Reflection, Graphische Benutzeroberflächen

8	Klassenbibliothek Teil 3 XML, ActiveX, DLL
9	ADO.NET&XML Verbindungsorientierter und Verbindungsloser Zugriff, DataSets, DataSets und XML Daten
10	ASP.NET Teil 1 dyn Webseiten, Web- Formulare, Ereignisbehandlung, Steuerelemente, Validierung
11	ASP.NET Teil 2 Eigene Steuerelemente, Zustandsverwaltung, Applikationsschicht, Konfiguration
12	Web-Services Überblick, .NET Client, Java Client, SOAP, WSDL, Finden von Web-Services
13	VB.NET
14	.NET 3.0, 3.5, 4.0, ...

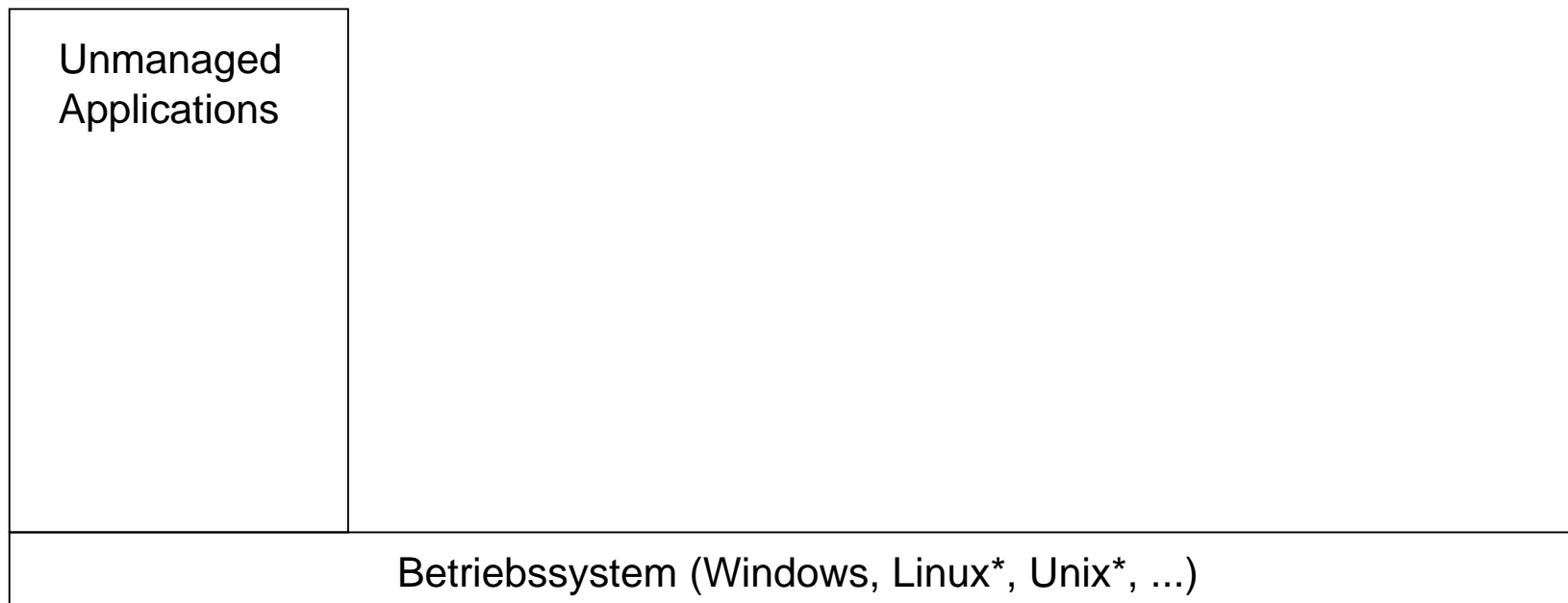
Unterlagen

- Folien: www.zhaw.ch/~rege
- Buch: .NET Kompaktkurs C# 4.0
Mössenböck et. al.
dpunkt Verlag
- ISBN: 978-3-89864-645-1



Was ist .NET?

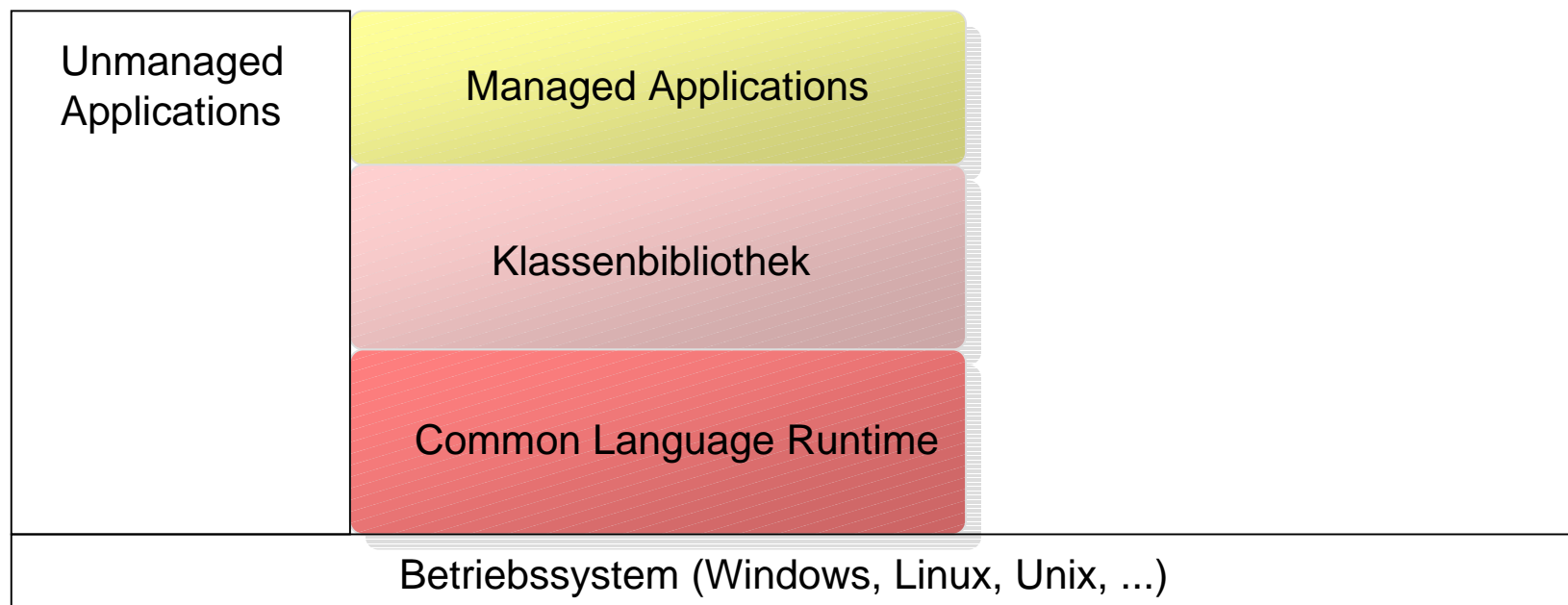
- Eine Software-Plattform für Desktop und Web Anwendungen



*) Mono Projekt

Was ist .NET?

- Eine Software-Plattform für Desktop und Web Anwendungen



**Common Language
Runtime**

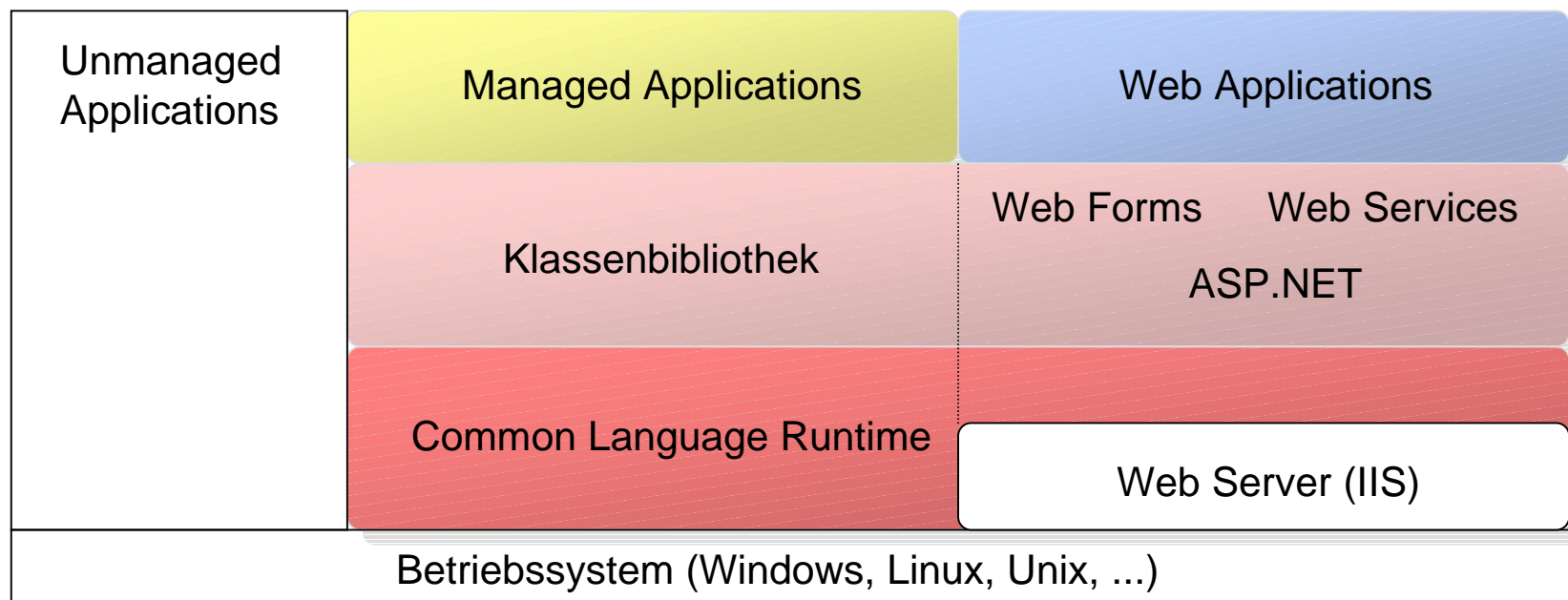
Interoperabilität, Sicherheit, Garbage Collection,
Just-In-Time Compilation, Versionierung, ...

Klassenbibliothek

GUI, Collections, Threads, Netzwerk, Reflection, XML, ...

Was ist .NET?

- Eine Software-Plattform für Desktop und Web Anwendungen



ASP.NET,
Web Forms

Web-GUI (objektorientiert, ereignisorientiert, browseunabhängig)

Web Services

verteilte Dienste über RPC (SOAP, HTTP)

Was ist .NET?

■ Ein Framework und mehr ...

Unmanaged Applications	Managed Applications	Web Applications
	Klassenbibliothek	Web Forms Web Services ASP.NET
	Common Language Runtime	Web Server (IIS)
Betriebssystem (Windows, Linux, Unix, ...)		

.NET-Framework

- + **Tools** (Visual Studio .NET, ildasm, gacutil, ...)
- + **Server** (SQL Server, BizTalk Server, ExchangeServer, ...)
- + **Services** (My Service, ...)

Ziele von .NET

■ Zusammenführung von Desktop- und Web-Programmierung

Bisher

Desktop-Programmierung

Objektorientiert
Compiliert (C/C++, Fortran, ...)
Klassenbibliothek

Web-Programmierung

ASP (nicht objektorientiert)
Interpretiert (VBScript, Javascript, PHP, ...)
Eigene Bibliothek

Ziele von .NET

■ Zusammenführung von Desktop- und Web-Programmierung

Bisher

Desktop-Programmierung

Objektorientiert
Compiliert (C/C++, Fortran, ...)
Klassenbibliothek

Web-Programmierung

ASP (nicht objektorientiert)
Interpretiert (VBScript, JavaScript, PHP, ...)
Eigene Bibliothek

Unter .NET

Desktop- und Web-Programmierung

Objektorientiert (auch ASP.NET)
Compiliert (C#, C++, VB.NET, Fortran, ...)
Einheitliche Klassenbibliothek

Ziele von .NET

- Interoperabilität zwischen Programmiersprachen
- *Bisher*
 - Millionen Zeilen Code in C++, Fortran, Visual Basic, ...
 - Nur sehr beschränktes Zusammenspiel: über OLE/ActiveX

■ Interoperabilität zwischen Programmiersprachen

■ *Bisher*

- Millionen Zeilen Code in C++, Fortran, Visual Basic, ...
- Nur sehr beschränktes Zusammenspiel: über OLE/ActiveX

Unter .NET

- Binärkompatibilität zwischen mehr als 20 Sprachen (C#, C++, VB.NET, Java, Pascal, PHP, Eiffel, Fortran, Cobol, ML, Haskell, Oberon, Perl, Python, ...)

Klasse in VB.NET

```
Public Class A
    Public x As Integer
    Public Sub Foo() ...
End Class
```

Unterklasse in C#

```
class B : A {
    public string s;
    public void Bar() {...}
}
```

Verwendung in Java

```
class C {
    private B obj;
    ...
    obj = new B();
    obj.Bar()
    ...
}
```

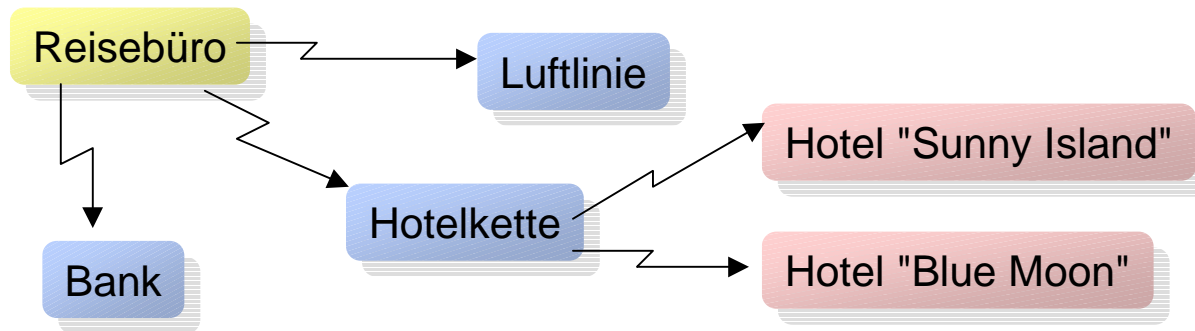
Ziele von .NET

- XML Web Services: Verteilte Applikationen am Internet
- *Bisherige Benutzung des Internets*
 - Email
 - Web-Browser (zeigt Daten für menschlichen Betrachter)

Ziele von .NET

- XML Web Services: Verteilte Applikationen am Internet
- *Bisherige Benutzung des Internets*
 - Email
 - Web-Browser (zeigt Daten für menschlichen Betrachter)

Unter .NET: B2B-Applikationen ohne Web-Browser



Basierend auf einfachen Standards

- HTTP
- SOAP, REST (XML)
- Remote Procedure Call

Ziele von .NET

- Einfachere dynamische Webseiten
- *Bisher*
 - ASP (Mischung von HTML und VBScript oder Javascript)

Ziele von .NET

- Einfachere dynamische Webseiten
- *Bisher*
 - ASP (Mischung von HTML und VBScript oder Javascript)

Unter .NET

- ASP.NET (saubere Trennung von HTML und Script-Code)



Objektorientiert
Ereignisorientiert
Interaktiv erstellbar (RAD)
Selbstgeschriebene GUI-Elemente mögl.
Effizient (compilierte Server-Scripts)
Zustandsverwaltung
Autorisierung / Authentifizierung
...

Ziele von .NET

■ Mehr Qualität und Komfort

■ Sicherheit

■ Side by Side Execution

■ Einfachere Software-Installation

■ Strenge Typenprüfung

■ Laufzeitprüfungen (keine Buffer Overruns mehr!)

■ Garbage Collection

■ CIL-Code-Verifier

■ Public Key Signierung von Code

■ Rollenbasierte Rechte

■ Codebasierte Rechte

■ Versionierung

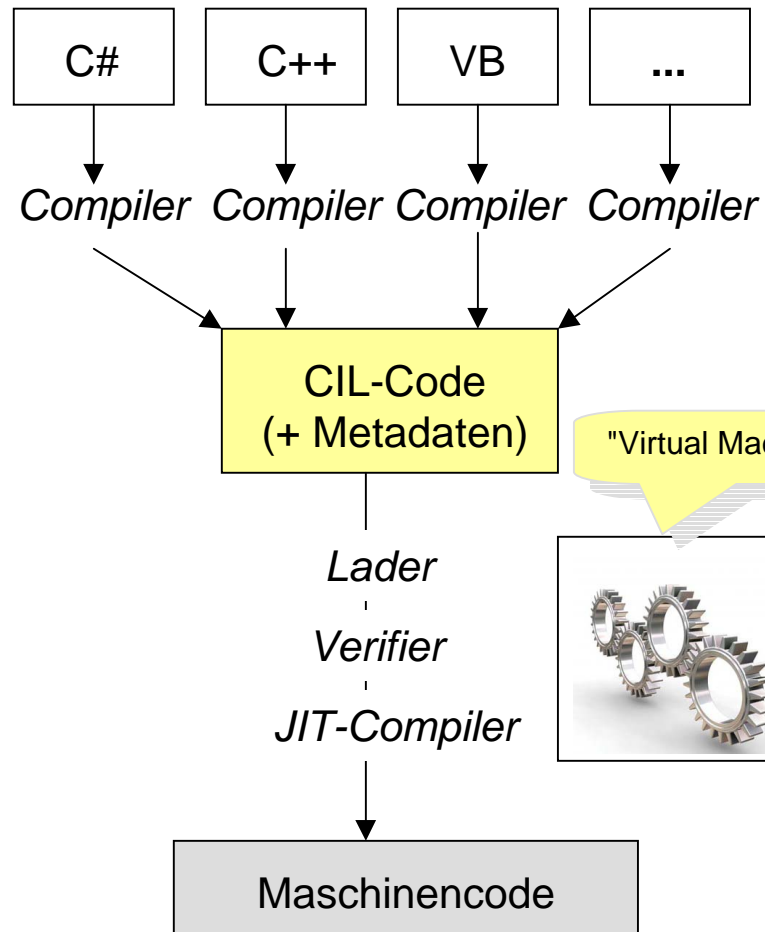
■ Ende der DLL-Konflikte

- DLLs verschiedener Versionen erlaubt

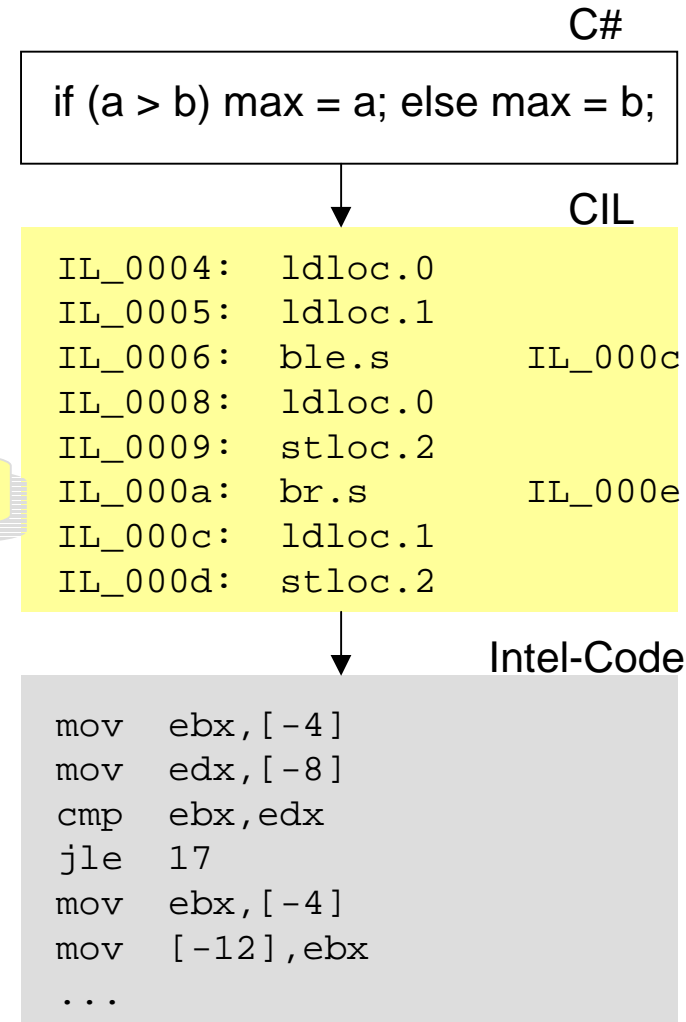
■ Keine Registry-Einträge mehr nötig

- aber u.U. Eintrag in GAC

■ Spurlose Deinstallation



"Virtual Machine"



.NET 2.0 Basis Technologie

■ Sprache C#, etc.

- Generics
- Partielle Klassen
- Typisierte Iteratoren
- Anonyme Methoden
- Nullable Types
- Plattform
- Unterstützung von 64 Bit
- Performanceverbesserungen

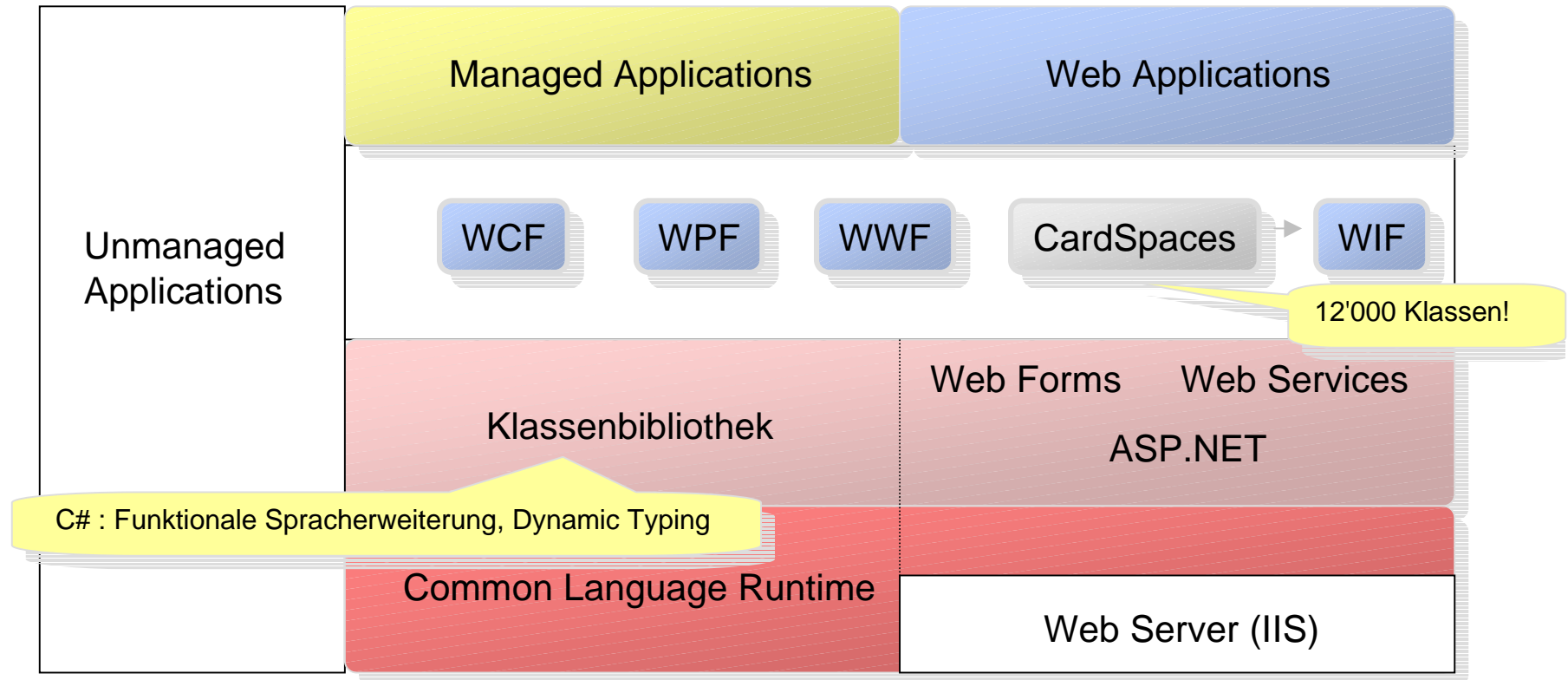
■ Bibliothek

- neue, erweiterte Steuerelemente
- ASP.NET 2.0: Master Seiten, Themes, etc.

■ Werkzeug: Visual Studio

- MS-UML Unterstützung
- Für alle Express Edition **gratis**, für ZHAW Studenten auch Professional Edition

.NET 3.5, 4.0 Erweiterungen



WCF

Windows Communication Foundation

WPF

Windows Presentation Foundation (Silverlight)

WWF

Windows Workflow Foundation

CardSpaces

Claims based Security (retired february 2011) -> AD und WIF

Vergleich Java-Welt/.NET

.NET

- C# Programming Language
- .NET common components (aka the ".NET Framework SDK")
- Active Server Pages+ (ASP.NET). Web Forms
- IL Common Language Runtime
- Win Forms
- ADO and SOAP-based Web Services
- andere Sprache: VB, Java, ...
- **verschiedene Sprachen, eine Plattform**

Java

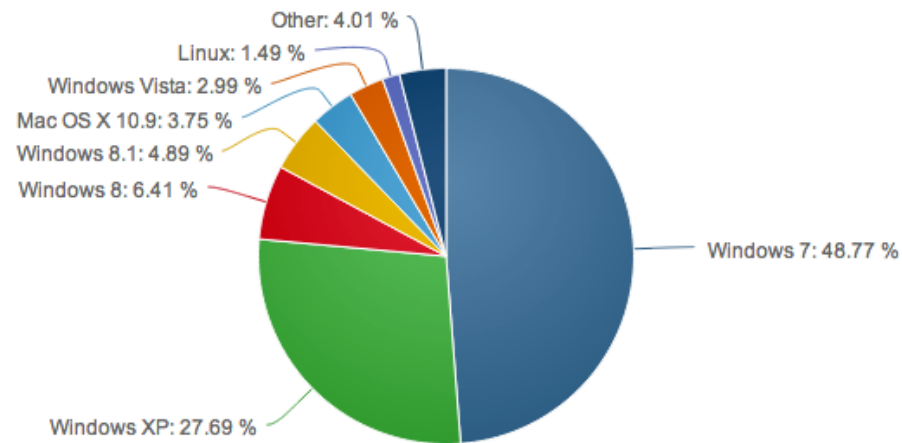
- Java Programming Language
- Java Core API JSE/JEE 5
- Java ServerPages (JSP)
- Java Server Faces (JSF)
- Java Virtual Machine
- Java Swing
- JDBC, EJB, JAXP
- Andere Plattformen Unix, Linux ...
- **eine Sprache verschiedene Plattformen**

Schlacht um das OS

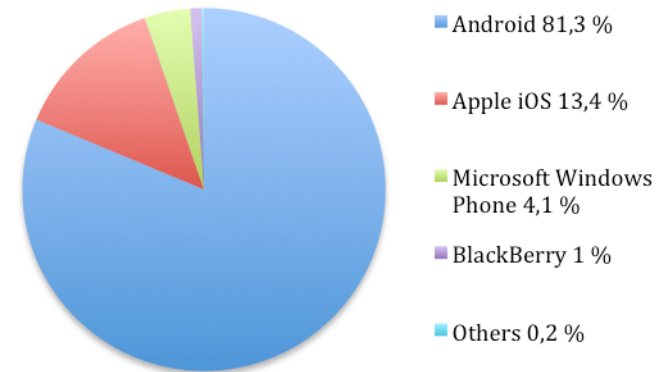


Global OS Aufteilung 2014 Q1

Desktop OS



Mobile OS



Microsoft's/.NET Zukunft?

- War vor 10 Jahren dominante Plattform - auch Mobile (Windows CE)
- Was hat S. Ballmer nur falsch gemacht?



... Microsoft's/.NET Zukunft?

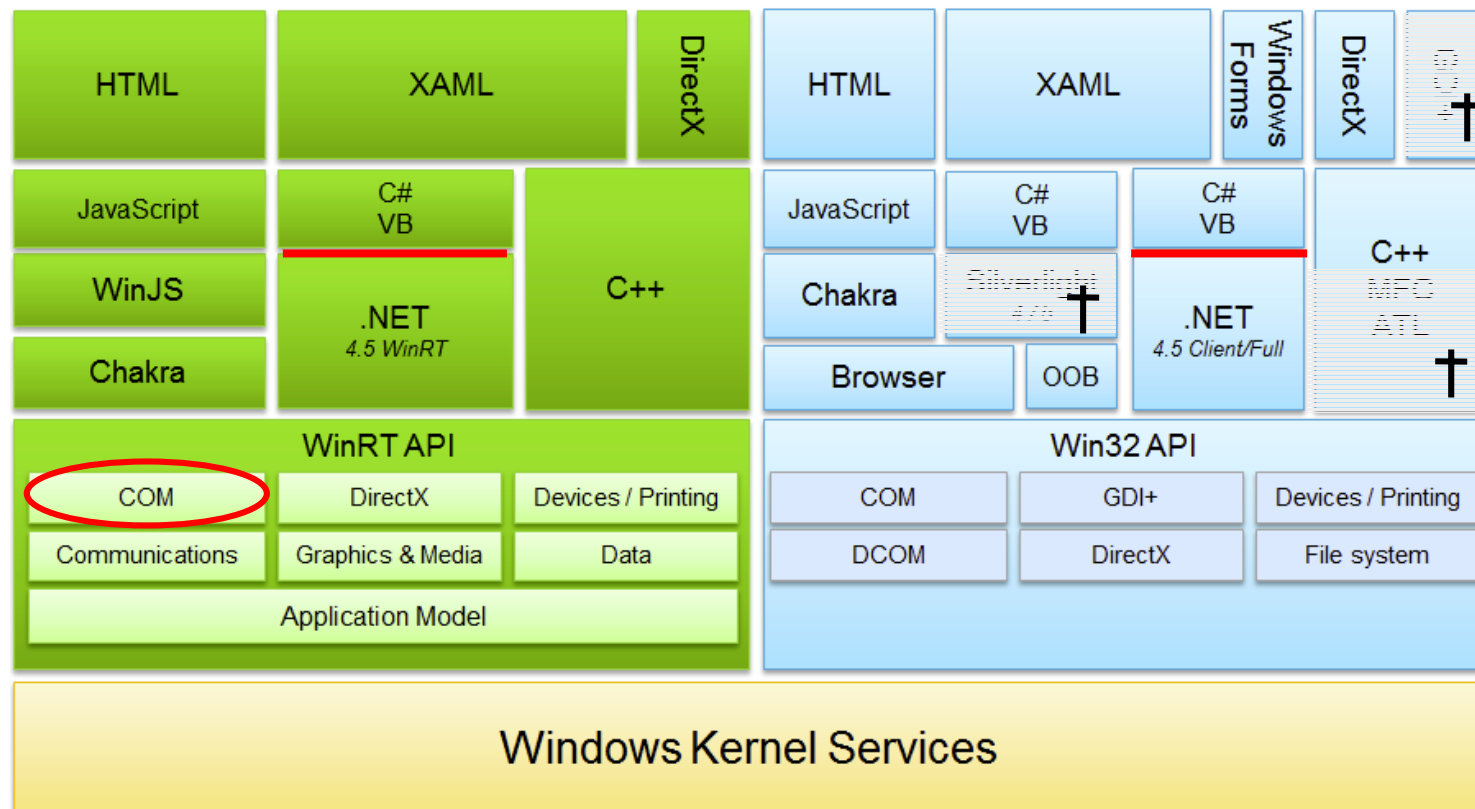
- 23. August 2013 gibt S. Ballmer seinen geplanten Rücktritt bekannt
- Microsoft wird dadurch auf einen Schlag 20 Mia \$ "wertvoller"
- Nachdem sich der Wert während seiner Zeit um 300 Mia \$ verringert hat
- Er "verdient" dadurch als Microsoft Aktionär 768 Mio \$



- 4. Februar 2014 wird Satya Nadella CEO
- 6 Mt später:
- gibt neue Strategie bekannt
- und entlässt 18'000 Mitarbeiter

Windows 8

- WinRT als die neue low-level Schnittstelle für das Betriebssystem
- (Für Entwicklung) nur für Windows 8 verfügbar



- WinRT umfasst ca. 1800 Klassen (Native Rückportierung .NET Libs auf C++)
 - deckt nicht den vollen Funktionsumfang von Win32 ab
- Windows 8 basierte Anwendungen können nur WinRT verwenden
- WinRT ist in C++ als COM Komponenten implementiert (nativer Code)
- COM ist das Microsoft Softwarekomponentenmodell aus dem Jahre 2002
 - HRESULT als Rückgabewerte
 - Fehler als COM Exception
 - COM Typensystem für Projektionen auf andere Sprachen
- In C# geschriebener Code wird auch unter W8 von der CLR ausgeführt
 - es steht aber nur Untermenge der .NET Klassenbibliothek zur Verfügung
- W8 Apps kennen kein Konzept der "Shared Libraries" DLLs
 - jede App muss eigenen Satz von DLLs mitbringen
- WinRT soll nicht als Zusatz zu anderen Betriebssystemen ausgeliefert werden
- Vermehrte asynchrone Aufrufe (liefern spezielle WinRT Typen zurück)

<http://www.it-visions.de/lserver/artikeldetails.aspx?b=6241>

C# die primäre Sprache für 

Inhalt Sprachteil (C# 2.0)

Teil 1

- Überblick
- Symbole
- Typen
- Ausdrücke

Teil 2

- Deklarationen
- Anweisungen
- Klassen und Structs

■ Literatur

- Hejlsberg: The C# Programming Language, Addison Wesley
- Mössenböck: Softwareentwicklung mit C# 2.0, dpunkt
- S.Robinson et al: Professional C#, Wrox Press
- Referenzinformation und Tutorials auf .Net-SDK-CD

Teil 3

- Vererbung
- Interfaces
- Delegates

Teil 4

- Ausnahmen
- Namensräume und Assemblies
- Attribute
- Threads
- XML-Kommentare

Teil 5

- Generics

Überblick: Merkmale von C#

Sehr ähnlich zu Java

70% Java, 10% C++, 5% Visual Basic, 15% neu

Wie in Java

- Objektorientierung (einf. Vererbung)
- Interfaces
- Exceptions
- Threads
- Namensräume (analog zu Paketen)
- Strenge Typenprüfung
- Garbage Collection
- ...



Wie in C++




- Operator Overloading
- Zeigerarithmetik in Unsafe Code
- Einige syntaktische Details
- Objekte am Stack (Structs)

Wie in VB




- Namensparameter (bei Attributen)

Neue Features in C#

Neu (Vergleich zu Java)

- Referenzparameter
- Blockmatrizen
- Enumerationen 
- Uniformes Typsystem
- Attribute 
- Systemnahes Programmieren
- Versionierung
- Generische Typen 
- Funktionale Programmierung (3.5)
- Dynamik Typing (3.5)
- await, async (4.5)

Syntactic Sugar

- Komponentenunterstützung
 - Properties
 - Events
- Delegates
- Indexer 
- foreach-Schleife 
- Boxing/Unboxing 
- ...

Language Equivalents

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vxgrflanguageequivalents.asp>

Einfachstes C#-Programm

File Hello.cs

```
namespace Sample {  
    using System;  
  
    class Hello {  
  
        static void Main() {  
            Console.WriteLine("Hello World");  
        }  
    }  
}
```

- enthält den Namensraum `Sample`
- benutzt Namensraum `System`
- Hauptmethode muss immer **Main** heissen
- Gibt auf Konsole aus
- Dateiname und Klassenname müssen *nicht* übereinstimmen.

Übersetzen (im Konsolenfenster; erzeugt Hello.exe)

`csc Hello.cs` (*Linux: msc Hello.cs*)

Ausführen

`Hello` (*Linux: mono Hello.exe*)

Programm aus 2 Dateien

Counter.cs

```
public class Counter {  
    int val = 0;  
    public void Add (int x) { val = val + x; }  
    public int Val () { return val; }  
}
```

Prog.cs

```
using System;  
  
public class Prog {  
  
    static void Main() {  
        Counter c = new Counter();  
        c.Add(3); c.Add(5);  
        Console.WriteLine("val = " + c.Val());  
    }  
}
```

Übersetzen

csc /target:exe Counter.cs Prog.cs
Linux: **msc** /target:exe Counter.cs Prog.cs

Ausführen

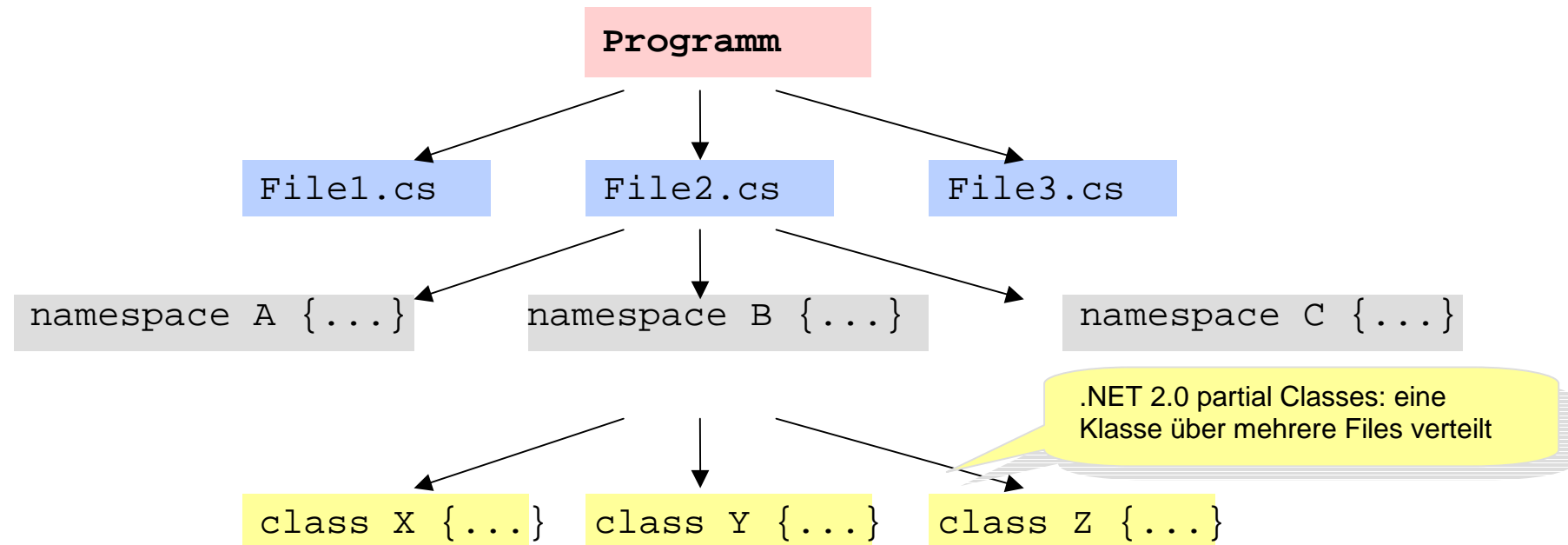
Windows: Prog
Linux: **mono** Prog.exe

Arbeiten mit DLLs

csc /t:library Counter.cs
=> erzeugt Counter.dll

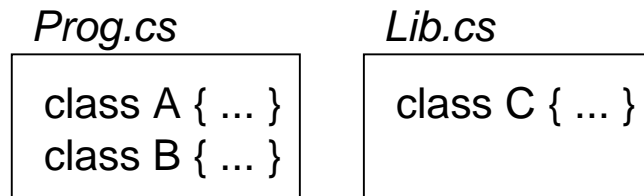
csc /r:Counter.dll Prog.cs
=> erzeugt Prog.exe

Gliederung von C#-Programmen



- Wenn kein Namensraum angegeben => Namenloser Standardnamensraum
- Namensraum kann auch Structs, Interfaces, Delegates, Enums enthalten
- Namensraum kann in verschiedenen Files "wiedereröffnet" werden
- Einfachster Fall: 1 File, 1 Klasse (wie Java)
 - Konvention: Namen der Hauptklasse mit File-Namen identisch.

Zur Laufzeit: Assemblies



csc Prog.cs,Lib.cs

Prog.exe



Lader

■ Assemblies sind kleinste Einheit für

- Auslieferung
- Versionierung
- Laden

Versionsnummer
Public Key
Schnittstellenbeschreibung
- Klassen
- Methoden
- Variablen
- Parameter
- Typen
- ...

■ ermöglicht:

- dynamisches Laden
- Versionsprüfung
- Reflection

Lexikalische Symbole

Symbole aus denen die Sprache C# aufgebaut ist.

Syntax

```
Name = (letter | '_' | '@') {letter | digit | '_'}
```

EBNF

= definiert

| alternativ

[] optional

{} 0..∞

■ Unicode!

■ Gross/Kleinschreibung ist signifikant

■ Können Unicode-Escapesequenz enthalten (z.B. \u03c0 für p)

Beispiele

```
someName  
sum_of3  
_10percent  
@while  
p  
\u03c0  
b\u0061ck
```

```
Der Name while  
Der Name p  
Der Name p  
Der Name back
```

Schlüsselwörter

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	...		

ca. 100 Schlüsselwörter in C# im Gegensatz zu ca. 70 Schlüsselwörtern in Java dürfen nicht für Variablen-Namen verwendet werden (@vorangestellt erlaubt)

Gross/Kleinschreibung

- Jeden Wortanfang gross schreiben (z.B. *ShowDialog*)
- Anfangsbuchstabe gross, ausser bei Variablen, Konstanten und Feldern, die man nicht von aussen sieht.

■ Konstanten	gross	<i>SIZE, MAX_VALUE</i>	wie in Java
■ lokale Variablen	klein	<i>i, top, sum</i>	
■ private Felder	klein	<i>data, lastElement, _feld</i>	Durch Gross/Kleinschreibung wird Sichtbarkeit ausgedrückt;
■ öffentliche Felder	gross	<i>Width, BufferLength</i>	
■ Properties	gross	<i>Length, FullName</i>	
■ Enum-Konstanten	gross	<i>Red, Blue</i>	
■ Methoden	gross	<i>Add, IndexOf</i>	
■ Typen	gross	<i>StringBuilder</i> (vordefinierte Typen klein: int, string)	
■ Namensräume	gross	<i>System, Collections</i>	

Erstes Wort

- void-Methoden sollten mit Verb beginnen (z.B. *DoCompute*)
- Alles andere sollte mit Substantiv beginnen (z.B. *size, IndexOf, Collections*)
- enum-Konstanten oder bool-Members können mit Adjektiv beginnen (*Red, Empty*)

Weiter

- Accronyme sollen "dumm" Gross-Klein geschrieben werden: XML -> Xml

Kommentare

Zeilenende-Kommentare

`// a comment`

Klammerkommentare

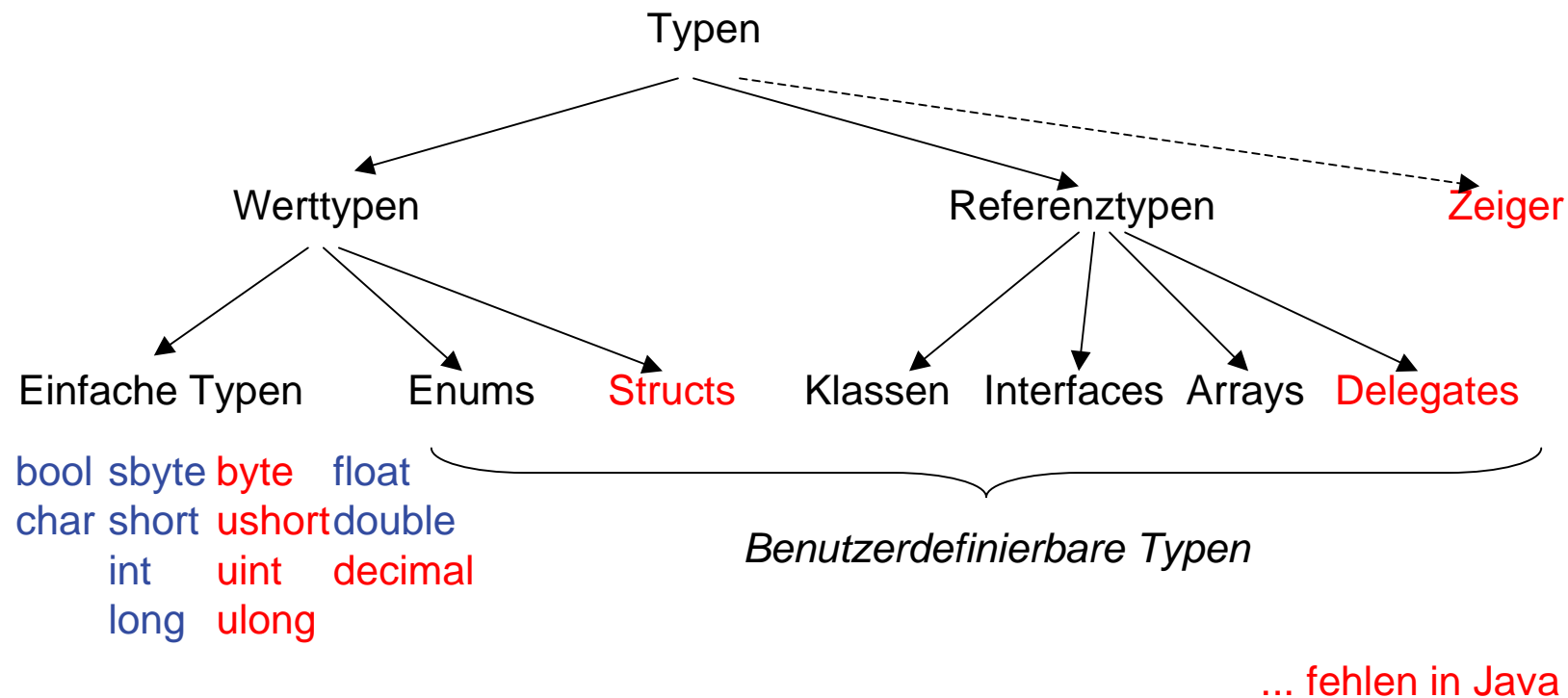
`/* a comment */`

Dürfen nicht geschachtelt werden

Dokumentationskommentare

`/// a documentation comment`

Typen



Alle Typen sind kompatibel mit *Object*

- können *Object*-Variablen zugewiesen werden
- verstehen *Object*-Operationen, z.B. 5.1.ToString()

Werttypen versus Referenztypen



	Werttypen	Referenztypen
Variable enthält	Wert	Referenz auf ein Objekt
gespeichert am	Stack	Heap
Initialisierung	0, false, '\0'	null
Zuweisung	kopiert Wert	kopiert Referenz
Beispiel	<div><pre>int i = 17;</pre><div><div>Stack</div><div><div>i</div><div>17</div></div><div><div>j</div><div>17</div></div></div><pre>int j = i;</pre></div>	<div><pre>class C {string name} C o = new C(); o.name = "Kurt"; C o2 = o;</pre><div><div>Heap</div><div><div>Kurt</div></div></div><div><div>o</div><div>o2</div></div></div>

Einfache Typen

Schlüsselwort		abgebildet auf	in Java Wertebereich
sbyte	System.SByte	byte	-128 .. 127
short	System.Int16	short	-32768 .. 32767
int	System.Int32	int	-2147483648 .. 2147483647
long	System.Int64	long	$-2^{63} .. 2^{63}-1$
byte	System.Byte	---	0 .. 255
ushort	System.UInt16	---	0 .. 65535
uint	System.UInt32	---	0 .. 4294967295
ulong	System.UInt64	---	0 .. $2^{64}-1$
float	System.Single	float	$\pm 1.5E-45 .. \pm 3.4E38$ (32 Bit)
double	System.Double	double	$\pm 5E-324 .. \pm 1.7E308$ (64 Bit)
decimal	System.Decimal	---	$\pm 1E-28 .. \pm 7.9E28$ (128 Bit)
bool	System.Boolean	boolean	true, false
char	System.Char	char	Unicode-Zeichen

= Byte in Java

Syntax

CharConstant = ' char '.

StringConstant = " {char} ".

char kann sein

beliebiges Zeichen ausser Ende-Hochkomma, Zeilenende oder \

Escape-Sequenz

\'	'
\"	"
\\	\
\0	0x0000
\a	0x0007 (alert)
\n	0x000a (new line)
\r	0x000d (carriage return)
\t	0x0009 (horizontal tab)

Unicode- oder Hex-Excape-Sequenz

\u0061	a
\x0061	a

... Zeichen und Zeichenketten

Beispiele für Escape-Sequenzen in Zeichenketten

"file \"C:\\sample.txt\\"

file "C:\\sample.txt"

"file \x0022C:\u005c sample.txt\x0022"

Wenn @ vor einer Zeichenkette steht

- gilt \ nicht als Metazeichen
- wird " durch Verdopplung ausgedrückt
- dürfen Zeilenumbrüche vorkommen

Beispiel

@ "file file

""C:\\sample.txt"" "C:\\sample.txt"



Syntax

DecConstant = digit {digit} {IntSuffix}.

HexConstant = "0x" hexDigit {hexDigit} {IntSuffix}.

IntSuffix = 'u' | 'U' | 'l' | 'L'.

Typ

ohne Suffix: kleinster aus int, uint, long, ulong

Suffix u, U: kleinster aus uint, ulong

Suffix l, L: kleinster aus long, ulong

Beispiele

17	int
9876543210	long
17L	long
17u	uint
0x3f	int
0x10000	long
0x3fL	long

Syntax (vereinfacht)

RealConstant = [Digits] ["." [Digits]] [Exp] [RealSuffix].
muss zumindest 1 Ziffer und entweder ".", Exp oder RealSuffix enthalten

Digits = digit {digit}.

Exp = ("e" | "E") ["+" | "-"] Digits.

RealSuffix = "f" | "F" | "d" | "D" | "m" | "M".

Typ

Suffix f, F: float (7 Ziffern)
Suffix d, D: double: (15 Ziffern) - default
Suffix m, M: decimal: (28 Ziffern)

Beispiele

3.14	double
1E-2	double
.1	double
10f	float

... Gleitkommazahlen: Typ decimal

Gleitkommazahl mit 128 Bit Genauigkeit

$$(-1)^s * c * 10^{-e}$$

$$s = 0 \text{ oder } 1$$

$$0 \leq c < 2^{96}$$

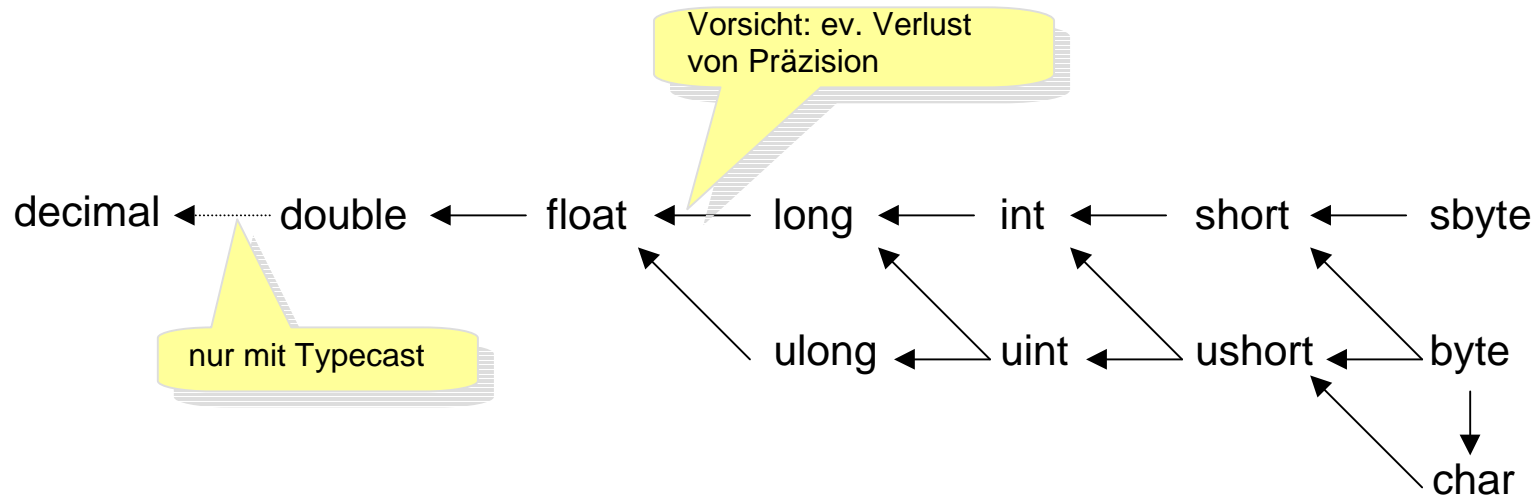
$$0 \leq e \leq 28$$

Für Berechnungen

- mit grossen Zahlen (28 Ziffern)
- mit exakter Dezimalgenauigkeit (z.B. $0.1 = 1 * 10^{-1}$)
- aber in der Ausführung langsam (nicht von HW unterstützt)

=> in der Finanzmathematik oft grosse/sehr genaue Zahlen benötigt,
z.B. für amerikanische Staatsschulden

Kompatibilität bei numerischen Typen



Folgende Zuweisungen sind ohne Typecast sicher

```
intVar = shortVar;  
intVar = charVar;  
floatVar = charVar;
```

aber

```
decimalVar = (decimal)doubleVar;
```

Enumerationen



Aufzählungstypen aus benannten Konstanten

Deklaration (auf Namespace-Ebene)

```
enum Color {Red, Blue, Green} // Werte: 0, 1, 2  
enum Access {Personal=1, Group=2, All=4}  
enum Access1 : byte {Personal=1, Group=2, All=4}
```

Beispiel

```
Color c = Color.Blue; // Enum-Konstanten müssen qualifiziert werden  
Access a = Access.Personal | Access.Group;  
// a enthält nun eine "Menge" von Werten  
if ((Access.Personal & a) != 0) Console.WriteLine("access granted");
```

Operationen mit Enumerationsen



Erlaubte Operationen

Vergleiche	<code>if (c == Color.Red) ...</code> <code>if (c > Color.Red && c <= Color.Green) ...</code>
<code>+, -</code>	<code>c = c + 2;</code>
<code>++, --</code>	<code>c++;</code>
<code>&</code>	<code>if ((c & Color.Red) == 0) ...</code>
<code> </code>	<code>c = c Color.Blue;</code>
<code>~</code>	<code>c = ~ Color.Red;</code>

Es wird nicht geprüft, ob der erlaubte Wertebereich über-/unterschritten wird.

Enumerationen sind nicht zuweisbar an *int* (ausser nach Type Cast).

■ Enumerationstypen erben alle Eigenschaften von *object* (*Equals*, *ToString*, ...)

■ Klasse ***System.Enum*** stellt Operationen auf Enumerationen bereit

z.B.

- `Enum.GetNames(typeof(Colors))` liefert die Namen der Enumerationswerte als Array
- `Colors myOrange = (Colors)Enum.Parse(typeof(Colors), "Red, Yellow");`

Eindimensionale Arrays

```
int[] a = new int[3];  
int[] b = new int[] {3, 4, 5};  
int[] c = {3, 4, 5};  
SomeClass[] d = new SomeClass[10];    // Array von Referenzen
```



Mehrdimensionale Arrays ("ausgefranst", jagged)

```
int[][] a = new int[2][];    // Array von Referenzen auf Arrays  
a[0] = {1, 2, 3, 4};  
a[1] = {4, 5, 6};
```



Mehrdimensionale Blockarrays (rechteckig)

```
int[,] a = new int[2, 3];    // Block-Matrix  
int[,] b = {{1, 2, 3}, {4, 5, 6}};    // Können direkt initialisiert werden  
int[,] c = new int[2, 4, 2];
```

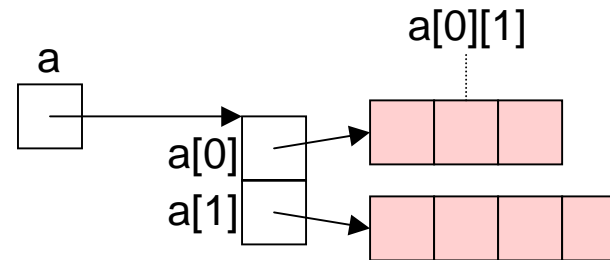
C#

Mehrdimensionale Arrays

Ausgefranst (wie in Java)

```
int[][] a = new int[2][];  
a[0] = new int[3];  
a[1] = new int[4];
```

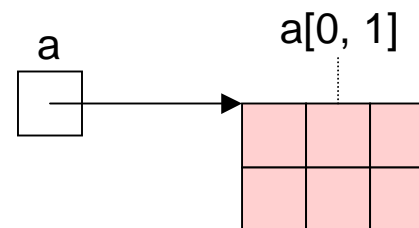
```
int x = a[0][1];
```



Rechteckig (kompakter, effizienterer Zugriff)

```
int[,] a = new int[2, 3];
```

```
int x = a[0, 1];
```



C#

Variabel lange Arrays: Listen

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

class Test {
    static void Main() {
        List<string> a = new List<string>();
        a.Add("Caesar");
        a.Add("Dora");
        a.Add("Anton");
        a.Sort();
        for (int i = 0; i < a.Count; i++)
            Console.WriteLine(a[i]);
    }
}
```

Ausgabe

Anton

Caesar

Dora

Assoziative Arrays: Hashtables

```
using System;
using System.Collections.Generic;
class Test {
    static void Main() {
        Dictionary<string,int> phone = new Dictionary<string,int>();
        phone["Karin"] = 7131;
        phone["Peter"] = 7130;
        phone["Wolfgang"] = 7132;
        string[] keys = new string[phone.Count];
        phone.Keys.CopyTo(keys,0);
        for (int i = 0; i < keys.Length; i ++) {
            string name = keys[i];
            Console.WriteLine("{0} = {1}",name, phone[name]);
        }
    }
}
```

foreach (String name in phone.keys)
 Console.WriteLine("{0} = {1}",
 name, phone[name]);

Ausgabe

Wolfgang = 7132

Peter = 7130

Karin = 7131

School of Engineering

© K. Rege, ZHAW

58 von 67

Indizierung beginnt bei 0

Arraylänge

```
int[] a = new int[3];  
Console.WriteLine(a.Length); // 3  
  
int[][] b = new int[3][];  
b[0] = new int[4];  
Console.WriteLine("{0}, {1}", b.Length, b[0].Length); // 3, 4  
  
int[,] c = new int[3, 4];  
Console.WriteLine(c.Length); // 12  
Console.WriteLine("{0}, {1}", c.GetLength(0), c.GetLength(1)); // 3, 4
```

System.Array enthält nützliche Array-Operationen

```
int[] a = {7, 2, 5};  
int[] b = new int[2];  
Array.Copy(a, b, 2); // kopiert a[0..1] nach b  
Array.Sort(b);  
...
```

String Methoden

Benutzbar als Standardtyp *string*

```
string s = "Alfonso";
```

Bemerkungen

- Strings sind nicht modifizierbar (dazu *StringBuilder*)
- Können mit + verkettet werden: "Don " + s
- Können indiziert werden: s[i]
- Längenprüfung: s.Length
- Referenztyp, daher Referenzsemantik in Zuweisungen
- aber Wertevergleich mit == und != : if (s == "Alfonso") ...
- Klasse *String* definiert viele nützliche Operationen:
CompareTo, CompareOrdinal, IndexOf, StartsWith, Substring, ...
- Konvertierung String in Zahl: Double.Parse, Int32.Parse, etc

Deklaration

```
class Rectangle {  
    private Point origin;  
    public int width, height;  
    public Rectangle() { origin = new Point(0,0); width = height = 0; }  
    public Rectangle (Point p, int w, int h) { origin = p; width = w; height = h; }  
    public void MoveTo (Point p) { origin = p; }  
}
```

Verwendung

```
Rectangle r = new Rectangle(new Point(10, 20), 5, 5);  
int area = r.width * r.height;  
r.MoveTo(new Point(3, 3));  
Rectangle r1 = r ; // Zeigerzuweisung
```

Bemerkungen

- Klassen sind *Referenztypen* (wie in Java)
Objekte werden am Heap angelegt
- Konstruktor-Aufruf erzeugt neues Objekt am Heap und initialisiert es
Parameterloser Konstruktor darf deklariert werden

■ .NET Beinhaltet:

- CLR, Klassenbibliothek, Web Forms, Web Services, ASP.NET
- und Sprachen C#, VB.NET.

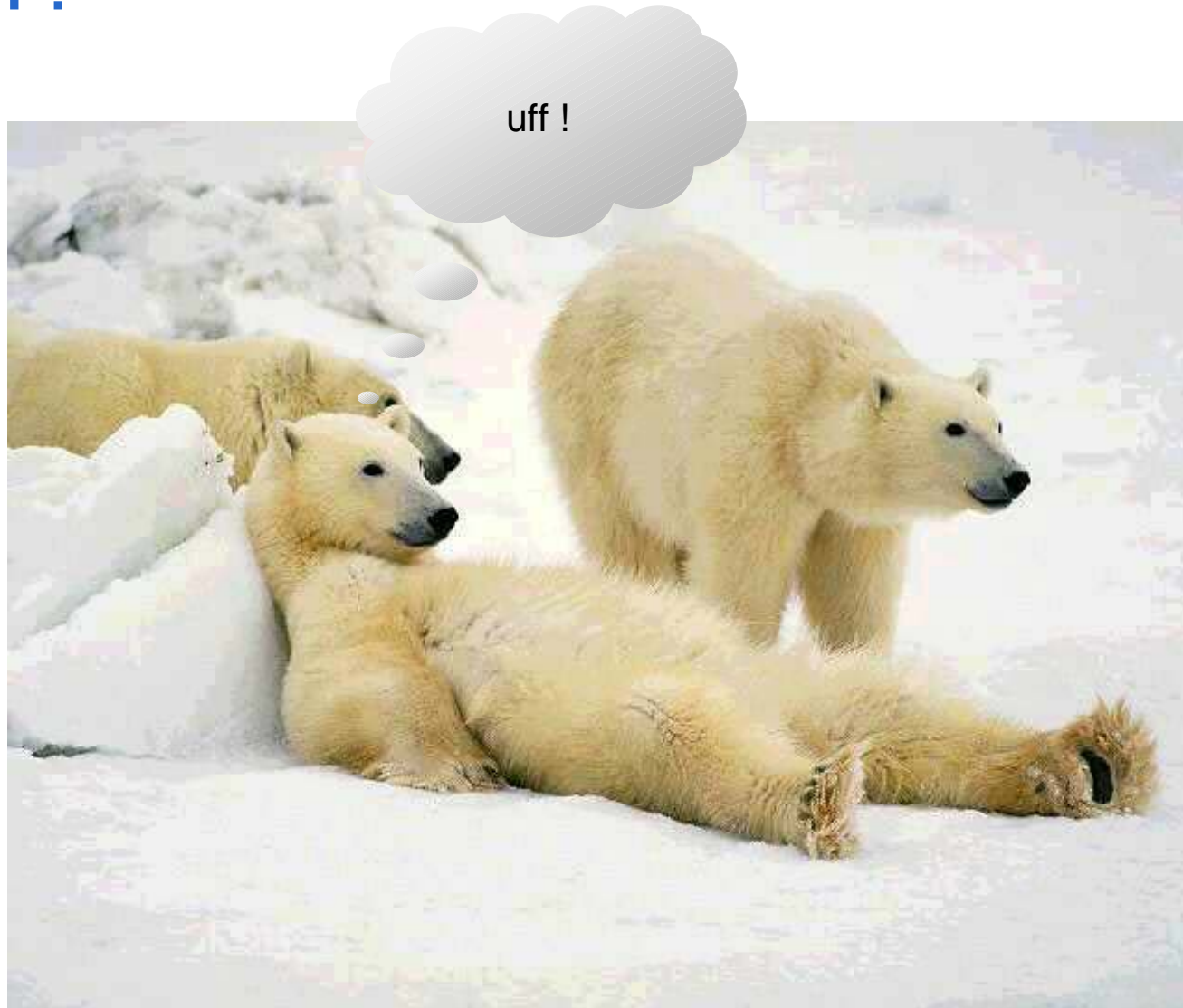
■ C# viele Gemeinsamkeiten zu Java (statt C# besser Java#)

- C-Syntax, OO Konzepte, Strenge Typenprüfung Garbage Collection, etc.
- aber auch neue Konzepte: Structs, Delegates(später)

■ Unterscheidung von Referenz und Werttypen

- automatische Umwandlung in Referenztyp (Boxing)

Fragen ?



Werkzeuge des .NET-Frameworks

Gratis Werkzeuge

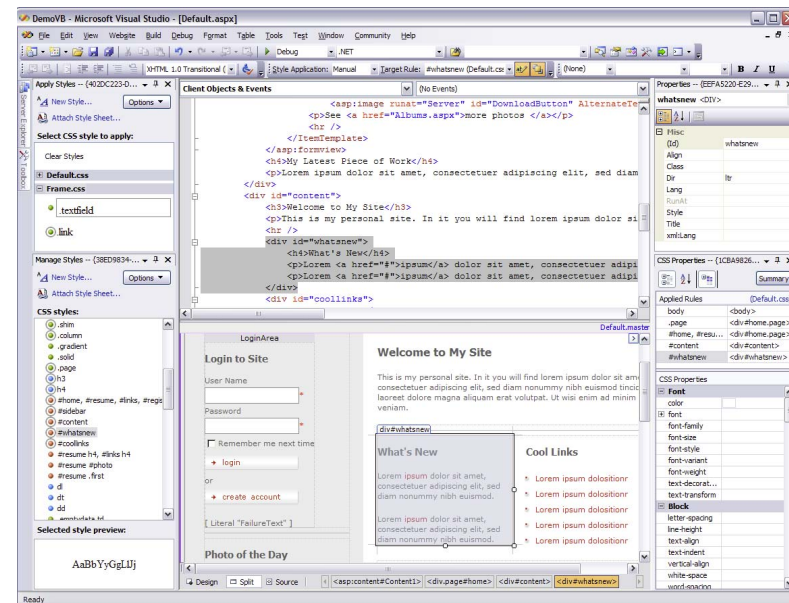
- C#-Compiler
- Visual Basic .NET-Compiler
- J#-Compiler
- JScript-Compiler
- IL-Disassembler
- Global Assembly Cache Utility
- CodeReflect
- Cassini
- Debugger
- .NET Framework Configuration Tool
- Weitere Tools
- SharpDevelop
- Visual Studio Express

- ⇒ **csc.exe / msc**
- ⇒ **vbc.exe**
- ⇒ **vjc.exe**
- ⇒ **jsc.exe**
- ⇒ **ildasm.exe**
- ⇒ **gacutil.exe**
- ⇒ <http://www.devextras.com>
- ⇒ **CassiniWebServer.exe**
- ⇒ **cordbg.exe**
- ⇒ **mmc mscorcfg.msc**
- ⇒ <http://www.it-visions.de/dotnet/tools.aspx>
- ⇒ <http://sharpdevelop.net/opensource/sd/>
- ⇒ <http://www.microsoft.com/express>

■ Für Studierende ZHAW gratis

- empfohlen VS 2010
- Microsoft Visual Studio 2010 Premium englische Version

■ <http://e5.onthehub.com/WebStore/ProductsByMajorVersionList.aspx?ws=5ae86fb1-836f-e011-971f-0030487d8897&vsro=8&JSEnabled=1>



■ <http://www.jetbrains.com/resharper/>

Sharp Develop

- Unter dem Link
 - <http://www.icsharpcode.net>

- schlank: (!)
 - ca. 1/100 von VS

- Version 3.2 noch ohne WPF (viel schneller)

- .NET SDK separat installieren

