



# C# Crashkurs für Java Programmierer

im Rahmen des  
Studienprojekts A 2007

# Vorwort

Autor: Sebastian Krysmanski

Inhalt: C# und .NET

Inhalt gilt für .NET 2.0 und höher

vieles existierte bereits in .NET 1.x, einiges ist in .NET 2.0 aber neu hinzugekommen

Es ist zu viel! Ergo: Hohes Tempo.

- Viele Sachen werden nur sehr kurz angesprochen. Ausführlichere Infos gibt es im Paper zu diesem Vortrag.
- Manche Sachen werden gar nicht angesprochen, da sie mit Java identisch sind oder es den Rahmen sprengen würde. Eine Aufzählung kommt am Ende.

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.

Außer bei 1. und 2. geht es bei allen anderen Punkten um C#.

# 1. Geschichtlicher Abriss

Datum	Ereignis
Juni 2000	Bill Gates stellt erstmals die .NET-„Vision“ vor.
Oktober 2000	C# wird zur Standardisierung eingereicht
Januar 2002	<i>.NET 1.0</i> und <i>Visual Studio .NET 2002</i> werden veröffentlicht
April 2003	<i>.NET 1.1</i> und <i>Visual Studio 2003</i> werden veröffentlicht
November 2005	<i>.NET 2.0</i> und <i>Visual Studio 2005</i> werden veröffentlicht
November 2006	<i>.NET 3.0</i> wird veröffentlicht
November 2007	<i>.NET 3.5</i> und <i>Visual Studio 2008</i> werden veröffentlicht

Im Vergleich:

- 1996 - Java 1.0
- 1998 – Java 1.2 bereits mit Swing

# 1. Was ist .NET?

.NET ist nicht nur ein Framework, sondern ein Konzept, das aus folgenden Teilen besteht:

- Common Language Specification (CLS)  
inkl. Common Type System (CTS)
- Common Language Runtime (CLR)  
inkl. Common Intermediate Language (CIL)
- .NET Framework

.NET-Framework: Sammlung von Klassen; aber was sind diese Common ...?

# 1. Common Language Specification

Java-Anwendung: In Java geschrieben

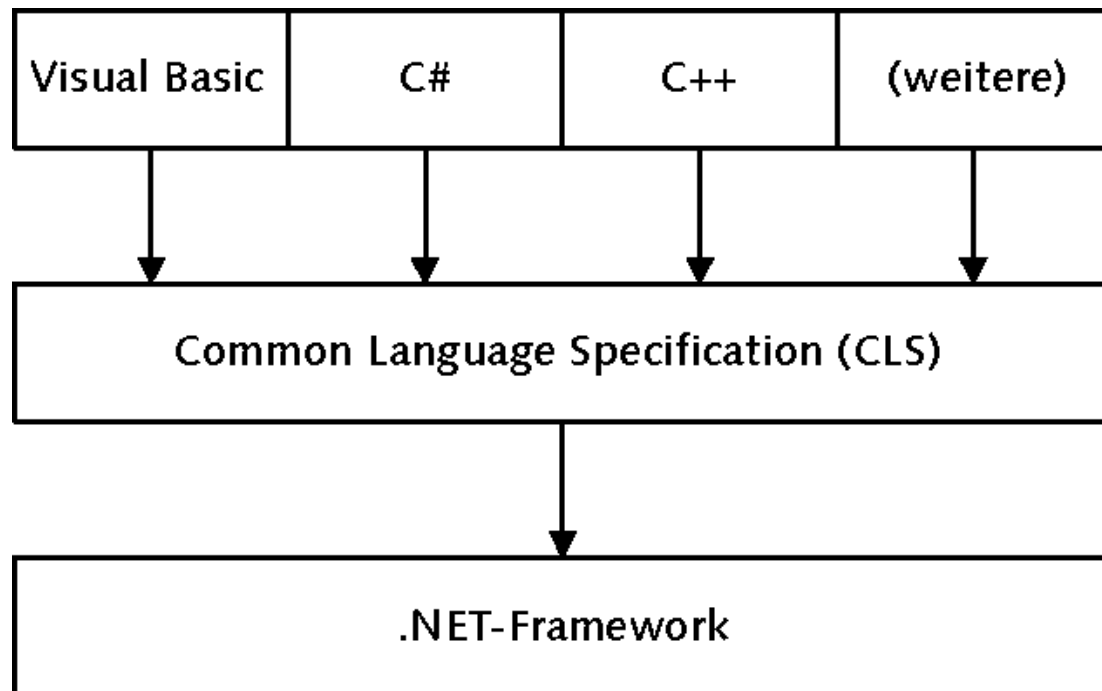
.NET-Anwendung: Für .NET geschrieben;  
Programmiersprache: verschiedene (z.B. C#, J#,  
C++/CLI, Visual Basic .NET)

D.h.: Eine Klasse, die in C# geschrieben ist, kann von einer Klasse in Visual Basic 2005 beerbt werden. Beide Klassen können Daten miteinander austauschen und Ausnahmen weiterreichen.

Es gibt unter .NET keine bevorzugte Programmiersprache. Vorteil: Jeder kann in seiner „Lieblingsprogrammiersprache“ programmieren.

# 1. Common Language Specification

CLS bestimmt, wie die Programmiersprache mit dem .NET-Framework bzw. allgemein mit .NET-Bibliotheken umgehen muss.



# 1. Common Type System

„Hilft“ der CLS:

- Definiert gültige Typen für die öffentlichen Schnittstellen von .NET-Bibliotheken. (z.B. sind vorzeichenlose Typen wie `unsigned int` – nur an öffentlichen Schnittstellen – verboten)
- Definiert Größe der Typen (z.B. dass `long` immer 8 Byte groß ist)

# 1. Common Language Runtime

Ist die *Virtual Machine* von .NET

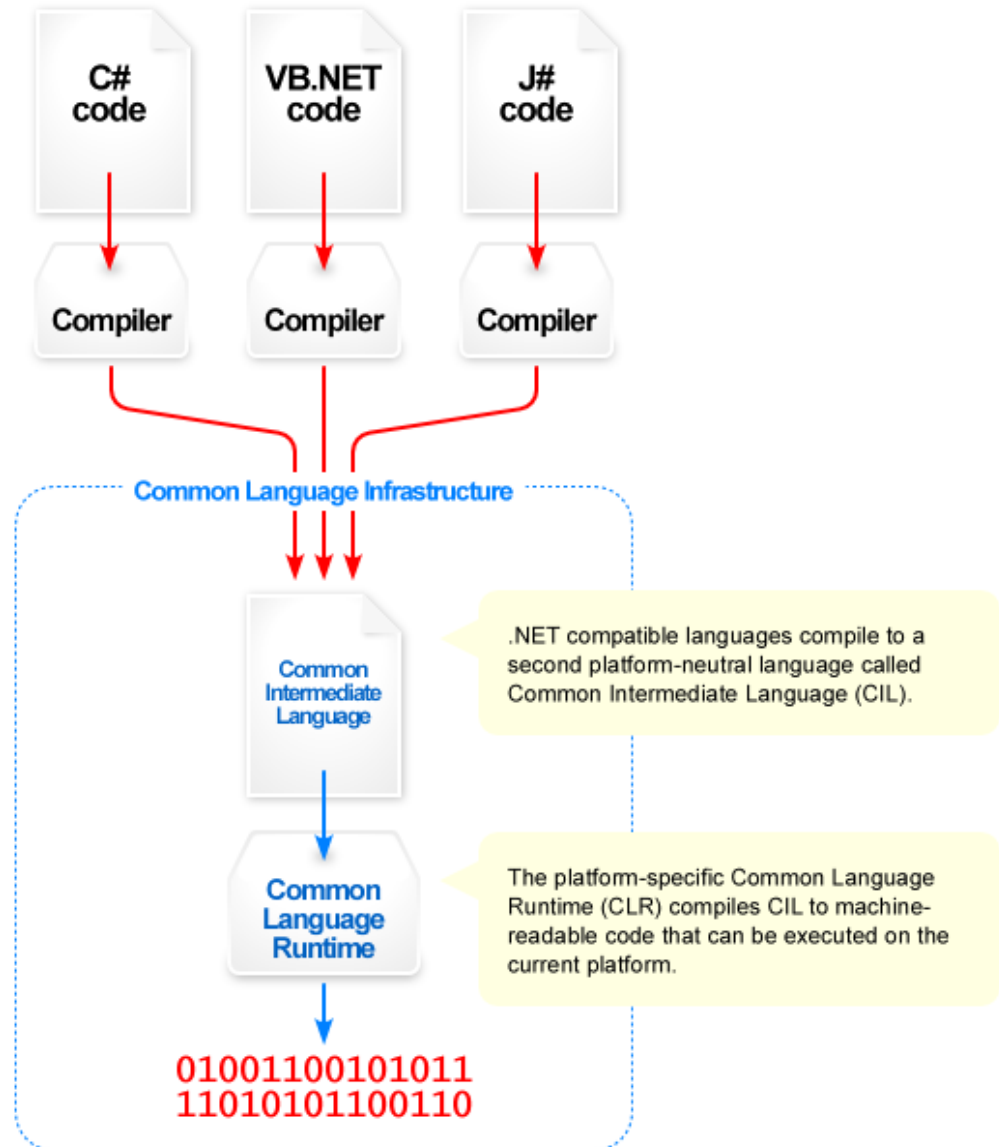
D.h. .NET-Anwendungen sind plattform-unabhängig –  
geschrieben in CIL

CIL: Objekt-orientierte Assemblersprache

```
.method public static void Main() cil managed
{
    .entrypoint
    .maxstack 1
    ldstr "Hallo Welt!"
    call void [mscorlib]System.Console::WriteLine(string)
    ret
}
```

# 1. Common Language Runtime

C++/CLI nimmt Sonderstellung ein: wird (meistens) direkt in Maschinencode kompiliert (d.h. CIL wird nicht verwendet)



# 1. Zusammenfassung .NET

- .NET ist programmiersprachen-unabhängig
- .NET wird (meistens) in plattform-unabhängigen Code übersetzt

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.

## 2. Die IDE: Visual Studio



Profession Edition:  
knapp 950 Euro



Standard Edition:  
knapp 350 Euro



Express Edition:  
kostenlos

Unterschiede zwischen den Versionen und Links im Paper

- Aktuelle Version auf deutsch: Visual Studio 2005
- Aktuelle Version auf englisch: Visual Studio 2008 (aber irgendwie nur beta)

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.

### 3. Hello World!

 DEMO!

Kleiner Hinweis: Sichtbarkeit bei der `Main`-Methode egal. (Ist standardmäßig `private`.)

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.

# 4. Anzahl der Klassen

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

namespace Hello_World
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    // weitere Namespaces, Klassen, Enums, ...
}

// weitere Namespaces, Klassen, Enums, ...
```

Unterschied zu Java: Beliebig viele Klassen usw. pro Datei definierbar.

# 4. Namespace

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

namespace Hello_World
{
    class Program { ... }
}
```

`namespace`: wie `package` unter Java, nur nicht an Verzeichnisstruktur oder Dateinamen gebunden

`using`: Klassen ohne Voll-Qualifizierten-Namen verwenden.

 DEMO!

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.



# 5. Gemeinsamkeiten mit Java

1. Klassenaufbau
2. Sichtbarkeitsmodifizierer
3. Polymorphie
4. Exceptions
5. Nice to know

# 5.1 Klassenaufbau

Klassenaufbau ist genau wie in Java:

```
class Beispiel
{
    private int m_iEineVariable;
    private string m_strNochEineVariable = "Hallo";

    public Beispiel(int p_iWert)
    {
        m_iEineVariable = p_iWert;
    }

    public int getWert()
    {
        return this.m_iEineVariable;
    }
}
```

# 5.1 Klassenaufbau

Der Aufruf von anderen Konstruktoren ist ein wenig anders aufgebaut:

- Konstruktor der gleichen Klasse:

```
public Circle(double Radius, int X, int Y) : this(Radius) { }
```

- Konstruktor der Basis-Klasse:

```
public Circle(double Radius, int X, int Y) : base(Radius) { }
```

- Statischer Konstruktor (keine Sichtbarkeit, keine Parameter):

```
static Circle() { ... }
```

## 5.2 Sichtbarkeitsmodifizierer

Gleiche Sichtbarkeitsmodifizierer wie in Java, mit Ausnahme von `package`: Ersetzt durch `internal`.

- `internal`: Nur innerhalb des Programms bzw. der Bibliothek sichtbar. (ansonsten wie `public`)
- kann mit `protected` kombiniert werden (zu `protected internal`)
- Es gibt keinen `namespace`-Modifizierer (d.h. nur sichtbar innerhalb des Namespaces; wie `package`)

# 5.3 Polymorphie

Ableitung von Basisklasse bzw. Implementierung von Interface  
nur mit `:` (nicht mit `extend` bzw. `implements`):

```
public class GraphicCircle : BasisKlasse, IInterface1, IInterface2
```

# 5.3 Polymorphie

**Wichtig: Das Überladen von Methoden funktioniert in C# anders als in Java.**

Beispiel:

```
class A {  
    public void Methode() {  
        Console.WriteLine("A.Methode()");  
    }  
}
```

```
class B : A {  
    public void Methode() {  
        Console.WriteLine("B.Methode()");  
    }  
}
```

```
A obj = new B();  
obj.Methode(); // Liefert unter Java „B.Methode()“,  
               // unter C# aber „A.Methode()“.
```

# 5.3 Polymorphie

D.h.: Methoden werden standardmäßig immer an Klassennamen gebunden.

Um das Java-Verhalten wiederherzustellen, benötigt man `virtual` und `override`:

```
class A {  
    public virtual void Methode() {  
        Console.WriteLine("A.Methode()");  
    }  
}
```

```
class B : A {  
    public override void Methode() {  
        Console.WriteLine("B.Methode()");  
    }  
}
```

```
A obj = new B();  
obj.Methode(); //Liefert unter Java und unter C# „B.Methode()“
```

## 5.3 Polymorphie


Unterschied zu Java:

In Basis-Klasse muss festgelegt werden, welche Methoden "*ersetzt*" werden dürfen. In Java muss man mit "final" festlegen, bei welchen Methoden das *nicht erlaubt* ist.

# 5.4 Exceptions

Exception-Handling funktioniert im Großen und Ganzen wie in Java.

Zwei Ausnahmen:

1. Eigene Exceptions sollten von der Klasse `ApplicationException` und nicht von der Klasse `Exception` abgeleitet werden.
2. Exceptions müssen nicht behandelt werden. (D.h. es gibt kein `throws`)  **DEMO!**

# 5.5 Nice to know

Diese Themen sind im Paper erklärt.

1. **Mehrdimensionale Arrays**  
andere Syntax, Array-Arrays
2. **Typen-Operatoren**  
`is`, `as`, `typeof` und `GetType()`
3. **Primitive Datentypen**  
vorzeichenlose Typen und `decimal`
4. **Schleifen mit `foreach`**
5. **Generics**  
auch mit primitiven Datentypen, `default()`, `new()` und Constraints

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.



# 6. Neuerungen gegenüber Java

1. Verbessertes String-Handling
2. Partielle Klassen
3. Eigenschaften
4. Funktionszeiger (Delegate)
5. Nice to know

# 6.1 Verbessertes String-Handling

Zwei Verbesserungen gegenüber Java:

1. Der Vergleichsoperator funktioniert (endlich):

```
if (string1 == string2) { ... }
```

2. `switch` funktioniert auch mit Strings:

```
switch (string1)
{
case "hallo":
    break;
}
```

## 6.2 Partielle Klassen

Klassen usw. können über mehrere Dateien aufgespalten werden: `partial`

```
// in der Quellcodedatei 'ClassA1.cs'  
partial class ClassA {  
    public int intX;  
    ...  
}
```

```
// in der Quellcodedatei 'ClassA2.cs'  
partial class ClassA {  
    public int intY;  
    ...  
}
```

# 6.3 Eigenschaften

Möglichkeit, Setter- und Getter-Methode direkt in der Variablendeklaration anzugeben.

Aus:

```
public class Circle {  
    public double Radius;  
}
```

# 6.3 Eigenschaften

wird:

```
public class Circle {
    private double radius = 0;
    // Eigenschaftsmethode
    public double Radius {
        get { return radius; }
        set {
            if (value >= 0)
                // bei einer Wertzuweisung wird der übergebene Wert im
                // impliziten Parameter value entgegengenommen
                radius = value;
            else
                Console.Write("Unzulässiger Wert.");
        }
    }
    ...
}
```

# 6.3 Eigenschaften

Aufruf einer der „Accessoren“:

- **get:** `Variable = meinKreis.Radius;`
- **set:** `meinKreis.Radius = Variable;`

## 6.3 Eigenschaften

Vorteile gegenüber der "normalen" Setter- und Getter-Methoden:

- Spart Schreibarbeit (in der Klassenbeschreibung und beim Verwenden, da nur der Zuweisungsoperator verwendet werden muss)
- Setter und Getter bleiben bei der Variable
- Nachträgliches Hinzufügen eines Getters oder Setters zieht keine Änderungen am Code beim Verwenden nach sich

## 6.4 Funktionszeiger (Delegate)

„Speichert“ eine Methode in einer Variable.

- `public int Addition(int x, int y) { return x + y; }`
- `public delegate int ProcessOperation(int x, int y);`
- `ProcessOperation process = Addition;`
- `int Wert = process(5, 6);`

Praktisch z.B. um eine Methode für eine Operation zurückzugeben (Beispiel: Taschenrechner).

Noch im Paper: anonymer Delegate.

```
MyDelegate del = delegate(int value1, int value2) { ... }
```

# 6.5 Nice to know

Diese Themen sind im Paper erklärt.

1. **Event-Handling**  
Schlüsselwort `event`, Hinzufügen von Event-Handlern und das Auslösen des Events
2. **Operator-Überladung**
3. **Indexer**  
Überladen der Array-Klammern [ ]
4. **Parameter mit `ref` und `out`**  
Alternative für Rückgabewerte über Parameter

# Gliederung

1. Was ist .NET?
2. Die IDE: Visual Studio
3. Hello World!
4. Standard-Struktur einer C#-Datei
5. Gemeinsamkeiten mit Java
6. Neuerungen gegenüber Java
7. Was noch fehlt.



# 7. Was noch fehlt, ...

...aber im Paper vorhanden ist:

- Assemblies: Struktur der kompilierten Dateien
- Strukturen (`struct`, billige Klassen)

# 7. Was noch fehlt, ...

...aber *nicht* im Paper zu finden ist:

- Aufzählungstypen (`enum`; etwas mächtiger als in Java)
- Destruktoren und Dispose-Methoden
- Überladene Typecast-Operatoren
- Reflection
- Unsicherer Code (`unsafe`; Pointerarithmetik)
- Collections
- Multicast-Delegate (Delegate-Arrays)
- Attribute (wie Annotations in Java)
- und mehr: Kapitel 8 – 27 (aus Visual C# 2005)



Ende des Vortrags

**Fragen?**