Grundlagen in Visual Studio MFC und .net

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt

Überblick

- **C#** ist das Äquivalent zu Java, aber auch eine Weiterentwicklung der Sprache C/C++, keine Pointer, (".", "::" und "->") nun .
- Plattformunabhängig, eher im Sinne MS Desktop, PDA, Web

- **.net**
- Common language runtime (CLR), Garbage Collection
- Common language subset (CLS), gemeinsame Sprachbasis. Es kann jede Sprache verwendet werden. Standardvokabular (Menge von Befehlen)
- Microsoft intermediate language (MSIL), Zwischensprache
- Just in Time Compiler (JIT)
- Windows Forms (Swing)
- ASP.net (Web Services)
- ADO.net (Datenbanken)
- XML, SOAP, UDDI (Kommunikation zw. den Komponenten

Visual Studio, C# und .net

- Entwickler: Andreas Hejlsberg
- Heilsberg hatte Turbo Pascal und Delphi mit entwickelt
- Absprung nach Redmond
- Entwicklung des Programmpaket .net / C#
- Philosophie weitgehend identisch
  - Drag & Drop der GUI
  - Propertyfenster mit Property-Methoden kein normales get/set
  - Antwort auf Java, Plattform unabhängig
  - Sprache C#
  - Aktuell Framework 3,5
  - Common Language Runtime (CLR)
  - Unterschied: Übersetzung, kein Intepreter,
  - Übersetzen VOR oder während der Ausführung

Visual Studio, C# und .net

- Weitere Eigenschaften
  - Unterstützt viele Programmiersprachen (C#, VB, Delphi)
  - C#, VB, Delphi, C++
  - Umwandlung in einem Zwischencode
  - Sehr viele GUI-Elemente, bis zu Listview / Grid
  - Datenbank-Anbindung
  - Web-Server
  - Delegates statt Funktionspointer
  - Operator Überladen
  - Eigenschaften (readX, writeX, get/set)
  - Alles Objekte
  - Anweisungen weitgehend identisch zu Java / C++
  - keine Header-Dateien
  - statt implement verwendet man using
  - Mehr Datentypen
  - Arrays mittels Blockstruktur, anders als in Java

Visual Studio, C# und .net

- Datentypen in C#
  - byte vorzeichenlos, 0 bis 255
  - sbyte vorzeichenbehaftet, -128 bis +127
  - short vorzeichenbehaftet, -32768 bis +32767
  - ushort vorzeichenlos, 0 bis 65535
  - int vorzeichenbehaftet, -2.147.483.648 bis + 2.147.483.648
  - uint vorzeichenlos, 0 bis 4.294,967,295
  - long vorzeichenbehaftet -263 bis 263-1
  - ulong vorzeichenlos 0 bis bis 264-1
  - single 32 Bit Gleitkommazahl, 7 Stellen
  - double 64 Bit Gleitkommazahl, 16 Stellen
  - bool boolscher Wert
  - char Zeichen
  - decimal 96 Bit Dezimalwert
  - string readonly Zeichenfolge

Visual Studio, C# und .net

- Hello World

```
using System;
namespace bsp1
{

   public class Programm  {
     public static void Main(string[] args)    {
       Console.WriteLine("Hello {0} im Jahr {1}", "ix", 2008);
       Console.ReadLine();
     } // Programm
   }// bsp1

}
```

- Format: printf

Visual Studio, C# und .net

- using System // Package
- namespace // definiert einen Prefix für Definitionen
- Namespace System // int32 statt System.int32

- Console ist ein Unterbereich von System
- Console.WriteLine(…) // statt System.Console.WriteLine

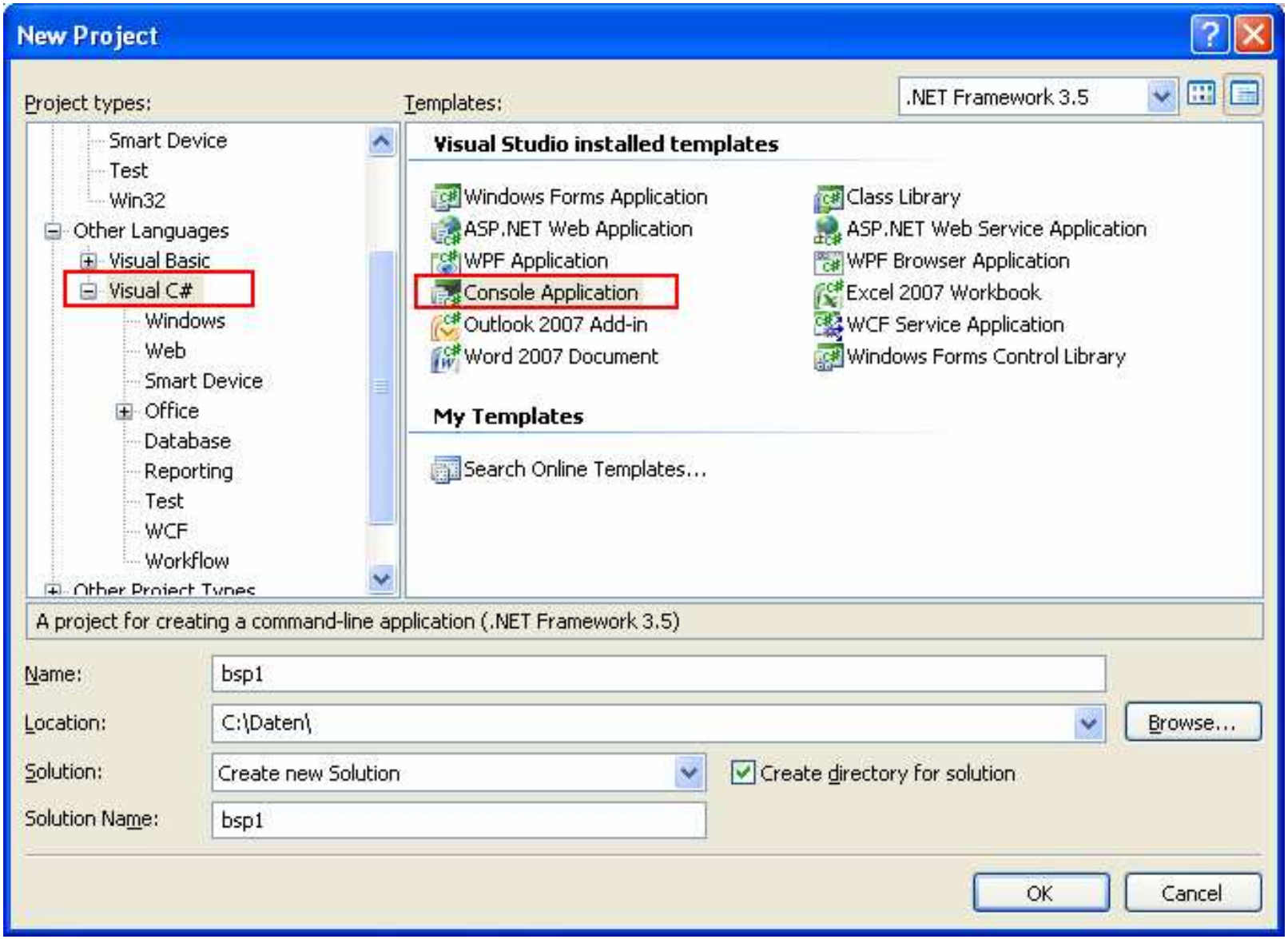
- Console.ReadLine(); // wartet auf Eingabe

Visual Studio, C# und .net

Beispiel 1

- File New Project
- Visual C#
- Console Application
- Einfügen der beiden Zeilen
  - Console.WriteLine("Hello {0} im Jahr {1}", "ix", 2008);
  - Console.ReadLine();
- Taste F5 startet das Programm

Visual Studio, C# und .net



Zeichenketten-Operationen

Methode	**Beschreibung**
Length	Länge eines Strings
Clone	Liefert Verweis des Strings
Compare	Vergleicht zwei String (true, false)
CompareTo	Vergleicht zwei String (-1, 0, +1)
Concat	Verbindet zwei Strings
Copy	Kopieren des Strings
CopyTo	Kopieren von Zeichen
EndsWidth	Vergleich mit Ende des String (\)
Equals	Haben zwei String denselben Inhalt
Format	Printf
GetEnumerator	Funktion mit Parameter des String Durchlaufen des String

Zeichenketten-Operationen

Methode	**Beschreibung**
IndexOf	Sucht das erste Vorkommen
IndexOfAny	Liefert alle Vorkommen in einer Liste
Insert	Zeichen einfügen
Join	Die Methode fügt zwischen je zwei Elementen eines Strings-Arrays einen angegebenen trennenden Strings ein und liefert das Ergebnis a,b,c $\Rightarrow$ a_b_c
LastIndexOf	Sucht das letzte Vorkommen
LastIndexOfAny	Liefert alle Vorkommen in einer Liste Umgekehrte Reihenfolge
PadLeft	Rechtsbündige Ausrichtung, Blanks links
PadRight	Linksbündige Ausrichtung, Blanks rechts
Remove	Löschen von Zeichen
Replace	Ersetzen von Zeichen

Zeichenketten-Operationen

Methode	**Beschreibung**
Split	Aufteilen (Parsen oder \)
StartsWith	Prüft die ersten Zeichen auf Übereinstimmung
SubString	Liefert einen Teilstring
ToCharArray	Umwandlung von String in Char-Array
ToLower	Kleinbuchstaben
ToUpper	Großbuchstaben
Trim	Löschen von Leerzeichen am Anfang und am Ende
TrimEnd	Löschen von Leerzeichen am Ende
TrimStart	Löschen von Leerzeichen am Anfang

C# Sprache

Weitgehend identisch mit Java,

- If-else-Anweisung identisch
- Switch-case-Anweisung unterschiedlich
- Liefert eine Fehlermeldung, break fehlt

```
switch (i)  {
        case 0: erg = a + b;
        case 1: erg = a - b;
        case 2: erg = a * b;

    }
```

- Nun okay, mit leerer Anweisung

```
switch (i)  {
        case 0:
        case 1: erg = a - b; // korrekt ohne Anweisung
        case 2: erg = a * b;
    }
```

C# Sprache

- For-Schleife — identisch

- foreach-Schleifen und Iteration — neu, aber ähnlich in Java

```
int [] array = { 1,2,3,4,5 };
foreach (int i in array) {
    Console.WriteLine(i);
}
```

- While-Schleife — identisch
- Do-While-Schleife — identisch

C# Sprache

- Enumerationen neu
- Automatische Konstanten mit ganzzahligem Wert

```
enum FB {
    AI,
    VW,
    W
}
enum FB {
    AI=0x0000FF,
    VW=0xFF0000,
    W=0x00FF00
}
```

```
FB f;
f = FB.AI;
int i = (int)f;
i = (int)FB.W;
```

```
static void testfor() {
  foreach (int j  in Enum.GetValues(typeof(FB)))  {
     Console.WriteLine("Fachbereich {0} {1}",
       Enum.GetName(typeof(FB),j),j);
  }
}  // Ausgabe ???
```

- FB.IsDefined(typeof(FB), i)

C# Sprache

- Exception fast identisch
- public double calc(double i) {

```
    try {
        double j = sqrt(i);
    } catch (MyException me)
    {
        Console.WriteLine(me.m_msg);
    } catch {
        /* alle restlichen  Exceptions */
    } finally {
        /* Aufraeumen */
    }
} // calc
```

C# Sprache

- Formatierte Ausgabe:

```
double d = 1.0 / 3.0;
// #  leer oder zahl
// 0  0 oder zahl  besser ###0.00 slas ###.##
Console.WriteLine("d: {0:##0.00}", d);
```

Format-Code	**Beschreibung**
{0} {1} {2}	Zahl mit mehreren Nachkommastellen
{0:##0.00}	Zahl mit zwei Nachkommastellen Dezimaltrennzeichen ist ein Punkt
{0:p2}	Zahl mit zwei Nachkommastellen Dezimaltrennzeichen ist ein Komma
{0,7:c}	Ausrichtung
("").PadRight(24, '-')	Ergibt 24 waagerechte Striche

C# Sprache

- Parameter-Übergabe: 4 Typen

- a) Parameter per Value

- b) Parameter per Referenz (ref)
  Kann sofort gelesen oder geschrieben werden

- c) Parameter für die Ausgabe (out)
  Muss erst gesetzt werden, dann darf gelesen werden

- d) Parameter für Arrays (params)
  muss immer am Ende der Liste stehen
  nur ein einfaches Feld, keine Matrix
  auch mit drei Zahlen möglich

C# Sprache

```
static void add2(int a, int b, out int erg)    {
    erg = a + b;
}
static void add3(int a, ref int b)   {
   b = a + b;
}

int a = 11;
int b = 22;
int c;
add2(a, b,out c);      // out signalisiert Ausgabeparameter
Console.WriteLine("a {0}   b {1}  c{2}", a, b, c);


a = 11;
b = 22;
add3(a, ref b);        // ref signalisiert Referenzparameter
Console.WriteLine("a {0}   b {1}", a, b);
```

C# Sprache: Parameter mit params

```
void ShowNumbers (params int[] numbers) {
        foreach (int x in numbers) {
                Console.Write (x+" ");
        }
        Console.WriteLine();
}

int[] x = {1, 2, 3};
ShowNumbers (x);
ShowNumbers (4, 5);
ShowNumbers (4, 5,6,7,8);
```

C# Sprache: struct

- Vorteile
  - Zusammenfassen von Daten
  - Keine Klasse
  - Als Parameter für Methoden geeignet

```
public struct Student
{
    public string name;
    public int matrnr;
    public int unummer;
}
```

C# Sprache: struct

```
static void Main(string[] args)
{
    Student s1, s2;
    s1.name = "Bender";
    s1.matrnr = 1234;
    s1.unummer = 23233;
    s2 = s1;  // kopy
    Console.WriteLine("Studentendaten");
    Console.WriteLine("Name: {0}  Matrnr: {1}  UNummer: U{2}",
            s2.name, s2.matrnr, s2.unummer);
    Console.ReadLine();
}
```

C# Sprache: struct

```
static void print(Student std)
{
        Console.WriteLine("Studentendaten");
        Console.WriteLine("Name: {0}  Matrnr: {1}  UNummer: U{2}",
                std.name, std.matrnr, std.unummer);
         Console.ReadLine();
}
```

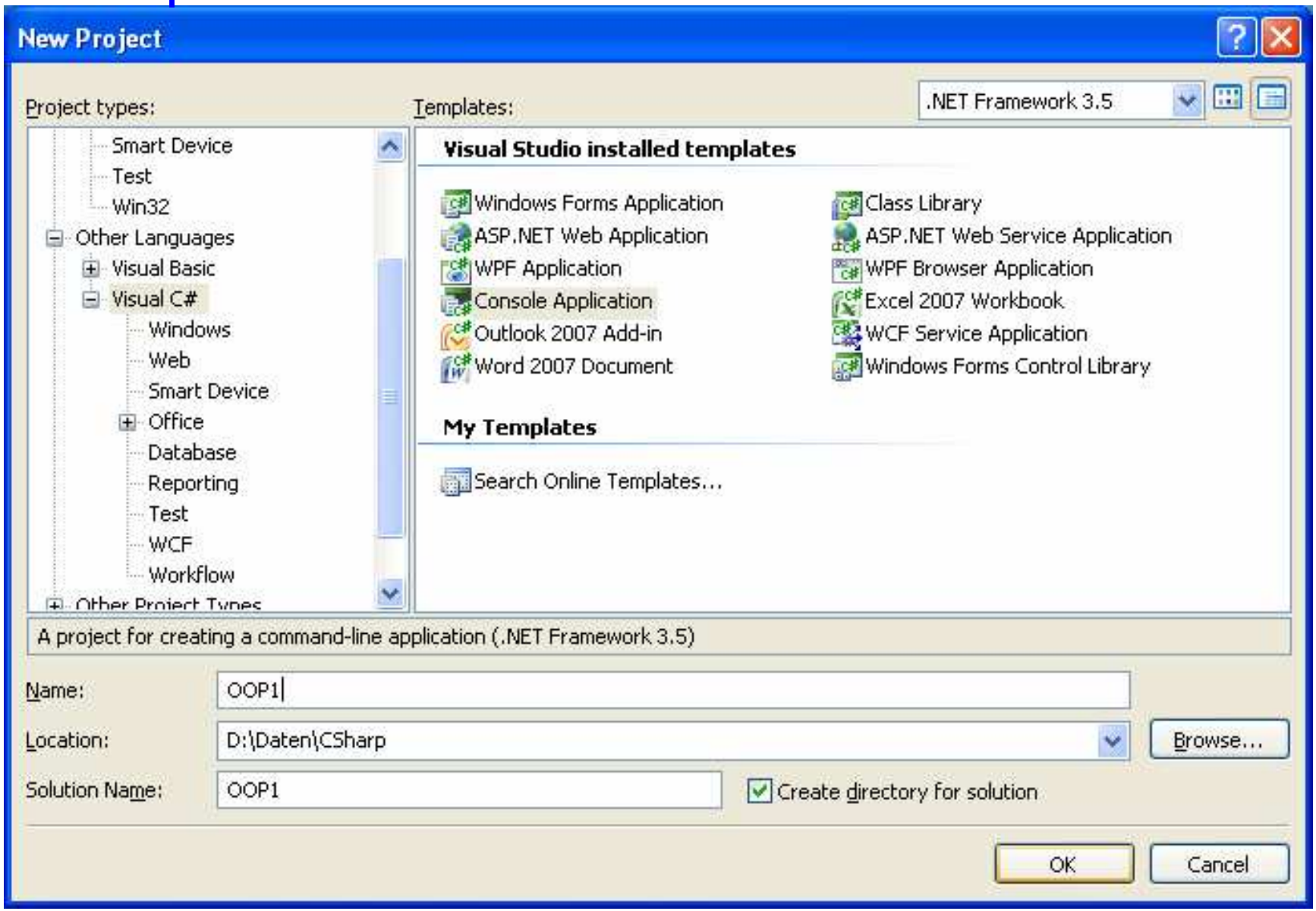
C# Sprache: OOP

- Zugrif auf Variablen:

```
private
internal                    // nur in Assembly, Packages
public
protected
```

Modifizierer	**Innerhalb der Klasse**	**Innerhalb der Ableitungen**	**Außerhalb der Klasse**	**Außerhalb der Anwendung**
public	Ja	Ja	Ja	Ja
internal	Ja	Ja	Ja	Nein
protected	Ja	Ja	Nein	Nein
private	ja	Nein	Nein	Nein

C# Sprache: OOP



C# Sprache: OOP

- Konstruktor — wie java
- Dekonstruktor — fast wie java

```
~student()
{
}
```

- Methoden: set/get-explizit

- Ableitung einer neuen Klasse:

```
class linie  : punkt
{
   public linie(int x, int y): base(y,x)
  {
   }
}
```

C# Sprache: OOP

```
class student   {
    private int m_Matrnr;

    public student()
    {
       m_Matrnr = 0;
    }
    public student(int Matrnr)
    {
       m_Matrnr = Matrnr;
    }
    ~student()
    {
       m_Matrnr = 0;
    }
```

C# Sprache: OOP

```
class student   {
      private int m_Matrnr;
      public int Matrnr
      {
         get
         {
            return m_Matrnr;
         }
         set
         {
            if ( value>0 && value < 99999 ) m_Matrnr=value;
         }
      }
      public void print()
      {
         Console.WriteLine(m_Matrnr);
      }
  }
```

```
public void test
{
  int k;
  student  std = new student();
  std.Matrnr=17421;  // ruft set-Methode auf
  k = std.Matrnr;  // ruft die get-Methode auf
}
```

C# Sprache: OOP

```
class Program
   {
      static void Main(string[] args)
      {
         student hase, igel;  // keine Erzeugung, kein C++
         hase = new student();
         hase.Matrnr = 14721;
         hase.print();
         igel = new student(12345);
         igel.print();
         Console.ReadLine();
      }
   }
```

C# Sprache: Mehrere Konstruktoren

```
class koerper    {
        private int m_volume;


        public koerper():this(0)        {
        }

        public koerper(int volume)        {
            m_volume = volume;
        }

        public string getVolumen        {
              return m_volumen; }
        }
}
```

C# Sprache: Überscheiben von Methoden

```
class koerper    {
        private string m_volume;
        public koerper(int volume)        {
            m_name = name;
        }
        public string name        {
            get           {
                return m_name;
            }
        }
        virtual public void print()        {
            Console.WriteLine("Name: {0}  Volumen {1}", m_name, volume);
        }
    }
```

C# Sprache: Überschreiben von Methoden

```
class koerper2 : koerper    {
        private string m_name;
        public koerper2(string name, int volume):base(volume)        {
            m_name = name;
        }
        public string name        {
            get            {
                return m_name;
            }
        }
        public override void print()        {
            Console.WriteLine("Name: {0}  Volumen {1}", m_name, volume);
        }
    }
```

public virtual void print()

C# Sprache: Operator Überladen

Operatoren	Overloadbar
+, -, *, /, %, &, |, <<, >>	Alle binären Operatoren können überladen werden
+, -, !, ~, ++, --, true, false	Alle unären Operatoren können überladen werden
==, !=, <, >, <= , >=	Alle relationalen Operatoren können, paarweise, überladen werden
&&, ||	Keine Überladung
()	Cast-Operator
+=, -=, *=, /=, %=	Können überladen werden. Funktioniert aber automatisch
=, . , ?:, ->, new, is, as, sizeof	Keine Überladung

C# Sprache: Operator Überladen

```
class Koerper
{
        private int m_volume;

        public static Koerper operator +(Koerper k1, Koerper k2)
        {
            Koerper temp = new Koerper();
            temp.volume = k1.volume + k2.volume;
            return temp;
        }

}
```

C# Sprache: Delegates

- Delegates sind der Ersatz für Pointerfunktionen
- Delegates können sowohl statische als auch Instanzmethoden referenzieren.
- Sie müssen aber denselben Rückgabewert und dieselben Parametertypen besitzen wie die zu referenzierenden Methoden.

```
public delegate int calcDel(int val1, int val2);  // "Deklaration der Funktion

public class Programm    {
   public static int Addint(int a, int b)
   {
      return a + b;
   }

   public static int Multint(int a, int b)
   {
      return a * b;
   }

public static void test1(calcDel calc)
{
    int[] a = {1,2,3,4,5};
     int i;
     for (i=0; i<a.Length; i++)
     {
        a[i] = calc(a[i], 2);
     }
     i=0;
     foreach (int j in a)
     {
       i++;
       Console.WriteLine("A[ {0} ] = {1}", i, j);
     }
     Console.WriteLine();
}
```

```
public static void Main()
{
      // Die Instanzmethode dem Delegate zuweisen
      calcDel calc1  = new calcDel(Addint);
      calcDel calc2  = new calcDel(Multint);
       test1(calc1);
       test1(calc2);

     Console.ReadLine();
}


}  // Programm
```

C# Sprache: Literatur und Links

Softwareentwicklung mit C#
Hanspeter Mössenböck
dpunkt.Verlag
ISBN 3-89864-406-5

Visual C+ 200
Günter Born, Benjamin Born
Entwickler.press
ISBN 978-3-939084-40-2

- http://www.guidetocsharp.de
- http://msdn.microsoft.com/de-de/library/kx37x362.aspx