



# Überblick

## Visual Studio und C#

Justus Bisser  
Microsoft Student Partner  
Universität Freiburg

# Agenda

---

- Visual Studio 2005
- C#
- Debuggen

01000100100100101001010100110011  
01010100101010101010101010101010  
01010101011111011010000101011  
01000100100100101010100110011  
01010100101010101010101010101010  
010101011111011010000101011

# .Net-Framework

## Common Language Runtime

Windows  
Forms

ASP.Net

.Net  
Compact  
Framework

XNA

- Gleiches Framework auf verschiedenen Plattformen

# Windows Anwendung

- Hauptfenster wird automatisch erzeugt
- Referenz auf `System.Windows.Forms`
- Erzeugt lauffähige Anwendungen (.exe)



# Windows Anwendung

---

- Fensterschleife

```
Application.Run(new Form1());
```

0100010010010100101010011001  
0101010010101010101010101010  
0101010101111101101000010101  
0100010010100101010100110011  
0101010010101010101010101010  
01010101111101101000010101

# Klassenbibliothek

---

- Nicht allein lauffähig (.dll)
- Enthält Klassen für andere Programme
- DLL-Hölle

0100010010010100101010011001  
0101010010101010101010101010  
0101010101111101101000010101  
0100010010010010101010011001  
0101010010101010101010101010  
0101010101111101101000010101



# XNA-Spiel

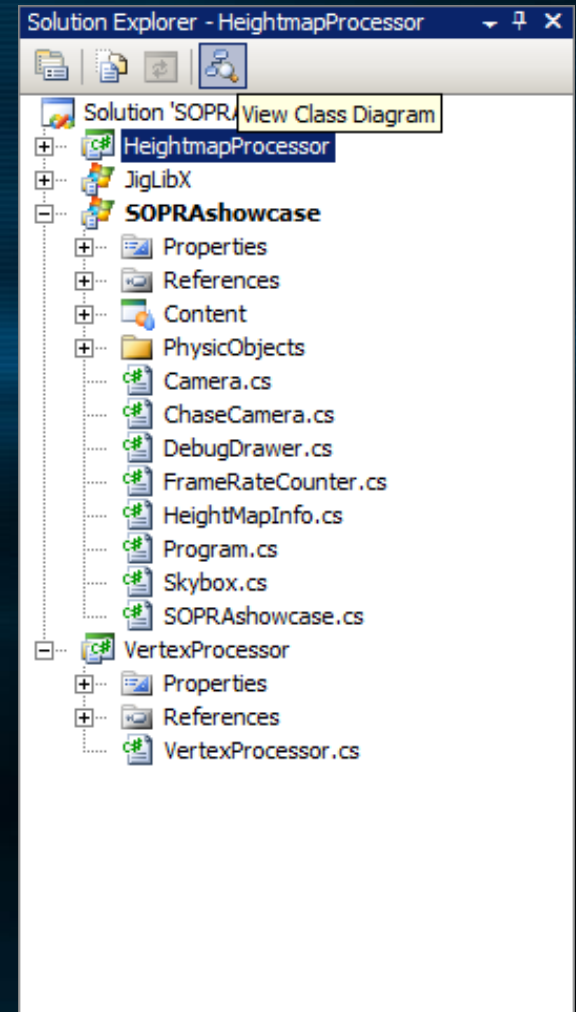
- Allein lauffähig
- Basiert auf .Net Compact Framework
- Auch für Xbox 360 und bald Zune





# Projektstruktur

- Solution
  - Projekt 1
    - Klasse 1
    - Klasse 2
  - Projekt 2
    - Klasse 3
  - ...



# C#

---

- "Nachfolger" von Java
- Bevorzugte .Net-Sprache
- Pointer



# Konventionen

- Camelcase
- Klassen- und Funktionsnamen werden groß geschrieben

```
Console.WriteLine();
```

- Interfacenamen beginnen mit "I"

```
IEnumerable
```

```
01000100100101001010100110011
01010100101010101010101010101
01010101011111011010000101011
01000100100100101010100110011
01010100101010101010101010101
01010101011111011010000101011
```

# C#

- foreach

```
foreach (Object i in IEnumerable)
{
    Console.WriteLine(i.ToString());
}
```

# C#

- using

```
using (IDisposable tempObject)
{
    ...
}
```

- Automatischer Aufruf von  
tempObject.Dispose()



# C#

- switch

```
switch ("string")  
{  
    case "string":  
        break;  
    default:  
}
```

# C#

- switch
  - Fälle müssen beendet werden

```
switch (i)
{
    case 1:
    case 2:
        Do something
    case 3:
        Go on doing sth.
        break;
    default:
        Do something different
}
```

# C#

- **Pointer**

```
unsafe
```

```
{
```

```
    int * ptri;
```

```
}
```

- Nur wenn `unsafe` markiert
- Nur mit Compileroption `/unsafe`

# C#

- Getter & Setter

```
public static class Bar {  
    private static String foo = "Foo";  
    public static String Foo {  
        get { return foo; }  
        set { foo = value; }  
    }  
}
```

# C#

- Getter & Setter

```
Console.WriteLine (Bar.Foo);  
Bar.Foo = "Bar";
```

## Anstatt in Java

```
System.out.println (Bar.getFoo());  
Bar.setFoo ("Bar");
```



# Parameter

- Default: per value
- Keywords:
  - `ref`
  - `out`
  - `params int[] zahlen`

# Parameter

```
void Foo (StringBuilder x) {  
    x = null;  
}
```

```
...  
StringBuilder y = new StringBuilder();  
y.Append ("hello"); Foo (y);  
Console.WriteLine (y==null);
```

Ausgabe:

False

# Parameter

- out

```
public Object Creator(out Object o) {  
    o = new Object();  
}  
  
Object o;  
Creator(o);  
Console.WriteLine(o == null);
```

Ausgabe:

False

# Parameter

```
public static add(params int[] zahlen)
{
    int sum = 0;
    foreach(int i in zahlen)
        sum += i;
    return sum;
}
```

- **Aufruf:**

```
Console.WriteLine(sum(1, 2, 3));
```

# Generics

- Typisierte Listen
- Dadurch weniger Laufzeitfehler, da Casten entfällt
- Compiler-Fehler falls inkompatible Objekte eingefügt werden
- In Namespace  
`System.Collections.Generic`
- Langsamer da Typprüfung



# Generics

- Ohne Generics

```
ArrayList liste = new ArrayList();  
liste.Add("Hallo World!");  
liste.Add(new Button);  
Console.WriteLine((string)liste[0]);  
Console.WriteLine((string)liste[1]);
```

- Objekte werden als object gespeichert

# Generics

- Generics

```
List<String> liste = new  
    List<String>();  
liste.Add("Hallo World");  
Console.WriteLine(liste[0]);
```

- Objekte werden mit Type gespeichert

- liste.Add(new Button());

→ Compiler-Fehler

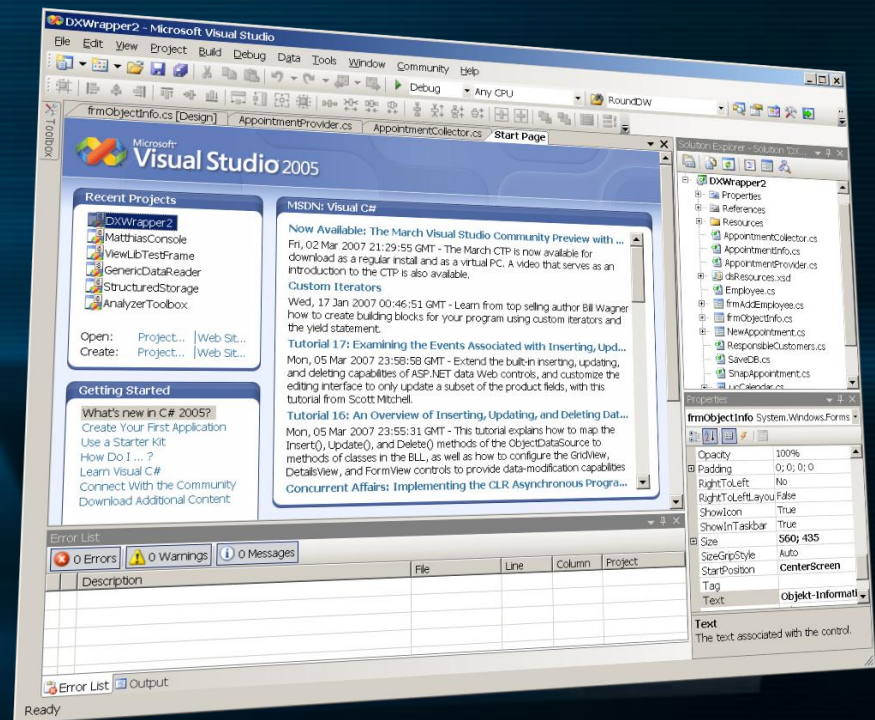
# Konfigurationen

---

- Programme können Konfiguration einfach speichern
- Benutzerspezifisch oder Global
- App.config wird automatisch erzeugt

0100010010010100101010011001  
0101010010101010101010101010  
0101010101111101101000010101  
0100010010010010101010011001  
0101010010101010101010101010  
0101010101111101101000010101

# DEMO



# Visual Studio 2005



# Debuggen

- `#if DEBUG`

`//Anweisungen, die im Debug-Build ausgeführt werden`

- `#endif`

- Ähnlich auch `#if XBOX`

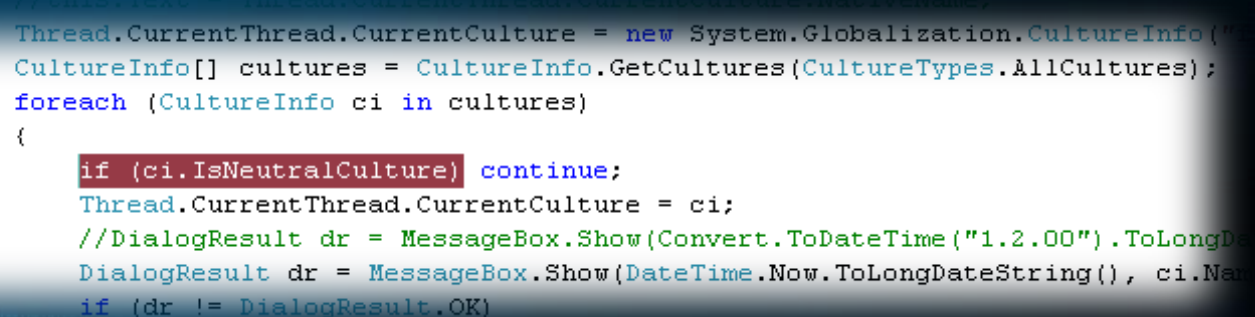
```
public frmSimpleControls()
{
    #if DEBUG
        MessageBox.Show("Test");
    #endif

    //this.Text = Thread.CurrentThread.CurrentCulture
    Thread.CurrentThread.CurrentCulture = new System.
    CultureInfo("de-DE");
}
```



# Debuggen

- Breakpoints
  - Unterbrechen Programmfluß
  - (fast) beliebig setzbar

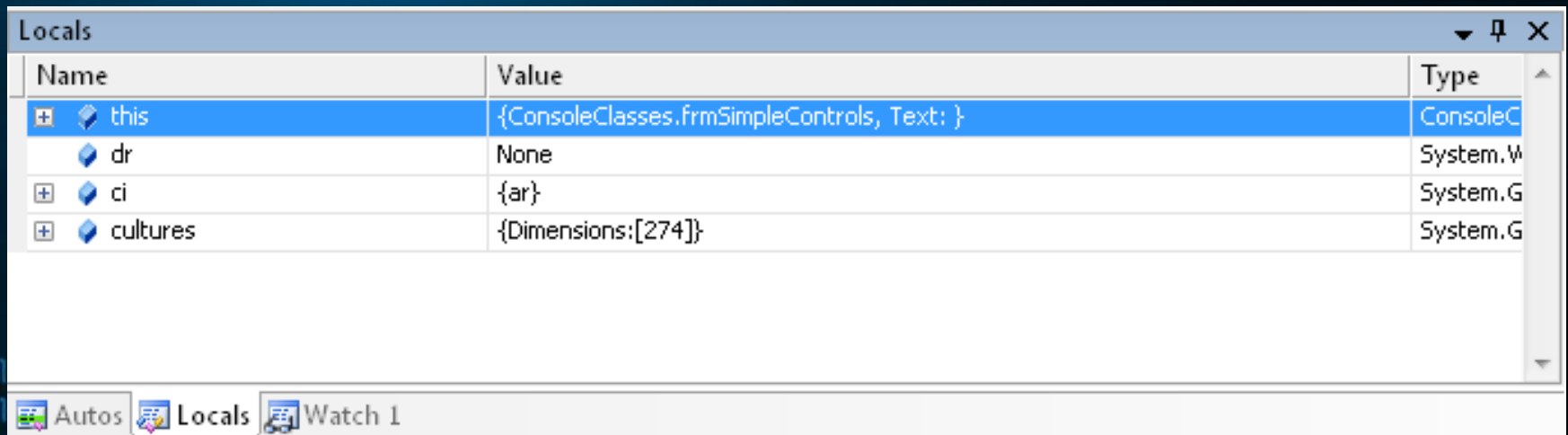


```
// this.Text = Thread.CurrentThread.CurrentCulture.NativeName;  
Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en");  
CultureInfo[] cultures = CultureInfo.GetCultures(CultureTypes.AllCultures);  
foreach (CultureInfo ci in cultures)  
{  
    if (ci.IsNeutralCulture) continue;  
    Thread.CurrentThread.CurrentCulture = ci;  
    //DialogResult dr = MessageBox.Show(Convert.ToDateTime("1.2.00").ToLongDateString(), ci.Name);  
    DialogResult dr = MessageBox.Show(DateTime.Now.ToLongDateString(), ci.Name);  
    if (dr != DialogResult.OK)  
        continue;  
}
```

The screenshot shows a code editor with a red dot breakpoint set on the line `if (ci.IsNeutralCulture) continue;`. The code is in C# and deals with thread culture and message boxes. The background of the slide features a faint pattern of binary code (0s and 1s) on the left side.

# Debuggen

- Variablen beobachten
- Einzel- oder Prozedurschritt



# Debuggen

- Breakpoints durch Code
  - `Debugger.Break();`
  - Funktional genau wie ein Breakpoint
- `Debugger.Abort();`
- `Debugger.Ignore();`

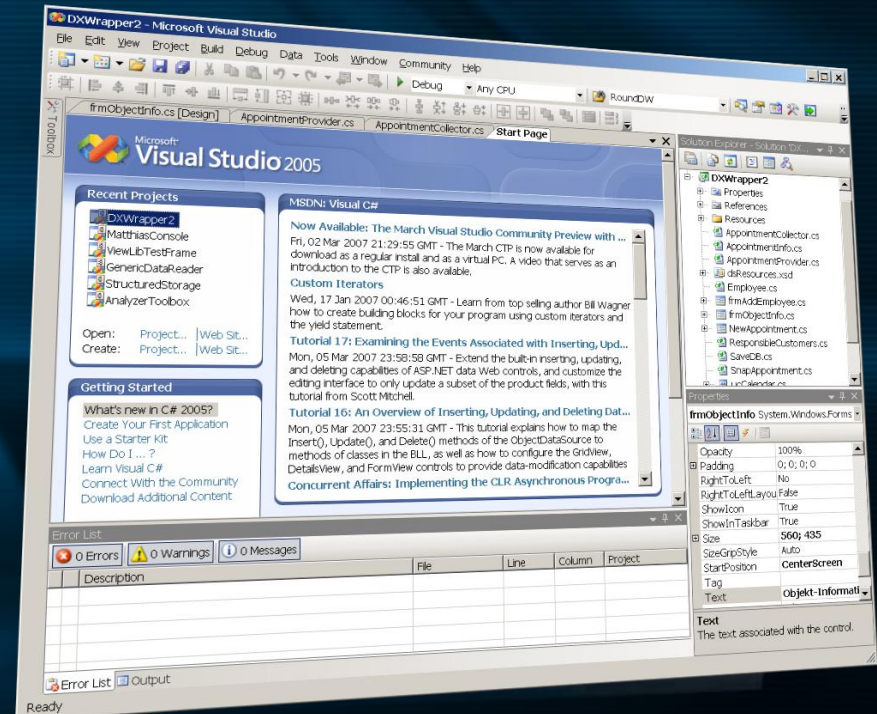
0100010010010100101010011001  
0101010010101010101010101010  
0101010101111101101000010101  
0100010010010010101010011001  
0101010010101010101010101010  
01010101111101101000010101

# Debuggen

- `Debug.WriteLine("Text");`
- `Debug.Flush();`
- `Debugger.Log(priority,  
"Kategorie", "Message");`



# DEMO



# Debuggen



# Vielen Dank

---

Noch Fragen ?

Justus Bisser

[Justus.Bisser@studentprogram.de](mailto:Justus.Bisser@studentprogram.de)



# Links

---

- Vergleich Java <-> C#

<http://www.25hoursaday.com/CsharpVsJava.html>

- Unsafe code in C#

<http://www.csharpfriends.com/Articles/getArticle.aspx?articleID=351>

- Parameter

<http://www.yoda.arachsys.com/csharp/parameters.html>

0100010010010100101010011001  
0101010010101010101010101010  
0101010101111101101000010101  
010001001010010101010011001  
0101010010101010101010101010  
01010101111101101000010101

# Mehr Links

- Einführung Reflection

<http://www.codeproject.com/csharp/IntroReflection.asp>

- CodeDOM Beispiel

[http://msdn2.microsoft.com/de-de/library/saf5ce06\(VS.80\).aspx](http://msdn2.microsoft.com/de-de/library/saf5ce06(VS.80).aspx)

- Dynamisches Subclassing

<http://www.devx.com/dotnet/Article/28783>

- Programming C#

<http://www.oreilly.com/catalog/progcsharp/chapter/ch18.html>

<http://www.oreilly.com/catalog/progcsharp/>

- Reflector .Net

<http://www.aisto.com/roeder/dotnet/>