Die Sprache C# 4. Teil

School of Engineering

- Delegates and Events
- Vereinfachungen, Anonyme Methoden
- Ausnahmen
- Attribute
- Debug, Trace,Test
- Quellendokumentation

Delegates und Events

Delegate = Methodentyp

Funktionstyp in C

Deklaration eines Delegate-Typs

```
delegate void Notifier (string sender);  // normale Methodensignatur
                                         // mit Schlüsselwort delegate
```

Deklaration einer Delegate-Variablen

```
Notifier greetings;
```

Funktionsvariable in C

Zuweisung einer Methode an eine Delegate-Variable

```
void SayHello(string sender) {
  Console.WriteLine("Hello from " + sender);
}

greetings = new Notifier(SayHello);
```

Aufruf der Delegate-Variablen

```
greetings("Max"); // Aufruf von SayHello("Max") => "Hello from Max"
```

Zuweisung unterschiedlicher Methoden

School of Engineering

Jede "passende" Methode kann einer Delegate-Variablen zugewiesen werden

```
void SayGoodBye(string sender) {
  Console.WriteLine("Good bye from " + sender);
}
greetings = new Notifier(SayGoodBye);
greetings("Max");    // SayGoodBye("Max") => "Good bye from Max"
```

Bemerkungen

- Einer Delegate-Variablen darf *null* zugewiesen werden.
- Delegate-Variable darf nicht aufgerufen werden, wenn sie keinen Delegate-Wert enthält (sonst gibt es eine Exception).
- Delegate-Variable kann in Datenstruktur gespeichert, als Parameter übergeben werden etc.

Erzeugen von Delegate-Werten

```
new DelegateType (obj.Method)
```

es geht auch einfacher

- *obj* kann *this* sein (und somit fehlen)
- *Method* kann *static* sein. In diesem Fall steht statt *obj* der Klassenname.
- *Method* darf nicht *abstract*, wohl aber *virtual*, *override*, *new* sein.
- *Method*-Signatur muss mit *DelegateType* übereinstimmen
  - gleiche Anzahl von Parametern
  - gleiche Parametertypen (inklusive Return-Typ)
  - gleiche Parameterarten (ref, out, value)

Delegate-Variable speichert Methode *und* Empfängerobjekt !

Multicast-Delegates

School of Engineering

Delegate-Variable kann mehrere Werte gleichzeitig aufnehmen

```
Notifier greetings;
greetings = new Notifier(SayHello);
greetings += new Notifier(SayGoodBye);
greetings("Max");       // "Hello from Max"
                        // "Good bye from Max"
```

```
greetings -= new Notifier(SayHello);
greetings("Max");       // "Good bye from Max"
```

Bemerkungen

- Wenn ein Multicast-Delegate eine **Funktion** ist, wird letzter Funktionswert geliefert.
- Wenn ein Multicast-Delegate ***out*-Parameter** hat, wird Parameter des letzten Aufrufs geliefert. ***ref-Parameter*** werden durch alle Methoden durchgereicht.

Analogie zu Java

School of Engineering

```
interface Notifyer {            // entspricht Delegate-Typ
  void greetings(String sender);
}

class HelloSayer implements Notifier {       // entspricht Delegate-Wert
  public void greetings(String sender) {
    System.out.println("Hello from" + sender);
  }
}

Notifier n = new HelloSayer();  // Initialisierung der Delegate-Variablen

n.greetings("Max");
```

- Interface übernimmt Rolle des Delegate Typs.
- Rund um die aufzurufende Methode braucht aber man eine Klasse, die das Interface implementiert.
- Keine Aufrufe statischer Methoden möglich.
- Multicasts erfordern zusätzlich Registrierungsmechanismus und Aufrufschleife.

Events = eingeschränkte Delegate-Variable

```
class Model {
   public event Notifier notifyViews;
   public void Change() { ... notifyViews("Model"); }
}
```

```
class View {
   public View (Model m) { m.notifyViews += new Notifier(Update); }
   void Update (string sender) {Console.WriteLine(sender + " was changed"); }
}
```

```
class Test {
   static void Main() {
     Model m = new Model();
     new View(m); new View(m);
     m.Change();
   }
}
```

Warum Events statt normaler Delegate-Variablen?

- Nur die Klasse, die das Event deklariert, darf es auslösen (bessere Kapselung).
- event-Felder dürfen von aussen nur mit += oder -= modifiziert werden (nicht "=")

Event-Handling in der .NET-Bibliothek

Delegates für das Event-Handling haben folgende Signatur

```
delegate void SomeEvent (object source, MyEventArgs e);
```

- Rückgabetyp void
- 1. Parameter = Sender des Events (Typ *object*)
- 2. Parameter = Event-Parameter (Unterklasse von *System.EventArgs*)

```
public class EventArgs {
   public static readonly EventArgs Empty;
}
```

Beispiel für .Net-Events

```
public delegate void KeyEventHandler (object sender, KeyEventArgs e);
public class KeyEventArgs : EventArgs {
  public virtual bool Alt { get {...} }      // true if Alt-Key was pressed
  public virtual bool Shift { get {...} }    // true if Shift-Key was pressed
  public bool Control { get {...} } // true if Ctrl-Key was pressed
  public bool Handled { get{...} set {...} } // indicates if event was handled
  public int KeyValue { get {...} } // the typed keyboard code
  ...
}
```

```
class MyKeyEventSource {
  public event KeyEventHandler KeyDown;
  public KeyPressed() {
    KeyDown(this, new KeyEventArgs(...));
  }
}
```

```
class MyKeyListener {
  public MyKeyListener(...) { keySource.KeyDown += new
  KeyEventHandler(HandleKey);}
  void HandleKey (object sender, KeyEventArgs e) {...}
}
```

Vereinfachungen

Vereinfachte Delegate-Erzeugung

```
delegate void Printer(string s);

void Foo(string s) {
  Console.WriteLine(s);
}
```

```
Printer print;
print = new
Printer(this.Foo);
print = this.Foo;
print = Foo;
```

Vereinfachte Form:
Delegate-Typ wird aus dem Typ der linken Seite abgeleitet

```
delegate double Function(double x);

double Foo(double x) {
  return x * x;
}
```

```
Printer print = Foo;
Function square = Foo;
```

weist *Foo(string s)* zu
weist *Foo(double x)* zu

Überladung kann durch Typ der linken Seite aufgelöst werden

Anonyme Methoden

Normale Delegates

```
delegate void Visitor(Node p);

class List {
  Node[] data = ...;
  ...
  public void ForAll(Visitor visit) {
    for (int i = 0; i < data.Length; i++)
      visit(data[i]);
  }
}
```

```
class C {
  int sum = 0;

  void SumUp(Node p) {
     sum += p.value; }
  void Print(Node p) {
     Console.WriteLine(p.value); }

  void Foo() {
    List list = new List();
    list.ForAll(SumUp);
    list.ForAll(Print);
  }
}
```

- erfordern Deklaration einer benannten Methode (*SumUp*, *Print*, ...)
- *SumUp* und *Print* können nicht auf lokale Variablen von *Foo* zugreifen
  => *sum* muss als Instanzvariable deklariert werden

Verwendung Anonymer Methoden

```
delegate void Visitor(Node p);

class C {

  void Foo() {
    List list = new List();
    int sum = 0;
    list.ForAll(delegate (Node p) { Console.WriteLine(p.value); });
    list.ForAll(delegate (Node p) { sum += p.value; });
  }
}
```

```
class List {
  ...
  public void ForAll(Visitor visit) {
    ...
  }
}
```

formale Parameter

Code

Bemerkungen

- Methodencode wird in-place angegeben
- keine Deklaration einer benannten Methode nötig
- aufgerufene Methode kann auf lokale Variable *sum* zugreifen
- return beendet anonyme Methode, nicht die äussere Methode

Einschränkungen

- anonyme Methoden dürfen keine formalen Parameter der Art *params T[]* enthalten
- anonyme Methoden dürfen nicht an *object* zugewiesen werden
- anonyme Methoden dürfen keine ref- und out-Parameter äusserer Methoden ansprechen

Weglassen unbenutzter Parameter

```
delegate void EventHandler (object sender, EventArgs arg);
```

```
Button button = new Button();
button.Click += delegate (object sender, EventArgs arg)
                                    { Console.WriteLine("clicked"); };
```

vereinfacht sich zu

```
button.Click += delegate { Console.WriteLine("clicked"); };
```

Formale Parameter können weggelassen werden, wenn

- sie im Methodenrumpf nicht benutzt werden
- wenn der Delegate-Typ keine out-Parameter hat

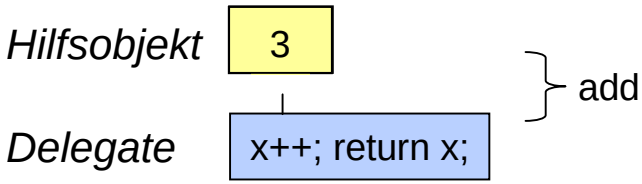
Äussere Variablen

- Annonyme Methoden können auf Variablen der umgebenden Methode zugreifen

```
delegate int Adder();

class Test {
  static Adder CreateAdder() {
    int x = 0;
    return delegate { x++; return x; };
  }
  static void Main() {
    Adder add = CreateAdder();
    Console.WriteLine(add());
    Console.WriteLine(add());
    Console.WriteLine(add());
  }
}
```

Ausgabe:

```
1
2
3
```



Das Hilfsobjekt lebt so lange wie das Delegate-Objekt

Ausnahmen (Exceptions)

try-catch-finally-Anweisung

School of Engineering

```
FileStream s = null;
try {
  s = new FileStream(curName, FileMode.Open);
  ...
} catch (FileNotFoundException e) {
  Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
  Console.WriteLine("some IO exception occurred");
} catch {
  Console.WriteLine("some unknown error occurred");
} finally {
  if (s != null) s.Close();
}
```

- *catch*-Klauseln werden in der Reihenfolge ihrer Aufschreibung getestet.
- optionale *finally*-Klausel wird immer ausgeführt.
- Exception-Name in *catch*-Klausel kann entfallen.
- Exception-Typ muss von *System.Exception* abgeleitet sein.
  Fehlt er, wird automatisch *System.Exception* angenommen.

System.Exception

zh aw School of Engineering

Properties

e.Message	die Fehlermeldung als String; wird eingestellt durch *new Exception(msg);*
e.Source	Name der Applikation oder des Objekts, das die Ausnahme ausgelöst hat
e.StackTrace	die Methodenaufrufkette als String
e.TargetSite	das Methodenobjekt, das die Ausnahme ausgelöst hat
...	

Methoden

e.ToString()	liefert den Namen der Ausnahme
...	

Auslösen von Ausnahmen

School of Engineering

Durch ungültige Operation (implizit)

- Division durch 0
- Indexüberschreitung
- null-Zugriff
- ...

Durch throw-Anweisung (explizit)

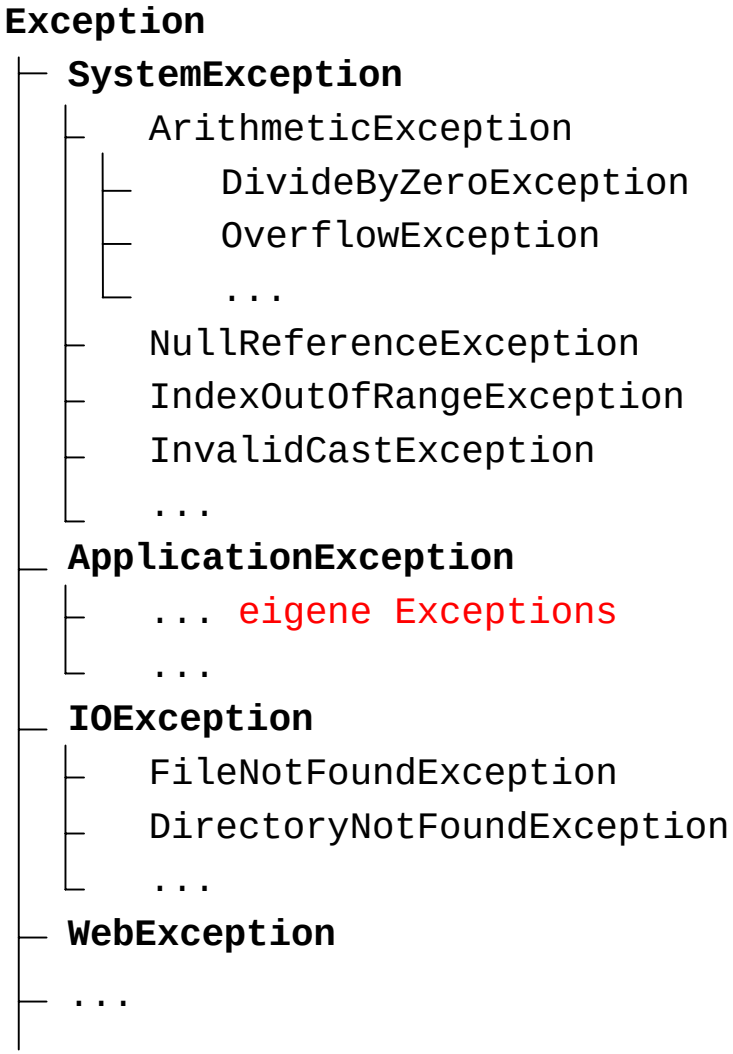
```
throw new FunnyException(10);
```

```
class FunnyException : ApplicationException {
   public int errorCode;
   public FunnyException(int x) { errorCode = x; }
}
```

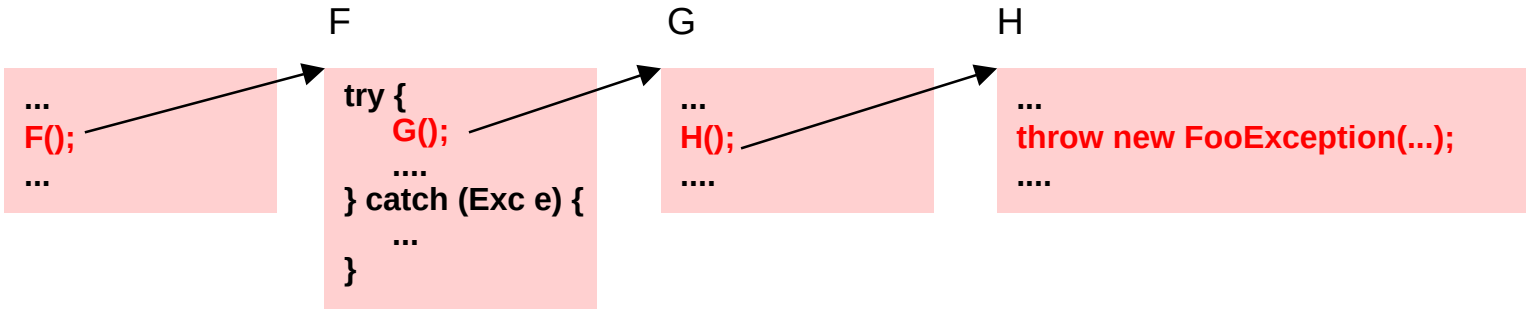
Durch Aufruf einer Methode, die eine Ausnahme auslöst und nicht behandelt

```
s = new FileStream(...);
```

Exception-Hierarchie (Auszug)



Suche nach passender catch-Klausel



- In Aufrufkette wird rückwärts nach passender catch-Klausel durchsucht.
- Wenn keine gefunden => Programmabbruch mit Fehlermeldung und Stack-Trace

C#

- **Ausnahmen müssen in C# nicht (wie in Java) behandelt werden**
- Keine Unterscheidung zwischen *Checked Exceptions* und *Unchecked Exceptions*
- Grund (wahrscheinlich): nicht alle .NET-Sprachen unterstützen Checked Exceptions (z.B. C++)

Ausnahmen müssen nicht im Methodenkopf angegeben werden

```
void myMethod() throws IOException {
   ... throw new IOException(); ...
}
```

Rufer von *myMethod* müssen

- *IOException* behandeln, oder
- *IOExceptions* in eigenem Methodenkopf spezifizieren

C#

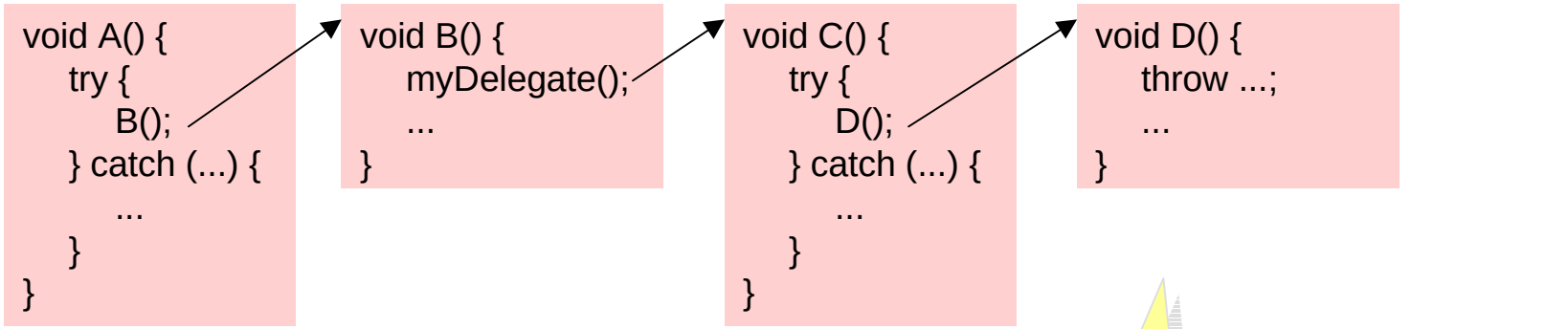
```
void myMethod() {
   ... throw new IOException(); ...
}
```

Rufer von *myMethod* können *IOException* behandeln, müssen es aber nicht.

+ kürzer und bequemer
- weniger sicher und robust

Ausnahmen in Delegates - keine

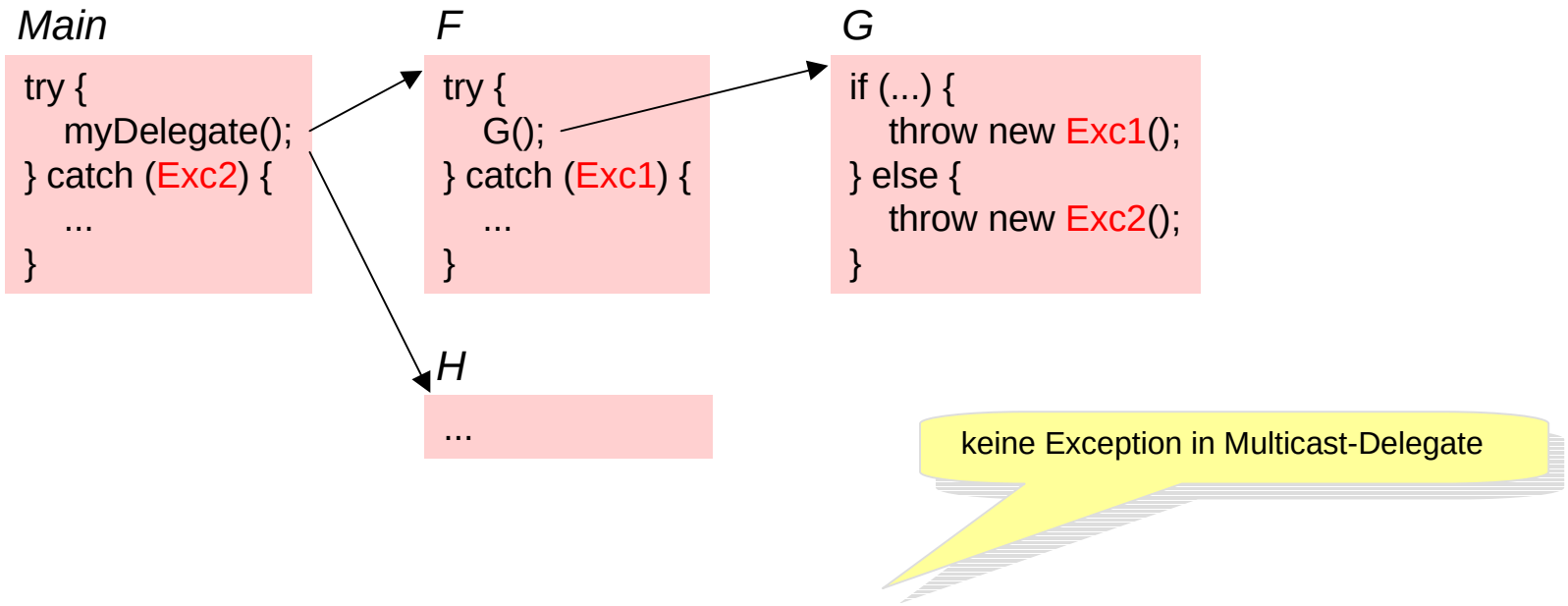
Delegates werden bei der Suche nach catch-Klausel wie normale Methoden behandelt.



als Delegate aufgerufen

kein Unterschied ob Methode oder Delegate aufgerufen

Ausnahmen in Multicast-Delegates



- tritt in *G()* die Ausnahme *Exc1* auf, wird auch noch *H()* aufgerufen
- tritt in *G()* die Ausnahme *Exc2* auf, wird *H()* nicht mehr aufgerufen, weil *Exc2* erst in *Main()* abgefangen wird.

Attribute

Attribute

Benutzerdefinierbare Informationen über Programmelemente

- Kann man an Typen, Members, Assemblies, etc. anhängen.
- Erweitern vordefinierte Attribute wie *public*, *sealed* oder *abstract*.
- Werden als Klassen implementiert, die von *System.Attribute* abgeleitet sind.
- Werden in Metadaten gespeichert.
- Werden teilw. von CLR-Services benutzt (Serialisierung, Remoting, COM-Interoperabilität)
- Können zur Laufzeit abgefragt werden.

Beispiel

```
[Serializable]
class C {...}// Klasse serialisierbar
```

Auch mehrere Attribute zuordenbar

```
[Serializable] [Obsolete]
class C {...}
[Serializable, Obsolete]
class C {...}
```

Attribute

Attributverwendung

- Attribut in "[" "]"

```
[Obsolete]
class C {...}// Klasse serialisierbar
```

Attributdeklaration

- Attribut erbt von System.Attribute

```
public class Obsolete : Attribute {
}
```

Attribute mit Parametern

School of Engineering

Beispiel

Positionsparameter

Namesparameter

```
[Obsolete("Use class C1 instead", IsError=true)]  // bewirkt
Compilefehlermeldung, dass
public class C {...}                    // C obsolet ist
```

Positionsparameter = Parameter des Attributkonstruktors

Namensparameter = alle Attributproperties möglich

Attributdeklaration

```
public class Obsolete : Attribute {
  public string Message { get; }
  public bool IsError { get; set; }
  public Obsolete() {...}
  public Obsolete(string msg) {...}
  public Obsolete(string msg, bool error) {...}
}
```

Daher auch möglich

```
[Obsolete]// Message == "", IsError == false
[Obsolete("some Message")]          // IsError == false
[Obsolete("some Message", false)]
[Obsolete("some Message", IsError=false)]
```

Werte müssen Konstanten sein

Beispiel: AttributeUsage

Verwendung neuer Attribute muss durch *AttributeUsage* festgelegt werden

```
public class AttributeUsage : Attribute {
  public AttributeUsage (AttributeTargets validOn) {...}
  public bool AllowMultiple { get; set; }  // default: false
  public bool Inherited { get; set; } // default: false
}
```

validOn An welche Programmelemente darf das Attribut angehängt werden?

AllowMultiple Darf es mehrfach angehängt werden?

Inherited Wird es von Unterklassen geerbt?

Verwendung

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method,
AllowMultiple=false)]
public class MyAttribute : Attribute {...}
```

Eigene Attribute

Deklaration des Attributs

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Method, Inherited=true)]
class Comment : Attribute {
   string text, author;
   public string Text { get {return text;} }
   public string Author { get {return author;} set {author = value;} }
   public CommentAttribute (string text) { this.text = text; author ="HM"; }
}
```

Anwendung des Attributs

```
[Comment("This is a demo class for attributes", Author="XX")]
class C { ...
    [Comment("This is a demo method for attributes", Author=”YY")]
   void foo(string s) {… }
}
```

Abfrage des Attributs zur Laufzeit

```
class Attributes {

   static void Main() {
      Type t = typeof(C);
      object[] a = t.GetCustomAttributes(typeof(CommentAttribute), true);
      CommentAttribute ca = (CommentAttribute)a[0];
      Console.WriteLine(ca.Text + ", " + ca.Author);
   }
}
```

es soll auch in Oberklassen gesucht werden

Eigene Attribute

Abfrage des Attributs einer Methode

```
class Attributes {

   static void Main() {
      Type t = typeof(C);
      object[] a = t.GetMethod(”foo").GetCustomAttributes(typeof(CommentAttribute),
true);
      CommentAttribute ca = (CommentAttribute)a[0];
      Console.WriteLine(ca.Text + ", " + ca.Author);
      ...
```

Abfrage des Attributs der Methoden in einer Delegate-Variablen

```
using System.Reflection;

class Attributes {

    delegate void Notifier (string sender);
    Notifier greetings;

    static void Main() {
              greetings += new Notifier(foo);
      Delegate[] invlist = greetings.GetInvocationList();
      foreach (Delegate d in invlist) {
        object[] a = d.Method.GetCustomAttributes(typeof(CommentAttribute), true);
          CommentAttribute ca = (CommentAttribute)a[0];
```

Vordefinierte Attribute

Serialisierung

```
[Serializable]
class List {
  int val;
  [NonSerialized] string name;
  List next;

  public List(int x, string s) {val = x; name = s; next = null;}
}
```

[Serializable] Anwendbar auf Klassen.
Daten von Objekten dieser Klassen werden automatisch serialisiert.

[NonSerialized] Anwendbar auf Felder.
Diese Felder werden von der Serialisierung ausgenommen.

Conditional-Attribut

Bedingter Aufruf von Methoden

oder bei Compilation

csc /d:debug

```
#define debug                    // Präprozessor-Anweisung
class C {

  [Conditional("debug")]         // geht nur bei void-Methoden
  static void Assert (bool ok, string errorMsg) {
    if (!ok) {
      Console.WriteString(errorMsg);
      System.Environment.Exit(0);  // geordneter Programmabbruch
    }
  }

  static void Main (string[] arg) {
    Assert(arg.Length > 0, "no arguments specified");
    Assert(arg[0] == "...", "invalid argument");
    ...
  }
}
```

Assert wird nur aufgerufen, wenn *debug* definiert ist.
Auch gut verwendbar für Hilfsdrucke.

Preprozessor, Debug, Trace Ausgabe, Test

Preprozessor

#define definiere ein Symbol für bedingte Compilation (siehe auch **#undef**)

#if, #endif markiere Block der bedingt compiliert wird

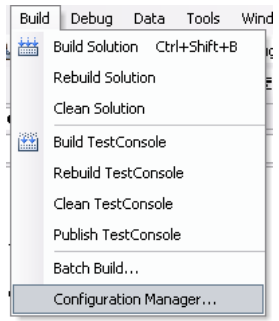
#else, #elif markiere alternativen Block, (mit Zusatzbedingung)

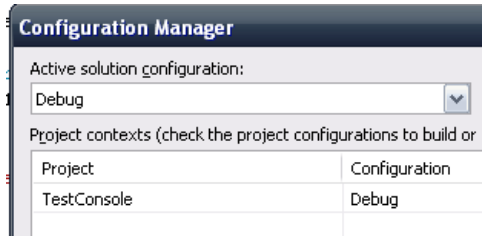
#warning,#error Ausgabe während Compilation

#region, #endregion Block für das Editor-Zusammenfalten

```
#define VC_V7
using System;
#region MyClass definition
public class MyClass {
   public static void Main() {
      #if (DEBUG)
     #warning DEBUG is defined
         Console.WriteLine("DEBUG is defined");
      #elif (DEBUG || VC_V7)
         Console.WriteLine("DEBUG or VC_V7 are defined");
      #else
         Console.WriteLine("DEBUG and VC_V7 are not defined");
      #endif
}}
#endregion
```

gesetzt im Debug Build





Debug und Trace

- Verwendet um den Zustand des Programms zu protokollieren
  - Die beiden Klassen haben dieselben Methoden
  - Trace Ausgabe wird im *Debug und Release Build* ausgegeben
  - Debug Ausgabe wird im *Debug Build* ausgegeben

- Einfaches Beispiel

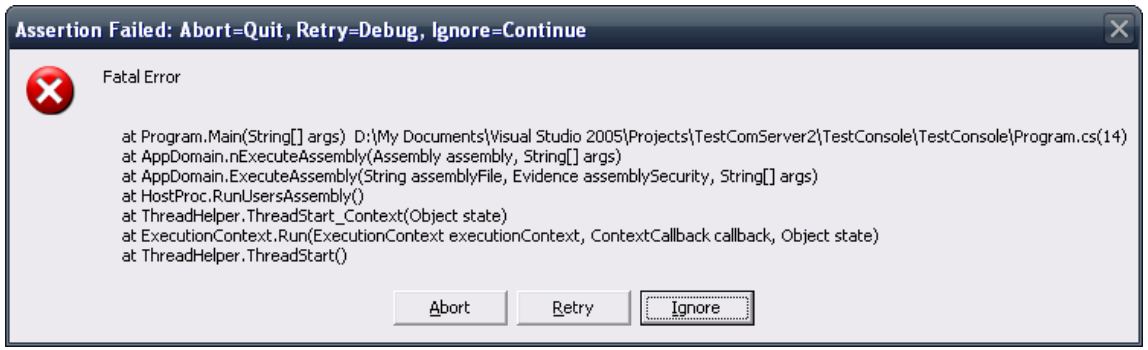
```
using System.Diagnostics;

namespace TestConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            Debug.WriteLine("Hello Debug");
            Trace.WriteLine("Hello Trace");
        }
    }
}
```

Debug und Trace

School of Engineering

- Write, WriteLine — Ausgabe von String, mit Zeilenumbruch
- WriteLineIf, — Ausgabe wenn Bedingung erfüllt ist
- Indent — Setzen der Einrückung
- Unindent — Zurücksetzen der Einrückung
- Flush — Ausgabe von angefangenen Zeilen

- Assert — Falls *false*, dann öffne Dialogbox zum Programmabbruch

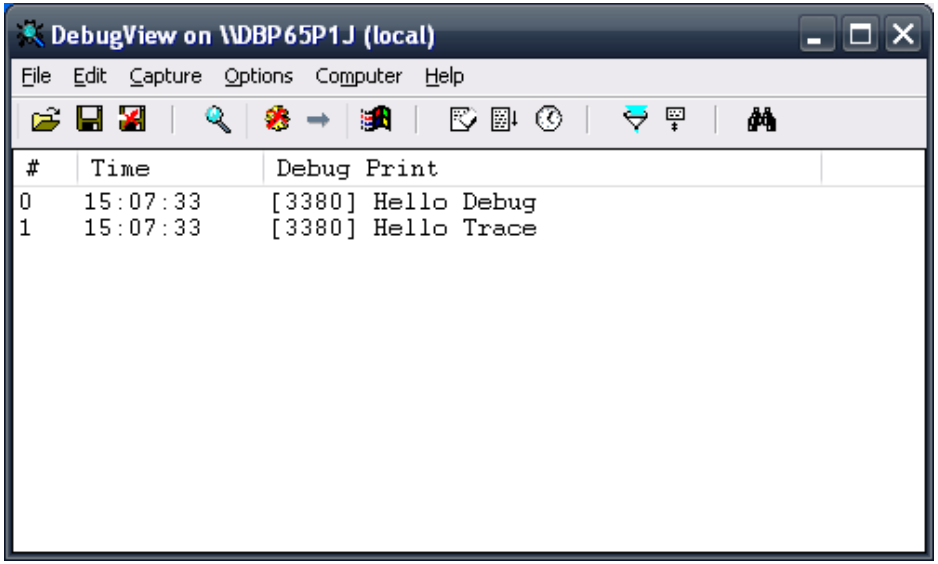


Anzeige der Debug Anweisungen

- Output wird über Setzen von Listener gesteuert

```
TextWriterTraceListener tr1 = new TextWriterTraceListener(System.Console.Out);
Debug.Listeners.Add(tr1);
```

- Anzeige der Debug/Trace Anweisungen mit einem Tool möglich: DebugView
  - http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx



Steuerung der Ausgabe über Config File

Ausgabe der Meldungen auf Console: folgender Eintrag in <AppName>.config

```
<configuration>
  <system.diagnostics>
    <sources>
      <source name="TraceTest" switchName="SourceSwitch"
        switchType="System.Diagnostics.SourceSwitch" >
        <listeners>
          <add name="console" />
          <remove name ="Default" />
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="SourceSwitch" value="Warning" />
    </switches>
    <sharedListeners>
      <add name="console" type="System.Diagnostics.ConsoleTraceListener" initializeData="false"/>
    </sharedListeners>
    <trace autoflush="true" indentsize="4">
      <listeners>
        <add name="console" />
      </listeners>
    </trace>
  </system.diagnostics>
</configuration>
```

Unit Test

- Innerhalb VS können Test Klassen definiert werden
  - ev. zusätzliche Assembly *Microsoft.VisualStudio.QualityTools.UnitTestFramework.dll*
- Ausführen direkt unter VS oder mit MSTest.exe

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class TestClass {
    [TestInitialize]
    public void Initialize() {
         MessageBox.Show("TestMethodInit");
    }

    [TestCleanup]
    public void Cleanup()
        MessageBox.Show("TestMethodCleanup");
    }

    [TestMethod]
    public void MyTest()
    {
        Assert.IsTrue(true);
    }
}
```

Quellendokumentation

Spezialkommentare (ähnlich javadoc)

School of Engineering

Beispiel

```
/// ... comment ...
class C {
  /// ... comment ...
  public int f;
  /// ... comment ...
  public void Foo() {...}
}
```

Übersetzung csc /doc:MyFile.xml MyFile.cs

- *Prüft Kommentare auf Vollständigkeit und Konsistenz*
  - ob alle Parameter dokumentiert werden
  - Namen von Programmelementen müssen korrekt geschrieben werden.
- *Expandiert Querverweise in qualifizierte Namen*
  - z.B. C => T:C, f => F:C.f, Foo => M:C.Foo
- *Erzeugt XML-Datei mit kommentierten Programmelementen*
  - XML-Datei kann mit XSL für Browser formatiert werden,
    z.B. mit <?xml:stylesheet href="doc.xsl" type="text/xsl"?>

Beispiel für Quellprogramm

```
/// <summary> A counter for accumulating values and computing the mean
  value.</summary>
class Counter {
  /// <summary>The accumulated values</summary>
  private int value;

  /// <summary>The number of added values</summary>
  public int n;

  /// <summary>Adds a value to the counter</summary>
  /// <param name="x">The value to be added</param>
  public void Add(int x) {
    value += x; n++;
  }

  /// <summary>Returns the mean value of all accumulated values</summary>
  /// <returns>The mean value, i.e. <see cref="value"/> / <see
  cref="n"/></returns>
  public float Mean() {
    return (float)value / n;
  }
}
```

Daraus erzeugte XML-Datei

XML-Datei kann in Visual Studio angezeigt werden.

Elemente sind nicht hierarchisch geschachtelt
Typen von Feldern und Methoden werden nicht mitgespeichert

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>MyFile</name>
    </assembly>
    <members>
        <member name="T:Counter">
            <summary> A counter for accumulating values ... value.</summary>
        </member>
        <member name="F:Counter.value">
            <summary>The accumulated values</summary>
        </member>
        <member name="F:Counter.n">
            <summary>The number of added values</summary>
        </member>
        <member name="M:Counter.Add(System.Int32)">
            <summary>Adds a value to the counter</summary>
            <param name="x">The value to be added</param>
        </member>
        <member name="M:Counter.Mean">
            <summary>Returns the mean value of all accumulated values</summary>
            <returns>The mean value, i.e. <see cref="F:Counter.value"/> / <see
  cref="F:Counter.n"/></returns>
        </member>
    </members>
</doc>
```

Vordefinierte XML-Tags

School of Engineering

Alleinstehend

```
<summary> Kurzbeschreibung eines Programmelements </summary>
<remarks> Ausführliche Beschreibung eines Programmelements </remarks>
<example> Beliebiger Beispieltext (z.B. Aufrufbeispiel) </example>
<param name="ParamName"> Bedeutung des Parameters </param>
<returns> Bedeutung des Rückgabewerts </returns>
<exception [cref="ExceptionType"]> Bei der Dokumentation einer Methode:
Beschreibung der Exception </exception>
```

Teil einer anderen Beschreibung

```
<code> Mehrzeilige Codestücke </code>
<c> kurze Codestücke im Text </c>
<see cref="ProgramElement"> Name des Querverweises </see>
<paramref name="ParamName"> Name des Parameters </paramref>
```

Benutzerdefinierte Tags

- Beliebige eigene XML-Tags, z.B. <author>, <version>, ...

Erstellung von HTML Kommentardatei

- doc.xsl ist ein XSLT Stylesheet, das XML->HTML umwandelt
- doc.css ist Stylesheet auf das im generierten HTML verwiesen wird
- folgendes C# Programm führt die Umwandlung durch

```
using System;
using System.Xml.Xsl;
using System.Xml;
public class CSDOC {
   public static void Main(string[] args) {
        XslTransform xt = new XslTransform();
            xt.Load("doc.xsl");
             xt.Transform(args[0], args[0].Split('.')[0]+".html", new
   XmlUrlResolver() );
   }
}
```

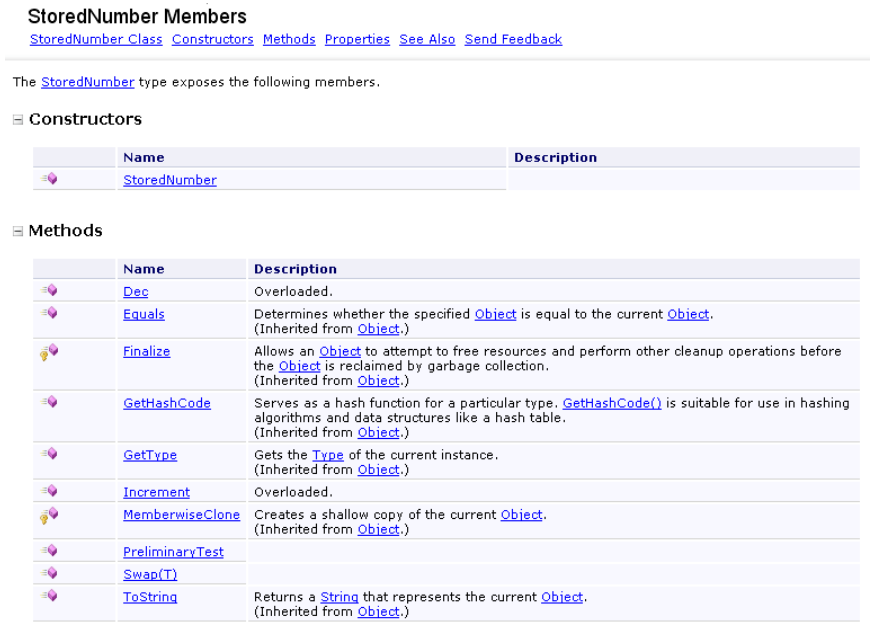
Sandcastle

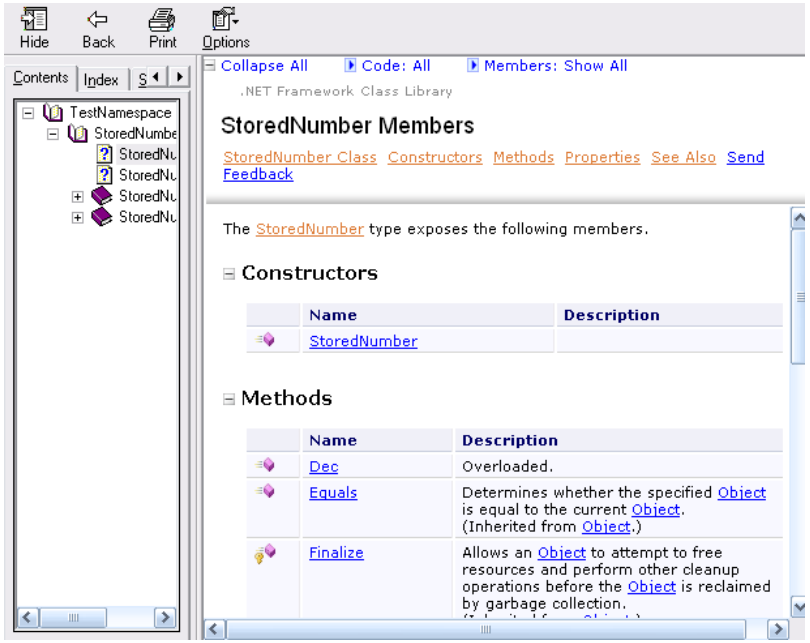
- Microsoft Tool zur Erzeugung von Dokumentation
  - HTML
  - Standard Windows Help

Sandcastle - Version 2.4.10520

Brief Description

Documentation compilers for managed class libraries
Enabling managed class library developers throughout the world to easily create accurate, informative documentation with a common look and feel.





Zusammenfassung

School of Engineering

- Delegates
  - sind Funktionstypen (Funktion kann in Variable gespeichert werden)
  - Delegate Variable kann mehrere Werte enthalten -> Broadcast
  - Events: Delegates, die nur von eigener Klasse aufgerufen werden können
- Ausnahmen (Exceptions)
  - wie Java: ausser keiner Checked-Exceptions
- Attribute
  - Programme können mit Attributen "annotiert" werden
  - können zur Lauf- oder Kompilationszeit Verhalten steuern
- XML-Kommentare
  - Dokumentation kann aus diesen Kommentaren generiert werden.

Fragen?