

Die .NET Klassenbibliothek Teil 2

- Threading
- XML Verarbeitung
- Networking
- Reflection

Threading

Beispiel

```
using System;
using System.Threading;

class Printer {
    char ch;
    int sleepTime;

    public Printer(char c, int t) {ch = c; sleepTime = t;}

    public void Print() {
        for (int i = 1; i < 100; i++) {
            Console.Write(ch);
            Thread.Sleep(sleepTime);
        }
    }
}

class Test {
    static void Main() {
        Printer a = new Printer('.', 10);
        Printer b = new Printer('*', 100);
        Thread t1 = new Thread(new ThreadStart(a.Print)); t1.Start();
        Thread t2 = new Thread(new ThreadStart(b.Print)); t2.Start();
    }
}
```

Zwei Threads, die ständig Zeichen auf dem Bildschirm ausgeben

einfacher new Thread(a.Print)

Das Programm läuft so lange, bis der letzte (Foreground-)Thread beendet ist.

■ Namensraum System.Threading unterstützt Arbeiten mit leichtgewichtigen Prozessen (= Threads)

- Ablaufplanung
- Synchronisation
- Thread-Pooling

■ Wichtige Typen von System.Threading sind

- Klassen Thread und ThreadPool
- Enumerationen ThreadState und ThreadPriority
- Klasse Monitor
- Exceptions ThreadAbortException und ThreadInterruptedException
- Delegates TreadStart, WaitCallback, TimerCallback, IOCompletionCallback, ...
- ...

Klasse Thread

```
public sealed class Thread {  
  
    public Thread(ThreadStart start);  
  
    public ThreadPriority Priority  
        {get; set;}  
    public ThreadState ThreadState  
        {get;}  
    public bool IsAlive {get;}  
    public bool IsBackground  
        {get; set;}  
    public void Start();  
    public static void Sleep(int time);  
    public void Suspend();  
    public void Resume();  
    public void Join();  
    public void Interrupt();  
    public void Abort();  
    public static void ResetAbort();  
  
    public static Thread CurrentThread  
        {get;}  
}
```

■ Konstruktor mit ThreadStart-Delegate

■ Setzen/Lesen der Priorität

■ aktueller Zustand

■ Eigenschaften Lebendigkeit,
Background

■ Methoden zur Steuerung des Threads

■ Zugriff auf aktuell laufenden Thread

ThreadStart, ThreadPriority und ThreadState

```
public sealed class Thread {  
    public Thread( ThreadStart  
        start);  
  
    public ThreadPriority Priority  
        {get; set;}  
  
    public ThreadState ThreadState  
        {get;}  
    ...  
}
```

```
public delegate void ThreadStart();
```

```
public enum ThreadPriority {  
    Highest,  
    AboveNormal,  
    Normal,  
    BelowNormal,  
    Lowest,  
}
```

```
public enum ThreadState {  
    Background,  
    Unstarted,  
    Running,  
    WaitSleepJoin,  
    SuspendRequested,  
    Suspended,  
    AbortRequested,  
    Stopped  
}
```

Anlegen eines neuen Threads

■ ThreadStart-Delegate implementieren

```
using System.Threading
public class ThreadExample {
    public static void RunT0() {    // parallel laufende Methode
        for(int i=0; i<10000; i++) {
            Console.Write("x");
            Thread.Sleep(100);
        }
    }
}
```

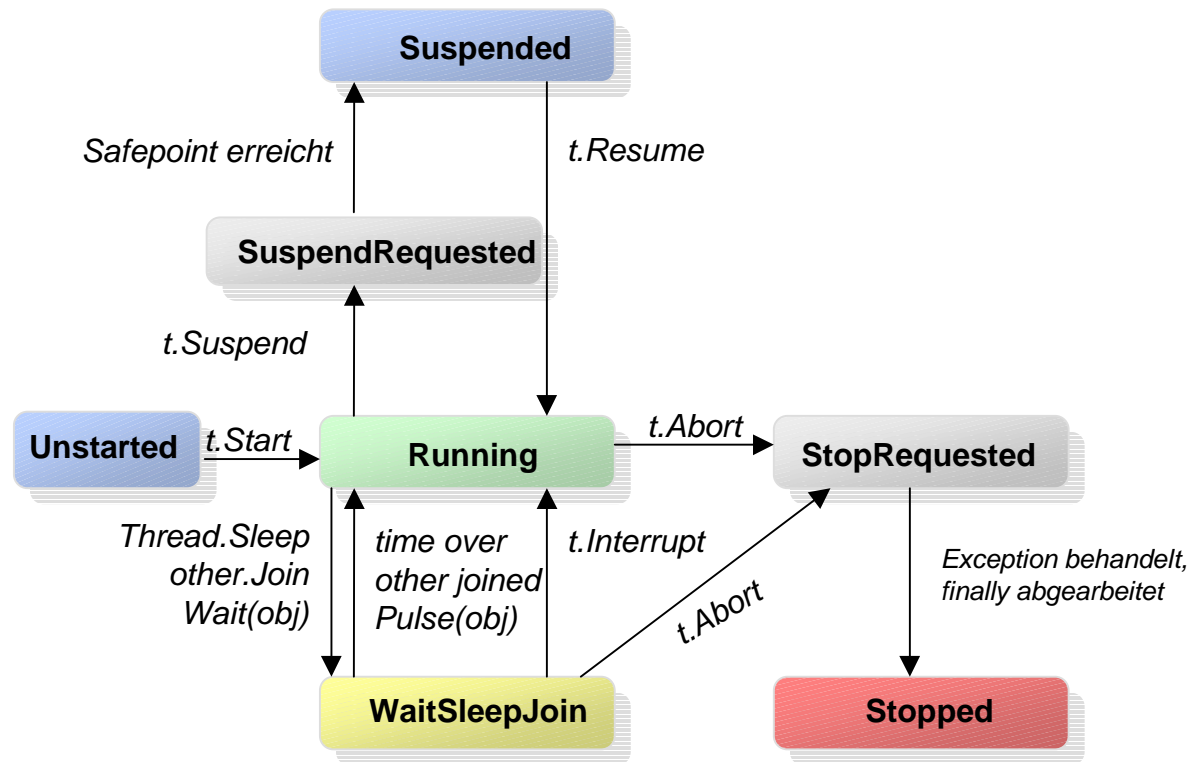
Thread mit Delegate zu Methode RunT0 erzeugen und starten

```
public static void main(string[] args) {
    // main Thread startet einen neuen Thread der RunT0 ausführt
    Thread t0 = new Thread(RunT0);
    t0.Start();
}
```

Thread Zustände

- Enumeration ThreadState definiert die möglichen Zustände eines Thread

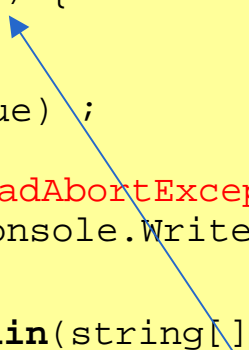
```
public enum ThreadState {  
    Background,  
    Running,  
    Stopped,  
    StopRequested,  
    Suspended,  
    SuspendRequested,  
    Unstarted,  
    WaitSleepJoin  
}
```



Behandeln von Abort

Abort löst Exception aus, die abgebrochener Thread behandeln kann.

```
using System; using System.Threading;
class Test {
    static void P() {
        try {
            while (true) ;
        }
        catch (ThreadAbortException) { Console.WriteLine("-- aborted"); }
        finally { Console.WriteLine("-- finally"); }
    }
    static void Main(string[] arg) {
        Thread t = new Thread(P);
        t.Start(); Thread.Sleep(1);
        t.Abort(); t.Join(); Console.WriteLine("done");
    }
}
```



Ausgabe

```
-- aborted
-- finally
done
```

Beispiel für Join

```
using System;
using System.Threading;

class Test {

    static void P() {
        for (int i = 1; i <= 20; i++) {
            Console.Write('-');
            Thread.Sleep(100);
        }
    }

    static void Main() {
        Thread t = new Thread(new ThreadStart(P));
        Console.Write("start");
        t.Start();
        t.Join(); // wartet auf t
        Console.WriteLine("end");
    }
}
```

Ausgabe

```
start-----end
```

lock-Anweisung

```
lock(Variable) Statement
```

Beispiel

```
class Account {           // diese Klasse stellt einen Monitor dar
    long val = 0;

    public void Deposit(long x) {
        lock (this) { val = val + x; } // 1 Thread darf diese Anweisung sein
    }

    public void Withdraw(long x) {
        lock (this) { val = val - x; }
    }
}
```

Lock lässt sich auch auf beliebiges anderes Objekt setzen

```
object semaphore = new object();
lock (semaphore) { ... critical region ... }
```

"synchronized" Methoden durch entsprechendes Attribut gesteuert

```
[MethodImpl(MethodImplOptions.Synchronized)]
public void Deposit(long x) {...}
```

Console Lock Beispiel

- lock-Anweisung für Synchronisierung von Tasks bei Zugriff auf gemeinsame Ressourcen
- lock-Anweisung setzt Sperre auf ein Objekt
- erreicht dadurch wechselseitiger Ausschluss

```
public class LockExample {  
    public static void RunT0() {  
        lock(Console.Out) {  
            for(int i = 0; i < 10; i++) {  
                // Konsole wird hier exklusiv verwendet  
                Console.Write("x");  
                Thread.Sleep(100);  
            }  
        }  
    }  
}
```

Collection Lock Beispiel

- lock-Anweisung für Synchronisierung von Tasks bei Zugriff auf gemeinsame Ressourcen
- lock-Anweisung setzt Sperre auf eine Collection

```
public class LockExample {  
    public void foo(ICollection collection {  
        if (!collection.IsSynchronized)  
            lock(collection.SyncRoot) {  
                //  
            }  
        }  
    }  
}
```

■ Klasse Monitor realisiert Basismechanismus für Synchronisierung

```
public sealed class Monitor {  
    public static void Enter(object obj);  
    public static bool TryEnter(object obj);  
  
    public static void Exit(object obj);  
  
    public static void Wait(object obj);  
    public static bool Pulse(object obj);  
    public static void PulseAll(object obj);  
}
```

- versucht Sperre für obj zu erhalten und blockiert
- versucht Sperre zu erhalten und blockiert *nicht*
- Gibt Sperre obj frei
- versetzt Thread in Wartezustand
- Weckt nächsten auf obj wartenden Thread auf
- Weckt alle auf obj wartenden Threads auf

■ lock-Anweisung wird mittels Monitor realisiert; ist Kurzform für folgende Anweisungsfolge:

```
lock (obj) {  
    ...  
}
```



```
Monitor.Enter(obj)  
try {  
    ...  
} finally {  
    Monitor.Exit(obj)  
}
```

Verwendung von Monitor

- Enter blockiert, wenn Sperre nicht verfügbar
- TryEnter versucht Sperre zu erhalten ohne zu blockieren (liefert false, wenn Sperre nicht verfügbar)

Enter: mit Blockierung

```
public class MonitorExample {  
    private Queue lpt;  
  
    public void AddElemBlocking  
    (object elem) {  
        try {  
            Monitor.Enter (lpt.SyncRoot);  
            lpt.Enqueue (elem);  
        } catch (Exception e) {  
            //Sollte nicht passieren  
        }  
        finally {  
            Monitor.Exit (lpt.SyncRoot);  
        }  
    }  
}
```

TryEnter: ohne Blockierung

```
public bool AddElemNonBlocking  
(object elem) {  
    try {  
        if (! Monitor.TryEnter (lpt.SyncRoot))  
            return false;  
        lpt.Enqueue (elem);  
    } catch (Exception e) {  
        //Sollte nicht passieren  
    }  
    finally {  
        Monitor.Exit (lpt.SyncRoot);  
    }  
    return true;  
}
```

Wait und Pulse

- Mit Wait und Pulse können sich Threads auf Basis eines Zustandes synchronisieren

```
public static void Wait(object obj);  
public static bool Wait(object obj, int millies);
```

Gibt Sperre für obj frei und wartet bis wieder aufgeweckt

```
public static bool Pulse(object obj);  
public static void PulseAll(object obj);
```

Weckt nächsten bzw. alle bei obj wartenden Threads auf

```
lock (obj) {  
    ...  
    Monitor.Wait(obj);  
    ...  
}
```

```
lock (obj) {  
    ...  
    Monitor.Pulse(obj);  
    ...  
}
```


Beispiel Wait und Pulse: RingBuffer

```
public class Buffer {  
    const int size = 16;  
    char[ ] buf = new char[size];  
    int head = 0, tail = 0, n = 0;  
  
    public void Put(char ch) {  
        lock(this) {  
            while (n >= size) Monitor.Wait(this);  
            buf[tail] = ch; tail = (tail + 1) % size;  
            n++;  
            Monitor.PulseAll(this);  
        }  
    }  
  
    public char Get() {  
        lock(this) {  
            while (n <= 0) Monitor.Wait(this);  
            char ch = buf[head]; head = (head + 1) % size;  
            n--;  
            Monitor.PulseAll(this);  
            return ch;  
        }  
    }  
}
```

Sperre Buffer, um ein Zeichen anzufügen
Wenn Buffer voll, gib Sperre frei und warte

Wecke alle wartenden Thread auf

Sperre Buffer, um Zeichen zu entnehmen
Wenn Buffer leer, gib Sperre frei und warte

Wecke alle wartenden Threads auf

Foreground und Background Threads

■ Zwei Arten von Threads: *Foreground* und *Background*

- So lange ein Foreground-Thread läuft wird ein Programm nicht beendet
- laufende Background-Threads können das Beenden eines Programms nicht verhindern

■ Erzeugen von Background-Thread durch Setzen von `IsBackground`

```
Thread bgThread = new Thread(new ThreadStart(...));  
bgThread.IsBackground = true;
```

Unterschiede zu Java



C#

```
void P() {  
    ... thread actions ...  
}  
  
Thread t = new Thread(new ThreadStart(P));
```

```
class MyThread extends Thread {  
    public void run() {  
        ... thread actions ...  
    }  
}  
  
Thread t = new MyThread();
```

- Erfordert keine Unterklasse von *Thread*
- **Beliebige Methode kann als Thread gestartet werden.**
- *Abort*-Methode => ***ThreadAbortException* Kann abgefangen werden,**
 - wird aber am Ende von catch automatisch wieder ausgelöst, ausser man ruft *ResetAbort* auf.
 - Alle finally-Blöcke werden ausgeführt, auch das Exit aus einem Monitor.

- Eigener Thread muss Unterklasse von *Thread* sein (oder *Runnable* implementieren).
- Thread-Aktionen müssen in Methode *run* stecken.
- *stop*-Methode ist *deprecated* weil gefährlich (gibt u.U. Monitor in inkonsistentem Zustand frei).

Verarbeitung von XML Daten

Was ist XML ?

- XML steht für "EXtensible Markup Language"
- XML ist eine **Markup Sprache** gleich wie HTML
- XML wurde entwickelt um Daten zu **beschreiben**
- XML Tags sind nicht vordefiniert in XML.
Tags müssen/können selber definiert werden → **Extensible**
- XML verwendet **Document Type Definition** (DTD) oder **XML Schema** um die Struktur der XML Dokumente zu beschreiben
- können mit Hilfe von CSS oder XSL dargestellt werden

markup

Syntactically delimited characters added to the data of a document to represent its structure.

HTML wurde entwickelt um Daten darzustellen

XML wurde entwickelt um Daten zu beschreiben

HTML vs. XML

HTML

HTML tags:
presentation, fixed

```
<h1>Wahlfächer</h1>
<h2>XML-Konzepte</h2>
<h3>Dozierende</h3>
<p>Peter Früh (<i>frp</i>),
Karl Rege (<i>Rea</i>)</p>
<h3>Inhalt</h3>
<ul>
  <li>XML Grundlagen
  <li>DTDs und Schemas
  <li>Namespaces
  <li>...
</ul>
<h2>Automatisierungstechnik</h2>
<h3>Dozierende</h3>
```

Wahlfächer

XML-Konzepte

Dozierende

Peter Früh (*frp*), Karl Rege (*Rea*)

Inhalt

- XML Grundlagen
- DTDs und Schemas
- Namespaces
- ...

Automatisierungstechnik

Dozierende

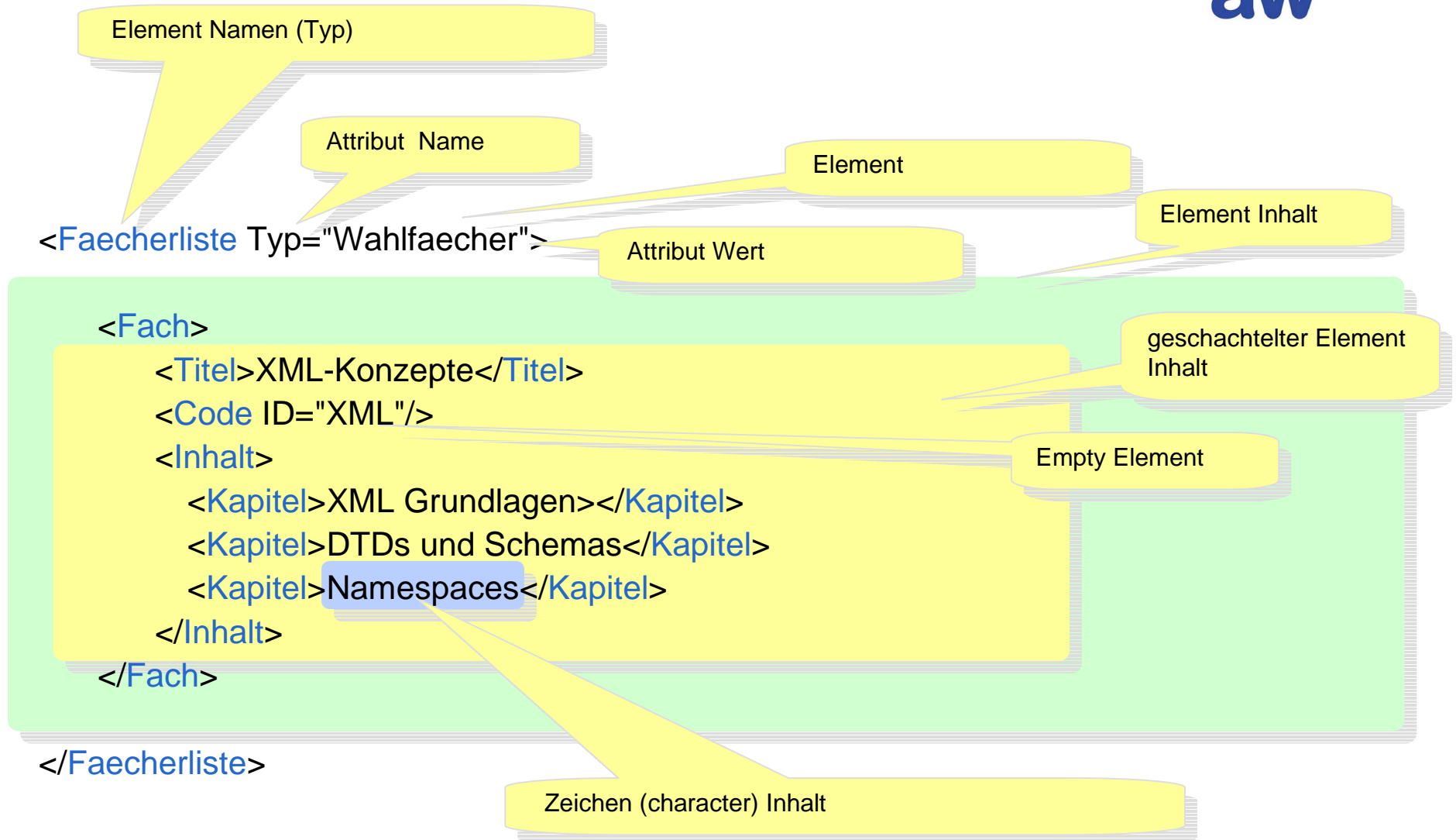
XML

XML tags:
content, (DTD-) specific

```
<Faecherliste Typ="Wahlfaecher">
  <Fach>
    <Titel>XML-Konzepte</Titel>
    <Dozenten>
      <Dozent>
        <Vorname>Peter</Vorname>
        <Nachname>Früh</Nachname>
        <Kurzzeichen>frp</Kurzzeichen>
      </Dozent>
      <Dozent>
        <Vorname>Karl</Vorname>
        <Nachname>Rege</Nachname>
        <Kurzzeichen>rea</Kurzzeichen>
      </Dozent>
    </Dozenten>
    <Inhalt>
      <Kapitel>XML Grundlagen</Kapitel>
      <Kapitel>DTDs und Schemas</Kapitel>
      <Kapitel>Namespaces</Kapitel>
      <Kapitel>...</Kapitel>
    </Inhalt>
  </Fach>
  <Fach>
    <Titel>Automatisierungstechnik</Titel>
    <Dozenten>...</Dozenten>
  </Fach>
</Faecherliste>
```

```
<Faecherliste Typ="Wahlfaecher">
  - <Fach>
    <Titel>XML-Konzepte</Titel>
    - <Dozenten>
      - <Dozent>
        <Vorname>Peter</Vorname>
        <Nachname>Frueh</Nachname>
        <Kurzzeichen>frp</Kurzzeichen>
      </Dozent>
      - <Dozent>
        <Vorname>Karl</Vorname>
        <Nachname>Rege</Nachname>
        <Kurzzeichen>rea</Kurzzeichen>
      </Dozent>
    </Dozenten>
    - <Inhalt>
      <Kapitel>XML Grundlagen</Kapitel>
      <Kapitel>DTDs und Schemas</Kapitel>
      <Kapitel>Namespaces</Kapitel>
      <Kapitel>...</Kapitel>
    </Inhalt>
  </Fach>
  - <Fach>
    <Titel>Automatisierungstechnik</Titel>
    <Dozenten> </Dozenten>
  </Fach>
</Faecherliste>
```

Elemente und Inhalt



XML Syntax: Elemente

- Alle XML Elemente haben öffnendes und schliessendes Tag

```
<para>This is a paragraph</para>  
<para>This is another paragraph</para>
```

- Ein leeres Element kann abgekürzt werden

```
<paid/>
```

is gleichwertig

```
<paid></paid>
```

- Gross-/Kleinschreibung wird bei Tags berücksichtigt (im Gegensatz zu HTML)

- Alle XML Elemente müssen richtig eingebettet sein

```
<person><name>Donald Duck</name></person>
```


XML Syntax: Dokumentstruktur

- XML Dokumente haben genau ein Wurzel (root) Element

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

- XML Dokumente können Kommentare enthalten

```
<!-- This is a comment -->
```

- XML Dokumente dürfen gewisse Zeichen nicht im Zeicheninhalt enthalten
-> durch & ... ; Sequenz ersetzen

<	<
>	>
&	&
'	'
"	"

- XML-Dokumente können (sollten) mit einem Prolog beginnen

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Es existiert nur die
Version 1.0

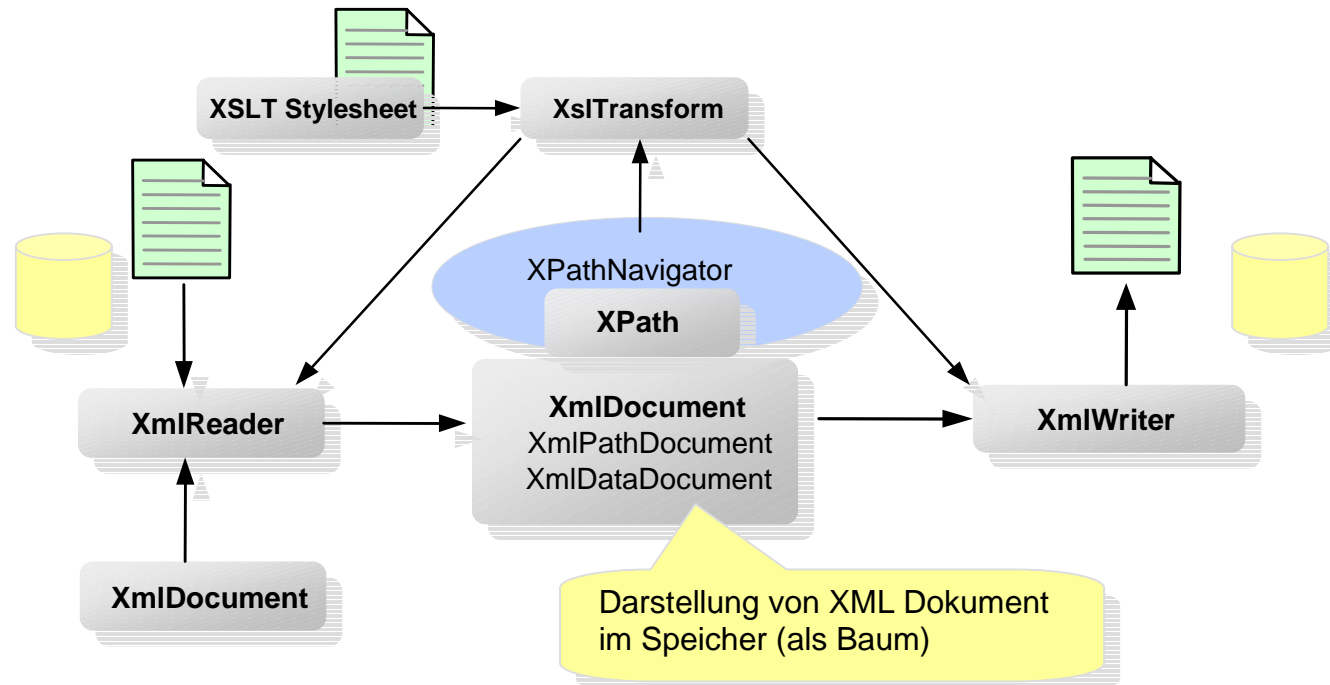
Bechreibt die verwendete
Buchstabenkodierung

- Default ist UTF-8
- ISO-8859-1 (Latin-1) oft
verwendet (Umlaute)

zeigt an, dass im
Dokument keine
Referenzen zu
anderen Dokumenten

- .NET stützt sich stark auf XML ab
 - siehe ADO.NET, WSDL, UDDI, SOAP, ...
- Die Klassenbibliothek bietet Implementierungen von Standards wie:
 - XML, XSL, XPath, ...
- Unterstützt werden die beiden Verarbeitungsmodi
 - **In-Memory**: DOM (Document Object Model)
 - **Streaming**: Serieller Zugriff ähnlich zu SAX
- Namensräume
 - System.Xml
 - System.Xml.Xsl
 - System.Xml.XPath
 - System.Xml.Schema
 - System.Xml.Serialization

Überblick: Verarbeitung von XML-Daten



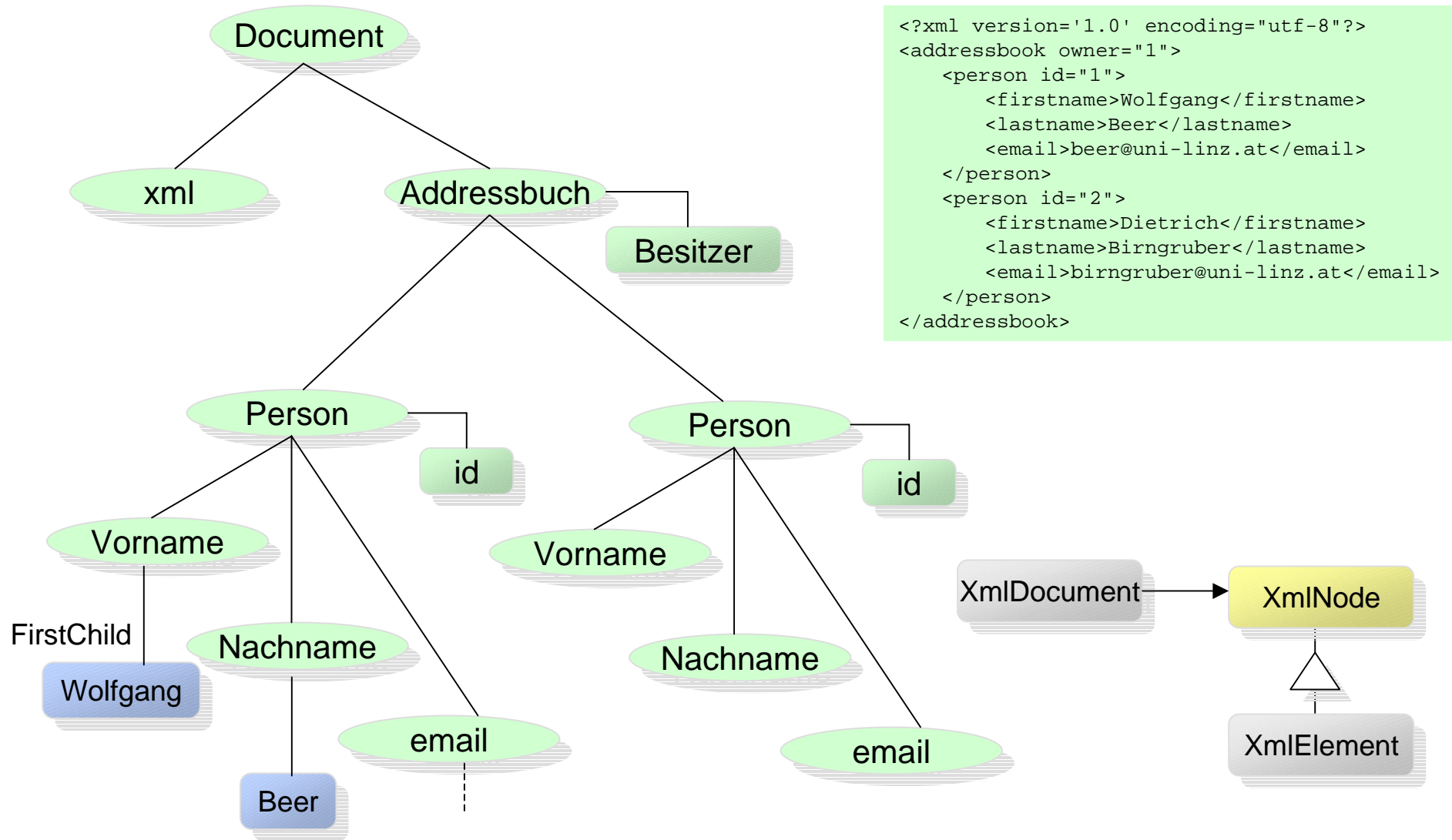
- **XmlDocument, XmlNode**: Objektmodell von XML-Daten (DOM)
- **XmlReader**: Lesen von XML-Daten
- **XmlWriter**: Schreiben von XML-Daten
- **XPathNavigator**: XPath-Abfragen
- **XsltTransform**: Transformation von XML-Dokumenten

- Aufbau einer Datenstruktur im Speicher
 - + Bequeme Editierbarkeit von XML-Daten
 - Grössenlimit für XML-Daten (alles im Hauptspeicher)
 - langsam, wenn sehr grosse Dateien
- XML-Elemente durch XmlNode-Objekte repräsentiert.
- XmlDocument repräsentiert ganzes Dokument und ermöglicht Verarbeiten

z.B.: Laden eines XML-Documents in den Hauptspeicher:

```
XmlDocument xDoc = new XmlDocument();  
xDoc.Load( "datei.xml" );
```

Beispiel DOM



Klasse XmlNode - lesen

```
public abstract class XmlNode : ICloneable, IEnumerable,
IXPathNavigable {

    Node FirstChild {get;}
    public abstract string Name { get; }
    public abstract string LocalName { get; }
    public abstract XmlNodeType NodeType { get; }
    public virtual string Value { get; set; }
    public virtual XmlAttributeCollection Attributes { get; }
    public virtual XmlDocument OwnerDocument { get; }
    public virtual bool IsReadOnly { get; }
    public virtual bool HasChildNodes { get; }
    public virtual string Prefix { get; set; }

    public virtual XmlNodeList ChildNodes { get; }
    public virtual XmlNode FirstChild { get; }
    public virtual XmlNode LastChild { get; }
    public virtual XmlNode NextSibling { get; }
    public virtual XmlNode PreviousSibling { get; }
    public virtual XmlNode ParentNode { get; }
    public virtual XmlElement this[string name] { get; }
    public virtual XmlElement this[string localname, string ns] {
get; }
    public virtual XmlNodeList GetElementsByTagName(string localname);
    ...
}
```

■ Eigenschaften des Knotens

- voller Name
- lokaler Name
- Typ
- Wert
- Attribute
- ...

■ Zugriff auf weitere Knoten

- Children
- Siblings
- Parent

- benannte Knoten

Klasse XmlNode -schreiben

```
...
public virtual XmlNode AppendChild(XmlNode newChild);
public virtual XmlNode PrependChild(XmlNode newChild);
public virtual XmlNode InsertAfter(XmlNode newChild,
    XmlNode refChild);
public virtual XmlNode InsertBefore(XmlNode newChild,
    XmlNode refChild);
public virtual XmlNode RemoveChild(XmlNode oldChild);
public virtual void RemoveAll();

public XPathNavigator CreateNavigator();
public XmlNodeList SelectNodes(string xpath);
public XmlNode SelectSingleNode(string xpath);

public abstract void WriteContentTo(XmlWriter w);
public abstract void WriteTo(XmlWriter w);
...
}
```

■ Anfügen und Löschen
neuer Knoten

■ Selektion von Knoten

■ Schreiben

```
public enum XmlNodeType {
    Attribute, CDATA, Comment, Document, DocumentFragment, DocumentType, Element,
    EndElement, EndEntity, Entity, EntityReference, None, Notation, ProcessingInstruction,
    SignificantWhitespace, Text, Whitespace, XmlDeclaration
}
```


Klasse XmlDocument laden/speichern

```
public class XmlDocument : XmlNode {  
  
    public XmlDocument();  
  
    public XmlElement DocumentElement { get; }  
    public virtual XmlDocumentType DocumentType { get; }  
  
    public virtual void Load(Stream in);  
    public virtual void Load(string url);  
    public virtual void LoadXml(string data);  
  
    public virtual void Save(Stream out);  
    public virtual void Save(string url);  
    ...  
}
```

■ Root-Element

■ Dokumenttyp

■ Laden von XML-Daten

■ Speichern

Klasse XmlDocument - "Fabrikmuster"

```
....  
public virtual XmlDeclaration CreateXmlDeclaration  
    (string version, string encoding, string standalone);  
public XmlElement CreateElement(string name);  
public XmlElement CreateElement  
    (string qualifiedName, string namespaceURI);  
public virtual XmlElement CreateElement  
    (string prefix, string lName, string nsURI);  
public virtual XmlText CreateTextNode(string text);  
public virtual XmlComment CreateComment(string data);  
}
```

■ Erzeugen von

■ Deklaration

■ Elementen

■ Textknoten

■ Kommentaren

Beispiel: Lesen von Nachnamen aus XML-Dokument

- Dokument laden
- XmlDocument ermöglicht XML-Dokumente zu verarbeiten

```
XmlDocument doc = new XmlDocument();  
doc.Load("addressbook.xml");
```

- lastname-Elemente suchen

```
XmlElement root = doc.DocumentElement;  
XmlNodeList list = root.GetElementsByTagName("lastname");
```

- Person-Element Text Inhalt ausgeben

```
foreach(XmlNode n in list) {  
    Console.WriteLine(n.FirstChild.Value);  
}
```

Beer, Birngruber, Moessenboeck, Woess,

Beispiel: Erzeugen von XML-Dokumenten

- XmlDocument ermöglicht XML-Dokumente aufzubauen
- Dokument erzeugen und Deklaration anfügen

```
XmlDocument doc = new XmlDocument();  
XmlDeclaration decl = doc.CreateXmlDeclaration("1.0", null, null);  
doc.AppendChild(decl);
```

- Wurzelement erzeugen

```
XmlElement rootElem = doc.CreateElement("addressbook");  
rootElem.SetAttribute("owner", "1");  
doc.AppendChild(rootElem);
```

- Person-Element und Unter Elemente erzeugen und anfügen

```
XmlElement person = doc.CreateElement("person");  
person.SetAttribute("id", "1");  
XmlElement e = doc.CreateElement("firstname");  
e.AppendChild(doc.CreateTextNode("Wolfgang"));  
person.AppendChild(e);  
e = doc.CreateElement("lastname");  
...  
...
```

```
<?xml version="1.0" encoding="iso-8859-1"?>  
<addressbook owner="1">  
  <person id="1">  
    <firstname>Wolfgang</firstname>  
    <lastname>Beer</lastname>  
    <email>beer@uni-linz.at</email>  
  </person>  
</addressbook>
```

- *XPath* ist ein Adressierungsmechanismus für Elemente in XML-Dokumenten
- XPath-Ausdruck (*Location path*) selektiert eine Menge von Knoten
- Ein *Location path* besteht aus *Location steps*, die mit "/" getrennt sind

/step/step/step/

Beispiele von Location path:

"*"

Selektiert alle Knoten

"/addressbook/*"

Selektiert alle Elemente unter einen addressbook-Element

"/addressbook/person[1]"

Liefert die ersten person-Elemente von addressbook-Elementen

"/addressbook/*/firstname"

Liefert die firstname-Element unter den addressbook-Elementen

- Klasse XPathNavigator, stellt eine XPath-Abfrageschnittstelle bereit

```
public abstract class XPathNavigator : ICloneable {  
  
    public abstract string Name { get; }  
    public abstract string Value { get; }  
    public abstract bool HasAttributes { get; }  
    public abstract bool HasChildren { get; }  
  
    public virtual XPathNodeIterator Select(string xpath);  
    public virtual XPathNodeIterator Select  
        (XPathExpression expr);  
    public virtual XPathExpression Compile(string xpath);  
  
    public abstract bool MoveToNext();  
    public abstract bool MoveToFirstChild();  
    public abstract bool MoveToParent();  
    ...  
}
```

- Eigenschaften des aktuellen Knotens
- Selektion von Knoten mit XPath
- Kompilieren eines XPath
- Springen zu einem benachbarten Knoten

- XPathNavigable (von XmlNode implementiert) liefert XPathNavigator

```
public interface XPathNavigable {  
    XPathNavigator CreateNavigator();  
}
```

Beispiel XPathNavigator

■ XmlDocument laden und XPathNavigator erzeugen

```
XmlDocument doc = new XmlDocument();  
doc.Load("addressbook.xml");  
XPathNavigator nav = doc.CreateNavigator();
```

■ lastname-Elemente selektieren, über selektierte Elemente iterieren und Namen ausgeben

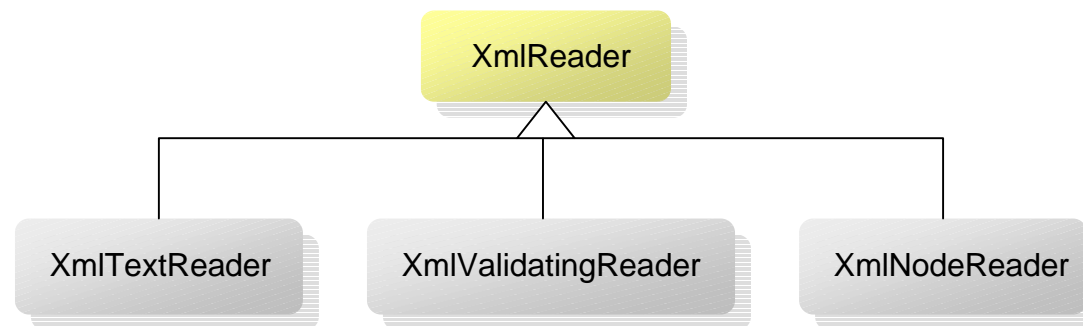
```
XPathNodeIterator iterator = nav.Select("/addressbook/*/lastname");  
while (iterator.MoveNext())  
    Console.WriteLine(iterator.Current.Value);
```

■ Für bessere Laufzeit Ausdruck kompilieren und kompilierten Ausdruck anwenden

```
XPathExpression expr = nav.Compile("/addressbook/*/lastname");  
iterator = nav.Select(expr);  
while (iterator.MoveNext()) Console.WriteLine(iterator.Current.Value);
```

Beer, Birngruber, Moessenboeck, Woess,

- XmlReader für streaming Parsen
- ähnlich zu SAX, arbeitet aber nach der *Pull*-Methode.
- Implementierungen sind:
 - XmlTextReader: performant, keine Zwischenspeicherung
 - XmlValidatingReader: validiert auf DTD oder XSD
 - XmlNodeReader: XML-Datenzugriff ausgehend von einem XmlNode



Klasse XmlReader

```
public abstract class XmlReader {  
  
    public abstract string Name { get; }  
    public abstract string LocalName { get; }  
    public abstract string Value { get; }  
    public abstract XmlNodeType NodeType { get; }  
    public abstract int AttributeCount { get; }  
    public abstract int Depth { get; }  
  
    public abstract bool Read();  
    public abstract bool IsStartElement(string s);  
  
    public virtual void Skip();  
    public abstract string GetAttribute(int i);  
  
    public abstract void Close();  
    ...  
}
```

■ Eigenschaften des aktuellen Elements

- voller Name
- lokaler Name
- Wert
- Typ
- Anzahl der Attribute
- Tiefe im Dokument

■ Lesen des nächsten Elements

- Überspringen des Elements
- Zugriff auf die Attribute

■ Schliessen des Readers

Beispiel XmlTextReader

- Lesen der Datei addressbook.xml
- Ausgabe aller Werte der lastname-Elemente

```
XmlReader r;  
r = new XmlTextReader("addressbook.xml");  
while (r.Read()) {  
    if (r.IsStartElement("lastname")) {  
        r.Read(); // read the name  
        Console.WriteLine("{0}, ", r.Value);  
    }  
}  
r.Close();
```

- Ausgabe

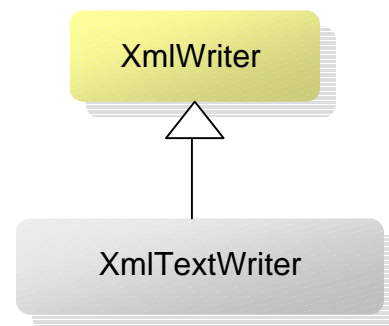
Beer, Birngruber, Moessenboeck, Woess,

XML-Datei

```
<?xml version='1.0' encoding="iso-8859-1"?>  
<addressbook owner="1">  
    <person id="1">  
        <firstname>Wolfgang</firstname>  
        <lastname>Beer</lastname>  
        <email>beer@uni-linz.at</email>  
    </person>  
    <person id="2">  
        <firstname>Dietrich</firstname>  
        <lastname>Birngruber</lastname>  
        <email>birngruber@uni-linz.at</email>  
    </person>  
    <person id="3">  
        <firstname>Hanspeter</firstname>  
        <lastname>Moessenboeck</lastname>  
        <email>moessenboeck@uni-linz.at</email>  
    </person>  
    <person id="4">  
        <firstname>Albrecht</firstname>  
        <lastname>Woess</lastname>  
        <email>woess@uni-linz.at</email>  
    </person>  
</addressbook>
```

Klasse XmlWriter

- XmlWriter für erstellen von XML Documenten
- ähnlich zu SAX (auch push)
- Implementierungen sind:
 - XmlTextWriter: performant, keine Zwischenspeicherung



Klasse XmlWriter

```
public abstract class XmlWriter {  
  
    public abstract void WriteStartDocument(bool);  
    public abstract void WriteEndDocument();  
  
    public abstract void WriteStartElement(string n);  
    public abstract void WriteEndElement();  
  
    public abstract void WriteAttributeString  
        (string name, string value);  
  
    public abstract void Close();  
    ...  
}
```

- Am Anfang
- Am Ende

- Beginne neues Element
- Schliesse Element ab

- schreibe Attribut

- Schliessen des Writers

Beispiel XMLWriter

■ Erzeuge ein einfaches XML Dokument

```
FileStream fs = new FileStream("c:\\tmp\\Test.xml", FileMode.Create);
XmlWriter xwriter = new XmlTextWriter(fs, Encoding.GetEncoding("iso-8859-1"));
xwriter.WriteStartDocument(true); // stand alone

xwriter.WriteStartElement("adressBook");
xwriter.WriteStartElement("person");
xwriter.WriteAttributeString("id", "1");
xwriter.WriteStartElement("firstName");
xwriter.WriteString("Wolfgang");
.....
xwriter.WriteEndElement(); // firstName
xwriter.WriteEndElement(); // person
xwriter.WriteEndElement(); // adressBook

xwriter.WriteEndDocument();
xwriter.Close();
fs.Close();
```

```
<?xml version="1.0" encoding="iso-8859-1"
standalone = "true" ?>
<addressbook owner="1">
  <person id="1">
    <firstname>Wolfgang</firstname>
    <lastname>Beer</lastname>
    <email>beer@uni-linz.at</email>
  </person>
</addressbook>
```

Serialisierung von Objekten: XmlSerializer

volle Kontrolle über XML
Serialisierungsformat; durch
Attribute gesteuert

```
static readonly Encoding encoding = Encoding.GetEncoding("i
```

```
/* Methode zum Serialisieren eines Objekts in eine XML-Datei */  
public static void SerializeToXmlFile(string fileName, object obj) {  
    // XmlSerializer für den Typ des Objekts erzeugen  
    XmlSerializer serializer = new XmlSerializer(obj.GetType());  
    FileStream fs = new FileStream(fileName, FileMode.Create);  
    // Objekt über ein StreamWriter-Objekt serialisieren  
    StreamWriter streamWriter = new StreamWriter(fs, encoding);  
    serializer.Serialize(streamWriter, obj);  
    streamWriter.Close();  
    fs.Close();  
}
```

```
/* Methode zum Deserialisieren eines Objekts aus einer XML-Datei */  
public static object DeserializeFromXmlFile(string fileName, Type objectType) {  
    // XmlSerializer für den Typ des Objekts erzeugen  
    XmlSerializer serializer = new XmlSerializer(objectType);  
    // Objekt über ein StreamReader-Objekt serialisieren  
    StreamReader streamReader = new StreamReader(fileName, encoding);  
    object o = serializer.Deserialize(streamReader);  
    streamReader.Close();  
    return o;  
}
```

```
SerializeToXmlFile("c:\\test.xml", address);  
address = (Address)DeserializeFromXmlFile("c:\\test.cml", typeof(Address));
```

Serialisierung von Objekten: SoapFormatter

Einfacher in der
Anwendung

```
...  
using System.Runtime.Serialization;  
using System.Runtime.Serialization.Formatters.Soap;
```

```
class Test {
```

```
    static void Write(IList head) {  
        FileStream s = new FileStream("MyFile", FileMode.Create);  
        IFormatter f = new SoapFormatter();  
        f.Serialize(s, head);  
        s.Close();  
    }
```

```
    static IList Read() {  
        FileStream s = new FileStream("MyFile", FileMode.Open);  
        IFormatter f = new SoapFormatter();  
        IList head = f.Deserialize(s) as IList;  
        s.Close();  
        return head;  
    }
```

```
    static void Main() {  
        Write(list);  
        IList newList = Read();
```

```
}
```

- XSLT ist eine Sprache zur Beschreibung einer Transformation von XML-Dokumenten
- *XSL-Stylesheet* ist ein Dokument mit einer Menge von Regeln
- Regeln (*templates*) beschreiben die Transformation von XML-Elementen
- XSLT verwendet XPath: z.B. XPath-Ausdrücke definieren die Prämissen der Regeln (*match*), d.h bestimmen wann die Regeln angewandt werden
- im Rumpf der Regel wird die Generierung des transformierten Elements beschrieben

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  <xsl:template  
    match=xpath-expression  
      construction of transformed elements  
  </xsl:template>  
</xsl:stylesheet>
```


Beispiel XSL-Stylesheet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:template match="/addressbook">
    <html>
      <head>
        <title>XML Address Book</title>
      </head>
      <body>
        <table border="3" cellspacing="10" cellpadding="5">
          <xsl:apply-templates select="person"/>
        </table>
      </body>
    </html>
  </xsl:template>
```

```
  <xsl:template match="person">
    <tr>
      <td><xsl:value-of select="firstname"/></td>
      <td><b><xsl:value-of select="lastname"/></b></td>
      <td><xsl:value-of select="email"/></td>
    </tr>
  </xsl:template>
```

```
</xsl:stylesheet>
```

Beispiel Transformation

■ ursprüngliches XML-Dokument

```
<?xml version='1.0' encoding="utf-8"?>
<addressbook owner="1">
  <person id="1">
    <firstname>Wolfgang</firstname>
    <lastname>Beer</lastname>
    <email>beer@uni-linz.at</email>
  </person>
  <person id="2">
    <firstname>Dietrich</firstname>
    <lastname>Birngruber</lastname>
    <email>birngruber@uni-linz.at</email>
  </person>
</addressbook>
```

■ erzeugtes HTML-Dokument

```
<html>
  <head>
    <META http-equiv="Content-Type"
    content="text/html; charset=utf-8">
    <title>XML-AddressBook</title>
  </head>
  <body>
    <table border="3" cellspacing="10" cellpadding="5">
      <tr>
        <td>Wolfgang</td>
        <td><b>Beer</b></td>
        <td>beer@uni-linz.at</td>
      </tr>
      <tr>
        <td>Dietrich</td>
        <td><b>Birngruber</b></td>
        <td>birngruber@uni-linz.at</td>
      </tr>
    </table>
  </body>
</html>
```

Klasse XslTransform

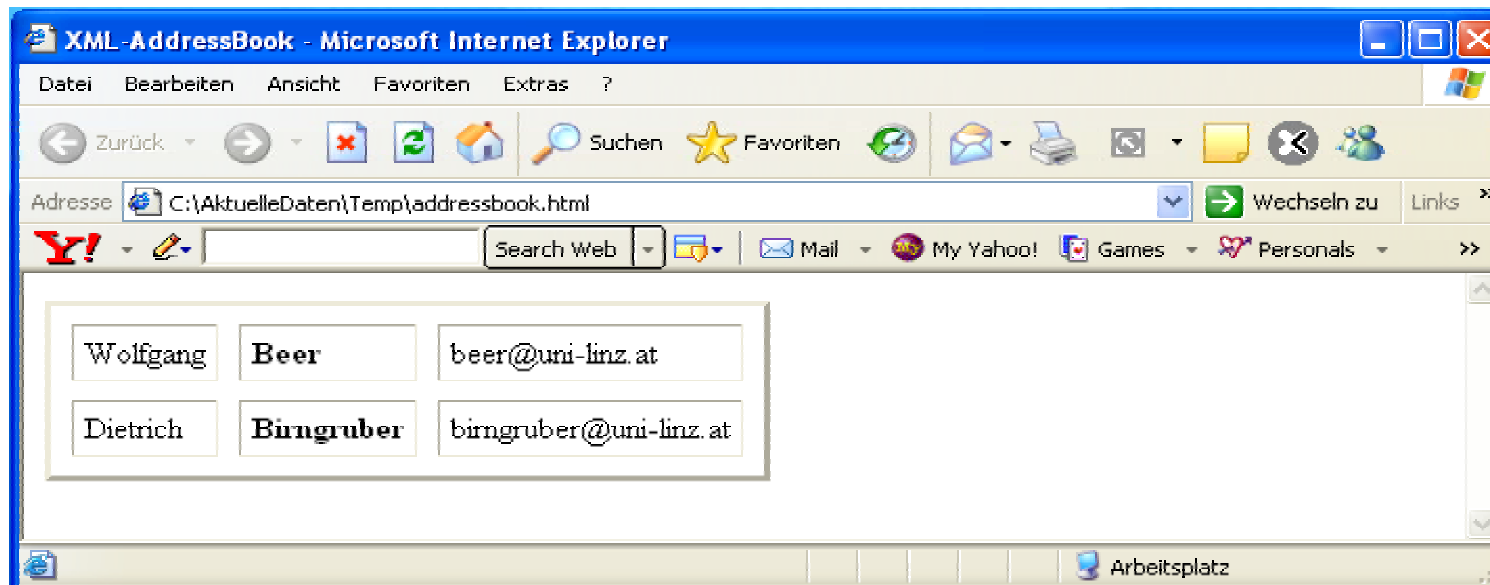
- Namensraum System.Xml.Xsl bietet Unterstützung für XSLT
- Klasse XslTransform realisiert XSL-Transformation

```
public class XslTransform {  
  
    public void Load(string url);  
  
    public XslTransform();  
    public void Transform(string infile,  
        string outfile, XmlResolver resolver);  
  
    ... // + overloaded methodds Load and Transform  
}
```

- Laden einer XSL-Stylesheet-Datei
- Transformieren

XML Transformation mit XSL

```
XslTransform xt = new XslTransform();  
xt.Load("addressbook.xsl");  
xt.Transform("addressbook.xml", "addressbook.html", new XmlUrlResolver());
```



Networking

- Namensraum System.Net zur Erstellung von typischen Klienten-Server Applikationen.
- System.Net bietet Implementierungen von:
 - Internetprotokollen, wie z.B.: TCP, UDP, HTTP
 - Internetdiensten, wie z.B.: DNS (Domain Name System)
 - diversen Protokollen, wie z.B.: IrDA
- System.Net.Sockets bietet Klassen für Datenstrom orientierte Kommunikation über Sockets

- Addressierung erfolgt über Klassen

IPAddress: repräsentiert einen IP-Adresse

EndPoint: repräsentiert einen Endpunkt mit IP-Adresse und Portnummer

Beispiel:

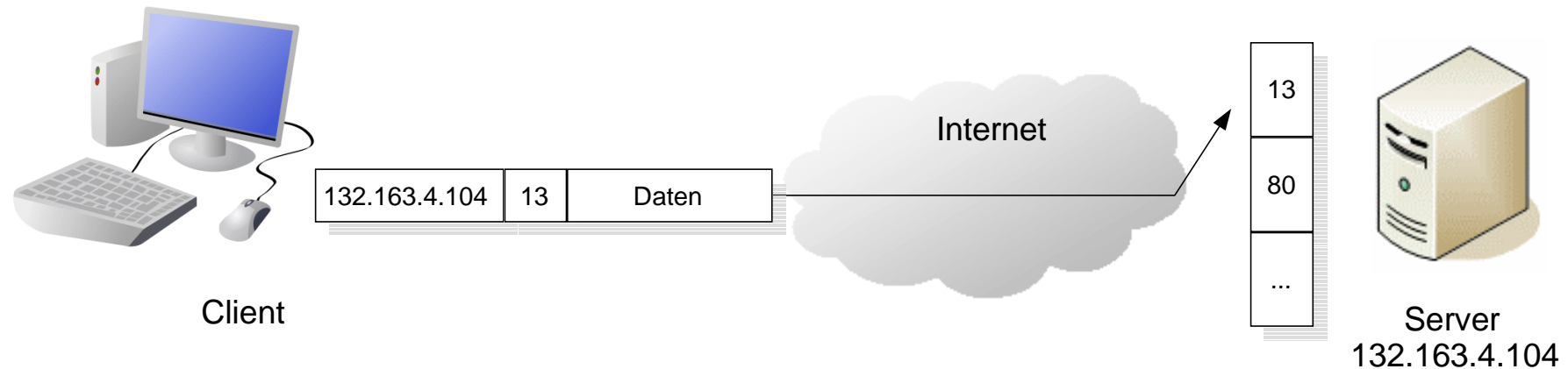
```
IPAddress ipAdr = new IPAddress("254.10.120.3");  
// Erstellen eines Endpunktes mit Portnummer 80 (HTTP)  
EndPoint ep = new EndPoint(ipAdr, 80);
```

DNS (Domain Name System)

- DNS ermöglicht Übersetzung von Adressen in sprechende Namen
- Die Klasse Dns hilft bei der Übersetzung von Namen in Adressen
- Die Klasse IPHostEntry stellt dabei eine Containerklasse für Adressinformationen bereit

Beispiel:

```
// Anzeigen aller Adressen die einem Domainnamen zugeordnet sind
IPHostEntry host = Dns.Resolve("dotnet.jku.at");
foreach (IPAddress ip in host.AddressList)
    Console.WriteLine(ip.ToString());
```

- Sockets stellen bidirektionale Kommunikationskanäle dar, durch die Daten gesendet und empfangen werden können
- Client/Server Architektur
 - Client schickt Anforderungen an Server
 - Server behandelt Anforderung und
 - schickt Antwort zurück
- Adressierung über IP-Adressen und Ports
- Austausch von Daten über Streams (siehe Streaming)

Client

Socket und Endpunkt für Client erzeugen

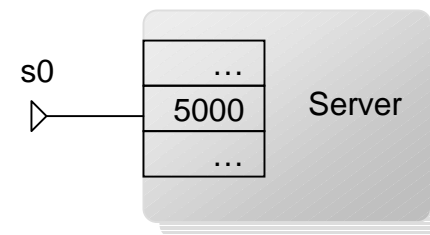
```
Socket s2 = new Socket();  
IPAddress ip = IPAddress.Parse(...);  
IPEndPoint ep = new IPEndPoint(ip, 5000);
```



Server

- Socket erzeugen und an Endpunkt binden
- Socket für max. gepufferte 10 Klienten öffnen

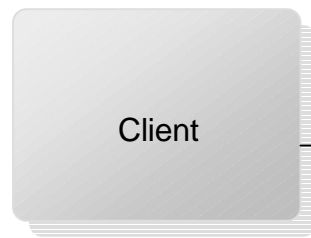
```
Socket s0 = new Socket();  
IPAddress ip = IPAddress.parse(...);  
IPEndPoint ep = new IPEndPoint(ip, 5000);  
s0.bind(ep);  
s0.Listen(10);
```



Verbindungsaufname & Datenaustausch

Verbindung mit Endpunkt aufnehmen

```
s2.Connect(ep);
```

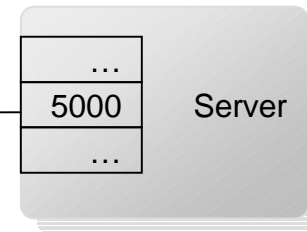


s2

Auf Verbindung warten

```
Socket s1 = s0.Accept();
```

s0
s1



Mit Server kommunizieren, dann
Verbindung trennen

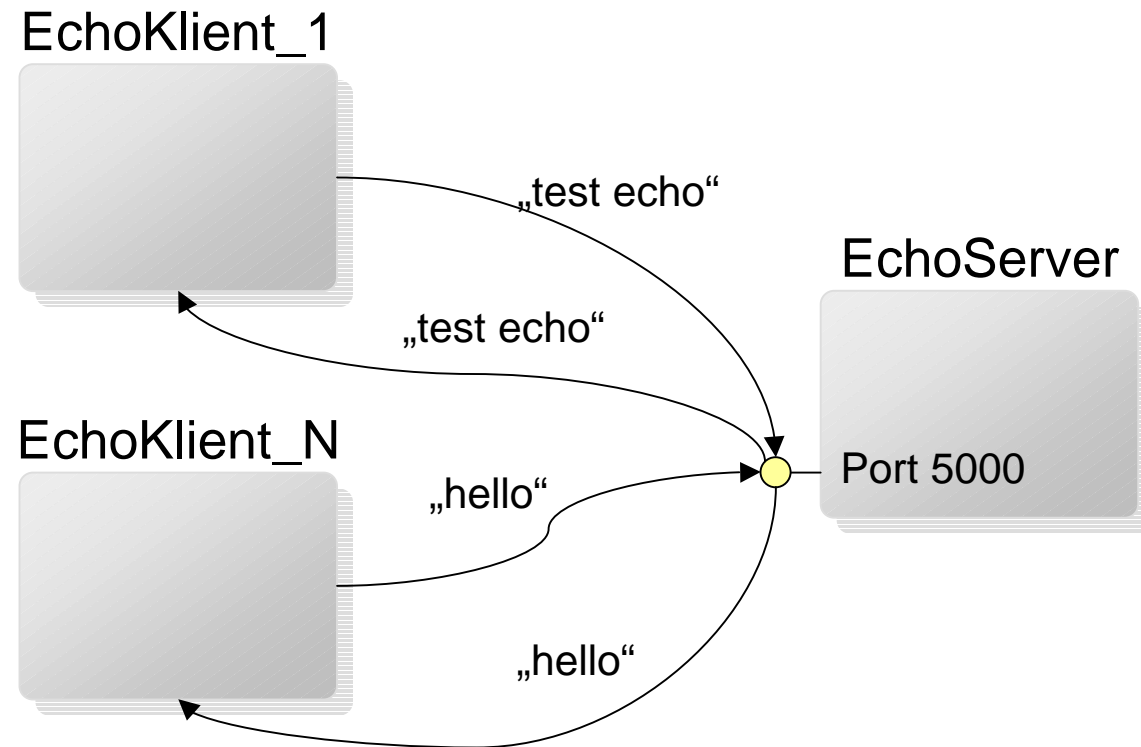
```
s2.Send(msg1);  
...  
s2.Receive(msg2);  
s2.Shutdown(SocketShutdown.Both);  
s2.Close();
```

Mit Client kommunizieren, dann
Verbindung trennen

```
s1.Receive(msg1);  
...  
s1.Send(msg2);  
s1.Shutdown(SocketShutdown.Both);  
s1.Close();
```

Beispiel EchoServer

- implementiert eine einfache Client-Server-Anwendung
- Der EchoServer übernimmt beliebige Daten von einem Klienten
- und schickt diese unverändert an den Client zurück.



Beispiel EchoServer: Klasse EchoServer (1)

```
class EchoServer {
    socket s;

    public bool StartUp(IPAddress ip, int port) {
        try {
            s = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                ProtocolType.Tcp);
            s.Bind(new IPEndPoint(ip, port));
            s.Listen(10); // maximal 10 Klienten in Queue
        } catch (Exception e) { ... }
        for(;;) {
            Communicate(s.Accept()); // wartet auf sich verbindende Klienten
        }
    }
}
```

Beispiel EchoServer: Klasse EchoServer (2)

```
class EchoServer {  
    ...  
    // sendet alle empfangenen Daten wieder an den Klienten zurück  
    public void Communicate(Socket clSock) {  
        try {  
            byte[] buffer = new byte[1024];  
            while (clSock.Receive(buffer) > 0) // Daten empfangen  
                clSock.Send(buffer); // Daten zurückschicken  
            clSock.Shutdown(SocketShutdown.Both); // Sockets schliessen  
            clSock.Close();  
        } catch (Exception e) { ... }  
    }  
  
    public static void Main() {  
        EchoServer = new EchoServer();  
        server.StartUp(IPAddress.Loopback, 5000); // starten des Servers  
    }  
}
```

Beispiel EchoServer: Klasse EchoClient

```
class EchoClient {  
    ...  
    public static void Main() {  
        try {  
            // Verbindung zum Server herstellen  
            Socket s = new Socket( AddressFamily.InterNetwork,  
                                   SocketType.Stream, ProtocolType.Tcp);  
            s.Connect(new InetEndPoint(IPAddress.Loopback, 5000));  
            s.Send( Encoding.ASCII.GetBytes("This is a test")); // message senden  
            byte[] echo = new byte[1024];  
            s.Receive(echo); // echo empfangen  
            Console.WriteLine(Encoding.ASCII.GetString(echo));  
        } catch (Exception e) { ... }  
    }  
}
```

- Sockets bieten Schnittstelle für Übertragung von byte oder byte-Arrays
- Klasse NetworkStream stellt Stream für Lesen und Schreiben auf Sockets dar
- Reader und Writer können somit für formatiertes Lesen und Schreiben verwendet werden
- Z.B liest XmlTextReader Daten im XML-Format

Beispiel NetworkStream und XmlTextReader

■ Socket definieren und zum Endpunkt verbinden

```
Socket s = new Socket(...);  
s.Connect( new IPEndPoint(ip, port));
```

■ NetworkStream für Socket erzeugen

```
NetworkStream ns = new NetworkStream(s);
```

■ XmlTextReader für NetworkStream erzeugen

```
XmlTextReader r = new XmlTextReader(ns);
```

■ XML-Daten lesen

```
for (int i = 0; i < r.AttributeCount; i++) {  
    r.MoveToAttribute();  
    Console.WriteLine("{0} = {1}", r.Name, r.Value);  
}
```

WebRequest und WebResponse

- Für Laden von Ressourcen aus dem Web
- Abstrakte Klassen mit konkreten Implementierungen:

HttpRequest und HttpResponse

➔ *Übertragung mittels HTTP Protokoll*

FileHttpRequest und FileHttpResponse

➔ *Übertragung mittels Microsoft File Protokoll*

Klassen WebRequest und WebResponse

```
public abstract class WebRequest {  
    public static WebRequest Create(string uri);  
  
    public virtual string Method { get; set; }  
  
    public virtual string ContentType { get; set; }  
    public virtual WebHeaderCollection Headers { get; set; }  
  
    public virtual Stream GetRequestStream();  
    public virtual WebResponse GetResponse();  
    ...  
}
```

■ Webrequest für URI erzeugen

■ HTTP-Method-Typ (GET oder POST)

■ Mime-Type
■ Headers

■ Stream, um Request zu schreiben

■ Response-Objekt

```
public abstract class WebResponse {  
    public virtual long ContentLength { get; set; }  
  
    public virtual string ContentType { get; set; }  
    public virtual WebHeaderCollection  
        Headers { get; set; }  
  
    public virtual Uri ResponseUri { get; }  
  
    public virtual Stream GetResponseStream();  
    ...  
}
```

■ Länge der Response

■ Mime-Type
■ Headers

■ URI der Response

■ Stream, um Response zu lesen

Beispiel WebRequest

- Laden und anzeigen der HTML Seite "http://www.zhaw.ch/~rege"

```
WebRequest rq = WebRequest.Create("http://www.zhaw.ch/~rege");
rq.Credentials = new NetworkCredential("rege", "*****");
WebResponse rsp = rq.GetResponse();

// Auslesen der Zeilen der HTML Seite
StreamReader r = new StreamReader(rsp.GetResponseStream());
for (string line = r.ReadLine(); line!=null; line = r.ReadLine())
    Console.WriteLine(line);
```

Reflection

■ Zugriff auf Metainformation zu Typen zur Laufzeit

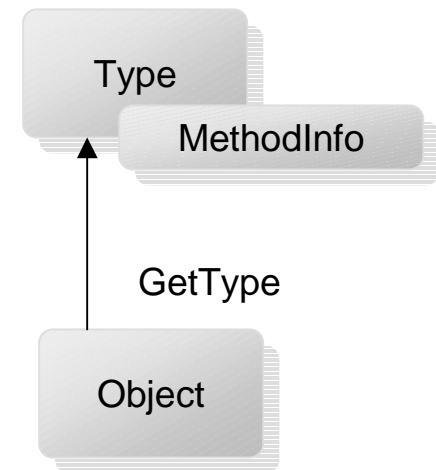
■ **System.Reflection** ermöglicht:

- Auslesen von Metainformationen über Assemblies, Module und Typen
- Auslesen von Metainformation über Members eines Typs
- Erzeugen einer Instanz eines Typs
- Auswahl und Aufruf von Methoden zur Laufzeit (*dynamic invocation*)
- Zugriff auf die Werte von Feldern und Properties eines Objekts
- Erstellen eines neuen Typs zur Laufzeit
 - ➔ Namensraums *System.Reflection.Emit*

Beispiel: Aufruf einer Methode

```
public class Test {  
    public void Hello() {  
        Console.WriteLine("Hello Test");  
    }  
}  
  
public void CallHello(object o) {  
    Type tp = o.GetType(); // tp = typeof(Test)  
    MethodInfo mt = tp.GetMethod("Hello");  
    mt.Invoke(o, null);  
}  
  
class MainClass {  
    public static void Main(string[] args) {  
        object o = new Test();  
        CallHello(o);  
    }  
}
```

beliebiges Objekt



Klasse Type

- Type dient der Metabeschreibung aller Typen zur Laufzeit
- erlaubt Zugriff auf Metainformation über Members

typeof(<class>)
oder
<object>.GetType()

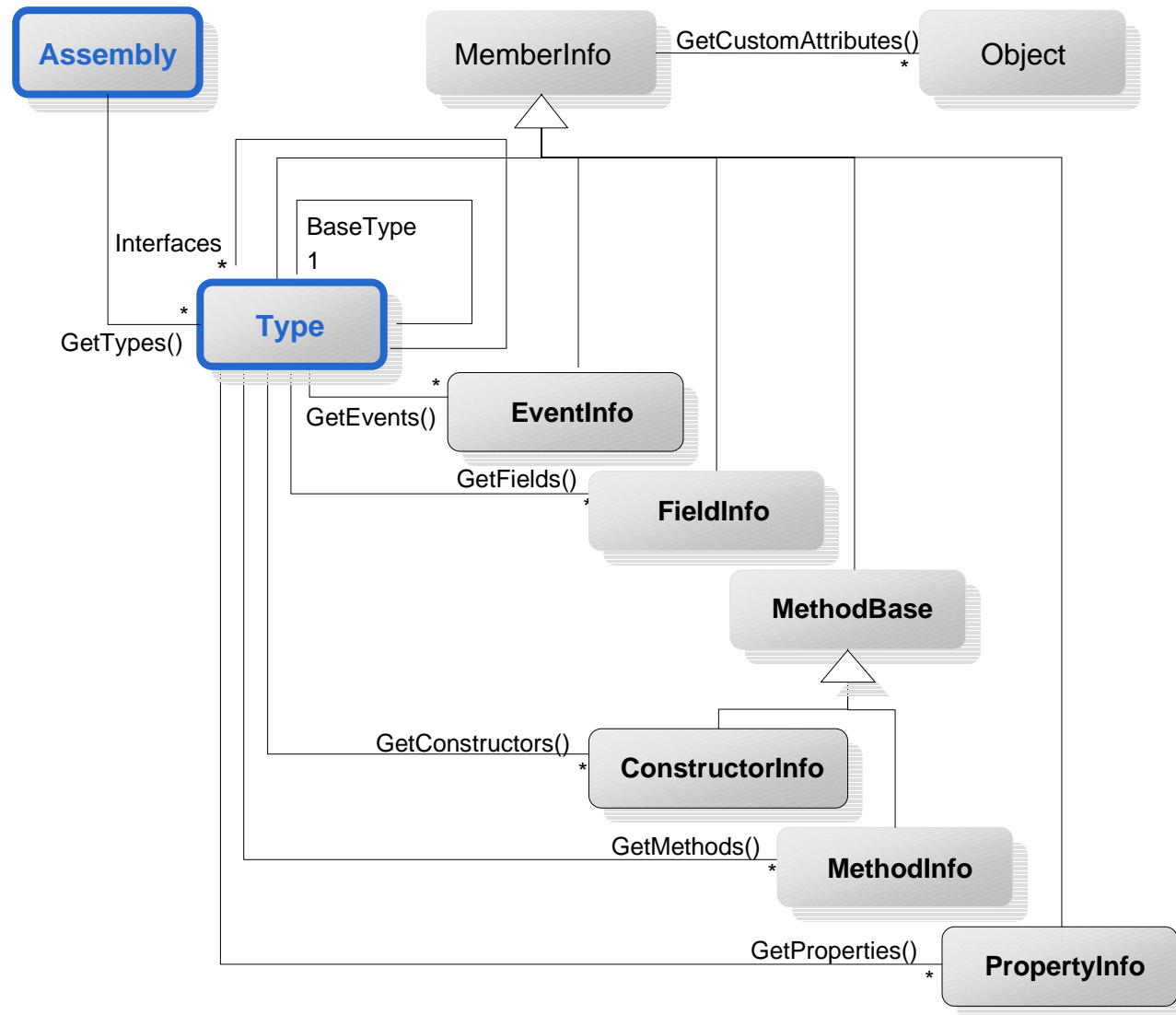
```
public abstract class Type : MemberInfo, IReflect
{
    public abstract Type BaseType {get;};
    public abstract string FullName {get;};
    public abstract string Name {get;};
    public Type[] GetInterfaces();

    public bool IsAbstract {get;};
    public bool IsClass {get;};
    public bool IsPublic {get;};

    public ConstructorInfo[] GetConstructors();
    public virtual EventInfo[] GetEvents();
    public FieldInfo[] GetFields();
    public MethodInfo[] GetMethods();
    public PropertyInfo[] GetProperties();
    ...
}
```

- direkter Basistyp
- Name des Typs
- Liste aller implementierten Interfaces
- Eigenschaften des Typs
- Konstruktoren, Ereignisse, Felder, Methoden, Properties,

Reflection Klassen Hierarchie



Klasse Assembly

- Die Klasse Assembly lädt Assemblies und ihre Metadaten.
- bietet Methoden für Zugriff auf Metadaten

```
public class Assembly {  
  
    public static Assembly Load(string name);  
  
    public virtual string Name {get;}  
    public virtual string FullName {get;}  
    public virtual string Location {get;}  
    public virtual MethodInfo EntryPoint {get;}  
  
    public Module[] GetModules();  
    public virtual Type[] GetTypes();  
    public virtual Type GetType(string typeName);  
  
    public object CreateInstance(string typeName);  
    ...  
}
```

- Laden eines Assembly
- Name
Speicherort
Einsprungspunkt des
Assembly
- Zugriff auf Module
alle im Assembly definierten
Typen
Type mit Name typeName
- Erzeugen eines Objekts vom
Typ typeName

Beispiel Reflection

■ C#-Programm "HelloWorld"

```
namespace World {  
    using System;  
    public class HelloWorld {  
  
        public static void Main(string[] args) {  
            Console.WriteLine("HelloWorld");  
        }  
  
        public override string Hello() {  
            return "Example HelloWorld";  
        }  
    }  
}
```

■ Übersetzen und Assembly erzeugen

csc HelloWorld.cs



HelloWorld.exe

■ Laden des Assemblies "HelloWorld.exe":

```
Assembly a = Assembly.Load("HelloWorld");
```

... Beispiel Reflection

■ Ausgabe der Namen aller **Typen** im Assembly:

```
Type[] types = a.GetTypes();  
foreach (Type t in types)  
    Console.WriteLine(t.FullName);
```

Ausgabe der Namen aller **Methoden** eines Typs:

```
Type hw = a.GetType("World.HelloWorld");  
MethodInfo[] methods = hw.GetMethods();  
foreach (MethodInfo m in methods)  
    Console.WriteLine(m.Name);
```

Ausgabe der Namen und **Werte** aller Felder eines Typs:

```
FieldInfo[] fa = o.GetType().GetFields();  
foreach (FieldInfo f in fa) {  
    string name = f.Name;  
    string val = (string)f.GetValue(o);  
    Console.WriteLine("{0} = {1}", name, val);  
}
```

... Beispiel Reflection

■ Erzeugen einer **Instanz** eines Typs:

```
Assembly a = Assembly.Load("HelloWorld");  
object o = a.CreateInstance("World.HelloWorld");
```

■ **Aufruf** der **Methode** Hello() ohne Parameter:

```
Type hw = a.GetType("World.HelloWorld");           // Typ HelloWorld  
MethodInfo mi = hw.GetMethod("Hello");  
object retVal = mi.Invoke(o, null); // Methode besitzt keine Parameter,  
liefert String zurück  
Console.WriteLine((string)retVal);
```

- GetCustomAttributes liefert Attributes eines Members oder Types

```
public abstract class MemberInfo : ICustomAttributeProvider {  
    public abstract object[] GetCustomAttributes( bool inherit );  
    public abstract object[] GetCustomAttributes( Type attributeType, bool  
inherit);  
    ...  
}
```

- Diese können zur Laufzeit ausgewertet werden

Beispiel Attribute

■ Definition einer Attribute-Klasse

```
using System;
using System.Reflection;

[AttributeUsage(AttributeTargets.All)]
public class MyAttribute : Attribute {
    private string myName;
    public MyAttribute(string name) {
        myName = name;
    }
    public string Name {
        get {
            return myName;
        }
    }
}
```

■ Verwendung des Attributes

```
[My("This is a class attribute.")]
public class MyClass1 {
    [My("This is a method attribute.")]
    public void MyMethod(int i) {
        return;
    }
}
```

Auslesen des Attributes und ausgeben

```
public class MemberInfo_GetCustomAttributes {
    public static void Main() {

        Type tp = typeof(MyClass1);

        // get class Attribute
        Object[] attrs =
            tp.GetCustomAttributes(typeof(MyAttribute), false);
        MyAttribute myA = (MyAttribute)attrs[0];
        Console.WriteLine(myA.Name);

        // get Method Attribute
        attrs = tp.GetMethod("MyMethod").
            GetCustomAttributes(typeof(MyAttribute), false);
        MyAttribute myA = (MyAttribute)attrs[0];
        Console.WriteLine(myA.Name);

    }
}
```

- Reflection.Emit erlaubt neue Assemblies und Typen zur Laufzeit zu erzeugen und sofort zu verwenden
 - Erzeugen von Assemblies
 - Erzeugen von neuen Modulen
 - Erzeugen von neuen Typen
 - Erzeugen symbolischer Metainformationen für bestehende Module

- System.Reflection.Emit ist zur Unterstützung für .NET Compiler und Interpreter gedacht.

- wichtige Klassen von Reflection.Emit sind
 - AssemblyBuilder um Assemblies zu definieren
 - ModuleBuilder um Module zu definieren
 - TypeBuilder um Typen zu definieren
 - MethodBuilder um Methoden zu definieren
 - ILGenerator um IL-Code zu erzeugen

■ Erzeugen eines neuen Assemblies und Modules

```
AssemblyName assemblyName = new AssemblyName();  
assemblyName.Name = "HelloWorldAssembly";  
AssemblyBuilder newAssembly = Thread.GetDomain().DefineDynamicAssembly(  
    assemblyName, AssemblyBuilderAccess.RunAndSave);  
ModuleBuilder newModule =  
    newAssembly.DefineDynamicModule("HelloWorldModule");
```

■ Definition eines neuen Typs

```
TypeBuilder newType = newModule.DefineType("HelloWorld", TypeAttributes.Public);
```

■ Definition einer neuen Methode mit ihren Parametertypen

```
Type[] paramTypes = new Type[] { typeof(string) };  
Type retType = Type.GetType("System.String");  
MethodBuilder newMethod = newType.DefineMethod("SayHelloTo",  
    MethodAttributes.Public | MethodAttributes.Virtual, retType, paramTypes);
```

... Beispiel Reflection.Emit

■ Einfügen der MSIL Operationen in die neu erzeugte Methode

```
ILGenerator ilGen = newMethod.GetILGenerator ();
ilGen.Emit (OpCodes.Ldstr, "Hello ");
ilGen.Emit (OpCodes.Ldarg_1);
Type t = Type.GetType ("System.String");
MethodInfo mi = t.GetMethod ("Concat", new Type[] {typeof(string), typeof(string)});
ilGen.Emit (OpCodes.Call, mi);
ilGen.Emit (OpCodes.Ret);
```

■ neuen Typen abschliessen

```
newType.CreateType ();
```

■ Erzeugen einer Instanz und ausführen der Methode SayHelloTo

```
MethodInfo method = newType.GetMethod ("SayHelloTo", new Type[] {typeof(string)});
object obj = Activator.CreateInstance (newType);
object ret = method.Invoke (obj, new string[] { "Wolfgang" });
Console.WriteLine (ret);
```

- Threading
 - Bibliothek basierend auf Monitor Konzept (wie Java)
- XML Verarbeitung
 - DOM und Serializer
- Networking
 - Sockets und Streams
- Reflection
 - Lesen von Klasseneigenschaften
 - Erzeugen von Datenstrukturen und Code

Fragen?

