

# .NET Klassenbibliothek

## Teil 1 die Basis

- Überblick und Aufbau
- Strings
- Collections
- Streaming

## Teil 1

- Überblick und Aufbau
- Strings
- Collections
- Streaming

## Teil 2

- Threading
- Reflection
- Netzwerkkommunikation
- XML

## Teil 3

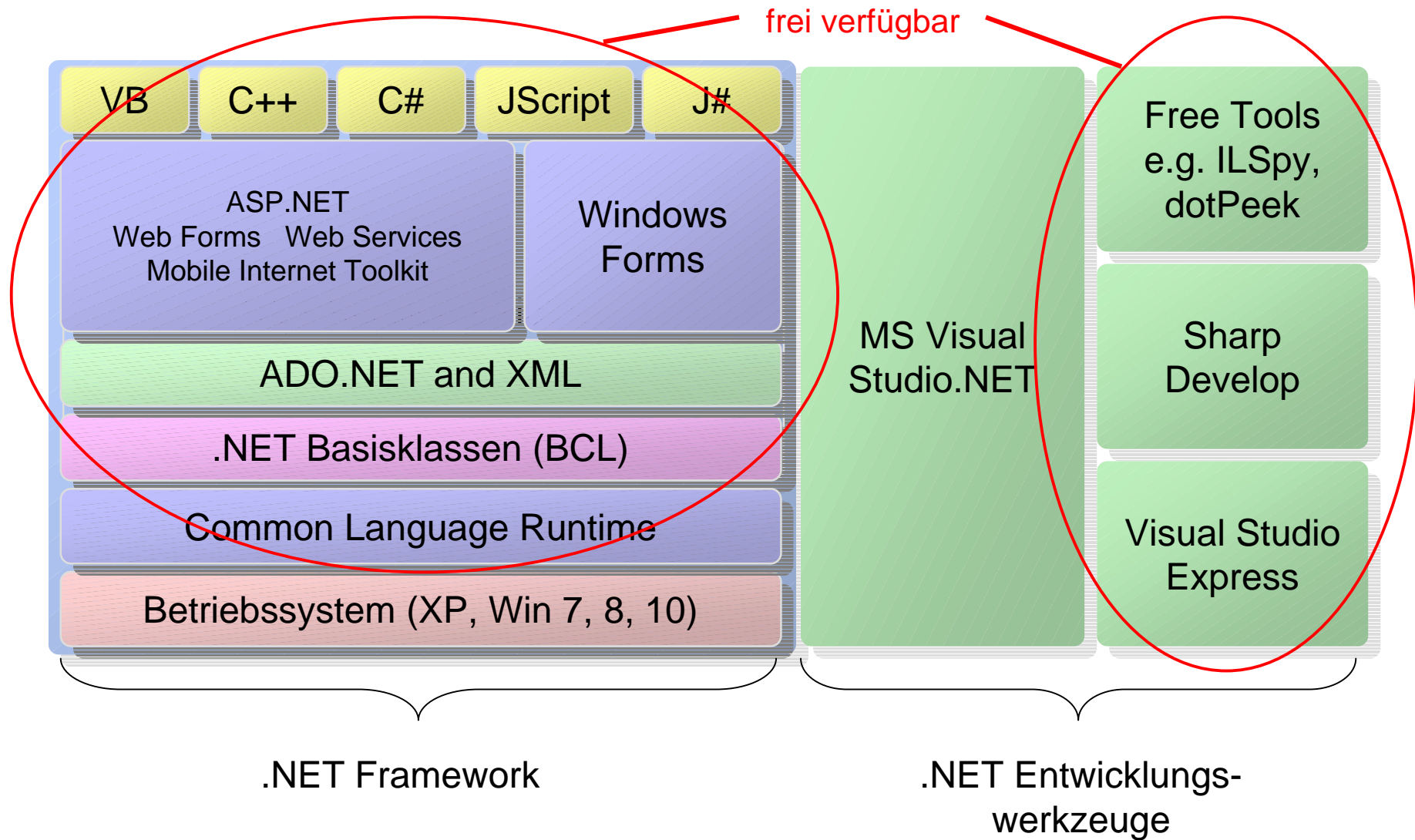
- Graphische Benutzeroberflächen
- ActiveX
- Low-Level Programmierung
- Sicherheit

## Teil 4 und folgende

- ADO.NET und XML
- Datenbankprogrammierung
- XML Verarbeitung
- ASP.NET
- Web Programmierung
- Web Services
- Verteilte Programmierung

# Überblick und Aufbau

# .NET Technologie



- Basisbausteine in Form von Klassen oder/und Komponenten

- Ein/Ausgabe
- GUI-Programmierung
- Networking
- Mengen und Listen
- Zeichenketten
- ...

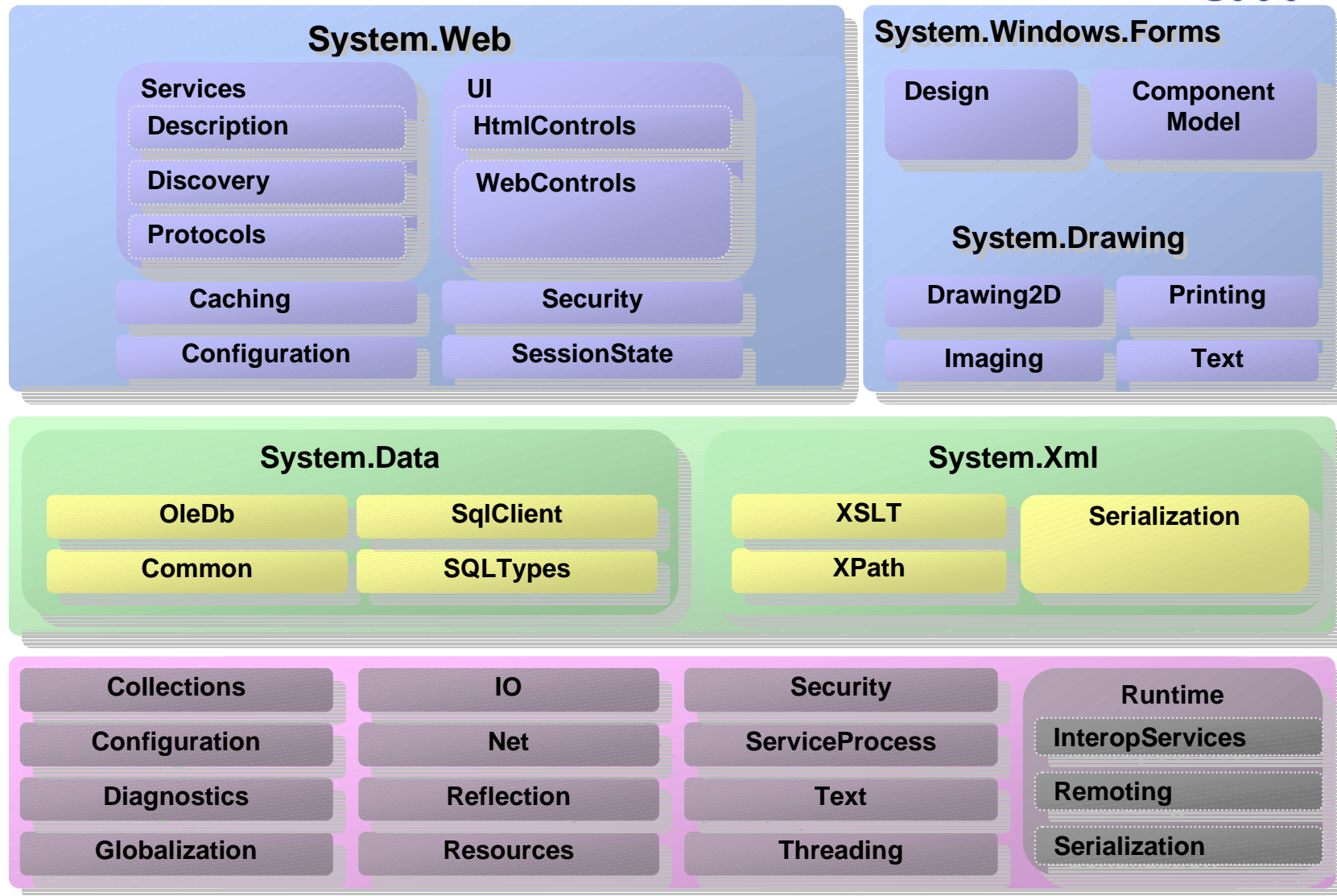
- robuste, stabile und qualitativ hochwertige Basis

- ermöglicht schnelle und einfache Entwicklung

- .NET bietet eine umfangreiche Klassenbibliothek
  - mehrere 100 Namensräume mit insgesamt mehreren 12'000 Typen (Klassen, Interfaces, Enumerationen)
  - für alle möglichen Bereiche (siehe Überblick)
  
- unterstützt wichtige Technologien wie:
  - XML
  - Kryptographie
  - Reflection
  - Threading
  - ...
  
- gemeinsame Basis für verschiedene Programmiersprachen

# Wichtigste Namensräume .NET

## Klassenbibliothek



# Klasse Math (1)

```
public sealed class Math {  
    public const double PI = 3.14...;  
    public const double E = 2.71...;  
  
    public static T Abs(T val);    // T sbyte, short, int, long, float, double,  
    decimal  
    public static T Sign(T val);  
    public static T1 Min(T1 x, T1 y);    // T1 ... T, byte, ushort, uint, ulong  
    public static T1 Max(T1 x, T1 y);  
  
    public static double Round(double x);  
    public static double Floor(double x);  
    public static double Ceiling(double x);  
    ...  
}
```



## Klasse Math (2)

```
...  
public static double Sqrt(double x);  
public static double Pow(double x, double y);  
public static double Exp(double x);  
public static double Log(double x);  
public static double Log10(double x);  
  
public static double Sin(double x);  
public static double Cos(double x);  
public static double Tan(double x);  
public static double Asin(double x);  
public static double Acos(double x);  
public static double Atan(double x);  
}
```

# Klasse Random

```
public class Random {  
    public Random();  
    public Random(int seed);  
  
    public virtual int Next();           // 0 <= res <= int.MaxValue  
    public virtual int Next(int x);     // 0 <= res < x  
    public virtual int Next(int x, int y); // x <= res < y  
  
    public virtual double NextDouble(); // 0.0 <= res < 1.0  
  
    public virtual void NextBytes(byte[] b); // füllt b mit Zufallszahlen  
}
```

# Klasse DateTime

```
public class DateTime {  
    public DateTime(long ticks); //100 ns unit  
    public DateTime(int year, int month, int day, int hour, int minute, int  
        second);  
    public long Ticks;  
    public int Second;  
    public int Hour;  
    .....  
    public DateTime Now();  
    public DateTime Today();  
  
    public static DateTime Parse(string s);  
    public ToString (string format);  
    .....  
}
```

d :24.11.2004  
D :Mittwoch, 24. November 2004  
f :Mittwoch, 24. November 2004 19:25  
F :Mittwoch, 24. November 2004 19:25:17  
g :24.11.2004 19:25  
G :24.11.2004 19:25:17  
m :24 November  
r :Wed, 24 Nov 2004 19:25:17 GMT  
s :2004-11-24T19:25:17  
t :19:25  
T :19:25:17  
u :2004-11-24 19:25:17Z  
U :Mittwoch, 24. November 2004 18:25:17  
y :November 2004  
dddd, MMMM dd yyyy :Mittwoch, November 24  
2004  
ddd, MMM d ""yy :Mi, Nov 24 '04  
dddd, MMMM dd :Mittwoch, November 24

# Zeichenketten - Strings

# Verarbeitung von Zeichenketten

- Klassen `System.String` und `System.Text.StringBuilder`
- Objekte der Klasse `String` sind unveränderlich (*immutable*)!

## Beispiel "Zeichenketten":

```
string s = "Hello";  
s += ", There";  
char c = s[5]; // Indexer liefert ','
```

- Operation `==` vergleicht den Inhalt von Zeichenketten!

```
string s2 = "Hello, There";  
if(s == s2) // liefert true!
```

- Objektvergleich durch:

```
if((object)s == (object)s2) // liefert false!
```

# Klasse String

```
public sealed class String : IComparable, ICloneable, IConvertible,
    IEnumerable
{
    public char this[int index] {get;}
    public int Length {get;}
    public static int Compare(string strA, string strB);    // Culture!
    public static int CompareOrdinal(string strA, string strB); // ohne
    Culture!
    public static string Format(string format, object arg0);
    public int IndexOf(string);
    public int IndexOfAny(char[] anyOf);
    public int LastIndexOf(string value);
    public string PadLeft(int width, char c);    // s.PadLeft(10, '.'); =>
    ".....Hello"
    public string[] Split(params char[] separator);
    public string Substring(int startIndex, int length);
    ...
}
```

Vorsicht: in Java  
endsWith

# Formatierung von Zeichenketten

```
Console.WriteLine("{0,3:X}", 10); // ergibt "  A"
```

äquivalent zu:

```
string f;  
f = string.Format("{0,3:X}", 10);  
Console.WriteLine(f);
```

Argumentnummer (0,1,..)

Feldbreite positiv = rechtsbündig,  
negativ = linksbündig

C	Währung
D	Integer
E	Numerisch E+ Darstellung
F	Fixpunkt Dezimal
P	Prozentdarstellung
X	Hexadezimaldarstellung
...	

# Formatierungscode für Zahlen

d, D	<b>Dezimalformat</b> (ganze Zahl mit führenden Nullen)	-xxxxx
	precision = Anz. Ziffern	
f, F	<b>Fixpunktformat</b>	-xxxxx.xx
	precision = Anz. Nachkommastellen (Default = 2)	
n, N	<b>Nummernformat</b> (mit Tausender-Trennstrich)	-xx,xxx.xx
	precision = Anz. Nachkommastellen (Default = 2)	
e, E	<b>Exponentialformat</b> (gross/klein signifikant)	-x.xxxE+xxx
	precision = Anz. Nachkommastellen	
c, C	<b>Currency-Format</b>	\$xx,xxx.xx
	precision = Anz. Nachkommastellen (Default = 2)	
	Negative Werte werden in Klammern gesetzt (\$xx,xxx.xx)	
x, X	<b>Hexadezimalformat</b> (gross/klein signifikant)	xxx
	precision = Anzahl Hex-Ziffern (evtl. führende 0)	
g, G	<b>General</b> (kompaktestes Format für gegebenen Wert; Default)	



# Beispiele

```
int x = 17;
```

<code>Console.WriteLine("{0}", x);</code>	17
<code>Console.WriteLine("{0,5}", x);</code>	17
<code>Console.WriteLine("{0:d}", x);</code>	17
<code>Console.WriteLine("{0:d5}", x);</code>	00017
<code>Console.WriteLine("{0:f}", x);</code>	17.00
<code>Console.WriteLine("{0:f1}", x);</code>	17.0
<code>Console.WriteLine("{0:E}", x);</code>	1.700000E+001
<code>Console.WriteLine("{0:E1}", x);</code>	1.7E+001
<code>Console.WriteLine("{0:x}", x);</code>	11
<code>Console.WriteLine("{0:x4}", x);</code>	0011

# Klasse StringBuilder

in Java StringBuffer



- StringBuilder effektiver bei Operationen, die eine Zeichenkette verändern.

```
public sealed class StringBuilder {  
  
    public int Capacity {get; set;}  
    public int Length {get; set;}  
  
    StringBuilder Append(...);  
    StringBuilder AppendFormat(...);  
    StringBuilder Insert(int index, ...);  
    StringBuilder Remove(int startIndex, int  
        length);  
    StringBuilder Replace(char oldChar, char  
        newChar);  
  
    string ToString();  
}
```

- reservierte Grösse
- die Länge der Zeichenkette
- Anfügen  
Einfügen  
Löschen  
Ersetzen
- String erzeugen

# Zerlegen von Zeichenketten

```
string s = "Hans,Fritz,Peter;Anna"
```

## Zerlegen des Strings


```
string[] tok = s.Split(',', ';', '\n');  
foreach (string t in tok)  
    Console.WriteLine(t);
```

# Regex

# Klasse Regex und Match

```
using System.Text.RegularExpressions;
```

```
public class Regex {  
    public Regex(string regex); // erstelle Ausdruck für Regex  
    public bool IsMatch(string s); // ist der String ein Match  
    public Match Match(string s); // erste Übereinstimmung mit Muster  
    public string Replace(string s, string pattern) // ersetze Teilstrings  
    ...  
}
```



```
public class Match {  
    public bool Success; // Übereinstimmung wurde gefunden  
    public string Value; // liefert String-Wert der Übereinstimmung  
    public GroupCollection Groups; // Gruppen von Übereinstimmungen  
    public Match NextMatch(); // nächster Match  
    ...  
}
```

. : beliebiger Buchstaben  
\ : Spezialzeichen folgt  
\* : beliebig oft  
{n} : n mal  
+ : mindestens 1-mal

^ : nicht  
[abc],[a-z] : Zeichengruppe  
[^a] : alle Zeichen ausser a  
(?<groupname>pattern) : definiere Name der Gruppe in Muster

# Anwendung der Klasse Regex und Match

```
using System;
using System.Text.RegularExpressions;
...

Regex emailregex = new Regex("(?<user>[^@]+)@(?<host>[^\s;]+)");
String s = "johndoe@tempuri.org;bill.gates@microsoft.com;";

Match m = emailregex.Match(s);

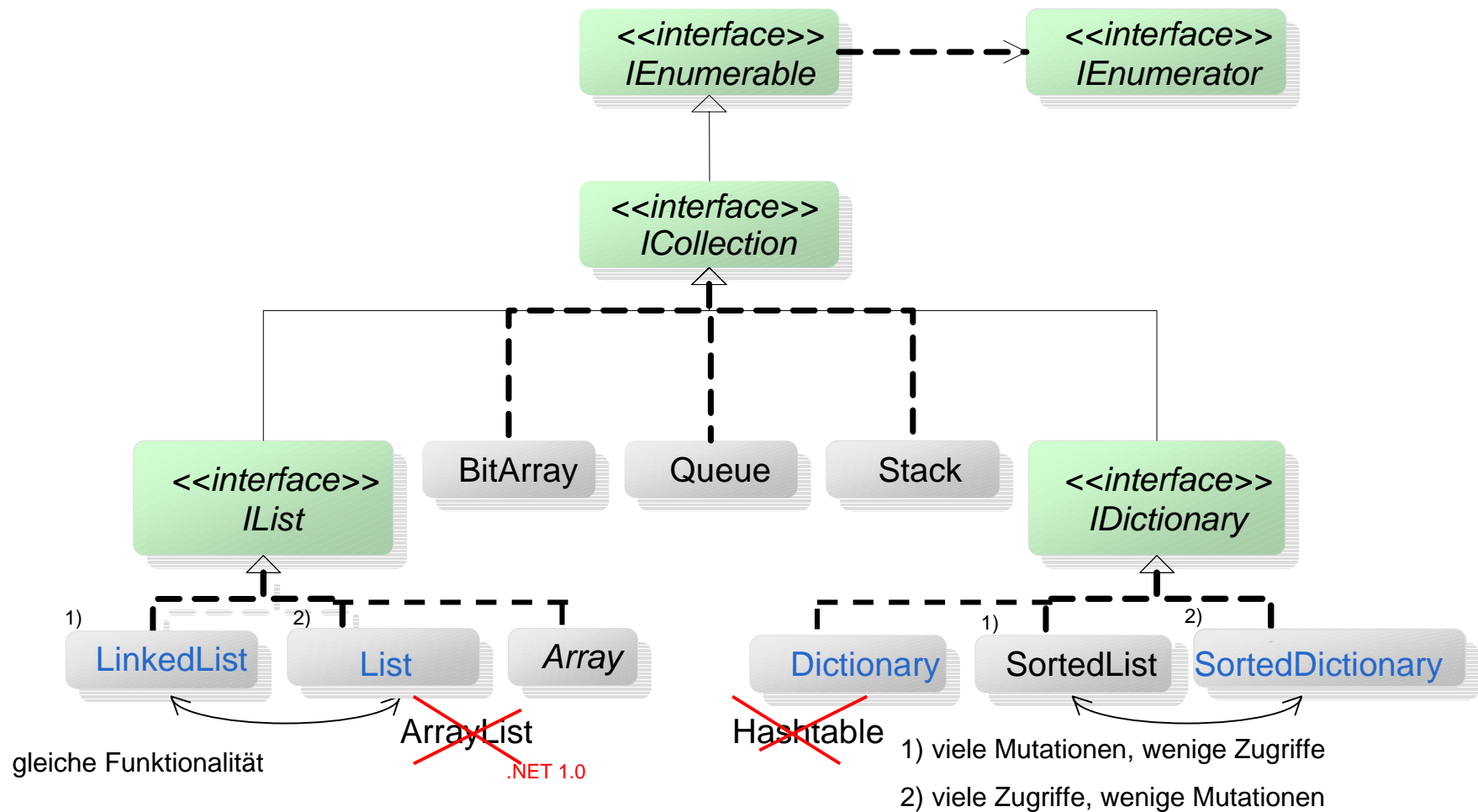
while ( m.Success ) {
    System.Console.WriteLine("E-Mail:" + m.Value);
    System.Console.WriteLine("User:  " + m.Groups["user"].Value);
    System.Console.WriteLine("Host:  " + m.Groups["host"].Value);
    m = m.NextMatch();
}
```

```
E-Mail:johndoe@tempuri.org
User:  johndoe
Host:  tempuri.org
E-Mail: bill.gates@microsoft.com
User:  bill.gates
Host:  microsoft.com
```

# Collections

# Neue Collections

- Neue in System.Collections.Generic und System





# Namensraum System.Collections.Generic

## ■ Ab .NET 2.0 alle Collections generisch

### ■ Beispiel

```
class Temperature : IComparable<Temperature> {  
    int CompareTo(Temperature t) {  
        ..  
    }  
    ...  
}  
IList<Temperature> list = new List<Temperature>();
```

#### **Interfaces**

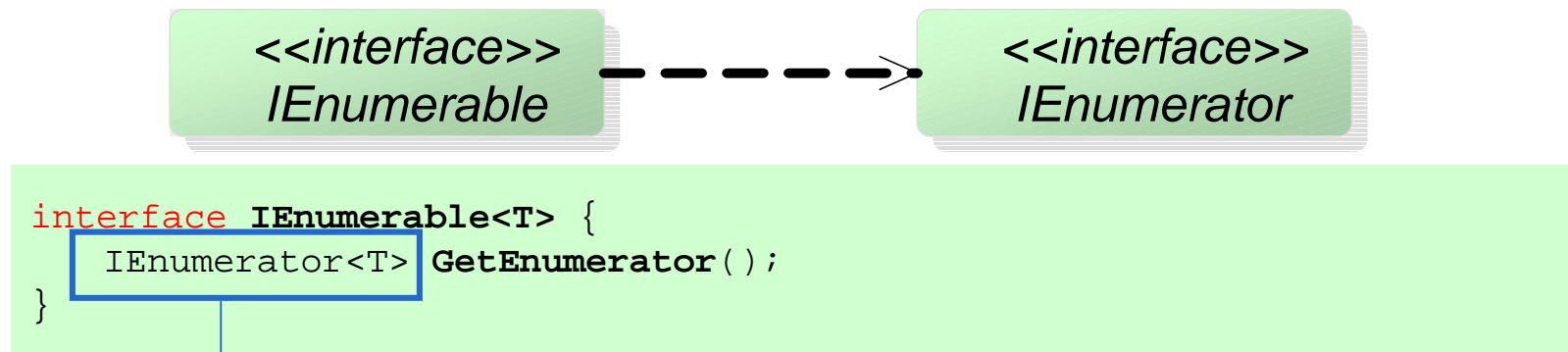
```
ICollection<T>  
IList<T>  
IDictionary<T, U>  
IEnumerable<T>  
IEnumerator<T>  
IComparable<T>  
IComparer<T>
```

#### **Klassen**

```
List<T>  
Dictionary<T, U>  
SortedDictionary<T, U>  
Stack<T>  
Queue<T>
```

# IEnumerable<T> und IEnumerator<T>

- Alles Aufzählbare wird durch die Schnittstelle IEnumerable repräsentiert:



- IEnumerator realisiert einen Iterator

```
interface IEnumerator<T> {  
    T Current {get;}  
    bool MoveNext();  
    void Reset();  
}
```

# IEnumerable<T> und IEnumerator<T>

■ zu den Klassen, die IEnumerable implementieren, gehören:

- Array
- List,
- LinkedList
- String
- Dictionary
- uvm.

■ bei Typen die IEnumerable implementieren kann **foreach** verwendet werden;

Beispiel Array:

```
int[] a = {1, 6, 8, 9, 15};    // Objekt vom abstrakten Typ Array
foreach (int i in a) System.Console.WriteLine(i);
```

- Es wird eine Instanz einer IEnumerator implementierenden Klasse mittels GetEnumerator angefordert
- Funktioniert auch mit eigenen Klassen

```
class MyClass: IEnumerable<MyClass> {  
    ...  
    public IEnumerator<MyClass> GetEnumerator() {  
        return new MyEnumerator<MyClass>(...);  
    }  
}
```

```
class MyEnumerator : IEnumerator<MyClass> {  
    public MyClass Current { get { ... } }  
    public bool MoveNext() { ... }  
    public void Reset() { ... }  
}
```

```
interface IEnumerable<T> {  
    IEnumerator<T> GetEnumerator();  
}
```

Musste in .NET 1.1  
implementiert werden

```
MyClass x = new MyClass();  
...  
foreach (string s in x) Console.Write(s + " ");  
// liefert "first second third"
```

## Funktionsweise

- MyEnumerator liefert eine Folge von Werten
- foreach-Anweisung durchläuft diese Werte

## ... Implementation mit yield (ab .NET 2.0)

```
class MyClass {  
    string first = "first";  
    string second = "second";  
    string third = "third";  
    ...  
    public IEnumerator<MyClass> GetEnumerator()  
    {  
        yield return first;  
        yield return second;  
        yield return third;  
    }  
}
```

an NET 2.0 mit  
Hilfe von yield  
implementiert

```
MyClass x = new MyClass();  
...  
foreach (string s in x) Console.Write(s + " ");  
// liefert "first second third"
```

### Merkmale einer Iteratormethode

- Hat die Signatur  
public IEnumerator GetEnumerator
- Anweisungsteil enthält zumindest  
eine yield-Anweisung, Bsp

```
string[] _value;  
public IEnumerator<T> GetEnumerator() {  
    for (int i = 0;  
        i < _value.length; i++)  
        yield return _value[i];  
}
```

### Funktionsweise von yield

- Stoppt Ausführung; liefert eine  
Folge von Werten
- foreach-Anweisung durchläuft diese  
Werte

### ■ Bemerkungen:

- Statt *IEnumerator* sollte besser *IEnumerator<T>* verwendet werden (kein Cast nötig)
- *MyClass* muss *IEnumerable* gar nicht implementieren, sondern nur Methode *GetEnumerator* !

## 2 Arten

**yield return *expr*;**

- liefert einen Wert an die foreach-Schleife
- darf nur in einer Iterator-Methode vorkommen
- Typ von *expr* muss kompatibel sein zu
  - *T* (wenn *IEnumerator<T>*)
  - *object*(sonst)

**yield break;**

- bricht die Iteration ab
- darf nur in einer Iterator-Methode vorkommen

# Mehrere Iteratoren pro Klasse

```
class MyList {
    int[] data = ...;

    public IEnumerator<int> GetEnumerator() {
        for (int i = 0; i < data.Length; i++)
            yield return data[i];
    }

    public IEnumerable<int> Range(int from,
int to) {
        if (to > data.Length) to = data.Length;
        for (int i = from; i < to; i++)
            yield return data[i];
    }

    public IEnumerable<int> Downwards {
        get {
            for (int i = data.Length - 1;
                i >= 0; i--)
                yield return data[i];
        }
    }
}
```

```
MyList list = new MyList();
foreach (int x in list) Console.WriteLine(x);
foreach (int x in list.Range(2, 7)) Console.WriteLine(x);
foreach (int x in list.Downwards) Console.WriteLine(x);
```

## Standard-Iterator

### Spezifischer Iterator als Methode

- beliebiger Name und Parameterliste
- Rückgabetyp IEnumerable!

### Spezifischer Iterator als Property

- beliebiger Name
- Rückgabetyp IEnumerable!

# Beispiel: Iterieren über einen Baum

```
class Tree {  
    Node root = null;  
  
    public void Add(int val) {...}  
    public bool Contains(int val) {...}  
  
    public IEnumerator<int> GetEnumerator() {  
        return root.GetEnumerator();  
    }  
}
```

```
class Node {  
    public int val;  
    public Node left, right;  
  
    public Node(int x) { val = x; }  
  
    public IEnumerator<int> GetEnumerator() {  
        if (left != null)  
            foreach (int x in left) yield return x;  
        yield return val;  
        if (right != null)  
            foreach (int x in right) yield return x;  
    }  
}
```

## Verwendung

```
...  
Tree tree = new Tree();  
...  
foreach (int x in tree)  
    Console.WriteLine(x);
```

## ■ Hinweis

- Es wird hier allerdings für jeden Knoten des Baums ein IEnumerator-Objekt erzeugt!



# Interface ICollection<T>

## ■ Basisschnittstelle für Sammlungen

**int** Count {get;}

- Anzahl der Elemente

**bool** IsSynchronized {get;}

- Collection synchronisiert?

**object** SyncRoot {get;}

- liefert Objekt zur Synchronisierung

**void** CopyTo(Array a, **int** index);

- Kopiert die Elemente in Array a (ab Position index)

```
interface ICollection<T> {  
    //---- Properties  
    int Count {get;}  
    bool IsSynchronized {get;}  
    object SyncRoot {get;}  
    //---- Methods  
    void CopyTo(Array a, int index);  
}
```

# Interface IList<T>

## ■ Interface für Objektsammlungen mit definierter Reihenfolge

```
interface IList<T> {  
    object this [ int index ] {get; set;}  
  
    int Add(T value);  
    void Insert(int index, T value);  
    void Remove(T value);  
    void RemoveAt(int index);  
    void Clear();  
  
    bool Contains(T value);  
  
    bool IsFixedSize {get;}  
    bool IsReadOnly {get;}  
    ...  
}
```

■ Indexer für Zugriff auf Elemente über Position

■ Anfügen,  
■ Einfügen,  
■ Löschen von Elementen

■ Abfrage auf Enthaltensein von Elementen

■ Hat Liste fixe Länge?  
■ Ist Liste nur lesbar?

# Klasse Array

- Arrays sind Instanzen von Klassen abgeleitet von Basisklasse Array
- Array implementiert IList, ICollection und IEnumerable
- Arrays haben fixe Grösse (IsFixedSize() == true)
- Array stellt eine reiche Schnittstelle zur Verfügung

```
public abstract class Array :  
    ICloneable, IList, ICollection, IEnumerable {  
    //---- Properties  
    public int Length {get;}  
    public int Rank {get;}  
  
    //----- Methoden  
    public int GetLength(int dimension);  
    public int GetLowerBound(int dimension);  
    public int GetUpperBound(int dimension);  
  
    public object GetValue(int idx);  
    public object GetValue(int[] idx);  
    public void SetValue(object val, int idx);  
    public void SetValue(object val, int[] idx);  
}
```

- Abfrage von Länge und Anzahl Dimensionen

- Abfrage von Länge, untere und obere Grenze für jede Dimension

- Lesen und Setzen der Werte im Array

## ... Klasse Array

```
...
//----- statische Methoden
public static int IndexOf(Array a, object val);
public static int LastIndexOf(Array a, object value);

public static void Sort(Array a);
public static void Sort(Array a, IComparer comparer);
public static void Reverse(Array a);

public static int BinarySearch(Array a, object val);
public static int BinarySearch(Array a, object val,
                               IComparer c);

public static void Copy(Array srcArray,
                       Array destArray, int len);
public static Array CreateInstance(Type elementType,
                                   int len);
public static Array CreateInstance(Type elementType,
                                   int[] len);
...
}
```

■ Suchen nach Positionen von Elementen

■ Sortieren von Arrays

■ Binäres Suchen in sortierten Arrays

■ Kopieren und erzeugen

# Beispiel Array

## ■ Erzeugen von Array mit Array.CreateInstance

```
int[] i = (int[]) Array.CreateInstance(typeof(Int32), 6);
```

## ■ ist equivalent zu:

```
int[] i = new int[6];
```

## ■ Werte setzen, sortieren

```
i[0] = 3; i[1] = 1; i[2] = 5; i[3] = 2; i[4] = 9; i[5] = 4;  
Array.Sort(i); // Sortiert die einzelnen Elemente des Arrays
```

## ■ mit foreach ausgeben

```
foreach (int elem in i)  
    Console.Write("{0} ", elem); // Ausgabe in sortierter Reihenfolge
```

Elemente: 1 2 3 4 5 9

# Klasse List <T>

- List realisiert dynamisch wachsende Liste
- als Array implementiert

```
public class List<T> :  
    IList<T>, ICollection, IEnumerable {  
  
    public List();  
    public List(ICollection<T> c);  
    public List(int capacity);  
  
    virtual int Capacity {get;set;}  
  
    public virtual List GetRange  
        (int index, int count);  
    public virtual void AddRange(ICollection<T> c);  
    public virtual void InsertRange  
        (int index, ICollection c);  
    public virtual void SetRange  
        (int i, ICollection<T> c);  
    public virtual void RemoveRange  
        (int index, int count);  
    ...  
}
```

■ Konstruktoren

■ Kapazität

■ Zugreifen,  
Einfügen,  
Setzen,  
Löschen

## ... Klasse List <T>

```
...  
public virtual void Sort();  
public virtual void Reverse();  
public virtual int BinarySearch(T o);  
public virtual int LastIndexOf(T o);  
  
public static List<T> Adapter(Ilist<T> list);  
public static List<T> FixedSize(List<T> l);  
public static List<T> ReadOnly(List<T> l);  
public static List<T> Synchronized(List<T> list);  
  
public virtual void CopyTo(Array a);  
public virtual T[] ToArray<T>();  
  
public virtual void TrimToSize();  
}
```

■ Sortieren und  
Suchen

■ Erzeugen von Wrapper

■ Kopien erstellen

■ Grösse anpassen

# Beispiel List

## ■ List erzeugen und Werte anfügen

```
ICollection<int> a = new List<int>();  
a.Add(3); a.Add(1); a.Add(2); a.Add(4); a.Add(9); a.Add(5);
```

## ■ List sortieren und ausgeben

```
a.Sort();  
foreach (int i in a) Console.WriteLine(i);
```

Elemente: 1 2 3 4 5 9

## ■ List invertieren und ausgeben

```
a.Reverse();  
foreach (int i in a) Console.WriteLine(i);
```

Elemente: 9 4 3 2 1



# Klasse LinkedList<T>

- LinkedList realisiert dynamisch wachsende Liste
- als "echte" Liste implementiert

```
public class LinkedList<T> :  
    IList<T>, ICollection, IEnumerable {  
  
    public LinkedList();  
    public LinkedList(ICollection<T> c);  
  
    public virtual LinkedList GetRange  
        (int index, int count);  
    public virtual void AddRange(ICollection c);  
    public virtual void InsertRange  
        (int index, ICollection<T> c);  
    public virtual void SetRange  
        (int i, ICollection<T> c);  
    public virtual void RemoveRange  
        (int index, int count);  
    ...  
}
```

■ Konstruktoren

■ Zugreifen,  
Einfügen,  
Setzen,  
Löschen

# IComparable<T> und IComparer<T>

## ■ IComparable ist Interface für Typen mit Ordnung

```
public interface IComparable<T> {  
    int CompareTo(T obj); // <0 if x < y, 0 if x == y, >0 if x > y  
}
```

## ■ Klassen, die IComparable implementieren

- Wertetypen wie Int32, Double, DateTime, ...
- Klasse Enum als Basisklasse aller Enumerationstypen
- Klasse String

## ■ IComparer ist Interface zur Realisierung von Vergleichsobjekten

```
public interface IComparer<T> {  
    int Compare(T x, T y); // <0 if x < y, 0 if x == y, >0 if x > y  
}
```

## IComparer-Implementierungen:

- Comparer, CaseInsensitiveComparer: Zeichenkettenvergleich

# Beispiel: IComparable

- Eigener Typ Vector stellt zweidimensionalen Vektor (in der Ebene) dar
- implementiert IComparable, wobei Sortierung nach Länge erfolgt

```
public class Vector : IComparable<Vector> {  
    private double x, y;  
    public Vector(double x, double y) { this.x = x; this.y = y; }  
    public double Length { get { return Math.Sqrt( x*x + y*y ); } }  
    public int CompareTo(Vector obj) {  
        if(this.Length < obj.Length) return -1;  
        else if(this.Length > obj.Length) return 1;  
        else return 0;  
  
        throw new ArgumentException();  
    }  
}
```

Besser:  
return this.Length - obj.Length;

## ... Beispiel IComparable

### ■ Array von Vektoren erzeugen

```
Vector[] vArray = { new Vector(1.5,2.3), new Vector(3,6), new Vector(2,2) };
```

### ■ Array wird nach der Länge der Vektoren aufsteigend sortiert, bzw. die Sortierung invertiert:

```
Array.Sort(vArray);  
printArray(vArray);  
Array.Reverse(vArray);  
printArray(vArray);
```

# Beispiel IComparer

- Erzeugen eines Arrays von Strings

```
string[] names = string[] { "frank", "john", "Bill", "paul", "Frank" };
```

- Sortieren ohne Berücksichtigung von Gross- und Kleinschreibung

```
IComparer<string,string> ciComparer = new CaseInsensitiveComparer ();  
Array.Sort(names, ciComparer);
```

- Namen binär suchen

```
int pos = Array.BinarySearch("John", ciComparer);
```

- Array invertieren.

```
names = Array.Reverse(names, ciComparer);
```

# Interface IDictionary<K,V>

- IDictionary ist Interface für Sammlungen von Schlüssel-Wert-Paaren

```
interface IDictionary<K,V> :  
    ICollection<KeyValuePair<K, V>>,  
    IEnumerable<KeyValuePair<K, V>>,  
    IEnumerable  
  
    ICollection<K> Keys {get;};  
    ICollection<V> Values {get;};  
  
    V this[K key] {get; set;}  
  
    void Add(K key, V value);  
    void Remove(K key);  
    bool Contains(K key);  
  
    ...  
}
```

- Schlüssel
- Werte
- Indexer für Zugriff mit Schlüssel
- Anfügen,  
Löschen,  
Enthaltensein

# KeyValuePair

- IDictionaryEnumerator ist Iterator über Schlüssel-Wert-Paare
- IDictionaryEntry stellt ein Schlüssel-Wert-Paar dar

```
interface IDictionary : IEnumerable {  
    ...  
    IDictionaryEnumerator<KeyValuePair<K, V>> GetEnumerator();  
    ...  
}
```



```
public struct KeyValuePair<K,V> {  
    //----- Properties  
    public K Key {get;};  
    public V Value {get;};  
}
```

# Dictionary<K,V>

- Dictionary ist Implementierung von IDictionary
- nach Hashcode des Schlüssels organisiert
  - ➔ Schlüsselobjekte müssen GetHashCode und Equals überschreiben

```
public class Dictionary<K,V> : IDictionary<T,V>, ...
{
    public Dictionary();
    public Dictionary(IDictionary<T,V> d);
    public Dictionary(int capacity);

    public virtual V this[K key]
        {get; set;}

    public virtual bool ContainsKey(K key);
    public virtual bool ContainsValue(V val);

    ...
}
```

- Konstruktoren
- Indexer für Zugriff mit Schlüssel
- Prüfen, ob Schlüssel und Wert enthalten



# Beispiel: Dictionary

- Dictionary erzeugen und Person-Objekte mit Versicherungsnummer als Schlüssel einfügen

```
IDictionary<int, Person> h = new Dictionary<int, Person>();  
h.Add(3181030750, new Person("Mike", "Miller"));  
h.Add(1245010770, new Person("Susanne", "Parker"));  
h.Add(2345020588, new Person("Roland", "Howard"));  
h.Add(1245300881, new Person("Douglas", "Adams"));
```

```
public class Person {  
    public Person(string fn, string ln)  
    {  
        ...  
    }  
    public override string ToString() {  
        ...  
    }  
}
```

- Über die Einträge iterieren und Wert und Schlüssel ausgeben

```
foreach (DictionaryEntry x in h)  
    Console.WriteLine(x.Value + ": " + x.Key);
```

- Enthaltensein eines Eintrags mit einem bestimmten Schlüssel prüfen

```
if (h.Contains(1245010770))  
    Console.WriteLine("Person mit SNr. 1245010770: " + h[1245010770]);
```

# Dictionary SortedList

- SortedList ist zweite Implementierung von IDictionary
- dynamische Liste von Schlüssel-Wert-Paaren nach Schlüssel sortiert !

```
public class SortedList<K,V> :IDictionary<T,V>,  
ICollection, ... {  
    public SortedList();  
    public SortedList(IComparer<K> c);  
  
    public virtual V this[K key]  
        {get; set;};  
    public virtual V GetByIndex(int i);  
    public virtual K GetKey(int i);  
  
    public virtual IList<K> GetKeyList();  
    public virtual IList<V> GetValueList();  
  
    public virtual int IndexOfKey(K key);  
    public virtual int IndexOfValue(V value);  
  
    public virtual void RemoveAt(int i);  
    ...  
}
```

- Konstruktoren
- Indexer für Zugriff mit Schlüssel
- Zugriff auf Wert und Schlüssel über eine Indexposition
- Liste von Schlüssel und Werten
- Position von Schlüssel und Wert
- Löschen eines Eintrags an einer Position

# Spezielle Collections

## ■ Queue

```
public class Queue<T> : ICollection, IEnumerable<T> {  
    public virtual void Clear();  
    public virtual bool Contains(T o);  
    public virtual T Dequeue();  
    public virtual void Enqueue(T o);  
    public virtual T Peek();  
    ...  
}
```

## ■ Stack

```
public class Stack<T> : ICollection, IEnumerable<T> {  
    public virtual void Clear();  
    public virtual bool Contains(T o);  
    public virtual T Peek();  
    public virtual T Pop();  
    public virtual void Push(T o);  
    ...  
}
```

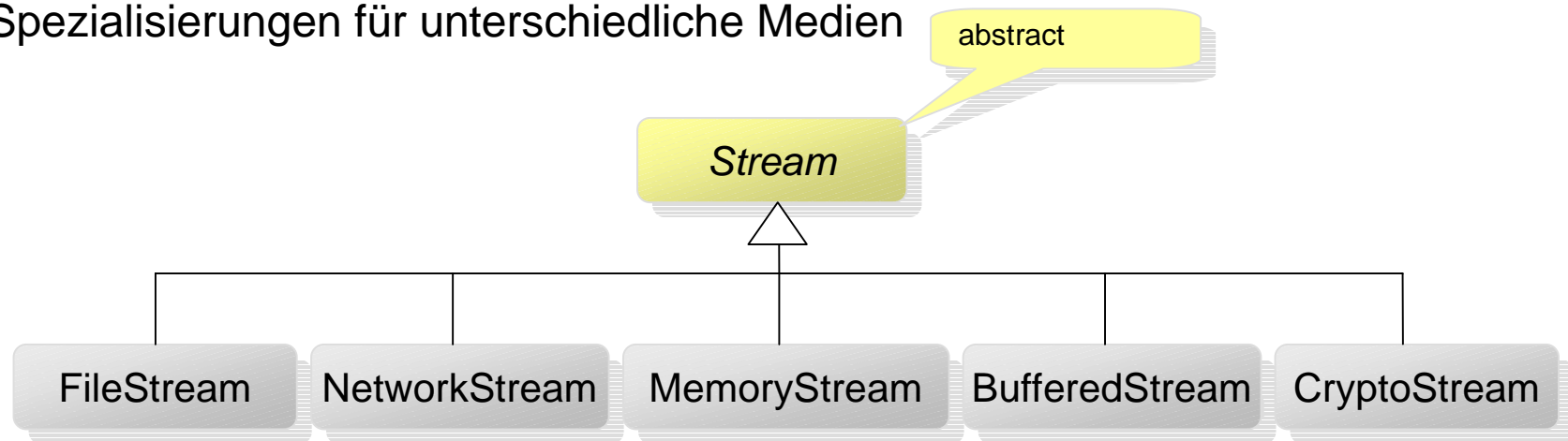
## ■ BitArray

```
public sealed class BitArray<T> : ICollection, IEnumerable<T> {  
    public bool this[int index] {get; set;}  
    public int Length {get; set;}  
    public BitArray And(BitArray val);  
    public BitArray Not();  
    public BitArray Or(BitArray a);  
    ...  
}
```

# Streaming

# Streaming Framework

- System.IO enthält Typen für Ein- und Ausgabe
- Basisklasse Stream definiert abstraktes Protokoll für byteorientiertes Schreiben und Lesen
- Spezialisierungen für unterschiedliche Medien



- Streams unterstützen synchrones und *asynchrones* Protokoll
- Readers und Writers für Formatierung

# Abstract Klasse Stream

```
public abstract class Stream : MarshalByRefObject,
IDisposable {

    public abstract bool CanRead { get; }
    public abstract bool CanSeek { get; }
    public abstract bool CanWrite { get; }

    public abstract int Read(out byte[] buff,
        int offset, int count);
    public abstract void Write(byte[] buff,
        int offset, int count);
    public virtual int ReadByte();
    public virtual void WriteByte(byte value);

    public virtual IAsyncResult BeginRead(...);
    public virtual IAsyncResult BeginWrite(...);
    public virtual int EndRead(...);
    public virtual int EndWrite(...);

    public abstract long Length { get; }
    public abstract long Position { get; set; }
    public abstract long Seek(long offset,
        SeekOrigin origin);
    public abstract void Flush();
    public virtual void Close();
    ...
}
```

■ grundlegende Eigenschaften  
des Streams

■ Synchrones Lesen und  
Schreiben

■ Asynchrone Lese und  
Schreibmethoden

■ Länge  
und aktuelle Position

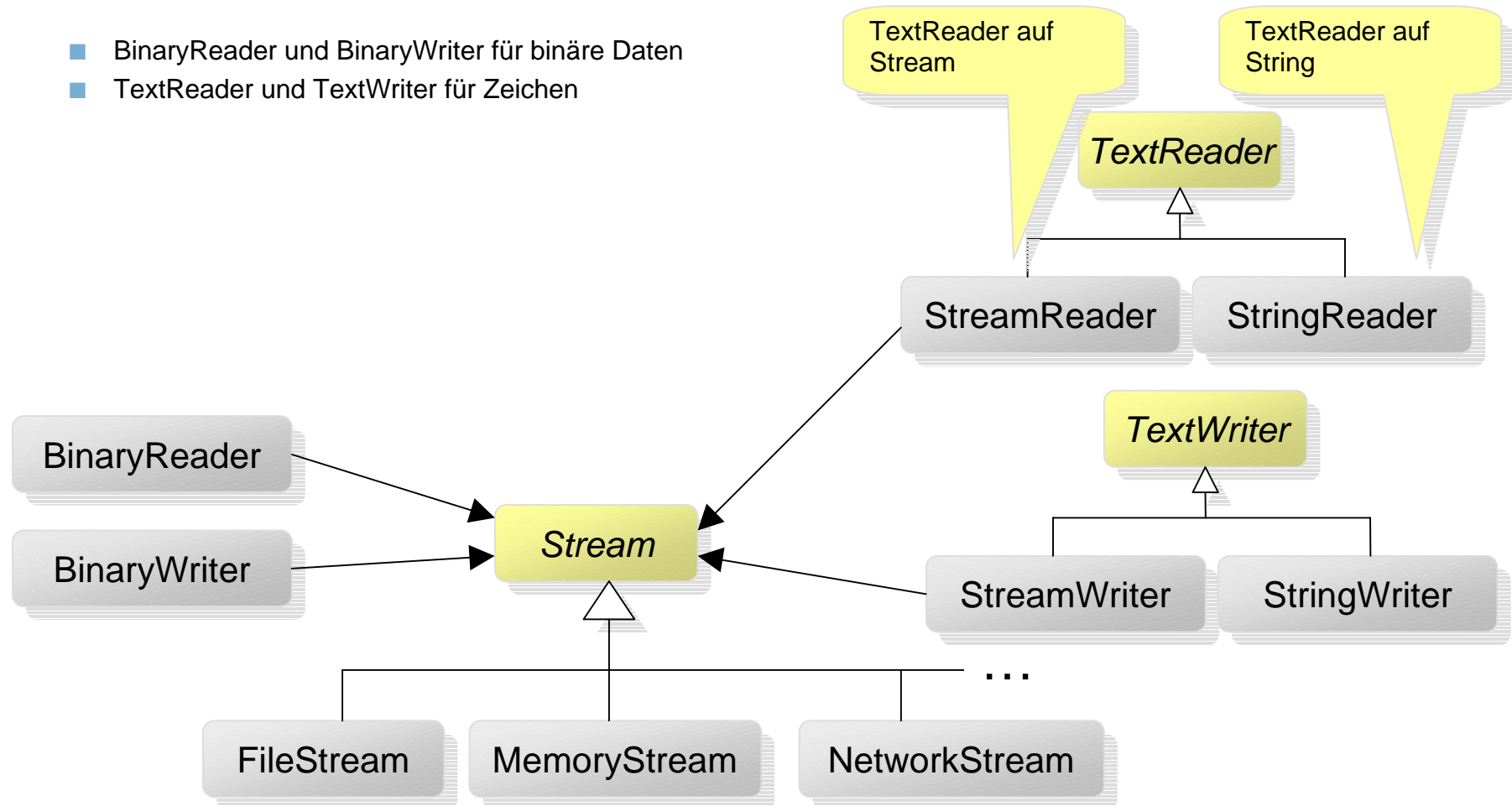
■ Positionierung

■ Flush und Close

# Readers und Writers

## ■ Readers und Writers übernehmen die Formatierung

- BinaryReader und BinaryWriter für binäre Daten
- TextReader und TextWriter für Zeichen



# Klassen TextReader und TextWriter

```
public abstract class TextReader :
MarshalByRefObject, IDisposable {
    public virtual int Read();
    public virtual int Read(out char[] buf, int idx,
int count);
    public virtual int ReadBlock(out char[] buf, int
index, int count);
    public virtual string ReadLine();
    public virtual string ReadToEnd();
    public virtual int Peek();
    ...
}
```

- Verschiedene Leseoperationen

```
public abstract class TextWriter :
MarshalByRefObject, IDisposable {
    public virtual void Write(bool val);
    public virtual void Write(string s);
    public virtual void Write(int val);
    ... // + overloads methods
    public virtual void WriteLine();
    public virtual void WriteLine(bool val);
    ... // + overloaded methods

    public virtual string NewLine { get; set; }

    public abstract Encoding Encoding { get; }
    ...
}
```

- Schreiboperationen für alle primitiven Datentypen
- Schreiboperationen mit Linebreak
- Characters für neue Zeile
- verwendetes Encoding



# Beispiel Lesen von Text

```
using System;
using System.IO;
using System.Text; // for encoding definitions
public class StreamReaderExample {
    public static void Main() {
```

Encoding.ASCII (7Bit!)  
Encoding.Unicode  
Encoding.UTF8  
Encoding.GetEncoding("windows-1252")

## ■ FileStream erzeugen

```
FileStream fs;
fs = new FileStream(@"c:\log.txt", FileMode.Open, FileAccess.Read);
```

## ■ StreamReader für Textausgabe erzeugen

```
TextReader sr = new StreamReader(fs, Encoding.GetEncoding("windows-1252"));
```

## ■ Text zeilenweise lesen

```
string line = null;
while ((line = sr.ReadLine()) != null)
    Console.WriteLine(line);
```

## ■ oder als Ganzes

```
String s = sr.ReadToEnd();
Console.WriteLine(s);
```

## ■ Reader und Stream schliessen

```
sr.Close(); fs.Close();
}
}
```

# Beispiel Anhängen von Text an Datei

```
using System;
using System.IO;
using System.Text; // for encoding definitions
public class StreamWriterExample {
    public static void Main() {
```

## ■ FileStream erzeugen

```
FileStream fs;
fs = new FileStream("log.txt", FileMode.OpenOrCreate, FileAccess.Write);
```

## ■ StreamWriter für Textausgabe erzeugen

```
StreamWriter sw = new StreamWriter(fs, Encoding.Unicode);
```

## ■ Text am Schluss anhängen

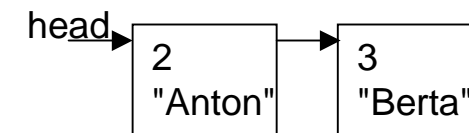
```
sw.BaseStream.Seek(0, SeekOrigin.End);
sw.WriteLine("log entry 1");
sw.WriteLine("log entry 2");
```

## ■ Writer und Stream schliessen

```
sw.Close();
fs.Close();
}
}
```

# Serialisierung von Datenstrukturen

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
class Test {
    static void Write(IList<string> head) {
        FileStream s = new FileStream("MyFile", FileMode.Create);
        IFormatter f = new BinaryFormatter();
        f.Serialize(s, head);
        s.Close();
    }
    static IList<string> Read() {
        FileStream s = new FileStream("MyFile", FileMode.Open);
        IFormatter f = new BinaryFormatter();
        IList<String> head = f.Deserialize(s) as IList<string>;
        s.Close();
        return head;
    }
    static void Main() {
        IList<string> list = new List<string>();
        list.Add("Anton"); list.Add("Berta");
        Write(list);
        IList<string> newList = Read();
    }
}
```



# Asynchrone Operationen

- BeginRead und BeginWrite setzen asynchrone Lese- und Schreibbefehle ab

```
public virtual IAsyncResult  
BeginRead(  
    byte[] buffer,  
    int offset,  
    int count,  
    AsyncCallback callback,  
    object state );  
public virtual IAsyncResult  
BeginWrite(  
    byte[] buffer,  
    int offset,  
    int count,  
    AsyncCallback callback,  
    object state );
```

- BeginRead und BeginWrite werden mit AsyncCallback-Delegate aufgerufen
- Delegate wird nach Beendigung des Aufrufs mit IAsyncResult aufgerufen

```
public delegate void AsyncCallback(  
    IAsyncResult ar  
);
```

```
public interface IAsyncResult {  
    object AsyncState {get;}  
    WaitHandle AsyncWaitHandle {get;}  
    bool CompletedSynchronously {get;}  
    bool IsCompleted {get;}  
};
```

- Mit EndRead und EndWrite wird asynchroner Aufruf beendet

```
public virtual int EndRead(  
    IAsyncResult asyncResult );  
public virtual void EndWrite(  
    IAsyncResult asyncResult);
```

# Beispiel Asynchrones Read

## ■ Felder für Stream, Buffer und Callback deklarieren

```
namespace AsyncIO {  
    public class AsyncIOTester {  
        private Stream inputStream;  
        private byte[] buffer = new byte[256];  
        private AsyncCallback callback;
```

## ■ BeginRead bei Input-Stream mit Callback aufrufen

```
        public static void Main() {  
            inputStream = File.OpenRead("...");  
            callback = new AsyncCallback(this.OnCompletedRead)  
            inputStream.BeginRead(buffer, 0, buffer.Length, callback, null);  
            ... // continue with some other tasks  
        }
```

## ■ Callback-Methode:

- Anzahl der gelesenen Bytes mit EndRead abfragen
- Daten verarbeiten

```
        void OnCompletedRead(IAsyncResult result) {  
            int bytesRead = inputStream.EndRead(result);  
            ... // process the data read  
        }  
        ...
```

# Datei und Verzeichnis Operationen

# Dateien und Verzeichnisse

- Namensräume System.IO.File und System.IO.Directory
- für Arbeiten mit Dateien und Verzeichnissen

## File:

- *statische Methoden* für Bearbeitung von Dateien

## Directory:

- *statische Methoden* für Bearbeitung von Verzeichnissen

## FileInfo:

- repräsentiert ein File

## DirectoryInfo:

- Repräsentiert ein Verzeichnis

# Klasse File

statische Methoden

```
public sealed class File {  
    public static FileStream Open(string path, FileMode mode);  
    public static FileStream Open(string path, FileMode m, FileAccess a);  
    public static FileStream Open(string p, FileMode m, FileAccess a, FileShare s);  
    public static FileStream OpenRead(string path);  
    public static FileStream OpenWrite(string path);  
    public static StreamReader OpenText(string path); // returns Reader for reading text  
    public static StreamWriter AppendText(string path); // returns Writer for appending text  
    public static FileStream Create(string path); // create a new file  
    public static FileStream Create(string path, int bufferSize);  
    public static StreamWriter CreateText(string path);  
    public static void Move(string src, string dest);  
    public static void Copy(string src, string dest); // copies file src to dest  
    public static void Copy(string src, string dest, bool overwrite);  
    public static void Delete(string path);  
    public static bool Exists(string path);  
    public static FileAttributes GetAttributes(string path);  
    public static DateTime GetCreationTime(string path);  
    public static DateTime GetLastAccessTime(string path);  
    public static DateTime GetLastWriteTime(string path);  
    public static void SetAttributes(string path, FileAttributes fileAttributes);  
    public static void SetCreationTime(string path, DateTime creationTime);  
    public static void SetLastAccessTime(string path, DateTime lastAccessTime);  
    public static void SetLastWriteTime(string path, DateTime lastWriteTime);  
}
```



# Klasse Directory

statische Methoden

```
public sealed class Directory {
    public static DirectoryInfo CreateDirectory(string path); // creates directories
    public static void Move(string src, string dest); // moves directory src to dest
    public static void Delete(string path); // deletes an empty directory
    public static void Delete(string path, bool recursive); // deletes non empty directory
    public static bool Exists(string path); // checks if directory exists
    public static string[] GetFiles(string path); // returns all file names in path
    public static string[] GetFiles(string path, string searchPattern);
    public static string[] GetDirectories(string path); // returns all directory names in
path
    public static string[] GetDirectories(string path, string searchPattern);
    public static DirectoryInfo GetParent(string path); // returns the parent directory
    public static string GetCurrentDirectory(); // returns current working directory
    public static void SetCurrentDirectory(string path); // sets current working directory
    public static string[] GetLogicalDrives(); // returns names of logical drives ("c:\")
    public static DateTime GetCreationTime(string path); // returns creation date & time
    public static DateTime GetLastAccessTime(string path);
    public static DateTime GetLastWriteTime(string path);
    public static void SetCreationTime(string path, DateTime t); // sets creation date &
time
    public static void SetLastAccessTime(string path, DateTime t);
    public static void SetLastWriteTime(string path, DateTime t);
}
```

# Klasse FileInfo

ähnlich zu File aber wird instanziiert  
-> Zugriffsrechtüberprüfung nur einmal  
-> effizienter wenn mehrere Operationen

```
public sealed class FileInfo : FileSystemInfo {
    //----- constructors
    public FileInfo(string fileName); // creates a new FileInfo object for a file fileName
    //----- properties
    public override string Name { get; } // name of this file
    public long Length { get; } // size of this file
    public override bool Exists { get; } // indicates if this file exists
    public DirectoryInfo Directory { get; } // directory containing this file
    public string DirectoryName { get; } // name of the directory containing this
file
    //----- methods
    public FileStream Open(FileMode m); // open a FileStream to this file
    public FileStream Open(FileMode m, FileAccess a);
    public FileStream Open(FileMode m, FileAccess a, FileShare s);
    public FileStream OpenRead(); // opens a read-only FileStream to this file
    public FileStream OpenWrite(); // open a write-only FileStream to this file
    public StreamReader OpenText(); // returns a UTF8 reader for reading text
    public StreamWriter AppendText(); // returns a StreamWriter for appending text
    public FileStream Create(); // returns FileStream to this newly created file
    public StreamWriter CreateText(); // returns Writer to this newly created text file
    public void MoveTo(string dest); // move this file to dest
    public FileInfo CopyTo(string dest); // copies this file to dest
    public FileInfo CopyTo(string dest, bool overwrite); // copies to and overwrites dest
    public override Delete(); // deletes this file
    public override string ToString(); // returns entire path of this file
}
```

# Klasse DirectoryInfo

ähnlich zu Directory aber wird instanziiert  
-> Zugriffsrechtüberprüfung nur einmal  
-> effizienter wenn mehrere Operationen

```
public sealed class DirectoryInfo : FileSystemInfo {
    //----- constructor
    public DirectoryInfo(string path);           // path specifies the directory
    //----- properties
    public override string Name { get; }        // returns directory name without the path
    public override bool Exists { get; }        // indicates if this directory exists
    public DirectoryInfo Parent { get; }        // returns the parent directory
    public DirectoryInfo Root { get; }          // returns the root directory
    //----- methods
    public void Create();                        // create a new directory, if it does not exist
    public DirectoryInfo CreateSubdirectory(string path); // creates a subdirectory
    public void MoveTo(string destDir);         // moves this directory to destDir
    public void Delete();                       // deletes this directory, if it is empty
    public void Delete(bool recursive);         // deletes this directory and its contents
    public FileInfo[] GetFiles();               // returns all files in this directory
    public FileInfo[] GetFiles(string pattern); // returns matching files in this directory
    public DirectoryInfo[] GetDirectories();    // returns all directories in this directory
    public DirectoryInfo[] GetDirectories(string pattern); // returns all matching
directories
    public FileSystemInfo[] GetFileSystemInfos(); // returns all files and
directories
    public FileSystemInfo[] GetFileSystemInfos(string pattern); // returns files and
directories for pattern
    public override ToString();                // returns the path given in the constructor
}
```

# Beispiel Dateien und Verzeichnisse

## ■ Ausgeben von Verzeichnissen und Dateien in "c:\\"

Ausgabe

```
using System; using System.IO;
public class DirectoryExample {

    public static void Main() {
        DirectoryInfo dir = Directory.CreateDirectory("c:\\");

        Console.WriteLine("----- Directories -----");
        DirectoryInfo[] dirs = dir.GetDirectories();
        foreach (DirectoryInfo d in dirs)
            Console.WriteLine(d.Name);

        Console.WriteLine("----- Files -----");
        FileInfo[] files = dir.GetFiles();
        foreach (FileInfo f in files)
            Console.WriteLine(f.Name);
    }
}
```

```
----- Directories -----
Documents and Settings
I386
Program Files
System Volume Information
WINNT
----- Files -----
AUTOEXEC.BAT
boot.ini
CONFIG.SYS
IO.SYS
MSDOS.SYS
NTDETECT.COM
ntldr
pagefile.sys
```

# FileSystemWatcher

- Überwachung des Dateisystems mit FileSystemWatcher
- Änderungen werden mit Ereignissen gemeldet

```
public class FileSystemWatcher : Component, ...{  
  
    public FileSystemWatcher(string path);  
  
    public string Path { get; set; }  
  
    public string Filter { get; set; }  
  
    public bool IncludeSubdirectories { get; set; }  
  
    public event FileSystemEventHandler Changed;  
    public event FileSystemEventHandler Created;  
    public event FileSystemEventHandler Deleted;  
    public event RenamedEventHandler Renamed;  
  
    public WaitForChangedResult WaitForChanged(  
        WatcherChangeTypes types);  
  
}
```

- Pfad und Filter, um zu überwachenden Teils des Dateisystems zu bestimmen
- Unterverzeichnisse einschliessen
- Ereignisse, die Änderungen melden
- auf bestimmte Änderung warten

# Beispiel FileSystemWatcher

## ■ Ereignismethoden definieren

```
public static void Changed(object sender, FileSystemEventArgs args) {  
    Console.WriteLine("Changed -> {0}", args.Name);  
}  
public static void Created(object sender, FileSystemEventArgs args) {...}  
public static void Deleted(object sender, FileSystemEventArgs args) {...}  
public static void Renamed(object sender, RenamedEventArgs args) {...}
```

## ■ FileWatcher erzeugen und Ereignismethoden registrieren

```
public static void Main() {  
    FileSystemWatcher fsw = new FileSystemWatcher("c:\\");  
    fsw.IncludeSubdirectories = true;  
    fsw.Changed += new FileSystemEventHandler(Changed);  
    fsw.Created += new FileSystemEventHandler(Created);  
    ...  
}
```

## ■ Filter setzen und auf Ereignisse warten

```
fsw.Filter = "*.cs";  
while ( ... ) fsw.WaitForChanged(WatcherChangeTypes.All);  
}
```

- Umfangreiche Klassenbibliothek
  - allgemeine Klassen (z.B. Math, DateTime)
  
- Behälter für Datenstrukturen (Collections)
  - Array, List, Dictionary, etc.
  
- Datei-Zugriff: Trennung Medium, Zugriffspfad
  - Streams: FileStream, MemoryStream, etc.
  - Reader: StreamReader
  - Writer: StreamWriter



# Fragen?

