Base Class Library (auszugsweise)

Überblick

Namespaces	*Klassen*
System	Object, Type, Int32, Boolean, Array, Enum, String, Random, Math, ...
Collections	ArrayList, BitArray, Hashtable, SortedList, Stack, Queue, ...
Drawing	Bitmap, Color, Font, Pen, Point, Rectangle, Region, ...
Design	
Drawing2D	
Imaging	
Printing	
Text	
IO	Stream, File, Directory, FileStream, StreamReader, StreamWriter, ...
Net	WebClient, WebRequest, WebResponse, ...
Reflection	Assembly, FieldInfo, MethodInfo, PropertyInfo, EventInfo, ...
Runtime	
InteropServices	DllImportAttribute, StructLayoutAttribute, ...
Remoting	
Serialization	Formatter, ISerializable, ...
Security	CodeAccessPermission, PermissionSet, SecurityException, ...
Text	StringBuilder, Encoder, Decoder, ...
Threading	Thread, Monitor, Mutex, Timer, ...
Web	HttpRequest, HttpRespose, ..
Windows	
Forms	Form, Button, CheckBox, Menu, RadioButton, ListBox, ...
XML	XmlDoxument, XmlElement, XmlReader, ...

Klasse Math


```
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);

    public static double Sqrt(double x);
    public static double Pow(double x, double y);  // x^y
    public static double Exp(double x)          // e^x
    public static double Log(double x);         // log_e x
    public static double Log10(double x);       // log_10 x

    public static double Sin(double x);         // Winkel im Bogenmaß (Winkel * π / 180)
    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.MaxVal
    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 String (string)


```
public sealed class String : ICloneable, IComparable, IEnumerable {
    public String(char[] a);
    public String(char[] a, int start, int len);

    public static string Copy(string s);
    public static string Format(String s, params object[]);
    public static string Join(string separator, string[] parts);
    public static bool operator ==(bool a, bool b);
    public static bool operator !=(bool a, bool b);

    public int Length { get; }
    public char this[int i] { get; }

    public override bool Equals(object o);
    public int CompareTo(object o);
    public static int CompareOrdinal(string a, string b);
    public void CopyTo(int from, char[] dest, int to, int len);
    public bool StartsWith(string s);
    public bool EndsWith(string s);
    public int IndexOf(T s);                    // T ... string, char
    public int LastIndexOf(T s);
    public string Substring(int pos);           // bis zum Ende
    public string Substring(int pos, int len);
    public string[] Split(params char[]);
    public char[] ToCharArray();
    ...
}
```

Konversion zw. String und num.Typen


```
public sealed class Convert {                   // namespace System
    public static string    ToString(T x);      // T ... jeder numerische Typ

    public static bool      ToBoolean(string s);
    public static byte      ToByte(string s);
    public static sbyte     ToSByte(string s);
    public static char      ToChar(string s);
    public static short     ToInt16(string s);
    public static int       ToInt32(string s);
    public static long      ToInt64(string s);
    public static ushort    ToUInt16(string s);
    public static uint      ToUInt32(string s);
    public static ulong     ToUInt64(string s);
    public static float     ToSingle(string s);
    public static double    ToDouble(string s);
    public static decimal   ToDecimal(string s);
    ...
}
```

Beispiel

```
    string s = Convert.ToString(123);       // "123"
    s = 123.ToString();                     // "123" (Alternative)

    int i = Convert.ToInt32(s);             // 123
```

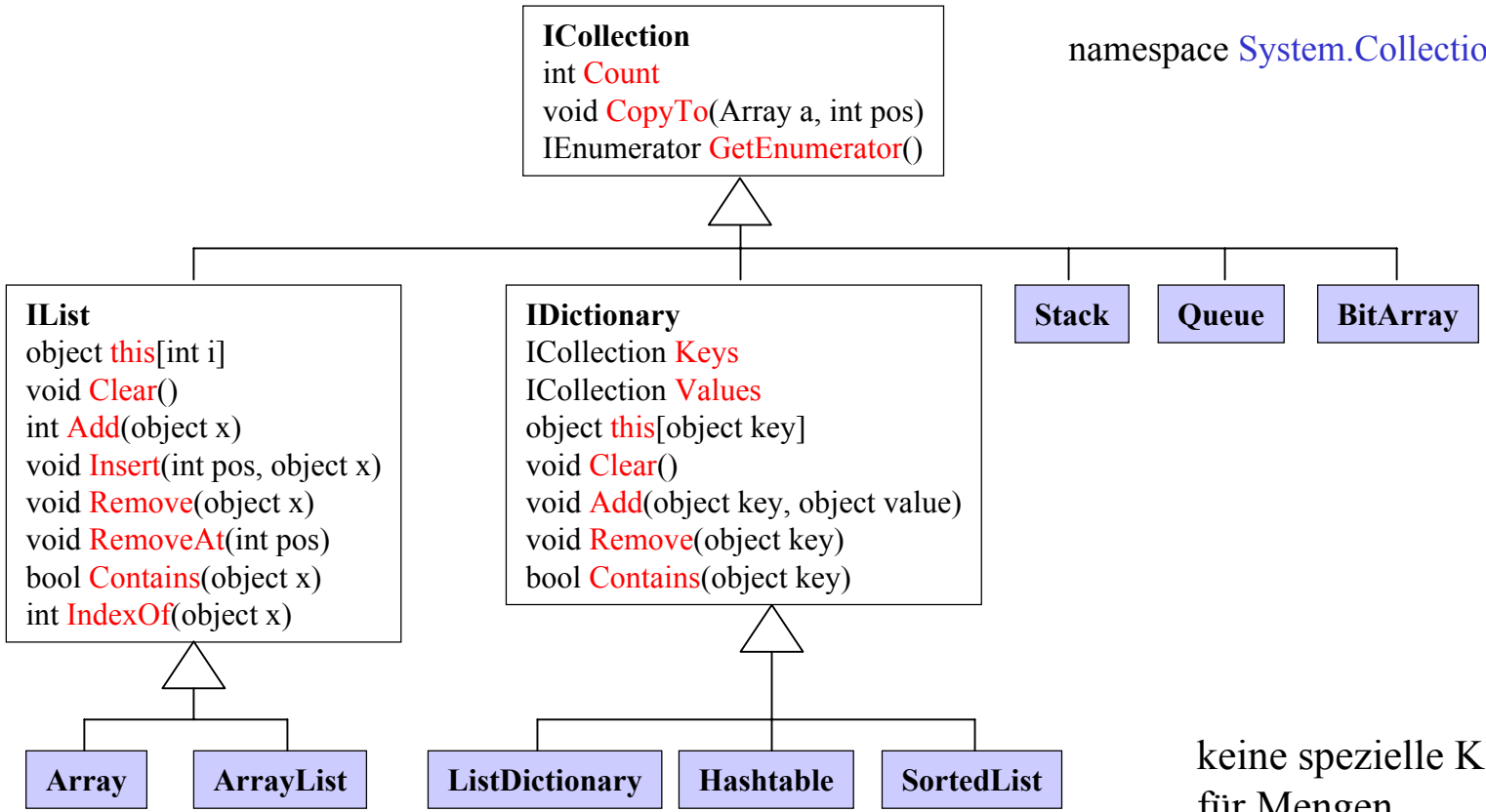
Klasse StringBuilder


```
public sealed class StringBuilder {                 // namespace System.Text
    public StringBuilder();                         // Anfangskapazität = 16, wächst dynamisch
    public StringBuilder(int initCapacity);
    public StringBuilder(string s);
    public StringBuilder(string s, int from, int len);

    public int Length { get; set; }
    public int Capacity { get; set; }
    public char this[int i] { get; set; }

    public StringBuilder Append(T x);               // T ... string oder numerischer Typ
    public StringBuilder Insert(int pos, T x);
    public StringBuilder Remove(int pos, int len);
    public StringBuilder Replace(T x, T y);         // T ... string, char
    public bool Equals(object x);
    public string ToString();
    ...
}
```

Beispiel

```
StringBuilder b = new StringBuilder("A.C");
b.Insert(2, "B.");
b.Replace('.', '/');
Console.WriteLine(b.ToString());  // A/B/C
```

Collection-Klassen (Auszug)



namespace System.Collections

keine spezielle Klasse für Mengen

Klasse Array

Alle Arrays sind von dieser Klasse abgeleitet

```
public abstract class Array : IList, ICloneable, IEnumerable{          // namespace System
    public int Length { get; }          // Gesamtanzahl aller Elemente aller Dimensionen
    public int Rank { get; }            // Anzahl der Dimensionen

    public static void Clear(Array a, int pos, int length);
    public static void Copy(Array src, int from, Array dst, int to, int len);
    public static int IndexOf(Array a, object x);                  // Varianten
    public static int LastIndexOf(Array a, object x);              // Varianten
    public static void Reverse(Array a);                           // Varianten
    public static void Sort(Array a);                              // Varianten
    public static int BinarySearch(Array a, object x);             // Varianten

    public virtual object Clone();
    public virtual bool Equals(object x);
    public int GetLength(int dimension);
    ...
}
```

Beispiel

```
int[] a = {17, 3, 5, 9, 6};
Array.Sort(a);
Array.Reverse(a);
foreach (int x in a) Console.Write("{0} ", x);                     // 17 9 6 5 3
```

Klasse ArrayList

Dynamisch wachsende Arrays (Mischung zwischen Arrays und Listen)

```
public class ArrayList : IList, IEnumerable, ICloneable {     // namespace System.Collections
    public ArrayList();                                        // default capacity = 16
    public ArrayList(int initCapacity);
    public ArrayList(ICollection c);

    public virtual int      Count { get; }
    public virtual int      Capacity { get; set; }
    public virtual object   this[int i] { get; set; }

    public virtual void     Clear();
    public virtual int      Add(object x);
    public virtual void     Insert(int pos, object x);
    public virtual void     Remove(object x);
    public virtual void     RemoveAt(int pos);
    public virtual object   Clone();
    public virtual bool     Equals(object x);
    public virtual bool     Contains(object x);
    public virtual int      IndexOf(object x);             // Varianten
    public virtual void     Reverse();                     // Varianten
    public virtual void     Sort();                        // Varianten
    public virtual int      BinarySearch(object x);        // Varianten
    ...
}
```

ArrayList-Beispiel


```
using System;
using System.Collections;

class Test {

    static void Main(string[] arg) {
        ArrayList a = new ArrayList();
        foreach (string s in arg) a.Add(s);
        a.Sort();
        Console.Write("a = ");
        for (int i = 0; i < a.Count; i++) Console.Write("{0} ", a[i]);
        Console.WriteLine();
        int pos = a.BinarySearch("Tom");
        if (pos >= 0)
            Console.WriteLine("Tom found at {0}", pos);
        else
            Console.WriteLine("Tom not found");
    }
}
```

Aufruf: Test Anton Tom Berta Gustav

Ausgabe:

```
    a = Anton Berta Gustav Tom
    Tom found at 3
```

Klasse ListDictionary

IDictionary-Implementierung als verkettete Liste (nur für kleine Datenmengen)

```
public class ListDictionary : IDictionary, IEnumerable {  // System.Collections.Specialized
    public ListDictionary();
    public int Count { get; }
    public ICollection Keys { get; }
    public ICollection Values { get; }
    public object this[object key] { get; set; }

    public void Clear();
    public void Add(object key, object value);
    public void Remove(object key);
    public bool Contains(object key);
    public virtual bool Equals(object x);
    public IDictionaryEnumerator GetEnumerator();
    ...
}
```

Beispiel

```
ListDictionary population = new ListDictionary();
population["Wien"] = 1592600;   // dasselbe wie population.Add("Wien", 1592600);
population["Oberoesterreich"] = 1383800;
population["Salzburg"] = 508400;
...
foreach (DictionaryEntry x in tab) Console.WriteLine("{0} = {1}", x.Key, x.Value);
```

Klasse Hashtable

IDictionary-Implementierung als Hash-Tabelle

```
public class Hashtable : IDictionary, IEnumerable, ISerializable, ICloneable { // System.Collections
      public HashTable();                              // Varianten
      //... gleiche Members wie ListDictionary; zusätzlich:
      public virtual object Clone();
      public virtual bool ContainsKey(object key);
      public virtual bool ContainsValue(object value);
}
```

Gleich zu benutzen wie ListDictionary

```
      Hashable population = new Hashtable();
      population["Wien"] = 1592600;
      population["Oberoesterreich"] = 1383800;
      population["Salzburg"] = 508400;
      ...
      foreach (DictionaryEntry x in tab) Console.WriteLine("{0} = {1}", x.Key, x.Value);
```

Klasse SortedList

IDictionary-Implementierung als Array, das nach Key sortiert ist

```
public class SortedList : IDictionary, IEnumerable, ICloneable {     // System.Collections
    public SortedList();                                  // Varianten
    //... gleiche Members wie Hashtable; zusätzlich:
    public virtual int Capacity { get; set; }
    public virtual object GetKey(int i);                  // liefert Key[i]
    public virtual object GetByIndex(int i);              // liefert Value[i]
    public virtual void SetByIndex(int i, object value);
    public virtual IList GetKeyList();
    public virtual IList GetValueList();
    public virtual int IndexOfKey(object key);
    public virtual int IndexOfValue(object value); // liefert Index des 1. Vorkommens von value
    public virtual void RemoveAt(int i);
}
```

Gleich zu benutzen wie ListDictionary und Hashtable

```
    SortedList population = new SortedList();
    population["Wien"] = 1592600;
    population["Oberoesterreich"] = 1383800;
    ...
    foreach (DictionaryEntry x in tab) Console.WriteLine("{0} = {1}", x.Key, x.Value);
    // gibt Elemente nach Schlüsseln sortiert aus
```

Klasse Stack


```
public class Stack : ICollection, IEnumerable, ICloneable { // System.Collections
    public Stack();                                  // default capacity = 10
    public Stack(int initCapacity);
    public virtual int Count { get; }
    public virtual void Clear();
    public virtual void Push(object x);
    public virtual object Pop();
    public virtual object Peek();                    // liefert oberstes Element, ohne es zu entfernen
    public virtual bool Contains(object x);
    public virtual object Clone();
    public virtual IEnumerator GetEnumerator();
    public virtual object[] ToArray();
    ...
}
```

Beispiel

```
Stack s = new Stack();
for (int i = 0; i <= 20; i++) s.Push(i);    // Stack wächst bei Bedarf
while (s.Count > 0) Console.Write("{0} ", (int)s.Pop());      // liefert 20 19 18 17 ... 0
```

Klasse Queue


```
public class Queue : ICollection, IEnumerable, ICloneable { // System.Collections
    public Queue();                                          // Varianten
    public virtual int Count { get; }
    public virtual void Clear();
    public virtual void Enqueue(object x);
    public virtual object Dequeue();
    public virtual object Peek();                            // liefert vorderstes Element, ohne es zu entfernen
    public virtual bool Contains(object x);
    public virtual object Clone();
    public virtual IEnumerator GetEnumerator();
    public virtual object[] ToArray();
    ...
}
```

Beispiel

```
Queue q = new Queue();
q.Enqueue("one");
q.Enqueue("two");
q.Enqueue("three");
foreach (string s in q) Console.Write(s + " ");  // one two three
// q noch immer voll
Console.WriteLine();
while (q.Count > 0) Console.Write((string)q.Dequeue() + " ");   // one two three
// q leer
```

Klasse BitArray


```
public sealed class BitArray : ICollection, IEnumerable, ICloneable { // System.Collections
    public BitArray(int capacity);                  // Varianten
    public int Length { get; set; }                 // max. Anzahl von Elementen
    public bool this[int i] { get; set; }

    public BitArray And(BitArray x);                // beide BitArrays müssen gleich groß sein
    public BitArray Or(BitArray x);
    public BitArray Xor(BitArray x);
    public BitArray Not();
    public void SetAll(bool val);
    public object Clone();
    public IEnumerator GetEnumerator();
}
```

Beispiel

```
    BitArray a = new BitArray(16);              // alle Elemente sind false
    BitArray b = new BitArray(16);
    a[3] = a[4] = a[5] = true;
    b[4] = b[5] = b[6] = true;
    a.And(b);                                   // a[4] und a[5] sind true
    b.Xor(a);                                   // nur b[6] ist true
```

Struct BitVector32

Leichtgewichtiger Typ für Bitarrays der Länge 32

```
public struct BitVector32 {                    // namespace System.Collections.Specialized
    public BitVector32(int val);               // val wird als Bitmuster interpretiert
    public int Data { get; }                   // liefert das Bitmuster als int
    public bool this[int i] { get; set; }
    public override bool Equals(object x);
}
```

Beispiel

```
    BitVector32 v = new Bitvector32(12);       // v[2] und v[3] gesetzt (0000 0000 0000 1100)
    v[0] = v[1] = true;
    Console.WriteLine(v.Data);                 // 15
```

Iterieren über Collections

foreach-Schleife anwendbar, wenn Klasse *IEnumerable* implementiert

(alle Collections implementieren *IEnumerable*)

```
public interface IEnumerable {
    IEnumerator GetEnumerator();
}

public interface IEnumerator {
    object Current { get; }
    bool MoveNext();                    // true, wenn er weiterbewegt werden konnte
    void Reset();
}
```

Beispiel eines Iterators


```
class Node {
    public object val;
    public Node next;
}

class MyList : IEnumerable {
    Node head = null;
    public void Add(object x) {...}
    public object Remove() {...}
    public IEnumerator GetEnumerator() { return new MyEnumerator(this); }

    private class MyEnumerator : IEnumerator {
        MyList list;
        Node cur;

        public MyEnumerator(MyList list) { this.list = list; cur = null; }

        public object Current { get { if (cur == null) return null; else return cur.val; }}

        public bool MoveNext() {
            if (cur == null) cur = list.head; else cur = cur.next;
            return cur != null;
        }

        public void Reset() { cur = null; }
    }
}
```

Iterieren mit einer foreach-Schleife


```
class Enumerators {

    static void Main() {
        MyList list = new MyList();
        list.Add("Anton");
        list.Add("Berta");
        list.Add("Caesar");
        foreach (string s in list) Console.WriteLine(s);
    }
}
```

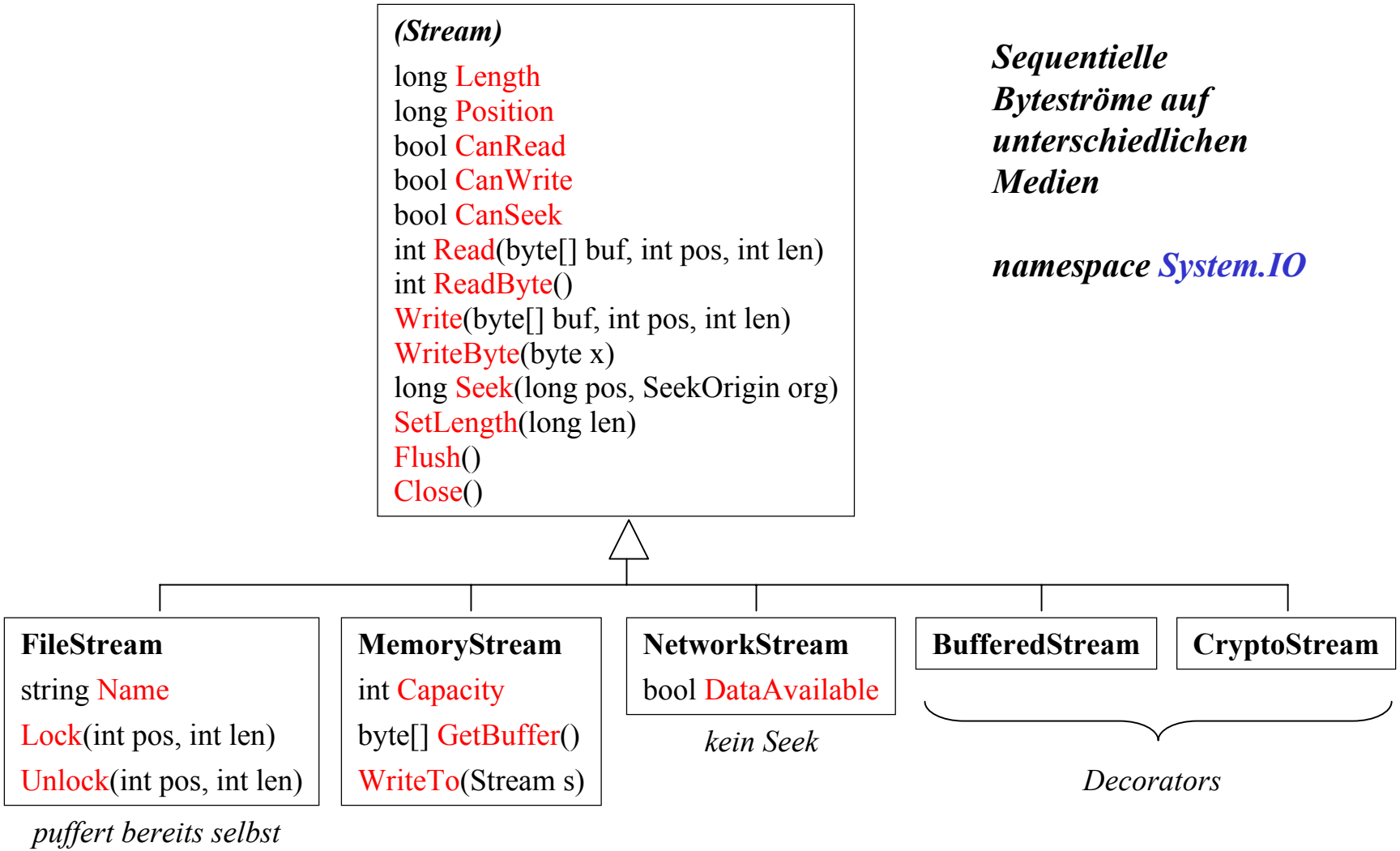
Ausgabe

```
Caesar
Berta
Anton
```

Erspart Cast, der mit Enumerator nötig wäre

```
IEnumerator enumerator = list.GetEnumerator();
while (enumerator.MoveNext())
    Console.WriteLine((string)enumerator.Current);
```

Stream-Klassen



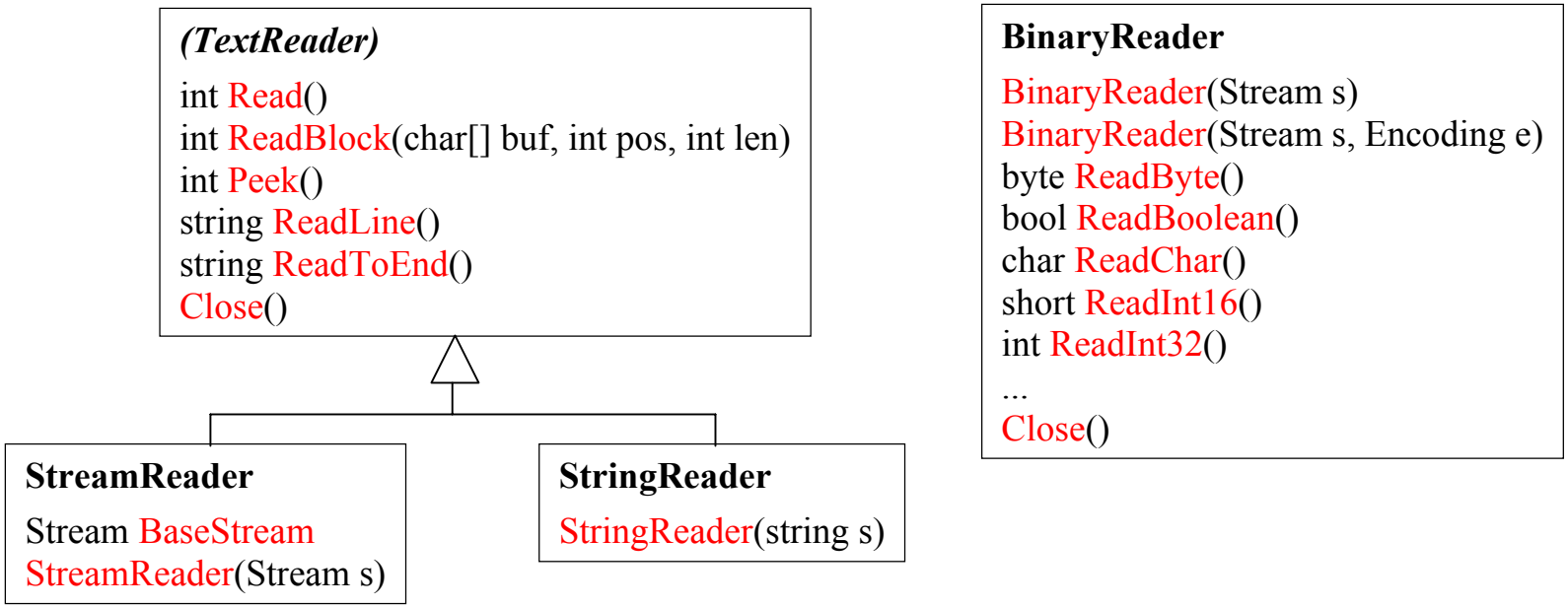
Sequentielle Byteströme auf unterschiedlichen Medien

namespace System.IO

Beispiel: FileStream


```
using System;
using System.IO;

class Test {

    static void Copy(string from, string to, int pos) {
        try {
            FileStream sin = new FileStream(from, FileMode.Open);
            FileStream sout = new FileStream(to, FileMode.Create);
            sin.Seek(pos, SeekOrigin.Begin);       // Seek hinter Dateiende => Position = Dateiende
            int ch = sin.ReadByte();
            while (ch >= 0) {
                sout.WriteByte((byte)ch);
                ch = sin.ReadByte();
            }
            sin.Close();
            sout.Close();
        } catch (FileNotFoundException e) {
            Console.WriteLine("-- file {0} not found", e.FileName);
        }
    }

    static void Main(string[] arg) {
        Copy(arg[0], arg[1], 10);
    }
}
```

Reader-Klassen

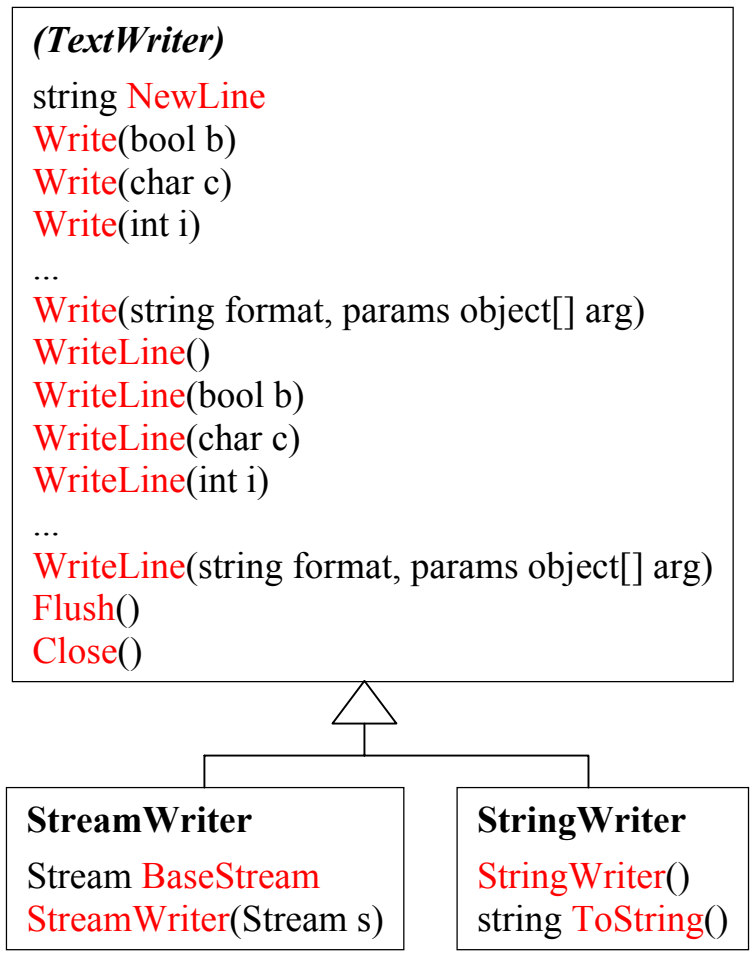


TextReader	liest char-basiert
BinaryReader	liest Standardtypen im Binärformat

Leider können nicht mehrere Reader gleichzeitig auf einem Stream arbeiten.

Es gibt keine formatierte Eingabe von Zahlen, Wörtern, etc.
Auch keinen Tokenizer.

Writer-Klassen



Formatierte Textausgabe
(nur sequentiell; kein Seek)

BinaryWriter

Stream BaseStream
BinaryWriter(Stream s)
BinaryWriter(Stream s, Encoding e)
Write(bool b)
Write(char c)
Write(int i)
...
long Seek(int pos, SeekOrigin org)
Flush()
Close()

Ausgabe von Standardtypen im Binärformat

Beispiel StreamWriter


```
using System;
using System.IO;

class Test {

    // writes command line arguments to a file
    static void Main(string[] arg) {
        FileStream s = new FileStream("xxx.txt", FileMode.Create);
        StreamWriter w = new StreamWriter(s);
        for (int i = 0; i < arg.Length; i++)
            w.WriteLine("arg[{0}] = {1}", i, arg[i]);
        w.Close();
    }
}
```

Klassen File und FileInfo

Statische Utility-Operationen auf Dateien

File

static FileStream OpenRead(string path)
static FileStream OpenWrite(string path)
static FileStream Create(string path)
static StreamReader OpenText(string path)
static StreamWriter CreateText(string path)
static Delete(string path)
static bool Exists(string path)
static Copy(string srcPath, string destPath)
static Move(string srcPath, string destPath)
static FileAttributes GetAttributes(string path)
static SetAttributes(string path, FileAttributes a)
static DateTime GetCreationTime(string path)
...

Dasselbe auf Datei-Objekten

FileInfo

FileInfo(string path)
string Name
string FullName
string DirectoryName
DirectoryInfo Directory
int Length
FileAttributes Attributes
DateTime CreationTime
bool Exists
FileStream OpenRead()
FileStream OpenWrite()
FileStream Create()
StreamReader OpenText()
StreamWriter CreateText()
Delete()
CopyTo(string path)
MoveTo(string path)
...

Klassen Directory und DirectoryInfo

Statische Utility-Operationen auf Verzeichnissen

Directory

static DirectoryInfo CreateDirectory(string path)
static bool Exists(string path)
static Delete(string path)
static Move(string srcDirName, string dstDirName)
static DirectoryInfo GetParent(string path)
static string[] GetFiles(string path)
static string[] GetDirectories(string path)
static string[] GetLogicalDrives()
static string GetCurrentDirectory()
static DateTime GetCreationTime(string path)
...

Dasselbe auf Verzeichnis-Objekten

DirectoryInfo

DirectoryInfo(string path)
string Name
string FullName
DirectoryInfo Parent
bool Exists
DateTime CreationTime
FileAttributes Attributes
Create()
DirectoryInfo CreateSubdirectory(string parent)
Delete()
MoveTo(string dstDirName)
DirectoryInfo[] GetDirectories()
FileInfo[] GetFiles()
...

Beispiel


```
using System;
using System.IO;

class Test {

    // list all files in the path to the current directory
    static void Main() {
        DirectoryInfo d = new DirectoryInfo(".");
        while (d != null) {
            Console.WriteLine(d.FullName);
            FileInfo[] files = d.GetFiles();
            foreach (FileInfo f in files)
                Console.WriteLine("  {0}, created: {1}", f.Name, f.CreationTime.ToString("D"));
            d = d.Parent;
        }
    }
}
```

Ausgabe

```
C:\hm\DotNet
    Attributes.cs, created: Friday, August 17, 2002
    Cloning.cs, created: Thursday, June 14, 2002
C:\hm
    ...
```

"BCL-freundliche" eigene Klassen

Um gut mit anderen BCL-Klassen zusammenzuarbeiten, sollten eigene Klassen

- Methoden *Equals*, *ToString* und *GetHashCode* überschreiben
- Operatoren == und != überschreiben
- Eventuell *ICloneable* implementieren

```
public interface ICloneable {
    object Clone();
}
class MyClass : ICloneable {
    public object Clone() { return MemberwiseClone(); }
}
```

- Eventuell *IComparable* implementieren

```
public interface IComparable {
    int CompareTo(object o);              // -1: this < o,   0: this == o,   1: this > o
}
class Fraction : IComparable {
    int n, d;
    public int CompareTo(object o) {
        return n*((Fraction)o).d - ((Fraction)o).n*d
    }
}
```