

## C# Base Class Library - Ausschnitt

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

## C# Klasse Math

---

```

public sealed class Math {
    public const double PI = 3.141592653589793238462643383279502884197169399375105820974944597;
    public const double E = 2.71828182845904523536028747135266249775724709665859246191414141;

    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);
}

```

## C# 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  
}
```

## C# Klasse Convert - Konversion zwischen 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

```

## C# Klasse DateTime

---

Mit einer Variablen des Typs DateTime lässt sich ein Datum zwischen dem 1. Januar 01 und dem 31. Dezember 9999 behandeln – nach dem gregorianischen Kalender.

### Konstruktoren:

```
public DateTime (int year, int month, int day);
public DateTime (int year, int month, int day, int h, int min, int sec);
public DateTime (int year, int month, int day, int h, int min, int sec, int msec);
public DateTime (long ticks); → 1 Tick = 100 Nanosek. = 0.0000001 sek.
public DateTime (long ticks, DateTimeKind kind);
```

### Eigenschaften:

```
DateTime newDate = DateTime.Today; → Liefert akt. Datum
DateTime newDate = DateTime.Now; → Liefert akt. Datum und akt. Uhrzeit
Console.WriteLine("Jahr = {0}", newDate.Year);
Console.WriteLine("Monat = {0}", newDate.Month);
Console.WriteLine("Tag = {0}", newDate.Day);
Console.WriteLine("({0})", newDate.DayOfWeek.ToString()); *)
Console.WriteLine("Stunde = {0}", newDate.Hour);
Console.WriteLine("Minute = {0}", newDate.Minute);
Console.WriteLine("Sekunde = {0}", newDate.Second);
Console.WriteLine("MilliSek. = {0}", newDate.Millisecond);
```

\*) aus einem gegebenen Datum wird der Wochentag ermittelt

## C# Klasse DateTime

---

### Methoden:

*public long ToFileTime();* → liefert die Anzahl der Ticks seit dem 1. Januar 1601, 12 Uhr  
*ToLongDateString* und *ToShortDateString*  
*ToLongTimeString* und *ToShortTimeString*

### Beispiel:

```
DateTime myDate = new DateTime(2006, 2, 3, 5, 25, 30);  
Console.WriteLine(myDate.ToLongDateString()); // Ausgabe: Mittwoch, 3. Februar 2006  
Console.WriteLine(myDate.ToShortDateString()); // Ausgabe: 03.02.2006
```

### Rechenoperationen mit DateTime-Objekten:

*AddDays()* *AddHours()* *AddMonths()* ...  
Allen Methoden wird ein Wert vom Typ *int* bzw. *double* übergeben,  
der Rückgabewert ist vom Typ *DateTime*

### Beispiel:

```
DateTime now = new DateTime(1995, 8, 2, 23, 0, 0);  
now = now.AddHours(-30); // → 30 Stunden werden abgezogen  
Console.WriteLine(now); // Ausgabe: 01.08.1995 17:00:00
```

## C# Klasse TimeSpan

---

***public DateTime Add(TimeSpan);***

*diese Methode erwartet ein Objekt vom Typ TimeSpan.*

Mit DateTime wird ein definitives Datum beschrieben, mit TimeSpan eine **Zeitspanne**.

**Beispiel:**

*DateTime now = new DateTime(2002, 2, 3, 12, 0, 0),*

*TimeSpan mySpan = new TimeSpan(3, 12, 15);*

*now = now.Add(mySpan);*

*Console.WriteLine(now);*

*Der 3-Arg. Konstruktor übernimmt als Argumente Stunden, Minuten und Sekunden,  
– er beschreibt demnach eine Zeitspanne von drei Stunden, zwölf Minuten, 15 Sekunden,  
die hier im Beispiel mit der Add-Methode aufaddiert wird.*

Die Zeitspanne lässt sich in einem String abbilden, der dem folgenden Format entspricht:

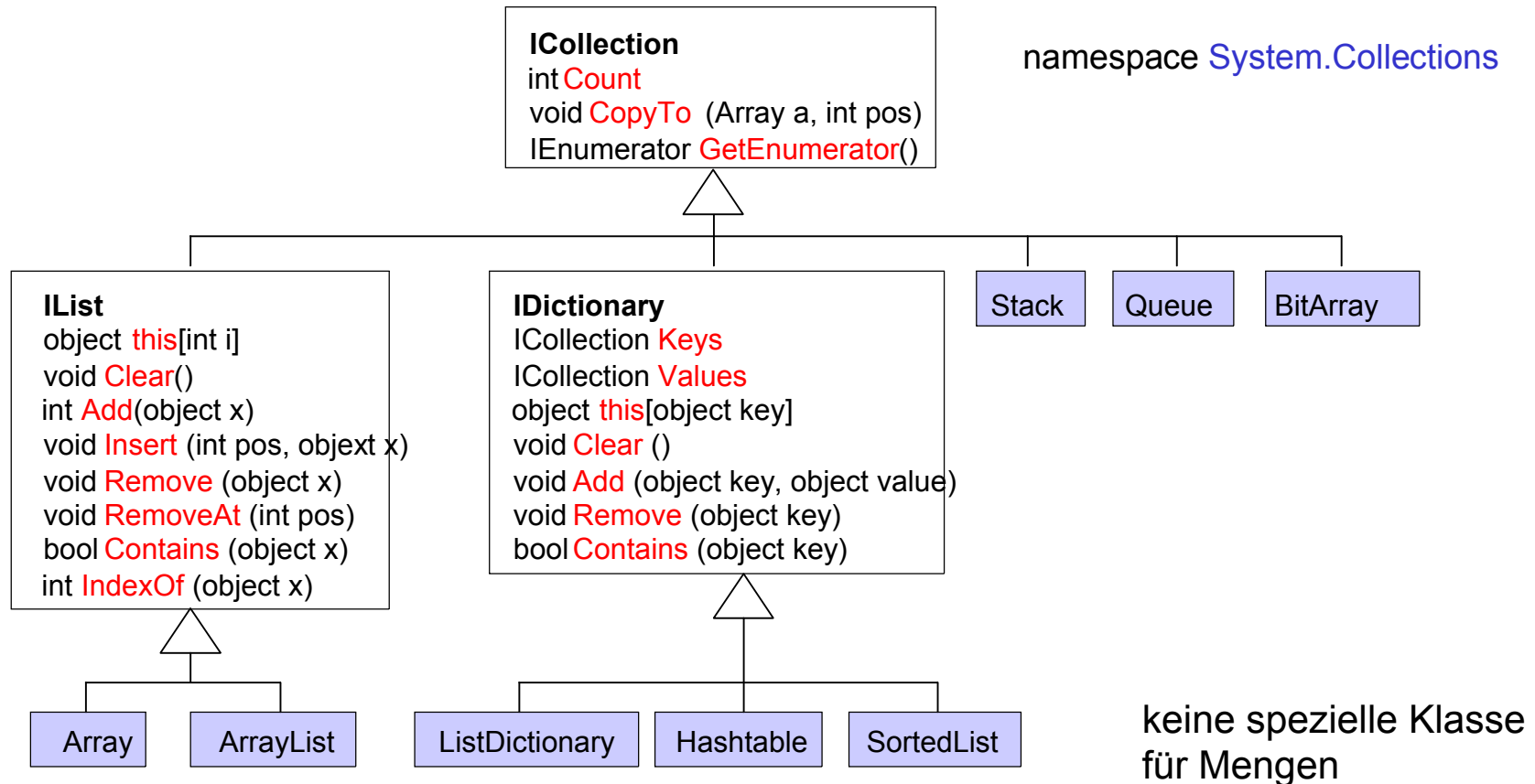
Tag.Stunden:Minuten:Sekunden.Hunderstelsekunden

**Beispiel:**

*TimeSpan sp = new TimeSpan(2, 12, 30, 22, 100); Console.WriteLine(sp.ToString());*

**→** Ausgabe 2.12:30:22.1000000

## C# Collection Klassen - Ausschnitt



## C# 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);
}
```

### Beispiel:

```
Hashtable population = new Hashtable();
population["Wien"] = 1512600;
population["Oberoesterreich"] = 1383800;
population["Salzburg"] = 508400;
...
foreach (DictionaryEntry x in population) Console.WriteLine("{0} = {1}", x.Key, x.Value);
```