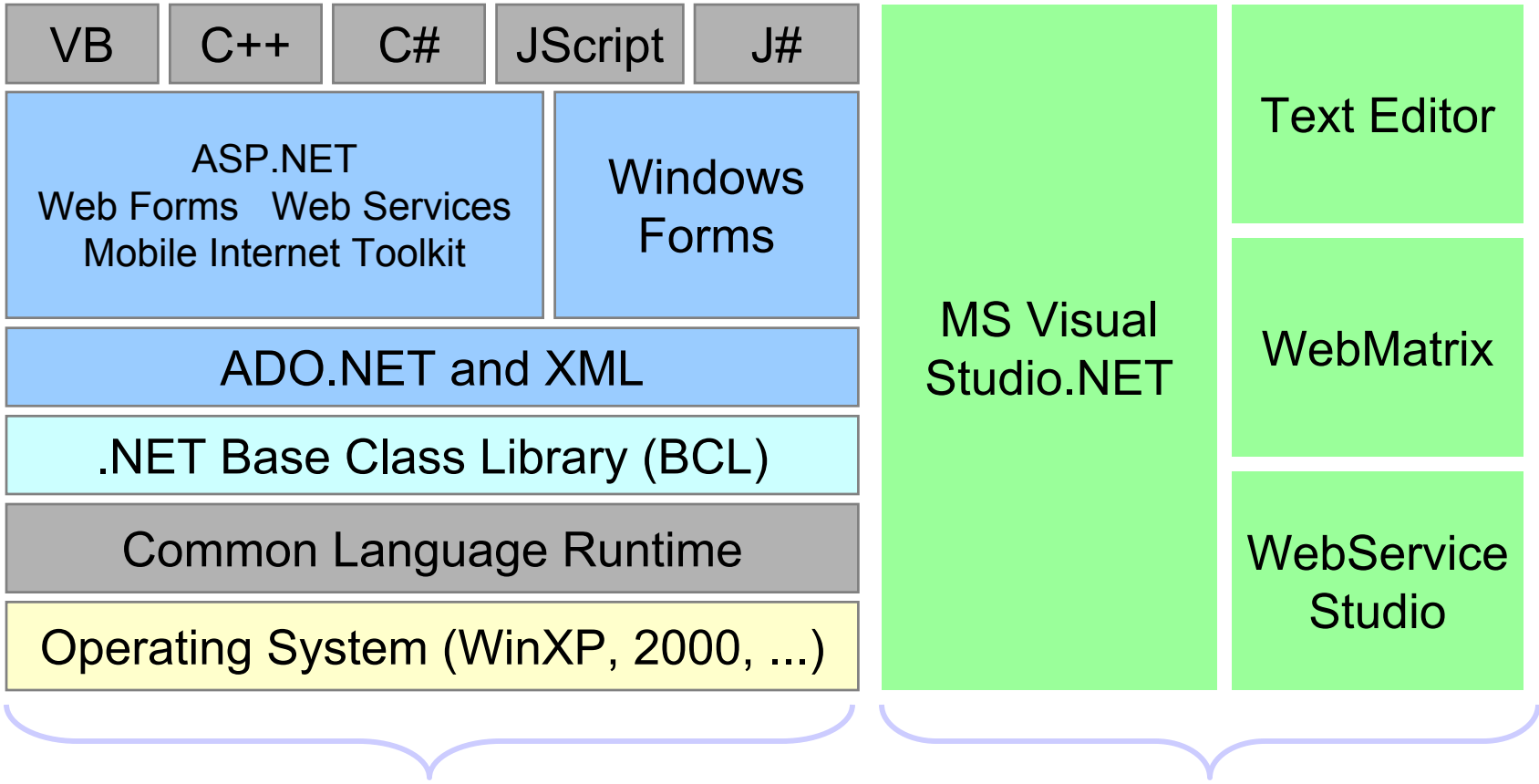
The .NET Framework Class Library

Wolfgang Beer

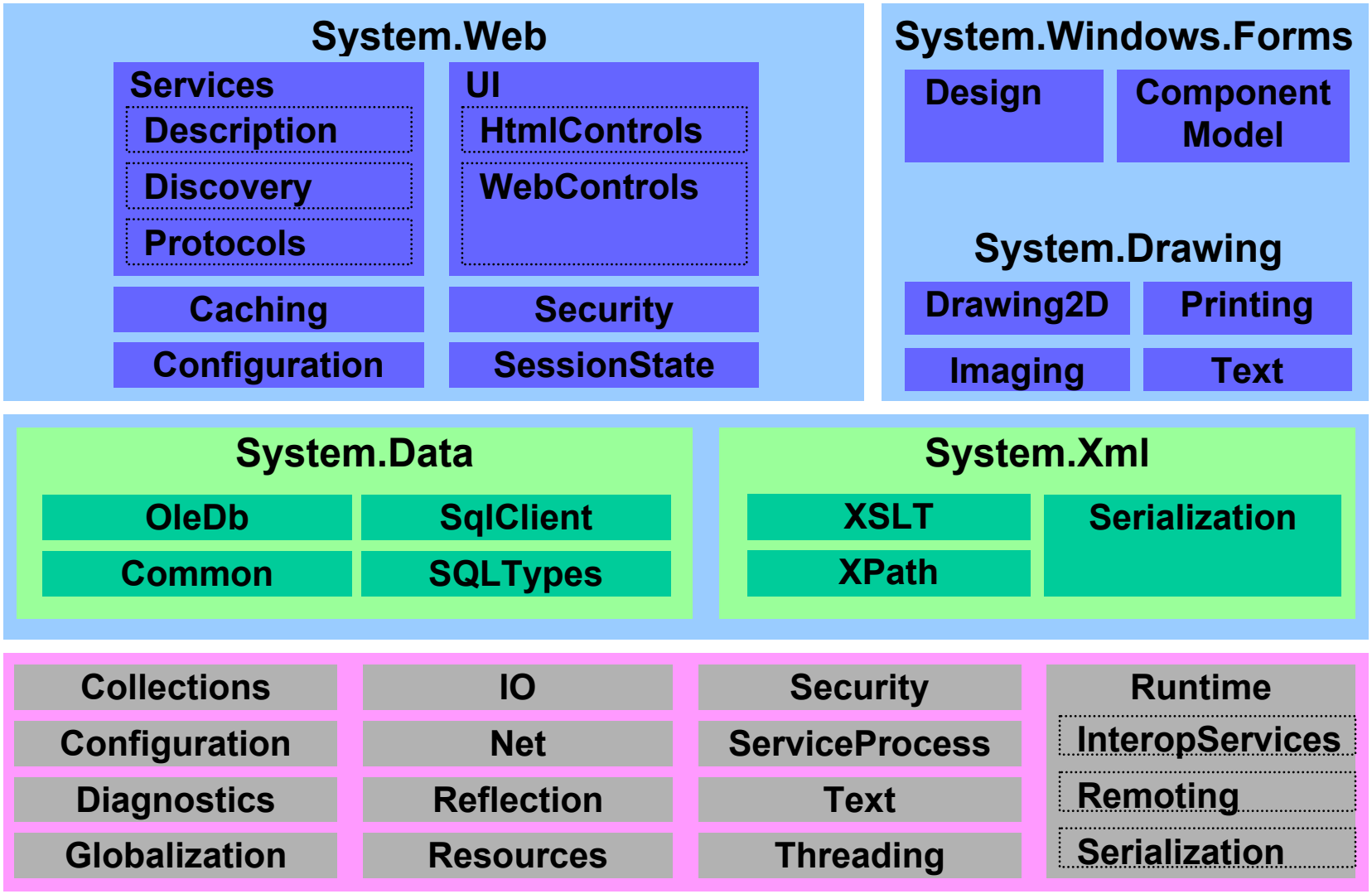
Importance of the Base Class Library

- Software developer use a personalized set of tools in terms of classes and components.
- The more complete this set of tools is, the faster is the development process of a new application.
  - No common base class library under C++! Many different string classes.
- The .NET class library adds some modern aspects:
  - XML
  - Cryptography
  - Reflection
  - Windows Forms
- The .NET class library provides a common interface between all the different .NET programming languages.

.NET Class Library



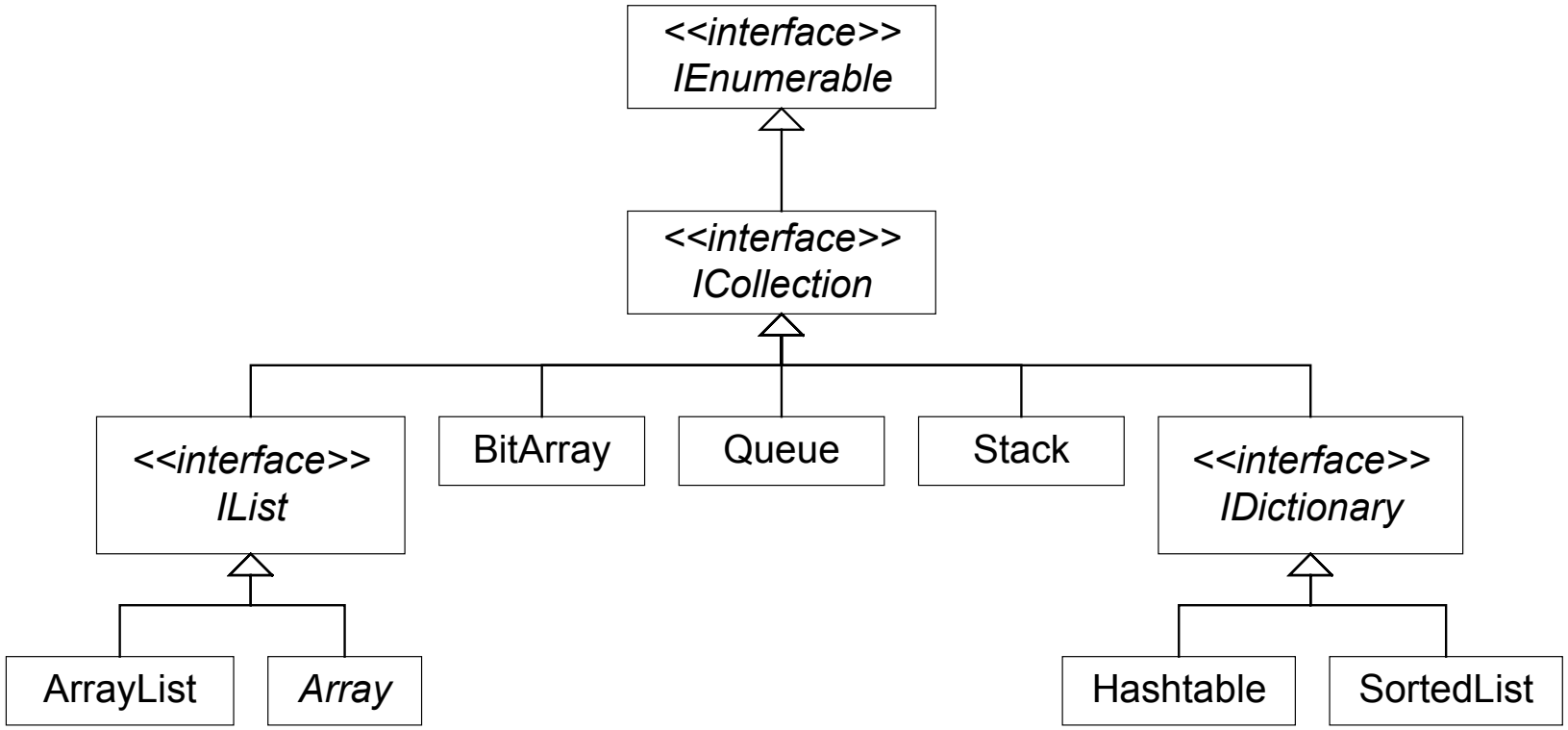
.NET Class Library



Session Overview

- *Collections*
  - Management of collections of objects
- *Strings*
  - Working with the classes String and StringBuilder
- *Reflection*
  - Working with metadata of classes, members and assemblies
- *Processing of XML-coded Data*
  - Parsing, processing, transforming and writing of XML coded data
- *Windows Forms*
  - Design of Windows applications

Collections: Class diagram



Collections

- The .NET Base Class Library supports specific kinds of element sets (e.g. Stack, Queue, Hashtable, SortedList, and many more)
- Collections that can be iterated are represented through the interface ***IEnumerable***:

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

- Only objects that implement the interface IEnumerable can be iterated using a *foreach*-statement.

Collections: IEnumerator


```
interface IEnumerator {
    object Current {get;}
    bool MoveNext();
    void Reset();
}
```

Example "Array":

```
int[] a = {1, 6, 8, 9, 15};    // object is of abstract type Array
foreach (int i in a) System.Console.WriteLine(i);
```

Run Array Example

Collections: Hashtable

- Set of key-value pairs that are wrapped by a *DictionaryEntry* object:

```
Hashtable ht = new Hashtable();
ht.add(object key, object value);

object value = ht[key];

foreach(object key in ht.Keys) { ... }
foreach(object value in ht.Values) { ... }

foreach(DictionryEntry de in ht) { ... }
```

Example „Hashtable"


```
using System.Collections;
using System;

public class HashExample {

    public static void Main(string[] args) {
        Hashtable ht = new Hashtable();
        ht.Add("key1", "value1");
        ht.Add("key2", "value2");
        ht.Add("key3", "value3");
        ht.Add("key4", "value4");
        Console.WriteLine("Value for Key:{0} ist {1}", "key1", ht["key1"]);

        foreach(DictionaryEntry de in ht)
            Console.WriteLine("Key:{0}, Wert:{1}", de.Key, de.Value);

        foreach(object key in ht.Keys)
            Console.WriteLine("Key:{0}", key);
    }
}
```

Run *Hashtable* example

Collections: Comparison of Objects

- The interfaces ***IComparable*** and ***IComparer*** are used to compare objects and therefore to sort any sets of objects.

- ***IComparable*** provides the method int CompareTo(object o) to compare the callee with the given object.

- ***IComparer*** provides a method to compare two objects:

  int Compare(object x, object o)

<0	x < o
=0	x == o
>0	x > o

Example "Sorting"

- In this example, a two dimensional Vector class is implemented, which implements the IComparable interface. The provided method **CompareTo** is used to sort a list of Vector elements.

```
public class Vector : IComparable {
    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(object obj) {
        if(obj is Vector) {
            if(this.Length < ((Vector)obj).Length) return -1;
            else if(this.Length > ((Vector)obj).Length) return 1;
            else return 0;
        }
        throw new ArgumentException();
    }
```

Example "Sorting"

- Build an array of Vector objects:

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

- Sort the array of Vector objects in ascending order:

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

Run *Vector* Example

Session Overview

- *Collections*
  - Management of collections of objects
- *Strings*
  - Working with the classes String and StringBuilder
- *Reflection*
  - Working with metadata of classes, members and assemblies
- *Processing of XML-coded Data*
  - Parsing, processing, transforming and writing of XML coded data
- *Windows Forms*
  - Design of Windows applications

Working with strings

- Classes System.String and System.Text.StringBuilder
- Objects of type String are immutable!

Example "Strings":

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

- Operation == compares the values not the references (≠Java)!

```
string s2 = "Hello, Tom";
if(s == s2)  // returns true!
```

- Compare references:

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

Class System.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);   // CultureInfo!
    public static int CompareOrdinal(string strA, string strB);  // without CultureInfo!
    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);
    ...
}
```

Class System.Text.StringBuilder

- StringBuilder is not immutable.
- StringBuilder reserves more storage than necessary for possible changes.
- Length returns the length of the char array.
- Capacity returns the size of the reserved storage.

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

String Formatting


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

equivalent to:

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

C	Currency
D	Integer
E	Numeric E+ Representation
F	Fixed-point Decimal
P	Percent Representation
X	Hexadecimal Representation
...	

Session Overview

- *Collections*
  - Management of collections of objects
- *Strings*
  - Working with the classes String and StringBuilder
- *Reflection*
  - Working with metadata of classes, members and assemblies
- *Processing of XML-coded Data*
  - Parsing, processing, transforming and writing of XML coded data
- *Windows Forms*
  - Design of Windows applications

Reflection

- Permits access to metainformation of types at runtime.
- ***System.Reflection*** enables the following tasks:
  - Gathering of metainformation about assemblies, modules and types.
  - Gathering of metainformation about the members of a type.
  - Dynamic creation of instances of a type at run time.
  - Search of methods and their dynamic invocation at run time.
  - Access to the values of properties and fields of an object.
  - Design of new Datatypes at run time with the help of the namespace: *System.Reflection.Emit*.

- ***System.Reflection.Emit*** is a powerful library for the design of .NET compilers and interpreters.

Reflection: Assemblies

- The class Assembly is used to load the metainformation of given .NET assemblies.

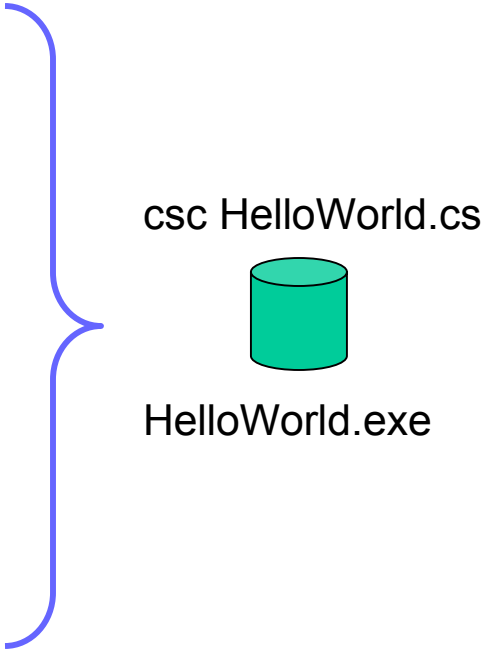
```
public class Assembly {
        public virtual string FullName {get;}
        public virtual string Location {get;}
        public virtual MethodInfo EntryPoint {get;}
        public static Assembly Load(string name);
        public Module[] GetModules();
        public virtual Type[] GetTypes();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

Reflection: Assemblies

Example "HelloWorld Reflection"

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

    public override string ToString() {
        return "Example HelloWorld";
    }
}
```



- Load the .NET assembly called: "HelloWorld.exe":

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

Reflection: Type

- Print all existing types in a given assembly:

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

- Print all existing methods in a given type:

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

Run *LoadAssembly* Example

Reflection: Dynamic Method Invocation

- Create a new instance of a given type:

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

- Search the method ToString(), which has no parameters :

```
Type hw = a.GetType("Hello.HelloWorld");     // type HelloWorld
MethodInfo mi = hw.GetMethod("ToString");
object retVal = mi.Invoke(o, null);  // method has no parameters
```

Invoke method ToString

- Search a method with a specific parameter list:

```
MethodInfo mi = hw.GetMethod(string name, Type[] types);
```

Session Overview

- *Collections*
  - Management of collections of objects
- *Strings*
  - Working with the classes String and StringBuilder
- *Reflection*
  - Working with metadata of classes, members and assemblies
- *Processing of XML-coded Data*
  - Parsing, processing, transforming and writing of XML coded data
- *Windows Forms*
  - Design of Windows applications

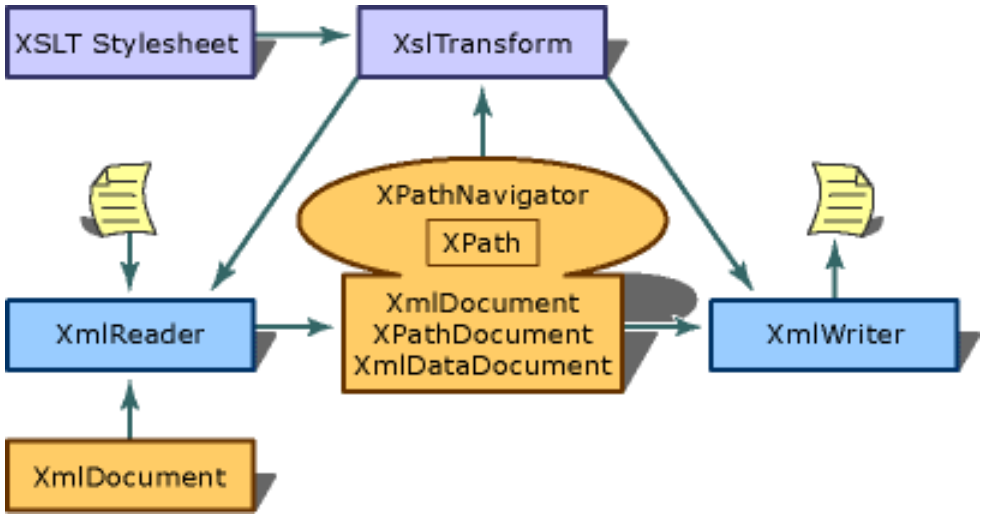
Processing XML-coded data

- The .NET Framework makes heavy use of XML standards (e.g. WSDL, UDDI, SOAP, ...).

- The base class library supports the .NET infrastructure through the implementation of these XML standards:
  - XML, XSL, XPath, ...
  - System.Xml, System.Xml.Xsl, System.Xml.XPath

- The XML processing is supported with two different methods
  - DOM (Document Object Model)
  - Serial data access, similar to SAX (Simple API for XML)

Processing XML-coded data



Abstract class XmlReader is responsible for the sequential XML parsing process.

Implementations of abstract class XMLReader are:

- XmlTextReader (fastest, non cached, forward only)
- XmlValidatingReader (validating DTD, XDR and XSD)
- XmlNodeReader (fast, non cached, access to XML data out of an XMLNode)

Abstract class XPathNavigator enables a powerful method for XPath data queries on:

filesystem, registry ;) , relational databases, any XML data sources;

Sequ. processing of XML-coded data

- Abstract class XmlReader is responsible for forward-only non caching XML data parsing.
- XmlReader is similar to SAX but uses a Pull model instead of an event-triggered Push model.
  - XmlReader demands next XML data element = *Pull*
  - Already read data elements cannot be read a second time
  - Typical SAX method uses event based notification mechanism = *Push*

bool Read(), read the next XML data element

XmlNodeType NodeType {get;}

bool HasValue {get;}

string Value {get;}

DOM processing of XML-coded data

- DOM parser maps the XML data to a memory structure.
  - The memory size limits the parseable XML data size
  - Convenient method to process the XML data structure

- XML elements are represented through objects of type *XmlNode*.
- XmlDocument is a specific XmlNode, which enables the processing of XML data.
- e.g.: Load a XML document:

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

Session Overview

- *Collections*
  - Management of collections of objects
- *Strings*
  - Working with the classes String and StringBuilder
- *Reflection*
  - Working with metadata of classes, members and assemblies
- *Processing of XML-coded Data*
  - Parsing, processing, transforming and writing of XML coded data
- *Windows Forms*
  - Design of Windows applications

Design of Windows GUI Applications

- Ultimately, the Microsoft Foundation Classes (MFC) and the Active Template Library (ATL) are replaced by a usable framework (=Windows.Forms) :)
- Classes that are used to design Windows Forms are located in the namespace: System.Windows.Forms
- Drawing functionality is located in the namespace System.Drawing
- A typical Windows form application consists of following elements:
  - Controls and user-defined UserControls
  - Forms (derived from ContainerControl) also in form of dialogs and MDIs.

Event-based GUI Applications

- Application waits for events triggered by:
  - Users (Keyboard, Mouse, ...)
  - Controls
  - Operating system (Idle, ...)
- The class Application is responsible for starting a standard application message loop.

```
public sealed class Application {
        static void Run(Form mainForm);
        static void Exit();
        static event EventHandler ApplicationExit;
        static event EventHandler Idle;
}
```

Example: GUI-HelloWorld


```
class HelloWorldForm : Form {
        Label lab;

        HelloWorldForm() {
            this.Text = "HelloWorldForm Title";
            this.Size = new Size(200,100);
            lab = new Label();
            lab.Text = "HelloWorld";
            lab.Location = new Point(20, 20);
            this.Controls.Add(lab);
        }

        public static void Main(string[] argv) {
            Application.Run(new HelloWorldForm());
        }
}
```

csc /t:winexe HelloWorldForm

Start HelloWorldForm Example

Example: Menu

- Design of a menu for a Windows Form object:

```
MainMenu m = new MainMenu();
MenuItem mi = new MenuItem("&File");
mi.MenuItems.Add(new MenuItem("&Open"));
mi.MenuItems.Add(new MenuItem("&Close"));
m.MenuItems.Add(mi);
this.Menu = m;
```

Show Menu Example

- Design of a context menu for a control object

```
ContextMenu m = new ContextMenu();
MenuItem mi = new MenuItem("&File");
mi.MenuItems.Add(new MenuItem("&Open"));
mi.MenuItems.Add(new MenuItem("&Close"));
m.MenuItems.Add(mi);
label.ContextMenu = m;
```

Show ContextMenu Example

GUI Events

- Control changes its state = **Event**
- Registration of EventHandler delegates at the event source object (Control)

```
public delegate void EventHandler( object sender, EventArgs e );
```

- Example: Register for a button click event:

```
Button b = new Button();
b.Click += new EventHandler(clickHandler);
...
private void clickHandler(object sender, EventArgs evArgs) { ... }
```

GUI Layout Design

- Three different kinds of formatters:
  - ***Anchor***: The distance between the control and a container remains the same according to a given proportion.
  - ***Docking***: Control remains directly docked on another component.
  - ***Custom***: It is possible to implement one's own LayoutManager which handles events that may appear.
    - Resize, Add or Remove Controls, Hide or Show, ...

Example: Events and Layout

GUI: Multiple Document Interface

- Creation of child forms inside a form = MDI
- Set the property *IsMdiContainer* = *true* in the parent form

Run MDIForm Example