Zusammenfassung

MsTe

Roger Blum <rblum@hsr.ch>

HS 07/08

Inhaltsverzeichnis

1 NET Framework

1.1 CLR

- Common Language Runtime
- Laufzeitumgebung für .NET-Sprachen

1.2 .NET Class Library

- Eine Library für alle .NET-Sprachen
- Basisklassen:
  - ADO.NET
  - Windows-Forms
  - Umfangreiche XML-Klassen

1.3 ASP.NET

- Web Forms
- Web Services

1.4 MS Intermediate Language

- C#, VB.NET (etc.)-Compiler erzeugen keinen „nativen" Code, sondern eine prozessor-unabhängige Zwischensprache MSIL (Microsoft Intermediate Language).
- IL-Code wird während der Ausführung immer durch einen JIT-Compiler in echten Maschinencode übersetzt
- JIT = Just in time

2 Einführung in C#

2.1 Namespaces

- Analog zu Packages in Java
- Um Klassen in andere Namespaces nutzen zu können:
  - Entweder: using MeinNamespace;
  - Oder mit Qualifikation: MeinNamespace.MeineKlasse...

2.2 Kompilieren

2.2.1 File: HelloMsgWinForm.cs

```
using System.Windows.Forms;

class HelloMessage {
      public void Speak() {
            MessageBox.Show(„Hello ...“);
      }
}
```

2.2.2 File: TestAppWinForm.cs

```
class TestApp {
      public static void Main() {
            HelloMessage h = new HelloMessage();
            h.Speak();
      }
}
```

2.2.3 Kompilieren mit Referenz auf Assemblies

```
csc /t:exe /out:TestWinForm.exe /r:System.Windows.Forms.dll TestAppWinForm.cs
HelloMsgWinForm.cs
```

2.2.4 Eigenes Assembly erstellen

```
csc /t:library /r:System.Windows.Forms.dll HelloMsgWinForm.cs

csc /t:exe /out:TestWinForm.exe /r:HelloMsgWinForm.dll TestAppWinForm.dll
```

2.3 Typen

2.3.1 Werttypen

- Einfache Typen:
  - bool
  - char
  - sbyte
  - short
  - int
  - long
  - byte
  - ushort
  - uint
  - ulong
  - float
  - double
  - decimal
- Enums
- Structs

2.3.2 Referenztypen

- Klassen
- Interfaces
- Arrays
- Delegates

2.4 Boxing

- Kopiert einen Value Type ind einen Reference Type
- Value Type wird implizit konvertiert

```
int i = 123;
object o = i;
```

2.5 Unboxing

- Inverse Operation von Boxing
- Kopiert einen Reference Type zurück in einen Value Type
- Erfordert explizite Konversion

```
object o = new Integer(32);
int j = (int) o;
```

2.6 Arrays

```
//Eindimensional
int[] a = new int[3];
int[] b = new int[]{3, 4, 5};
int[] c = {3, 4, 5};
MeineKlasse[] d = new MeineKlasse[];

//Mehrdimensionl
int[][] a = new int[2][];
a[0] = {1, 2, 3};
a[1] = {4, 5, 6};
```

```
//Mehrdimensionale Blockarrays
int[,] a = new int [2,3];
int[,] b = new {{1, 2, 3}, {4, 5, 6}};
int[,,] c = new int [2,4,2];
```

2.7 Enum

```
enum Color {red, blue, green}; //Werte: 0, 1, 2
```

2.8 Klassen

Unterschiede zu Java:

C#	Java
const	static final
base.Foo();	super.foo();
virtual	*Gibt es nicht*
override	*Gibt es nicht*

2.8.1 Konstruktoren

```
class Rectangle {
      int x, y, width, height;
      public Rectangle (int x, int y, int width, int height) {this.x=x;
this.y=y; wdith=x; height=y;
      public Rectangle (int w, int h) : this (0, 0, w, h) {}
      public Rectangle () : this (0, 0, 0, 0) {}
}
```

2.8.2 Destruktoren

```
class Test {
      ...
      ~Test() {
            ...Abschlussarbeiten...
      }
      ...
}
```

2.8.3 Properties

```
public string Name {
      set { name = value; }
      get { return name; }
}
```

2.8.4 Vererbung

```
class A {
      private int a;
      public A() {...}
      public void F() {...}
}

class B : A {
      int b;
      public B() {...}
      public void G() {...}
}
```

2.8.5 Methoden überschreiben

```
class A {
      private int a;
      public A() {...}
      public void F() {...}
      public virtual void G() {...}
```

```
}

class B : A {
      int b;
      public B() {...}
      public override void G() {
            base.G(); //ruft geerbtes G() auf
      }
}
```

2.8.6 Verdecken von Members

```
class A {
      private int a;
      public A() {...}
      public void G() {...}
}

class B : A {
      int b;
      public B() {...}
      public new void G() {...} //überdeckt G() in der Klasse A
}
```

2.8.7 Sichtbarkeit

- protected: Sichtbar in deklarierender Klasse und ihren Unterklassen.
- internal: Sichtbar in deklarierenden Assembly.
- protected internal: Sichtbar in deklarierender Klasse, ihren Unterklassen und im deklarierenden Assembly.

2.9 Felder und Konstanten

2.9.1 Konstante

- const long size = 5.23423;
- Muss einen Initialisierungswert haben
- Wert muss zur Kompilierzeit berechenbar sein

2.9.2 ReadOnly-Feld

- readonly int = 12;
- Muss bei Deklaration oder im Konstruktor initialisiert werden
- Wert muss nicht zur Kompilierzeit berechenbar sein
- Wert darf später nicht mehr verändert werden
- Wert belegt Speicherplatz

2.9.3 Statische Felder und Klassen

- static int counter;
- Konstanten dürfen nicht statisch sein!

2.9.4 Statische Konstruktoren

- Werden genau einmal ausgeführt, bei der Initialisierung des ersten Objekts

2.10 Abstrakte Klassen und Methoden

```
abstract class Stream {
      public abstract void Write(char ch) {}
      public void WriteString(string s) { foreach (char ch in s) Write(s); }
}

class File : Stream {
      public override void Write(char ch) {... write ch to disk...};
}
```

- Abstrakte Methoden haben keinen Anweisungsteil
- Abstrakte Methoden sind implizit virtual
- Wenn eine Klasse abstrakte Methoden enthält, muss sie selbst abstract deklariert werden.

2.11 Versiegelte Klassen

- sealed class A {...}
- Können nicht mehr erweitert werden
- Gleichbedeutend wie „final" in Java

2.12 Interfaces

2.12.1 Erstellen

```
public interface IList : ICollection, IEnumerable {
      int Add(object value); //Methoden
      bool Contains (object value);
      ...
      bool IsReadOnly{ get; } //Property
}
```

2.12.2 Implementieren

```
class MeineKlasse : MeineAbstrakteKlasse, MeinInterface {
      ...
}
```

2.13 Delegates

2.13.1 Beispiel

```
delegate void Notifier (string sender); //Deklaration des Delegate-Typs

Notifier notify; //Deklaration einer Delegate-Variable

void SayHello(string sender) {
      Console.WriteLine(„Hello from „ + sender);
}

notify = new Notifier(SayHello); //Zuweisung der Methode an Delegate-Variable

notify(„Max“); //Aufruf der Delegate-Variable
```

2.13.2 Multicast-Delegate

```
Notifier notify;
notify = new Notifier(SayHello);
notify += new Notifier(SayGoodBye);
```

2.13.3 Delegate als Callback (Variante II)

```
public delegate void TickEventHandler(int ticks, int interval)

class Clock {
      private TickEventHandler OnTickEvent;
      private int ticks;

      public void add_OnTickEvent(TickEventHandler handler) {
            OnTickEvent += handler;
      }

      public void remove_OnTickEvent(TickEventHandler handler) {
            OnTickEvent -= handler;
      }
```

```
	private void Tick() {
		ticks++;
		if(OnTickEvent != null) {
			OnTickEvent(ticks, interval);
		}
	}

	//Worker-Thread ruft Tick-Methode regelmässig auf
}
```

2.14 Events

- Event = Instanz eines Delegates, implementiert als eine Art Property
- Events werden in der Regel als „public" definiert. Allerdings sind nur die Methoden für das Registrieren/Deregistrieren „public". Die Methoden für das Auslösen des Events sind „private".
- Im Prinzip genügen Delegates für Event-basierte Programmierung, Events sind „syntactical sugar" und erleichtern den Umgang mit Delegates.
- Events werden im .NET Framework intensiv genutzt als eine Art Observer-Pattern.

```
//Definiere Delegate mit der gewünschten Signatur
public delegate void TickEventHandler (int ticks, int interval)

public class Clock {
	public event TicketEventHandler OnTickEvent;

	protected void Tick() {
		ticks++;
		if(OnTickEvent != null) {
			OnTickEvent(ticks, interval);
		}
	}
}

public class ClockClient {
	public ClockClient(Clock c) {
		c.OnTickEvent += new TickEventHandler(this.Update);
	}

	private void Update(int ticks, int interval) {
		Console.WriteLine(ticks);
	}
}

public class TheApp {
	public static void Main() {
		Clock c = new Clock();

		ClockClient client = new ClockClient(c);
	}
}
```

2.15 Generische Typen

```
class Buffer<Element> {
	private Element[] data;
	public Buffer(int size) {...}
	public void Put(Element x) {...}
	public Element Get() {...}
}
```

2.15.1 Generische Delegates

```
delegate book Check<T>(T value);
```

2.16 Collections

- Namespace System.Collections.Generic

2.16.1 Klassen

- List<T>, entspricht ArrayList
- Dictionary<T,U>, entspricht HashTable
- SortedDictionary<T,U>
- Stack<T>
- Queue<T>

2.16.2 Interfaces

- ICollection<T>
- IList<T>
- IDictionary<T>
- IEnumerable<T>
- IEnumerator<T>
- IComparable<T>
- IComparer<T>

2.17 Interation

- Iteration mit „foreach" möglich, sofern IEnumerable Interface implementiert ist

2.17.1 Beispiel eines Iterators

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

class MyList : IEnumerable {
      Node head = null;
      public void Add(object y) {...}
      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;
            }
      }
}
```

2.17.2 Einfacher Iterator

```
class MyClass {
      string first = „first“;
      string second = „second“;
      string third = „third“;

      public IEnumerator GetEnumerator() {
            yield return first;
            yield return second;
            yield return third;
```

```
      }
}
```

- MyClass muss IEnumerable Interface nicht implementieren
- Statt IEnumerator sollte besser IEnumerator<string> verwendet werden, da kein Cast nötig
- IEnumerator<T> ist in System.Collections.Generic

2.18 Yield

2.18.1 Variante 1

```
yield return expr;
```

- Liefert einen Wert an die foreach Schleife
- Darf nur in einer Iterator-Methode vorkommen

2.18.2 Variante 2

```
yield break;
```

- Bricht die Iteration ab
- Darf nur in einer Interator-Methode vorkommen

2.19 Vereinfachte Delegates

```
delegate void Printer(string s);

void Foo(string s) {
      Console.WriteLine(s);
}

Printer print;
print = new Printer(this.Foo); //Klassisch
print = this.Foo; //Vereinfacht
print = Foo; //Delegate-Typ wird aus dem Typ der linken Seite abgeleitet

delegate double Function(double x);

double Foo(double x) {
      return x*x;
}

Printer print = Foo; //weist Foo(string s) zu
Function square = Foo; //weist Foo(double x) zu
```

2.20 Anonyme Methoden

```
delegate void Visitor(Node p);

class List {
      Node[] data = ...;

      public void ForAll(Visitor visit) {
            for(int i=0; i<data.Length; i++) {
                  visit(data[i]);
            }
      }
}

class C {
      void Foo(){
            List list = new List();
            int sum = 0;

            list.ForAll(delegate (Node p) { Console.WriteLine(p.value); });
```

```
            list.ForAll(delegate (Node p) { sum += p.value; });
      }
}
```

2.21 Partielle Typen

Klasse wird in einzelne Dateien gesplittet.

```
//Datei Part1.cs
public partial class C {
      int x;
      public void M1() {...}
      public void M2() {...}
}
```

```
//Datei Part2.cs
public partial class C {
      string y;
      public void M3() {...}
      public void M4() {...}
      public void M5() {...}
}
```

2.22 Statische Klassen

- Dürfen nur statische Methoden und Felder besitzen

3 Windows Forms

- GUI-Framework für die Entwicklung von Desktop-Applikationen
- Voll objektorientiert
- GUI kann mit VS.NET oder Text-Editor erstellt werden
- Namensräume:
  - System.Windows.Forms: GUI-Formulare/Fenster
  - System.Drawing: Zeichenfunktionalität

3.1 Forms

- Ein Form-Objekt stellt eine Fenster in einer Applikation dar
- Properties bestimmen wie das Fenster erscheint
- Form können selbst Fenster enthalten = MDI (Multiple Document Interface)
- Forms können auch modal (z.B. als Dialogfenster) geöffnet werden

3.2 Controls

- Standard Controls wie Button, Label, Radiobutton, TextBox...
- Custom Controls wie DataGrid, MonthCalendar
- User Controls, welche der Benutzer selbst impelemntiert

3.3 HelloWorld GUI

```
class HelloWorldForm : Form {

      HelloWorldForm() {
            this.Text = „HelloWorld“;
            this.Size = new Size(200, 100);
      }

      protected override void OnPaint(PaintEventArgs e) {
            base.OnPaint(e);
            e.Graphics.DrawString(„Hello World!“, new Font(„Arial“, 35),
Brushes.Red, 10, 100);
      }

      public static void Main() {
```

```
            Application.Run(new HelloWorldForm());
      }
}
```

3.4 Button mit Events

3.4.1 Variante 1: Eigener Button erstellen

Eigener Button definieren, der von Button erbt. Anschliessend die Methode OnClick überschreiben.

3.4.2 Variante 2: Events

```
public class Form1 : Form {
      ...
      private Button button2;
      ...
      public Form1() {
            button2 = new Button(); button2.Text = „Cancel“;
            button2.Location = new Point(150, 10);
            this.Controls.Add(button2);
            button2.Click += new System.EventHandler(this.button2_Click);
      }

      private void button2_Click(object sender, System.EventArgs e) {
            MessageBox.Show(„You pressed the cancel button“);
      }
}
```

3.5 Forms

- Forms können Controls, Menüs, etc. enthalten. Diese Objekte werden in einer hierarchischen Struktur angeordnet.
- Jedes Form-Objekt definiert dazu eine Collection „Controls“ mit der Liste der „Children“.
- Positionsangaben zu einem „Child“ sind immer relativ zum „Parent“-Fenster

3.5.1 Verhalten bei Änderung der Grösse

- Docking
  - Control wird an einen Rand angedockt
  - Control ändert Grösse
- Anchoring
  - Abstand zum Anker bleibt immer gleich
  - Control ändert Grösse
- Default
  - Control ändert Grösse nicht

3.6 Menüs

```
//Menü erstellen
MainMenu menu = new MainMenu();
MenuItem item = menu.MenuItems.Add(„&File“);
Item.MenuItems.Add(new MenuItem(„E&xit“, new EventHandler(OnExit)));

//Menü dem Form hinzufügen
this.Menu = menu;

//OnExit Methode
private void OnExit{Object sender, EventArgs) {
      Close();
}
```

3.7 Databinding

- Binding zwischen Property eines Controls mit dem Property eines Objekts (Data Source) geschieht über eine Binding Objekt.
  - Binding(propertyName, dataSource, dataMember);
- Zum Property Count muss ein Event CountChanged definiert werden.
- Anmeldung erfolgt dynamisch über Reflection

3.7.1 Einfaches Databinding

```
//CounterGUI
...
      tbxCount.DataBinding.Add(new Binding(„Text“, myCounter, „Count“);
      //Text ist das Property der Textbox
      //Count ist das Property des Counters
...

//Counter
public class Counter
      private int m_Count;
      public event EventHandler CountChanged;

      public int Count {
            set {
                  m_Count = value;
                  if (CountChanged != null) {CountChanged(this, EventArgs.Empty}
            }
            get {
                  return m_Count;
            }
      }
}
```

3.7.2 Databinding an eine Collection

- Als DataSource eignen sich:
  - IBindingList (DataView)
  - IList (Collections)
  - IListSource (DataSet, DataTable);
- Problem: Position auf aktuelles Objekt muss gespeichert sein, Binding Manager nötig.

```
DataGrid1.DataSource = productsDataSet;
DataGrid1.DataMember = „Products“;
```

3.8 Scrollbars

- Bei Controls die Scrollbars unterstützen können diese mit myControl.AutoScroll = true aktiviert werden.
- Je nach dem können Horizantale- bzw. Vertikale-Scrollbars mit HScroll=fals bzw. VScroll=false deaktiviert werden.

3.9 Koordinaten

- World-Koordinatensystem für das Modell/Dokument („undendlich“ gross)
- Page-Koordinatensystem für die Device-unabhängige Ausgabe
- Device-Koordinatensystem für die Device-abhängige Ausgabe
- Page-Transformation: World -> Page
- Device-Transformation: Page -> Device

4 Windows Forms und Multi-Threading

- Methoden von Controls dürfen nur vom Thread aufgerufen werden, der das Controls erzeugt hat (=Owner)
- Direkte Methode-Aufrufe von einem anderen Thread sind nicht erlaubt und müssen via Message-Queue zur Ausführung an den Owner-Thread weitergeleitet werden.

- Win-Forms unterstützt dies wie folgt:
  - Jedes Control hat eine Methode Invoke für das „verzögerte" Ausführen einer Methode aus einem anderen Thread.
  - Der auszuführende Code muss in ein Delegate-Objekt verpackt werden.

5 Remoting

5.1 AppDomains

- Die CLR abstrahiert OS-Prozesse und arbeitet mit „virtuellen Prozessen":
  - Isolierter Ausführungsraum für Anwendungen
  - Unabhängigkeit vom OS-Konzept
  - Diese virtuellen Prozesse werde **AppDomains** genannt
  - AppDomains dienen als „Ausführungscontainer" für Assemblies
- Eine AppDomain existiert immer genau in einem Prozess
  - Jeder Prozess, der Managed Code ausführt, besitzt mindestens eine AppDomain
  - Ein Prozess kann mehrere AppDomains beinhalten
  - Ein Kontext-Switch zwischen Appdomains ist viel schneller als zwischen Prozessen
- Kommunikation zwischen verschiedenen AppDomains erforder Marshalling

5.1.1 Erzeugen einer AppDomain und ausführen eines Prozesses

```
//fibapp.exe
class Fib {
      static int Main(string[] args) {
            return Int32.Parse(args[0]) + Int32.Parse(args[1]);
      }
}

//main.exe
public class App {
            public static void Main() {
            AppDomain ad = AppDomain.CreateDomain(„FibAppDomain“, null, null);

            int x=1; int y=1; int next;
            for(int n=0; n<100; n++) {
                  string[] args = {x.ToString(), y.ToString()};
                  next = ad.ExecuteAssembly(„fibapp.exe“, null, args);
                  Console.WriteLine(„{0} „, next);
                  x = y; y=next;
            }
      }
}
```

5.2 Channels

- Kommunikation zwischen Client und Server
- Client Channel sendet Messages zum Server Channel
- TCP Channel, HTTP Channel
- TcpServerChannel ch = new TcpServerChannel(8086);

```
HttpServerChannel ch1 = new HttpServerChannel(8087);
```

- Port wird dem Konstruktor mitgegeben (nur Serverseitig nötig)
- TCP und HTTP Channel können gleichzeitig existieren
- Channels müssen registriert werden:

```
ChannelServices.RegisterChannel(ch);
ChannelServices.RegisterChannel(ch1);
```

5.3 Formatters

- In welches Format soll das Objekt de/serialisiert werden
- Framework implementiert BinaryFormatter, SOAPFormatter
- Defaults: Binary für TCP, Soap für HTTP
- Beispiel: Registrieren eines Formatters

```
SoapServerForammterSinkProvider sp = new SoapServerFormatterSinkProvider();
TcpServerChannel ch = new TcpServerChannel(„name“, 8086, sp);
//TCP Channel arbeitet jetzt mit einem Soap Formatter
```

5.4 Remote Objects

- Server-Activated Objects (SAO):
  - Werden vom Server erzeugt und publiziert
  - Single-Call (Service-basiert)
  - Singleton
- Client-Activated Objects (CAO):
  - Werden vom Client erzeugt (wie normale .NET-Objekte)
  - Distributed garbage collection nötig

5.4.1 SAO Singleton

- Nur eine Instanz dieser Klasse
- Bei jeder Aktivierung erhält der Client ein Proxy dieser Instanz
- „Sharing“ von globalen Ressource
- Konkurrierende Zugriffe auf Ressourcen erfordern Synchronisation
  - Bei hoher Clientzahl schlechte Skalierung

5.4.2 SAO SingleCall

- Client erhält Proxy, ohne dass dabei eine Instanz erzeugt wird
- Bei jedem Methodenaufruf wird eine neue Instanz erzeugt
- Kein Status (Service basiert)
  - Objekt besteht nur aus Methoden, Zustand muss manuell verwaltet werden
  - Hohe Skalierbarkeit

5.4.3 CAO

- Jeder Client bekommt sein eigenes Objekt
  - Verbindungsorientiertes Modell
  - Lebensdauer der Objekte muss verwaltet werden (Leases).
- Server meldet Objekt im System an

5.5 Beispiel Implementation SAO

```
//common.dll
public interface IMyRemoteCounter {
      void setCount(int newval);
      int getCount;
      int inc();
      int dec();

      int Count
      {
            get;
            set;
      }
}

//MyRemoteCounter.cs
class MyRemoteCounter : MarshalByRefObject, IMyRemoteCounter {
      int count;
      public MyRemoteCounter () {
            Console.WriteLine(„New Counter created“);
```

```
      }

      public int inc() {
            count++;
            retunr count;
      }
      ...
}

//Server.cs
class Server {
      static void Main(string[] args) {
            TcpChannel chn = new TcpChannel(1235);
            ChannelServices.RegisterChannel(chn);

            RemotingConfiguration.RegisterWellKnownServiceType(
                        typeof(MyRemoteCounter),
                        „MyRemoteCounter“, //Application name
                        WellKnownObjectMode.Singleton);

            Console.WriteLine(„Hit to exit“);
            Console.ReadLine();
      }
}

//Client1.cs mit Activator.GetObject()
class Client1 {
      static void Main(string[] args) {
            Channel.Services.RegisterChannel(new TcpClientChannel());
            string remoteObjURI = „tcp://localhost:1235/MyRemoteCounter“;

            IMyRemoteCounter obj =
(IMyRemoteCounter)Activator.GetObject(typeof(IMyRemoteCounter), remoteObjURI);

            obj.Count = 20;
            obj.inc();
      }
}

//Client2.cs mit Remoting.Services.Connect()
class Client2 {
      static void Main(string[] args) {
            Channel.Services.RegisterChannel(new TcpClientChannel());
            string remoteObjURI = „tcp://localhost:1235/MyRemoteCounter“;

            IMyRemoteCounter obj =
(IMyRemoteCounter)RemotingServices.Connect(typeof(IMyRemoteCounter),
remoteObjURI);

            obj.Count = 20;
            obj.inc();
      }
}
```

5.6 Beispiel Implementation CAO

```
//Server.cs
class Server {
      static void Main(string[] args) {
            TcpChannel chn = new TcpChannel(1235);
            ChannelServices.RegisterChannel(chn);

            RemotingConfiguration.ApplicationName = „MyCounterServer“;
            RemotingConfiguration.RegisterActivatedServiceType(
                        typeof(MyRemoteCounter));
```

```
            Console.WriteLine(„Hit to exit“);
            Console.ReadLine();
      }
}

//Client1.cs mit Activator.CreateInstance()
class Client1 {
      static void Main(string[] args) {
            Channel.Services.RegisterChannel(new TcpClientChannel());

            object[] activationAtt =
            (new UrlAttribute(„tcp://localhost:1235/MyRemoteCounter“);
            object[] ctorArgs = {40};

            IMyRemoteCounter obj =
(IMyRemoteCounter)Activator.CreateInstance(typeof(Counter.MyRemoteCounter),
ctorArgs1, activationAtt);

            obj.Count = 20;
            obj.inc();
      }
}

//Client2.cs mit New()
class Client1 {
      static void Main(string[] args) {
            Channel.Services.RegisterChannel(new TcpClientChannel());

            string remoteObjURI = „tcp://localhost:1235/MyRemoteCounter“;

            RemotingConfiguration.RegisterActivatedClientType(
                  typeof(MyRemoteCounter), remoteObjURI);
            IMyRemoteCounter obj = new MyRemoteCounter(40);
            IMyRemoteCounter obj1 = new MyRemoteCounter(20);

            obj.Count = 20;
            obj1.Count += 2;
      }
}
```

6 Web-Services

- Middleware für verteilte Anwendungen
- Für Remote-Procedure-Calls und Datenaustausch
- Offener Standard auf der Basis von XML
- Für lose gekoppelte Softwaredienste
- Unabhängig von Programmiersprache, Laufzeitumgebung und Betriebssystem
- Verwendung bestehender Internet-Protokolle und Server-Architekturen

6.1 Definition Web-Service (nach W3C)

- Durch eine URI identifizierbare Softwareanwendung
- Mit Schnittstellendefinition in XML
- Mit Interaktion auf Basis XML-basierten Nachrichten
- Nachrichtenaustausch über Internet-Protokolle

6.2 Beispiel: TimeService

```
//TimeService.asmx
<%@ WebService Language=“C#“ Class=“TimeService“ %>

using System;
using System.Web.Services;
```

```
public class TimeService : WebService {
      [WebMethod(Description="Returns the current time")]
      public string GetTime(bool shortForm) {
            if(shortForm) return DateTime.Now.ToShortTimeString();
            else return DateTime.Now.ToLongTimeString();
      }
}

//Aufruf in Browser
http://localhost/time/TimeService.asmx/GetTime?shortForm=true
```

7 ADO.NET

- ADO.NET ist .NET-Technologie für den Zugriff auf strukturierte Daten
- Einheitliche objektorientierte Schnittstelle für unterschiedliche Datenquellen
  - Relationale Datenbanken
  - XML-Daten
  - Andere Datenquellen
- Konzipiert für verteilte Anwendungen und Web-Anwendungen
- Bietet zwei unterschiedliche Modelle für Datenzugriff
  - Verbindungsorientiert:
    - Verbindung zur Datenquelle bleibt erhalten
    - Typische Verwendung: Kurze Zugriffe, wenige parallele Zugriffe, immer aktuelle Daten
  - Verbindungslos:
    - Keine permanente Verbindung zur Datenquelle
    - Daten im Hauptspeicher zwischengespeichert
    - Änderungen im Hauptspeicher ≠ Änderung in Datenquelle
    - Typische Verwendung: Viele parallele, lange, lesende Zugriffe

7.1 Komponenten

- DbConnection: Repräsentiert Verbindung zu einer Datenquelle
- DbCommand: Repräsentiert SQL-Kommando
- DbTransaction: Repräsentiert Transaktion, Commands können innerhalb Transaktion ausgeführt werden
- DataReader: Ergebnis einer Datenbankabfrage, erlaubt sequentielles Lesen der Zeilen

7.2 DataSet

- Ein DataSet ist eine in-memory Datenbank, keine Connection zur DB nötig
- Wir über DataAdapter, XML-File oder via Code mit Daten gefüllt

7.2.1 Schema definieren

```
//DataSet erstellen
DataSet ds = new DataSet(„PersonContacs");

//Tabelle erstellen
DataTable personTable = new DataTable(„Person");

//Spalte ID erstllen und Eigenschaften festlegen
DataColumn col = new DataColumn();
col.DataType = typeof(System.Int64);
col.ColumnName = „ID";
col.ReadOnly = true;
col.Unique = true;
col.AutoIncrement = true;
col.AutoIncrementSeed = -1;
col.AutoIncrementStep = -1;

//Spalte zur Tabelle hinzufügen
personTable.Columns.Add(col);
personTable.PrimaryKey = new DataColumn[]{col};
```

```
//Spalte Firstname definieren und anfügen
col = new DataColumn();
col.DataType = typeof(string);
col.ColumnName = „Firstname“;
personTable.Columns.Add(col);
//Spalte Lastname definieren und anfügen
col = new DataColumn();
col.DataType = typeof(string);
col.ColumnName = „Lastname“;
personTable.Columns.Add(col);

//Tabelle zu DataSet hinzufügen
ds.Tables.Add(personTable);
```

7.2.2 Relation zwischen zwei Tabellen

```
DataColumn parentCol = ds.Tables[“Person“].Columns[“ID“];
DataColumn childCol = ds.Tables[“Contact“].Columns[“PersonID“];

DataRelation rel = new DataRelation(„PersonHasContacts“, parentCol, childCol);
ds.Relations.Add(rel);
```

7.2.3 Datensatz einfügen

```
//Leere Zeile erstellen
DataRow personRow = personTable.NewRow();
personRow[1] = „Wolfgang“; //Zugriff über Index
personRow[“Lastname“] = „Beer“; //Zugriff über Feldname

//Zeile hinzufügen
personTable.Rows.Add(personRow);

//Änderungen im DataSet übernehmen
ds.AcceptChanges();
```

7.2.4 Auf Daten zugreiffen

```
foreach(DataRow person in ds.Tables[“Person“].Rows) {
      Console.WriteLine(„Lastname: {0}“, person[“Lastname“]);
}
```

7.3 nHibernate

- Auf C# portiertes Persistenzframework, ursprünglich für Java entwickelt.
- Zustand eines Objekts kann ich einer relationalen Datenbank gespeichert werden
- Die Abfrage erfolgt über eine SQL-ähnliche Sprache namens HQL
- Hibernate ist mit fast allen aktuellen relationalen Datenbanken kompatibel

7.3.1 OR Mapping

- Wie werden Klassen auf Tabellen abgebildet
  - Klasse auf Tabelle
  - Vererbungshierarchie auf abhängigen Tabellen
  - 1:n Assoziationen über Fremdschlüssel
  - N:m Assoziationen über Beziehungstabellen
- Definition der Abbildung mit
  - Attributierung der Klassen
  - XML-Mapping-File

7.3.2 Mapping von Klassen

- Mapping von Klassen auf Tabellen (oder Stored Procedures
- Hibernate benötigt parameterlosen Konstruktor
- Angaben fürs Mapping
  - Name der DB-Tabelle

- Zu persistierende (=speichernde) Properties oder Felder
- Das Property, welches dem Primärschlüssel entspricht
- Wie der Primärschlüssel generiert werden soll

- Abbildung durch Attributierung oder mit XML

7.3.3 Mapping mit Attributierung

```
//Mapping durch Attributierung
[Class]
public class Artikel {
      private int m_ID;
      [Id(0, Name = "m_ID", Column = "Id", Access = "field", UnsavedValue= "0")]
      [Generator(1, Class = "identity")]
      public int Id { get { return m_ID; }  }

      [Version(Column = "Version", Access = "field")]
      private int m_Version;

      private string m_Bezeichnung;

      [Property]
      public string Bezeichnung {
            get { return m_Bezeichnung; }
            set { m_Bezeichnung= value; }
      }
. . .
}

//Passende Tabelle zur Klasse
CREATE TABLE  Artikel (
      Id int IDENTITY(1,1) PRIMARY KEY,
      Bezeichnung nvarchar(50) NOT NULL,
      Version INT NOT NULL
)
```

```
//Artikel mit Vererbung
[JoinedSubclass(ExtendsType = typeof(Artikel))]
public class FarbigerArtikel : Artikel {
      [Key(Column = "id_Artikel")]
      private Farbe m_Farbe;
      [Property]
      public Farbe Farbe{
            get{ return m_Farbe; }
            set{ m_Farbe = value; }
      }
}

public enum Farbe {
      Rot,
      Grün,
      Blau
}
```

7.3.4 1:n bidirektionale Beziehung

```
[Class]
public class Artikel {
. . .
      [Bag(0, Inverse = true, Access = "field", Cascade = CascadeStyle.All,
Lazy=true)]
      [Key(1, Column = "id_artikel")]
      [OneToMany(2, ClassType= typeof (Lagerort))]
      private IList m_Lagerorte= new ArrayList();

      public void AddLagerort(Lagerort ort) {
```

```
            ort.Artikel = this;
            m_Lagerorte.Add(ort);
      }

      public void RemoveLagerort (Lagerort ort)
      {
            m_Lagerorte.Remove(this);
            ort.Artikel = null;
      }
}

[Class]
public class Lagerort {
      . . .
      private Artikel m_Artikel;
      [ManyToOne(Column ="id_artikel", NotNull = false)]
      public Artikel Artikel {
            get  { return m_Artikel; }
            set  { m_Artikel = value; }
      }
}
```

7.3.5 Mapping mit XML

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0" assembly="Lagerverwaltung"
namespace="Lagerverwaltung.Entities">
<class name="Artikel“  table=“ARTIKEL">
      <id name="m_ID" column="Id" access="field" unsaved-value="0">
            <generator class="identity" />
      </id>
      <version name="m_Version" column="Version" access="field"/>
      <property name="Bezeichnung" />
      <bag name="m_Lagerorte" access="field" inverse="true" cascade="all"
lazy="true">
            <key column="id_artikel" />
            <one-to-many class="Lagerort" />
      </bag>
</class>

<joined-subclass name="FarbigerArtikel" extends="Artikel">
      <key column="id_Artikel" />
      <property name="Farbe" />
</joined-subclass>
```

7.3.6 Session

- Wird via Session Factory erzeugt
- Kapselt DB-Sessions und Transaktionen
- Führt Cache mit persistenten Objekten
- Stellt Object-Identity innerhalb der Session sicher
- Unterstützt Operationen zum Laden und Speichern von Objekten

```
<?xml version='1.0' encoding='utf-8'?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">

<!--an ISessionFactoryinstance -->
<session-factory>
      <!--properties -->
      <property name="connection.provider">
            NHibernate.Connection.DriverConnectionProvider
      </property>
      <property name="connection.driver_class">
            NHibernate.Driver.SqlClientDriver
      </property>
```

```
      <property name="connection.connection_string">
            Server=localhost;initial catalog=nhibernate;User Id=;Password=pass
      </property>
      <property name="show_sql">false</property>
      <property name="dialect">NHibernate.Dialect.MsSql2000Dialect</property>
      <property name="use_outer_join">true</property>
      <!--mapping files -->
      <mapping resource="NHibernate.Auction.Item.hbm.xml"
      assembly="NHibernate.Auction" />
      <mapping resource="NHibernate.Auction.Bid.hbm.xml"
      assembly="NHibernate.Auction" />
</session-factory>

</hibernate-configuration>
```

7.3.7 Erzeugen von Objekten

```
using (ISession session = NHibernateSessionFactory.CreateSession()) {
      ITransaction tx= session.BeginTransaction();
      Gebäude g = new Gebäude(); //transient
      g.Bezeichnung = "HSR";
      session.Save(g); //g wird persistent

      Lagerort ort = new Lagerort(); //transient
      ort.Gestell = "Weingestell"; ort.AblageNr = 1; ort.Gestell = "W";
      g.AddLagerort(ort);
      session.Save(ort); //ort wird persistent
      Artikel artikel = new Artikel();
      artikel.Bezeichnung = "Rotwein";
      artikel.AddLagerort(ort);
      session.Save(artikel); //artikel wird persistent
      tx.Commit();
} //calls session.Close
```

7.3.8 Laden von Objekten

```
using (ISession session = NHibernateSessionFactory.CreateSession()) {
      // Kriterien für Datenbank-Abfrage: Gebäude mit Bezeichnung “HSR”
      ICriteria crit = session.CreateCriteria(typeof (Gebäude));
      crit.Add(Expression.Eq("Bezeichnung", "HSR"));
      IList result = crit.List(); //query ausfuehren
      if (result.Count == 1) {
            Gebäude hsr = (Gebäude) result[0]; //”root”-Element
            foreach (Lagerort l in hsr.Lagerorte) {
                  //Lagerorte werden über Beziehung automatisch geladen
                  Console.WriteLine(l.GangNr);
                  if (l.Artikel != null)
                  //Artikel werden über Beziehung automatisch geladen
                        Console.WriteLine(l.Artikel.Bezeichnung);
            }
      }
}
```

7.3.9 Updaten

```
using (ISession session = NHibernateSessionFactory.CreateSession())
{
      Gebäude g = (Gebäude) session.Load(typeof (Gebäude), idGebaeude);
      if (g!= null) {
            g.Bezeichnung = "HSR Gebäude 6";
            session.Flush();
      }
}
```

7.3.10 Löschen

```
using (ISession session = NHibernateSessionFactory.CreateSession())
{
      Gebäude g = (Gebäude) session.Load(typeof (Gebäude), idGebaeude);
      if (g!= null) session.Delete(g);
      session.Flush();
}
```