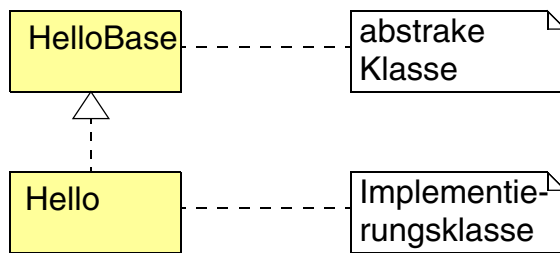


## 5.3 Gemeinsam genutzte Basisklasse

---

- Abstrakte Basisklasse für Implementierungsklasse



- ◆ Shared assembly für Basisklasse, private Assembly für Implementierungsklasse (nur Server)

- ★ Vorteil

- ◆ Trennung möglich
- ◆ Objektreferenzen weitergebbar



## 5.3 Gemeinsam genutzte Basisklasse (2)

---

- ▲ Nachteil

- ◆ lediglich mit Activator nutzbar
- ◆ keine direkte Objekterzeugung möglich
  - Compiler unterbindet Erzeugung von Objekten einer abstrakten Klasse

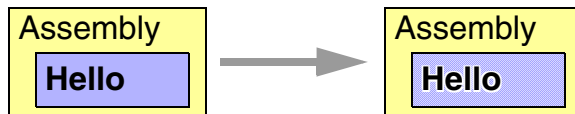


## 5.4 Generierte Metadaten

---

### ■ Werkzeugunterstützung

- ◆ Programm **SoapSuds** erzeugt zweite Assembly für reine Metadaten



### ■ Wrapped-proxy-Ansatz

- ◆ **SoapSuds** integriert Kommunikationsadresse in Metadaten
- ◆ Client-Zugriff

```
m= new Hello();
```

- erzeugt Objekt im Server

### ■ Nonwrapped-proxy-Ansatz

- ◆ Client-Zugriff über **Activator** oder über Typ-Registrierung



## 6 Lebenszeitverwaltung

---

### ■ Verwaltung über Leases

- ◆ Abräumen des Objekts sobald Lease abgelaufen (Garbage Collection)
- ◆ Default-Einstellungen
  - Lease gültig bis 5min nach Objekterzeugung
  - Erneuerung der Lease bis zu 2min bei jedem Aufruf

### ■ Sponsoren

- ◆ registrierte Ansprechpartner bei Ablauf einer Lease
- ◆ innerhalb von 2min (Default) muss Sponsor Verlängerung der Lease beantragen



## 6.1 Konfiguration der Einstellungen

---

- Anwendungsabhängige Änderung der Einstellungen
  - ◆ programmatisch: Zuweisung von Zeitspannen an gewisse Systemvariablen
    - z.B. `LifetimeServices.LeaseTime = ...`
  - ◆ deklarativ: XML-Elemente in Konfigurationsdatei
    - z.B. `<lifetime leaseTimeout="..." />`



## 6.1 Konfiguration der Einstellungen (2)

---

- Klassenabhängige Änderung der Einstellungen
  - ◆ programmatisch: Methode `InitializeLifetimeService()` in Basisklasse `MarshalByRefObject`
    - erzeugt Objekt mit Interface `ILease`
      - enthält Einstellungsparameter
    - Rückgabe eines `null`-Zeigers möglich
      - keine Garbage-Collection
  - ◆ deklarativ: XML-Elemente in Konfigurationsdatei
    - Setzen der Parameter muss in `InitializeLifetimeService()` erfolgen
    - Parameter können aus Konfigurationsdatei stammen
      - anwendungsabhängige Konfigurationsparameter (vgl. Java Properties)
      - z.B. `<appSettings>`
        - `<add key="Hello_LeaseTime" value="..."> ...`



## 6.2 Sponsoren

---

- Automatische Befragung des Sponsors über Lease-Verlängerung
  - ◆ gibt neue Lease-Lebenszeit zurück
    - Lebenszeit gleich Null: Objekt wird aufgeräumt
  
- Client-seitige Sponsoren
  - ◆ entspricht periodischem Prüfen, ob Server-Objekt noch vorhanden
    - entspricht früherem DCOM-Ansatz
  - ◆ Achtung: Client wird kurzzeitig zum Server, Server zum Client
    - Client muss Channel für Anfragen besitzen
      - z.B. HTTP-Channel: Portnummer angeben oder `port="0"`
  - ◆ Achtung: Sponsoren sind Remote-Objects
    - benötigen eigenes Lease-Management (evtl. durch sich selbst)



## 6.2 Sponsoren (2)

---

- Server-seitige Sponsoren
  - ◆ auf gleichem oder anderem Server-Knoten
  - ◆ Bindung der Lebenszeit an anwendungsabhängige Bedingungen
    - z.B. Verweigerung der Lease-Erneuerung, falls essenzielle Anwendungskomponenten abgestürzt oder unerreichbar



## 6.3 Versionierung

---

- Strong-name für Assemblies
  - ◆ Name der Assembly
  - ◆ *AssemblyCulture*: Beschreibung von kulturbedingten Konventionen
  - ◆ *AssemblyVersion*: Versionskennung, z.B. **1.0.0.1**
  - ◆ Fingerprint der Assembly
    - digitale Signatur über Assembly-Inhalt
  
- Global-Assembly-Cache (GAC)
  - ◆ Systemverzeichnis mit installierten Assemblies
    - erzeugte Assembly muss in den GAC installiert werden
  - ◆ Strong-name für jede Assembly
    - Installation von Assemblies unterschiedlicher Version möglich



## 6.3 Versionierung (2)

---

- Server-activated objects
  - ◆ Server: Registrierung der Objektklasse mit Strong-name der Assembly
    - z.B. `<wellknown type="Hello, Server, Version=1.0.0.1, Culture=neutral, PublicKeyToken=84d24aff2014ab" .../>`
  - ◆ Client: Verweis auf Strong-name der Assembly oder generierten Dateinamen der versionsspezifischen Assembly
  
- Client-activated objects
  - ◆ **SoapSuds** integriert Strong-name in erzeugte Client-Assembly
    - spezielle Attribute
    - Codierung des Strong-name in XML-Namespace URIs
      - Namespace für SOAP-Nachricht



## 6.3 Versionierung (3)

---

- **Serializable objects**
  - ◆ ähnlich wie bei Client-activated objects
  - ◆ Custom-serialization zur Interopabilität zwischen alten und neuen Versionen
    - fehlende Daten müssen ergänzt werden
    - überzählige Daten müssen verworfen werden



## 7 Sicherheit

---

### 7.1 Authentisierung

---

- **Basic authentication**
  - ◆ Basismechanismus eines Webservers
    - Benutzername und Passwort im Klartext in Kopfzeilen einer HTTP-Nachricht (SOAP-Nachricht)
  - ◆ Webserver überprüft Authentisierung
    - Abweisen, falls Benutzername und Passwort ungültig
    - sonst interne Weiterleitung der Nachricht mit Benutzername



## 7.1 Authentisierung (2)

---

### ■ Basic authentication in .Net

- ◆ Server-Seite: Einrichtung eines eingetragenen Benutzers
  - z.B. in den IIS (MS Webserver)
    - HTTP-Channels werden vom IIS verwaltet
- ◆ Client-Seite: Senden des Passworts
  - Eintragen des Benutzernamen und Passworts in Channel-Properties
    - Channel-Properties: Liste von Einstellungen pro Objektreferenz/Proxy
  - Beispiel:

```
Hello hello= new Hello();
IDictionary props=
    ChannelServices.GetChannelSinkProperties( hello );
props["username"]= "alice";
props["password"]= "qo(dfDw98";
res= hello.sayHello();
```



## 7.1 Authentisierung (3)

---

### ■ Windows-Authentisierung

- ◆ Basismechanismus von Windows zur Identifikation von Windows-Nutzern
  - Challenge-Response-Protokoll zur Verifikation des Windows-Passworts

### ■ Windows-Authentisierung in .Net

- ◆ Server-Seite: Konfiguration des Web-Servers
- ◆ Client-Seite: keine Änderung
  - Client erkennt Windows-Authentisierung
  - oder Client ist in Domäne authentisiert und gibt seine Credentials weiter
    - keine Benutzer/Passwort-Angabe im Code
    - stattdessen Konfiguration des Channel:

```
<channels>
  <channel ref="http" useDefaultCredentials="true"/>
</channels>
```



## 7.2 Autorisierung

---

### ■ Ermittlung des aufrufenden Benutzers (Principal)

- ◆ Kontext am aufrufenden Thread

```
IPrincipal prin= System.Threading.Thread.CurrentPrincipal;
```

### ■ Gruppenzuordnung prüfen

- ◆ Prüfung der Benutzerzuordnung zu einer Windows Gruppe

```
principal.IsInRole( "mayestril\admins" );
```

### ■ Programmatische Autorisierung

- ◆ Programmierung der Zugangsüberprüfung in Methoden



## 7.3 Verschlüsselung der Kommunikation

---

### ■ Einsatz von SSL (Secure Socket Layer)

- ◆ Server-Seite: Erzeugung eines Server-Zertifikats
  - Server-Zertifikat: privater Schlüssel
    - oft signiert von einer Zertifizierungsstelle (Certificate Authority)
- ◆ Client-Seite: Änderung der URL
  - Voranstellen von **“https”** statt **“http”**

### ■ Vorteile

- ◆ Überprüfung, dass (Web-)Server authentisch ist
- ◆ verschlüsselte Datenverbindung zwischen Client und Server (z.B. für SOAP-Nachrichten)
- ◆ Basic-authentication wird geschützt



## 8 Literatur

---

### ■ .Net

- ◆ I. Rammer: Advanced .Net Remoting. Apress, 2002.
- ◆ Articles on Microsoft Developer Network:  
.NET Framework “Deploying & Distribution”.  
<<http://msdn.microsoft.com/netframework/using/deploying/default.aspx>>

