Middleware - Übung

Tobias Distler, Michael Gernoth, Rüdiger Kapitza

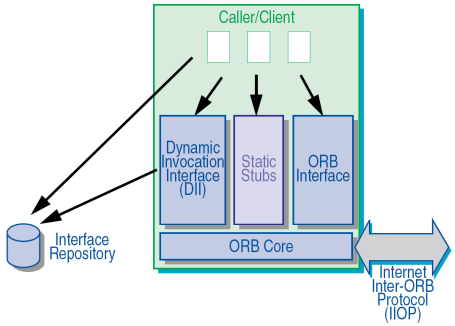
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

Wintersemester 2009/2010

Überblick

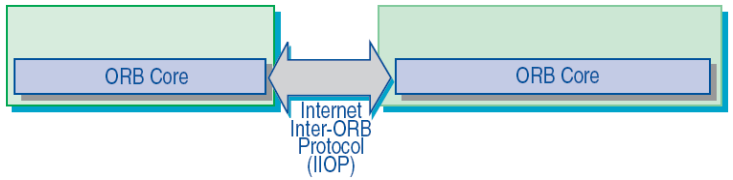
Client-Seite

- Client
  - Aufrufer der Operation am CORBA-Objekt
  - Muss selbst kein CORBA-Objekt sein
- Statischer Stub
  - Automatisch erzeugt aus IDL-Schnittstelle
  - Marshalling der Aufrufparameter
  - Unmarshalling der Rückgabewerte/Exceptions
- ORB-Schnittstelle
  - Export von initialen Objektreferenzen (z.B. Namensdienst)
  - Verarbeitung von Objektreferenzen (z.B. Konvertierung zwischen String und IOR)



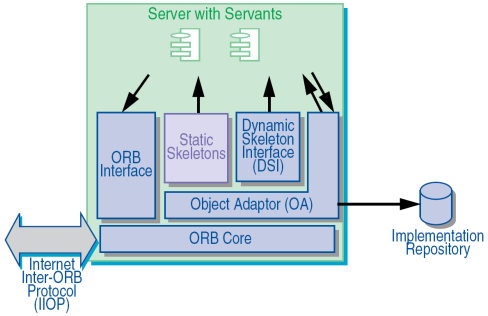
ORB-Core

- Hauptaufgabe
  - Übertragung von Aufrufen mit Hilfe von Informationen in den Objektreferenzen
- General Inter-ORB Protocol (GIOP)
  - Standard-Transportprotokoll zwischen ORBs
  - Grundlage für die Interoperabilität
  - GIOP über TCP-Verbindungen: Internet Inter-ORB Protocol (IIOP)



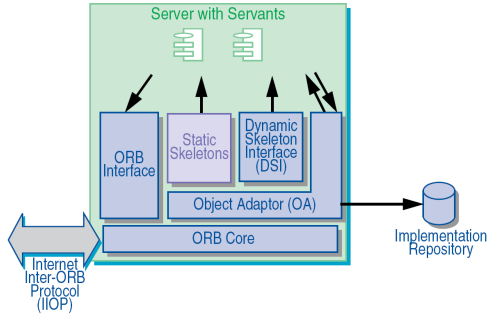
Server-Seite

- Server und Servants
  - Server: Prozess zur Verwaltung von CORBA-Objekt-Implementierungen
  - Servant
    - Implementierung von genau einem CORBA-Objekt
    - Existenz unabhängig von CORBA-Objekt → verschiedene Servants zu verschiedenen Zeiten für das selbe CORBA-Objekt möglich
- Statische Skeletons
  - Automatisch erzeugt aus der IDL-Schnittstelle
  - Unmarshalling der Aufrufparameter
  - Weiterleitung an den Servant
  - Marshalling der Rückgabewerte/ Exceptions
- ORB-Schnittstelle
  [siehe Client-Seite]



Server-Seite (Fortsetzung)

- Object Adaptor
  - Aufrufweiterleitung vom ORB-Core an die Skeletons
  - Erzeugung und Verwaltung von Objektreferenzen
  - Dynamische Aktivierung von Servants
  - Basis-Funktionalität: Portable Object Adaptor
- Implementation Repository
  - Datenbank für Implementierungen von CORBA-Objekten (z.B. Informationen darüber, welches Objekt von welchem Servant implementiert wird)
  - Oft mit *Location-Forwarding Service* kombiniert



Grundlegendes

- Beschreibung von Datentypen und Schnittstellen
- Unabhängig von der/den Implementierungs-Programmiersprache(n)
- Syntax ist stark an C++ angelehnt
  - Beschreibung von Daten und Schnittstellen (Typen, Attribute, Methoden,...)
  - Keine steuernden Anweisungen (`if`, `while`, `for`,...)
- Präprozessor wie in C++
  - `#include` um andere IDL-Dateien einzubinden
  - `#define` für Makros
- Kommentare wie in C++ und Java

```
// Das ist ein einzeiliger Kommentar
/*
 * Das ist ein mehrzeiliger
 * Kommentar
 */
```

Bezeichner (Identifiers)

- Bestimmte reservierte Wörter
  - `module`, `interface`, `struct`, `void`, `long`,...
  - Alle anderen Kombinationen von kleinen und großen Buchstaben, Zahlen und Unterstrichen sind erlaubt (einzige Einschränkung: 1. Zeichen muss ein Buchstabe sein)
  - "_" als Escape-Zeichen für reservierte Wörter
- Sobald ein Bezeichner benutzt ist, sind alle anderen Varianten mit anderer Gross-/Kleinschreibung verboten!
  - Beispiel

```
module Beispiel1 { ... };
module BEISPIEL1 { ... }; // illegal in IDL
```

  - Erlaubt Abbildung von IDL zu Sprachen, die nicht "case-sensitive" sind
  - Erhalte Schreibweise von Bezeichner für "case-sensitive" Sprachen

Module

- Namensraum (*scope*) für IDL-Deklarationen
- Syntax

```
module Name {
    // Deklarationen
};
```

- Zugriff auf andere Namensräume
  - „::"-Operator
  - Beispiel

```
module Beispiel1 {
    typedef long IDNumber;
};

module Beispiel2 {
    typedef Beispiel1::IDNumber MyID;  // typedef long MyID;
};
```

Primitive Datentypen

- Ganzzahlen

`short`	-2^{15} to $2^{15}-1$
`unsigned short`	0 to $2^{16}-1$
`long`	-2^{31} to $2^{31}-1$
`unsigned long`	0 to $2^{32}-1$
`long long`	-2^{63} to $2^{63}-1$
`unsigned long long`	0 to $2^{64}-1$

- Fließkommazahlen

`float`	einfache Genauigkeit
`double`	doppelte Genauigkeit
`long double`	erweiterte Genauigkeit

- Zeichen: `char`
- Wahrheitswert: `boolean`
- Byte: `octet`
- `void`
- Any (Kapselung für beliebigen CORBA-Datentyp): `any`

Strukturen

- Gruppierung von mehreren Typen in einer Struktur
- Beispiel

```
struct AmountType {
    float value;
    char currency;
};
```

- Geschachtelt
  - Strukturen können innerhalb anderer Strukturen definiert werden
  - Beispiel

```
struct AmountType {
    struct ValueType {
        long integerPart;
        short fractionPart;
    } amount;
    char currency;
};
```

  - Strukturen erzeugen einen eigenen Namensraum (*scope*) → Zugriff auf innere Strukturen mit dem „::"-Operator, z.B. `AmountType::ValueType`

Arrays, Sequences & Strings

- Ein- und Mehrdimensionale Arrays (feste Größe)
  - Deklaration mit `typedef`
  - Beispiel

```
typedef long Matrix[3][3];
```

- Eindimensionales Array (variable Größe)
  - Deklaration mit `typedef`
  - Beispiel

```
typedef sequence<long> Longs;
```

- Zeichenkette
  - Ähnlich zu `sequence<char>`
  - Spezieller Datentyp → keine eigene Deklaration mit `typedef` nötig
  - Optional: maximale Größe `n` festlegbar mit `string<n>` (→ `typedef` nötig)
  - Beispiel

```
typedef string<80> Name;
```

Schnittstellen — Übersicht

- Allgemein
  - Schlüsselwort: `interface`
  - Definition eines eigenen Namensraums
  - Eindeutige Definition von Attributen und Operationen: kein Überladen
- Attribute
  - Deklaration öffentlicher Objektvariablen mit `attribute`
  - Schreibzugriff kann verhindert werden (Rein-lesendes Attribut: `readonly`)
  - Beispiel

```
interface Account {
    readonly attribute float balance;
};
```

- Operationen
  - Spezifikation von Methoden-Name, Rückgabe-Datentyp, Aufruf-Parameter, Exceptions
  - Nur Methodenname signifikant: kein Überladen durch Parametertypen
  - Syntax

```
return_type name( parameter_list ) raises( exception_list );
```

Schnittstellen — Parameterübergabe

- Angabe der Übertragungsrichtung für jeden Parameter notwendig
  - `in`: nur vom Client zum Server
  - `out`: nur vom Server zum Client
  - `inout`: in beiden Richtungen
- Beispiel

```
interface Account {
    void makeDeposit( in float sum );
    void makeWithdrawal( in float sum,
                         out float newBalance );
};
```

Schnittstellen — Vererbung

- Ableitung neuer Schnittstellen von existierenden
- Mehrfache Vererbung möglich
- Namen von geerbten Attributen und Operationen müssen eindeutig sein (Ausnahme: Bezeichner, die auf verschiedenen Pfaden geerbt werden, aber von der selben Basisklasse stammen.)
- Beispiel: Weder *Overloading* noch *Overriding* ist erlaubt

```
module Foo {
    interface A {
        void draw( in float num );
    };

    interface B {
        void print( in float num );
        void print( in string str ); // Fehler: Overloading
    };

    interface C: A, B {
        void draw( in float num );   // Fehler: Overriding
    };
};
```

Exceptions

- *Benutzer-Exceptions* werden im Benutzer-Code auf Server-Seite erzeugt und zum Client weitergereicht
- Exception ist eine spezielle Form einer Struktur
  - Nur Datenelemente, keine Operationen
  - Keine Vererbung von Exceptions
- Beispiel

```
interface Account {
    // Deklaration der Exception
    exception Overdraft { float howMuch; };

    // Deklaration einer Methode, die diese Exception wirft
    void makeWithdrawal( in float sum ) raises( Overdraft );
};
```

- Zusätzlich: Bei internen Fehlern wirft der ORB *System-Exceptions*

Allgemeine Bemerkungen

- Abbildungsspezifikation für Java
  - Abbildung von IDL-Datentypen auf Java-Schnittstellen und Klassen
  - Enthält Abbildung für die POA-Schnittstellen
- Ziele
  - Portable Stubs
    - Stubs können über das Netzwerk geladen werden
    - Stubs müssen mit jedem lokal installierten ORB funktionieren, unabhängig vom ORB-Kern
    - Schnittstelle zwischen Stubs und ORB festgelegt, um Austauschbarkeit zu garantieren
  - Umgekehrte Abbildung von Java nach IDL sollte möglich sein
- Hinweise
  - Bezeichner von IDL werden unverändert in Java verwendet
  - Bei Namenskollisionen wird _ (underscore) vorangestellt

Helper-Klassen

- Allgemeines
  - Automatisch erzeugt aus IDL-Schnittstelle
  - Bereitstellung von statischen Hilfsmethoden
- Wichtigste Methoden (Beispiel-Typ `Example` → `ExampleHelper`)
  - Einfügen und Entnehmen des Typs in ein/aus einem `any`-Objekt

```
void insert(org.omg.CORBA.Any a, Example t);
Example extract(org.omg.CORBA.Any a);
```

  - Erfragen von Typ-Code und Typ-Information (Repository-ID)

```
org.omg.CORBA.TypeCode type();
String id();
```

  - Marshalling und Unmarshalling in portablen Stubs

```
void write(org.omg.CORBA.portable.OutputStream ostream,
           Example value);
Example read(org.omg.CORBA.portable.InputStream istream);
```

  - Casten (nur in `Helper`-Klassen von Schnittstellen)

```
Example narrow(org.omg.CORBA.Object obj);
```

Holder-Klassen

- Problem
  - `out` und `inout`-Parameter benötigen *call-by-reference*
  - Java besitzt für Objektreferenzen nur *call-by-value*-Semantik
    → Objektreferenzen können nicht verändert werden
- Lösung: *Holder-Objekte*
  - Statt eigentlicher Objektreferenz wird eine Referenz auf das Holder-Objekt übergeben (→ zusätzliche Indirektionsstufe)
  - Holder-Objekt besitzt Referenz auf das eigentliche Objekt

→ Referenz auf die „Nutzdaten" kann umgebogen werden, da die Referenz auf das Holder-Objekt stabil bleibt

- Beispiel

```
interface Account {                                          // IDL
    void makeWithdrawal( in float sum, out float newBalance );
};

public interface AccountOperations {                         // Java
    public void makeWithdrawal(float sum,
                               FloatHolder newBalance);
}
```

IDLEntity

- Leeres Marker-Interface
- Hinweis darauf, dass
  - eine Klasse/Schnittstelle im Rahmen von CORBA-IDL verwendet wird
  - eine zugehörige Helper-Klasse existiert.
- Verwendung bei Serialisierung und Deserialisierung
- Code

```
package org.omg.CORBA.portable;

public interface IDLEntity extends java.io.Serializable {}
```

Abbildung von Datentypen

- Ganzzahlen: Große `unsigned`-Werte werden in Java negativ!

IDL	Java
`short`	`short`
`unsigned short`	`short`
`long`	`int`
`unsigned long`	`int`
`long long`	`long`
`unsigned long long`	`long`

- Fließkommazahlen

`float`	`float`
`double`	`double`
`long double`	[nicht spezifziert]

- Sonstige

`char`	`char`
`boolean`	`boolean`
`octet`	`byte`

- Any
  - Klasse `org.omg.CORBA.Any`
  - `insert-` und `extract`-Methoden für primitive Datentypen

Abbildung von Modulen und Strukturen

- Module
  `module` → `package`
- Strukturen (`structs`)
  - Abbildung auf "`public final`"-Klasse mit Helper und Holder
  - Beispiel

```
struct Beispiel {                                            // IDL
    float value;
    char currency;
};

final public class Beispiel                                  // Java
             implements org.omg.CORBA.portable.IDLEntity {
    public float value;
    public char currency;

    public Beispiel() {}
    public Beispiel(float value, char currency) {
        this.value = value;
        this.currency = currency;
    }
}
```

Abbildung von Arrays, Sequences & Strings

- Arrays und Sequences
  - Abbildung auf Java-Arrays sowie Helper und Holder
  - Längenüberprüfung beim Marshalling
  - Beispiel

```
typedef long Matrix[3][3];                              // IDL

public class MatrixHelper {...}                         // Java

final public class MatrixHolder
             implements org.omg.CORBA.portable.Streamable {
    public int[][] value;
    [...]
}
```

- Strings
  - Abbildung auf `java.lang.String`
  - Exceptions beim Marshalling, wenn die Länge überschritten wird
  - Holderklasse: `org.omg.CORBA.StringHolder`

Generierte Schnittstellen und Klassen

- Schnittstellen und Hilfsklassen
  - IDL: `interface Example {...};`
  - Java

```
public interface ExampleOperations {...}
public interface Example extends org.omg.CORBA.Object,
                    ExampleOperations,
                    org.omg.CORBA.portable.IDLEntity {...}

final public class ExampleHolder implements
                    org.omg.CORBA.portable.Streamable {...}
abstract public class ExampleHelper {...}
```

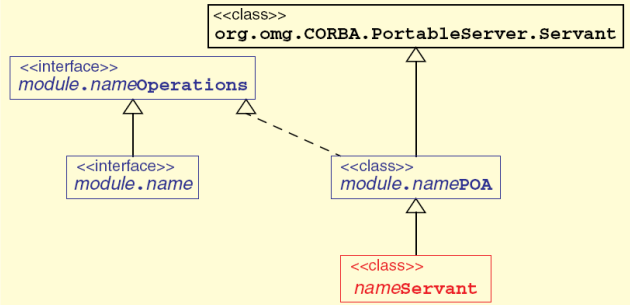
- Stubs und Skeletons

```
public class _ExampleStub                               // Stub
             extends org.omg.CORBA.portable.ObjectImpl
             implements Example {...}

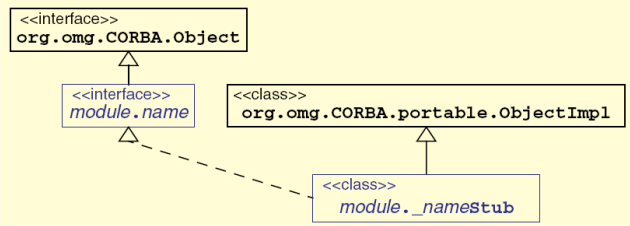
public abstract class ExamplePOA                        // Skeleton
             extends org.omg.PortableServer.Servant
             implements ExampleOperations,
                  org.omg.CORBA.portable.InvokeHandler {...}
```

Stubs & Skeletons — Klassenhierarchien

- Servant-Klassen für das IDL-Interface `module::name`



- Client-Klassen für das IDL-Interface `module::name`



Schnittstellen — Attribute & Operationen

- Attribute
  - Schreibbar: Abbildung auf ein Paar Zugriffsmethoden (Getter und Setter)
  - Nur lesbar: nur Abbildung auf Getter-Methode
- Operationen: direkte Abbildung
- Beispiel
  - IDL

```
interface Example {
    attribute float a;
    readonly attribute float b;
    void foo( in string s ) raises( ExampleException );
};
```

  - Java

```
public interface ExampleOperations {
    public float a();
    public void setA(float a);
    public float b();
    public void foo(String s) throws ExampleException;
}
```

CORBA-Objekt-Schnittstelle

`org.omg.CORBA.Object`

- Basisklasse für CORBA-Objekte: `org.omg.CORBA.Object`
  - Vergleiche: `java.lang.Object` für Java-Objekte
  - Unterscheidung im Code notwendig: `Object` vs. `org.omg.CORBA.Object`
- Wichtigste Methoden
  - `InterfaceDef _get_interface()`
    - Liefert eine Interface-Beschreibung (vom Interface Repository) für dieses Objekt zurück
    - Verwendung in Verbindung mit dem Dynamic Invocation Interface (DII)
  - `boolean _is_a(String identifier)`

    Überprüft, ob das Objekt eine bestimmte Schnittstelle implementiert
  - `boolean _is_equivalent(Object that)`
    - Überprüft, ob zwei Referenzen auf das selbe CORBA-Objekt zeigen
    - Achtung: nur "best effort"-Semantik (wahr ? selbes Objekt : **wahrscheinlich** verschieden)
  - `int _hash(int maximum)`

    Hash, um Objektreferenzen zu unterscheiden

ORB-Schnittstelle

`org.omg.CORBA.ORB`

- Abbildung auf abstrakte Klasse `org.omg.CORBA.ORB`
- Wichtigste Methoden
  - `String object_to_string(org.omg.CORBA.Object obj)`

    Umwandlung von Objektreferenzen in eindeutige Strings
  - `org.omg.CORBA.Object string_to_object(String str)`

    Umwandlung von Strings in Objektreferenzen
  - `String[] list_initial_services()`

    Liste von Diensten, die der ORB kennt, z.B. Namensdienst
  - `org.omg.CORBA.Object resolve_initial_references(String object_name)`
    - Liefert Objektreferenz auf den angeforderten ORB-Dienst zurück
    - `object_name` spezifiziert den Dienst, z.B. "NameService"
  - `ORB init(Strings[] args, Properties props)`

    Statische Methode zur Initialisierung eines ORBs

CORBA-Skeletons

- Abstrakte Basisklasse: `org.omg.PortableServer.Servant`
  - Alle vom IDL-Compiler generierten Skeletons (`*POA`) sind direkte (ebenfalls abstrakte) Unterklassen
  - Servant-Implementierungen sind wiederum (nicht-abstrakte) Unterklassen der generierten Skeletons
- Skeleton-Methoden
  - ```
    OutputStream _invoke(String op,
                         org.omg.CORBA.portable.InputStream i,
                         org.omg.CORBA.portable.ResponseHandler rh)
    ```
    - Unmarshalling der Paramter von `InputStream`
    - Weiterleitung an die Operation namens op
    - `ResponseHandler` stellt `OutputStream` bereit
    - Marshalling der Rückgabewerte nach `OutputStream`
  - `Example _this(org.omg.CORBA.ORB orb)`
    - Aktiviert ein neues CORBA-Objekt
    - Ordnet den Servant dem neuen CORBA-Objekt zu
    - Liefert eine Objektreferenz zurück (einen lokalen Stub)
    - Mechanismus wird auch „implizite Aktivierung" genannt


    ```

Initialisierung

- POA-Aktivierung
  - Referenz auf RootPOA via `resolve_initial_references`

```
org.omg.CORBA.Object o = orb.resolve_initial_references(``RootPOA'');
```

  - "Narrow" auf das POA-Interface (`org.omg.PortableServer.POA`)

```
org.omg.PortableServer.POA poa =
                org.omg.PortableServer.POAHelper.narrow(o);
```

  - POA aktivieren via `org.omg.PortableServer.POAManager`

```
poa.the_POAManager().activate();
```

- ORB Hauptschleife
  - Verarbeitung von Anforderungen starten
  - Methode: `run()` in `org.omg.CORBA.ORB`

```
orb.run();
```

"Hello World"-Servant

- IDL-Interface: `Hello.idl`

```
module Beispiel {
    interface Hello {
        string say( in string msg );
    };
};
```

- Servant-Implementierung mittels Vererbung

```
public class HelloServant extends HelloPOA {
    // Constructor
    public HelloServant() {
        super();
    }

    // Operation Beispiel::Hello::say from IDL
    public String say(String msg) {
        return "Hello" + msg;
    }
}
```

"Hello World"-Server

Server-Implementierung

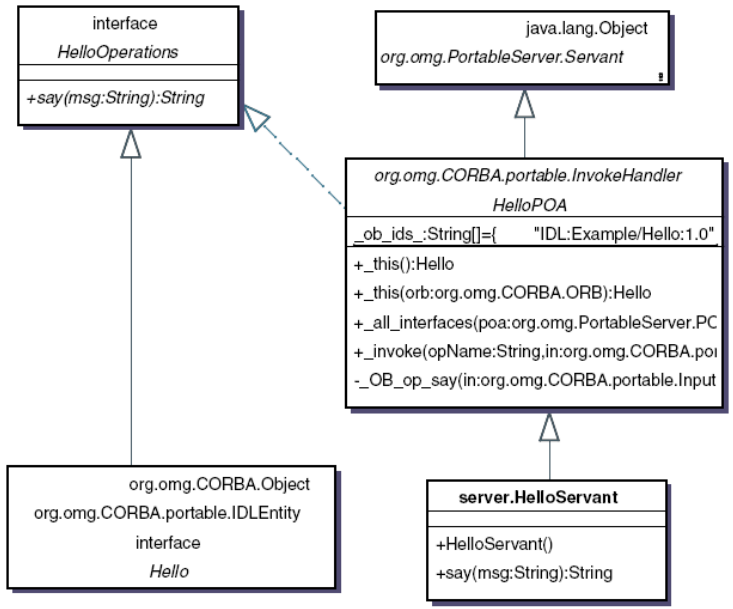
```
public class HelloServer {
    public static void main(String[] args) throws Exception {
        ORB orb = ORB.init(args, null);
        POA poa = POAHelper.narrow(
                  orb.resolve_initial_references("RootPOA"));
        poa.the_POAManager().activate();

        HelloServant hServ = new HelloServant();
        org.omg.CORBA.Object obj =
                  poa.servant_to_reference(hServ);

        // Write object reference to file "Hello.ior"
        PrintWriter pw = new PrintWriter(
                             new FileWriter("Hello.ior"));
        pw.println(orb.object_to_string(obj);
        pw.close();

        orb.run(); // Wait for request
    }
}
```

"Hello World"-Server Klassen



"Hello World"-Client — Implementierung

Client-Implementierung

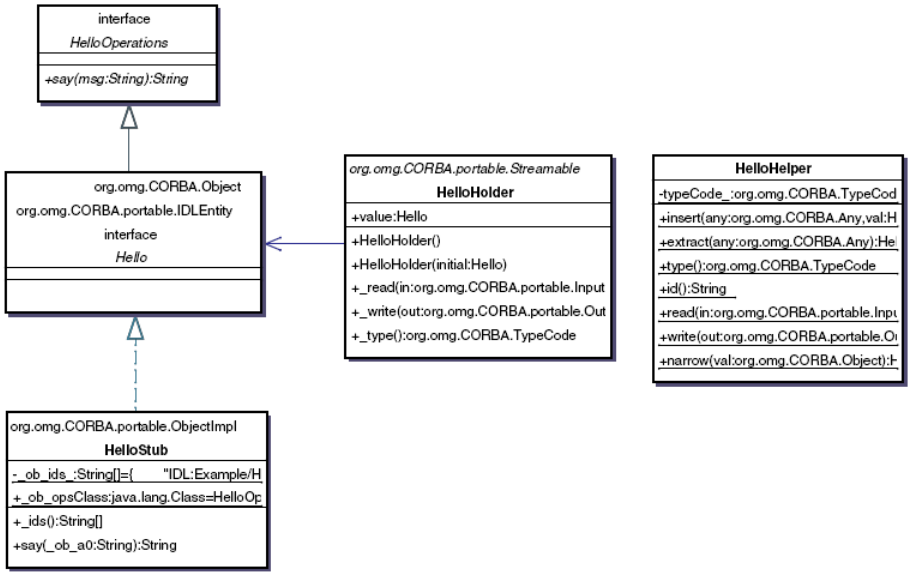
```
public class HelloClient {
    public static void main(String[] args) throws Exception {
        ORB orb = ORB.init(args, null);

        // Read object reference from file "Hello.ior"
        BufferedReader br = new BufferedReader(
                                new FileReader("Hello.ior"));
        String s = br.readLine();
        br.close();

        // Create a stub object
        org.omg.CORBA.Object o = orb.string_to_object(s);
        // Narrow to the Hello interface
        Hello hello = HelloHelper.narrow(o);

        // Do the call
        System.out.println(hello.say(" world!"));
    }
}
```


"Hello World"-Client — Klassen



"Hello World" — Starten

- Client
  - Kompilieren und Starten mit JDK 1.5 (`/local/java-1.5/`)

```
> idlj [-fclient] Hello.idl
> javac HelloClient.java
> java HelloClient
```

  - Kompilieren und Starten mit JacORB (`/local/JacORB/`)

```
> idl Hello.idl
> javac HelloClient.java
> jaco HelloClient

> java -Xbootclasspath:${JACORB_HOME}/lib/jacorb.jar:\
    ${JRE_HOME}/lib/rt.jar:${CLASSPATH} \
    -Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB \
    -Dorg.omg.CORBA.ORBSingletonClass=
                org.jacorb.orb.ORBSingleton HelloClient
```

- Server: analog, außer

```
> idlj -fserver Hello.idl
```


Aufgabe 3

- Once again...

  Bibliotheksverwaltung (dieses Mal eben mit CORBA)
- Details
  - Jedes Medienobjekt ist ein CORBA-Objekt (mit eigenem Servant)
  - Verwendung des CORBA-Namensdients
  - Einsatz von SunORB und JacORB
- Herausforderung
  - Viele Medienobjekte → großer Ressourcenverbrauch
  - Deshalb: Beschränkte Anzahl an gleichzeitig aktiven Servants

→ Dynamische Deaktivierung und Aktivierung von Servants

→ Speicherung von Daten in einer persistenten Datenbank