C Abbildung von IDL nach Java

- Abbildung der IDL-Datentypen nach Java
- Abbildung von Objektschnittstellen nach Java

1 Allgemeine Bemerkungen

- Abbildung von IDL-Datentypen auf Java-Schnittstellen und Klassen
- Abbildungs-Spezifikation Version 1.2 vom August 2002
- Enthält Abbildung für die POA-Schnittstellen.
- Wichtige 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.
- Bezeichner von IDL werden unverändert in Java verwendet.
  - Bei Namenskollisionen wird _ (underscore) vorangestellt.

2 Primitive Typen

- Ganzzahlen
  - **`short`** wird zu **`short`**
  - **`unsigned short`** wird zu **`short`**
  - **`long`** wird zu **`int`**
  - **`unsigned long`** wird zu **`int`**
  - **`long long`** wird zu **`long`**
  - **`unsigned long long`** wird zu **`long`**
  - Grosse **`unsigned`**-Werte werden in Java negativ!
- Fliesskommazahlen
  - **`float`** wird zu **`float`**
  - **`double`** wird zu **`double`**
  - **`long double`** bisher nicht spezifiziert

2 Primitive Types (2)

- Zeichen
  - **`char`** wird zu **`char`**
  - **`wchar`** wird zu **`char`**
  - Weil Java **`char`** eine Obermenge von IDL **`char`** enthalten kann, kann das Marshalling eine **`CORBA::DATA_CONVERSION`**-Exception auslösen.
- **`boolean`** bleibt **`boolean`**
- **`octet`**
  - wird zu **`byte`**
- **`any`**
  - Klasse **`org.omg.CORBA.Any`**
  - **`insert`**- und **`extract`**-Methoden für primitive Datentypes
  - Für andere Datentypen **`insert-`** and **`extract`**-Methoden in der Helper-Klasse dieses Types

3 Helper-Klassen

- Eine Helper-Klasse für jeden IDL-Typ (hier *name*)

```
public class nameHelper {
    public static void insert( org.omg.CORBA.Any a, Name t )
        {...}
    public static name extract( org.omg.CORBA.Any a ) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static name read(
        org.omg.CORBA.portable.InputStream istream ) {...}
    public static void write(
        org.omg.CORBA.portable.OutputStream ostream,
        name value ) {...}

    // nur für Helper von einem Interface
    public static name narrow( org.omg.CORBA.Object obj )
        {...}
}
```


4 Holder-Klassen

- Java besitzt nur call-by-value-Semantik (Objektreferenzen können nicht verändert werden!).
- **out**- und **inout**-Parameters benötigen call-by-reference.
- Kapselung der Parameter in einem Holder-Objekt (hier für Typ *name*)

```
final public class nameHolder
          implements org.omg.CORBA.portable.Streamable {
    public name value;

    public nameHolder() {...}
    public nameHolder( name initial ) {...}

    public void _read( org.omg.CORBA.portable.InputStream i )
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o )
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```


3 Helper-Klassen (2)

- **extract** und **insert**
  - ◆ Methoden zum Einfügen und Entnehmen dieses Tyes in/aus **any**-Object
- **type** und **id**
  - ◆ Erfragen von Typ-Code und Typ-Information (Repository-ID) für diesen Typ
- **read** und **write**
  - ◆ Methoden zum Marshalling und Demarshalling in portablen Stubs
- **narrow**
  - ◆ Exisitiert nur in Helper-Klassen für **interface**
  - ◆ Modifikation der sichtbaren Schnittstelle einer Objektreferenz (Casting)

5 IDLEntity

- Leere Markierungs-Schnittstelle
- Geerbt bei allen aus IDL generierten Schnittstellen und Klassen
- Deklaration:

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

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

6 Module

- IDL:

```
module name {
    Deklarationen
};
```

- Abbildung zu Java "**package**"
- Java:

```
package name;

Abbildung für Deklarationen
```

- Abbildung vom Modul **CORBA** zum Package **org.omg.CORBA**

7 Typ-Deklarationen

- IDL:

```
typedef existing_type alias;
```

- Java:
  - Nur Helperklasse für *alias*
  - Für den Typ selbst muss Abbildung von *existing_type* benutzt werden.
- Beispiel:

```
// IDL
typedef long IDNumber;

// Java
public class IDNumberHelper {...}
final public class IDNumberHolder
        implements org.omg.CORBA.portable.Streamable {
    public int value;
    ...
}
```


8 Strukturen

- IDL:

```
struct name {
    Deklaration von struct-Elementen
};
```

- Abbildung zu einer "**public final**"-Klasse mit Helper und Holder
  - Elemente werden zu "**public**" Variablen
  - Leerer Konstruktor, sowie Konstruktor, der alle Variablen initialisiert
- Java:

```
public final class name
    implements org.omg.CORBA.portable.IDLEntity
{
    Mapping for structure elements as public variables
    public name() {}
    public name( Mapping_for_structure_elements ) {...}
}
```


8 Strukturen (2)

- Beispiel:

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

// Java
final public class Beispiel
    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;
    }
}
```

8 Geschachtelte Strukturen

- Beispiel:

```
struct Outer {
    struct Inner {
        char foo;
    } fooBar;
};
```

- **`Inner`** wird auf Klasse **`Inner`** mit Helper- und Holderklasse abgebildet, innerhalb eines Unter-"**`package`**" namens **`OuterPackage`**.
  - Outer.java:

```
final public class Outer ... {
    public OuterPackage.Inner fooBar;
}
```

  - OuterPackage/Inner.java:

```
package OuterPackage;
final public class Inner ... {
    public char foo;
}
```


9 Unions

- IDL:

```
union name switch( switch_type ) {
    case switch_constant: Declaration
    ...
    default: Declaration
};
```

- Abbildung auf eine "**`public final`**" Klasse mit **`Helper`**- und **`Holder`**
  - ◆ Zugriffs-Methode für den Diskriminator
  - ◆ Zugriffs-Methoden für alle Zweige in der Union
  - ◆ Zugriffs-Methoden für einen "default"-Diskriminator (setzt Diskriminator auf nicht verwendeten Wert)
- Java:

```
public final class name
    implements org.omg.CORBA.portable.IDLEntity
{
    public name() {}
    public Mapping_for_switch_type discriminator() {...}
    public Mapping_for_union_element union_element_name() {...} //get
    public union_element_name( Mapping_for_union_element ){...} //set
}
```


9 Unions (2)

- Beispiel:

```
// IDL
union Beispiel switch( long ) {
    case 1:         long l;
    case 2:         float f;
};

// Java
final public class Beispiel
    implements org.omg.CORBA.portable.IDLEntity
{
    private int ___l;
    private float___f;
    private int _discriminator;
    private int _defdisc = -2147483648;
    public Beispiel() {}
    public int discriminator() { return _discriminator; }
    public int l() {...}                        // get l
    public void l( int value ) {...}            // set l
    public float f() {...}                      // get f
    public void f( float value ) {...}          // set f
    public void _default() {...}                // set to impossible discr.
}
```


10 Aufzählungen

- IDL:

```
enum name {
    value1, value2, ...
};
```

- Abbildung auf eine "**`public final`**"-Klasse mit **`Helper`** and **`Holder`**
  - ◆ Werte werden auf Integer-Werte abgebildet (Bezeichner *_value1*, ...).
  - ◆ Statische Instanzen innerhalb der Enumeration-Klasse
- Java:

```
public final class name
        implements org.omg.CORBA.portable.IDLEntity
{   public static final int _value1 = int_value1;
    public static final name value1 = new name( _value1 );
    ...
    private final int __value;
    private name( int value ) { this.__value = value; }
    public int value() { return __value; }
    public static name from_int( int value ) {...};
}
```

10 Aufzählungen

- Beispiel:

```
// IDL
enum Color { GREEN, RED, BLUE };

// Java
final public class Color
    implements org.omg.CORBA.portable.IDLEntity
{
    private int __value;
    private int __size = 3;

    final public static int _GREEN = 0;
    final public static Color GREEN = new Color( _GREEN );
    final public static int _RED = 1;
    final public static Color RED = new Color( _RED );
    final public static int _BLUE = 2;
    final public static Color BLUE = new Color( _BLUE );

    private Color( int value ) { this.__value = value; }
    public int value() { return __value; }
    public static Color from_int( int value ) {
        switch( value ) {...}
    }
}
```


11 Arrays

- IDL:

```
typedef element_type name[positive_constant][positive_constant]...;
```

- Abbildung auf Java Arrays, sowie **`Helper`** und **`Holder`**
    - ◆ Array-Elemente haben den Datentyp, der sich aus der Java-Abbildung von *element_type* ergeben.
- Beispiel:

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

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


12 Sequences

- IDL:

```
typedef sequence<element_type> name;                          // unbounded
typedef sequence<element_type,positive_constant> name;// bounded
```

- Abbildung genauso wie bei eindimensionalen Arrays
    - ◆ Längenüberprüfung für "bounded sequences" wird nur beim Marshalling gemacht.
- Beispiel:

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

// Java
public class LongsHelper {...}
final public class LongsHolder
        implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    ...
}
```


13 Zeichenketten

- IDL:

```
typedef string name;                                  // unbounded
typedef string<positive_constant> name;               // bounded
typedef wstring name;                                 // unbounded
```

- Abbildung auf **`java.lang.String`**
    - ◆ Exceptions beim Marshalling, wenn die Länge überschritten wird oder Zeichen nicht auf CORBA **`char`** abgebildet werden können
    - ◆ Holderklasse: **`org.omg.CORBA.StringHolder`**
- Beispiel:

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

// Java
public class NameHelper {...}
```

14 Festkommazahlen

- IDL:

```
typedef fixed<positive_constant,scaling_constant> name;
```

- Abbildung auf **java.math.BigDecimal**
  - Bereichsüberprüfung beim Marshaling
  - Holderklasse: **org.omg.CORBA.FixedHolder**
- Beispiel:

```
// IDL
typedef fixed <20,2> amount;

// Java
public class amountHelper {...}
final public class amountHolder
        implements org.omg.CORBA.portable.Streamable {
    public java.math.BigDecimal value;
    ...
}
```


15 Konstanten

- Symbolische Namen für spezielle Werte
- IDL:

```
const type name = constant_expression;
```

- Abbildung von lokalen Konstanten in IDL-Schnittstellen
  - **"final public static"**-Variablen im Java-Interface
  - Beispiel:

```
// IDL
interface Beispiel {
    const Color WARNING = RED;
};

// Java
public interface Beispiel ... {
    final public static Color WARNING = (Color) Color.RED;
    ...
}
```


15 Konstanten (2)

- Abbildung von Konstanten außerhalb von IDL-Schnittstellen
  - Eigene Klasse mit Namen der Konstanten und lokalen Wert *value*
  - Beispiel:

```
// IDL
module Beispiel {
    const Color WARNING = RED;
};

// Java
package Beispiel;
public interface WARNING {
    final public static Color value = (Color) Color.RED;
};
```


16 Schnittstellen

- IDL:

```
interface name {
    Deklaration von Attributen und Operationen (sowie Typen und Exceptions)
};
```

- Abbildung auf:
  - **public** Java interface *name***Operations**
  - **public** Java interface *name*
  - *name***Helper**- und *name***Holder**-Klassen
  - Stub- und Skeleton-Klasse

16 Schnittstellen (2)

- Java:

```
public interface nameOperations
{
    Mapping for attributes and operations
}
public interface name extends org.omg.CORBA.Object,
        nameOperations, org.omg.CORBA.portable.IDLEntity
{...}

final public class nameHolder
        implements org.omg.CORBA.portable.Streamable {...}
public class nameHelper {...}
```


16 Schnittstellen – Attribute

- IDL:

```
attribute type name;                 // read & write
readonly attribute type name;        // read-only
```

- Abbildung auf ein Paar von Zugriffsmethoden
- Java:

```
public Mapping_for_type name();           // get attribute
public void name( Mapping_for_type );     // set attribute (not if read-only)
```

- Beispiel:

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

// Java
public interface AccountOperations {
    public float balance();
}
```


16 Schnittstellen – Vererbung

- IDL:

```
interface name : inherited_interface1, inherited_interface2, ... {
    Declaration of additional attributes and operations
};
```

- Abbildung auf Vererbung von Java-Schnittstellen
- Java:

```
public interface nameOperations
        extends inherited_interface1Operations,
                inherited_interface2Operations, ...
{
    Mapping for additional attributes and operations
}
public interface name
        extends inherited_interface1, inherited_interface2, ...,
        nameOperations, org.omg.CORBA.portable.IDLEntity
{}
final public class nameHolder
        implements org.omg.CORBA.portable.Streamable {...}
public class nameHelper {...}
```


16 Schnittstellen – Operationen

- IDL:

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

- Abbildung auf Methoden im Java-Interface
- Java:

```
public Mapping_for_return_type name( Mapping_for_parameter_list )
    throws Mapping_for_exception_list;
```

16 Schnittstellen – Parameterübertragung

- IDL:
  `( copy_direction1 type1 name1, copy_direction2 type2 name2, ... )`
- Abbildung von Parametertypen hängt von der Kopierrichtung ab:
  - `in` nach *Mapping_for_type*
  - `out` und `inout` nach *type*`Holder`
- Beispiel:

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

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


16 Schnittstellen

- Beispiel:

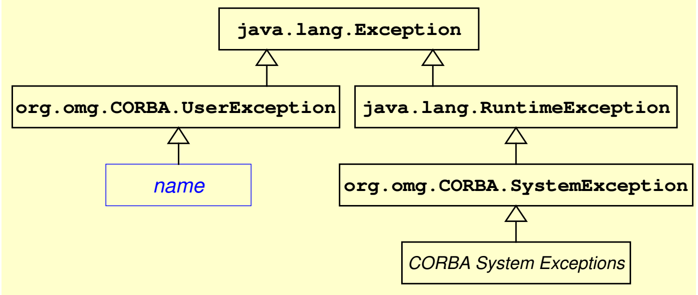
```
//IDL
module Bank {
    interface Account {
        void withdraw(in double amount);
        void deposit(in double amount);
        void transfer(inout Account src, in double amount);
        readonly attribute double balance;
    };
};

// Java
package Bank;
public interface AccountOperations {
    public void withdraw(double amount);
    public void deposit(double amount);
    public void transfer(AccountHolder src, double amount);
    public double balance();
}
public interface Account extends org.omg.CORBA.Object,
    AccountOperations, org.omg.CORBA.portable.IDLEntity {}
```


17 Exceptions

- IDL:

```
exception name {
    Declarations of data elements
};
```

- Abbildung auf "**`final public`**"-Klasse in folgender Klassenhierarchie:



17 Exceptions (2)

- Abbildung von lokalen Exceptions innerhalb eines IDL-Interface *name* auf "**`final public`**"-Klassen im Paket *name*`Package`
- Beispiel:

```
// IDL
interface Account {
    exception Overdraft { float howMuch; };
    void withdraw( in double amount )raises( Overdraft );
};

// Java
public interface AccountOperations {
    public void withdraw( double amount )
        throws AccountPackage.Overdraft;
}
public interface Account extends ... {}

package AccountPackage;
final public class Overdraft
        extends org.omg.CORBA.UserException {
    public float howMuch;
    ...
}
```

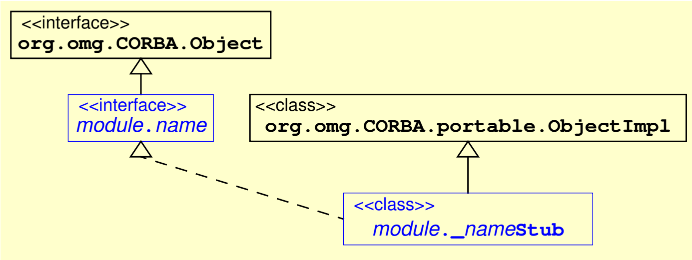
17 Exceptions (3)

- Zusätzlich: Erzeugung von Helper- und Holder-Klassen
- Abbildung von CORBA-Systemexceptions auf "**`final public`**"-Unterklassen von **`org.omg.CORBA.SystemException`**

```
package org.omg.CORBA;
abstract public class SystemException
        extends java.lang.RuntimeException {
    public int minor;
    public CompletionStatus completed;
    protected SystemException(      String reason, int minor,
                                    CompletionStatus status) {
        super(reason); this.minor = minor;
        this.status = status;
    }
}
// CORBA::UNKNOWN
final public class UNKNOWN
        extends org.omg.CORBA.SystemException {
    public UNKNOWN() ...
    public UNKNOWN(String reason) ...
    ...
}
```


18 Stubs

- Klient hat lediglich eine Java-Referenz auf das lokale Proxy-Objekt.
  - Stub-Objekt
  - Stub-Klasse wird automatisch aus der IDL-Beschreibung erzeugt
  - Stub-Objekte sind transparent für den Benutzer – Automatische Erzeugung und Vernichtung durch das CORBA-System.
- Klassenhierarchie für IDL-**`interface`** *module::name*



19 Zusammenfassung der Java-Abbildung

- Für jede IDL-Definition gibt es zwei Klassen:
  - Eine Holder-Klasse zur Übertragung von "out"- und "inout"-Parametern
  - Eine Helper-Klasse zum Marshalling und zum Einfügen/Extrahieren in/aus **`any`**-Objekten
- Primitive Datentypen werden auf Java-Typen abgebildet (Achtung: keine eins-zu-eins-Beziehung).
- IDL-Arrays und Sequenzen sind Java-Arrays.
- Für andere konstruierte Typen gibt es spezielle Klassen.
- IDL-Schnittstellen werden auf Java-Schnittstellen abgebildet.
- IDL-Exceptions sind spezielle Java-Exception-Klassen.
- Klienten besitzen Java-Referenzen auf Stub-Objekte.

C.1 Java-Klienten

- Was fehlt noch für einen vollständigen Klienten?
- Wie bekommt man die erste Referenz auf ein CORBA-Objekt?
- Beispiel "Hello World!"

1 CORBA Pseudo-Objekte

- Keine echten CORBA-Objekte
  - Nur lokal sinnvoll
  - Keine entfernter Zugriff möglich.
- Beschreibung der Schnittstelle in Pseudo-IDL (PIDL)
  - Syntax wie IDL
  - Sprachabbildung kann spezielle Abbildung für jedes Pseudo-Interface definieren.
- Beispiele:
  - **`CORBA::Object`** Features von CORBA-Objektreferenzen
  - **`CORBA::ORB`** Schnittstelle zu ORB-Features
  - **`PortableServer::POA`** Schnittstelle zum Portable Object Adaptor
- Seit CORBA 2.5: Spezifikation von lokalen Schnittstellen in IDL ("local interface")

2 Objektreferenzen – CORBA::Object

- **`InterfaceDef get_interface()`**
  - Liefert eine Interface-Beschreibung (vom Interface Repository) für dieses Objekt zurück.
  - Normalerweise in Verbindung mit dem Dynamic Invocation Interface (DII) verwendet
- **`boolean is_nil()`**
  - Überprüft, ob dies eine gültige Objektreferenz ist.
- **`Object duplicate()`**
  `void release()`
  - Kopieren und Löschen von Objektreferenzen
  - Referenzzählung nur lokal im Klienten
  - Die Objektimplementierung wird nicht informiert.

2 Objektreferenzen – CORBA::Object

- Operationen, die ein Klient an jedem CORBA-Objekt aufrufen kann:

```
module CORBA {
    interface Object {                                    // PIDL
        InterfaceDef get_interface();
        boolean is_nil();
        Object duplicate();
        void release();
        boolean is_a( in string logical_type_id );
        boolean non_existent();
        boolean is_equivalent( in Object other_object );
        unsigned long hash( in unsigned long maximum );
        Status create_request(   in Context ctx,
                                 in Identifier operation,
                                 in NVList arg_list,
                                 inout NamedValue result,
                                 out Request request,
                                 in Flags req_flags );
        ...
    };
};
```


2 Objektreferenzen – CORBA::Object

- **`boolean is_a( in string logical_type_id )`**
  - Überprüft, ob das Objekt eine bestimmtes Interface implementiert.
  - Interface Repository ID als String, z.B. **`IDL:Bank/Account:1.0`**
- **`boolean non_existent()`**
  - Überprüft, ob dies eine Implementierung für dieses Objekt ist.
- **`unsigned long hash( in unsigned long maximum )`**
  - Hash, um Objektreferenzen zu unterscheiden
- **`boolean is_equivalent( in Object other_object )`**
  - Überprüft, ob zwei Referenzen auf das selbe CORBA-Objekt zeigen.
  - Achtung: nur "best-effort"-Semantics
    - true: Referenzen zeigen auf das selbe Objekt.
    - false: Referenzen zeigen wahrscheinlich auf verschiedene Objekte.
- **`Status create_request( ... )`** Erzeuge einen DII-Request

3 Objektreferenzen – org.omg.CORBA.Object

- Java-Abbildung auf das Interface **org.omg.CORBA.Object**

```
package org.omg.CORBA;

public interface Object {
    boolean _is_a( String Identifier );
    boolean _is_equivalent( Object that );
    boolean _non_existent();
    int _hash( int maximum );
    org.omg.CORBA.Object _duplicate();
    void _release();
    ImplementationDef _get_implementation();
    InterfaceDef _get_interface();
    ...
}
```

- **duplicate** und **release** nicht wirklich notwendig.
  - ◆ Java verwendet internen Garbage Collector anstatt Referenzzählung.
- Achtung: Einfach "**Object**" bedeutet immer **java.lang.Object**!

4 Die ORB-Schnittstelle – CORBA::ORB

- Standard-Operationen des ORBs

```
module CORBA {
    interface ORB {                                  // PIDL
        string object_to_string( in Object obj );
        Object string_to_object( in string str );

        typedef string ObjectId;
        typedef sequence<ObjectId> ObjectIdList;
        exception InvalidName {};
        ObjectIdList list_initial_services();
        Object resolve_initial_references(
                in ObjectId identifier ) raises (InvalidName);

        ...
    };
};
```


4 Die ORB-Schnittstelle – CORBA::ORB

- **string object_to_string(in Object obj)**
  Object string_to_object(in string str)
  - ◆ Umwandlung von Objektreferenzen in eindeutige Strings und umgekehrt
  - ◆ String-Format: **IOR:00202020**...
- **ObjectIdList list_initial_services()**
  - ◆ Liste von Diensten, die der ORB kennt, z.B. Namensdienst
- **Object resolve_initial_references(**
  in ObjectId identifier) raises (InvalidName)
  - ◆ Liefert Objektreferenz auf den angeforderten ORB-Dienst zurück.
  - ◆ **ObjectId** ist ein String, z.B. "**NameService**".

5 Die ORB-Schnittstelle – org.omg.CORBA.ORB

- Java-Abbildung auf abstrakte Klasse **org.omg.CORBA.ORB**

```
package org.omg.CORBA;

public abstract class ORB {

    public abstract String[] list_initial_services();
    public abstract org.omg.CORBA.Object
        resolve_initial_references( String object_name )
            throws org.omg.CORBA.ORBPackage.InvalidName;

    public abstract String
        object_to_string( org.omg.CORBA.Object obj );
    public abstract org.omg.CORBA.Object
        string_to_object( String str );

    ...
}
```

6 ORB-Initialisierung

- Erster Schritt in jeder CORBA-Anwendung
- Liefert Referenz auf ein **CORBA::ORB**-Objekt.
- PIDL-Spezifikation

```
module CORBA {                                   // PIDL
    typedef string ORBid;
    typedef sequence<string> arg_list;
    ORB ORB_init(  inout arg_list argv,
                   in ORBid orb_identifier);
};
```

- Auswahl eines ORBs (falls es mehr als einen gibt) via ORBid
- ORB-Parameter in Kommandozeilenargumenten
  - z.B. **-ORB**<suffix> <value>

7 ORB-Initialisation – org.omg.CORBA.ORB

- Java: Initialisierung über statische Methoden in **org.omg.CORBA.ORB**

```
public abstract class ORB {
    ...
    public static ORB init(Strings[] args,
                           Properties props );
    public static ORB init( Applet app, Properties props );
    public static ORB init();
    ...
}
```

  - Spezielle **init**-Methode für ORB innerhalb eines Applet
  - **init()** ohne Parameter liefert nur einen Singleton-ORB.
    - ➤ Kann nur spezielle Strukturen wie Typecodes erzeugen
    - ➤ Nicht geeignet für entfernte Methodenaufrufe
- Java-Properties zur Auswahl von ORB-Features, z.B.
  - **org.omg.CORBA.ORBClass** Klasse, die von **init** geliefert wird (implementiert **org.omg.CORBA.ORB**), z.B. **com.ooc.CORBA.ORB**

8 "Hello World"-Client - Interface

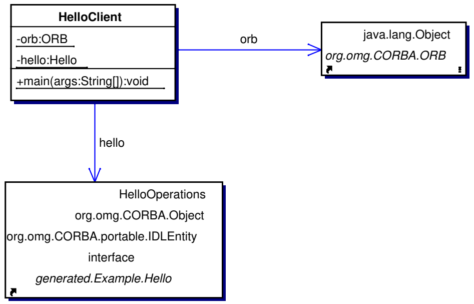
- IDL-Interface: Hello.idl

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

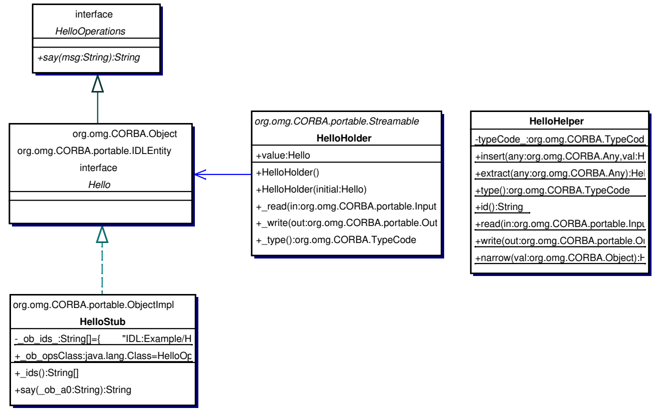

8 "Hello World"-Client - Implementierung

```
// client/HelloClient.java
package client;

import generated.Example.*;
import org.omg.CORBA.*;

public class HelloClient {
    public static void main( String[] args ) {
        try {
            // Initialise ORB
            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();
            // 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!" ) );
        } catch(Throwable t) {
            t.printStackTrace();
        }
    }
}
```

8 "Hello World"-Client - Klassen



8 "Hello World"-Client – Generierte Klassen



8 "Hello World"-Client - Starten

- In das Beispiel-Verzeichnis wechseln

```
> cd /proj/i4oovs/pub/hello
```

- Compilieren&Starten mit JDK 1.4 (/local/java-1.4/)

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

- Compilieren&Starten mit JacORB (/local/JacORB/)

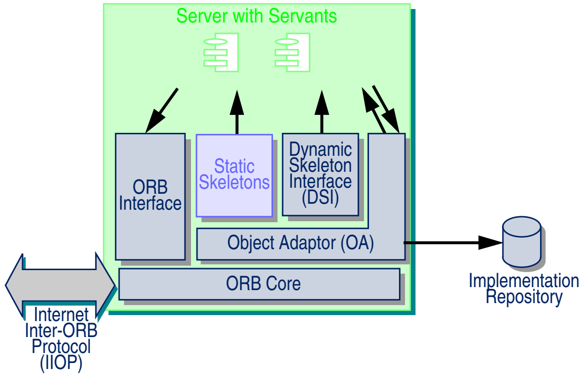
```
> idl Hello.idl
> javac HelloClient.java
> jaco HelloClient

> 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
```


C.2 Implementierung von CORBA-Objekten

1 CORBA-Objekte

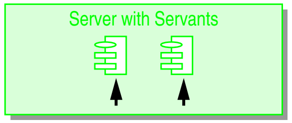
- Zugriff auf CORBA-Objekte über Objektreferenzen
  - ◆ CORBA-Objekt existiert, sobald die erste Referenz erzeugt wird.
- Funktionalität von CORBA-Objekten durch Servants bereitgestellt
  - ◆ Servants sind in einer Programmiersprache geschrieben.
  - ◆ Servant existiert möglicherweise noch nicht, wenn das CORBA-Objekt erzeugt wird.
  - ◆ Zu jeder Zeit höchstens ein Servant pro CORBA-Objekt
  - ◆ Aber verschiedene Servants zu verschiedenen Zeitpunkten möglich

2 Server-Architectur



3 Der Server und Servants

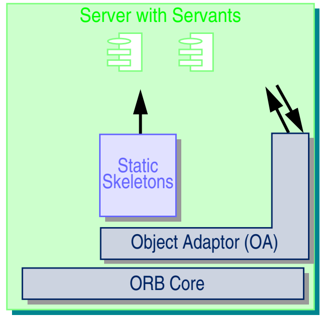
- Server
  - Prozess, der Implementierungen (Servants) von CORBA-Objekten verwaltet



- Servant
  - Implementierung von genau einem CORBA-Objekt
  - In OO-Sprachen: Spezielles Objekt, das die IDL-Schnittstelle implementiert
  - In Nicht-OO-Sprachen: Menge von Funktionen, die die IDL-Schnittstelle implementieren, und eine Datenstruktur zur Identifikation der Instanz
  - Viele Servants pro Server möglich

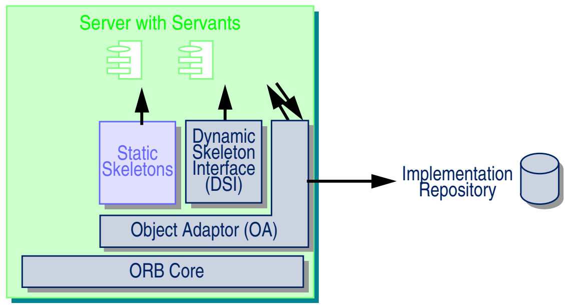
4 Statische Skeletons

- Können ebenfalls aus der IDL-Schnittstelle erzeugt werden
- Demarshalling von Aufrufparametern, Weiterleitung an den Servant
- Marshalling von Rückgabewerten oder Exceptions des Aufrufs



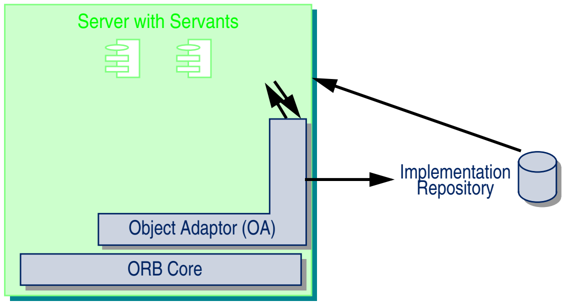
5 Object Adaptor

- Aufrufweiterleitung vom ORB-Core an die Skeletons
- Erzeugung und Verwaltung von Objektreferenzen
- Dynamische Aktivierung von Servants
- bis CORBA 2.1: Basic Object Adaptor (BOA), jetzt Portable OA (POA)



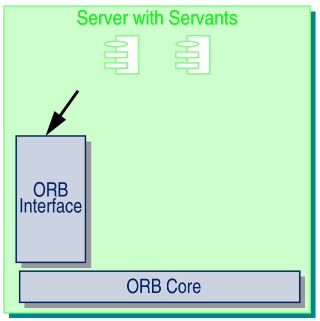
6 Implementation Repository

- Datenbank für Implementierungen von CORBA-Objekten
  - ◆ Informationen über welches Objekt von welchem Servant implementiert wird
- Oft mit "Location Forwarding Service" kombiniert
  - ◆ Dynamische Erzeugung von Server-Prozessen
  - ◆ Aufrufweiterleitung zu den dynamisch gestarteten Servern



7 ORB-Schnittstelle

- Export von initialen Objektreferenzen (ORB, OAs, Naming Service, ...)
- Manipulation von Objektreferenzen (Umwandlung in Strings und zurück)



8 Zusammenfassung Server

- Servants implementieren CORBA-Objekte.
- Server-Prozess verwaltet Servants.
- Skeletons leiten Aufrufe an Servants weiter.
- Object Adaptor verwaltet Lebenszyklus von Servants.
- Implementation Repository enthält Informationen über aktive Server und deren Servants.

C.3 Einfacher Java-Server (POA)

- Verwendung des "Portable Object Adaptor" (POA)
- Erzeugen und Aktivieren von Servants

1 Servant

- Objekt, das die Funktionalität das CORBA-Objekts implementiert
- Semantik:
  - Jede Erzeugung eines Servant erzeugt ein zugeordnetes CORBA-Objekt.
  - Nach Zerstörung des Servant existiert das zugeordnete CORBA-Objekt nicht mehr.
- IDL:

```
module PortableServer {                        // PIDL
    native Servant;
};
```

- Java-Abbildung in eine abstrakte Klasse:

```
package org.omg.PortableServer;

public abstract class Servant {
    ...
};
```


2 Skeleton

- Basisklasse für die Implementierung des Servants
- IDL:

```
module module {
    interface name { ... };
};
```

- Java-Abbildung in eine abstrakte Klasse *name*`POA`:

```
package module;

public abstract class namePOA
        extends org.omg.PortableServer.Servant
        implements org.omg.CORBA.portable.InvokeHandler,
            nameOperations
{
    public org.omg.CORBA.portable.OutputStream _invoke(
        String op, org.omg.CORBA.portable.InputStream i,
        org.omg.CORBA.portable.ResponseHandler handler)
    { ... }
    public name _this( org.omg.CORBA.ORB orb ) { ... }
    ...
}
```


2 Skeleton

- `OutputStream _invoke( String op, InputStream i, ... )`
  - Unmarshalling der Paramter von `InputStream`
  - Weiterleitung an die Operation namens `op`
  - Marshalling der Rückgabewerte nach `OutputStream`
- *name* `_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.
- Skeleton wird automatisch durch den IDL-Compiler erzeugt, z.B. mit

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

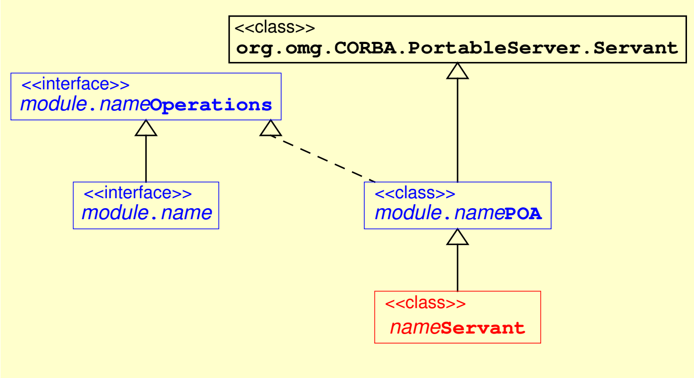

3 Implementierung Servant

- Eigene Implementierung der Objektfunktionalität
- Erbt von der Skeleton-Klasse
- Namenskonvention: *name*`Servant`

```
public class nameServant extends module.namePOA {
    Implementation of methods for attributes and operations
}
```

4 Hierarchie der Servant-Implementierung

- Klassenhierarchie für das IDL-Interface *module::name*



4 Hierarchie der Servant-Implementierung

- Beispiel: Erzeugte Schnittstellen und Klassen

```
public interface AccountOperations {
    public void withdraw( double amount )
        throws AccountPackage.Overdraft;
}
public interface Account extends org.omg.CORBA.Object,
    AccountOperations, ...
{}

public abstract class AccountPOA
    extends org.omg.PortableServer.Servant
    implements AccountOperations, ...
{ ... }
```

- Beispiel: Servant

```
public class AccountServant extends AccountPOA {
    public void withdraw( double amount )
        throws AccountPackage.Overdraft {
        // Implementation of withdraw
    }
}
```


5 Was fehlt noch?

- ORB-Initialisierung (siehe Klient)
- POA-Aktivierung
  - Referenz auf RootPOA via `resolve_initial_references`

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

  - "Narrow" auf das 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();
```


6 "Hello World"-Server - Interface

- IDL-Interface: Hello.idl

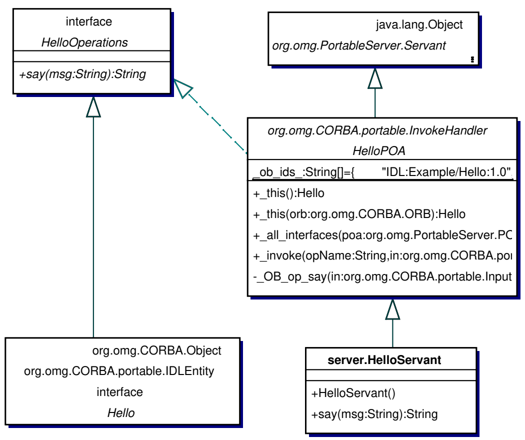
```
module Beispiel {
    interface Hello {
        string say( in string msg );
    };
};
```

6 "Hello World"-Server - Implementierung

- Servant-Implementierung mittels Vererbung

```
// server/HelloServant.java

import generated.Example.*;

public class HelloServant extends HelloPOA {

    // Constructor
    public HelloServant() {
        super();
    }

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


6 "Hello World"-Server - Klassen



6 "Hello World"-Server - Startup

```
// server/HelloServer.java
import generated.Example.*;
import org.omg.CORBA.*;
import java.io.*;
import org.omg.PortableServer.*;

public class HelloServer {
    public static void main( String[] args ) {
        try {
            // Initialize ORB
            ORB orb = ORB.init( args, null );

            // Get the RootPOA
            POA poa = POAHelper.narrow(
                orb.resolve_initial_references( "RootPOA" ));

            // Activate the RootPOA
            poa.the_POAManager().activate();

            // Create the hello servant object
            HelloServant hServ = new HelloServant();

            // Activate the object
            Hello h = hServ._this( orb );

                                        // to be continued ...
```


6 "Hello World"-Server - Startup (2)

```
            // Write reference
            PrintWriter pw = new PrintWriter(new FileWriter(
                        "/proj/i4oovs/pub/hello/Hello.ior"));
            pw.println( orb.object_to_string( h ) );
            pw.close();

            // Wait for request
            orb.run();

        } catch( org.omg.CORBA.SystemException e ) {
            //...
        } catch( Throwable t ) {
            //...
        }
    }
}
```

7 Zusammenfassung

- Initialisierung des ORB
- Aktivierung des POA
- Instantiierung von Servant(s)
- Aktivierung von Servant(s)
- Registrierung von Servant(s) beim Name Service
- Starten der ORB-Hauptschleife