Parallele und Verteilte Systeme

Einführung in die Netzwerk Programmierung mit Java :

JavaIDL und CORBA Einführung

Zeitlicher Ablauf

- **Verteilte Applikationen**
  - 3-Tier zu Multi-Tier
- **CORBA Ideen**
- **JavaIDL**
- **Die “Hello World” Applikation in CORBA**
  - IDL
  - Client
  - Server
  - Exceptions
  - Initialisierung
  - Naming Services

Lernziele

- Sie kennen die grundlegenden Eigenschaften von IDL und können in IDL einfache Objekte beschreiben
- Sie kennen den grundsätzlichen Aufbau von CORBA
- Sie wissen wie der Kommunikationsablauf mit Hilfe von ORB Objekten und IIOP aussieht
- Sie kennen ein Anwendungsbeispiel

Grundsätzliches

- **Multi-Tiered Applikationen**
  - User-Interface-Tier
  - Service/Server-Tier
  - Data Store (Database) Tier

Traditionelle Systeme

Traditionelle Applikationen entstanden in der Regel monolithisch, aus einem Guss.

Nachteile:

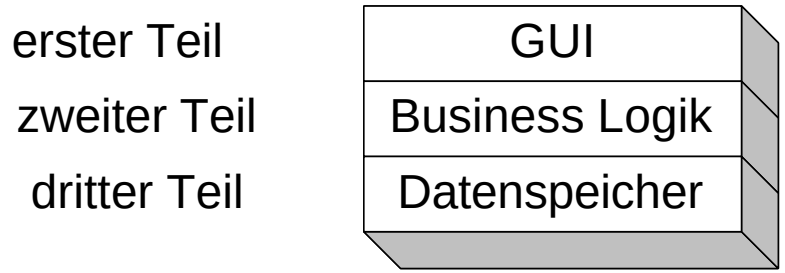
- schwierig herzustellen
- hohe Komplexität
- hohe Wartungskosten

Multi-Tiered Applikationen : erste Generation

- **3-Tiered:**
  - GUI
  - Business Logik
  - Datenspeicher

erste Generation

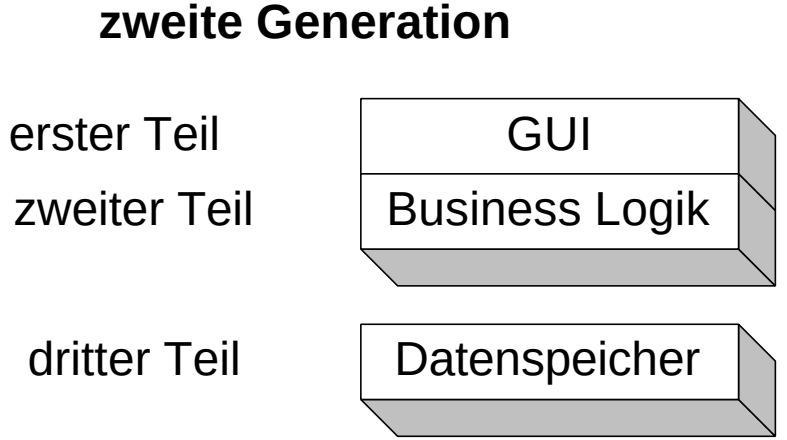
enge Kopplung :



Multi-Tiered Applikationen : zweite Generation

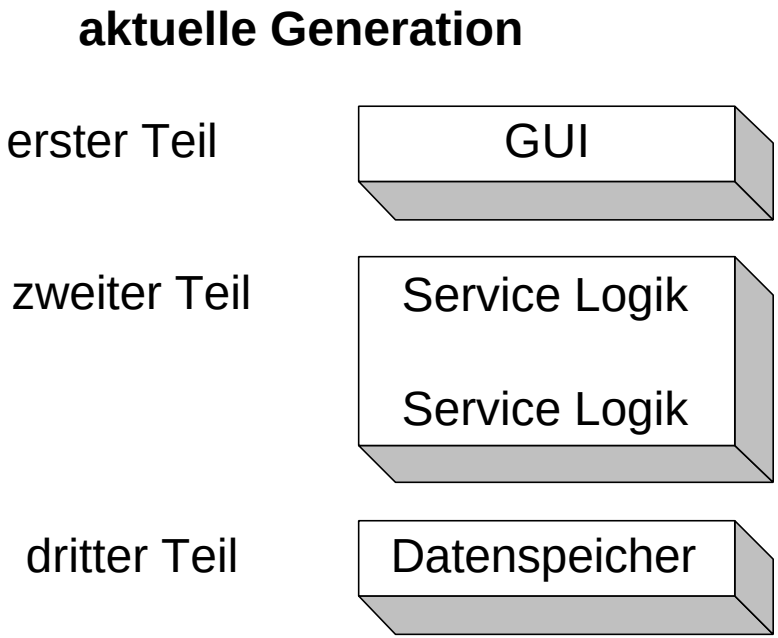
- **3-Tiered:**
  - GUI
  - Business Logik
  - Datenspeicher

Trennung der Daten
von den Programmen



Multi-Tiered Applikationen : neuste Generation

- **3-Tiered:**
  - GUI
  - Business Logik
  - Datenspeicher



User Interface Layer

- **Fokus auf**
  - effizienter Benutzerführung
  - einheitliches Design
- **hoher Entwicklungsaufwand**
- **auf dem Benutzer-Desktop**
- **mit Hilfe eines Interface-Layers können unterschiedliche Server / Services aktiviert werden**

- **verwendet in der Regel Methoden des Business-Logik Layers und stellt die Schnittstelle zu diesen dar**

Service / Server Layer

- **der Server Layer besteht aus Business Objekten, mit denen der Client arbeiten möchte / muss**
- **die einzelnen Business Objekte können im Falle von CORBA miteinander kommunizieren**
- **Beispielfunktionen:**
  - Lagerverwaltung
  - Debitoren
  - ...

Daten(bank)-Layer

- **der Daten(bank)-Layer besteht aus Objekten, die den Zugang zu den Datenbanken und Files kapseln**
- **Beispiel:**
  - lies_Umsatz(..):
    - liefert zum Beispiel den Umsatz mit einem Kunden und einem Produkt

Diese Methode kann mit Hilfe eines SQL Statements implementiert sein

zum Beispiel mit Hilfe eines Select Statements / Abfrage

CORBA

- **Common Object Request Broker =**
  Standard Objekt Architektur

- **von der OMG (Object Management Group) 1989 definiert, als offener Software Bus, zur Kommunikation heterogener Objekte, unterschiedlicher Hersteller**

- **CORBA erlaubt es, heterogen implementierte Objekte zu verteilen und miteinander kooperieren zu lassen, sofern das Objekt mit Hilfe von IDL beschrieben wurde**

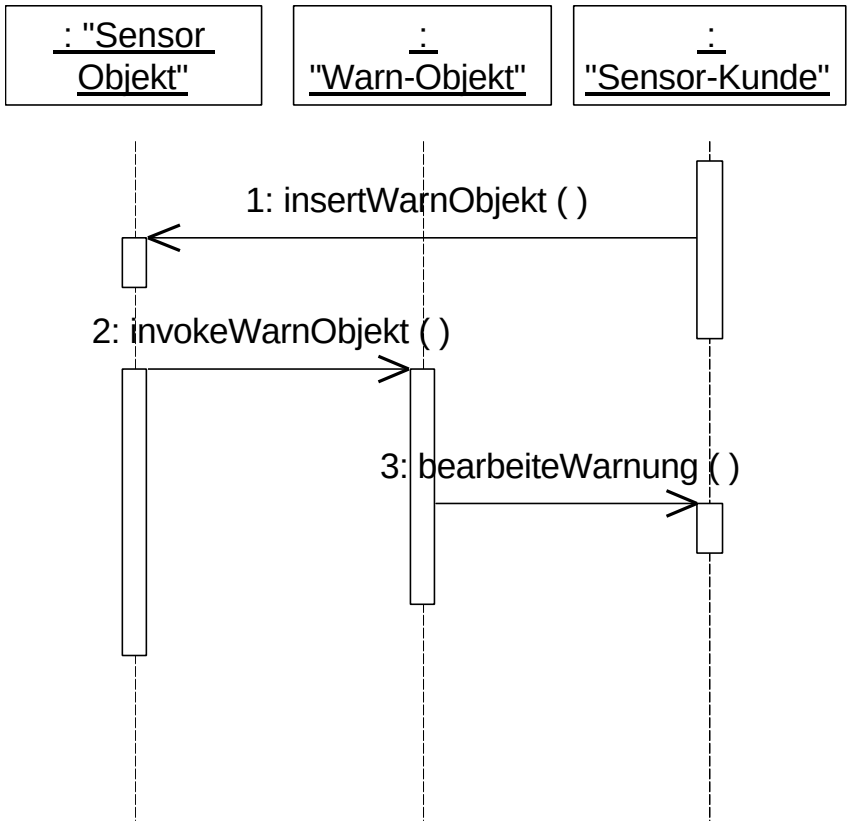
Verteilte CORBA Objekte

- **CORBA Objekte können mit anderen CORBA Objekte kommunizieren, ohne zu wissen wo die sich befinden und wie diese implementiert sind**
  - CORBA Objekte können sich irgendwo im Netzwerk befinden
  - CORBA Objekte können mit Objekten auf anderen Plattformen kommunizieren
  - CORBA Objekte können in unterschiedlichen Programmiersprachen implementiert werden

CORBA Clients und Server

- **ein CORBA *Client* ist ein Programm oder Objekt, welches eine Methode eines CORBA Objektes verwendet**
- **ein CORBA *Server* ist ein Programm, welches Dienste und Dienstobjekte implementiert**
- **ein Server Objekt kann aus anderer Sicht ein Client Objekt sein und umgekehrt**
- **gemäss OMG :**
  - ein Server Objekt ist eine Objekt Implementation

CORBA Client und Server : ein Beispiel



Observer Design Pattern (mod):

Die Rolle “Client” und
die Rolle “Server”
sind nicht fix für ein Objekt

In einer bestimmten Situation
ist ein Objekt CLIENT;
in einer anderen Situation
ist das Gleiche Objekt SERVER

OMG IDL (Interface Definition Language)

- **unabhängig vom Betriebsystem**
- **unabhängig von Programmsprachen**

  SCHNITTSTELLE zu ALLEN Services und Komponenten, die am CORBA (Software) Bus hängen

- **IDL ist DEKLARATIV und zeigt keinerlei Implementierungs-Details**
- **IDLtoJava Compiler generiert :**
  - Java Interface Definitionen
  - Java **Stubs** (Clientseitige Schnittstelle) und **Skeletons** (Serverseitig)
- **Mapping Java zu/von IDL gemäss OMG**

Object Request Broker (ORB)

- **ORB ist der Objekt Bus, mit dessen Hilfe Objekte:**
  - Requests senden und Meldungen empfangen können von Objekten, die lokal oder remote sein können
- **der Client weiss nicht, *wie* mit den verschiedenen Objekten kommuniziert wird**
- **man hat die Möglichkeit Objekte während der *Ausführzeit* zu bestimmen und deren Services zu nutzen (dynamic invocation interface : DII)**
- **ORBs sind der gemeinsame Nenner der unterschiedlich implementierten, an verschiedenen Stellen sich befindenden, in verschiedenen Programmiersprachen realisierten Objekten**

ORBs

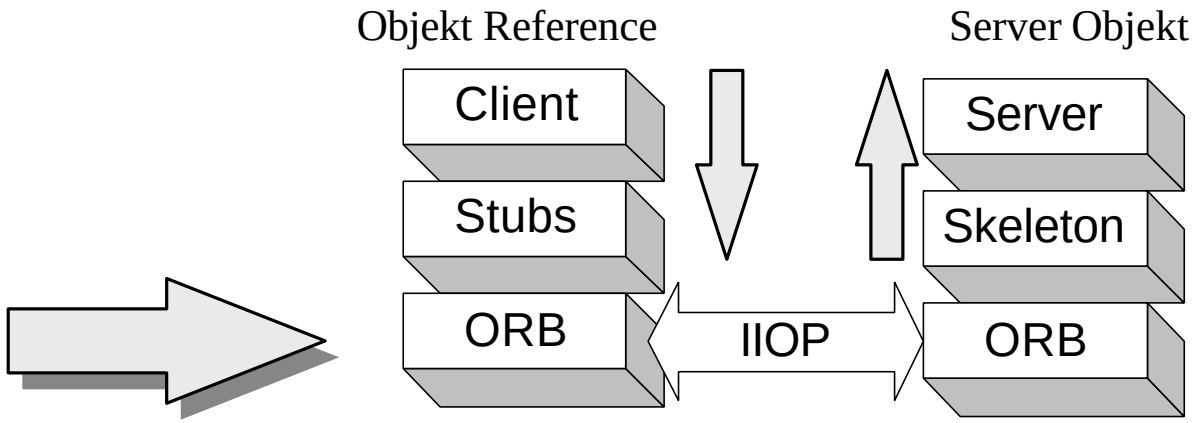
- **ORBs können miteinander kommunizieren, Objekt-Referenzen bilden und interpretieren und Parameter in verschiedene Formate umwandeln, so dass sie vom IIOP (Internet InterORB Protocol) verwendet / interpretiert werden können**
- **ORBs sind oft in Form von Bibliotheken implementiert**
  - diese werden dann mit den Clients und Servers gelinked

- **Optional kann ein ORB mit Hilfe eines ORB Daemons automatisch gestartet werden.**

  Dieser Mechanismus wurde bisher in Java nicht implementiert

CORBA und Java

- ***Java liefert als Standard-Bibliothek einen ORB***
- **dieser kommuniziert mit andern ORBs mit Hilfe des IIOP**
  IIOP = Internet InterORB Protocol



Internet InterORB Protocol (IIOP)

- **CORBA Implementationen kommunizieren (im Falle von Java ausschliesslich) mit Hilfe des IIOPs**
- **JavaIDL basiert auf einem portablen ORB, der offen ist für eventuell andere Kommunikations-Protokolle (Weiterentwicklungen von IIOP, ...)**
- **IDLtoJava Compiler generiert ORB unabhängige Schnittstellen, die sowohl von Applikationen, aber auch von Applets verwendet werden können**

Objekt Referenz

- ***Objekt Referenzen*** **in CORBA entsprechen dem “object pointer” in C++**
- **die Objekt-Referenz liefert die Informationen, die benötigt werden, um ein Objekt in einem ORB eindeutig zu identifizieren.**
- **Client und Server kennen nur ein “Bild” der Objektreferenz (mit Hilfe des Java Mappings); Details der Implementation sind ja unbekannt**
- **eine spezielle Referenz ist die “Null” Referenz, die in Java auf Java null abgebildet wird. null wird immer dann verwendet, wenn man eine sicher noch nicht verwendete Referenz benötigt, die auf kein Objekt weist**

Client Stubs

- **pro IDL Interface wird ein Java Interface für den Client *Stub* generiert**
- **der Stub ist ein lokaler Proxy, der mit dem Server Objekt kommuniziert, mit Hilfe des ORBs**
- **OMG IDL Spezifikation der Objekt Methoden sind fast ideale Vorgaben für den Programmierer, für die Implementation des Clients und / oder des Servers**

Server Skeletons

- ***Skeletons* sind die Server-seitigen Schnittstellen (analog zu den Stubs der Clients)**
- **Skeletons werden vom IDLtoJava Compiler generiert**
- **der ORB verwendet die Skeletons, um mit den Dienst-Objekten zu kommunizieren**

CORBA Interface Repository

- **der *Interface Repository* ist ein Dienst, mit dessen Hilfe Objekte zur Verfügung gestellt werden (die in der IDL spezifizierten Objekte)**

- **JavaIDL implementiert *kein* Interface Repository**
  - falls zur Ausführzeit ein Repository benötigt wird, dann

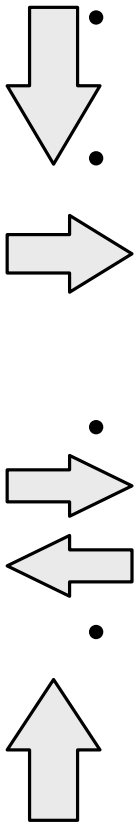
    wird vorausgesetzt, dass ein weiterer ORB zur Verfügung steht (zum Beispiel das kommerzielle Produkt ORBIX, oder Visigenics)

    der dann die benötigten, in JavaIDL noch nicht implementierten Services zur Verfügung stellt

CORBA Implementation Repository

- **der *Implementation Repository* enthält Informationen, mit deren Hilfe der ORB die Implementierung der Objekte finden und aktivieren kann**
- **die meiste Information im Implementation Repository wird ORB- und Betriebssystem-spezifisch implementiert**
- **im Implementation Repository werden auch zusätzliche Informationen abgespeichert, die zum Beispiel**
  - zum Debuggen verwendet werden,
  - Zugriffskontrollen ausführen
  - Resourcen zuteilen und verwalten
- **ein Implementation Repository ist nicht unbedingt nötig, erhöht aber die Flexibilität**
- **JavaIDL implementiert auch den Implementation Repository NICHT**

CORBA IIOP Konzept



- **Falls das Server Objekt remote ist, dann zeigt die Objektreferenz auf eine Stub-Funktion**
- **die Stub-Funktion verwendet die ORB Mechanismen, um festzustellen, auf welcher Maschinen das Server Objekt ist und bittet den ORB des Servers, eine Kommunikation zu starten**
- **sobald die Verbindung hergestellt ist, dann sendet der Stub die Objekt Referenz und die Parameter an den Skeleton des Servers**
- **der Skeleton Code wandelt die Parameter so um, dass sie vom Server Objekt interpretiert werden können**

CORBA IIOP Eigenschaften

- **der Client weiss nicht, wo das Server Objekt sich befindet**
- **der Client kennt keine Details über den Server**
- **der Client weiss nicht welcher ORB benutzt wird, um die Kommunikation zu ermöglichen**
- **der Client kann nur Objekte verwenden, deren Schnittstellen spezifiziert wurden (in IDL)**

- **das Server Objekt kann in einer andern Programmiersprache implementiert sein als der Client**
- **die verschiedenen ORBs kommunizieren mit Hilfe des IIOP Protokolls**

CORBA IIOP und IDL

- **die Objekte müssen mit Hilfe der IDL, der Interface Definition Language der OMG spezifiziert werden**
- **ein Interface beschreibt:**
  - den Objekt Typus
  - Methoden und Parameter
  - Exceptions / Ausnahmezustände

- **Die IDL Spezifikationen werden mit Hilfe des IDLtoJava Compilers in Java umgesetzt**

- **die Umsetzung geschieht gemäss dem von der OMG festgelegten Sprachmapping**

CORBA IIOP IDLtoJava

- **Stubs und Skeleton Files werden vom idltojava Compiler generiert**
- **der idltojava Compiler ist (heute) frei verfügbar und vom SunSite downloadbar**
- **neuere Versionen von JDK enthalten bereits einen idlj Compiler**

Definition / Implementation von CORBA Objekten ein IDL Beispiel

```
module HelloApp
   {
        interface Hello
        {
                string sayHello();
        };
   };
```

idltojava Hello.IDL

- **liefert im Unterverzeichnis HelloApp:**
  - Directory of C:\\TEST\HELLOAPP
  - <DIR> .
  - <DIR> ..
  - 260 Hello.java
  - 1'821 HelloHelper.java
  - 813 HelloHolder.java
  - 1'495 _HelloImplBase.java
  - 996 _HelloStub.java

Beispiel : Hello.java

```
/*
 * File: ./HELLOAPP/HELLO.JAVA
 * From: HELLO.IDL
 * Date: Mon Apr 06 18:56:08 1998
 *   By: IDLTOJAVA Java IDL 1.2 Nov 10 1997 13:52:11
 */
package HelloApp;
public interface Hello
    extends org.omg.CORBA.Object {
    String sayHello()
;
}
```

Interpretation ...

- **wir werden uns im Detail mit diesen Files befassen, nachdem wir etwas mehr Theorie behandelt haben**
- **ACHTUNG:**
  - die Spezifikation von JavaIDL ist recht neu und **NOCH NICHT** durch die OMG abgesegnet;es können sich also noch Änderungen ergeben
  - die Spezifikation von JavaIDL unterscheidet sich zum Teil von der Spezifikation der OMG IDL Sprache; es wäre wohl gescheiter gewesen, JavaIDL NICHT den Zusatz IDL zu geben, um Missverständnisse zu vermeiden

IDL Bausteine

- **Modules**
- **Schnittstellen / Interfaces**
- **Interface Vererbung**
- **Variablen und Strukturen**
- **Methoden**
- **konstruierte Datentypen**
- **Ausnahmen / Exceptions**

Schauen wir uns die Begriffe im Einzelnen an
Später werden wir am konkreten Beispiel genauer sehen, wie die Konzepte zu interpretieren sind

Modules

- **ein Module fasst verschiedene *Interfaces* zusammen**
- **Beispiel**

```
module Fernbedienung
    {
      interface Fernseher
      {
      // hier folgt die Definition des Fernseher Interfaces
      // d.h. der Funktionen, die fernbedient werden können
      }
      interface Radio
      {
      // hier folgt die Definition des Radio Interfaces
      }
    }
```

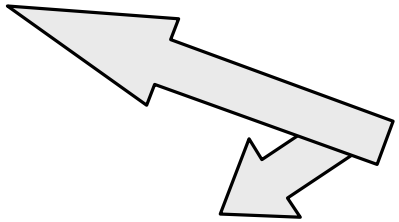
Interfaces

- **Basis der IDL Beschreibung**
- **im Gegensatz zu OO Programmiersprachen, in denen man das Interface UND die Implementation beschreibt, wird in IDL NUR die Signatur der Methode und die Variablen beschrieben, die zum Interface gehören**
- **Kommentare sind erlaubt**
- **Beispiel:**
  - interface TVSet
    {long aktuellerSender();
     // folgt später void wechsleSender(long neuerSender);
      void erhoeheLautstaerke();
      void reduziereLautstaerke();
    }

Interface Vererbung

- **wie in jeder echten OO Sprache, kann man auch in IDL Interfaces schrittweise aufbauen, mit Hilfe der Vererbung**
- **Beispiel:**

```
interface Fernbedienung
    {
    ...
    }
    interface ErweiterteFernbedienung : Fernbedienung
    {
    void aktiviereMethodeA();
    ...
    }
```



Variablen und Strukturen

- **die Standard-Datentypen orientieren sich an C++**
  - long : Integer zwischen -2**31 und 2**31
  - short : Integer zwischen -2**15 und 2**15
  - float: IEEE single-precision
  - double: IEEE double-precision
  - char: 8 Bit Zeichen
  - boolean: TRUE, FALSE
  - string: Zeichenkette
- **Strukturen (werden in Java als Klassen dargestellt)**
  - Beispiel
    - struct TestStruktur
      { string datum;
      string zeit;
      }

Methoden

- **Methoden müssen ebenfalls in IDL spezifiziert werden**
- **Parameter sind von einem von drei Typen:**
  - in
    - nur Eingabe
  - out
    - nur Ausgabe
  - inout
    - Ein- und Ausgabe
- **Parameter sind von einem bestimmten Typus**
  - der spezielle Typus *ANY* wird auf *Object* abgebildet
  - auf das konkrete Mapping kommen wir noch zurück

Constructed Data Types

- **Datentypen können analog wie in Pascal definiert werden**
- **Beispiel:**
  - module fernbedienung
    {
      interface fernsehSet
    {
    ...
    }
    interface radioSet
    {
    typedef enum _Baender (KW_BAND, LW_BAND, UKW_BAND) Band;
      Band aktuellesBand;
    ...
    }

Exceptions

- **Exceptions in IDL funktionieren analog zu den Java Exceptions**
- **Beispiel:**

```
interface Apfel
    {
    exception verfault { };

    void esseApfel() throws verfault;
    }
```

Abbildung von IDL auf Java

- **wir möchten uns an Hand der IDL Skizzen /Beispiele darüber informieren, wie das Mapping konkret funktioniert**
- **ein universelles Mapping ist äusserst schwierig:**
  - wie soll das (komplexe) Memory Management von C++ und der Garbage Collector von Java in einer verteilten Umgebung vereinheitlicht werden?

Module

- **IDL Module:**

  module bedienungsgeraet
  {
  ...
  }

- **Java:**

  package bedienungsgeraet;
  public class
  {
  ...
  }

Interfaces

- **IDL:**

```
module bedienung
    {
      interface fernseher
    {...}
    }
```

- **Java:**

```
package bedienung;
    public class fernseher
    {...}
```

Holders

- **IDL wurde wesentlich mit C++ im Hinterkopf entwickelt**
- **in Java gibt es aber KEIN pass-by-reference, da es keine Pointer gibt**
- ***out* und *inout* Parameter werden mit Hilfe von sogn. *HOLDERs* implementiert**
- **ein Holder besteht aus den Variablen PLUS Methoden, mit denen auf die Variablen zugegriffen werden kann**

  wir werden später sehen, wo und wie die Holder in speziellen Files “versteckt” werden

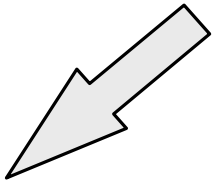
Interface Vererbung : IDL Teil

- **IDL:**

```
module bedienungsgeraet
    {
      interface lautsprecher
      {...}
      interface mikrofon
      {...}
    ...
      interface telefon : lautsprecher, mikrofon
      {...}
```

Interface Vererbung : Java Teil

- **Java:**



```
package bedienungsgeraet
    {public interface lautsprecherRef
      {...};
      public interface mikrofonRef
      {...};
      ...
      public interface telefonRef extends bediengeraet.lautsprecherRef,
      bediengeraet.mikrofonRef
      {...}
   ...
      public class microfon
      {...}
    }
```

Variabeln und Structure

- **long Java int**
- **short Java short**
- **float Java float**
- **...**
- **string Java String class (java.lang.String)**

- **Structure**

IDL

```
struct telefonNummer
{   string laenderCode;
    string gebietsCode;
    string telNummer;}
```

Java

```
public class telefonNummer
{public String laenderCode;
 public String gebietsCode;
 public String telNummer; }
```

CORBA Review für JavaIDL

- **CORBA Invokations Modell**
- **Struktur des Clients**
- **Struktur einer Objekt-Implementation**
- **wie komme ich zu den Objekt-Referenzen?**
- **Typenumwandlung**
- **kreieren und löschen von CORBA Objekten**