# Parallele und Verteilte Systeme

## Einführung in die Netzwerk Programmierung mit Java :

## JavaIDL Hello World

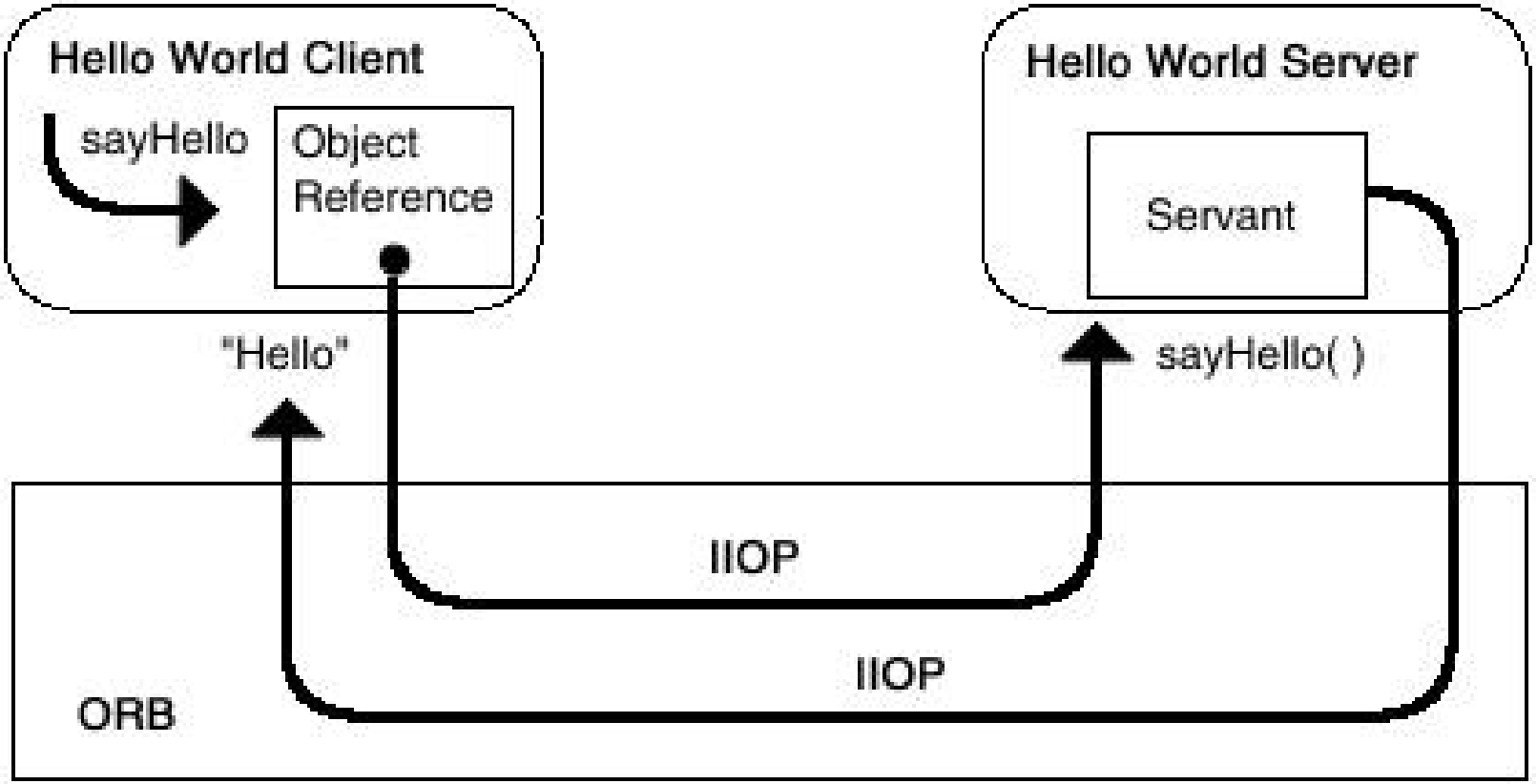
# Zeitlicher Ablauf

- **Hello World Übersicht**
- **IDL Interface Beschreibung**
- **Der Client**
- **(Der Client als Applet)**
- **Der Server**
- **Die Objekt Referenz**

# Lernziele

- Sie lernen an Hand eines bekannten Beispiels die Strukturen und “das Handwerk” der IDL und ORB Programmierung kennen

# Übersicht


Hello World Client
sayHello
Object Reference
Hello World Server
Servant
"Hello"
sayHello( )
IIOP
IIOP
ORB

# Um was geht es? Was macht das ORB “Hello”?

- **Der Client / das Applet ruft die Methode `sayHello()` des Hello Servers auf**
- **Der ORB übermittelt den Aufruf and das Server-Objekt**
  **Das Server-Objekt muss mit IDL beschrieben und registriert sein**
- **Die Server-Objekt führt seine Methode `sayHello()` aus**
  **und liefert einen Java `String`**
- **Der ORB transferiert den `String` zurück zum Client**
- **Der Client druckt bezw zeigt den Wert der `String` Variable**

**EINSCHRÄNKUNGEN:**

- **wir beschränken uns zuerst auf statische Aufrufe**

# Die Mechanik : was ist zu tun (lokaler Test)

- **beschreiben der Schnittstelle mit Hilfe von IDL**
  - Hello.IDL
- **übersetzen : IDL in Java**
  - idltojava Hello.IDL
    es wird ein Unterverzeichnis HelloApp angelegt; in ihm sind Stub und weitere Java Files
- **übersetzen der generierten Files**
  - javac *.java HelloApp*.java
- **starten des NameServers in einem DOS Fenster**
- **starten des HelloServers in einem zweiten DOS Fenster**
- **starten des HelloClient in einem dritten DOS Fenster**

# Beschreibung der Schnittstelle mit Hilfe von IDL

- **die einzelnen Schritte:**
  - schreiben von Hello.IDL
  - übersetzen / mappen von IDL nach Java
- **Interpretation des Ergebnisses**

# Hello.IDL : Module Beschreibung

```
module HelloApp
{
    //hier folgt die Interface Beschreibung
};
```

# Hello.IDL : Interface Beschreibung

```
module HelloApp
{
    interface Hello
    {
        // hier folgt die Beschreibung der
        // Methoden
    };
};
```

# Hello.IDL : Beschreibung der Methoden

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

# Mapping idltojava : das Hello.Java File

```
/* File: ./HELLOAPP/HELLO.JAVA
 * From: HELLO.IDL
 * Date: Wed Oct 29 14:29:04 1997
 * By: idltojava Java IDL 1.2 Nov 10 1997 13:52:11
 */
package HelloApp;                       // IDL module
public interface Hello                  // IDL interface
    extends org.omg.CORBA.Object {// CORBA Anbindung
    String sayHello()                   // IDL Methode
;
}
```

# Interpretation der idltojava Ausgabe

**_HelloImplBase.java**

- abstract class : **Server Skeleton**

**_HelloStub.java**

- public class : **Client Stub**

**Hello.java**

- enthält die Java Version der IDL Beschreibung
  package HelloApp

**HelloHelper.java**

- public class : enthält Hilfsmethoden für die CORBA Kommunikation
  (z.B. narrow(), insert(), extract())

**HelloHolder.java**

- final class : enthält die Instanzierung und implementiert die IDL `out` und `inout` Argumente

# Entwicklung der Client Applikation : Überblick

- **Basis schaffen : `import Packages`**
- **Kreieren eines ORB Objektes**
- **Finden des HelloServers**
- **Aufruf der `sayHello()`Methode**

# Definition der Client Klasse : Basis

- **File**

  HelloClient.java

- **Import der Packages**

```
import HelloApp.*                // Client Stubs
import org.omg.CosNaming.*       // Naming Service
import org.omg.CORBA.*           // CORBA Grundklassen
```

# Definition der Client Klasse : Grundstruktur

- **Client Klassendeklaration**

```
public class HelloClient
{
// hier folgt main()...
}
```

# Definition der Client Klasse : main() Methode

- **Client Klassendeklaration**

```
public class HelloClient
{
public static void main(String args[])
    {
    // try-catch Blöcke
    }
}
```

# Definition der Client Klasse : CORBA Exceptions

- **Client Klassendeklaration**

```
public class HelloClient
{
public static void main(String args[])
    {
    try {
    // Hello Code
    }catch (Exception e) {
      System.out.println("Error : "+e);
      e.printStackTrace(System.out);
    }
}
```

# Kreieren des ORB Objektes

- **Client Klassendeklaration**

```
public class HelloClient
{
public static void main(String args[])
    {
    try {
    ORB orb = ORB.init(args, null);  // siehe nächste Seite
    }catch (Exception e) {
      System.out.println("Error : "+e);
      e.printStackTrace(System.out);
    }
}
```

# Kreieren des ORB Objektes : System Properties

- **org.omg.CORBA.ORBClass**
  **implementiert das org.omg.CORBA.ORB Interface:**
  com.sun.CORBA.iiop.ORB.
- **org.omg.CORBA.ORBSingletonClass**
  **implementiert das org.omg.CORBA.ORB Interface. In der Regel für Applets und secure applications**

- **org.omg.CORBA.ORBInitialHost**
  - Host Name der Server-Maschine (Name Services und weitere Dienste)
- **org.omg.CORBA.ORBInitialPort**
  - Port, über den der Name Server versucht zu kommunizieren

# Suchen des HelloServers : die Schritte

- **Bestimmen des Naming Contexts**
- **Eingrenzen der Objekt Referenzen**
- **Auffinden des Hello Services**

# Suchen des HelloServers : bestimmen des Contextes

```
public class HelloClient
    {
    public static void main(String args[])
        {
        try {
        ORB orb = ORB.init(args, null);
        org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references("NameService");
        }catch (Exception e) {
          System.out.println("Error : "+e);
          e.printStackTrace(System.out);
        }
    }
```

# Suchen des HelloServers : einschränken der ObjektRefs

```
public class HelloClient
{
public static void main(String args[])
    {
    try {
    ORB orb = ORB.init(args, null);
    org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
    NamingContext ncRef = NamingContextHelper.narrow(objRef);


      }catch (Exception e) {
                System.out.println("Error : "+e);e.printStackTrace(System.out);
      }
    }
```

# Suchen des HelloServers : finden des Hello Services

```
try {
    ORB orb = ORB.init(args, null);
    org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
    NamingContext ncRef = NamingContextHelper.narrow(objRef);
    NameComponent nc = new NameComponent("Hello", "");
    NameComponent path[] = {nc};
    Hello helloRef =
    HelloHelper.narrow(ncRef.resolve(path));
      }
```

# Suchen des HelloServers : finden des Hello Services

- CORBA ist sehr universell aufgebaut, insbesondere sind verschiedene Name Services möglich
- der Name Service gemäss OMG (hier der COS = Common Object Service Name Service) ist aufgebaut wie ein Verzeichnis eines Filesystems:
  - jedes Objekt wird in einen Naming Context (entspricht dem Directory) eingetragen
  - jeder Knoten besteht aus zwei Teilen
    - der Objekt Referenz und dem Objekt Namen
  - gesucht wird in der Regel über den Namen
  - der Context geht verloren, sobald der Server “stirbt”
    - in der Java Implementation des ORBs gibt es explizit keine persitenten Objekte
    - wenn der Benutzer den Zustand des Objektes sichern will, dann muss er entweder diese Funktionalität bei einem kommerziellen ORB anfordern und nutzen oder die Funktionalität selber implementieren

# Suchen des HelloServers : die sayHello Methode

```
public class HelloClient
    {
    public static void main(String args[])
        {
        try {
        ...
        String Hello = helloRef.sayHello();
        System.out.println(Hello);
        ...
        }catch (Exception e) {
          System.out.println("Error : "+e);
          e.printStackTrace(System.out);
        }
    }
```

# Entwicklung des HelloWorld Servers

- **analog zum Client:**
  - schaffen der Basis
  - kreieren des ORB Objektes
  - verwalten des “Dieners” (aktive Server Teil)
  - einbinden der COS Name Services
  - warten auf den Kunden

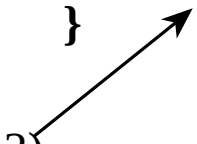
# Server : Basis

```
// stubs Teil einbinden
import HelloApp.*;
// naming services
import org.omg.CosNaming.*;
// exception handling
import org.omg.CosNaming.NamingContextPackage.*;
// CORBA root
import org.omg.CORBA.*;
```

# Server : Rahmen und main()

1)

```
public class HelloServer
{
// hier folgt der echte Server Code
}
```



2)

```
public class HelloServer
{
public static void main(String args[])
{
// try - catch Teil
}
```

# Server : Exceptions

```
try
    {
    // hier folgt der eigentliche Server Code
    }
catch (Exception e)
    {
    System.err.println("Error : "+e);
    e.printStackTrace(System.out);
    }
```

# Server : ORB Objekt initialisieren und instanzieren

im try Block::

```
// kreieren
ORB orb = ORB.init(args, null);
// instatieren
HelloServant helloRef = new HelloServant();
// anbinden an den ORB
orb.connect(helloRef);
```

# Server : die Servant Klasse

```
class HelloServant extends _HelloImplBase
    {
    // sayHello Methode
    public String sayHello()
    {
      return “\nHello world!!\n”;
    }
```

# Server : Name Services

```
im try catch Block
// Naming Context
org.omg.CORBA.Object objRef =
    obj.resolve_initial_references(“NameService”);
// umwandeln der Objekt Referenz
NamingContext ncRef = NamingContextHelper.narrow(objRef);
// registrieren des Servant beim Name Service
NameComponent nc = new NameComponent(“Hello”, ““);
NameComponent path[] ={nc};
ncRef.rebind(path, helloRef);
```

# Server : Warteschleife des Servers

```
java.lang.Object sync = new java.lang.Object();
synchronized(sync)
{
    sync.wait();
}
```