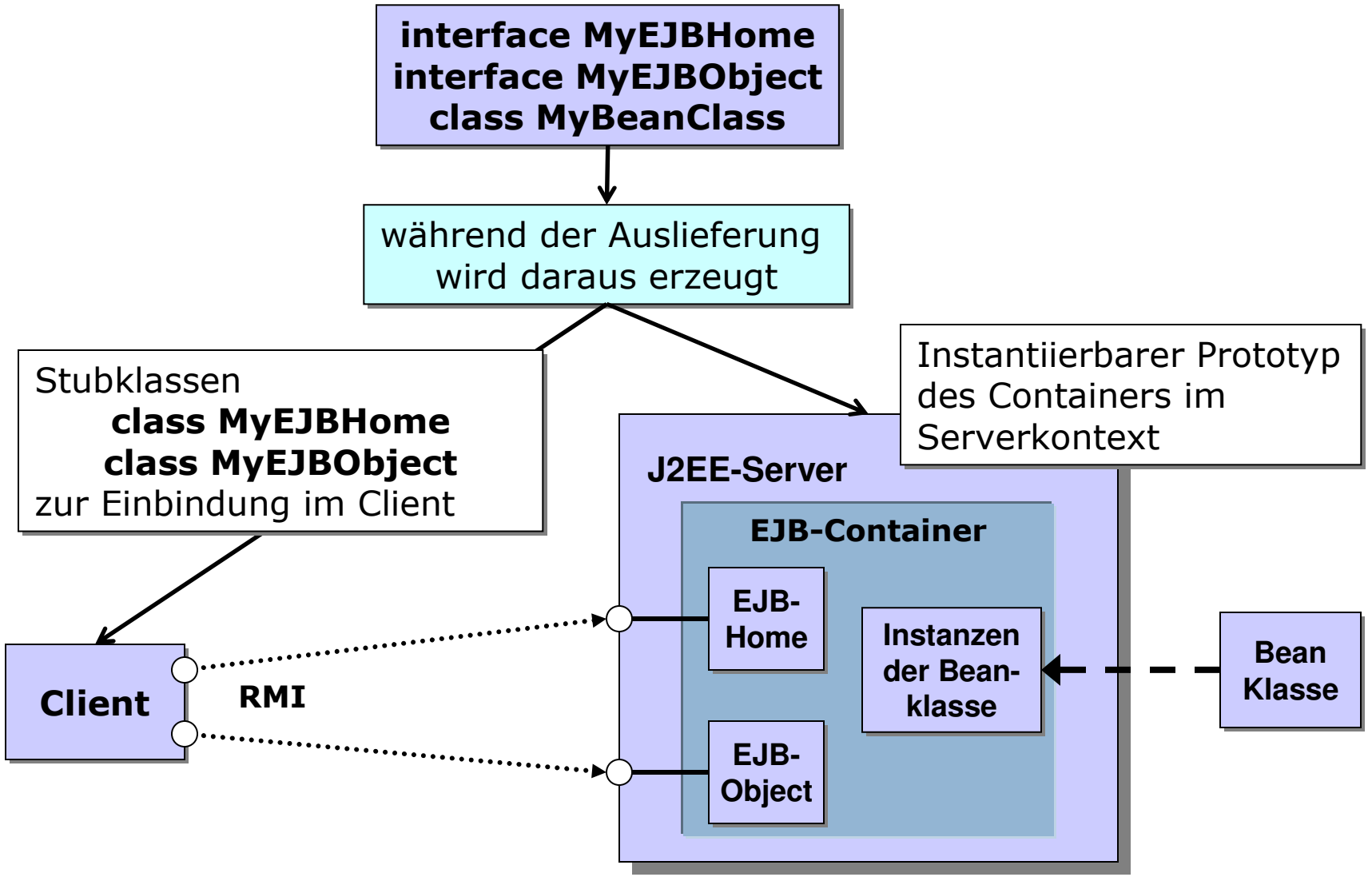
Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2005/06

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Schnittstelle

```
package SemOrg.Schnittstellen;
import java.rmi.*;
import javax.ejb.*;

public interface Buchung extends EJBObject {
    public void buchen(Kunde k, Seminartyp s) throws RemoteException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Container-Schnittstelle

```
package SemOrg.Schnittstellen;
import java.rmi.*;
import javax.ejb.*;

public interface BuchungHome extends EJBHome {
    public Buchung create() throws RemoteException, CreateException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klasse

```
package SemOrg.Server;
import SemOrg.Schnittstellen.*;
import java.rmi.*;
import javax.ejb.*;

public class BuchungBean implements SessionBean{

    // Konstruktor
    public BuchungBean() {}

    //Operationen der Remote-Schnittstelle Buchung
    public void buchen(Kunde k, Seminartyp s) throws RemoteException
        { /* Code zur Ausführung der Buchung */ }

    // Implementierung der Schnittstelle SessionBean
    public void ejbRemove() {}
    public void ejbActivate() {} // etc.
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Deployment-Deskriptor

```
<!-- Deployment descriptor -->
<enterprise-beans>
  <session>
    <ejb-name>SemOrg/Buchung</ejb-name>
    <home>SemOrg.Schnittstellen.BuchungHome</home>
    <remote>SemOrg.Schnittstellen.Buchung</remote>
    <ejb-class>SemOrg.Server.BuchungBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
  </session>
</enterprise-beans>
```

Ein einfacher Client

```
package SemOrg.Client;
import java.rmi.*;
import javax.naming.*;
import SemOrg.Schnittstellen.*;

public class Client {
   public static void main(String[] args) {
      Client einClient=new Client();
      einClient.run();
   }

  public void run() {
    Kunde einKunde=getKunde();
    Seminar einSeminar=getSeminar();
```

```
// Fortsetzung Methode run
try {
  // Namenskontext des Servers finden
  Context ctx = new InitialContext();
  Object temp = ctx.lookup("java:comp/env/Buchung");

  // EJB-Container-Objekt vom Typ BuchungHome auf dem Server
  // finden und instantiieren
  BuchungHome home=
    (BuchungHome) PortableRemoteObject.narrow(
            temp, BuchungHome.class);

  // Bean anfordern und Buchung auslösen
  Buchung bean = home.create();
  bean.buchen(einKunde, einSeminartyp);
 }
 catch(Exception e) { }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Server-Provider (Server-Anbieter)

- stellt Plattform zur Verfügung
- Netzwerkanbindung
- Skalierungsfunktion
- Prozess- und Ressourcenmanagement

Container-Provider (Container-Anbieter)

- setzt auf Plattform des Server-Providers auf
- Benutzerverwaltung
- Transaktionsmanagement
- Persistenz
- Installationswerkzeuge
- Implementation der in der EJB-Spezifikation definierten Schnittstellen
- Container- und Server-Provider oft identisch
  - bessere Performance und Wartbarkeit

Bean-Provider (Komponenten-Entwickler)

- realisiert geforderte Anwendungslogik (Geschäftslogik)
- keine grundlegenden Funktionalitäten (Persistenz, Netzwerk, etc.)
- benötigt Fachwissen über Anwendungsbereich
- Konzentration auf inhaltliche Problemstellung

Application-Assembler (Monteur)

- verbindet Komponenten zu einer Anwendung
- Clients
- Verwendung von Basiskomponenten
- Nutzung wiederverwendbarer Komponenten von Drittanbietern
- oftmals Bean-Provider und Application-Assembler in einem Haus

Bean-Deployer (Installation)

- installiert Komponenten der Anwendung auf Zielsystem
- verwendet Werkzeuge des Container-Providers
- nutzt Deployment-Deskriptor
- konfiguriert EJBs
- generiert Stummel- und Skelettklassen
- legt Benutzerdatenbank an
- benötigt detailliertes Wissen über Server und Container

Systemadministrator

- verantwortlich für reibungslosen Ablauf
- Benutzerverwaltung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java-Basisdienste für Komponenten

Grundlegende Dienste

Java core reflection erlaubt zur Laufzeit

- Inspektion von Klassen und Schnittstellen nach Attributen und Methoden
- Konstruktion neuer Klasseninstanzen und Felder
- Zugriff und Modifikation von Attributen in Verbundobjekten oder Feldern
- Aufruf von Methoden von Objekten und Klassen

- **java.lang.reflect** als eigene Klasse hierfür
  - einige Funktionalität historisch in der (finalen) Klasse **java.lang**.

Objektserialisierung

- standardisiertes Kodierungsschema für Serialisierung
- Klasse muss dafür Interface **java.io.Serializable** implementieren
- Objektserialisierung ist sicherheitskritisch

Objektserialisierung (Fortsetzung)

- Mechanismen zu Serialisierung und Deserialisierung ganzer Objekt-Webs
  - nicht zu serialisierende Attribute können als transient markiert werden
    - Bsp: große Cache-Strukturen
  - private Methoden **readObject** und **writeObject** werden statt Default genommen, wenn durch Reflektion gefunden
  - Mehrfachreferenzen auf ein Objekt werden rekonstruiert
- unterstützt einfaches Versionierungsschema:
  - 64-bit-hash-Code (Serial version UID = SUID) wird über die Signatur der Klasse berechnet und kann während **readObject** ausgewertet werden.

Java GuI-Klassensammlungen AWT und Swing

- delegierende Ereignisbehandlung
- Datentransfer und Zwischenablage wird unterstützt, drag and drop
- Java 2D rendering, eng damit zusammen Java printing model
- Internationalisierung
- pluggable look and feel, Palette von Standard-Komponenten

Ferne Objekte und RMI

- Auf ferne Objekte kann nie direkt zugegriffen werden, sondern nur über Interface **java.rmi.Remote**. Ressourcenbindung über Namensdienst.
- Remotezugriff kann immer fehlschlagen:
  Exception **java.rmi.RemoteException**
- Parameterübergabe:
  - Referenz, wenn Parameterwert selbst vom Remote-Typ ist
  - Durch Marshalling übergebene Kopie, wenn Parameterwert lokal im aufrufenden Kontext
  - Übergabe nicht serialisierbarer Objekte führt zu Laufzeitausnahme
- Unterstützt verteiltes Garbage Collection
  - durch genaue Buchführung über Remote-Referenzen
  - basiert auf Arbeit [Birrel 1993] über Network Objects
  - eines der bedeutendsten Features von Java RMI
- Konflikt mit Begriff der Objektidentität im Java-Standard
  - Referenzen auf ein fernes Objekt sind Java Referenzen auf das lokale Proxy des fernen Objekts
  - Backcall erzeugt ein lokales Proxy im ObjectHome, neben der eigentlichen Java-Referenz

Weitere Java-Dienste für Komponenten

JNDI: Java Naming and Directory Service

Aufgabe: Dienste über Namen bzw. Attribute finden

- Interface **Context** macht Namenskontext verfügbar
- Methode **lookup** findet Objekte über ihren Namen
- Interface **DirContext** erweitert **Context** zur Suche über Attributwerte
- Unterstützung von Kontexthierarchien, die rekursiv durchsucht werden.

JMS: Java Messaging Service

Aufgabe: Unterstützung asynchroner datengetriebener Kompositionsmodelle

- Standardisiert Java-Zugriff auf vorhandenes Nachrichtensystem, implementiert keins selbst.

JDBC: Java database connectivity

Aufgabe: Einheitlicher Zugriff auf Datenbanken über entsprechende Treiber

JTA: Java Transaction API

JTS: Java Transaction Service

Aufgabe: Unterstützung von Transaktionskonzepten

JCA: J2EE Connector Architecture (seit J2EE 1.3)

- Einheitliches Konzept des Ressourcen-Adapters, über welches externe Ressourcen aus einer EIS (enterprise information structure) in einen J2EE-Applikationsserver eingebunden werden können
- Definition eines entsprechenden **JCA common client interface** (CCI)
- Einsatz innerhalb von Enterprise Application Integration Frameworks

Java und XML: Java unterstützt mit entsprechenden Klassen

- XML-Dokumente (DOM)
- XML-Streaming (SAX)
- XML-Binding (JAXP)
- XML-Messaging (JAXM)
- XML-Processing (JAXP)
- XML-Registries (JAXR)

Java und CORBA

- Seit CORBA 2.2 (1998) gibt es OMG IDL to Java binding sowie Java to OMG IDL reverse binding
  - Java als die wichtigste CORBA-Referenzimplementierung
- Reverse binding interessant für Anbindung von Nicht-Java-Systemen an Java-Systeme über IIOP-Standard von CORBA
- Heute Koexistenz in fast allen Applikationsserver-Produkten
  - Zugriff auf CORBAservices über Java-spezifische Zugriffs-Schnittstellen sowie weitere Konzepte (POA, Namensdienst) seit Java 1.4
  - RMI-over-IIOP als eingeschränkte RMI-Version seit 1999
    - RMI nutzt spezifisches proprietäres Protokoll
    - keine Unterstützung des verteilten Garbage Collection, so dass explizites Lebenszyklus-Management erforderlich ist
      - dafür existieren aber CORBAservices

Entwicklung einer CORBA – Anwendung unter JAVA
(aus "Lehrbuch der Softwaretechnik" von Helmut Balzert)

1) Spezifikation der Schnittstelle in CORBA – IDL
2) Übersetzung mittels IDL – Compiler
   - Stummel- und Skelettklassen werden erzeugt
3) Implementierung der Operationen der Schnittstelle
4) Rahmenanwendung entwickeln
   - Erzeugt Objekt der Klasse
   - Objekt wird für Clients zugreifbar gemacht
5) Entwickeln des Clients

Definition der Schnittstelle mit OMG IDL

```
module SemOrg { // Schnittstelle der Klasse Firma
    interface Firma {
        attribute string Name;
        attribute float Umsatz;
        // Operationssignatur
        float berechneGewinn( in float Kosten );
    };
};
```

SemOrg.idl

`idlj –f all SemOrg.idl` erzeugt daraus Java-Package **SemOrg**

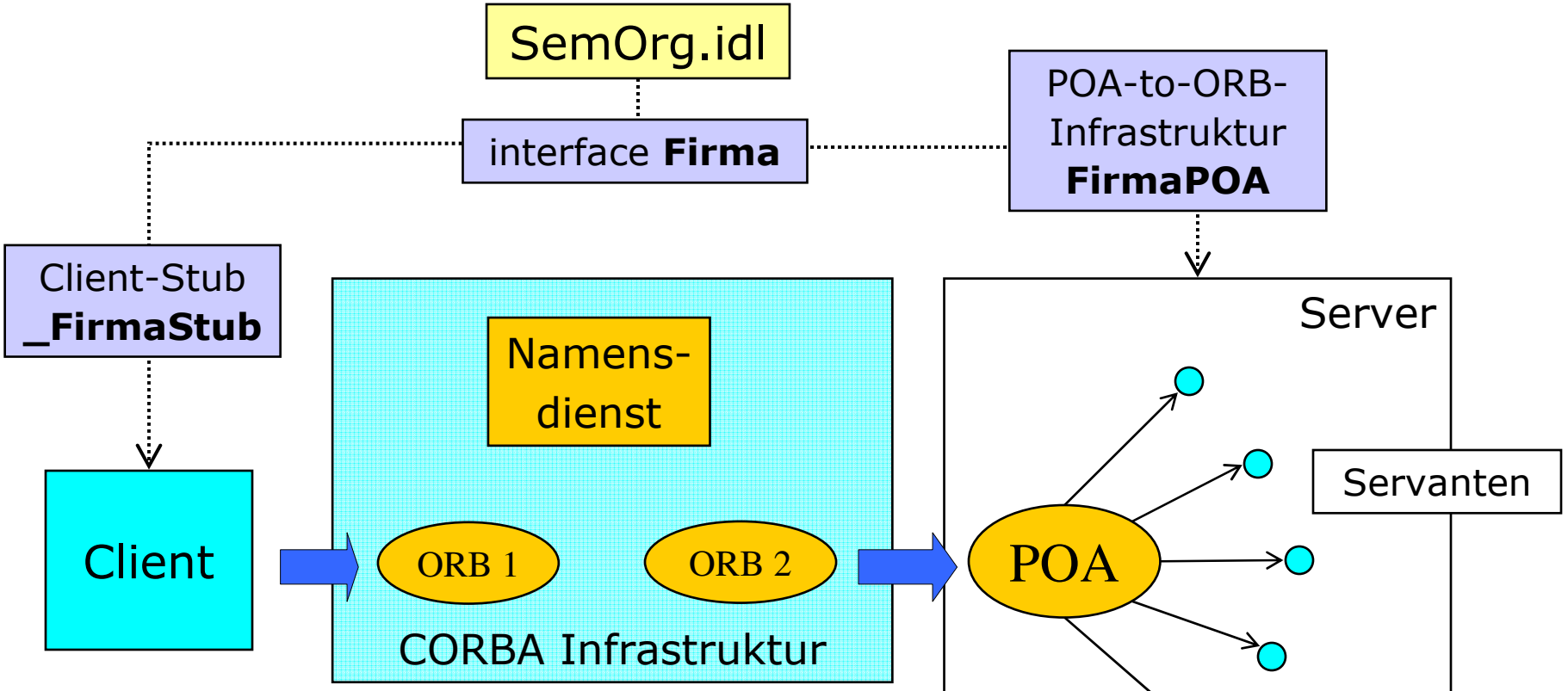
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vom IDL – Compiler erzeugte Klassen

- interface **FirmaOperations**
  - interface Firma als Java-Interface
- interface **Firma** extends FirmaOperations, ...
  - Firma-Operations + CORBA.Object ...
- class **_FirmaStub** extends CORBA.portable.ObjectImpl
  implements SemOrg.Firma
  - voll generierte Stummel – Klasse
- final class **FirmaHolder** implements CORBA.portable.Streamable
- abstract class **FirmaHelper**
  - „Cast" von CORBA.Object zu Firma
- abstract class **FirmaPOA** extends PortabeServer.Servant ....
  - portabler Objekt-Adapter
  - generierte Implementierung der ORB-seitigen Kommunikation

CORBA-Beispiel Seminarorganisation

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



Client:
Hält eine CORBA-Referenz, in welcher der Objekt-Key steht.

ORB 2:
Kennt POA, der das Objekt mit dem Objekt-Key bedient

POA:
Kennt Zuordnung Objekt-Key ↔ Objekt

Dienst nutzen ⇦ Dienst finden ⇦ Dienst bekannt machen ⇦ Dienst als Subinstanz des POA generieren

Implementierung des Servanten

```
package SemOrg;
public class FirmaImpl extends FirmaPOA {
    private String derName;
    private float derUmsatz;
    public FirmaImpl() { // Konstruktor }
    public float berechneGewinn(float Kosten)
        { return this.Umsatz – Kosten; }
    // get-/set-Operation für Attribut Name
    public String Name() { return this.derName }
    public void Name(String neuerName)
        { this.derName = neuerName; }
    //analog für Umsatz ...
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Default-Server-Modell: Portable Servant Inheritance Model

Aufgaben des Servers

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- Instanz des Servanten **FirmaImpl** anlegen und in der CORBA-Struktur verankern: als POA registrieren und den POAManager aktivieren.
- CORBA-Objektreferenz des Servanten besorgen
- Objektreferenz auf einen Namens-Kontext von Namens-Server der ORB-Infrastruktur holen und den Servanten als „eineFirma" registrieren.
- Auf Anfragen warten

Aufgaben des Client

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- CORBA-Objektreferenz auf den Servanten „eineFirma" über den Namens-Kontext besorgen
- Dienste des Servanten in Anspruch nehmen

Serverkomponente

```
package SemOrg;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class SemOrgServer {
  public static void main(String[ ] args) {
    try {
      // Kontakt zur ORB-Infrastruktur herstellen: ORB-Referenz holen
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      // POA-Referenz vom ORB besorgen und POA aktivieren
      POA poa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      poa.the_POAManager().activate();
      // Servanten anlegen als reales Java-Objekt
      FirmaImpl impl = new FirmaImpl();
```

```
      // CORBA-Objektreferenz auf den Servanten besorgen und „casten“
      org.omg.CORBA.Object implObjServer = poa.servant_to_reference(impl);
      Firma eineFirmaS = FirmaHelper.narrow(implObjServer);
      // vom ORB Referenz auf Namensdienst-Server besorgen und „casten“
      org.omg.CORBA.Object ncObj =
            orb.resolve_initial_reference("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
      // Servanten unter dem Namen "eineFirma" im Namensdienst bekannt
      // machen. Verbindung zwischen "eineFirma" und dem Java-Objekt impl
      // kennt nur poa
      ncRef.rebind(ncRef.to_name("eineFirma"), eineFirmaS);
      System.out.println(„SemorgServer gestartet ...“);
    } // end try
    catch (Exception e) { ... }
  } // end main
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Client-Komponente

```
package SemOrg;
import org.omg.CosNaming.*;

public class SemOrgClient {
  public static void main(String[ ] args) {
    try {
      // Verbindung zur ORB-Infrastruktur
       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

      // vom ORB Referenz auf Namensdienst besorgen
      org.omg.CORBA.Object ncObj = orb.resolve_initial_reference("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
```

```
        // CORBA-Objektreferenz auf das Servanten-Objekt „eineFirma“ besorgen
        org.omg.CORBA.Object firmaObj = ncRef.resolve_str("eineFirma");
        Firma eineFirmaC = FirmaHelper.narrow(firmaObj);

        System.out.println(“Handle auf das Server-Objekt”+eineFirmaC);

        //Aufrufe durchführen
        System.out.println(eineFirmaC.Name());
            ...
        }
        catch (Exception e) { ... }
    }
}
```