Vorlesung
Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Enterprise JavaBeans EJB 2.1

- **Modell**: Beans = ununterscheidbare Objekte in einem Container
- Container-Abstraktion repräsentiert die spezielle Art, in welcher Beans an Ressourcen gebunden sind.
- kein direkter Zugriff auf Attribute und Methoden von Beans, auch nicht innerhalb desselben Containers
  - Verhältnis wie zwischen DB-Server und Datensätzen
  - Zugriff nur über zwei Schnittstellen, die **javax.ejb.EJBHome** (für Container) und **javax.ejb.EJBObject** (für Beans) erweitern
  - EJBHome: Methoden zum Management des Lebenszyklus der Beans
  - EJBObject: Methoden der Bean-Instanzen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Schnittstelle EJBObject

- **interface MyEJBObject extends javax.ejb.EJBObject**
- Aufruf-Schnittstelle, ergänzt EJBObject um die Fachlogik, über welche ein Client auf den Dienst zugreifen kann
- **Beschreibt** Dienstleistung
  - Darstellung von Attributen über get/set-Methoden
- Nichtlokale Clients rufen Schnittstelle auf Stub-Objekt, welches über RMI oder RMI-über IIOP mit dem EJB Container kommuniziert
  - deklarierte Operationen müssen Exception **java.rmi.RemoteException** auslösen können
  - Abfangen von Kommunikationsproblemen im Netz
- Lokale Clients können über lokale Version der Schnittstelle (Erweiterung von **EJBLocalObject**) zugreifen, wenn von der e-bean bereitgestellt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Container-Schnittstelle EJBHome

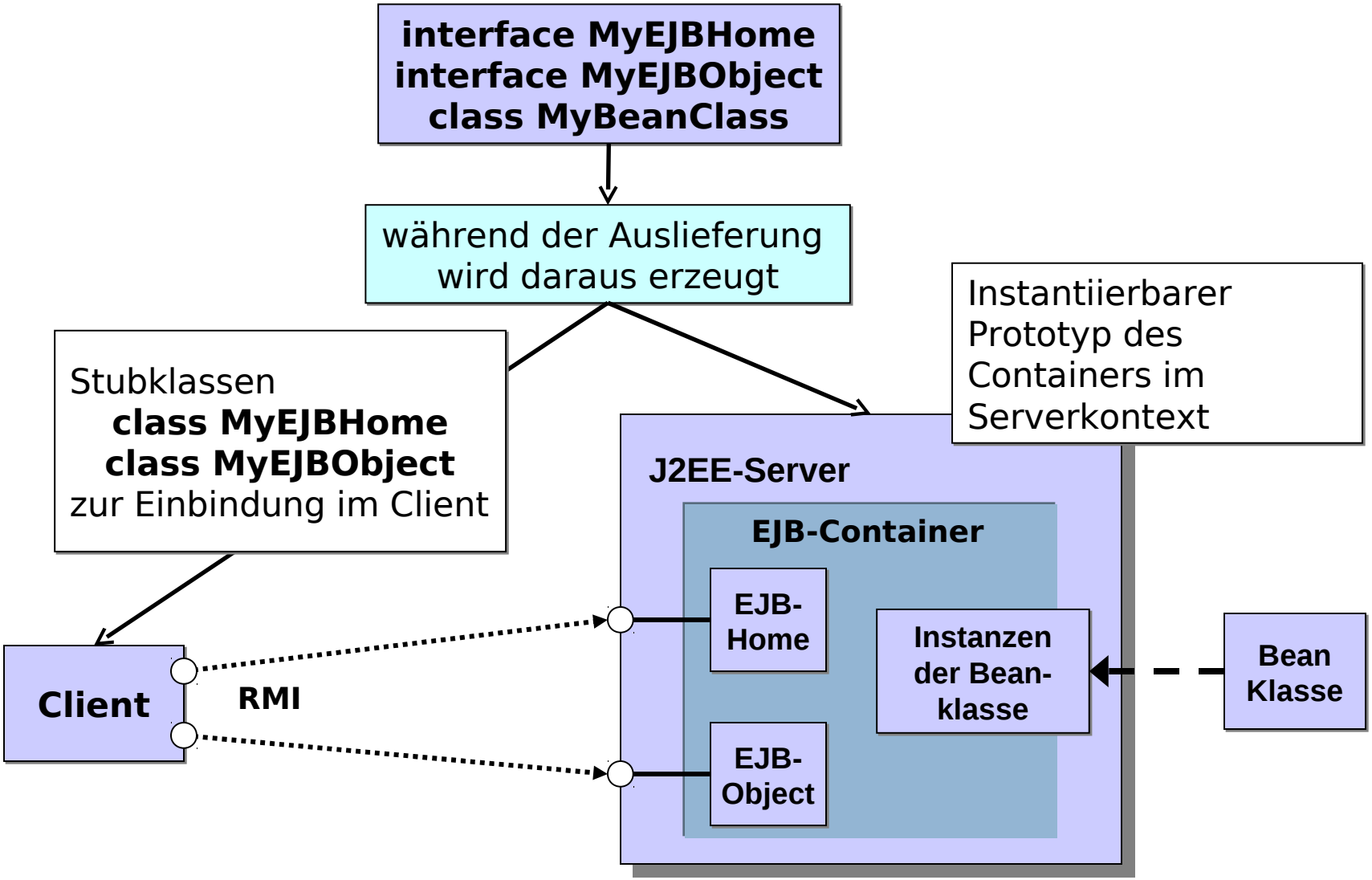
- **interface MyEJBHome extends java.ejb.EJBHome**
- Verwaltungs-Schnittstelle: verwaltet Bean-Instanzen im Container
  - Erzeugen neuer Instanzen
  - Auffinden vorhandener Instanzen
  - serialisiert alle Zugriffe auf seine Beans
- Operationen **create** und **findByPrimaryKey** (entity beans)
- optional weitere Methoden, die nicht mit einzelnen Beans zu assoziieren sind (analog zu statischen Methoden einer Klasse)
- begleitet Lebenszyklus seiner Beans
  - **setEntityContext** oder **setSessionContext** erzeugt Rückverweis auf den Container-Kontext
  - **ejbCreate** / **ejbRemove**
    - Ressourcenallokation bzw. –freigabe
  - **ejbPassivate** / **ejbActivate**
    - zeitweise Trennung von den Ressourcen und Speichern in serialisierter Form auf externem Medium

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klassen

- **public class MyBeanClass implements javax.ejb.xxBean**
- es gibt **EntityBeans**, **SessionBeans** und **MessageDrivenBeans**, alle sind Subklassen von **EnterpriseBeans**
- **Implementierung** der zur Verfügung gestellten Operationen
  - also eigentlich auch **... implements MyEJBObject**
  - wird aber nur informell überwacht und diese Verbindung erst bei der Installation hergestellt (ist in Wirklichkeit noch etwas komplexer)
  - Entwickler muss auf Übereinstimmung der Signaturen selbst achten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Anforderungen an den EJB–Container-Kontext

- JVM wird benötigt, ist aber allein nicht ausreichend
- EJB-Standard spezifiziert Schnittstelle und Verhalten von Container und J2EE
- Container benötigt zur Verwaltung seiner Beans
  - Namensdienst (Java Name and Directory Interface)
  - Transaktionsmonitor (Java Transaction API)
  - Datenbankzugriff (Java Data Base Connectivity)
  - eMail (Java Mail API)
  - Standard Java API
- Greifen dabei gewöhnlich auf weitere Dienste im Rahmen des J2EE Applikationsservers zu

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Schnittstelle

```
package SemOrg.Schnittstellen;
public interface Buchung extends javax.ejb.EJBObject {
    public void buchen(Kunde k, Seminartyp s)
        throws java.rmi.RemoteException;
}
```

Container-Schnittstelle

```
package SemOrg.Schnittstellen;
public interface BuchungHome extends javax.ejb.EJBHome {
    public Buchung create(int owner_id)
        throws java.rmi.RemoteException, javax.ejb.CreateException;
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Klasse

```
package SemOrg.Server;
public class BuchungBean implements javax.ejb.SessionBean {

    public BuchungBean() {}

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

    // Implementierung der Schnittstelle SessionBean
    private javax.ejb.SessionContext ctx;
    public void setSessionContext(javax.ejb.SessionContext sc) {
        this.ctx=sc; // Session-Context für Callback-Methoden
    }
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
}
```

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>
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ein einfacher Client

```
public class Client {

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

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

      try {
      // Namenskontext des Servers finden
      javax.naming.Context ctx = new javax.naming.InitialContext();
      Object temp = ctx.lookup(“java:comp/env/Buchung”);
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Deployment-Kontrakt

Komponente wird in einem speziellen Paketformat ausgeliefert, das eine genaue Entpackungsbeschreibung im XML-Format enthält

- Dienst-Schnittstelle, Factory-Schnittstelle, Bean-Implementierung, Ressourcen-Zuordnung, Metainformationen

Lebenszyklus-Kontrakt

Komponente implementiert spezielle Lebenszyklusfunktionen, die vom Container „automagisch“ aufgerufen werden können

- OnActivate() {…}
- OnPassivate() { … }

Container-Service-Kontrakt

Der Container stellt der Komponente ein Kontext-Objekt oder eine Schnittstelle zur Verfügung, über welche die Komponente transparent Dienste aus dem Kontext in Anspruch nehmen kann.

- WhoIsCaller() { … }
- AccessDataBase() { … }
- LocateComponent() { … }

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Umgebungs-Kontrakt

Der Container sichert eine funktionierende Umgebung für die Komponente entsprechend der Deployment-Information.

Erweiterungs-Kontrakt

Der Container kann selbst erweiterbar sein und Verhaltensänderungen ausgerollter Komponenten unterstützen

- nutzergetriebene Unterbrechungen
- Unterstützung der Laufzeitkonfiguration von Einheiten
- Einbindung weiterer Dienste zur Laufzeit
- Versionsmanagement ausgerollter Komponenten
- Aspektorientiertes Verhalten

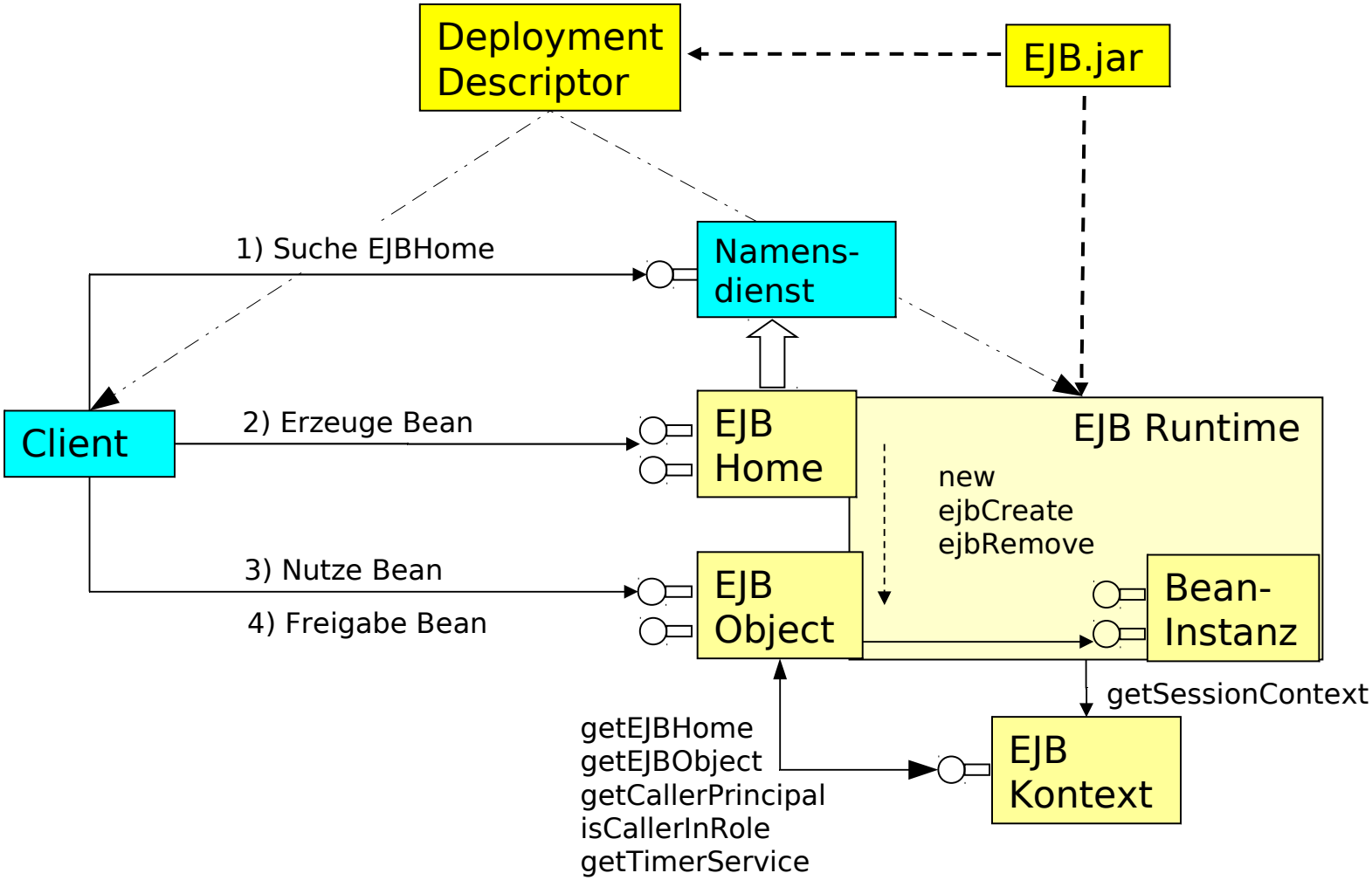
Client-Container-Kontrakt

Client nutzt Komponentendienste über den Container.

Framework bietet Dienste zum Auffinden der Komponente.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Architektur von EJB 2



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Probleme des Standards

- Viele Schnittstellen zu implementieren (EJBHome, EJBObject, callback zum Kontext)
- Nur eine Geschäftsmethoden-Schnittstelle pro EJB Bean
- Weitere Konventionen sind zu beachten (RMI, spezielle Basis-Schnittstellen)
- Zusätzliche und andere Konventionen für EJB-Klassen im Vergleich zu normalen (plain old) Java-Klassen
- Konfiguration von Beans und Applikationen für Beans erfolgt durch riesige XML-basierte Deployment-Deskriptoren
- EJB-Laufzeit zu rudimentär spezifiziert
- Komplexität der Interaktion mit der Persistenzschicht
- Keine Schnittstellen für Logging, Tracing und andere Basisdienste, um Komponenten zu testen und in der Laufzeit zu verfolgen.
- Beans müssen vom Client manuell gesucht und angesprochen werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ziele von EJB 3

- Genauere Spezifikation des Bean-Lebenszyklus durch mehr Erweiterungspunkte im Kontext
- Dependency Injection: Einzelne Attributwerte, Methoden- oder Klassendefinitionen werden zur Laufzeit aus dem Kontext importiert
- Verwendung von Meta-Daten
- Interzeptoren und aspektorientierte Ansätze für Infrastrukturdienste
- Vereinfachung der Persistenzbehandlung
- Reduktion der Zahl der Artefakte, die ein Bean-Entwickler bereitstellen muss
- Vereinfachung der EJB Typen durch Reduktion der Zahl der Schnittstellen
- Verbesserung der Testmöglichkeiten außerhalb des Containers

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wie wird das umgesetzt?

- Einsatz von POJO‘s (plain old java objects) und POJI‘s (plain old java interfaces)
  - Dienstschnittstelle als Java-Schnittstelle
  - kein Home-Interface mehr
  - Annotation durch Metadaten für die Konfiguration der EJB-Typen, Local/Remote, Transaktionen, Sicherheit
- Dependency Injection
  - für Attribute, Eigenschaften, Ressourcennutzung
  - werden aus der Umgebung durch Annotationen in die Laufzeit der Beaninstanz „injiziert“
- Erweiterter Lebenszyklus-Unterstützung
  - nutzerdefinierte Callback-Methoden
- Interzeptoren
  - Unterbrechung der Geschäftsmethode in AOP-Manier

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Die Bean-Klasse

- Bean-Klasse als zentrales Artefakt, das bereitgestellt wird
- keine Home-Schnittstelle mehr, Laufzeitunterstützung kommt direkt aus dem Kontext
- Beantyp wir durch Annotation festgelegt
- Beanklasse kann Ausnahmen über Annotation vom Typ @**ApplicationException** werfen
- Beanklasse wirft keine **java.rmi.RemoteException** mehr

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

@Stateful public class BuchungBean implements Buchung {
    public void buchen(Kunde k, Seminartyp s) {
        //Code zur Ausführung der Buchung
    }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Ereignisse im Lebenszyklus

- Ereignisse im Lebenszyklus werden durch den Container über annotierte Callbacks an die Bean-Klasse weitergegeben
- Callbacks können **RuntimeException** auslösen, die Rollback von Transaktionen nach sich ziehen, aber keine **ApplicationException**.
- Letzteres kann durch annotierte **CallBackListener** implementiert werden

```
@Stateful public class BuchungBean implements Buchung {
    @PreDestroy dispose() { ... }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Interzeptoren

- Interzeptoren unterbrechen die Abarbeitung einer Geschäftsmethode
- Interzeptormethoden können in der Beanklasse oder in eigenen Interzeptorklassen definiert sein
  - können **RuntimeException** oder **ApplicationException** werfen, wie in der Schnittstelle der Geschäftsmethode vereinbart
  - Abarbeitung im selben Transaktions- und Sicherheitskontext wie die Geschäftsmethode
  - Können JNDI, JDBC, JMS, andere Beans und den **EntityManager** (Schnittstelle zum Persistenzkontext) nutzen
  - verwendet Dependency Injection
- Für dieselbe Geschäftsmethode können mehrere Interzeptoren definiert sein
- Signatur einer Interzeptormethode:

  public Object methodName(invocationContext) throws Exception

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Programmiermodell: Interzeptoren (Fortsetzung)

- **InvocationContext** definiert als

```
public interface InvocationContext {
  public Object getBean();
  public Method getMethod();
  public Object[] getParameters();
  public void setParameters(Object[]);
  public EJBContext get EJBContext();
  public java.util.Map getContextData();
  public Object proceed() throws Exception;
}
```

- Kontext-Daten werden von allen Interzeptoren derselben Routine gemeinsam genutzt.
- Aufruf-Ende von **proceed**() innerhalb des Interzeptors - Rückkehr in die unterbrochene Routine

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige Annotationen und deren Defaultwerte

- **@Local** und **@Remote**: Annotation von Klassen und Schnittstellen
  - Default: @Local

```
@Remote public interface Buchung {
    public void buchen(Kunde k, Seminartyp s) ;
}
```

- **@TransactionManagement**: Annotation der Beanklasse
  - Default TransactionManagementType.CONTAINER
- **@TransactionAttribute**: Annotation der Klasse oder einzelner Methoden
  - Beschreibt das Transaktionsverhalten
  - Default: REQUIRED
- **@Timeout**: Definition einer Timeout-Methode der Bean
- **@ApplicationException**
- Weitere Annotationen für Sicherheitsaspekte:
  - **@RolesReferences**, **@RolesAllowed**, **@RunAs**, **@SecurityRoles**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandslose Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateless**
- Definierte Callbacks **@PostConstruct** und **@PreDestroy**
- Interzeptor **@AroundInvoke** um eine einzelne Session
- Unspezifizierte Transaktions- und Sicherheitskontexte
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean und Interzeptor. Beispiel

```
@Stateless @Interceptors ({MyInterceptor.class})
public class BuchungBean implements Buchung {
   public void buchen(Kunde k, Seminartyp s) { ... }
}


public class MyInterceptor {
   @AroundInvoke
   public Object zeitVerbrauch(InvocationContext ic) throws Exception {
            long time = System.currentTimeMillis();
            try { return ic.proceed(); }
            finally {
                  long total = System.currentTimeMillis() - time;
                  System.out.println("Aufruf von " + ic.getMethod().getName()
                  + " dauerte "+total+" Millisekunden.");
            }
}
```

Zustandsbehaftete Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateful**
- kann Schnittstelle **SessionSynchronization** implementieren
- Definierte Callbacks **@PostConstruct**, **@PreDestroy**, **@PostActivate**, **@PrePassivate**
- Interzeptoren
  - **@afterBegin** – Ausführung zu Beginn jeder Session
  - **@beforeCompletion** – Ausführung von Ende jeder Session
  - **@AroundInvoke** – Einbettung einer Session
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen
- Client bekommt Referenz auf Session direkt über JNDI oder über Dependency Injection
- Annotation **@Remove** für Methode zum Auflösen der Beaninstanz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandbehaftete Bean. Beispiel

```
@Stateful public class BuchungBean implements Buchung {
    Connection rc;
    @Resource SessionContext sc;

    @PostConstruct @PostActivate public void init() {
            rc = Connection.Open();
    }

    @PreDestroy @PrePassivate public void close() {
            try { rc.close(); }
            catch (CloseException) { /* no op */ }
    }
// ... weiter
```

```
    @Remove public void dispose() { /* was auch immer */ }

    @AroundInvoke public Object monitor(InvocationContext ic) {
            try {
                Object res = ic.proceed();
                if ( (OpResult) res == OpResult.SUCCEED ) {
                    System.out.println("Aufruf war okay");
                }
                return res;
            } catch (Exception ex) { throw ex; }
    }

    // Geschäftsmethode
    public void buchen(Kunde k, Seminartyp s) { ... }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandbehaftete Bean. Beispiel

Zugriff durch den Client:

```
static public void testBean(Kunde k, Seminartyp s) throws Exception {
    javax.naming.Context ctx = new javax.naming.InitialContext();

    Buchung test = ctx.lookup(Buchung.class.getName());

    // oder Injection, wenn innerhalb einer anderen Bean:
    /* @EJB Buchung test; */

    test.buchen(k,s);          // Test der Geschäftsmethode
    test.dispose();            // Test der Bean dispose-Methode
}
```