Internetprogrammierung 2006

Enterprise JavaBeans—Entity Beans/2

Peter Thiemann

Universität Freiburg, Germany

Vorlesung Internetprogrammierung, 05.07.2006

Inhalt

Lokale Interfaces

- Remote Interfaces sind
  - umständlich zu programmieren
    - jede Methode wirft `RemoteException`
    - RMI-Umstand: `PortableRemoteObject.narrow(...)`
  - ineffizient für Zugriffe innerhalb des gleichen Containers
    - Wertübergabe statt Referenzübergabe
- Beans im gleichen Container sind **colocated** und können über *lokale Interfaces* (ab EJB 2.0) kommunizieren
- Lokale Interfaces
  - `find`-Methode liefert lokales Interface
  - Typcast statt `narrow`
  - Referenzübergabe
- Ein Bean kann sowohl ein Remote wie auch ein Lokales Interface anbieten.

Beispiel: Entity Bean Address

Local-Home-Interface und Local-Interface

```
public interface AddressLocal extends EJBLocalObject {
  public String getStreetAddress();
  public void setStreetAddress(String sa);
  public String getCity();
  public void setCity(String city);
  /* get/set methods for all attributes */
}
public interface AddressHomeLocal
  extends javax.ejb.EJBLocalHome {
  public AddressLocal create(Integer pk) throws
    CreateException;
  public AddressLocal findByPrimaryKey(Integer pk) throws
    FinderException;
}
```

Deployment mit lokalen Interfaces

```
<entity>
  <ejb-name>AddressEJB</ejb-name>
  <home>proglang.j2ee.ejbs.AddressHomeRemote</home>
  <remote>proglang.j2ee.ejbs.AddressRemote</remote>
  <local-home>proglang.j2ee.ejbs.AddressHomeLocal</local-home>
  <local>proglang.j2ee.ejbs.AddressLocal</local>
  <ejb-class>proglang.j2ee.ejbs.AddressBean</ejb-class>
</entity>
```

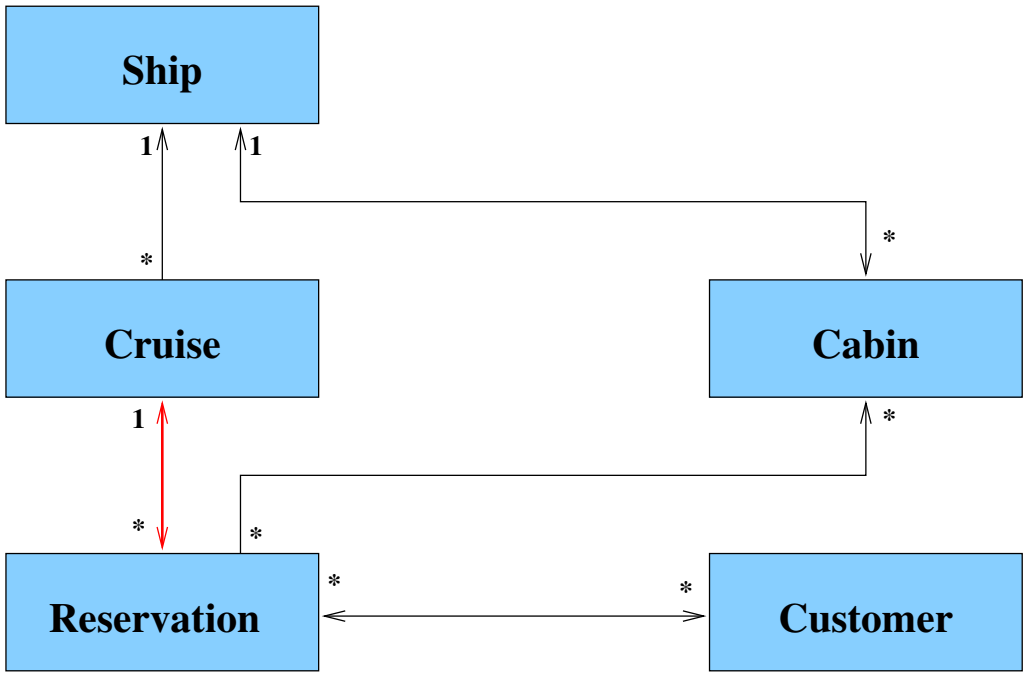
Assoziationen

- Standardmodellierung für Daten
  - Entity-Relationship Diagramm
  - Klassen Diagramm (Klassen und Assoziationen)
- Navigation über Assoziationen
- Assoziationen und Vielfachheiten

Container-Managed Relationships (CMR)

- Deklaration von Assoziationen im Deployment Descriptor
- Ziel muss lokales Interface sein
- Navigation durch `get`/`set`-Methoden
  - unidirektional oder
  - bidirektional
- Hauptvielfachheiten werden unterstützt
  - `One` für `0..1`
    Ergebnis: lokales Interface (Referenz oder `null`)
  - `Many` für `0..*`
    Ergebnis: `Collection` oder `Set` (nie `null`)
- Insgesamt sieben Assoziationstypen

Beispiel für CMR



- Assoziation zwischen `Cruise` und `Reservation`
- Bidirektional, One-to-many Assoziation

Local-Interface von `Cruise`

```
public interface CruiseLocal
    extends javax.ejb.EJBLocalObject
{
   public String getName();
   public void setName(String name);

   public ShipLocal getShip();
   public void setShip(ShipLocal ship);

   public void setReservations(Collection res);
   public Collection getReservations();

}
```

Local-Interface von `Reservation`

```
public interface ReservationLocal
    extends javax.ejb.EJBLocalObject
{
   public Date getDate();
   public void setDate(Date date);
   public double getAmountPaid();
   public void setAmountPaid(double amount);

   public CruiseLocal getCruise();
   public void setCruise(CruiseLocal cruise);

   public Set getCabins( );
   public void setCabins(Set customers);

   public Set getCustomers( );
   public void setCustomers(Set customers);
}
```

Bean-Klasse: von `Cruise`

`CruiseBean implements javax.ejb.EntityBean`

```
// persistent fields
public abstract void setId(Integer id);
public abstract Integer getId();
public abstract void setName(String name);
public abstract String getName( );

public abstract void setShip(ShipLocal ship);
public abstract ShipLocal getShip( );
```

```
// relationship fields
public abstract void setReservations(Collection res);
public abstract Collection getReservations( );
```

Bean-Klasse von Reservation

ReservationBean implements javax.ejb.EntityBean

```
// persistent fields
public abstract Integer getId();
public abstract void setId(Integer id);
public abstract Date getDate();
public abstract void setDate(Date date);
public abstract double getAmountPaid();
public abstract void setAmountPaid(double amount);
```

```
// relationship fields
public abstract CruiseLocal getCruise();
public abstract void setCruise(CruiseLocal cruise);
```

Deployment mit Assoziationen

```
<relationships>
  <ejb-relation>
    <ejb-relation-name>Cruise-Reservation</ejb-relation-name>
    <ejb-relationship-role>
      <ejb-relationship-role-name>
        Cruise-has-many-Reservations
      </ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
        <ejb-name>CruiseEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>reservations</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
```

Deployment mit Assoziationen/2

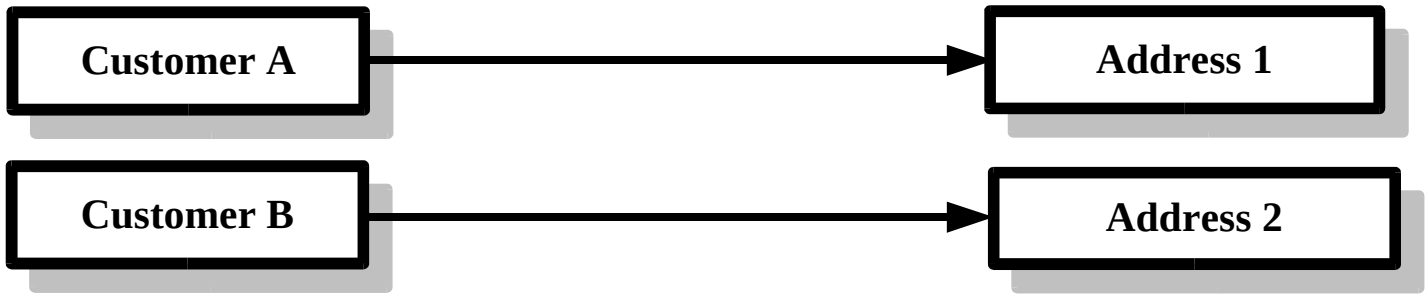
```
    <ejb-relationship-role>
      <ejb-relationship-role-name>
        Reservation-has-a-Cruise
      </ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source>
        <ejb-name>ReservationEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>cruise</cmr-field-name>
      </cmr-field>
    </ejb-relationship-role>
  </ejb-relation>
</relationships>
```

Modifikation von CMR-Feldern

- Löschen kann `cascade-delete` auslösen
- Kollektionen sind *lebendig*, d.h. Änderungen schlagen auf die Datenbank durch
- Bei bidirektionaler Assoziation:
  Modifikation eines Endes der Assoziation beeinflusst auch das andere Ende
- Beispiel:
  - Cruise `c1` ist enthalten in Reservation `r`
  - Weitere Cruise `c2`
  - Aufruf: `c2.getReservations().add(r)`
  - Danach gilt
    - `r.getCruise().isIdentical(c2)`
    - `r` ist nicht in `c1.getReservations()` enthalten

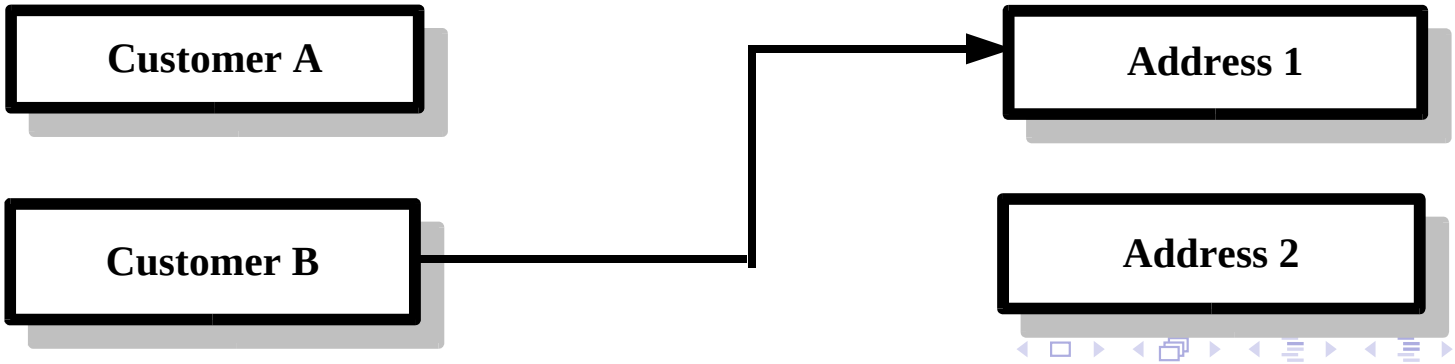
Modifikation von CMR-Feldern

Unidirektional, One-to-one Relationship



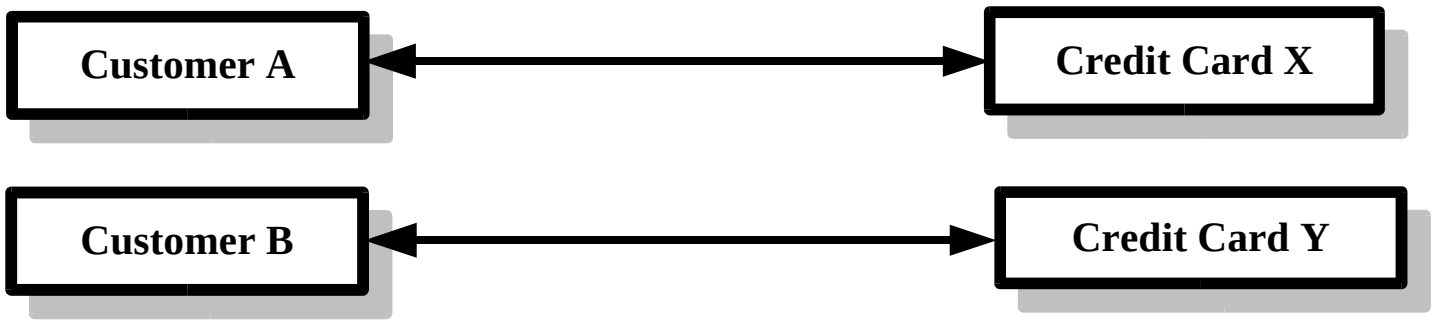
AddressLocal addr_1 = customer A.getHomeAddress();

customer B.setHomeAddress (addr_1);



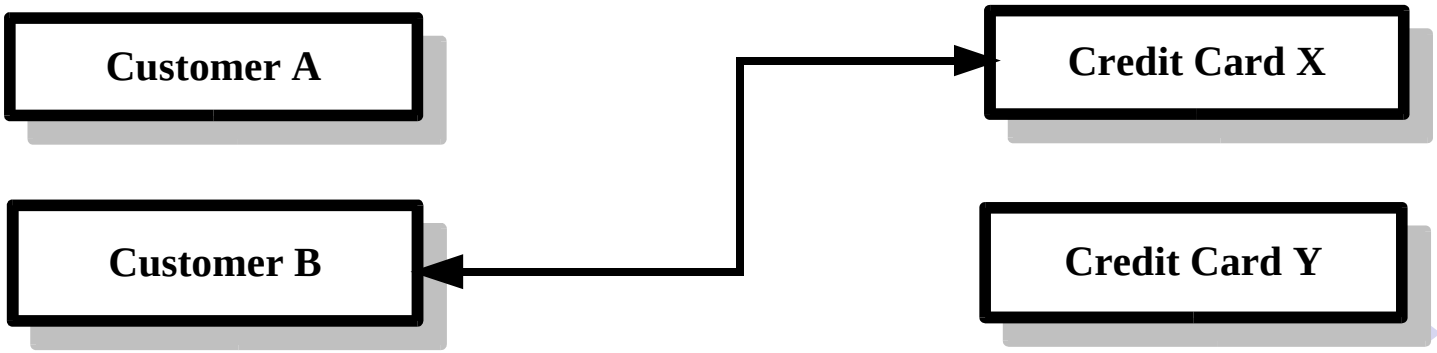
Modifikation von CMR-Feldern

Bidirektional, One-to-one Relationship



CreditCardLocal cardX = customer A.getCreditCard();

customer B.setCreditCard (cardX);



Initialisierung der CMP- und CMR-Felder

- `create`*`XYZ`*`(`*`args`*`)` im Home-Interface
  - `ejbCreate`*`XYZ`* `(`*`args`*`)` in Bean-Klasse
  - `ejbPostCreate`*`XYZ`* `(`*`args`*`)` in Bean-Klasse
- In `ejbCreate ()`: Initialisierung der CMP-Felder
- In `ejbPostCreate ()`: Initialisierung der CMR-Felder

EJB Query Language

- CMR verwendet datenbankunabhängiges, abstraktes Persistenzmodell
- EJB QL ist Anfragesprache gegen dieses Modell
- Verwendet zur Definition von
  - `find`*XYZ* Methoden
    - definiert in Home-Interfaces; soviele wie nötig
    - ein oder mehrere Rückgabewerte (`Collection`) möglich
    - spezifiziert im Deployment Descriptor
  - `select`*XYZ* Methoden
    - Verwendung innerhalb der Bean-Klasse
    - `ejbSelect`*XYZ* Methoden
    - spezifiziert im DD

Find-Methoden im Bookstore

`public interface OrdersLocalHome extends javax.ejb.EJBLocalHome`

```
public java.util.Collection findAll()
    throws javax.ejb.FinderException;

public java.util.Collection findByBuyerName()
    throws javax.ejb.FinderException;
```

OrdersLocalHome.findAll

```
<entity>
  <ejb-name>OrdersEJB</ejb-name>
  ...
  <abstract-schema-name>Orders</abstract-schema-name>
  ...
  <query>
    <query-method>
      <method-name>findAll</method-name>
      <method-params/>
    </query-method>
    <ejb-ql>
      SELECT OBJECT(o) FROM Orders o
    </ejb-ql>
  </query>
  <cmr-field>
    <field-name>customer</field-name>
  </cmr-field>
</entity>
```

OrdersLocalHome.findAll

```
<query>
  <query-method>
    <method-name>findByCustomerName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    SELECT OBJECT(o) FROM Orders o WHERE o.customer.fullName = ?1
  </ejb-ql>
</query>
```

Sprachelemente EJB QL

- Ähnlich SQL
- In `AddressBean`: `... WHERE a.zip = '4711'`
- Literale im Deployment Descriptor
- Operatoren in Anfragen: `LIKE`, `BETWEEN`, `IN`, `IS NULL`, `IS EMPTY`, `MEMBER OF`; `NOT`, `AND`, `OR`
- Aggregatfunktionen (EJB 2.1)
  `COUNT`, `MAX`, `MIN`, `AVG`, `SUM`

Navigation

- CMP-Felder

```
SELECT c.fullName FROM Customer AS c;
```

- CMR-Felder mit Rolle der Vielfachheit `One`
  Alle Kunden, die das Buch mit Titel `?1` bestellt haben

```
SELECT item.order.customer
       FROM OrderItem AS item
       WHERE item.book.title = ?1
```

- CMR-Felder mit Rolle der Vielfachheit `Many`
  Alle Bücher, die der Kunde `?1` bestellt hat

```
SELECT DISTINCT item.book
       FROM Order AS o, IN (o.orderItems) AS item
       WHERE o.customer.fullName = ?1
```

EJB-Methoden

	Stateless Session Bean	Stateful Session Bean
ejbCreate()	Container erzeugt neue Instanz. Nicht direkt mit Client-Aufruf assoziiert.	Client ruft create() am Home-Interface auf
setSessionContext()	direkt vor ejbCreate()	direkt vor ejbCreate()
ejbActivate()	wird nie aufgerufen	Wenn die EJB passiviert ist und eine Methode an ihr aufgerufen wird
ejbPassivate()	wird nie aufgerufen	Wenn Ressourcen freigemacht werden sollen. Muss Instanz in serialisierbarem Zustand hinterlassen (z.B. JDBC-Connections schließen)
ejbRemove()	Container zerstört Instanz. Nicht direkt mit Client-Aufruf assoziiert.	Container zerstört Instanz. Nicht direkt mit Client-Aufruf assoziiert.

EJB Methoden/2

	CMP Entity Bean	BMP Entity Bean
ejbCreate()	Client ruft create() am Home-Interface auf. Container erzeugt entsprechenden Datenbankeintrag. Setzen von CMP-Feldern.	Client ruft create() am Home- Interface auf. Muss persistente Repräsentation erzeugen.
setEntityContext()	direkt vor ejbCreate()	direkt vor ejbCreate()
ejbActivate()	Wenn die EJB mit einem Datenbankrecord assoziiert wird. Meistens leer.	Wenn die EJB mit einem Datenbankrecord assoziiert wird. Meistens leer.
ejbPassivate()	Wenn die Assoziation zwischen EJB und Datenbankrecord aufgelöst wird. Meistens leer.	Wenn die Assoziation zwischen EJB und Datenbankrecord aufgelöst wird. Meistens leer.
ejbRemove()	Container zerstört Instanz. Datenbankrecord wird gelöscht.	Container zerstört Instanz. Persistente Repräsentation muss gelöscht werden.

EJB Methoden/3

	CMP Entity Bean	BMP Entity Bean
ejbPostCreate()	Direkt nach ejbCreate(). Setzen von CMR-Feldern.	Direkt nach ejbCreate()
ejbLoad()	Instanz soll Identität eines Datenbankrecords bekommen. Vor Beginn jeder Transaktion. Normalerweise leer.	Instanz soll Identität eines Datenbankrecords bekommen. Vor Beginn jeder Transaktion. Instanzvariablen müssen z.B. via JDBC aus der DB geladen werden.
ejbStore()	Identität der Instanz soll in DB gespeichert werden. Nach Ende jeder Transaktion. Normalerweise leer.	Identität der Instanz soll in DB gespeichert werden. Nach Ende jeder Transaktion. Instanzvariablen müssen z.B. via JDBC in die DB geschrieben werden.
ejbFindByPrimaryKey()	Wird nie aufgerufen.	Wenn Client findByPrimaryKey() aufruft. Muss sicherstellen dass entsprechender Datensatz existiert, braucht diesen aber nicht zu laden!
ejbFindXXX()	Wird nie aufgerufen.	Wenn Client findXXX() aufruft. Liefert Collection der Primary Key Klasse zurück.