

## 6.3 Interaktion mit dem Container (3)

---

- Methoden der Bean zum Finden von Entity-Beans
  - ◆ eine oder mehrere Finder-Methoden zur Ermittlung von Entity-Beans aus der Datenbank
  - ◆ Methoden **ejbFindXYZ()**
    - mindestens **ejbFindByPrimaryKey( ... )**
    - Rückgabewert: entweder Primary-Key-Objekt oder **Collection** von Primary-Key-Objekten
  - ◆ Methoden tauchen als **findXYZ()** im Home-Interface bzw. LocalHome-Interface auf
    - geben EJBObject bzw. EJBLocalObject oder **Collection** von EJBObjects bzw. EJBLocalObjects zurück
  - ◆ Finder-Methoden müssen **javax.ejb.FinderException** werfen können



## 6.3 Interaktion mit dem Container (4)

---

- Methoden im Entity-Kontextobjekt
  - ◆ wie bei Session-Kontext: **getEJBObject()**, **getEJBLocalObject()**
  - ◆ zusätzlich: **getPrimaryKey()**
    - Ermittlung des aktuellen Primary-Key-Objekts
    - Primary-Key-Objekt muss von **ejbCreate()** zurückgegeben werden



## 6.4 Bean-Managed Persistence

---

- Entity-Bean-Implementierung sorgt für Persistenz der Daten
  - ◆ Zugriff auf Datenbank über JDBC-Verbindung
    - Container legt `javax.sql.DataSource`-Objekt im JNDI-Namensraum ab (über Deployment-Deskriptor konfigurierbar)
  - ◆ automatisches Pooling von Datenbankverbindungen
    - Anforderung und Freigabe der Verbindung vor und nach allen CRUD-Operationen (**C**reate, **R**ead, **U**ppdate, **D**eleate)
    - Anforderung: Methode `getConnection()` an DataSource-Objekt
    - Freigabe: Methode `close()` am Verbindungsobjekt
  - ◆ Verändern und Lesen der Datenbank über SQL-Anweisungen mit JDBC-Methoden
    - Erzeugung von `PreparedStatement`-Objekten über Methode `prepareStatement()` am Verbindungsobjekt
    - Ausführung des übergebenen SQL-Statements durch Methodenaufrufe



## 6.4 Bean-Managed-Persistence (2)

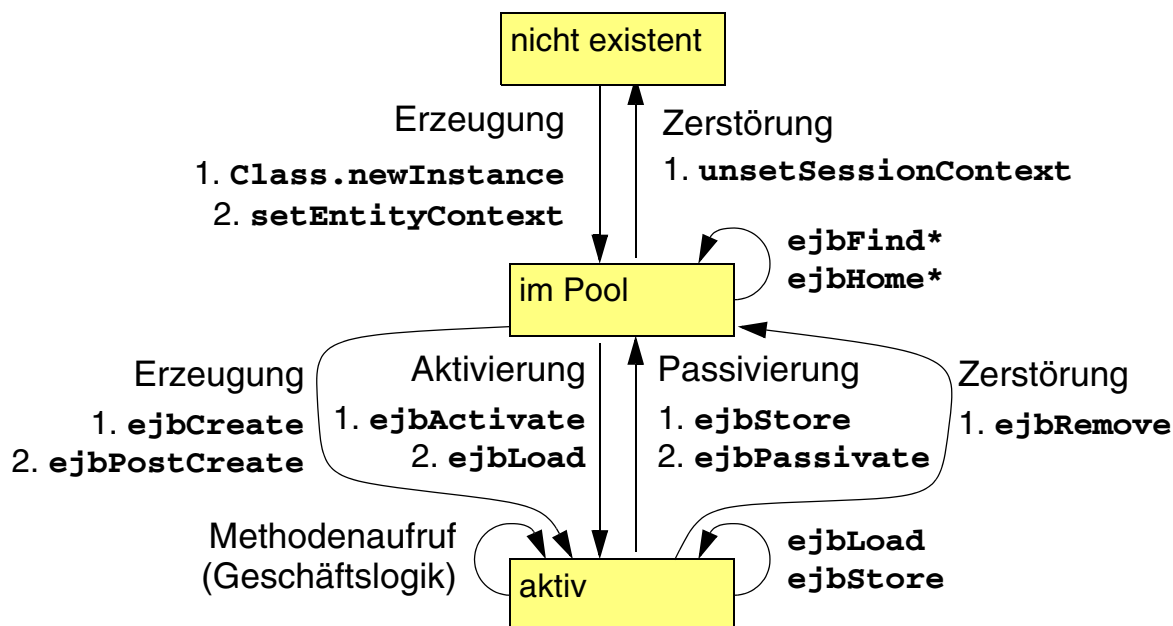
---

- Implementierung von Datenbankzugriffen in
  - ◆ `ejbCreate`-Methoden
  - ◆ `ejbRemove`-Methode
  - ◆ `ejbFind*`-Methoden
  - ◆ evtl. `ejbHome*`-Methoden, falls Zugriffe erforderlich sind
  - ◆ `ejbLoad`- und `ejbStore`-Methode
  
- ◆ Beispiel: Konto-Bean
  - `ejbCreate( AccountID id, String ownerName )`
  - `ejbFindByPrimaryKey, ejbFindByOwnerName`
  - `ejbHomeGetTotalBalance`



## 6.4 Bean-Managed Persistence (3)

### ■ Lebenszyklus einer BMP-Entity-Bean



## 6.4 Bean-Managed Persistence (4)

### ■ Deployment-Deskriptor

#### ◆ Beispiel

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>Account</ejb-name>
      <home>example.AccountHome</home>
      <remote>example.Account</remote>
      <local-home>example.AccountLocalHome</local-home>
      <local>example.AccountLocal</local>
      <ejb-class>example.AccountBean</ejb-class>
      <persistence-type>Bean</persistence-type>
      <prim-key-class>example.AccountPK</prim-key-class>
      <reentrant>False</reentrant>
    ...
```

#### ◆ rekursive Aufrufe

- Bean kann über andere Beans sich selbst aufrufen (`reentrant = True`)



## 6.4 Bean-Managed Persistence (5)

### ■ Deployment-Descriptor (fortges.)

```
<resource-ref>
  <res-ref-name>jdbc/ejbPool</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
</entity>
</enterprise-beans>
```

```
<assembly-descriptor>
  <container-transactions>...</container-transactions>
</assembly-descriptor>
</ejb-jar>
```

- ◆ Angabe zum Finden der JDBC-Ressourcen
- ◆ Angaben zu Transaktionen



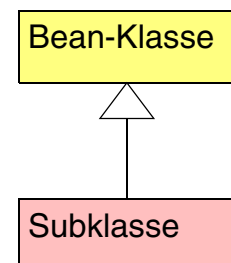
## 6.5 Container-Managed Persistence

### ■ Neuheit seit EJB 2.0

- ◆ Container kümmert sich um Datenbankbindung
  - CMP = Container-Managed Persistence
- ◆ starke Vereinfachung der Bean-Entwicklung

### ■ Implementierung der Datenbankbindung

- ◆ Bereitstellung einer abstrakten Entity-Bean-Klasse durch Entwickler
- ◆ Container stellt generierte, herstellerspezifische Subklasse bereit
  - Subklassen-Instanz ist eigentliche Bean-Instanz



## 6.5 Container-Managed Persistence (2)

---

- Abstrakte Bean-Klasse des Bean-Entwicklers
  - ◆ Weglassen aller Datenbankzugriffe
  - ◆ Weglassen aller persistenten Datenfelder
  - ◆ Deklaration von abstrakten **set\*/get\***-Methoden für alle Datenfelder
  - ◆ Verwendung dieser Methoden zum Datenzugriff in Businesslogik
  - ◆ Weglassen aller Implementierungen von **ejbFind\***-Methoden
    - bleiben aber im Home-Interface erhalten
  - ◆ Deklaration von abstrakten **ejbSelect\***-Methoden
    - nur bean-intern verwendbar
    - erlauben die Selektion bestimmter Daten aus der Datenbank (nicht
    - z.B. notwendig für Implementierung von **ejbHomeGetTotalBalance** des Konto-Beispiels (z.B. **ejbSelectTotalBalance**)



## 6.5 Container-Managed Persistence (3)

---

- Abstrakte Bean-Klasse des Bean-Entwicklers (fortges.)
  - ◆ Implementierung der **ejbHome\***-Methoden
    - keine direkten Datenbankzugriffe
    - stattdessen Nutzung der **ejbSelect\***-Methoden
  - ◆ Implementierung von **ejbCreate**-Methoden
    - aufgerufen durch entsprechende Methode aus der Subklasse
    - nicht unbedingt notwendig, wenn keine Datenerzeugung benötigt
    - kein Datenbankzugriff
    - stattdessen setzen der Daten durch **set\***-Methoden
    - Container fügt anschließend Daten in die Datenbank ein
    - ähnliches Vorgehen für **ejbPostCreate**-Methoden
  - ◆ Implementierung von **ejbActivate** und **ejbPassivate**
    - gleiche Funktion wie bei BMP-Beans



## 6.5 Container-Managed Persistence (4)

---

- Abstrakte Bean-Klasse des Bean-Entwicklers (fortges.)
  - ◆ Implementierung von **ejbLoad** und **ejbStore**
    - keine Datenbankzugriffe
    - Container hat vor **ejbLoad**-Aufruf Daten geladen
    - Container speichert nach **ejbStore**-Aufruf Daten ab
  - ◆ Implementierung von **ejbRemove**
    - keine Datenbankzugriffe
    - Container löscht Daten nach dem Aufruf



## 6.5 Container-Managed Persistence (5)

---

- Vom Container generierte Subklasse
  - ◆ implementiert **set \*/get \*-**Methoden für alle persistenten Entity-Daten
    - Namen kommen aus dem Deployment-Deskriptor
    - Typen kommen aus der Bean-Klasse (abstrakte Methodendeklarationen)
  - ◆ fängt **ejb\*-**Methoden ab, um Datenbankzugriffe durchzuführen
    - ruft meist **ejb\*-**Methoden der Bean-Klasse des Entwicklers auf
  - ◆ implementiert **ejbFind\*-**Methoden
    - Abfrageinformation kommt aus dem Deployment-Deskriptor
  - ◆ implementiert **ejbSelect\*-**Methoden
    - Abfrageinformation kommt aus dem Deployment-Deskriptor



## 6.5 Container-Managed Persistence (6)

### ■ Deployment-Deskriptor

#### ◆ Beispiel

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      ...
      <persistence-type>Container</persistence-type>
      ...
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>AccountBean</abstract-schema-name>
      <cmp-field>
        <field-name>accountID</field-name>
      </cmp-field>
      ...
      <prim-key-field>accountID</prim-key-field>
      ...
```

#### ◆ zu allen Feldern muss es `get*/set*`-Methoden geben



## 6.5 Container-Managed Persistence (7)

### ■ Deployment-Deskriptor (fortges.)

#### ◆ Deskriptor enthält die persistenten Felder der Entity-Bean

### ■ Container-abhängiges Deployment

#### ◆ Abbildung der Felder auf Datenbankdaten

- z.B. Abbildung auf Spalten einer Tabelle in relationaler Datenbank
- Abbildung herstellerabhängig zu konfigurieren

### ■ Beschreibung der `ejbFind*`- und `ejbSelect*`-Methoden

#### ◆ EJB-QL (Query-Language, Abfragesprache)

#### ◆ abstrakte Datenbankzugriffssprache

- unabhängig von konkreter Datenbankanbindung
- leicht in SQL-Anfragen überführbar



## 6.5 Container-Managed Persistence (8)

- Beispiel für EJB-QL-Beschreibung im Deployment-Descriptor

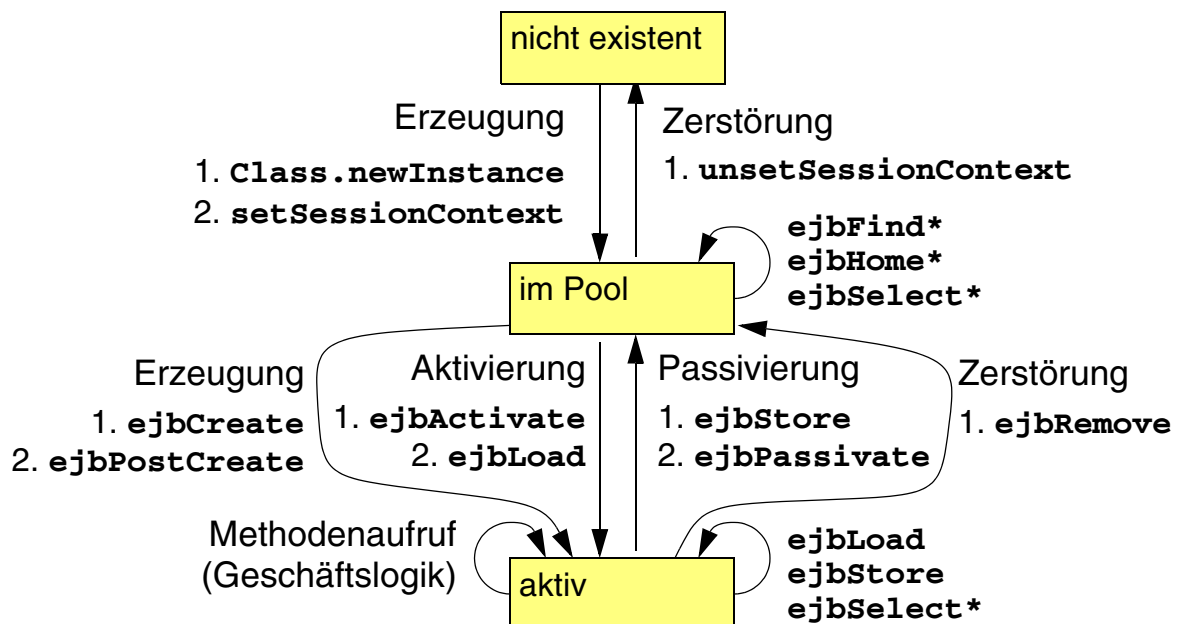
```
<ejb-jar>
  <enterprise-beans>
    <entity>
      ...
    <query>
      <query-method>
        <method-name>findByName</method-name>
        <method-params>
          <method-param>java.lang.String</method-param>
        </method-params>
      </query-method>
    <ejb-ql>
      <![CDATA[SELECT OBJECT(a) FROM AccountBean AS a
        WHERE name = ?1]]>
    </ejb-ql>
  </query>
  ...
```

- ◆ Angabe der Abfragen für jede `ejbFind*`- und `ejbSelect*`-Methode



## 6.5 Container-Managed Persistence (9)

- Lebenszyklus einer CMP-Entity-Bean



## 7 Message-Driven Bean

---

- Schnittstelle zu nachrichtenbasierten Systemen
  - ◆ MOM (Message-oriented Middleware)
  - ◆ JMS (Java Message Service)
    - Schnittstelle zu allen möglichen Nachrichtensystemen
  
- Message-Driven Bean
  - ◆ Enterprise-Java-Bean, die Nachrichten entgegen nehmen kann
  - ◆ kein Home-Interface
  - ◆ eine Methode: `onMessage ( )`
  - ◆ zustandslose Weiterverarbeitung der Nachricht
    - bedingte, verarbeitete Weiterleitung der Nachricht an andere Beans (typisch Session-Beans)



## 8 Bean-Referenzen

---

- Referenzen zwischen Beans
  - ◆ Herstellung wie von Client-Seite
    - Namensauflösung über JNDI
    - create() am Home-Interface ausführen
    - RMI-IIOP-Referenz oder lokale Referenz auf EJBObject oder EJBLocalObject zurückbekommen
  - ◆ Deklaration im Deployment-Descriptor möglich
    - Angabe eines Namens im JNDI-Namensraum
      - kann vom Container über symbolische Links auf echten Namen umgebogen werden
    - Angabe von Bean-Klasse und -typ



## 8 Bean-Referenzen (2)

---

- Langlebige Referenzen auf Beans
  - ◆ z.B. Abbruch der RMI-IIOP Verbindung
  - ◆ Umwandlung von EJBObject-Referenzen in EJB-Object-Handles
    - Aufruf von `getHandle()`
  - ◆ Handle kann serialisiert werden
  - ◆ Umwandlung einer Handle in EJBObject
    - Aufruf von `getEJBObject()`
    - anschließendes Narrow notwendig
- Langlebige Referenzen auf Home-Objects
  - ◆ selbe Prozedur nur für Home-Handles
- Handles sind nicht portabel von Container zu Container



## 9 Sicherheit

---

- Authentisierung
  - ◆ Feststellung der Identität
- Autorisierung
  - ◆ Feststellung der Erlaubnis für eine Aktion
    - benötigt typischerweise Authentisierung
- Nutzung von JAAS (Java Authentication and Authorization API)
  - ◆ komplexes Rahmenwerk zur Übermittlung von Authentisierungskontexten bei Aufrufen
    - transparenter Sicherheitskontext im Rahmen von IIOP-Kommunikation
    - EJB-Container kann Aufrufe einem Subjekt (Principal) zuordnen



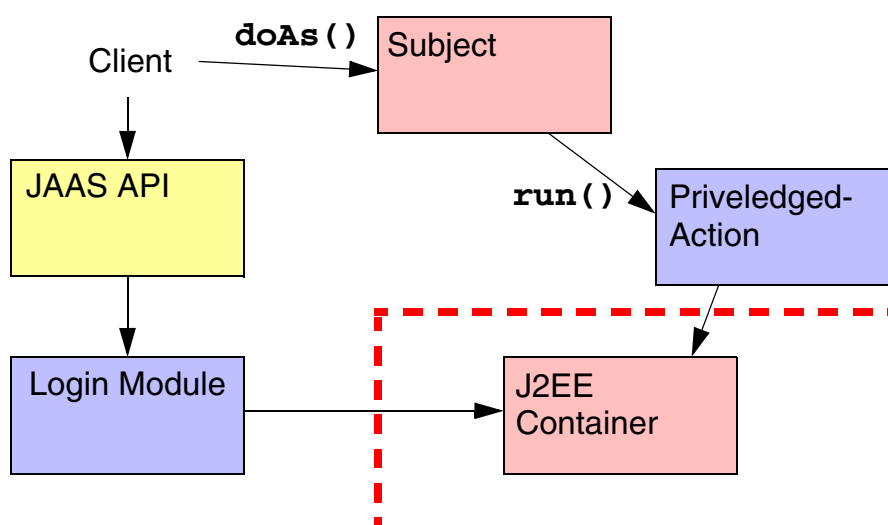
## 9.1 Authentisierung

- **Subject-Objekte** von JAAS
  - ◆ enthalten Informationen über authentisierten Principal
  - ◆ bekommt man als Ergebnis der Authentisierung
- **Aktionsobjekte**
  - ◆ leiten von `PrivilegedAction` ab
  - ◆ implementieren eine `run()`-Methode
    - z.B. Aufruf von `sayhello()` an einer neuen Hello-Bean
- **Ausführung einer privilegierten Aktion**
  - ◆ Aufruf einer statische Methode:  
`Subject.doAs( Subject s, PrivilegedAction a )`
  - ◆ Rückgabewert als `java.lang.Object`



## 9.1 Authentisierung (2)

- Schematischer Ablauf



- ◆ Subject-Objekt wird nach Authentisierung erzeugt



## 9.1 Authentisierung (3)

---

- Verschiedene Verfahren zur Erzeugung von Subject-Objekten
  - ◆ Angabe von X.509-Zertifikaten
    - zertifizierte Public-Keys
  - ◆ Angabe von Benutzernamen und Passwort
    - Verifikation des richtigen Passworts
    - z.B. über das Web
  - ◆ Ermittlung von Benutzer-IDs aus dem Betriebssystem
  
  - ◆ JAAS-interne Login-Module verifizieren Benutzerinformationen und erzeugen Subject-Objekt
    - proprietäres Protokoll zum dem Container
    - Erzeugung und Rückgabe von Subject-Objekten



## 9.2 Autorisierung

---

- Zwei Arten
  - ◆ programmatische Autorisierung
    - Bean-Managed Authorization
  - ◆ deklarative Autorisierung
    - Container-Managed Authorization
  
- Sicherheitsrollen
  - ◆ Aufrufe erfordern vom Principal das Innehaben einer Rolle
    - Unabhängigkeit von konkreten Personen bzw. Principals
  - ◆ externe Zuordnung von Personen zu Rollen
    - Zuordnung im container-spezifischem Deployment



## 9.3 Programmatische Autorisierung

- Ermittlung der Principal-Informationen
  - ◆ Aufruf am EJBContext-Objekt von `getCallerPrincipal()`
- Ermittlung der Rollenzugehörigkeit
  - ◆ Aufruf am EJBContext-Objekt von `isCallerInRole( String role )`
  - ◆ dynamische Zugriffsentscheidung
- Definition der erforderlichen Rolle im Deployment-Descriptor

```
...
<session>
...
<security-role-ref>
  <description>...</description>
  <role-name>administrators</role-name>
  <role-link>admins</role-link>
</security-role-ref>
...
```



## 9.3 Programmatische Autorisierung (2)

- Abbildung der Rolle innerhalb der Bean an reale Anwendungsrolle
  - ◆ z.B. Abbildung der Rolle `administrators` auf die Rolle `admins`
  - ◆ Definition der realen Rolle im Deployment-Descriptor

```
...
<assembly-description>
...
<security-role>
  <description>...</description>
  <role-name>admins</role-name>
</security-role>
</assembly-description>
...
```

- im Abschnitt zur Komposition der Anwendung



## 9.4 Deklarative Autorisierung

---

- Deklaration der Aufruferlaubnis im Deployment-Descriptor

```
<assembly-description>
...
<method-permission>
  <role-name>admins</role-name>
  <method>
    <ejb-name>Account</ejb-name>
    <method-name>*</method-name>
  </method>
  <method>
    <ejb-name>Hello</ejb-name>
    <method-name>sayHello</method-name>
    <method-params>void</method-params>
  </method>
</method-permission>
...
</assembly-description>
...
```

- ◆ zusätzlich Deklaration der Rollen wie bei programmatischer Autorisierung



## 10 Transaktionen

---

- Motivation
  - ◆ Konsistenz von Daten
    - bei Fehlern aller Art
    - bei nebenläufigem und parallelem Zugriff
- Transaktionen
  - ◆ fassen mehrere Teilaktionen zu einer atomaren Operation zusammen
    - entweder alles ausgeführt oder nichts (Atomicity)
  - ◆ Fehler führen zu einem Rückfahren der bisherigen Effekt (*Roll-Back*)

