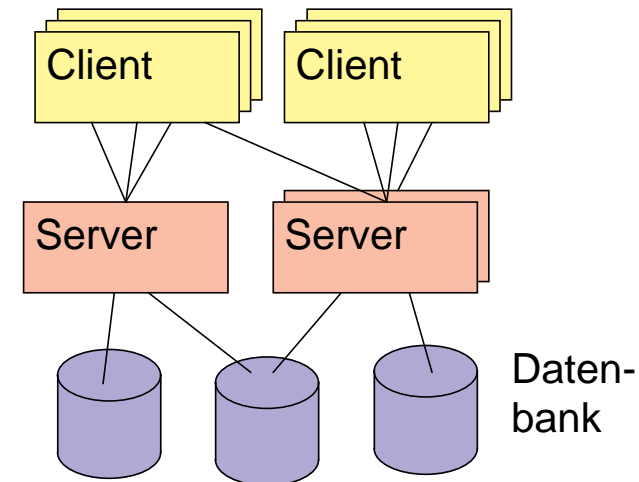


# J Enterprise Java Beans

## J.1 Motivation

### ■ Große verteilte Anwendungen im „Geschäftsleben“

- ◆ viele Clients
  - wollen Dienste nutzen
- ◆ einige Server
  - stellen Dienste bereit
- ◆ einige Datenbanken
  - halten die Geschäftsdaten



### ■ Problem

- ◆ Aufbau des Systems
  - ◆ Zergliederung in Einzelteile
  - ◆ Kommunikation der Teile
- ➔ Middleware

## J.1 Motivation (2)

---

### ★ Komponenten-Idee

- ◆ Zerlegung der Geschäftslogik („Business-Logik“) in Komponenten
  - z.B. Komponente zur Preisberechnung
- ◆ Wiederverwendung von Komponenten
  - Komponentenmarkt mit Komponentenanbietern
  - Zusammenschalten von neuen und eingekauften Komponenten zu einer neuen Anwendung
- ◆ Bereitstellung einer Umgebung für Komponenten
  - Umgebung unterstützt Sicherheit
  - Umgebung unterstützt Anwendungskonsistenz durch Transaktionen

### ■ Application Server

- ◆ Umgebung für Komponenten
  - Menge von Komponenten bilden eine Anwendung

## J.2 Architektur

■ Typisch: Architektur aus mehreren Schichten (*Multitiered Architecture*)

◆ Client-Tier

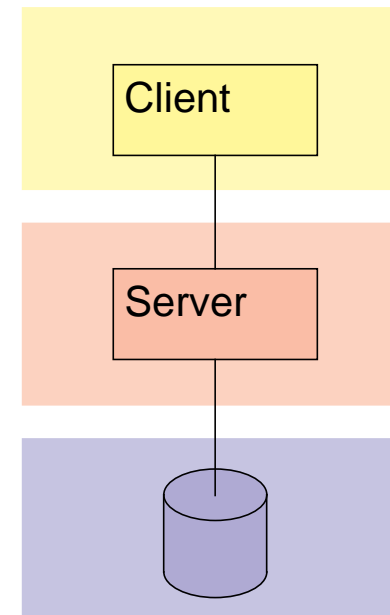
- Anwendungsteil des Client
- Webbrowser
- dedizierte Anwendung

◆ Middle-Tier

- Geschäftslogik
- Service-Bereitstellung

◆ EIS-Tier (Enterprise Information System)

- Datenbank
- Archiv der Geschäftsvorgänge



## J.2 Architektur (2)

---

- Client-Tier
  - ◆ Webbrowser als Client-Anwendung
    - Zugriff auf dynamische Webseiten  
(z.B. GMX, Webshop, Hotelreservierung)
    - Webseiten mit Applets  
(Applet-Programm tritt als Client zur Anwendung auf, z.B. Homebanking)
  - ◆ dedizierte Client-Anwendung
    - kommuniziert mit dem Rest der Anwendung
  - ◆ Web-Services-Schnittstelle
  - ◆ Benutzeroberfläche zur Anwendung
  - ◆ lokale Berechnung/Verarbeitung

## J.2 Architektur (3)

---

### ■ Middle-Tier

#### ◆ Web-Tier

- Web-Container für Java Server Pages oder Servlets
- (CGI-Skript)
- unnötig bei dedizierter Client-Anwendung

#### ◆ Business-Tier

- enthält eigentliche Geschäftslogik
- Einsatz von Geschäfts-Komponenten
- Komponenten-Container

#### ◆ Verarbeitung von Geschäftsprozessen und Geschäftsdaten

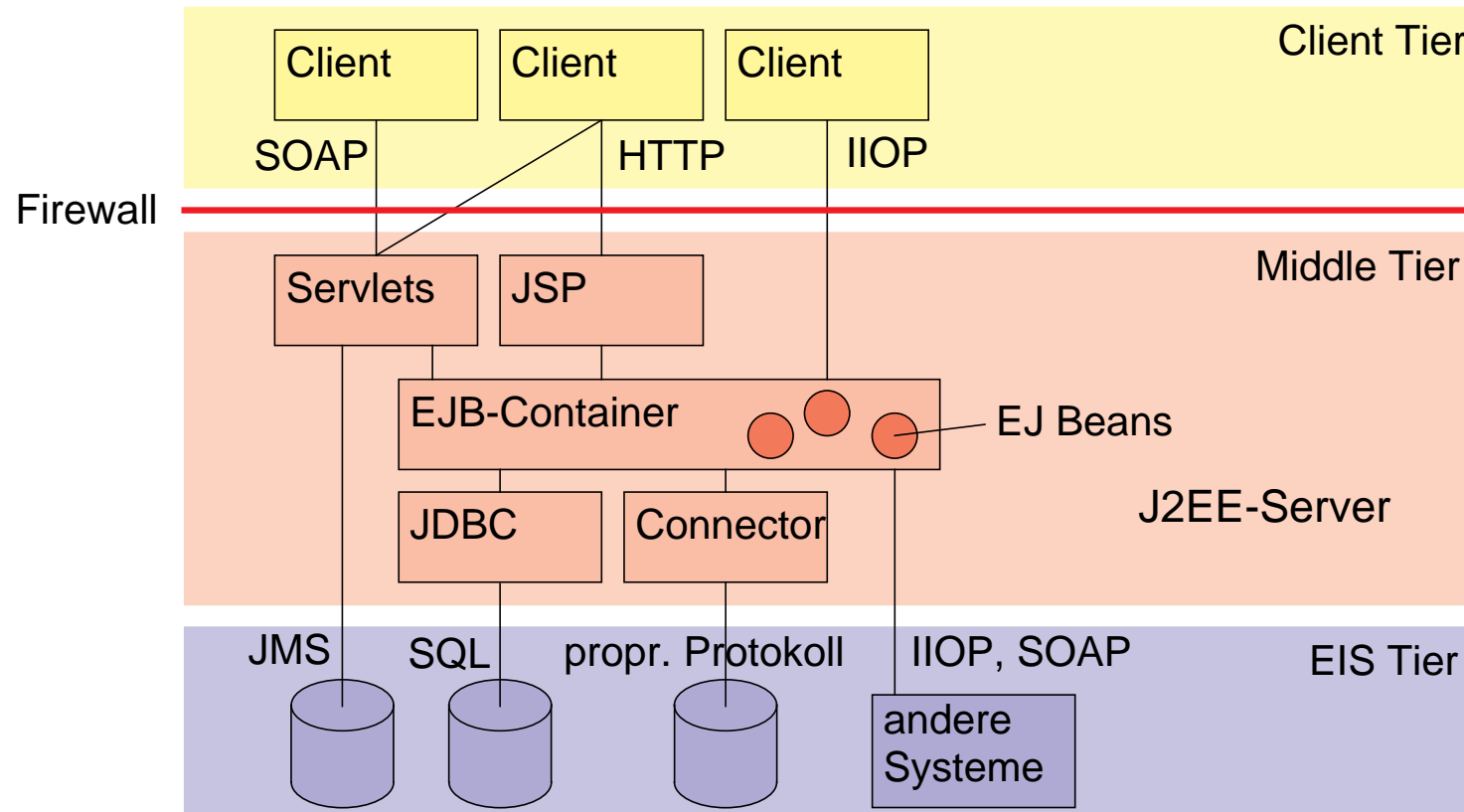
## J.2 Architektur (4)

---

- EIS-Tier
  - ◆ Datenbanksysteme
    - relationale Datenbanken
    - objektorientierte Datenbanken
  - ◆ Altanwendungen zum Zugriff auf Geschäftsdaten
  - ◆ Datenverwaltung von Geschäftsdaten
    - Konsistenz der Daten

# 1 EJB-Architektur

## ■ Globales Bild einer EJB-Anwendung



# 1 EJB-Architektur (2)

---

## ■ Beispielanwendungen

### ◆ Bankanwendung

- Client am Webbrowser
- Application-Server erlaubt Ansicht des Kontoauszug, Beauftragung für Überweisung, Dauerauftrag etc.
- Datenbanken im Hintergrund halten Buchungen und Kontostände sowie Benutzerdaten

### ◆ Webshop

- Client am Webbrowser
- mehrere Application-Server für Kreditkartenzahlung, Produktkatalog, Kundenprofilverwaltung

## 2 Rollen von EJB

---

- Bean-Entwickler
  - ◆ Komponentenverkäufer im Komponentenmarkt
  - ◆ Entwicklungsabteilung
  - ◆ ... liefern Enterprise-Java-Bean
    - d.h. Java Klassen gemäß EJB-Spezifikation für Komponenten
  
- Anwendungsentwickler
  - ◆ Entscheidung über Komponenteneinsatz (Zukauf, Eigenentwicklung)
  - ◆ Verbindungscode zwischen Komponenten
  - ◆ Entwicklung der Benutzerschnittstelle (JSP, Servlet, Applet)

## 2 Rollen von EJB (2)

---

- **Anwendungsinstallateur (Deployer)**
  - ◆ Aufstellen der Hardware für Application-Server
    - Stichworte: Redundanz und Fehlertoleranz
  - ◆ Verteilung der Beans auf Application-Server
  - ◆ Sicherung der Kommunikation durch Firewalls
  - ◆ Integration in Infrastruktur
    - Stichwort: Verknüpfung von Zugriffsrechten mit aktuellen Benutzern
  - ◆ Performance-Tuning
  
- **Systemadministratoren**
  - ◆ Betrieb der Anwendung
    - Managementfunktion
  - ◆ Überwachung der Anwendung
    - Monitoring, Fehlerbehebung

## 2 Rollen von EJB (3)

---

### ■ Application-Server-Anbieter

#### ◆ Bereitstellen des Bean-Containers

- Behausung für Enterprise Java Beans
- Unterstützung für Sicherheit, Transaktionen etc.

#### ◆ Beispiele

- |                       |                            |
|-----------------------|----------------------------|
| • WebLogic (BEA)      | Bluestone (HP)             |
| • iPlanet             | iPortal (IONA)             |
| • Websphere (IBM)     | Borland Application Server |
| • Oracle 9i           | JBoss (Open Source)        |
| • JRun (Macromedia)   | Powertier (Persistence)    |
| • Gemstone/J (Brokat) |                            |

### ■ Werkzeuganbieter

- ◆ Werkzeuge für die Code-Entwicklung (z. B. Visual Age, Eclipse)
- ◆ Werkzeuge zur Modellierung und Code-Erzeugung

### 3 Unterschied zu klassischer Middleware

---

■ Klassische Middleware ist explizit

◆ Middleware: CORBA, Java RMI

◆ Beispiel: Überweisungsvorgang von Konto zu Konto

```
account1.transfer( Amount s, Account other );
```

◆ notwendiger Code im Kontoobjekt

- Aufruf eines Sicherheitsservice, ob Aufrufer berechtigt
- Aufruf eines Transaktionsservice zum Start einer Transaktion
- Aufruf eines Datenbankservers zum Laden von Kontoinformationen
- lokale Kontostandsberichtigung
- Aufruf des zweiten Kontos zur Kontostandsberichtigung
- Aufruf des Datenbankservers zum Speichern der Kontoinformationen
- Aufruf des Transaktionsservice zum Beenden der Transaktion

### 3 Unterschied zu klassischer Middleware (2)

---

#### ▲ Problem

- ◆ komplexe Programmierung
- ◆ schwierige Wartung
- ◆ Interaktion verschiedener Produkte unter Umständen problematisch
  - z.B. Datenbankserver und Transaktionsdienst

#### ★ Vorteil

- ◆ hohe Flexibilität

### 3 Unterschied zu klassischer Middleware (3)

---

#### ■ Implizite Middleware wie bei EJB

##### ◆ Beispiel: Überweisungsvorgang von Konto zu Konto

```
account1.transfer( Amount s, Account other );
```

##### ◆ notwendiger Code in Bean

- lokale Kontostandsberichtigung
- Aufruf einer zweiten Bean zur Kontostandsberichtigung des anderen Kontos

##### ◆ Interaktion mit Services erfolgt implizit

- Container fängt Interaktionen ab

##### ◆ Beschreibung der Interaktion in der Deployment-Phase

- Deployment-Deskriptor (XML)

### 3 Unterschied zu klassischer Middleware (4)

---

#### ★ Vorteil

- ◆ einfach zu entwickelnden Beans
- ◆ leichte Wartung da übersichtlicher Code
- ◆ gesichertes Zusammenspiel der Komponenten

#### ▲ Nachteil

- ◆ weniger flexibel
- ◆ im Fehlerfall weniger durchschaubar
  - abhängig vom Reifegrad der Produkte

### 3 Unterschied zu klassischer Middleware (5)

---

- EJB bietet implizite Unterstützung für
  - ◆ verteilte Transaktionen
    - Abbruch oder Bestätigung der Ergebnisse einer Transaktion
    - Koordinierung nebenläufiger Aktionen
  - ◆ Sicherheitsdienst
    - Zugriffskontrolle
  - ◆ Ressourcen- und Life-Cycle-Kontrolle
    - Container verwaltet teilweise Bean-Lebenszyklus
  - ◆ Persistenz
    - automatisches Sichern persistenter Daten z.B. in Datenbanken
  - ◆ Monitoring
    - Container kann Last und Aufrufhäufigkeiten erfassen
  - ◆ entfernte Aufrufe
  - ◆ Ortstransparenz
    - wie klassische Middleware

## J.3 EJB-Grundlagen

---

### ■ Verschiedene Bean-Typen

#### ◆ Session-Bean

- Modellierung von Geschäftsprozessen (implementieren Anwendungslogik)
- kurzlebig, nur ein Client
- agieren wie Verben (repräsentieren mögliche Aktionen)  
z.B. „überweisen“, „autorisieren“
- interagieren typischerweise mit Entity-Beans und Session-Beans

#### ◆ Entity-Bean

- Modellierung von Geschäftsdaten
- langlebig, Nutzung durch mehrere Clients
- agieren wie Substantive (repräsentieren Daten aus der Datenbank)  
z.B. „Konto“, „Kreditkarte“, „Produkt“

#### ◆ Message-Driven-Bean

- ähnlich Session-Bean
- ansprechbar über Nachrichten

# 1 Bean-Klassen

---

- Beans werden durch Java-Klassen repräsentiert
  - ◆ müssen bestimmte Java-Interfaces implementieren
  
- Alle Beans
  - ◆ implementieren Marker-Interface: `javax.ejb.EnterpriseBean`
  - ◆ markiert Bean gleichzeitig als serializable
  
- Einzelne Bean-Typen
  - ◆ implementieren jeweils Typ-Interfaces: `javax.ejb.SessionBean`,  
`javax.ejb.EntityBean`, `java.ejb.MessageDrivenBean`

## 2 Interaktion mit Beans

---

### ■ Keine direkte Interaktion

#### ◆ Bean-Instanzen sind nicht direkt ansprechbar

- implizite Middleware-Aktionen erfordern ein unbedingtes Abfangen von Aufrufen

### ■ Repräsentant für eine Bean-Instanz ist das EJBObject

#### ◆ implementiert ein (entferntes) Bean-Interface

- Bean-Interface muss von `javax.ejb.EJBObject` erben
- dieses implementiert `java.rmi.Remote`
- deklariert alle Methoden der Geschäftslogik

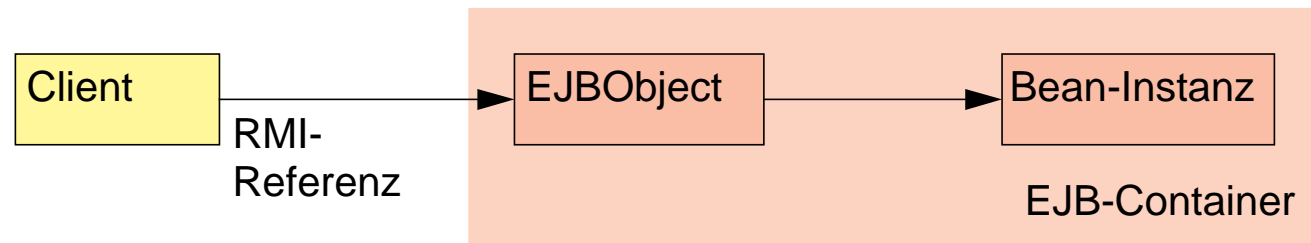
#### ◆ Implementierung des EJBObject herstellerspezifisch

#### ◆ Clienten rufen Bean über ein EJBObject auf

- entfernte Aufrufe über RMI bzw. RMI-IIOP möglich

## 2 Interaktion mit Beans (2)

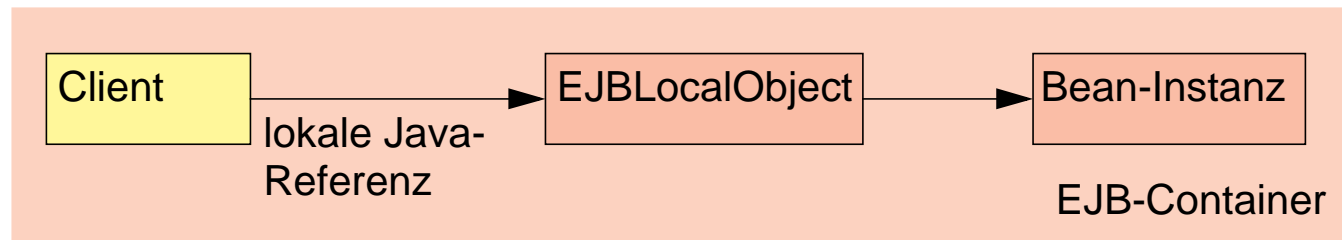
- EJBObject fängt Aufrufe an der Bean ab



- ◆ führt implizite Middleware-Interaktionen durch
  - Sicherheitsüberprüfung, Transaktionsverwaltung, Datenbankabfragen ...
- ◆ Interaktion mit EJBObject über RMI bzw. RMI-IIOP
  - Interaktion im lokalen Fall teuer (Marshalling und Demarshalling, lokaler Nachrichtentransport etc.)

## 2 Interaktion mit Beans (3)

- Lokaler Repräsentant für eine Bean-Instanz ist das EJBLocalObject
  - ◆ implementiert ein lokales Bean-Interface
    - Bean-Interface muss von `javax.ejb.EJBLocalObject` erben
    - deklariert alle Methoden der Geschäftslogik
  - ◆ Implementierung des EJBLocalObject herstellerspezifisch
  - ◆ Clienten rufen Bean über ein EJBLocalObject auf
    - kein entfernter Aufruf möglich
- EJBLocalObject fängt Aufrufe an der Bean ab



- ◆ auch hier: implizite Interaktion mit der Middleware

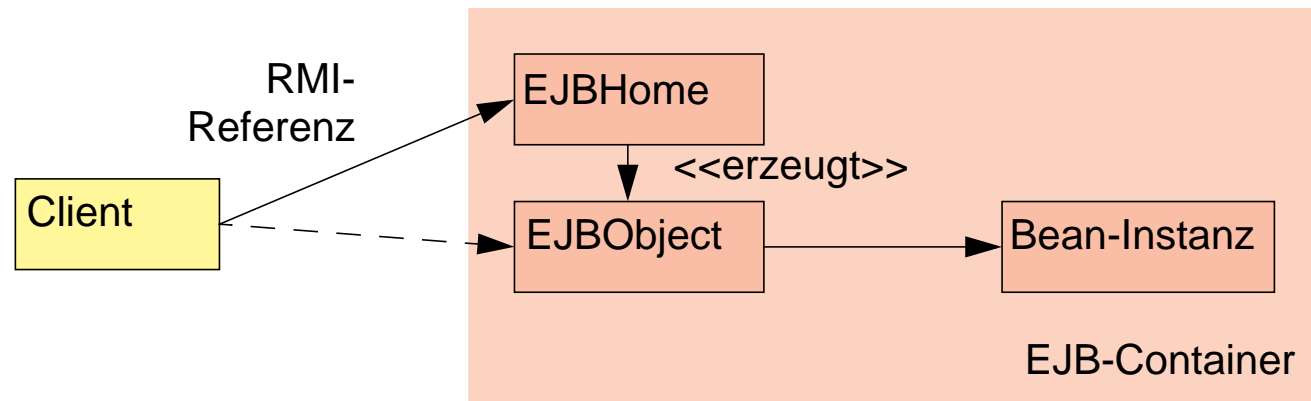
## 3 Erzeugung von Beans

---

- Eigentlich Erzeugung von EJBObjects bzw. EJBLocalObjects
  - ◆ Erzeugung der Bean-Instanz erfolgt implizit durch die Middleware bzw. den EJB-Container
  
- Erzeugung über Factory-Pattern
  - ◆ Schnittstelle zur Factory heißt Home-Object
  
- Repräsentant für ein Home-Object
  - ◆ implementiert ein entferntes oder lokales Home-Interface
    - Home-Interface muss von `javax.ejb.EJBHome` bzw. `java.ejb.EJBLocalHome` erben
    - ersteres implementiert `java.rmi.Remote`, letzteres nicht
    - deklariert Methoden zur Bean-Erzeugung, z. B. `create()`
  - ◆ Implementierung des Home-Object herstellerspezifisch
  - ◆ Finden des Home-Object durch Namensdienst (typisch über JNDI)

### 3 Erzeugung von Beans (2)

#### ■ Beispiel: entferntes Home-Object



- ◆ Erzeugung des EJBObject durch Aufruf der create()-Methode am EJBHome (z.B. über RMI-IIOP)
- ◆ Rückgabe der Referenz auf das EJBObject

## 4 Verwaltung des Lebenszyklus

---

- Klienten interagieren nur mit EJBObject- bzw. EJBLocalObject- und EJBHome- bzw. EJBLocalHome-Objekten
  - ◆ d. h. nur mit herstellerspezifischen Objekten des EJB-Containers
  
- Lebenszyklus der EJBObjects bzw. EJBLocalObjects
  - ◆ explizite Methode `remove()`
  - ◆ muss vom Client aufgerufen werden, falls Referenz nicht mehr benötigt wird
  
- Lebenszyklus der Bean-Instanz
  - ◆ völlig unabhängig vom Lebenszyklus der EJBObjects
    - Bean kann erst bei Aufruf erzeugt werden
    - Bean kann „gepoolt“ werden (Wiederverwendung „gebrauchter“ Beans)
    - Aufgabenwechsel für Bean-Instanzen während der Laufzeit (dynamische Zuordnung an verschiedene EJBObjects)

## 4 Verwaltung des Lebenszyklus (2)

---

- Zuordnung EJBObjects zu Bean-Instanzen nicht unbedingt 1:1
  - ◆ Erzeugung über Home-Interface benutzt u. U. Bean-Instanz wieder
    - z. B. Entity-Bean für bestimmtes Konto
  - ◆ mehrere EJBObjects pro Bean-Instanz möglich
    - z. B. so viele wie Clients eine Referenz zu einer Entity-Bean erzeugt haben
  - ◆ gepoolte Instanzen implementieren alle referenzierten Beans (EJBObjects)

## 5 Bean-Interaktion mit dem Container

---

- Interaktion mit Container bisher nur implizit
- Explizite Interaktion über Context-Objekt
  - ◆ Container übergibt bei Bean-Instanzerzeugung Context-Objekt
    - Methoden zum Ermitteln der Home-Objects (lokal u. entfernt)
    - Methoden zum Transaktionsdienst (z.B. ermittle Transaktionsinformationen)
    - Methoden zum Sicherheitsdienst (z.B. hole Aufruferinformationen)

## 6 Beispiel

---

### ■ Session-Bean für Hello-World

#### ◆ Java-Klasse für Bean

- z.B. `example.HelloBean`
- implementiert `javax.ejb.SessionBean`
- implementiert einige vorgegebene Methoden
  - `ejbCreate()`: Aufruf bei Erzeugung der Instanz
  - `ejbRemove()`: bei Zerstörung der Instanz
  - `ejbPassivate()`: bei Passivierung der Instanz
  - `ejbActivate()`: bei Aktivierung der Instanz
  - `setSessionContext()`: bekommt Session-Context-Object vom Container
- fügt `sayHello`-Methode hinzu

## 6 Beispiel (2)

---

- ◆ lokales und entferntes Bean-Interface für das EJBObject
  - z.B. `example.Hello` und `example.HelloLocal`
  - implementiert `javax.ejb.EJBObject` bzw. `EJBLocalObject`
  - fügt `sayHello`-Methode hinzu
- ◆ lokales und entferntes Home-Interface für Home-Object
  - z.B. `example.HelloHome` und `example.HelloLocalHome`
  - implementiert `java.ejb.EJBHome` bzw. `EJBLocalHome`
  - fügt `create`-Methode hinzu
  
- ◆ Kompilation der Java-Sourcen

## 6 Beispiel (3)

---

### ■ Hinzufügen eines Deployment-Descriptors

#### ◆ XML-File

#### ◆ Beispiel

```
<ejb-jar>
  <enterprise-beans>
    <sessions>
      <ejb-name>Hello</ejb-name>
      <home>example.HelloHome</home>
      <remote>example.Hello</remote>
      <local-home>example.HelloLocalHome</local-home>
      <local>example.HelloLocal</local>
      <ejb-class>example.HelloBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </sessions>
  </enterprise-beans>
</ejb-jar>
```

## 6 Beispiel (4)

---

### ■ Descriptorinhalt

#### ◆ Name (Nickname) für die Bean

- wird für den Eintrag des Home-Objects im Namensdienst verwendet

#### ◆ Benennung der Interfaces und der Bean-Klasse

#### ◆ Angaben zur impliziten Middleware-Interaktion

- hier: zustandslose Session-Bean  
(wird für Lebenszyklusverwaltung verwendet)
- hier: Container-basierte Transaktionsverwaltung  
(Container kümmert sich um Transaktion pro Aufruf)

### ■ Class-Files plus Descriptor

#### ◆ Zusammenpacken zu einem jar-File

#### ◆ „verkaufbare“ EJB-Komponente

## 7 Deployment

---

- Installation einer Komponente stark herstellerabhängig
- Vorfeld
  - ◆ Integration des jar-Files in den Application-Server / EJB-Container
  - ◆ Überprüfung der Konsistenz durch Werkzeuge
    - Passen Interfaces zur Bean-Klasse?
    - Sind die notwendigen Methoden implementiert?
  - ◆ Werkzeuge erzeugen EJBObject, EJBLocalObject, EJBHome- und EJBLocalHome-Objekte
  - ◆ Werkzeuge erzeugen RMI-IIOP-Stubs und -Skeletons für EJBObject und EJBHome-Objekt
- Eigentliches Deployment
  - ◆ veranlasse EJB-Container die Bean zu installieren

## 8 Interaktion mit der Bean

---

- Clients müssen folgende Schritte durchführen
  - ◆ JNDI anfragen (z. B. nach „Hello“)
  - ◆ das von JNDI gelieferte Objekt muss mittels `Narrow` in einen Stellvertreter vom Typ `example.HelloHome` gewandelt werden
  - ◆ Aufruf von `create()` gibt Objektreferenz auf Stellvertreter für ein `EJBObject` der Hello-Bean zurück
  - ◆ Aufruf von `sayHello()` am `EJBObject`

## J.4 Einordnung

---

- Basis RMI-IIOP/CORBA
- Unterstützung nichtfunktionaler Eigenschaften
  - ◆ Effizienz und Ressourcenverwaltung
    - Abkopplung der Lebenszeit von Bean-Instanzen von der Lebenszeit der Bean
  - ◆ Konsistenz und Nebenläufigkeit
    - Transaktionskonzept
  - ◆ Sicherheit
    - Sicherheitskonzept
- ★ Interessantes Programmiermodell
  - ◆ jedoch noch einige Schwächen im Modell: Portabilitäts- und Semantikprobleme (z.B. Transaktionssemantik)