

G Enterprise Java Beans

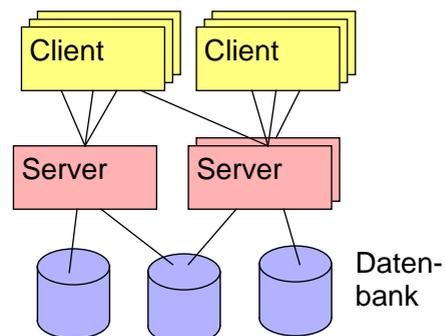
G.1 J2EE

- Java 2 Enterprise Edition
 - ◆ Erweiterungen zum Standard Java (J2SE, Java 2 Standard Edition)
 - ◆ EJB 2.0 (Enterprise Java Beans)
 - ◆ JDBC 2.0 (Java Database Connectivity)
 - ◆ Java Servlet Technology 2.3
 - ◆ JSP 1.2 (Java Server Pages)
 - ◆ JMS 1.0 (Java Message Service)
 - ◆ JNDI 1.2 (Java Naming and Directory Interface)
 - ◆ JTA 1.0 (Java Transaction API)
 - ◆ JavaMail API 1.2
 - ◆ JAXP 1.1 (Java API for XML Processing)
 - ◆ JCA (Java Connector Architecture)
 - ◆ JAAS 1.0 (Java Authentication and Authorization Service)

G.2 Motivation

G.2 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



G.2 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

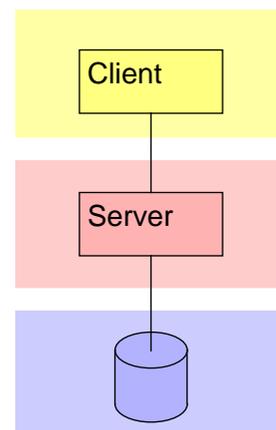
■ Application Server

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

G.3 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



G.3 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

G.3 Architektur (3)

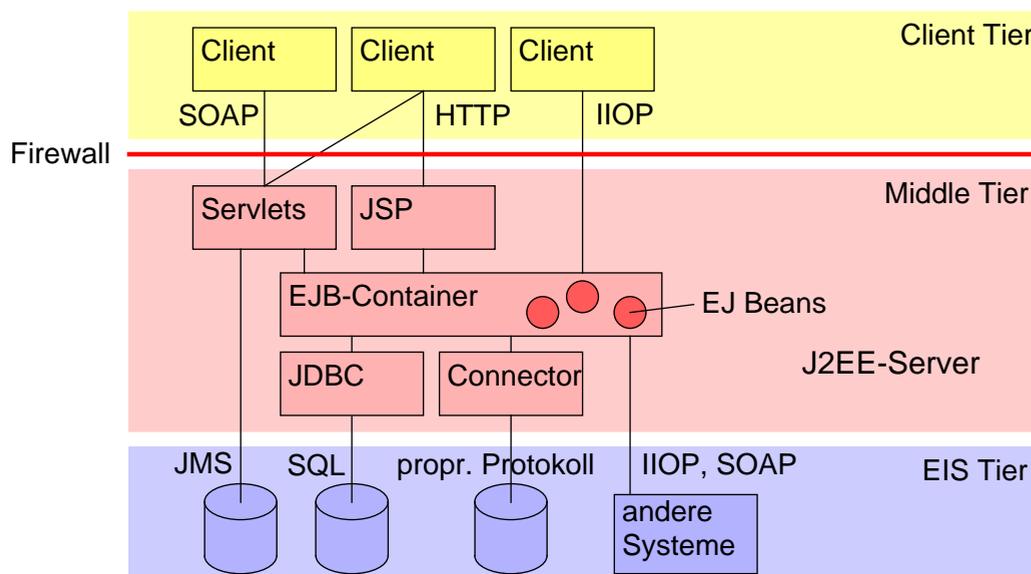
- Middle-Tier
 - ◆ Web-Tier
 - Web-Container für Java Server Pages oder Servlets
 - (CGI-Skript)
 - unnötig bei dedizierter Client-Anwendung
 - ◆ Business-Tier
 - enthält eigentliche Geschäftslogik
 - Einsatz von Geschäfts-Komponenten
 - Komponenten-Container
 - ◆ Verarbeitung von Geschäftsprozessen und Geschäftsdaten

G.3 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
- Abwicklung von Geschäftsprozessen
 - ◆ Zerlegung für mehrere hierarchisch aufgerufene Beans

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)	

2 Rollen von EJB (4)

- Werkzeuganbieter
 - ◆ Werkzeuge für die Code-Entwicklung
 - IDE, Integrated Development Environment
 - z.B. Visual Age (IBM)
 - ◆ Werkzeuge zur Modellierung und Code-Erzeugung
 - UML, Unified Modelling Language
 - z.B. Rational Rose, Together/J

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 Sicherheitservice, 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

3 Unterschied zu klassischer Middleware (6)

- Implizite EJB Unterstützung (fortges.)
 - ◆ entfernte Aufrufe
 - ◆ Ortstransparenz
 - wie klassische Middleware

G.4 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`

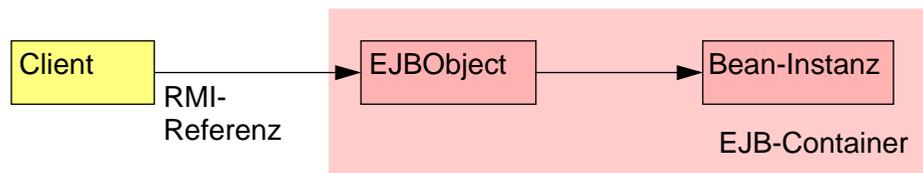
2 Interaktion mit Beans

- 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



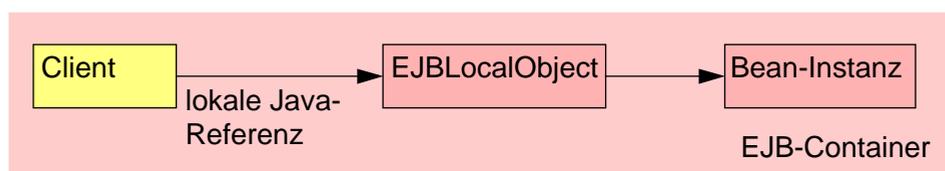
- ◆ führt implizite Middleware-Interaktionen durch
 - Sicherheitsüberprüfung, Transaktionsverwaltung, Datenbankabfragen ...
- ◆ Interaktion mit EJBObject über RMI bzw. RMI-IIOP
 - Interaktion im lokalen Fall teuer (Marshalling und Demarshalling, lokaler Nachrichtentransport etc.)

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
- ◆ Klienten rufen Bean über ein EJBLocalObject auf
 - kein entfernter Aufruf möglich

- EJBLocalObject fängt Aufrufe an der Bean ab



- ◆ auch hier: implizite Interaktion mit der Middleware

2 Interaktion mit Beans (4)

- Zusammenhänge
 - ◆ Methoden der Bean-Klasse
 - ◆ Methoden des EJBObject (entferntes Bean-Interface)
 - ◆ Methoden des EJBLocalObject (lokales Bean-Interface)

 - ◆ Methoden der Bean müssen mit gleicher Signatur im Bean-Interface des EJBObject auftreten (jedoch hier mit Exception `java.rmi.RemoteException`)
 - ◆ Methoden der Bean müssen mit gleicher Signatur im lokalen Bean-Interface des EJBLocalObject auftreten (jedoch hier u.U. ohne Exception `java.rmi.RemoteException`)
 - ◆ Methoden der Bean benötigen keine `java.rmi.RemoteException`
 - Methoden der Bean können EJB-System Exceptions werfen
 - diese werden nicht an den Client weitergegeben

2 Interaktion mit Beans (5)

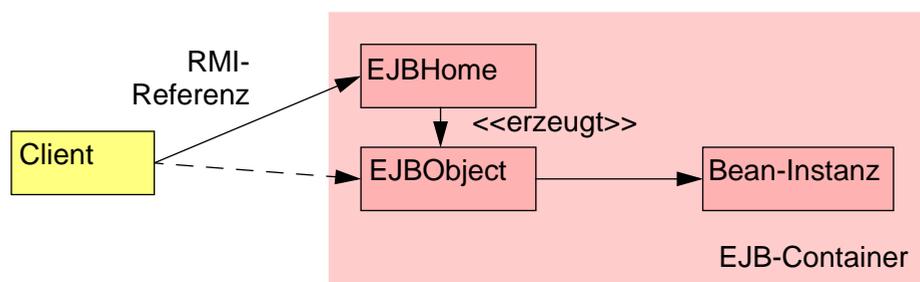
- Zusammenhänge (fortges.)
 - ◆ kein syntaktischer Zusammenhang zwischen den zwei Interfaces und der Bean-Klasse
 - ◆ Zusammenhang kann vom Bean-Entwickler durch eigenes Interface eingezogen werden
 - Bean-Klasse erbt von eigenem Business-Logic-Interface
 - EJBObject- und EJBLocalObject-Interface erbt von eigenem Business-Logic-Interface

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: entfernte 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, d. h. EJBObjects

5 Bean-Interaktion mit dem Container

- Interaktion mit Container bisher nur implizit
- Explizite Interaktion über Context-Objekt
 - ◆ Methode `setXYZContext` mit `XYZ` gleich `Session`, `Entity` oder `MessageDriven`
 - ◆ Container übergibt bei Bean-Instanzerzeugung Context-Objekt
 - ◆ indirekte und standardisierte Interaktion mit dem Container
 - 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
 - ◆ Spitzname (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`

G.5 Beispiel: Session-Beans

- Eigenschaften einer Session-Bean
 - ◆ Lebenszeit verknüpft mit einer Session
 - z.B. aktiv solange ein Einkaufsprozess läuft
 - z.B. aktiv solange Kunde auf seine Bankdaten zugreift
 - Session-Bean realisiert „Verben“
 - transiente Lebenszeit
(überlebt keinesfalls Rechnerausfall, Container-Ausfall)
 - ◆ zwei Varianten bzgl. Zustandsspeicherung während einer Session
 - zustandslos
 - zustandsbehaftet

1 Zustandslose Session-Bean

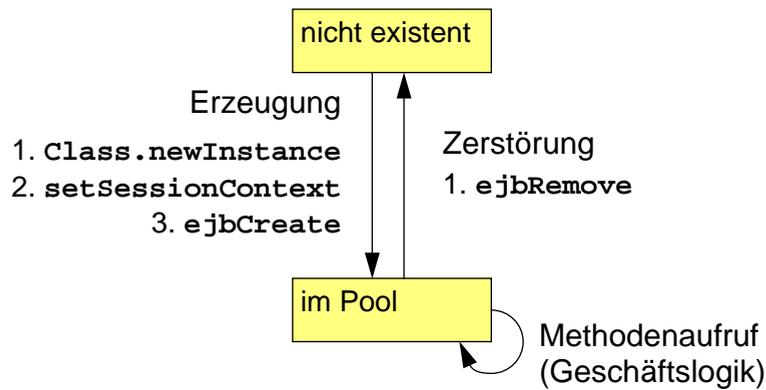
- Eigenschaften
 - ◆ Session-Bean speichert keinen Zustand
 - Aktion/Session besteht aus nur einem Methodenaufruf
 - z.B. Hello-Bean
 - z.B. Kreditkartenautorisierung
 - ◆ Zustand nur während der Methodenbehandlung
 - ◆ falls doch Zustand erforderlich:
 - Zustand per Parameterübergabe und -rückgabe oder
 - Zustand über Datenbank

1 Zustandslose Session-Bean (2)

- Behandlung im Container
 - ◆ beliebige Beaninstanz kann für alle EJBObjects des gleichen Bean-Typs benutzt werden
 - Wiederverwendung vorhandener Bean-Instanzen möglich (Vergangenheit der Instanz irrelevant)
 - ◆ Pooling von Bean-Instanzen
 - pro Bean nur ein aktiver Methodenaufruf (single-threaded)
 - Erzeugung einer Menge von Arbeitsinstanzen

1 Zustandslose Session-Bean (3)

■ Lebenszeit einer zustandslosen Session-Bean (typisch)



1 Zustandslose Session-Bean (4)

■ Interaktion des Containers mit der Bean

- ◆ Aufruf von `ejbCreate()` nach Erzeugung
 - zur Initialisierung (z.B. Reservierung von Ressourcen)
 - immer ohne Parameter
 - taucht im Home-Interface als `create()` auf
- ◆ Aufruf von `ejbRemove()` vor Zerstörung
 - zum Freigeben von Ressourcen

2 Zustandsbehaftete Session-Bean

- Eigenschaften
 - ◆ Session-Bean speichert Zustand
 - bis Aufrufer EJBObject freigibt (durch Aufruf von `remove()`)
 - ◆ Aktion/Session besteht aus mehreren Methodenaufrufen
 - z.B. Kaufvorgang in einem Webshop

- Behandlung im Container
 - ◆ Nutzung einer Bean-Instanz pro Client (pro erzeugtem EJBObject)
 - ◆ Problem: große Ressourcenverschwendung durch Vielzahl von Session-Bean-Instanzen, die selten genutzt werden
 - ◆ Lösung: Passivierung und Aktivierung
 - Passivierung sichert Zustand der Bean-Instanz
 - Aktivierung stellt Zustand der Bean-Instanz wieder her

2 Zustandsbehaftete Session-Bean (2)

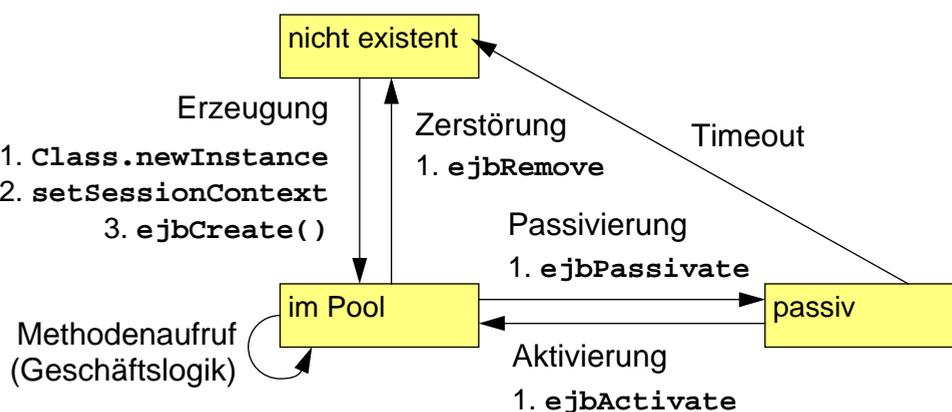
- Behandlung im Container (fortges.)
 - ◆ Passivierungs- und Aktivierungsmechanismen
 - Java-Serialisierung oder
 - proprietäres Verfahren (z.B. Serialisierung und Reflection-API)
 - alle nicht-transienten Variablen werden gesichert
 - besondere Sicherung für Referenzen auf EJBObjects, Home-Interfaces, Session-Kontextobjekte und JNDI-Namenskontexte

2 Zustandsbehaltete Session-Bean (3)

- Behandlung im Container (fortges.)
 - ◆ Pooling der Bean-Instanzen
 - Pool der Instanzen bilden aktive Bean-Instanzen
 - Zugriff auf passive Beans (*Activation on Demand*)
 - Passivierung einer aktiven Bean-Instanz (z.B. nach LRU-Algorithmus)
 - Aktivierung der gleichen/einer neuen Instanz mit neuem Bean-Zustand
 - Weiterleitung des Aufrufs vom EJBObjekt an dieser Instanz
 - vergleiche Virtueller Speicher!
 - ◆ keine Passivierung der Teilnehmer einer Transaktion

2 Zustandsbehaltete Session Bean (4)

- Lebenszeit einer zustandsbehalteten Session-Bean



- ◆ Zerstörung von Bean-Instanzen nach einstellbarem Timeout
 - Parameter des herstellerabhängigen Deployments

2 Zustandsbehaftete Session-Bean (5)

- Interaktion des Containers mit der Bean
 - ◆ Aufruf von `ejbCreate()` nach Erzeugung
 - zur Initialisierung (z.B. Reservierung von Ressourcen)
 - mehrere Varianten mit unterschiedlichen Parametern möglich
 - taucht im Home-Interface als Varianten von `create()` auf
 - ◆ Aufruf von `ejbRemove()` vor Zerstörung
 - zum Freigeben von Ressourcen
 - ◆ Aufruf von `ejbPassivate()` vor Passivierung
 - zur Freigabe von Ressourcen
 - zum Aufräumen des Zustands
 - ◆ Aufruf von `ejbActivate()` nach Aktivierung
 - zum Belegen von Ressourcen
 - zum Initialisieren des reaktivierten Zustands

G.6 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)