In diesem Kapitel:

Enterprise JavaBeans (EJB) - Grundlagen, Technologie und Anwendungen

1.1. Einleitung – Die Lernziele

1.1.1. Die Groblernziele

Nach dem Durcharbeiten dieser Unterlagen sollten Sie in der Lage sein :

- die grundlegende Architektur
- die Rolle der Clients und der Server
- den Lebenszyklus von Session- und Entity-Beans

zu verstehen.

1.1.2. Die Feinlernziele

Nach dem Durcharbeiten sollte Sie konkret folgendes können:

- entwickeln von EJB Technologie-basierte verteilte Systeme
- kreieren von Entity Beans
- kreieren von Session Beans
- Lösung "auslieferungsfertig" machen
- kreieren von selbständigen Enterprise Bean Clients
- Entity Beans aus Session Beans aufrufen

1.1.3. Voraussetzungen

Ausser den Java Kenntnissen sind keine weiteren Voraussetzungen nötig.

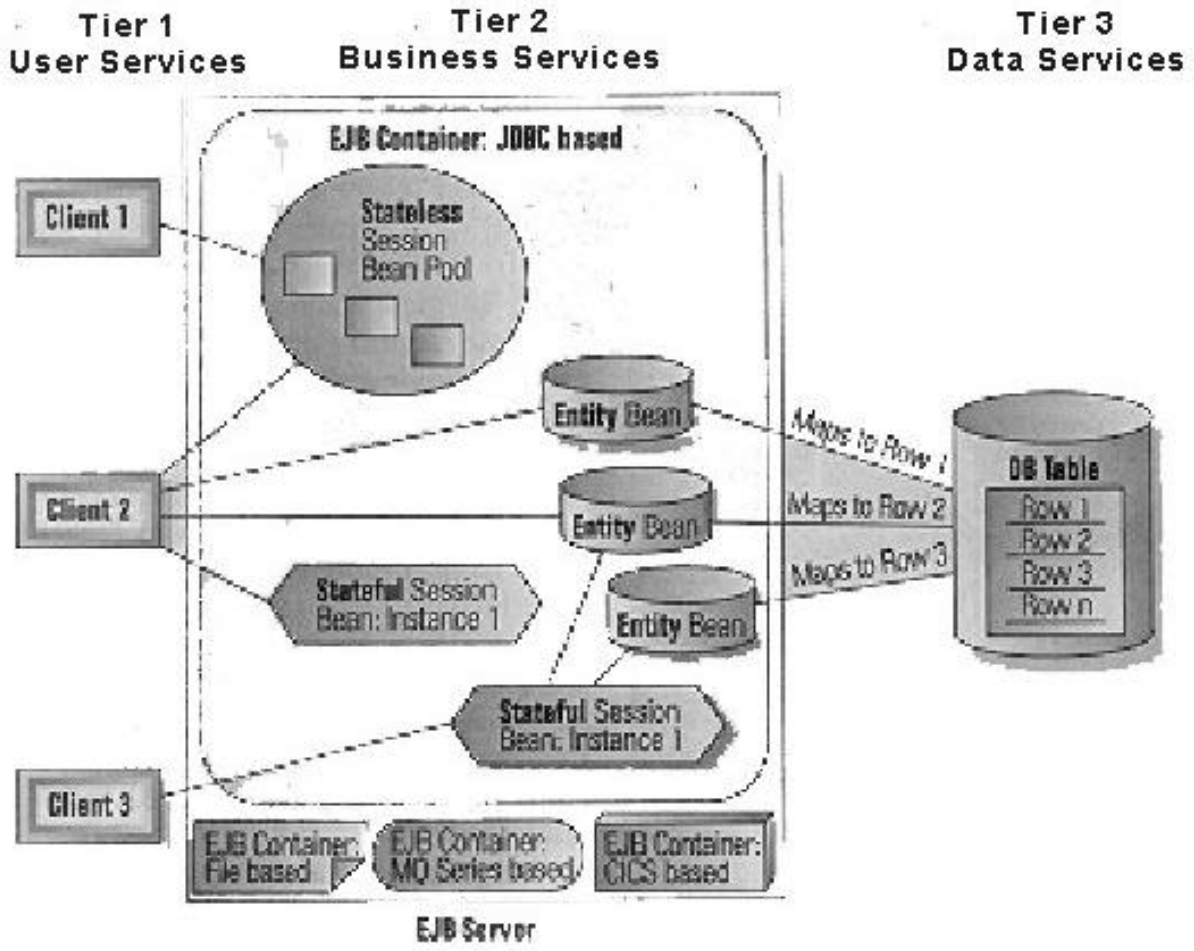
1.1.4. Übersicht

Die Enterprise JavaBeans™ (EJB) Spezifikation definiert eine Architektur für das Entwickeln und Verbreiten von transaktionsorientierten, objektbasierten, verteilten, Server-seitigen Softwarekomponenten.

Als Anwender kann ich Komponenten entwickeln oder von Anbietern kaufen. Diese serverseitigen Komponenten, Enterprise Java Beans (EJB) genannt, sind verteilte Objekte, welche in einem Enterprise JavaBeans Container existieren und Dienste für verteilte Clients über das Netzwerk anbieten.

1.1.5. Enterprise JavaBeans Technologie

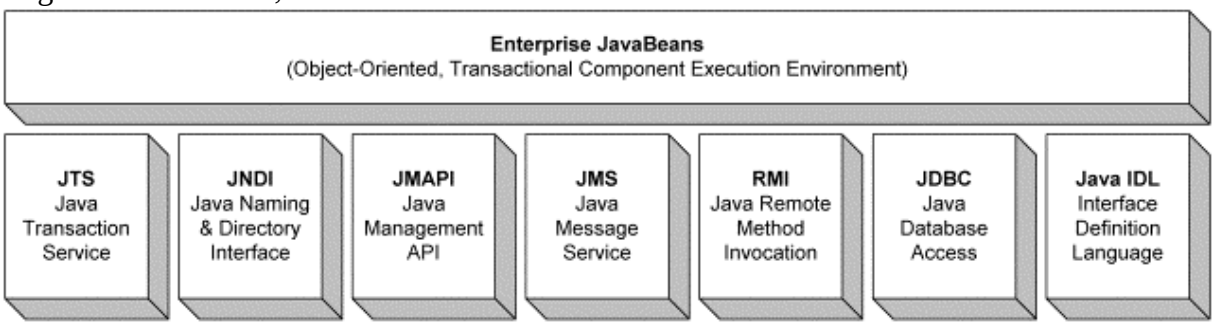
Die Enterprise JavaBeans Spezifikation definiert eine Architektur für ein transaktionsorientiertes, verteiltes Objektsystem, welches auf (Software) Komponenten basiert. Die Spezifikation schreibt ein Programmiermodell vor; dieses besteht aus Konventionen oder Protokollen und Klassen und Interfaces, die zusammen das EJB API ausmachen. Das EJB Programmiermodell stellt dem Bean Entwickler und dem EJB Server Anbieter Kontrakte zur Verfügung, welche als Ganzes eine gemeinsame Plattform für die Entwicklung zur Verfügung stellen. Ziel dieser Kontrakte ist es, die Portabilität einer Lösung herstellerunabhängig zu definieren ohne die Funktionalität einzuschränken.



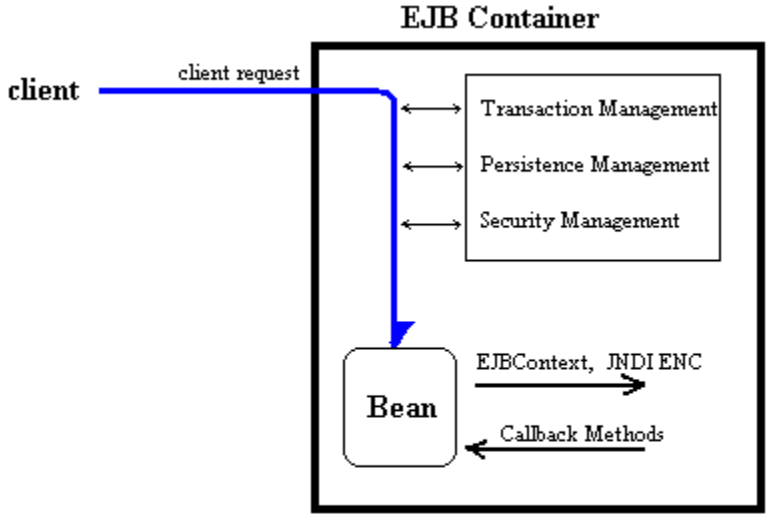
1.1.5.1. Der EJB Container

Enterprise Beans sind Software Komponenten, welche in einer speziellen Umgebung, dem EJB Container ablaufen. Der Container ist Basis für und managed eine Enterprise Bean auf die selbe Art und Weise wie der Java Web Server Servlet oder ein HTML Browser ein Java Applet. Ein Enterprise Bean kann ausserhalb eines EJB Container nicht existrieren. Der EJB Container managed alle Aspekte einer Enterprise Bean zur Laufzeit, inklusive remote Zugriff auf die Bean, Sicherheit, Persistenz, Transaktionen, Concurrency und Zugriff auf und Pooling von Ressourcen.
Der Container isoliert die Enterprise Bean vor direktem Zugriff durch Client Anwendungen. Wenn eine Client Applikation eine remote Methode einer Enterprise Bean aufruft, prüft der Container zuerst den Aufruf on Hinsicht darauf ob Persistenz, Transaktionen und Security korrekt angewandt werden, in jeder Operation von Clients in Bezug auf die Bean.

Der Container managed Security, Transaktionen und Persistenz automatisch für die Bohne, so dass der Bean Entwickler sich darum kümmern muss, und vorallem auch keinen Code dafür schreiben muss.Der Enterprise Bean Entwickler kann sich auf das Kapseln der Business Regeln konzentrieren, während der Container sich um den Rest kümmert.



EJBs bauen auf vielen Java Komponenten auf



EJB Container managen Enterprise Beans zur Laufzeit

Container managen viele Beans gleichzeitig auf die selbe Art und Weise wie der Java WebServer mehrere Servlets managed. Um den Speicherbedarf und die Verarbeitung zu optimieren fasst der Container wann immer möglich mehrere Beans zusammen und managen den Lebenslauf aller Beans sehr vorsichtig. Wenn ein Bean nicht benutzt wird legt der Container die Bohne in einen Pool von Bohnen, die von andern Clients verwendet werden können, oder möglicherweise aus dem Memory entfernt und nur zurückgebracht falls nötig.

Weil Client Applikationen keinen direkten Zugriff auf die Beans haben - der Container liegt zwischen dem Client und Bean - sehen die Client Applikationen die Ressource Management Aktivitiäten des Containers nicht. Ein Bean, welches nicht benutzt wird, kann aus dem Memory des Servers entfernt werden, wobei die remote Referenz auf der Clientseite intakt bleibt. Falls der Client eine Methode eine remote Referenz aufruft re-inkarniert der Container einfach das Bean, um den Request erfüllen zu können. Die Client Applikation ist sich der gesamten Komplexität des Prozesses ncht bewusst.

Ein Enterprise Bean hängt vom Container in jeder Beziehung ab. Falls eine Enterprise Bean eine JDBC Verbindung benötigt oder eine andere Enterprise Bean geschieht dies mit Hilfe des Containers; falls eine Enterprise Bean Informationen über den Aufrufer oder Zugriff auf Eigenschaften benötigt, setzt die Bean dafür den Container ein.

Die Enterprise Bean interagiert mit ihrem Container mit Hilfe von drei Mechanismen: Callback Methoden, das EJBContext Interface oder das Java™ Naming und Directory Interface (JNDI).

- **Callback Methoden**
  Jede Bean implementiert einen Subtyp vom EnterpriseBean Interface, welches mehrere Methoden definiert, Callback Methoden genannt. Jede Callback Methode nenachrichtigt die Bohne über ein anderes Event in ihrem Lebenszyklus und der Container benutzt diese Methoden, um die Bean zu aktivieren, die Daten persistent in einer Databank abzuspeichern, eine Transaktion abzuschliessen, ein Bean aus dem Memory zu entfernen, etc. Die Callback Methoden geben der Bean eine Chance Hausarbeiten direkt bevor oder nach einem Event zu erledigen. Callback Methoden diskutieren wir im Detail in weiter unten.
- **EJBContext**
  jede Bean erhält ein EJBContext Objekt, welches eine Referenz auf den Container ist. Das EJBContext Interface stellt Methoden für die Interaktion mit dem Container zur Verfügung, so dass diese Bean Information über ihr Environment wie die Identität seiner Clients, dem Status einer Transaktion oder remote Referenzen auf sich selbst erhält.
- **Java Naming and Directory Interface**
  Java Naming and Directory Interface (JNDI) ist eine Standard Erweiterung der Java Plattform für den Zugriff auf Namenssysteme wie LDAP, NetWare, Dateisysteme, etc. Jede Bean hat automatisch Zugriff auf ein specielles Naming System, den Environment Naming Context (ENC). Der ENC durch den Container gemanaged und durch Beans mittels JNDI genutzt. Der JNDI ENC erlaubt es einer Bean auf andere Resourcen, wie JDBC Connections, andern Enterprise Beans und spezifischen Eigenschaften dieser Bean zuzugreifen.

Die EJB Spezifikation definiert einen Bean-Container Kontrakt, welcher die obigen Mechanismen (callbacks, EJBContext, JNDI ENC) umfasst, sowie Regeln, welche beschreiben, wie sich Enterprise Beans und deren Containers zur Laufzeit verhalten, wie Security Zugriffe überprüft werden, wie Transaktionen gemanaged werden, wie Persistenz angwandt wird, etc.

Der Bean-Container Kontrakt ist so aufgebaut, dass Enterprise Beans zwischen EJB Container portablel sind und somit Enterprise Beans einmal entwickelt und in jedem EJB Container lauffähig sind. Hersteller wie BEA, IBM und GemStone verkaufen Applikationsserver, die EJB Container enthalten. Idealerweise sollte eine beliebige Enterprise Bean, die der Spezifikation entspricht, in jedem EJB Container lauffähig sein.

Portabilität ist zentral für EJBs. Portabilität garantiert, dass ein Bean, welche für einen Container entwickelt wurde, auf einen andern Container migriert werden kann, falls dieser andere zum Beispiel leistungsfähiger ist, zusätzliche Funktionalität anbietet oder kostengünstiger ist. Portabilität bedeutet auch, dass die Bean Entwickler Skills für mehrere Platformen und EJB Container ausgenutzt werden kann, und damit dem Entwickler mehr Potential anbietet.

Zusätzlich zur Portabilität kommt die Einfachheit des EJB Programmier Modells, welche das EJB wertvoll machen. Weil der Container komplexe Aufgaben wie Security, Transaktionen, Persistenz, Concurrency und Resource Management übernimmt, ist es dem Bean Entwickler überlassen sich auf die Business Regeln zu konzentrieren, neben dem einfachen Programmiermodell. Einfaches Programmiermodell heisst, dass Beans schneller entwickelt werden können, ohne dass man einen Doktertitel in verteilten Objektsystemen, Transaktionen und andern Enterprise Systemen dazu benötigt.

EJB bringen Transaktionsverarbeitung und verteilte Objektentwicklung in den Mainstream.

1.1.6. Übung - Installation und Konfiguration der J2EE Referenz Implementation

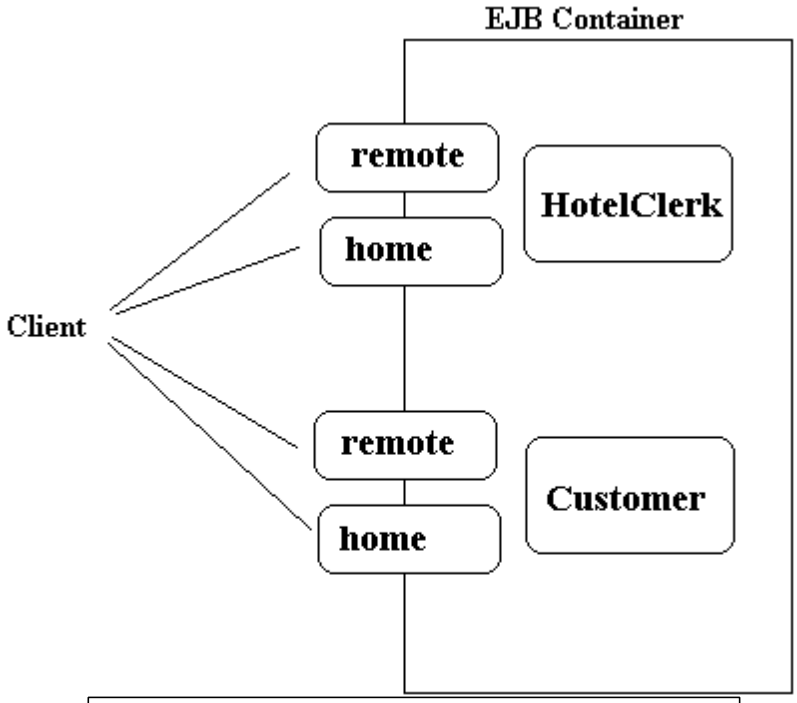
Lösen Sie jetzt die Aufgabe 1 aus dem Übungsteil.

1.1.6.1. Enterprise Beans

Um eine EJB Server-seitige Komponente zu entwickeln muss der Entwickler zwei Interfaces zur Verfügung stellen, welche die Business Methoden der Bean zur Verfügung stellen, plus die aktuelle Bean Implementation Klasse. Der Client verwendet die public Bean Interfaces um Beans zu kreieren, manipulieren und vom EJB Server zu entfernen. Die Implementationsklasse (die Bean Klasse) wird zur Laufzeit instanziert und wird ein verteiltes Objekt

.

Enterprise Beans leben in einem EJB Container und werden von Client Applikationen über das Netzwerk mit Hilfe der remote und normalen Interfaces eingesetzt. Die remote und normalen Interfaces repräsentieren die Fähigkeiten der Bean und stellen alle Methoden zur Verfügung, die benötigt werden, um die Bean zu kreieren, updaten, mit ihnen zu interagieren und um die Bean zu vernichten. Ein Bean ist eine Server-seitige Komponente, die ein Business Konzept wie etwa einen Kunden oder einen Hotelangestellten oder ein Pop3 oder ein FTP oder ... darstellt.

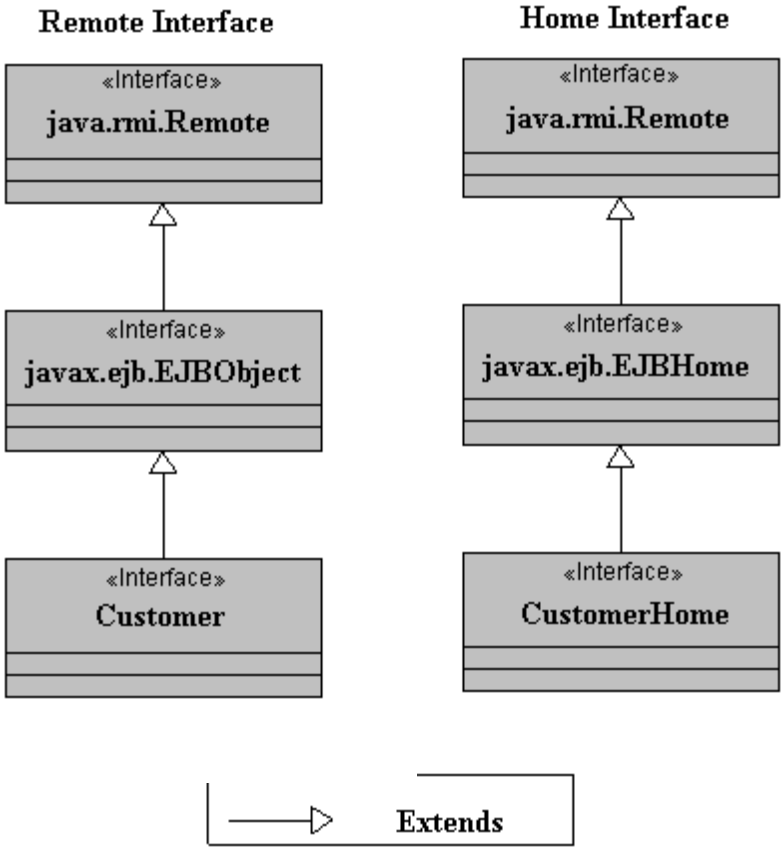


konzeptionelle Sicht der EJB Architektur

1.1.6.1.1. Remote und Home Interfaces

Die remote und lokalen (home) Interfaces repräsentieren die Bean, aber der Container isoloert die Beans vor direktem Zugriff von Client Applikationen. Jedes Mal wenn ein Bean aufgerufen, kreiert oder gelöscht wird, managed der Container den gesamten Prozess.

Das Home Interface repräsentiert die Lebenszyklus Methoden der Komponente (create, destroy, find) während die remote Interfaces die Business Methoden der Bean repräsentieren. Die remote und home Interfaces erweitern das javax.ejb.EJBObject respektive javax.ejb.EJBHome Interfaces. Diese EJB Interface Typen definieren ein Standard Set von Hilfsmethoden und stellen gemeinsame Basistypeen für alle remote und home Interfaces zur Verfügung.



Klassendiagramm: remote & home Interfaces

Clients verwenden das Bean home Interface um Referenzen auf die remote Interfaces der Bean zu erhalten. Das remote Interface definiert die Business Methoden wie beispielsweise Mutationsmethoden zum ändern eines Kundenname, oder Businessmethoden, die Aufgaben wie beispielsweise den Einsatz des Hotelangestellten zum Reservieren eines Zimmers im Hotel.

Das folgende Beispiel zeigt, wie eine Customer Bean aus einer Client Applikation heraus eingesetzt werden kann. In diesem Fall ist das home Interface `CustomerHome` und das remote Interface ist die `Customer` Klasse.

```
CustomerHome home = // ... Referenz auf ein Objekt, welches
                    // das home Interface implementiert

// kreieren einer neuen Instanz der Customer Bean
//
Customer customer = home.create(customerID);

// Einsatz einer Business Method
customer.setName(someName);
```

Das remote Interface definiert die Businessmethoden eine Bean also die Methoden, welche spezifisch sind für das Businesskonzept, die sie darstellen. Remote Interfaces sind Unterklassen vom javax.ejb.EJBObject Interface, welches eine Unterklasse vom java.rmi.Remote Interface ist.

Die Wichtigkeit der remote Interfaces Vererbungshierarchie diskutieren wir später. Fokusieren wir uns zuerst auf die Businessmethoden und deren Bedeutung. Unten sehen Sie die Definition eines remote Interfaces für eine Customer Bean.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Customer extends EJBObject {

    public Name getName()
                throws RemoteException;
    public void setName(Name name)
           throws RemoteException;
    public Address getAddress()
             throws RemoteException;
    public void setAddress(Address address)
                     throws RemoteException;

}
```

Das remote Interface definiert Zugriffs- und Mutationsmethoden zum lesen und mutieren von Informationen über ein Businesskonzept. Dies ist ein typischer Repräsentant einer sogenannten Entity Bean, welche persistente Businessobjekte darstellt also Businessobjekte, deren Werte (Zustand) in einer Datenbank abgespeichert sind. Entity Beans repräsentieren *Business Daten* in der Datenbank und definieren spezifisches Verhalten zu den Daten.

1.1.6.1.2. Business Methoden

Business Methoden können auch *Tasks* darstellen, die von einer Bohne ausgeführt werden. Obschon Entity Beans oft Task-orientierte Methoden haben werden Tasks typischerweise durch sogenannte Session Bean realisiert. Session Beans repräsentieren keine Daten (wie Entity Beans). Sie repräsentieren Business Processe oder Agenten, welche einen Dienst ausführen, wie zum Beispiel eine Reservation tätigen.

Unten finden Sie die Definition des remote Interface für einen Hotelangestellten (HotelClerk Bean), welche eine Art Session Bean ist.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HotelClerk
               extends EJBObject {

    public void reserveRoom(Customer cust,
                 RoomInfo ri,Date from, Date to)
                          throws RemoteException;

    public RoomInfo availableRooms(
         Location loc, Date from, Date to)
                    throws RemoteException;
}
```

Die Business Methoden, die im `HotelClerk` remote Interface definiert sind, repräsentieren Prozesse. Die HotelClerk Bean agiert als ein Agent im Sinne, dass sie Tasks für den Benutzer ausführt, aber selbst nicht persistent in der Datenbank ist. Sie brauchen über keine Informationen über den `HotelClerk` zu verfügen, Sie wollen, dass der Hotelangestellte etwas für Sie erledigt. Dies ist das typische Verhalten einer Session Bean.

Es gibt grundsätzlich zwei Typen von Enterprise Beans:

- Entity Beans
  welche Daten in einer Datenbank repräsentieren und

- Session Beans
  welche Prozesse darstellen oder als Agenten agieren, die bestimmte Aufgaben erfüllen.

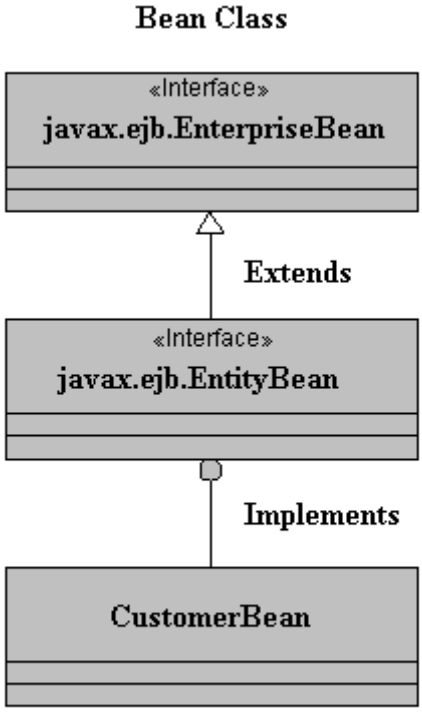
Eine EJB Applikation umfasst in der Regel viele Enterprise Beans, jede stellt ein anderes Business Konzept dar. Jedes Business Konzept wird entweder als eine Entity Bean oder eine Session Bean dargestellt. Sie wählen den passenden Typus aufgrund der Aufgaben der Bohne, wie oben skizziert.

1.1.6.1.3. Entity Beans

Für jedes remote Interface gibt es eine Implementationsklasse, ein Business Objekt, welches die Business Methoden, die im remote Interface definiert sind, auch tatsächlich implementiert. Das ist die *Bean Klasse*, das Schlüsselelement der Bohne. Unten sehen Sie eine partielle Definition der Customer Bean Klasse.

```
import javax.ejb.EntityBean;

public class CustomerBean
               implements EntityBean {

    Address    myAddress;
    Name       myName;
    CreditCard myCreditCard;

    public Name getName() {
         return myName;
    }

    public void setName(Name name) {
         myName = name;
    }

    public Address getAddress() {
         return myAddress;
    }
    public void setAddress(Address address) {
         myAddress = address;
    }
    ...
}
```

`CustomerBean` ist die Implementationsklasse. Sie enthalten die Daten und stellen Zugriffsmethoden und andere Business Methoden zur Verfügung. Als Entity Bean lieferen `CustomerBean` eine Objektsicht der Customer Daten. Statt Datenbank Zugriffslogik in eine Applikation zu fassen, kann die Applikation einfach das remote Interface einsetzen, um auf die Customer Bean zuzugreifen und damit auf Kundendaten. Entity Beans implementieren die javax.ejb.EntityBean Klasse, die ein Set von Notifikationsmethoden definiert, mit deren Hilfe die Bean benutzt, um mit seinem Container zu kommunizieren. Diese Notifikationsmethoden werden wir nach genauer untersuchen.



Class Diagram for the Bean Class

1.1.6.1.4. Session Beans

Die `HotelClerk` Bean ist eine Session Bean, welche in mancher Hinsicht einer Entity Bean sehr ähnlich ist. Session Beans repräsentieren ein Set von Prozessen oder Tasks, welche für eine Client Applikation ausgeführt werden. Session Beans können auch andere Beans einsetzen, um bestimmte Tätigkeiten ausführen zu lassen oder auf eine Datenbank zuzugreifen.

Die folgende Programmskizze zeigt ein Session Bean, welche beides tut. Die `reserveRoom()` Methode, die Sie unten sehen, verwendet mehrere andere Beans, um eine Aufgabe, eine Task zu erledigen, während die Methode `availableRooms()`JDBC verwendet, um direkt auf eine Datenbank zuzugreifen.

```
import javax.ejb.SessionBean;

public class HotelClerkBean
            implements SessionBean {

    public void reserveRoom(
          Customer cust, RoomInfo ri,
                    Date from, Date to) {
        CreditCard card = cust.getCreditCard();
        RoomHome roomHome =
        // ... get home reference
        Room room =
         roomHome.findByPrimaryKey(ri.getID());
        double amount = room.getPrice(from,to);
        CreditServiceHome creditHome =
            // ... get home reference
        CreditService creditAgent =
                     creditHome.create();
        creditAgent.verify(card, amount);
        ReservationHome resHome =
```

```
            // ... get home reference
        Reservation reservation =
           resHome.create(cust,room,from,to);
    }

    public RoomInfo[] availableRooms(
                  Location loc,Date from,
                                  Date to) {
        // Make an SQL call to find
        //available rooms
        Connection con = // ... get database
        //connection
        Statement stmt = con.createStatement();
        ResultSet results =
             stmt.executeQuery("SELECT ...");
        ...
        return roomInfoArray;
    }
}
```

Sie haben vielleicht bemerkt, dass die Bean Klassen, die oben definiert wurden, das remote oder home Interface nicht implementieren. EJB verlangt nicht, dass die Bean Klasse diese Interfaces implementiert; in realen Projekten wird man dies sogar zu vermeiden suchen, weil die Basisklassen, remote und home Interfaces (`EJBObject` und `EJBHome`), viele Methoden definieren, die bereits vom Container automatisch implementiert werden. Die Bean Klasse liefert allerdings Implementationen für alle Business Methods, die im remote Interface definiert wurden.

Callback Methods besprechen wir im Detail weiter unten

1.1.6.1.5. Life Cycle Methods

Zusätzlich zum remote Interface besitzen alle Beans ein home Interface. Das home Interface stellt life cycle Methoden für das Kreieren, Zerstören und Lokalisieren von Beans zur Verfügung. Dieses life cycle Verhalten wird losgelöst vom remote Interface behandelt, weil es Verhaltensmuster repräsentieren welche nicht typisch für nur eine Bean Instanz sind.

Die folgende Programmskizze definiert das home Interface für die Customer Bean. Beachten Sie, dass sie das javax.ejb.EJBHome Interface erweitert, welches seinerseits das java.rmi.Remote Interface erweitert.

```
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.rmi.RemoteException;

public interface CustomerHome
                  extends EJBHome {

    public Customer create(Integer
                    customerNumber)
                  throws RemoteException,
                          CreateException;

    public Customer findByPrimaryKey(Integer
                             customerNumber)
                  throws RemoteException,
                           FinderException;
```

```
    public Enumeration findByZipCode(int zipCode)
                        throws RemoteException,
                                FinderException;
}
```

Die `create`() Methode wird eingesetzt, um neue Entitäten zu kreieren. Dies führt zu einem neuen Datensatz in der Datenbank. Ein home kann mehrere `create`() Methoden besitzen. Die Anzahl und die Klasse der Argumente jeder create() Methode, sind dem Bean Entwickler überlassen, aber der return Type muss vom remote Interface Datentyp (Klasse, Interface) sein.

Im obigen Fall liefert der Aufruf von `create`() des `CustomerHome` Interface eine Instanz der Klasse `Customer`. Die `findByPrimaryKey`() und `findByZipCode`() Methoden werden eingesetzt, um spezifische Instanzen der Customer Bean zu finden. Sie können soviele `find` Methoden wie Sie benötigen.

1.1.6.1.6. Zurück zum Remote und Home Interface

Das remote und home Interface wird von Applikationen benutzt, um auf Enterprise Beans zur Laufzeit zuzugreifen.

- Das home Interface erlaubt der Applikation die Bohne zu kreieren oder zu lokalisieren,
- während das remote Interface der Applikation erlaubt die Business Methoden einer Bohne aufzurufen.

Das folgende Programmfragment illustriert dies:

```
CustomerHome home =
// Get a reference to the
//CustomerHome object

Customer customer =
  home.create(new Integer(33));

Name name = new Name("Richard",
       "Wayne", "Monson-Haefel");
customer.setName(name);

Enumeration enumOfCustomers =
          home.findByZip(55410);

Customer customer2 =
     home.findByPrimaryKey(
               new Integer(33));

Name name2 = customer2.getName();

// output is "Richard Wayne
//Monson-Haefel"
System.out.println(name);
```

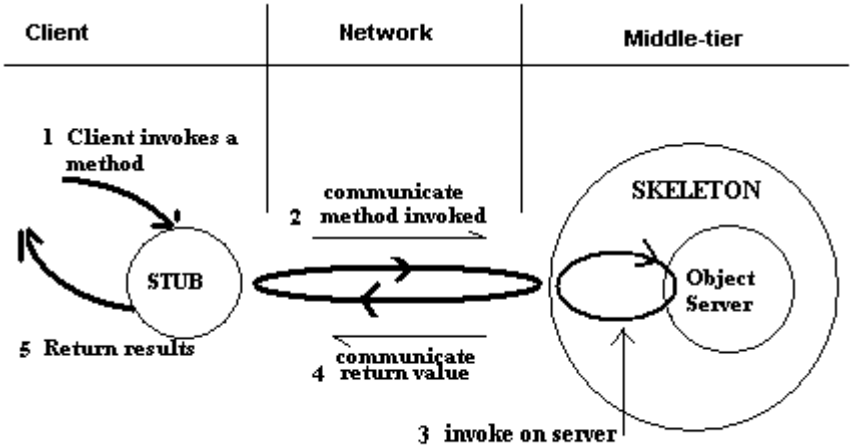
Das javax.ejb.EJBHome Interface definiert auch noch andere Methoden, die die `CustomerBean` automatisch erbt, insbesondere ein Set von remove() Methoden, welche der Applikation erlauben Bean Instanzen zu zerstören.

1.1.6.2. Enterprise Beans as Distributed Objects

Die remote und home Interfaces gehören zu den Java RMI Remote Interfaces. Das java.rmi.Remote Interface wird von verteilten Objekten eingesetzt, um Beans in unterschiedlichen Adressräumen (Prozess oder Machine) darzustellen. Ein Enterprise Bean ist ein verteiltes Objekt (in obigem Sinne : es implementiert Remote).

Das bedeutet, dass die Bean Klasse im Container instanziert wird und lebt, aber dass sie von einer Applikation aus einem andern Adressraum benutzt werden kann. Um eine Objektinstanz in einem Adressraum in einem andern Adressraum zur Verfügung stellen zu können, benötigt man bestimmte Tricks, unter anderem Netzwerk-Sockets.

Damit der Trick funktioniert, muss die Instanz in ein spezielles Objekt, ein Skeleton gewickelt werden (ummantelt, gewrapped). Dieses Objekt steht in Netzwerkverbindung mit einem weiteren Objekt, dem Stub. Stubs implementieren das remote Interface und sehen aus wie normale Business Objekte. Aber der Stub enthält keine Business Logik; es ist für die Neztwerk Socket Verbindung zum Skeleton zuständig. Jedes Mal wenn eine Business Methode des Stubs remote Interface aufgerufen wird, sendet der Stub eine Netzwerk Message zum Skeleton und teilt ihm mit, welche Methode aufgerufen wurde. Wenn das Skeleton eine Netzwerk Message vom Stub empfängt identifiziert es die aufgerufene Methode und die Argumente und ruft dann die entsprechende Methode des aktuellen Instanz auf. Die Instanz führt die Business Methode aus und liefert das Ergebnis dem Skeleton Objekt, welches das Ergebnis an den Stub zurück sendet. Das folgende Diagramm illustriert dies:



Der Stub liefert das Resultat an die Applikation, welche die remote Interface Method aufrief. Aus Sicht der Applikation, welche den Stub verwendet, sieht es so aus, als ob der Stub die Arbeit lokal erledigt hätte. In Wahrheit ist der Stub ein dummes Netzwerk Objekt, welches die Anfrage über das Netzwerk an das Skeleton Objekt, welches die Methode der aktuellen Instanz aufruft. Die Instanz erledigt die ganze Arbeit, Stub und Skeleton leiten die Methode (Identität und Argumente) über das Netzwerk hin und zurück.

In EJB werden Skeleton für das remote und home Interfaces durch den Container implementiert, nicht die Bean Klasse. Damit wird sichergestellt, dass alle Methodenaufrufe dieser Referenztypen zuerst durch den Container behandelt werden und erst dann an Bean Instanzen delegiert werden. Der Container muss Zugriffe auf Beans überprüfen und dafür sorgen, dass die Pesistenz (entity beans), Transaktionen und Zugriffe automatisch kontrolliert werden.

Distributed Objekt Protokolle definieren das Format der Netzwerk Messages, die zwischen den Adressräumen hin und her gesandt werden. Distributed Objekt Protokolle können recht komplex werden (wie Sie eventuell aus RMI wissen); aber zum Glück sehen Sie nichts davon, da alles automatisch gehandhabt wird.

Die meisten EJB Server unterstützen entweder das Java Remote Method Protocol (JRMP) aus RMI oder CORBAs Internet Inter-ORB Protocol (IIOP). Bean und Applikations-Programmierer sehen die Bean Klasse und sein remote Interface, die Details der Netzwerk Kommunikation bleiben verborgen. In Bezug auf das EJB API kümmert sich der Programmierer nicht darum, ob der EJB Server JRMP oder IIOP verwendet - das API ist das selbe. Die EJB Spezifikation verlangt, dass man eine spezielle Version des Java RMI API einsetzt, falls man mit remote Beans arbeiten möchte. Java RMI ist ein API für den Zugriff auf verteilte Objekte und ist in gewissem Sinn flexibel einsetzbar - genau wie JDBC im Datenbankbereich. Ein EJB Server kann JRMP oder IIOP unterstützen; aber Bean und Applikationsentwickler verwenden immer das selbe Java RMI API. Damit EJB Server optional IIOP einsetzen können, wurde eine spezielle Java RMI Version entwickelt, Java RMI-IIOP. Java RMI-IIOP verwendet IIOP als das Protokoll und das Java RMI API. EJB Server brauchen IIOP nicht einzusetzen; aber die EJB Server müssen Java RMI-IIOP Beschränkungen beachten: EJB 1.1 verwendet die Java RMI-IIOP Konventionen und Typen; aber das darunterliegende Protokoll spielt keine Rolle.

1.1.7. Entity Type Enterprise Beans

Die Entity Bean ist eine von zwei primären Bean Typen: Entity und Session. Die Entity Bean wird benutzt, um Daten in der Datenbank darzustellen. Sie stellen ein Object-orientiertes Interface für Daten zur Verfügung, auf die normalerweise mittels JDBC oder irgend ein anderes back-end API zugegriffen würde. Zudem stellen Entity Beans ein Komponenten Modell zur Verfügung, welches es dem Bean Entwickler erlaubt sich auf die Business Logik der Bean zu konzentrieren, während der Container sich um die Persistenz, Transaktionen und Zugriffskontrolle kümmern muss.

Es gibt zwei Typen von Entity Bean:
- Container-verwaltete Persistenz (CMP: Container Managed Persistence)
- und Bean-Managed Persistence (BMP).

Mit CMP managed der Container die Persistenz der Entity Bean. Mit Hilfe vorgefertigter Werkzeuge werden die Entity Felder auf die Datenbank abgebildet und absolutel keine Datenbank Zugriffsmethoden werden in der Bean Klasse geschrieben.

Mit BMP enthält die Entity Bean den Datenbankzugriffscode (in der Regel JDBC) und ist für das Lesen und Schreiben des eigenen Zustandes in die Datenbank verantwortlich. BMP Entitiäten werden bei diesen Aktivitäten vom Container unterstützt. Er informiert die Bean darüber wann es nötig ist den Zustand zu aus der Datenbank zu lesen, oder in der Datenbank auf den neusten Stand zu bringen. Der Container besorgt auch das Locking oder Transaktionen, so dass die Datenbank die Integrität behält.

1.1.7.1. Container-Managed Persistence

Container-managed Persistence Beans sind für den Bean Entwickler am leichtesten zu entwickeln, für den EJB Server allerdings am schwierigsten zu unterstützen. Das liegt daran, dass die gesamte Logik für die Synchronisation der Bean Zustände mit der Datenbank automatisch durch den Container behandelt wird. Das heisst, dass der Bean Entwickler keine

Programme für die Zugriffslogik auf Daten schreiben muss, da der EJB Server dies tun muss. Diese Aufgabe wird an den EJB Server Anbieter delegiert. Die meisten EJB Anbieter unterstützen automatische Persistenz zu einer relationalen Datenbank, aber der Supportlevel ist unterschiedlich. Einige Anbieter bieten komplexe Object-zu-Relational Abbildungen an; andere bieten sehr limitierte Funktionalitäten an.

Wir wollen nun unsere `CustomerBean`, die wir bereits besprochen haben, zu einer (Container-managed Persistence) CMP Bean erweitern. Im nächsten Abschnitt über Bean-gemanagte Persistenz werden Sie die `CustomerBean` so erweitern, dass sie ihre Persistenz selber managed.

1.1.7.1.1. Bean Class

Eine Enterprise Bean ist eine vollständige Komponente, welche aus mindestens zwei Interfaces und einer Bean Implementationsklasse besteht. All diese Typen werden wir und anschauen und deren Bedeutung und Anwendung erläutern, beginnend mit der Bean Klasse, die im folgenden Programmcode definiert wird:

```
import javax.ejb.EntityBean;

public class CustomerBean
             implements EntityBean {

    int        customerID;
    Address    myAddress;
    Name       myName;
    CreditCard myCreditCard;

    // CREATION METHODS
    public Integer ejbCreate(Integer id) {
        customerID = id.intValue();
        return null;
    }

    public void ejbPostCreate(Integer id) {
    }

    public Customer ejbCreate(Integer id, Name name) {
        myName = name;
        return ejbCreate(id);
    }

    public void ejbPostCreate(Integer id, Name name) {
    }

    // BUSINESS METHODS
    public Name getName() {
        return myName;
    }

    public void setName(Name name) {
        myName = name;
    }

    public Address getAddress() {
        return myAddress;
    }

    public void setAddress(Address address) {
```

```
        myAddress = address;
    }

    public CreditCard getCreditCard() {
        return myCreditCard;
    }

    public void setCreditCard(CreditCard card) {
        myCreditCard = card;
    }

    // CALLBACK METHODS
    public void setEntityContext(EntityContext cntx) {
    }

    public void unsetEntityContext() {
    }

    public void ejbLoad() {
    }

    public void ejbStore() {
    }

    public void ejbActivate() {
    }

    public void ejbPassivate() {
    }

    public void ejbRemove() {
    }
}
```

Dies ist ein gutes Beispiel für eine einfache CMP Entity Bean. Beachten Sie, dass keine Datenbankzugrifflogik in der Bohne definiert wird. Dies liegt daran, dass die Datenfelder aus der `CustomerBean` auf die Datenfelder einer Datenbank abgebildet werden. Die `CustomerBean` Klasse, zum Beispiel, könnte auf eine beliebige Datenbank abgebildet werden, sofern sie die Datenfelder enthält, die in den Datenfeldern der Bohne definiert wurden. Im Fall der obigen Bean Definition ergeben sich Datenfelder von Datentyp int und einfachen abhängigen Objekten (`Name`, `Address` und `CreditCard`) mit ihren eigenen Attributen. Unten sehen Sie die Definition des abhängigen Objekts:

```
// Die Name class
public class Name
        implements Serializable {

    public String lastName,
            firstName, middleName;

    public Name(String lastName, String firstName,
                String middleName) {
        this.lastName   = lastName;
        this.firstName  = firstName;
        this.middleName = middleName;
    }

    public Name() {}
}
```

```
// Die Address class
public class Address
      implements Serializable {

    public String street,
                city, state, zip;

    public Address(String street,
         String city,String state,
                       String zip) {
        this.street = street;
        this.city   = city;
        this.state  = state;
        this.zip    = zip;
    }

    public Address() {}
}

// Die CreditCard class
public class CreditCard
     implements Serializable {

    public String number,
            type, name;
    public Date expDate;

    public CreditCard(String
         number, String type,String name,
                          Date expDate) {
        this.number  = number;
        this.type    = type;
        this.name    = name;
        this.expDate = expDate;
    }

    public CreditCard() {}
}
```

Diese Felder bezeichnet man als *container-managed fields*, weil der Container für die Synchronisation dieser Datenfelder mit der Datenbank verantwortlich ist; der Container managed die Felder. Container-managed Felder können aus Basisdatentypen oder serialisierbaren Klassen bestehen. Im obigen Beispiel verwenden wir einen Basisdatentyp int (`customerID`) und serialisierbare Objekte (Address, Name, CreditCard).

Diese Abbildung der abhängigen Objekte von einer Datenbank bedingt ein recht komplexes Abbildungswerkzeug. Nicht alle Datenfelder einer Bean sind automatisch container-managed Datenfelder; einige sind vielleicht schlicht Instanzen, welche von der Bohne zum Arbeiten benötigt werden (also transient oder temporär). Ein Bean Entwickler unterscheidet container-managed Datenfelder von normalen Instanzen durch Angabe, welche der Datenfelder container-managed sind.

Die container-managed Datenfelder müssen entsprechende Datentypen (Spalten in der RDBMS) in der Datenbank besitzen, entweder direkt oder aufgrund eines Objekt-Relational Mappings.

Die CustomerBean könnte man beispielsweise auf eine CUSTOMER Tabelle in der Datenbank abbilden, die wiefolgt definiert ist:

```
CREATE TABLE CUSTOMER
{
    id            INTEGER PRIMARY KEY,
    last_name     CHAR(30),
    first_name    CHAR(20),
    middle_name   CHAR(20),
    street        CHAR(50),
    city          CHAR(20),
    state         CHAR(2),
    zip           CHAR(9),
    credit_number CHAR(20),
    credit_date   DATE,
    credit_name   CHAR(20),
    credit_type   CHAR(10)
}
```

Damit die EJBs die Persistenz Container-gemanaged behandeln können muss der Anbieter Werkzeuge zur Verfügung stellen, um die container-managed Datenfelder der Beans auf Spalten einer spezifischen Tabelle, CUSTOMER in unsrem Fall, abbilden zu können.

Falls die Beans Felder auf die Datenbank abgebildet sind und die Customer bean produktiv ist dann liegt es am Container die üblichen Operationen wie Kreieren neuer Zeilen, Laden von Zeilen, Mutieren von Zeilen und Löschen von Zeilen in der CUSTOMER Tabelle auf Anfrage der Methoden der Customer Bohne (remote und home Interfaces).

Ein Subset (ein oder mehrere) der Container-gemanagten Datenfelder werden auch als Primärschlüssel der Bean definiert. Der Primärschlüssel ist der Index oder Pointer auf einen eindeutigen Datensatz in der Datenbank, in welcher der Zustand der Bohne gespeichert wird. Im Falle der CustomerBean ist das `id` Datenfeld der Primärschlüssel und wird benutzt, um die Bohne in der Datenbank zu lokalisieren. Primärschlüssel, die aus primitiven (Basis-) Datentypen bestehen, werden als Objekte mit Hilfe ihrer Wrapperklassen dargestellt.

Beispiel :
der Primärschlüssel der Customer Bean ist vom Typ int, also vom Typ einer Basisklasse oder ein primitiver Datentyp.
Den Beans Clients zeigt sich dieser Primärschlüssel als Objekt vom java.lang.Integer Typus.

Falls der Primärschlüssel aus meheren Datenfeldern besteht, also ein zusammengesetzter (*compound*) Schlüssel ist, dann wird er durch eine spezielle, vom Entwickler zu definierende Klasse repräsentiert. Primärschlüssel von Beans haben also vieles gemeinsam mit den Primärschlüsseln in einer relationalen Datenbank - in der Realität sind die beiden dann oft identisch, wenn die Persistenz der Beans mit einer relationalen Datenbank realisiert wird..

1.1.7.1.2. Home Interface

Um eine neue Instanz einer CMP Entity Bean zu kreieren und damit Daten in die Datenbank einzufügen muss die Methode create() aus dem home Interface der Bohne aufgerufen werden. Dieses wird in unserem Beispiel durch das Interface CustomerHome definiert. Die Definition dieses Interfaces sieht wiefolgt aus:

```
public interface CustomerHome
               extends javax.ejb.EJBHome {
```

```
    public Customer create(
           Integer customerNumber)
             throws RemoteException,CreateException;

    public Customer create(Integer customerNumber,
                                        Name name)
            throws RemoteException,CreateException;

    public Customer findByPrimaryKey(Integer
                               customerNumber)
          throws RemoteException, FinderException;

    public Enumeration findByZipCode(int zipCode)
          throws RemoteException, FinderException;
}
```

Unten sehen Sie ein Beispiel für den Einsatz des home Interfaces durch eine Anwendung, um einen neuen Kunden, also eine Instanz der Klasse Customer zu kreieren:

```
CustomerHome home =
// stellt eine Referenz auf das
// CustomerHome Objekt her

Name name =
   new Name("John", "W", "Smith");

Customer customer =
   home.create(
      new Integer(33), name);
```

Das home Interface einer Bohne kann kein oder mehrere `create`() Methoden deklarieren, denen `ejbCreate`() und `ejbPostCreate`() Methoden in der Bean Kasse entsprechen müssen. Diese `create`() Methoden werden zur Laufzeit gelinkt (der Bohne zur Verfügung gestellt), so dass wann immer die `create`() Methoden des home Interface aufgerufen werden der Container den Aufruf an die entsprechenden Methoden `ejbCreate`() und `ejbPostCreate`() der Bean Klasse weiterleitet:

Wenn eine `create`() Methode eines home Interfaces aufgerufen wird, delegiert der Container den Aufruf der `create`() Method an die `ejbCreate`() Method der Instanz der Bohne. Die `ejbCreate`() Methoden werden eingesetzt, um beispielsweise einen Datensatz zu initialisieren, bevor er in die Datenbank eingefügt wird.

In unserem Beispiel werden die Datenfelder `customerID` und `Name` initialisiert. Nach dem Abschluss der `ejbCreate`() Methode (sie liefert im Falle von CMP ein null Objekt zurück) liest der Container die Container-gemanagten Datenfelder und fügt einen neuen Datensatz in die CUSTOMER Tabelle ein. Diese Tabelle ist in unserem Beispiel indexiert mit dem Primärschlüssel customerID dem die Spalte CUSOTMER.ID in der Tabelle entspricht.

In EJB existiert ein Entity Bean technisch erst nachdem seine Daten in die Datenbank eingefügt wurden und das geschieht mit Hilfe der ejbCreate() Methode. Nachdem die Daten einmal eingefügt wurden, existiert die Entity Bean, und es kann mit Hilfe ihres Primärschlüssels und einer remote Referenz auf sie zugegriffen werden; dies ist aber erst möglich nachdem die `ejbCreate`() Methode die Daten vollständig in die Datenbank geschrieben hat.

Ein Bean kann auch bereits auf seinen Priärschlüssel oder remote Referenzen zugreifen bevor sie irgendwelche Business Methoden zur Verfügung stellt und zwar in der `ejbPostCreate()` Methode. Die `ejbPostCreate()` Methode erlaubt der Bohne irgendwelche Verwaltungsaufgaben zu erfüllen, bevor Clients Anfragen beantwortet werden. Für jede `ejbCreate()` muss es eine entsprechende `ejbPostCreate()` Methode geben, wobei 'entsprechend' bedeutet, dass beide Methoden die selbe Signatur besitzen (Argumente, Argumentetyp).

Die Methoden des home Interface, welche mit "find" beginnen, nennt man ... (was wohl) ... 'find' Methoden. Mit Hilfe dieser Methoden kann man den EJB Server abfragen und spezifische Entity Beans suchen, basierend auf dem Namen der Methode und den Argumenten. Unglücklicherweise gibt es keine Standardabfragespreche für die 'find' Methoden. Die Folge ist, dass jeder Anbieter die 'find' Methoden anders implementieren wird.

In CMP Entity Beans werden die 'find' Methoden ohne korrespondierende Methoden in der Bean Klasse implementiert; Container implementieren diese in einer herstellerabhängigen Art und Weise. Der Entwickler wird das herstellerspezifische Werkzeug einsetzen, um dem Container mitzuteilen, wie sich eine bestimmte Methode verhalten soll. Einige Hersteller werden Werkzeuge für die Objekt-Relationale Abbildung verwenden, um das Verhalten der 'find' Methoden zu implementieren. Andere Hersteller werden einfach SQL einsetzen.

Es gibt grundsätzlich zwei Ausprägungen der 'find' Methoden: single-entity und multi-entity 'find' Methoden:

- Single-entity 'find' Methoden liefern eine remote Referenz auf genau eine Entity Bohne, die den Selektionskriterien entspricht. Falls keine Entity Beans gefunden wurde, wirft die Methode eine ObjectNotFoundException. Jede Entity Bean **muss** die single-entity 'find' Methode mit der Methode `findByPrimaryKey()` implementieren. Diese Methode verwendet den Primärschlüssel der Bean als Argument.

  Im obigen Beispiel verwendeten wir die `Integer` Klasse, welche den primitiven Datentyp des Feldes `id` in der Bean Klasse repräsentiert.

- Die multi-entity 'find' Methoden liefern eine Collection (Enumeration oder Collection Typ) von Entitäten, welche alle die Suchkriterien erfüllen. Falls keine Entität gefunden wird, gibt die multi-entity 'find' Methode eine leere Collection zurück.

  Zu beachten ist, dass eine leere Collection *nicht* das selbe ist, wie eine `null` Referenz.

1.1.7.1.3. Remote Interface

Jede Entity Bean muss ein remote Interface definieren, zusätzlich zu dem home Interface. Das remote Interface definiert die Business Methoden der Entity Bean.

Und so sieht die Definition des remote Interface für die Customer Bean aus:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Customer
          extends EJBObject {
```

```
    public Name getName()
                throws RemoteException;

    public void setName(Name name)
                throws RemoteException;

    public Address getAddress()
                   throws RemoteException;

    public void setAddress(
                 Address address)
                throws RemoteException;

    public CreditCard getCreditCard()
                      throws RemoteException;

    public void setCreditCard(CreditCard card)
                throws RemoteException;
}
```

Ein praktisches Beispiel für den Einsatz dieser Klasse sehen Sie in folgendem Programmfragment (Client Applikation verwendet das remote Interface, um mit einer Bohne zu kommunizieren):

```
Customer customer =
// ... remote
//reference auf die Bohne

// Customer Adresse
Address addr = customer.getAddress();

// zip code
addr.zip = "56777";

// mutieren der Customer Adresse
customer.setAddress(addr);
```

Die Business Methoden im remote Interface werden an die entsprechende Business Methoden in der Bean Instanz weitergeleitet.

Im Falle der Customer Bean sind die Business Methoden alle einfache Zugriffs- und Mutationsmethoden; aber die Situation könnte wesentlich komplexer sein.

In andern Worten: die Business Methoden einer Entität sind nicht darauf beschränkt zu lesen und zu mutieren; sie könnten auch komplexe Berechnungen und Aufgaben ausführen.

Falls der Kunde (`Customer`) Mitglied eines Loyalitätsprogramm wäre ("Golden Card", Joker Programm, ...), dann könnte die Mutationsmethode beispielsweise die Anzahl Übernachtungen zusammenzählen.

Als Programmskizze hätten wir dann:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Customer extends EJBObject {
```

```
    public MembershipLevel addNights(int nights_stayed)
                           throws RemoteException;

    public MembershipLevel upgradeMembership()
                           throws RemoteException;

    public MembershipLevel getMembershipLevel()
                           throws RemoteException;

    ... einfache Business Methoden
}
```

Die `addNights()` und `upgradeMembership()` Methoden sind komplexer als die einfache Zugriffsmethode, da sie komplexere Beschäftsregeln anwenden und über reines Lesen und Schreiben hinausgehen.

1.1.7.1.4. Callback Methods

Die Bean Klasse defininiert 'create' Methoden, die den Methoden im home Interface entsprechen, zudem Business Method, welche Methoden im remote interface entsprechen. Die Bean Klasse implementiert auch ein Set von *callback* Methoden, mit deren Hilfe der Container die Bean über Ereignisse in ihrem Lifecycle informiert. Die Callback Methoden werden im javax.ejb.EntityBean Interface definiert. Dieses Interface wird von allen Entity Beans implementiert, inklusive der CustomerBean Klasse. Das EntityBean Interface wird wie im folgenden Programmcode defininiert.

Beachten Sie, dass die Bean Klasse diese Methoden implementiert.

```
public interface javax.ejb.EntityBean {

    public void setEntityContext();

    public void unsetEntityContext();

    public void ejbLoad();

    public void ejbStore();

    public void ejbActivate();

    public void ejbPassivate();

    public void ejbRemove();
}
```

Die `setEntityContext()` Methode stellt der Bean ein Interface zum Container `EntityContext` zur Verfügung. Das EntityContext Interface enthält Methoden, um Informationen über den Kontext zu erhalten, unter denen die Bohne zur Zeit agiert.

Das `EntityContext` Interface wird eingesetzt,

- um Sicherheitsinformationen über den Aufrufer zu erhalten;
- um den Status einer laufenden Transaktion zu erfahren oder eine Transaction rückgängig zu machen (rollback);
- oder um eine Referenz auf die Bohne zu erhalten, ihr home oder ihren Primärschlüssel.

Der EntityContext wird pro Lebenszyklus einer Entity Bean Instanz nur einmal gesetzt. Damit wird die Referenz in eines der Instanzendatenfelder der Bohne abgelegt, damit sie später verwendet werden kann, falls nötig.

Die Customer Bean benutzt den EntityContext nicht, könnte aber.

Zum Beispiel:
sie könnte den `EntityContext` verwenden, um die Zugehörigkeit eines Aufrufers zu einer bestimmten Berechtigungsgruppe zu überprüfen.

Im folgenden Programmfragment sehen Sie, wie der `EntityContext` benutzt wird, um zu verifizieren, dass der Aufrufer ein Manager ist.

```
import javax.ejb.EntityBean;

public class CustomerBean
         implements EntityBean {

    int           customerID;
    Address       myAddress;
    Name          myName;
    CreditCard    myCreditCard;
    EntityContext ejbContext;

    // CALLBACK METHODS
    public void setEntityContext(
                        EntityContext cntx) {
        ejbContext = cntx
    }

    public void unsetEntityContext() {
        ejbContext = null;
    }

    // BUSINESS METHODS
    public void
         setCreditCard(CreditCard card) {
        if (card.type.equals("WorldWide"))
            if (ejbContext.isCallerInRole(
                                "Manager"))
                myCreditCard = card;
            else
                throw new SecurityException();
        else
            myCreditCard = card;
    }

    public CreditCard getCreditCard() {
        return myCreditCard;
    }
    ...
}
```

Die `unsetEntityContext()` Methode wird am Ende des Lebenszyklus der Bohne eingesetzt - bevor die Instanz aus dem Speicher entfernt werden -um den EntityContext zu dereferenzieren und 'last minute' Aufräumarbeiten zu erledigen.

Die `ejbLoad()` und `ejbStore()` Methoden der CMP Entitiäten werden benutzt, um den Zustand der Entity Bean mit der Datenbank zu synchronisieren. Die `ejbLoad()` Methode wird eingesetzt nachdem der Container die Container-gemanagten Datenfelder der Bohne mit dem gespeicherten Zustand in der Datenbank abgeglichen hat. Die `ejbStore()` Methode wird aufgerufen, bevor der Container die Container-gemanagten Datenfelder der Bohne in die Datenbank schreibt. Diese Methoden werden eingesetzt, um Daten, welche synchronisiert werden müssen, zu modifizieren. Die geschieht immer dann, wenn die Datenwerte in der Bohne sich von den Werten in der Datenbank (zur entsprechenden Bohne) unterscheiden. Die Methoden können zum Beispiel auch eingesetzt werden, um die Daten zu komprimieren, bevor sie in die Datenbank geschrieben werden bzw. zu dekomprimieren, falls sie aus der Datenbank gelesen werden.

In unserem Beispiel mit den Kunden und der 'Customer' Bean könnten die Methoden `ejbLoad()` und `ejbStore()` eingesetzt werden, um abhängige Objekte (`Name, Address, CreditCard`) in einfache `String` Objekte und primitive Datentypen umzuwandeln, falls der EJB Container nicht funktional reich genug ist, um die abhängigen Objekte auf die CUSTOMER Tabelle abzubilden.

Das Programmfragment zeigt, wie die Bohne modifiziert werden könnte:

```
import javax.ejb.EntityBean;

public class CustomerBean implements EntityBean {

    //container-managed Datenfelder
    int    customerID;
    String lastName;
    String firstName;
    String middleName;
    ...

    // nicht-container-managed Datenfelder
    Name       myName;
    Address    myAddress;
    CreditCard myCreditCard;


    // BUSINESS METHODS
    public Name getName() {
        return myName;
    }

    public void setName(Name name) {
        myName = name;
    }
    ...

    public void ejbLoad() {

        if (myName == null)
            myName = new Name();

        myName.lastName = lastName;
        myName.firstName = firstName;
        myName.middleName = middleName;
        ...
    }
```

```
    public void ejbStore() {

        lastName   = myName.lastName;
        firstName  = myName.firstName;
        middleName = myName.middleName;
        ...
    }
}
```

Die `ejbPassivate()` und `ejbActivate()` Methoden werden vom Container direkt vor der Deaktivierung ('Passivierung') bzw. direkt nach der Aktivierung der Bohne durch den Containeraufzurufen. *Passivierung* eines Entity Beans bedeutet, dass die Bean Instanz von ihrer remote Referenz disassoziiert wird, so dass der Container die Bohne aus dem Speicher entfernen oder wiederverwenden kann. Die Methoden helfen also die Ressourcen zu optimieren und einzuschränken. Eine Bean kann 'passiv' gesetzt werden ('passivated') falls sie während einer Weile nicht benutzt wurde oder als eine normale Operation des Containers, um die Wiederverwendung der Ressourcen zu maximieren. Einige Container werden Beans aus dem Speicher entfernen, andere werden Instanzen für andere aktive remote Referenzen wiederverwenden.

Die `ejbPassivate()` und `ejbActivate()` Methoden informieren die Bean 'passivated' (disassoziiert von der remote Referenz) oder 'activated' (assoziiert mit einer remote Referenz) werden.

1.1.7.1.5. Übung : Kreieren einer Entity Bean

Lösen Sie jetzt die Übungsaufgabe 2.

1.1.7.2. Bean-Managed Persistence (BMP)

Die Bean-managed Persistence (BMP) Enterprise Bean managed die Synchronisation ihres Zustands mit der Datenbank so wie der Container dies vorschlägt. Die Bean benutzt ein Datenbank API (normalerweise JDBC) zum lesen und schreiben ihrer Datenfelder aus / in die Datenbank, aber der Container teilt ihr mit, wann die Synchronisationsoperation zu erledigen sind, und der Container managed die Transaktionen für die Bean automatisch. 'Bean-managed persistence' (BMP) gibt dem Bean Entwickler die Flexibilität Persistenzoperationen auszuführen, die für den Container zu komplex wären oder die Datenquellen benutzen, die vom Container nicht unterstützt werden - kundenspezifische Datenbaken zum Beispiel.

In diesem Abschnitt werden wir die `CustomerBean` Klasse zu einer Bean-Managed Persistence Bean modifizieren. Diese Modifikation hat keinen Einfluss auf die remote oder home Interfaces. Wir werden die ursprüngliche `CustomerBean` nicht direkt modifizieren. Statt dessen werden wir die Klasse erweitern und die Methoden überschreiben. Sie finden unten die Definition der Klasse, welche die Klasse Customer Bean erweitert und aus ihr eine BMP Entität macht. In den meisten Fällen wird man allerdings nicht so vorgehen. Statt eine bestehende Bohne zu erweitern würde man in der Praxis direkt eine BMP Bohne definieren und implementieren.

Unser Vorgehen, die Erweiterung einer CMP Bean zu einer BMP Bean, bietet sich hier aus zwei Gründen an:

1. unsere Bohne kann anschliessend sowohl eine CMP als auch eine BMP Bean sein;
2. zudem ist der zusätzliche Codieraufwand minimal

Sie sehen dies in der folgenden Definition der BMP Klasse:

```
public class CustomerBean_BMP
          extends CustomerBean {
    public void ejbLoad() {
        // override implementation
    }
    public void ejbStore() {
        // override implementation
    }
    public void ejbCreate() {
        // override implementation
    }
    public void ejbRemove() {
        // override implementation
    }
    private Connection getConnection() {
        // new helper method
    }
}
```

Im Falle der BMP Beans werden die `ejbLoad()` und `ejbStore()` Methoden vom Container und der Bohne anders eingesetzt als im Falle der CMP. Im BMP Fall enthalten die `ejbLoad()` und `ejbStore()` Methoden Code zum lesen der Beans Daten aus der Datenbank, und zum Schreiben der Änderungen in die Datenbank. Diese Beans Methoden werden dann aufgerufen, wenn der EJB Server entscheidet, es wäre an der Zeit Daten zu lesen oder zu schreiben.

Die `CustomerBean_BMP` Bean managed ihre Persistenz selbst. In andern Worten, die `ejbLoad()` und `ejbStore()` Methoden müssen die Datenbankzugriffslogik enthalten, so dass die Bohne ihre Daten laden und spiechern kann, wann immer der Container dies von ihr verlangt. Der Container wird die Methoden `ejbLoad()` und `ejbStore()` ausführen, wenn dies nötig ist.

Die `ejbLoad()` Methode wird normalerweise zu Beginn einer Transaktion vom Container aufgerufen, bevor der Container eine Business Methode an die Bean delegiert.

Das folgende Programmbeispiel zeigt eine `ejbLoad()` Methode, welche JDBC einsetzt.

```
import java.sql.Connection;

public class CustomerBean_BMP
             extends CustomerBean {

  public void ejbLoad() {
    Connection con;
    try {
      Integer primaryKey =
       (Integer)ejbContext.getPrimaryKey();
      con = this.getConnection();
      Statement sqlStmt =
      con.createStatement("SELECT *
                      FROM Customer " +
                      " WHERE customerID = " +
                      primaryKey.intValue());
      ResultSet results = sqlStmt.executeQuery();
      if (results.next()) {
        // lesen der name Information
```

```
        // aus der customer Tabelle
        myName = new Name();
        myName.first = results.getString("
                                  FIRST_NAME");
        myName.last = results.getString("
                                   LAST_NAME");
        myName.middle = results.getString("
                                 MIDDLE_NAME");
        // lesen der Adress Information aus
        // der customer Tabelle
        myAddress = new Address();
        myAddress.street =
            results.getString("STREET");
        myAddress.city =
            results.getString("CITY");
        myAddress.state =
            results.getString("STATE");
        myAddress.zip = results.getInt("ZIP");
        // Credit card Information
        // aus der customer Tabelle
        myCreditCard = new CreditCard();
        myCreditCard.number =
           results.getString("CREDIT_NUMBER");
        myCreditCard.expDate =
             results.getString("CREDIT_DATE");
        myCreditCard.type =
             results.getString("CREDIT_TYPE");
        myAddress.name =
               results.getInt("CREDIT_NAME");
      }
    }
    catch (SQLException sqle) {
      throw new EJBException(sqle);
    }
    finally {
      if (con!=null)
        con.close();
    }
  }
}
```

In der `ejbLoad()` Methode wird die `ejbContext()` Referenz auf den `EntityContext` der Bohne eingesetzt, um den Primärschlüssel der Instanz zu erhalten. Dies garantiert, dass der korrekte Index auf die Datenbank verwendet wird.

Die `CustomerBean_BMP` wird die erweiterten `setEntityContext()` und `unsetEntityContext()` Methods einsetzen. Die `ejbStore()` Methode wird vom Container aufgerufen, als Beanmethode, am Ende der Transaktion, bevor der Container alle Änderungen in der Datenbank bestätigt.

```
import java.sql.Connection;

public class CustomerBean_BMP
                extends CustomerBean {

  public void ejbLoad() {
    ... Lesen der Daten eus der Datenbank
  }
```

```
  public void ejbStore() {
    Connection con;
    try {
      Integer primaryKey =
        (Integer)ejbContext.getPrimaryKey();
      con = this.getConnection();
      PreparedStatement sqlPrep =
      con.prepareStatement(
         "UPDATE customer set " +
         "last_name = ?, first_name = ?,
                    middle_name = ?, " +
         "street = ?, city = ?, state = ?,
            zip = ?, " +
         "card_number = ?, card_date = ?, " +
         "card_name = ?, card_name = ?, " +
         "WHERE id = ?"
      );
      sqlPrep.setString(1,myName.last);
      sqlPrep.setString(2,myName.first);
      sqlPrep.setString(3,myName.middle);
      sqlPrep.setString(4,myAddress.street);
      sqlPrep.setString(5,myAddress.city);
      sqlPrep.setString(6,myAddress.state);
      sqlPrep.setString(7,myAddress.zip);
      sqlPrep.setInt(8, myCreditCard.number);
      sqlPrep.setString(9, myCreditCard.expDate);
      sqlPrep.setString(10, myCreditCard.type);
      sqlPrep.setString(11, myCreditCard.name);
      sqlPrep.setInt(12,primaryKey.intValue());
      sqlPrep.executeUpdate();
    }
    catch (SQLException sqle) {
      throw new EJBException(sqle);
    }
    finally {
      if (con!=null)
        con.close();
    }
  }
}
```

In beiden Methoden, `ejbLoad()` und `ejbStore()`, synchronisiert die Bohne ihren Status mit der Datenbank mit Hilfe von JDBC. Falls Sie den obigen Programmteil genau gelesen haben, sollte Ihnen nicht entgangen sein, dass die Bohne die Datenbankverbindung von der Methode `this.getConnection()` erhält. Diese Methode muss noch implementiert werden. Die `getConnection()` Methode ist keine Standard- EJB Methode; es ist eine private Helpermethode, deren Aufgabe darin besteht eine Datenbankverbindung herzustallen. Eine mögliche Implementation der `getConnection()` Methode könnte etwa folgendermassen aussehen:

```
import java.sql.Connection;

public class CustomerBean_BMP
    extends CustomerBean {

  public void ejbLoad() {
    ... lesen der Daten aus der Datenbank
  }
```

```
  public void ejbStore() {
    ... schreiben der Daten in die Datenbank
  }

  private Connection getConnection()
         throws SQLException {

     InitialContext jndiContext =
         new InitialContext();
     DataSource source = (DataSource)
                jndiContext.lookup("
          java:comp/env/jdbc/myDatabase");
     return source.getConnection();
  }
}
```

Die Datenbankverbindungen werden mit Hilfe des Default- JNDI Context, dem JNDI Environment Naming Context (ENC) gefunden. ENC stellt eine Verbindung her zu transaktionsorientierten JDBC Connections. Dies geschieht mit Hilfe der Standard Connection Factory, der javax.sql.DataSource Klasse.

In BMP werden die Methoden `ejbLoad()` und `ejbStore()` vom Container aufgerufen, um die Bean Instanz mit den Daten in der Datenbank zu synchronisieren. Um Entitäten in die Datenbank einzufügen oder daraus zu entfernen, implementieren die Methoden `ejbCreate()` und `ejbRemove()` ähnliche Datenbankzugriffslogiken. Die `ejbCreate()` Methoden implementieren das Einfügen einer neuen Zeile, eines neuen Datensatzes in die Datenbankund die `ejbRemove()` Methoden werden so implementiert, dass sie die Daten einer Entität aus der Datenbank entfernen. Die `ejbCreate()` und die `ejbRemove()` Methode einer BMP Entity werden vom Container aufgerufen, als Reaktion auf einen Aufruf der entsprechenden Methoden in den home und remote Interfaces. Eine mögliche Implementation dieser Methoden könnte folgendermassen aussehen:

```
 public void ejbCreate(Integer id) {
    this.customerID = id.intValue();
    Connection con;
    try {
      con = this.getConnection();
      Statement sqlStmt =
      con.createStatement("INSERT INTO
               customer id VALUES (" +
                          customerID + ")");
      sqlStmt.executeUpdate();
      return id;
    }
    catch(SQLException sqle) {
      throw new EJBException(sqle);
    }
    finally {
      if (con!=null)
        con.close();
    }
  }

  public void ejbRemove() {
    Integer primaryKey =
     (Integer)ejbContext.getPrimaryKey();
    Connection con;
    try {
      con = this.getConnection();
```

```
      Statement sqlStmt =
      con.createStatement("DELETE FROM
                   customer WHERE id = "
                  primaryKey.intValue());
      sqlStmt.executeUpdate();
    }
    catch(SQLException sqle) {
      throw new EJBException(sqle);
    }
    finally {
      if (con!=null)
        con.close();
    }
  }
```

Im Falle BMP (Bean Managed Persistenz) ist die Bean Klasse für die Implementation der 'find' Methoden zusändig, die im home Interface definiert sind. Für jede 'find' Methode im home Interface muss eine entsprechende `ejbFind()` Methode in der Bean Klasse definiert sein. Die `ejbFind()` Methoden lokalisieren den passenden Bean Datensatz in der Datenbank und liefern deren Primärschlüssel an den Container. Der Container konvertiert die Primärschlüssel in Bean Referenzen und liefert diese an den Client. Unten sehen Sie an einem Beispiel, wie eine `ejbFindByPrimaryKey()` Methode in der `CustomerBean_BMP` Klasse implementiert werden könnte. Diese Methode entspricht der Methode `findByPrimaryKey()` im `CustomerHome` Interface.

```
public Integer ejbFindByPrimaryKey(
                  Integer primaryKey)
               throws ObjectNotFoundException {
    Connection con;
    try {
        con = this.getConnection();
        Statement sqlStmt =
            con.createStatement("SELECT *
                            FROM Customer " +
                            " WHERE customerID = " +
                            primaryKey.intValue());

        ResultSet results = sqlStmt.executeQuery();
        if (results.next())
            return primaryKey;
        else
            throw ObjectNotFoundException();
    }
    catch (SQLException sqle) {
        throw new EJBException(sqle);
    }
    finally {
        if (con!=null)
        con.close();
    }
}
```

Single-entity 'find' Methoden wie die oben definierte liefern einen einzigen Primärschlüssel oder werfen die ObjectNotFoundException falls kein Datensatz gefunden wurde.

Multi-entity 'find' Methoden liefern eine Collection (java.util.Enumeration oder java.util.Collection) mit Primärschlüsseln. Der Container konvertiert diese Collection in eine Collection von remote Referenzen, welche an den Client geliefert werden.

Unten sehen Sie ein Beispiel dafür, wie die multi-entity `ejbFindByZipCode()` Methode implementiert werden könnte. Diese Methode entspricht der `findByZipCode()` Methode, die im `CustomerHome` Interface definiert ist und in der `CustomerBean_BMP` Klasse implementiert wird.

```
public Enumeration ejbFindByZipCode(
                      int zipCode)  {
    Connection con;
    try {
        con = this.getConnection();
        Statement sqlStmt =
        con.createStatement("SELECT id
                     FROM Customer " +
              " WHERE zip = " +zipCode);

        ResultSet results =
                sqlStmt.executeQuery();
        Vector keys = new Vector();
        while(results.next()){
            int id = results.getInt("id");
            keys.addElement(new Integer(id));
        }
        return keys.elements();
    }
    catch (SQLException sqle) {
        throw new EJBException(sqle);
    }
    finally {
        if (con!=null)
        con.close();
    }
}
```

Falls keine Bean Datensätze gefunden wurden, wird eine leere Collection an den Container zurückgegeben. Und dieser liefert eine leere Collection an den Client.

Mit der Implementierung all dieser Methoden und ein paar kleineren Änderungen am Bean Deployment Descriptor (grob als "Einsatzbeschreibung" übersetzbar) ist `CustomerBean_BMP` bereit als BMP Entität eingesetzt zu werden.

1.1.8. Session Type Enterprise Beans

Session Beans werden eingesetzt, um die Wechselwirkung zwischen Entity und andern Session Beans zu managen, auf Ressource zuzugreifen undallgemeine Arbeiten im Dienste des Clients zu erledigen.

Session Beans sind keine persistente Business Objekte (wie es die Entity Beans sind). Sie repräsentieren also keine Daten in einer Datenbank.

Session Beans entsprechen dem Controller in einer Model-View-Controller Architektur, da sie die Businesslogik einer three-tier Architektur implementiert.

Es gibt grundsätzlich zwei Arten von Session Bean: Stateless und Stateful, zustandslose und solche mit einem Zustand (und damit Datenfeldern):

- Zustandslose Session Beans bestehen aus Business Methoden, die sich wie Prozeduren verhalten; sie agieren lediglich mit den Argumenten, die an sie beim Aufruf übergeben werden. Zustandslose Beans werden als "stateless" bezeichnet, weil sie transient sind (temporär); sie speichern den Businesszustand zwischen einzelnen Methodenaufrufen nicht.

- zustandsbehaftete Session Beans kapseln Businesslogik und Zustandsinformationen zu einem Client. Zustandsbehaftete Beans werden "stateful" genannt, weil sie den Zustand zwischen den einzelnen Aufrufen speichern, allerdings im Hauptspeicher, nicht persistent in einer Datenbank.

1.1.8.1. Stateless Session Beans

Stateless Session Beans sind nichtpersistente Businessobjekte, wie die Entity Beans. Sie repräsentieren keine Daten aus einer Datenbank.Statt dessen repräsentieren diese Beans Prozesse oder Tasks, die im Auftrag eines Client ausgeführt werden. Die Businessmethoden einer stateless Session Bean agiert also eher wie eine klassische Prozedur eines ganz normalen klassischen Programms oder einem Transaktionsminitor. Jeder Aufruf einer stateless Businessmethode ist unabhängig vom vorausgehenden Aufruf. Weil stateless Session Beans "stateless" sind, kann sie der EJB Container leichter managen. Solche Aufrufe sind in der Regel auch schneller ausführbar und benötigen weniger Ressourcen. Aber diese Performance hat ihren Preis : stateless Session Beans kennen ihren Zustand aus einem früheren Methodenaufruf im nächsten Aufruf schon nicht mehr.

Als Beispiel einer stateless Session Bean ist eine `CreditService` Bean, welche einen Kreditservice repräsentieren könnte. Seine Aufgabe wäre die Validation und Verarbeitung der Kreditkarten Gebühren. Eine Hotelkette könnte eine CreditService Bean entwickeln, um den Prozess:

"Verifikation einer Kreditkartennummer, belasten der Kreditkarte und speichern der Belastung in der Kontodatenbank"

Und so könnte das remote und home Interface für die CreditService Bean aussehen.

```
// remote Interface
public interface CreditService
    extends javax.ejb.EJBObject {
    public void verify(CreditCard
                card, double amount)
    throws RemoteException,
          CreditServiceException;
    public void charge(CreditCard
               card, double amount)
    throws RemoteException,
             CreditServiceException;
}

// home Interface
public interface CreditServiceHome
         extends java.ejb.EJBHome {
     public CreditService create()
           throws RemoteException,
                     CreateException;
}
```

Das remote Interface, `CreditService`, definiert zwei Methoden, `verify()` und `charge()`, die vom Hotel für die Verifikation und Belastung der Kreditkarden eingesetzt werden. Das Hotel könnte die `verify()` Method benutzen, um Reservationen vorzunehmen, ohne die Karte echt zu belasten. Die `charge()` Method würde eingesetzt für die aktuelle Belastung eines Kundenkontos. Das home Interface, `CreditServiceHome` stellt eine `create()` Methode ohne Argumente zur Verfügung. Alle home Interfaces für stateless Session Beans definieren genau eine Methode, die `create()` Method ohen Argumente, weil Session Beans keine 'find' Methoden besitzen und eine Initialisierung keinen Sinn macht, da die Beans ja keinen Zustand speichern. Stateless Session Beans beitzen keine 'find' Methods weil alle stateless Beans äquivalent und zustandslos sind. Jeder Client , der eine session Bean einsetzt, bakommt den selben Service.

Unten sehenSie die Definition einer CreditService Bean. Diese Bohne kapselt den Zugriff auf den Acme Credit Card Verarbeitungsdienst. Diese Bohne greift im Speziellen greift die Bean auf den Acme Secure Web Server zu und 'posted' Vaidations- oder Belastungsaufträge für die Kundenkreditkarte.

```
import javax.ejb.SessionBean;

public class CreditServiceBean
        implements SessionBean {
    URL acmeURL;
    HttpURLConnection acmeCon;

    public void ejbCreate() {
        try {
            InitialContext jndiContext =
                  new InitialContext();
            URL acmeURL = (URL)
            jndiContext.lookup("
               java:comp/ejb/env/url/acme");
            acmeCon =
              acmeURL.openConnection();
        }
        catch (Exception e) {
            throws new EJBException(e);
        }
    }

    public void verify(CreditCard card,
                       double amount) {
        String response = post("verify:" +
                      card.postString() +
                           ":" + amount);
        if (response.substring("
                       approved")== -1)
            throw new
             CreditServiceException("denied");
    }

    public void charge(CreditCard card,
                            double amount)
                throws CreditCardException {
        String response = post("charge:" +
                         card.postString() +
                              ":" + amount);
        if (response.substring("approved")== -1)
            throw new CreditServiceException("
```

```
                                        denied");
    }

    private String post(String request) {
        try {
            acmeCon.connect();
            acmeCon.setRequestMethod("
                        POST "+request);
            String response =
              acmeCon.getResponseMessage();
        }
        catch (IOException ioe) {
            throw new EJBException(ioe);
        }
    }

    public void ejbRemove() {
        acmeCon.disconnect();
    }

    public void setSessionContext(
            SessionContext cntx) {}

    public void ejbActivate() {}

    public void ejbPassivate() {}
}
```

Die `CreditService` stateless Bean demonstriert, dass eine stateless Bean eine ganze Kollektion unabhängiger aber verwandte Services umfassen kann. Im obigen Fall sind die Kreditkarten Validation und Belastung verwandt aber ncht notwendigerweise unabhängig. Stateless Beans können eingesetzt werden, um auf spezielle Ressource zuzugreifen - in unserem Fall der `CreditService` Bean - beispielsweise Datenbanken oder auch im komplexe Berechnungen durchzuführen. Die `CreditService` Bean wird hier eingesetzt, um die Servicenatur der stateless Bean zu demonstrieren und einen Kontext für die Diskussion der Callback Methoden zu haben. Stateless Session Beans Einsätze beschränken sich jedoch nicht auf diese Fälle. Stateless Beans kann man für beliebige Serviceaufgaben einsetzen.

Die `CreditServiceBean` Klasse benutzt eine URL Ressource Factory (`acmeURL()`) um eine Referenz auf den Acme Web Server zu erhalten, welcher auf einem weit entfernten Rechner installiert ist und aktiv ist. Die `CreditServiceBean` benutzt das `acmeURL()` Objekt, um eine Verbindung mit dem Webserver aufzubauen und Anfragen zu 'posten' insbesondere Kreditkartenvalidation. Die `CreditService` Bean wird von den Clients statt einer direkten Verbindung zu diesen Diensten eingesetzt und kann vom EJB Container besser gemaneged werden.

Die `ejbCreate()` Methode wird zu Anfang des Lebenszyklus der Bean einmal und nur einmal aufgerufen. Die `ejbCreate()` Methode wird sinnvollerweise eingesetzt, um Ressourceverbindungen und Variablen zu initialisieren, die von der zustandslosen Bohne während ihres Lebenszyklus eingesetzt werden.

Im obigen Beispiel unserer `CreditServiceBean` benutzt diese die Methode `ejbCreate()` um eine Referenz auf die `HttpURLConnection` Factory zu erhalten, die während der Lebensdauer der Bohne eingesetzt wird, um Verbindungen auf den Acme Web Server zu erhalten.

Die `CreditServiceBean` benutzt den JNDI ENC um eine URL Connection Factory zu erhalten, genau wie die `CustomerBean` das JNDI ENC benutzte, um eine `DataSource` Resource Factory für JDBC Connections zu erhalten. Die JNDI ENC ist ein default JNDI Context auf den alle Beans Zugriff haben. JNDI ENC um auf statische Eigenschaften, andere Beans und Ressource Factories wie die java.net.URL und JDBC javax.sql.DataSource zuzugreifen. Zusätzlich gestattet JNDI ENC auch den Zugriff auf JavaMail und Java Messaging Service Ressource Factories.

Die `ejbCreate`() und `ejbRemove`() Methoden werden nur je einmal im Lebenslauf der Bean aufgerufen, nämlich dann, wenn sie kreiert oder zerstört werden. Falls die `create`() und `remove`() Methoden im Rahmen der home und remote Interfaces des Clients eingesetzt werden, impliziert dies nicht, dass auch die Methoden `ejbCreate`() und `ejbRemove`() der Bean Instanz ebenfalls aufgerufen werden. Ein Methodenaufruf von create() liefert dem Client eine Referenz auf eine zustandslose Bean, ein Aufruf der Methode `remove`() entfernt die Referenz. Der Container entscheidet, wann eine Beaninstanz kreiert oder zerstört wird. Der Container ruft auch die entsprechenden Methoden `ejbCreate`() und `ejbRemove`() auf. Damit kann eine zustandslose Instanz einer Bean von mehreren Clients benutzt werden.

Im Falle der `CreditServiceBean` werden die `ejbCreate`() und `ejbRemove`() Methoden eingesetzt, um eine URL Verbindung am Beginn des Lebenszyklus der Bohne verzustellen und am Ende des Lebenszyklus die Verbindung aufzutrennen (dazwischen wird die URL Verbindung von den Business Methoden genutzt). Ziel dieses Mechanismus ist es, die Verbindungen optimal zu nutzen und dadurch Resourcen zu optimieren. Zustandlose Beans sind für solche Einsätze (fixe Funktion, unterschiedliche Clients) optimal geeignet.
.
In unserem Kreditkartenbeispiel delegierendie Methoden `verify`() und `charge`() die konkrete Ausführung an die Methoden `post`() als Helper Methode. Die post() Methode ihrerseits benutzt die HttpURLConnection, um die Kreditkarteninformationen an den Acme Web Server zu übermitteln. Die Ergebnisse werden an die Methoden verify() oder charge() zurückgeliefert. Da die HttpURLConnection automatisch vom Container getrennt werden kann, zum Beispiel, weil längere Zeit keine Anfragen bearbeitet werden mussten, ist das Programm so aufgebaut, dass die post() Methode immer die connect() Methode aufruft. Sollte bereits eine Verbindung bestehen, macht dies nicht. Die verify() und charge() Methoden prüfen den Rückgabewert und falls ein Substring "approved" gefunden wird, zeigt dies, dass die Kreditkarte akzeptiert wurde. Falls "approved" nicht gefunden wird, muss angenommen werden, dass die Kreditkarte abgelehnt wurde und eine Business Exception geworfen werden muss.

Die setSessionContext() Methode liefert der Bean Instanz einen SessionContext, der die selbe Funktion hat wie der EntityContext im Falle der CustomerBean im Abschnitt 'Entity beans'. Der SessionContext wird in unserem Beispiel nicht benutzt.
.
Die ejbActivate() und ejbPassivate() Methoden fehlen in der `CreditService` Bean weil 'passivation' in zustandslosen Session Beans nicht benötigt werden. Diese Methoden werden im Interface javax.ejb.SessionBean für zustandabehaftete Session Beans eingesetzt. Deswegen muss für zustandslose Session Beans eine (nötigenfalls leere) Implementation vorhanden sein. Zustandslose Session Bean werden für diese Methoden immer nur leere Implementation zur Verfügung stellen.

Zustandslose Session Beans können auch benutzt werden, um auf Datenbanken zuzugreifen oder die Wechselwirkung zwischen Beans koordinieren.

Das Beispiel `HotelClerkBean` illustriert dies:

```
import javax.ejb.SessionBean;
import javax.naming.InitialContext;

public interface HotelClerkBean
          implements SessionBean {

    InitialContext jndiContext;

    public void ejbCreate() {}

    public void reserveRoom(Customer
                    cust, RoomInfo ri,
                Date from, Date to) {
        CreditCard card =
                 cust.getCreditCard();

        RoomHome roomHome = (RoomHome)
        getHome("java:comp/env/ejb/RoomEJB",
                           RoomHome.class);
        Room room =
          roomHome.findByPrimaryKey(
                           ri.getID());
        double amount =
               room.getPrice(from,to);

        CreditServiceHome creditHome =
           (CreditServiceHome) getHome(
        "java:comp/env/ejb/CreditServiceEJB",
                    CreditServiceHome.class);
        CreditService creditAgent =
                         creditHome.create();
        creditAgent.verify(card, amount);

        ReservationHome resHome =
                         (ReservationHome) getHome(
                               "java:comp/env/ejb/ReservationEJB",
                               ReservationHome.class);
        Reservation reservation = resHome.create(cust.getName(),
                                                 room,from,to);
    }

    public RoomInfo[] availableRooms(Location loc,
                               Date from, Date to) {
        // do a SQL call to find available rooms
        Connection con = db.getConnection();
        Statement stmt = con.createStatement();
        ResultSet results =
                stmt.executeQuery("SELECT ...");
        ...
        return roomInfoArray;
    }

    private Object
      getHome(String path, Class type) {
        Object ref =
                jndiContext.lookup(path);
        return
          PortableRemoteObject.narrow(ref,type);
    }
}
```

Die `HotelClerkBean` ist eine zustandslose Bohne. Alle Informationen, um eine Reservation zu tätigen oder eine Abfrage durchzuführen (Liste der verfügbaren Zimmer) werden denMethoden als Argumente übergeben. Im Falle der `reserveRoom()` Methode werden Methoden anderer Beans (`Room, CreditService`, and `Reservation`) eingesetzt, um komplexere Aufgaben erledigen zu können. Dies ist ein Beispiel für den Einsatz einer Session Bean für die Interaktion mit mehreren Beans im Auftrag eines Clients. Die Methode `availableRooms()` wird eingesetzt, um Datenbanken abzufragen und eine Liste der verfügbaren Zimmer zu erhalten - die Information wird als Collection zurückgeliefert, als gewrappte, gemantelte Information der RoomInfo Klasse.

Das folgende Programm zeigt, wie dieses Design Pattern eingesetzt werden kann.

```
public class RoomInfo {
    public int     id;
    public int     bedCount;
    public boolean smoking;
}
To obtain a reference to another bean, as is done three times in the
reserveRoom() method, the private getHome() helper method is used. The
getHome() method uses the JNDI ENC to obtain references to other beans.

private Object getHome(String path,
                       Class type) {
   Object ref =
           jndiContext.lookup(path);
   return
      PortableRemoteObject.narrow(
                          ref,type);
}
```

Ab der EJB 1.1 Spezifikation wird RMI over IIOP als Kommunikationsinfrastruktur eingesetzt (kann eingesetzt werden). Damit wird ein Anschluss an CORBA ermöglicht. Damit die Datentypen CORBA kompatibel werden, muss die Methode PortableRemoteObject.narrow() eingesetzt werden.

1.1.9. Übung

3. Kreieren Sie eine zustandslose Session Bean

1.1.9.1. Stateful Session Beans

Stateful, zustandsbehaftete Session Beans sind einem einzelnen Client zugeordnet und erhaltenen einen sogenannten *conversational-state* zwischen den einzelnen Methodenaufrufen. Dies führt dazu, dass zustandsbehaftete Session Beans nicht von mehreren Clients gleichzeitig benutzt werden können.

Betrachten wir als Beispiel den Hotelangestellten, die `HotelClerk` Bean, die wir nun zu einer zzustandsbehafteten Session Bean machen, indem wir ihr einen 'conversational-state' zuordnen. Eine Beispielanwendung wäre folgender Businessabauf: es werden mehrere Reservationen entgegengenommen und dann gebündelt einer Kreditkarte zugeteilt. Dies ist immer dann realistisch, wenn eine ganze Familie im Hotel absteigt.

```
import javax.ejb.SessionBean;
import javax.naming.InitialContext;
```

```
public class HotelClerkBean
          implements SessionBean {

    InitialContext jndiContext;

    //conversational-state
    Customer cust;
    Vector resVector = new Vector();

    public void ejbCreate(Customer
                      customer) {}
        cust = customer;
    }

    public void addReservation(Name
                    name, RoomInfo ri,
                                Date
                      from, Date to) {
        ReservationInfo resInfo =
              new ReservationInfo(name,
                           ri,from,to);
        resVector.addElement(resInfo);
    }

    public void reserveRooms() {
        CreditCard card =
            cust.getCreditCard();
        Enumeration resEnum =
             resVector.elements();

        while
         (resEnum.hasMoreElements()) {

        ReservationInfo resInfo =
                     (ReservationInfo)
                  resEnum.nextElement();

        RoomHome roomHome = (RoomHome)
        getHome("java:comp/env/ejb/RoomEJB",
                           RoomHome.class);
        Room room =
             roomHome.findByPrimaryKey(
                 resInfo.roomInfo.getID());
        double amount = room.getPrice(
                 resInfo.from,restInfo.to);

        CreditServiceHome creditHome =
                       (CreditServiceHome)
        getHome("java:comp/env/ejb/CreditServiceEJB",
                CreditServiceHome.class);
    CreditService creditAgent = creditHome.create();
    creditAgent.verify(card, amount);

        ReservationHome resHome = (ReservationHome)
        getHome("java:comp/env/ejb/ReservationEJB",
                ReservationHome.class);
        Reservation reservation =
        resHome.create(resInfo.getName(),
        resInfo.roomInfo,resInfo.from,resInfo.to);
    }
```

```
    public RoomInfo[] availableRooms(Location loc,
                                Date from, Date to) {
        // Make an SQL call to find available rooms
    }

   private Object getHome(String path, Class type) {
        Object ref = jndiContext.lookup(path);
        return PortableRemoteObject.narrow(ref,type);
   }
}
```

Die zustandsbezogene Version der HotelClerkBean Klasse speichert den 'conversational' Zustand der Kundenreferenz als `Vector` von `ReservationInfo` Objekte. Die Bean merkt sich die Reservationen verarbeitet sie in einem Batch in der serverRooms() Methode.

Um Resourcen zu optimieren setzt man im Falle der zustandsbehafteten Session Beans die ejbActivate() und ejbPassivate() Methoden ein. Der Container ruft diese Methoden auf und informiert die Bohne darüber (falls sie passiv werden soll : 'passivated' ejbPassivate()), oder nach einer Aktivierung ejbActivate()).

Die ejbRemove() Methode wird dann aufgerufen, wenn die Methode remove() des home oder remote Interfaces durch den Client aufgerufen wird.

1.1.10. Einsatz / Deploying von Enterprise JavaBeans Technologie Lösungen

Welche Vorteile haben EJBs?
Auf Grund des obigen sollte Ihnen klar sein, dass EJBs mit Hilfe des Containers relativ einfach realisiert werden können. Der Container ist für die Persistenz, Transaktionen, Concurrency und Zugriffskontrollen zuständig. Die EJB Spezifikation beschreibt einen *deklarativen* Mechanismus wie diese Problembereiche mit Hilfe von XML 'deployment descriptor' / Einsatzbeschreibungen erledigt werden kann.

Wann immer eine Bohne in einen Container gelegt wird, liest der Container diese Einsatzbeschreibung und findet so heraus, wie die Themen "Transaktionen, Persistenz (im Falle der Entitätenbohnen), Zugriffskontrollen" gehandhabt werden sollen.

Personen, die eine Bean einsetzen möchten, brauchen nur diese Informationen zu lesen, und schon können sie die unktionalität der Bohne nutzen.

Die Einsatzbeschreibung ('deployment descriptor') besitzt ein fixes Format für alle EJB kompatiblen Beans: alle EJB Server müssen wissen,wie diese gelesen werden kann. Dieses Format wird mit einer XML Document Type Definition, oder DTD beschrieben. Die Einsatzbeschreibung beschreibt den Beantypus (session oder entity) und die Klasseen, die vom remote, home und der Bean Klasse eingesetzt werden. Auch die transaktiononsbezogenen Attribute jeder Methode der Bean, welche Zugriffsrechte benötigt werden, um eine Methode aufzurufen (access control) und ob die Persistenz in den Entity Beans automatisch oder durch die Bean gehandhabt wird.

Schauen Sie sich die XML Einsatzbeschreibung (deployment descriptor) an, mit der die Customer Bean beschrieben wird:

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC "-
```

```
//Sun Microsystems, Inc.
//DTD Enterprise
JavaBeans 1.1//EN"
 "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>
        Diese Bohne repräsentiert einen Kunden
      </description>
      <ejb-name>CustomerBean</ejb-name>
      <home>CustomerHome</home>
      <remote>Customer</remote>
      <ejb-class>CustomerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>Integer</prim-key-class>
      <reentrant>False</reentrant>

      <cmp-field><field-name>myAddress</field-name></cmp-field>
      <cmp-field><field-name>myName</field-name></cmp-field>
      <cmp-field><field-name>myCreditCard</field-name></cmp-field>
    </entity>
  </enterprise-beans>

  <assembly-descriptor>
    <security-role>
      <description>
        Diese Rolle beschreibt jene, die vollen Zugriff auf die
        Customer Bean haben.
      </description>
      <role-name>everyone</role-name>
    </security-role>

    <method-permission>
      <role-name>everyone</role-name>
      <method>
        <ejb-name>CustomerBean</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>

    <container-transaction>
      <description>
        Alle Methoden benötigen Transaktionen
      </description>
      <method>
        <ejb-name>CustomerBean</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

EJB-capable application servers usually provide tools which can be used to build the deployment descriptors; this greatly simplifies the process.
When a bean is to be deployed, its remote, home, and bean class files and the XML deployment descriptor must be packaged into a JAR file. The deployment descriptor must be stored in the JAR under the special name META-INF/ejb-jar.xml. This JAR file, called an ejb-jar, is vendor neutral; it can be deployed in any EJB container that supports the complete EJB

specification. When a bean is deployed in an EJB container its XML deployment descriptor is read from the JAR to determine how to manage the bean at runtime. The person deploying the bean will map attributes of the deployment descriptor to the container's environment. This will include mapping access security to the environment's security system, adding the bean to the EJB container's naming system, etc. Once the bean developer has finished deploying the bean it will become available for client applications and other beans to use.

1.1.11. Übung

4. Aufsetzen der Daenbank

5. Einsatz / Deploying Enterprise Beans in Sun's J2EE Reference Implementation

1.1.12. Enterprise JavaBeans Clients

Enterprise JavaBeans Clients können standalone Applikationen, Servlets, Applets oder andere Enterprise Beans sein. Unsere `HotelClerk` Session Bean (siehe oben) ist ein Client der `Room` Entity Beans. Alle Clients benutzen das Server Bean home Interface, um Referenzen auf die Server Bean zu erhalten. Diese Reference kennt / besitzt die Signatur des remote Interfaces der Server Bean. Damit kann der Client mit dem Server Bean kommunizieren, einzig und allein mit Hilfe der Methoden, die im remote Interface definiert wurden.

Das remote Interface defininiert die Business Methoden, wie beispielsweise Zugriffsmethoden und Mutationsmethoden, um beispielsweise einen Kundennamen zu ändern, oder Business Methoden einzusetzen, wie beispielsweise der `HotelClerk` Bean, um Zimmer in einem Hotel zu reservieren.

Im Folgenden sehen sie ein Beispiel, wie ein Client auf die Methoden der `Customer` Bean zugreifen können. In diesem Fall ist das home Interface CustomerHome und das remote Interface ist Customer.

```
CustomerHome home;
Object      ref;

// Reference auf CustomerHome
ref  = jndiContext.lookup("java:comp/env/ejb/Customer");

// Casten (JNDI lookup)
home = PortableRemoteObject.narrow(ref, CustomerHome.class);

// Einsatz vom home interface um eine
// neue Instance der Customer bean zu kreieren.
Customer customer = home.create(customerID);

// Business Methode von Customer.
customer.setName(someName);
```

Ein Client liest zuerst eine Referenz auf das home Interface mit Hilfe des JNDI ENC. Damit kennt der Client die Server Beans. Da ab EJB 1.1 Java RMI-IIOP das bevorzugte Programmiermodell (Kommunikationskonzept) ist, .müssen wir alle CORBA Referenztypen unterstützen (casting) . Die PortableRemoteObject.narrow() Methode muss dafür eingesetzt werden. Da JNDI immer ein Object liefert, müssen alle Referenzen dem Container entsprechend gecastet werden (siehe Programm).

Nach Erhalt des home Interface kann der Client die Methoden des home Interfaces eingesetzt werden, um Server Beans zu kreieren, finden oder zu entfernen. Die Methode create() liefert eine remote Referenz auf die Server Bean, mit welcher der Client seine Aufgaben erfüllen möchte.

1.1.13. Übung

6. Kreieren eines EJB Clients

1.1.14. Ressourcen

Auf dem Internet finden Sie jede Menge Ressourcen zum Thema Enterprise JavaBeans Technologie. Das Buch von O'Reilly gilt als besonders empfehlenswert..

1.1.14.1. Web Sites

Die Liste ändert sich täglich. Aber stabil dürften folgende URLs sein:

- Sun Microsystems, Inc. EJB Vendor List
- Object Management Group, Inc.
- EJB-INTEREST Mailing List Archive
- EJBNow - a resource for EJB technology developers
- jGuru's EJB technology FAQ

1.1.14.2. Dokumentation und Spezifikationen

Auf dem Java Technology Site von Sun Microsystems finden Sie eine Products and APIs Seite, auf der Sie unter anderem folgende Infos finden:

- Sun Microsystems, Inc. EJB Home page
- Sun Microsystems, Inc. J2EE Home page
- Writing Enterprise Applications for J2EE
- Java Naming and Directory Interface
- Java Servlet API
- Sun Microsystems, Inc. JDBC Home page
- Java Transactions
  - Java Transaction API
  - Java Transaction Service
- RMI over IIOP compiler

1.1.14.3. Bücher

Neuere Bücher über Enterprise JavaBeans und 'distributed computing technology' Themen:

- **Monson-Haefel, R. *Enterprise JavaBeans*, second edition. Sebastopol, CA: O'Reilly & Associates, 2000, ISBN: 1-56592-869-5.**
- Roman, E. *Mastering Enterprise JavaBeans and the Java 2 Platform.* New York: John Wiley & Sons, 1999, ISBN: 0-471-33229-1.
- Farley, J. Java Distributed Computing. Sebastopol, CA: O'Reilly & Associates, 1998, ISBN: 1-56592-206-9.
- David Flanagan, et al. *Java Enterprise in a Nutshell : A Desktop Quick Reference*. New York: O'Reilly & Associates; ISBN: 1-56592-483-5.
- **Orfali, R., Harkey, D., and Edwards, J. *The Essential Client/Server Survival Guide, 2nd Ed.* New York: John Wiley & Sons, 1996, ISBN: 0-471-15325-7.**

Mit der Installation des Dokumentationssets für J2EE werden jede Menge Beispiel Enterprise Beans mitgeliefert (im Verzeichnis : ...\guides\ejb\examples\...).

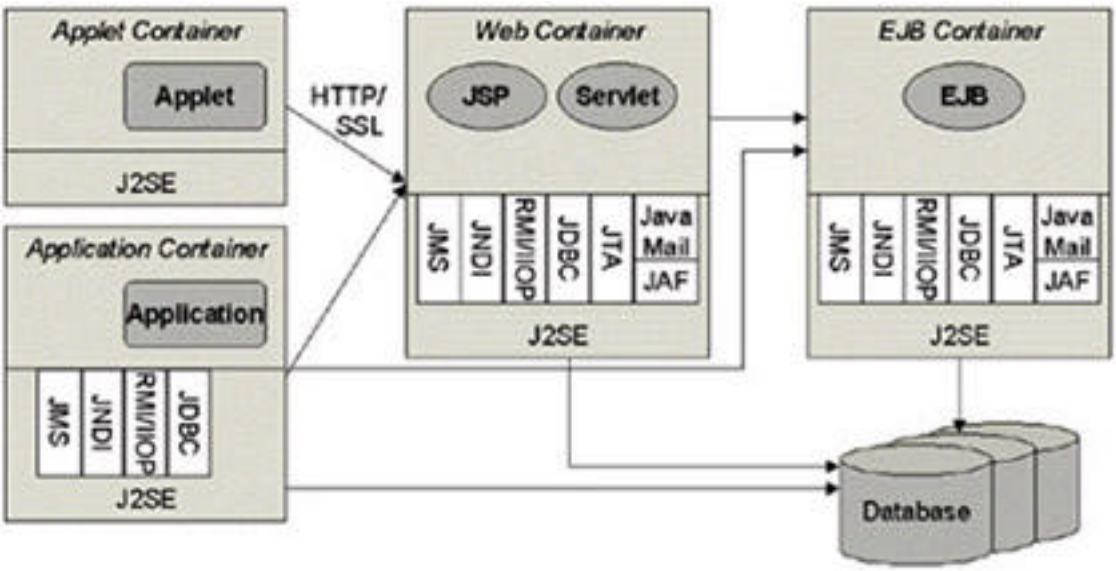


Abbildung 0-1 Zusammenspiel mehrerer EJBs

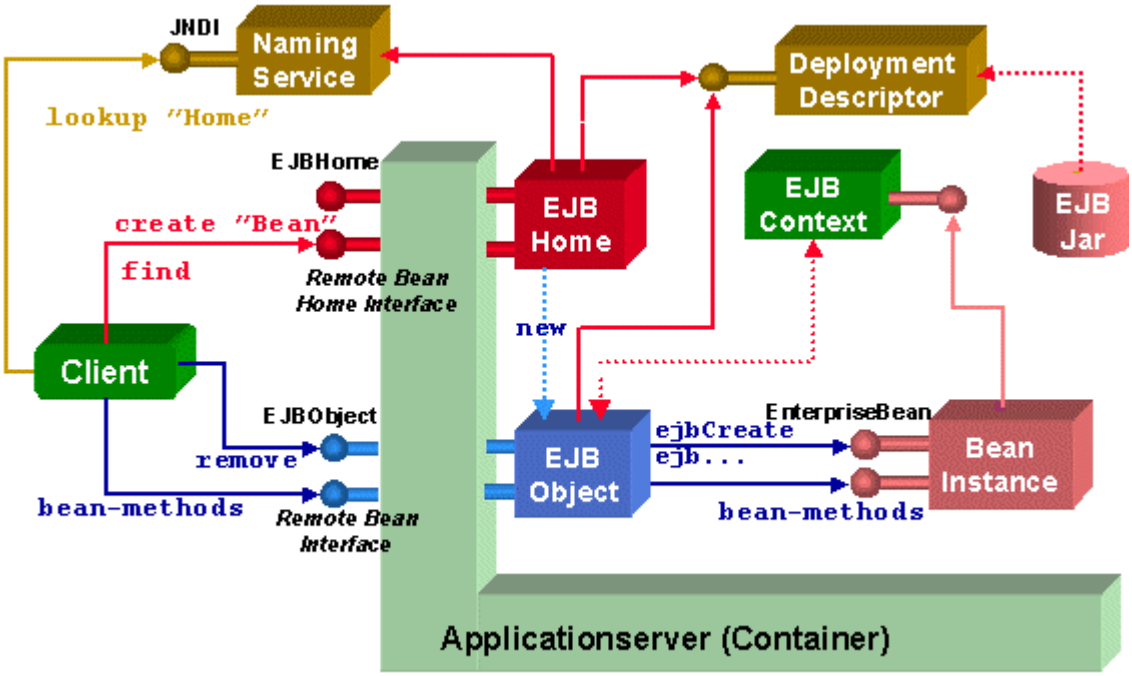


Abbildung 0-2 EJBs im Container