



# Java 2 Enterprise Edition

Franz Lackinger  
Java Enterprise Architect  
Sun Microsystems GesmbH  
<mailto:franz.lackinger@sun.com>





Java 2  
Enterprise  
Edition

Java 2  
Standard  
Edition

Java 2 Micro Edition

Java Platform

HotSpot

Classic VM

K VM

Card VM

Memory:

10MB  
64 bit

1MB

500kB  
32 bit

10kB  
16 bit

8 bit

# Java 2 Enterprise Edition (J2EE)

- Overview
  - About Components & Containers
  - Deployment
  - Architectures
- In More Detail
  - Presentation Tier
  - Business Tier



# Java 2 Enterprise Edition (J2EE)

- Overview
  - **About Components & Containers**
  - Deployment
  - Architectures
- In More Detail
  - Presentation Tier
  - Business Tier



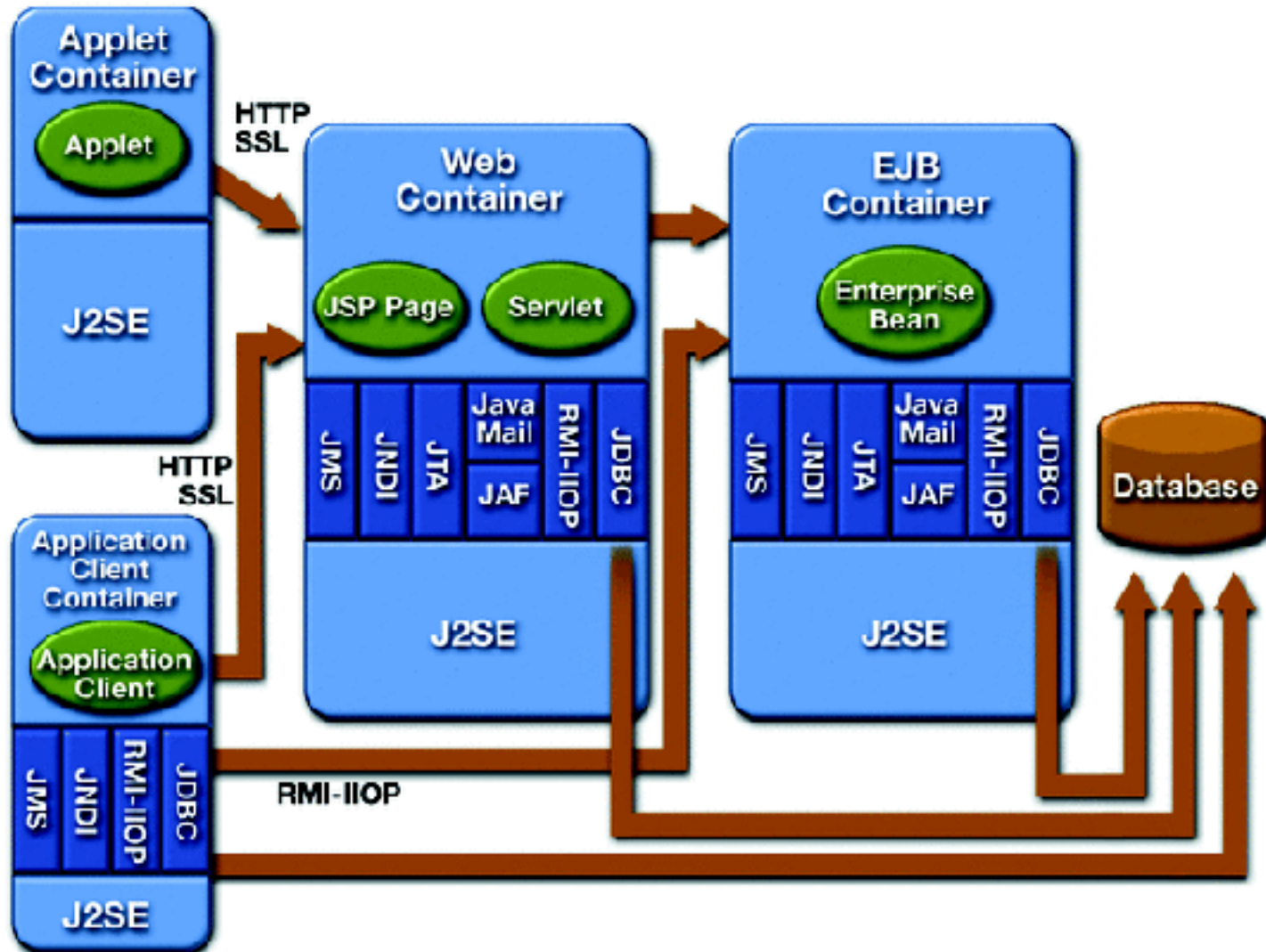
# Components

- Web Components
  - Servlets, JSPs, Filters, Event Listeners
- Enterprise JavaBeans
  - Session (stateful/less), Message Driven, Entity
- Resource Manager Drivers ("Connectors")
- Application Clients
- Applets

# Containers

- Web Container
- EJB Container
- Application Client Container
- Applet Container

# The J2EE Platform



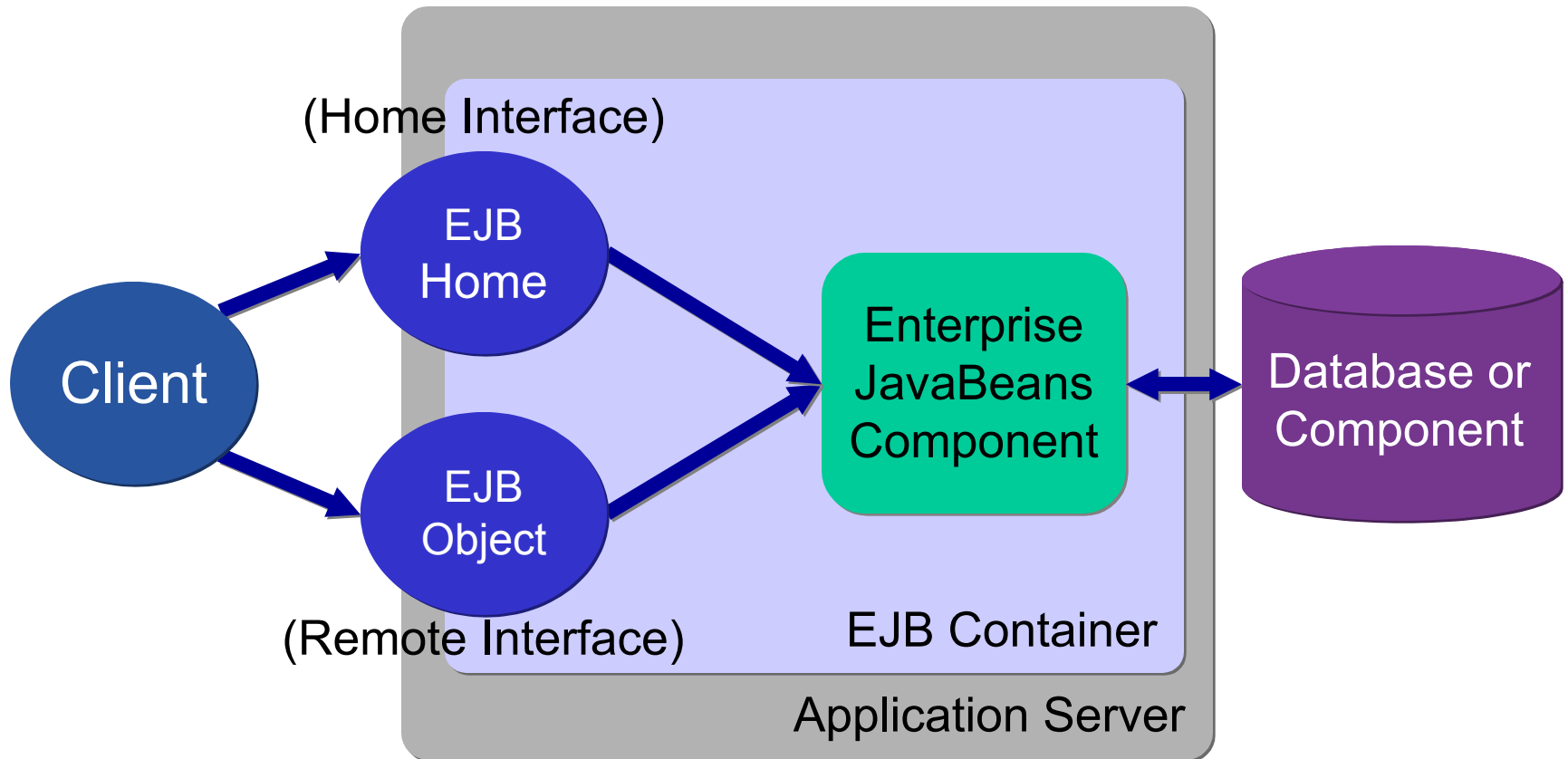
# Container Services

- Life cycle management for components
- Distribution (Corba IIOP)
- Resource management
  - Multi-threading, state management, resource pooling
- Transaction Handling
  - Container managed (EJBs) vs. component managed
- Security management
  - Authentication, authorization, secure communication
  - Declarative vs. programmatic

# Web Components

- Servlets
  - Sophisticated web server extension
  - Consume HTTP request, produce HTTP response
- JSPs
  - Compiled to servlets
  - Inside-out servlets
  - For producing text-based content (HTML, XML, ...)
  - Taglibs instead of code (?)

# EJB Model



# EJB Ingredients

- Home interface
  - Create, remove, find bean instances (factory)
  - Local home interface and/or remote home interface
- Component interface
  - Business methods
  - Local interface and/or remote interface
- Bean class
- Deployment descriptor(s)

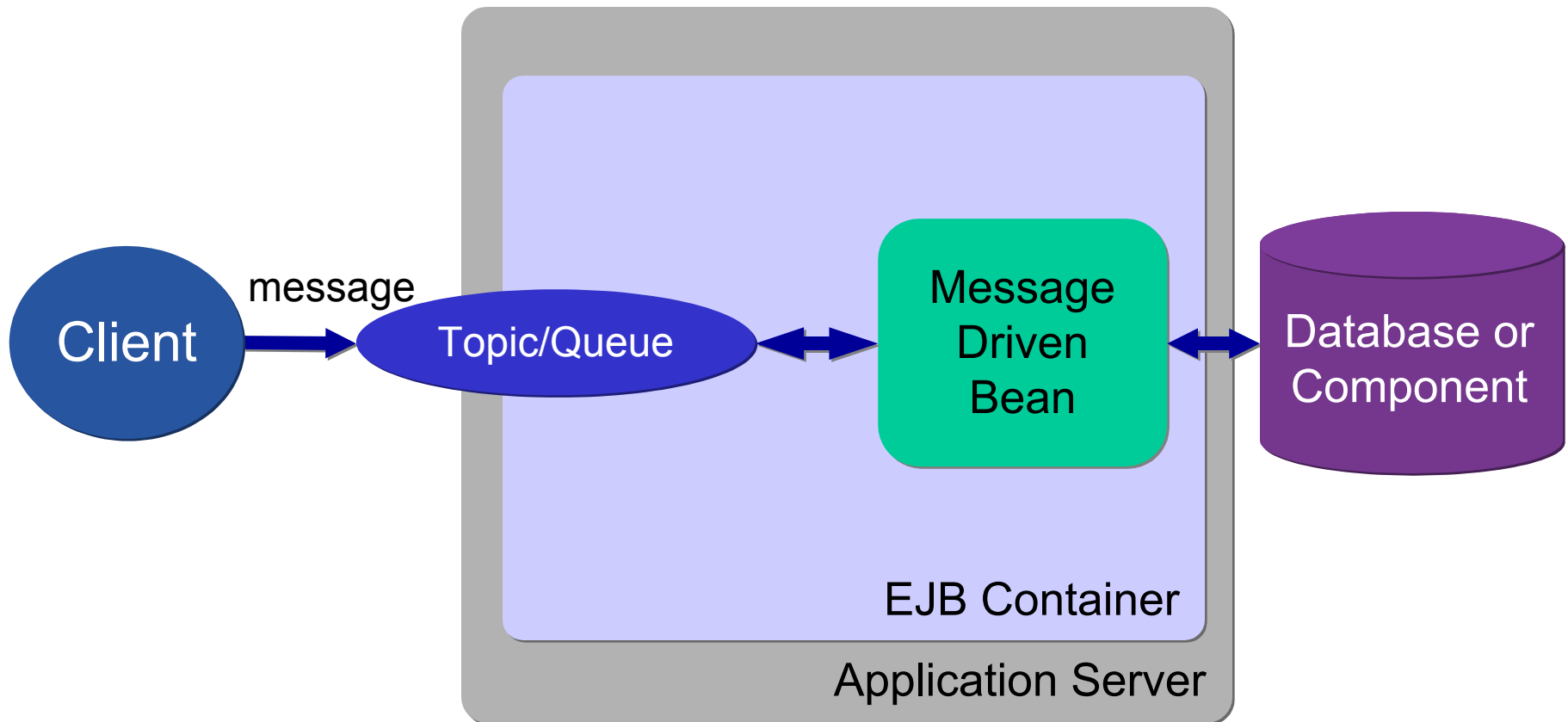
# Session Beans

- Represent non-persistent objects, workflow, logic
- Stateless session beans
  - Scoped collection of methods
  - No state maintained from one method call to next (RPCs)
- Stateful session beans
  - To be used as server-side extension of client
  - Maintain client-related (conversational) state
  - Consume server-side resources for each client

# Entity Beans

- Represent persistent domain objects
- Backed by persistent store (database)
  - Persistent state may change "behind" entity bean!
- Persistence management
  - Container managed (CMP)
    - Usually better performance
  - Bean managed (BMP)
    - Code database access in callbacks (ejbLoad(), ejbStore(), ...)

# Message Driven Beans



# Message Driven Beans

- No home or local interface
- No direct call from client
  - Instead client sends message to JMS destination
- Asynchronous message listeners on JMS destination
- Stateless

# Java 2 Enterprise Edition (J2EE)

- Overview
  - About Components & Containers
  - **Deployment**
  - Architectures
- In More Detail
  - Presentation Tier
  - Business Tier



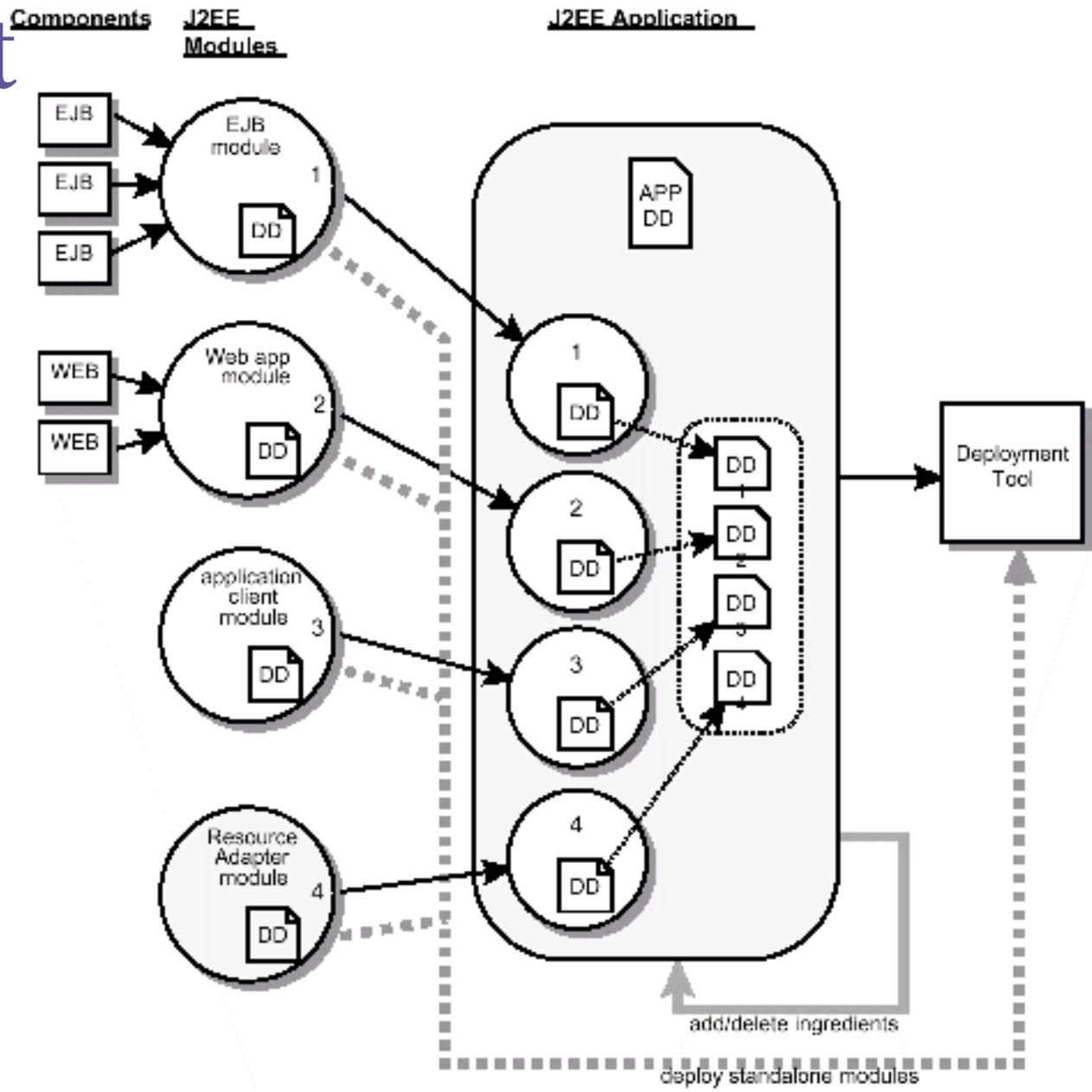
# Component Packaging

- Web Application Archive (WAR)
  - WEB-INF/web.xml
- EJB-JAR
  - META-INF/ejb-jar.xml
- Resource Archive (RAR)
  - META-INF/ra.xml
- Application Client JAR
  - META-INF/application-client.xml
- Enterprise Application Archive (EAR)
  - META-INF/application.xml

# Deployment Descriptors

- Can be edited before/at deployment
- Describe the component and its requirements
  - Name, description, icon, classes/interfaces, JMS destination
  - Requirements for container services (TX, security, pooling, persistence mgmt., ...)
  - External dependencies of component (= Environment)
    - **Environment entry**(ies) (typed name/value pairs)
    - **EJB references**
    - **Resource** manager connection factory **references**
      - JDBC DataSource, JMS ConnectionFactory, URL, mail session
    - **Resource environment references** (JMS destination)
    - Provided to component via JNDI (java:comp/env)

# Deployment

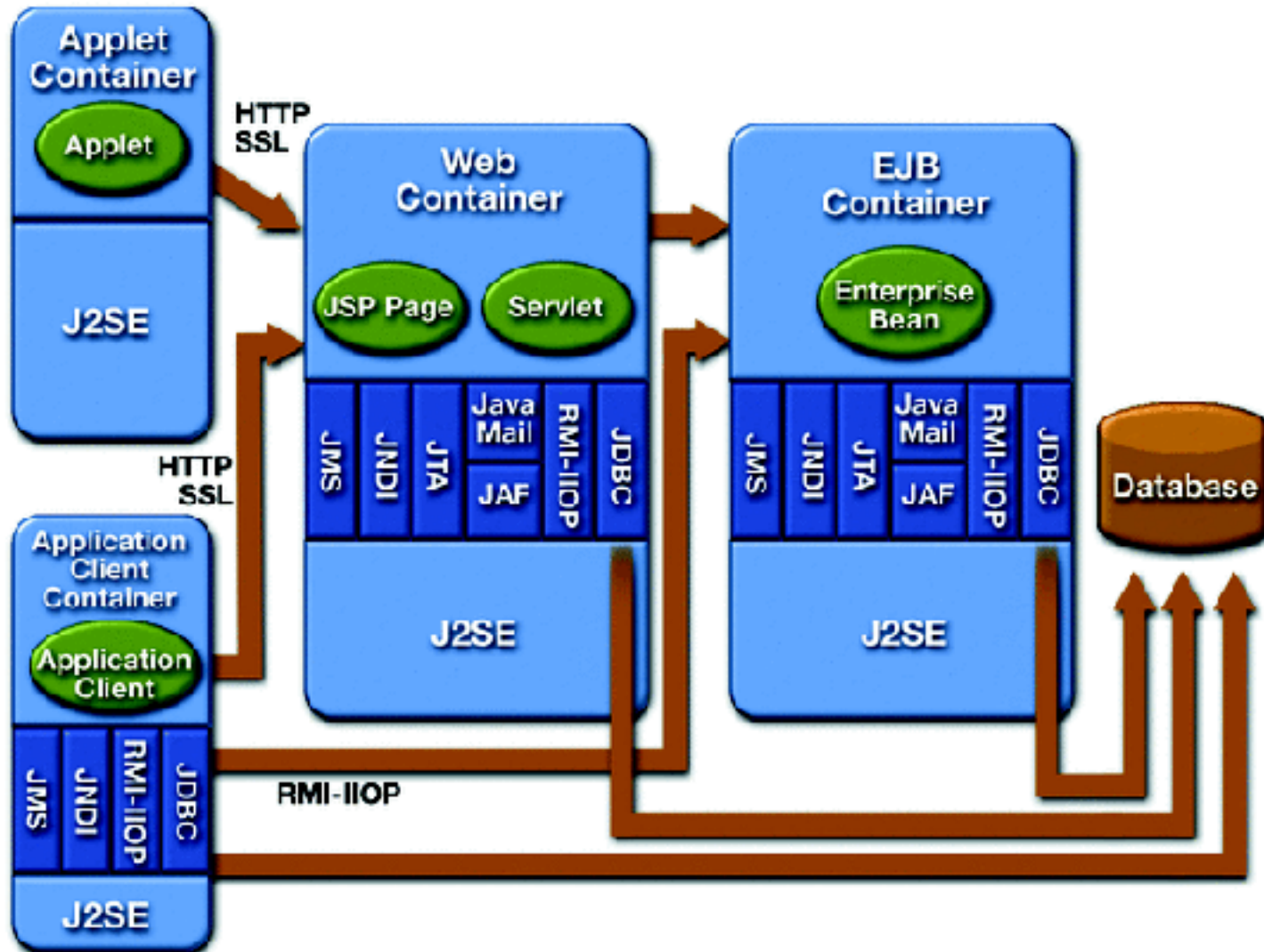


# Java 2 Enterprise Edition (J2EE)

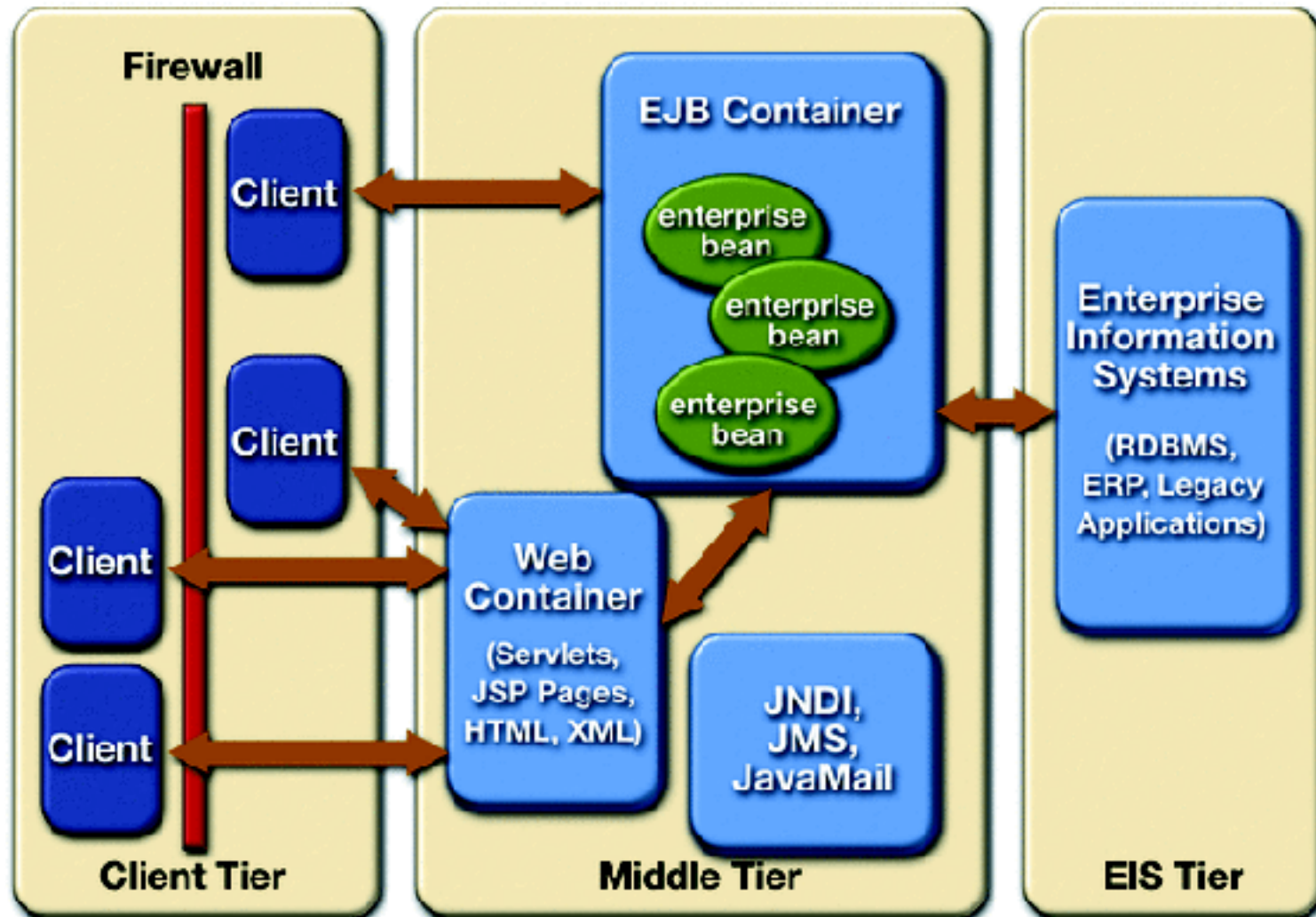
- Overview
  - About Components & Containers
  - Deployment
  - **Architectures**
- In More Detail
  - Presentation Tier
  - Business Tier



# The J2EE Platform

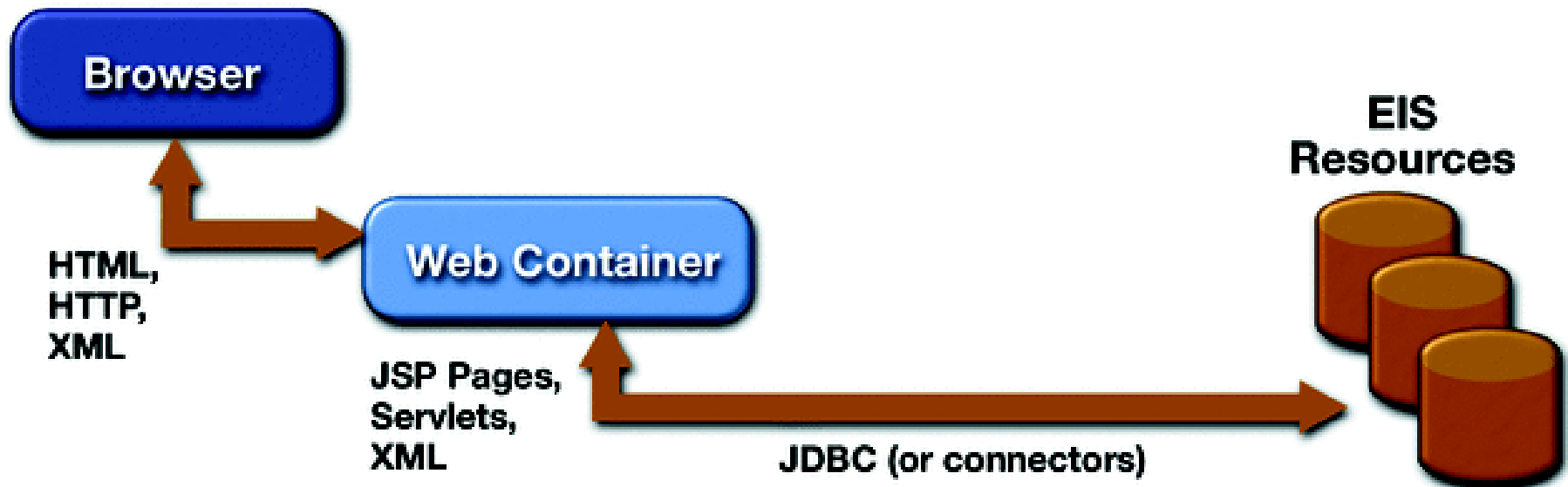


# Typical Application Szenario



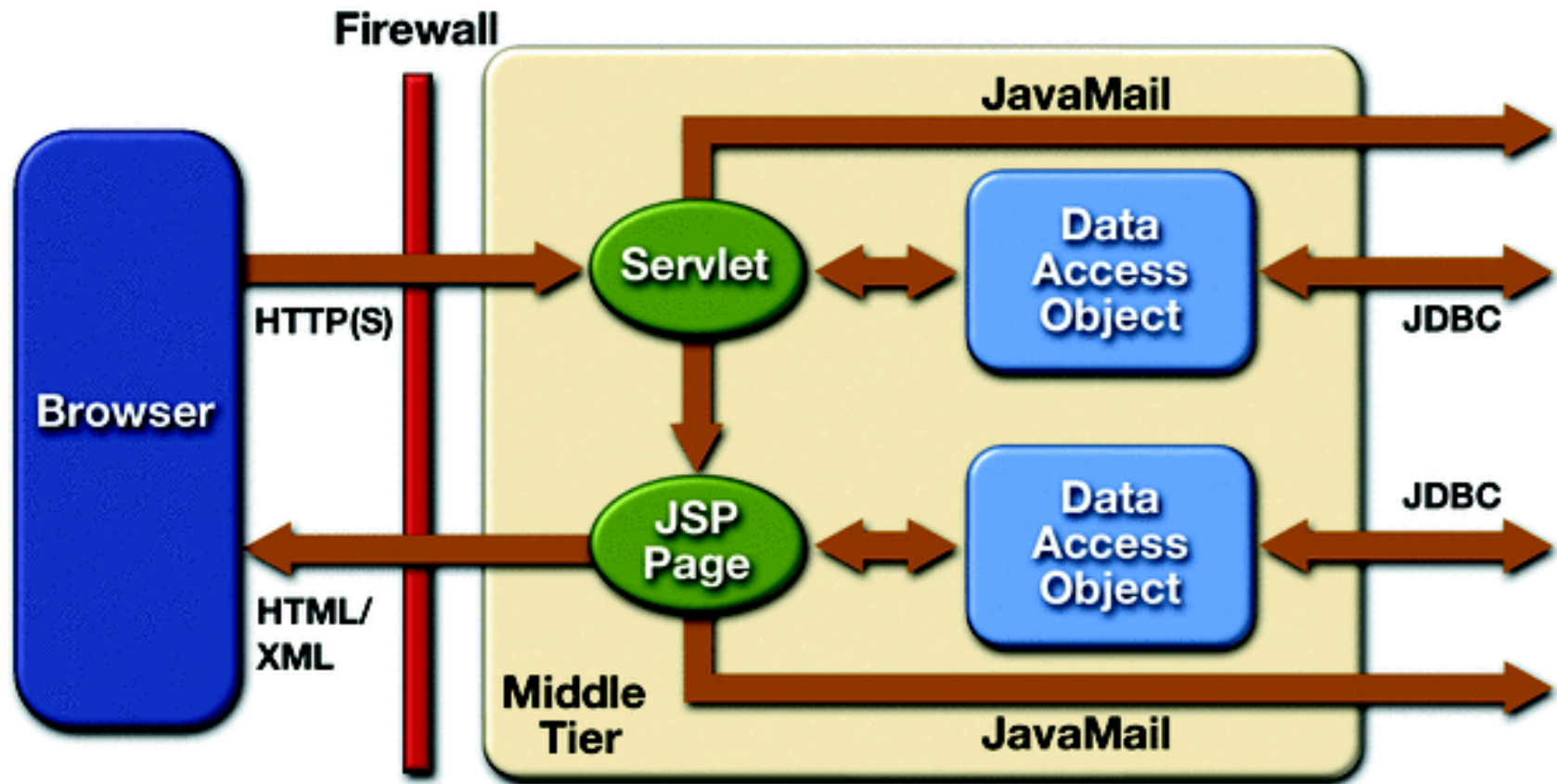
# Restricting Yourself 1/4

## Simple Web-Centric Application



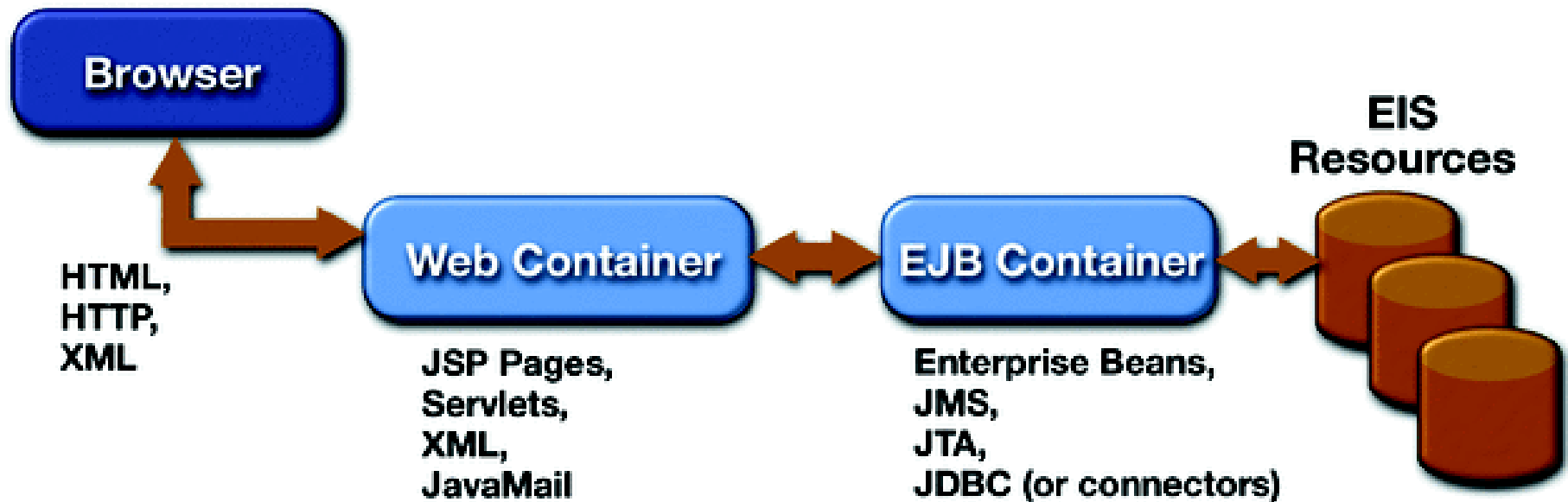
# Restricting Yourself 2/4

## Layered Web Application



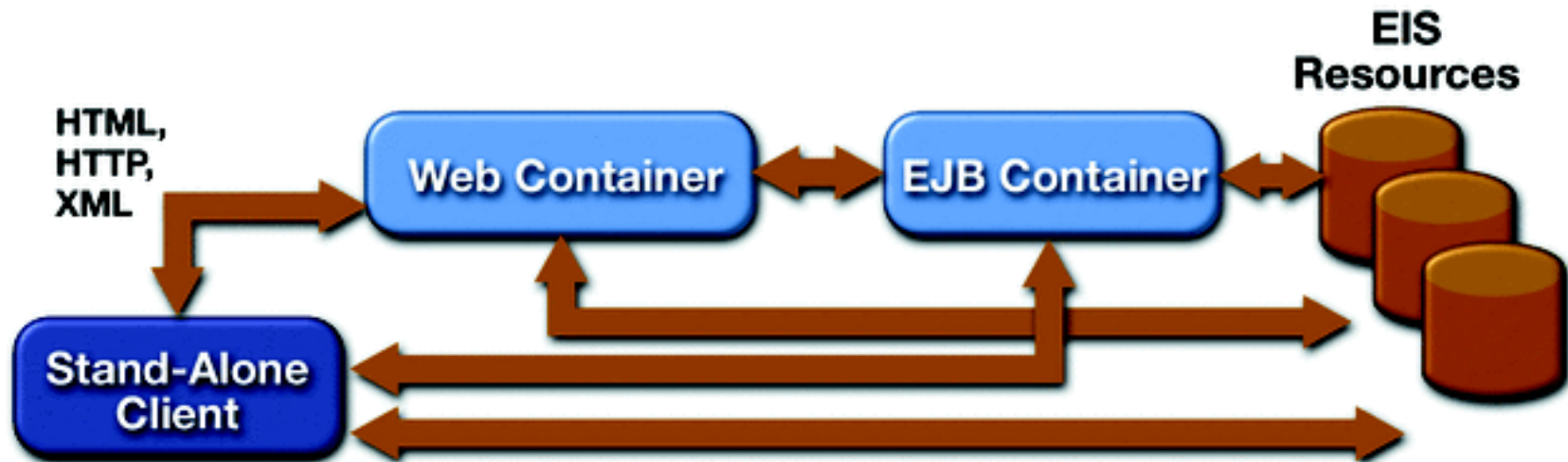
# Restricting Yourself 3/4

## Web Application Using EJBs

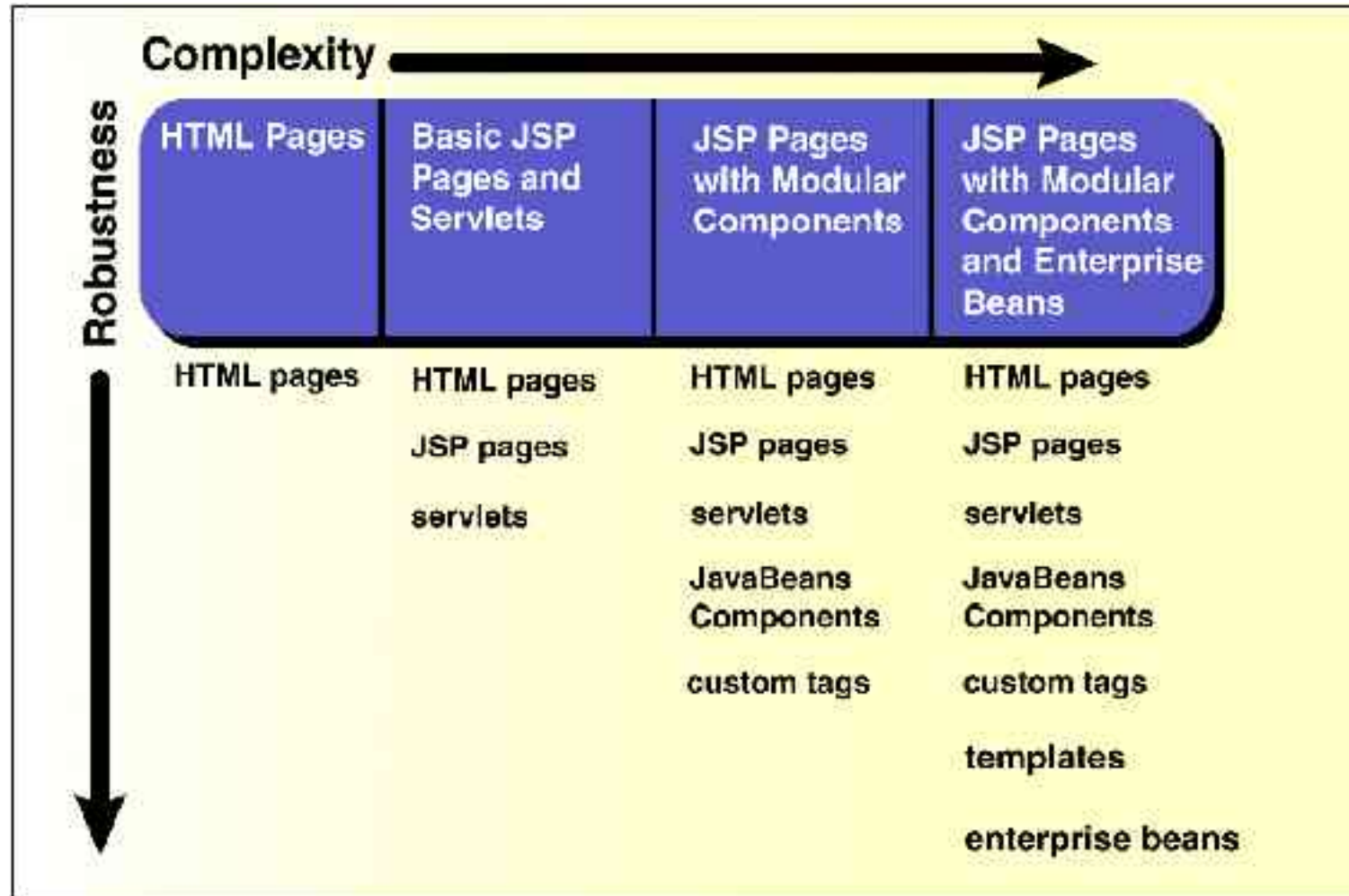


# Restricting Yourself 4/4

Rich Client Distributed Application



# Web Applications



# Java 2 Enterprise Edition (J2EE)

- Overview
  - About Components & Containers
  - Deployment
  - Architectures
- In More Detail
  - **Presentation Tier**
  - Business Tier



# Präsentationsschicht

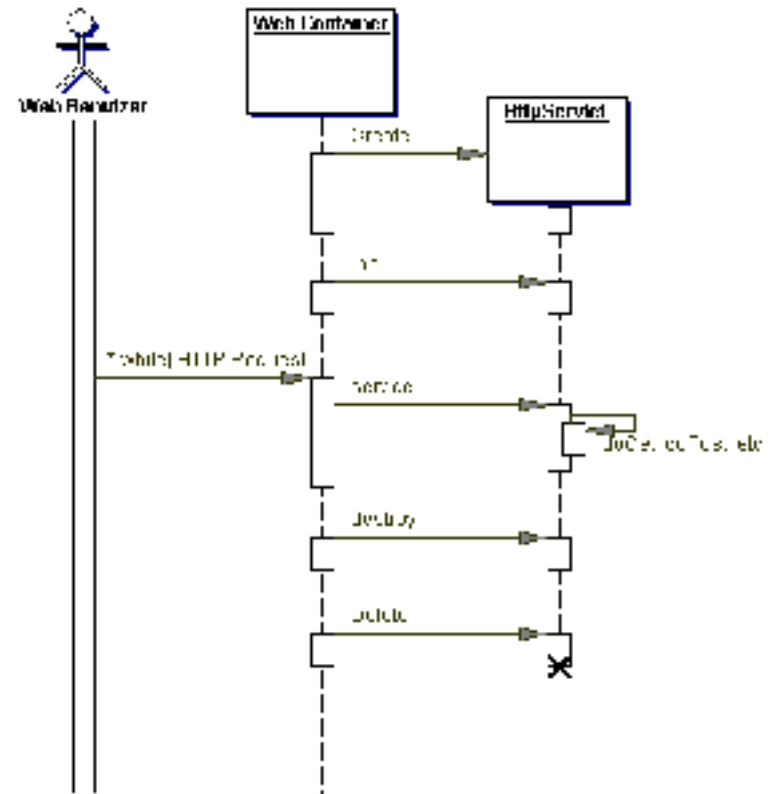
- Servlets
  - Java-Klassen, die das HTTP-Protokoll erfüllen
- Java Server Pages (JSP)
  - HTML-Seiten mit:
    - “Actions” und “Directives”
    - Custom Tags
    - Scriptlet Code
- Web-Applikationen

# Servlet API

- Servlet-Klasse implementiert die Interface `HttpServlet`
- Weitere Interfaces:
  - `HttpServletRequest`
  - `HttpServletResponse`
  - `HttpSession`
- Container startet Threads, legt Instanzen an, und steuert den Lebenszyklus

# Lebenszyklus eines Servlets

- Anlegen als “lazy loading” oder beim Serverstart:
  - `<load-on-startup/>`  
in `web.xml`
- In der Servlet-Klasse:
  - `init()`
  - `doGet()`
  - `doPost()`
  - `destroy()`
- Statt Löschen könnte die Servlet-Instanz zum Pool zurückgegeben werden



# Servlet Beispiel

```
public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        // Must set the content type first
        res.setContentType("text/html");
        // Now we can obtain a PrintWriter
        PrintWriter out = res.getWriter();

        out.println("<html><head><title>Hello World!</title></head>");
        out.println("<body><h1>Hello World!</h1></body></html>");
        // Do not close the output stream or print writer.
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        doGet(req, res);
    }
}
```

# Servlets: Session Handling

```
HttpSession session = request.getSession(true);
```

- Beim Parameter `true` wird eine neue Session angelegt, falls keine existiert
- Ein Cookie `JSESSIONID` enthält dann eine Session-ID
- Alternative: URL-Rewriting

```
http://www.aol.de/myervlet;jsessionid=xyz123?foo=bar
```

```
String newurl = response.encodeURL(url)
```

- Timeout wird in `web.xml` festgelegt

# Servlets: Request Dispatching

```
String path = "/newpath/anotherservlet";  
RequestDispatcher dispatcher =  
    request.getRequestDispatcher(path);  
dispatcher.forward(request, response);
```

... oder ...

```
dispatcher.include(request, response);
```

- Bei `forward()` darf die Response nicht schon committed sein
- Request- & Response-Objekte werden weitergeleitet

# Servlets: Attribute

- Folgende Objekte haben eine Attributentabelle (Hash-Tabelle)
  - Request
  - Response
  - Session
  - ServletContext (global in der Web-Applikation)

```
request.setAttribute("mykey", value);
```

```
String theValue = session.getAttribute(theKey);
```

- Teilt Information unter verschiedenen Skopi

# Java Server Pages

- Java's Antwort auf Active Server Pages

```
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>

<head>
<title>Hello World</title>
</head>

<body bgcolor=#FFFFFF>
<h2>Hello World</h2>

<%
out.print("<p><b>Hello World!</b>");
%>

<hr>

</font>
</body>
</html>
```

# JSPs: Directives

- Nachrichten an den Container:  
Werden zur Compile-Zeit ausgeführt

```
<%@ directive { attr="value" }* %>
```

- Beispiele

```
<%@ page import="java.util.*, java.lang.*" %>
```

```
<%@ page buffer="5kb" autoFlush="false" %>
```

```
<%@ page errorPage="error.jsp" %>
```

```
<%@ include file="date.jsp" %>
```

```
<%@ taglib uri="http://www.aol.de/tags" prefix="aol" %>
```

# JSPs: Actions

- Werden zur Laufzeit ausgeführt
- Einige Standard-Actions:

```
<jsp:include page="scripts/login.jsp" />
```

```
<jsp:forward page="/servlet/login" />
```

# JSPs und JavaBeans

```
<jsp:useBean id="cart" scope="session"  
            type="de.aol.shop.CartBean" />
```

```
<jsp:getProperty name="cart" property="username" />
```

```
<jsp:setProperty name="mybean" property="username" />
```

```
<jsp:setProperty name="mybean" property="*" />
```

- Beans werden falls nötig bei `useBean` angelegt
- Mögliche Skopi: `page` | `request` | `session` | `application`
- `setProperty` mit `property="*"` setzt alle Properties auf die Werte der entsprechend benannten Request-Parameter
- Vorsicht: Setzt jede Property auf `null`, für die der entsprechende Parameter fehlt!

# JSPs: Scriptlets

- Code-Fragmente in der Seite
- Aus `banner.jsp` des PetStores:

```
<%  
  if (!customer.isLoggedIn()) {  
%>  
  
    <a href="<%=request.getContextPath()%>/control/signin" onmouseover="img_on  
('signin')" onmouseout="img_off('signin')"></a>  
  
%> } else{ %>  
  
    <a href="<%=request.getContextPath()%>/control/signout" onmouseover="img_o  
n('signout')" onmouseout="img_off('signout')"></a>  
      
    <a href="<%=request.getContextPath()%>/control/editaccount" onmouseover="i  
mg_on('myaccount')" onmouseout="img_off('myaccount')"></a>  
  
%> } %>
```

# JSPs: Custom Tags

- Selbst entwickelte Actions
- Beispiele aus dem PetStore:  
`productdetails.jsp`  
`template.jsp`
- TagLibraries werden in eigenen Java-Klassen entwickelt
- TagLibraries: Struts oder JSTL

# JSPs: Äquivalenz zu Servlets

- JSPs werden in Servlet-Klassen übersetzt, oft zuerst in die Source-Datei für ein Servlet
- Implizite Objekte entsprechen Objekten aus der Servlet API
- Beans für `useBean` sind Einträge in den Attributen-Tabellen

# Web-Applikationen

- Portierbare Sammlung von im Web deploybaren Elementen:
  - Servlets
  - JSPs
  - Java-Klassen
  - Statische Dokumente (HTML, Bilder, Multimedia, usw.)
  - Client-seitige Applets, Beans und Klassen
  - Deskriptive Meta-Information, die diese Elemente in Verbindung setzt.

# WAR-Archive

- Beispiel (siehe auch den PetStore):

```
/index.html  
/howto.jsp  
/feedback.jsp  
/images/banner.gif  
/images/jumping.gif  
/WEB-INF/web.xml  
/WEB-INF/lib/jspbean.jar  
/WEB-INF/classes/com/mycorp/servlets/MyServlet.class  
/WEB-INF/classes/com/mycorp/util/MyUtils.class
```

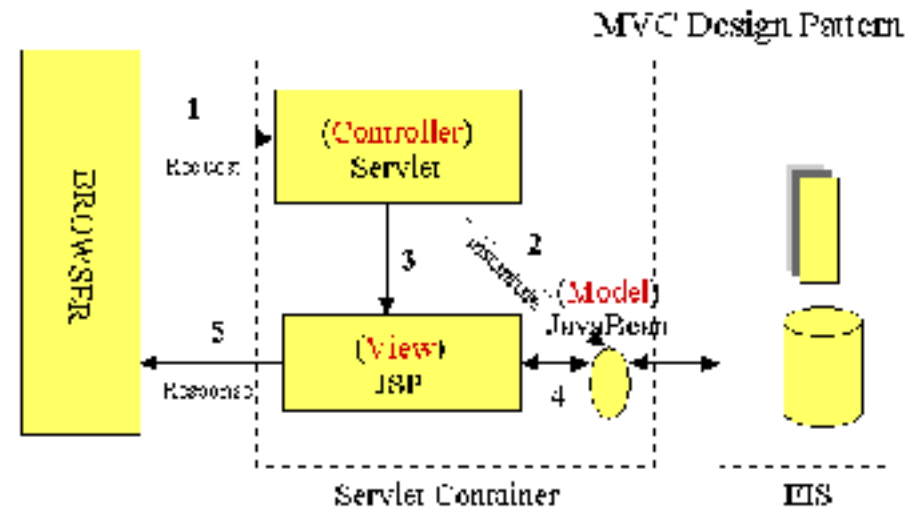
- Deployed unter einem URL-Präfix
- `WEB-INF/classes` **und** `WEB-INF/lib` sind **implizit im CLASSPATH**

# web.xml

- Konfiguriert folgende Parameter:
  - Servlet-Klassen und deren Abbildung auf URLs
  - Session-Timeout
  - JNDI-Namen für EJB-Referenzen
  - Namen von “welcome”-Dateien (index.html)
- Siehe `web.xml` des PetStores

# Model-View-Controller

- Model (z.B. EJB) enthält Datenstrukturen
- View (JSP) enthält nur Layout. Ideal: Kein Scriptlet-Code, sondern nur Actions, Directives und Custom Tags
- Controller Servlet steuert ScreenFlow, und gibt kein HTML-Code aus.
- ScreenFlow am besten extern konfiguriert (vgl. PetStore)



# Best Practices für die Präsentationsschicht

- Model-View-Controller Design Pattern
  - JSPs als Views, mit möglichst wenig Scriptlets
  - Servlets als Controller, mit externer Konfiguration des ScreenFlows
- Lose Kopplung zur Business-Schicht
  - Keine direkte Referenzen auf EJBs, sondern durch vermittelnde Objekte (Business Delegate Design Pattern)

# Präsentationsschicht: Referenzen

- Hunter & Crawford, Java Servlet Programming, 2<sup>nd</sup> Edition
  - <http://www.oreilly.com/catalog/jservlet2/>
- Hans Bergsten, Java Server Pages
  - <http://www.oreilly.com/catalog/jserverpages/>
- Java Blueprints, Kapiteln zur Präsentationsschicht
- Alur et al., Core J2EE Patterns, Kapiteln zur Präsentationsschicht
- Dustin Marx, “JSP Best Practices” (JavaWorld)
  - <http://www.javaworld.com/javaworld/jw-11-2001/jw-1130-jsp.html>

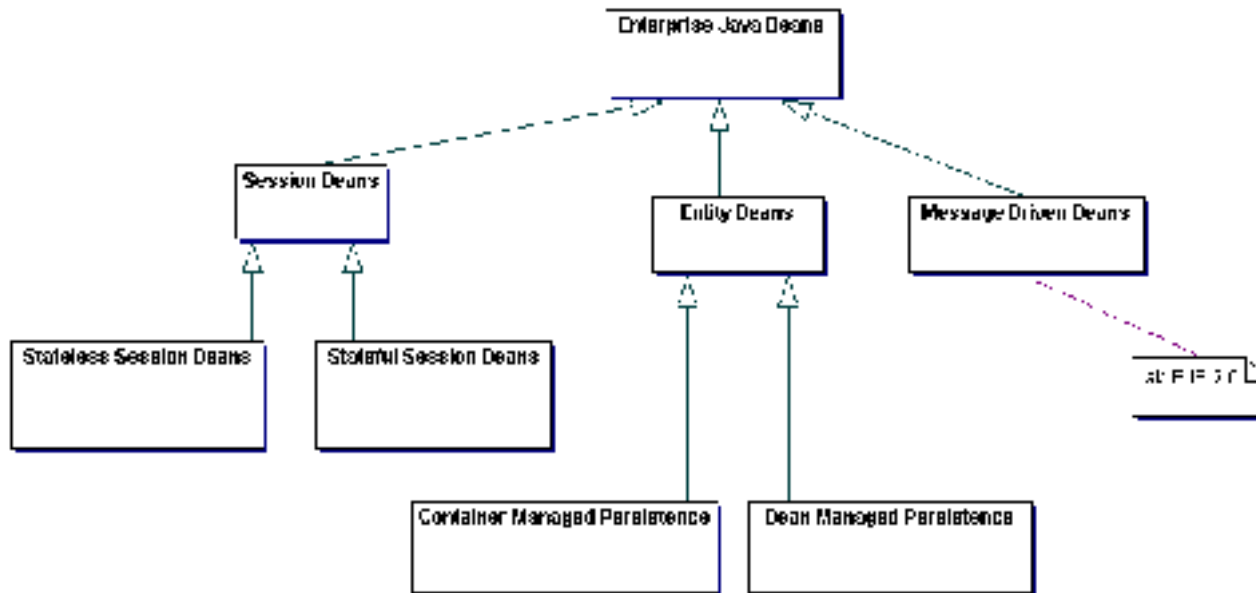
# Java 2 Enterprise Edition (J2EE)

- Overview
  - About Components & Containers
  - Deployment
  - Architectures
- In More Detail
  - Presentation Tier
  - **Business Tier**



# Enterprise Java Beans

## Typen



# EJBs: Gemeinsamkeiten

Was alle EJB-Typen gemeinsam haben

- Lebenszyklus wird vom EJB-Container gesteuert
- Portierbares Deployment: “Lokation-Unabhängigkeit”
  - (aber: Local Interfaces in EJB 2.0)
- Mindestens eine Source-Datei
  - Bean Implementations-Klasse

# EJBs: Source-Dateien

- Bean Implementations-Klasse
  - Eigentliche Implementation der Methods
- Remote-Interface
  - Deklariert Methoden für Remote-Aufrufe
- Home-Interface
  - Buchhaltung: `create()`, `findByXXX()`, `remove()`
- Primary Key Klasse
  - Nur bei Entity Beans
- Verbindung im Deployment-Deskriptor

# Session Beans

- Implementieren Business-Methoden bzw. Workflows
- Greifen auf verschiedene konzeptuelle Objekte (EntityBeans) zu
- Werden nicht persistent gehalten
- `create()`-Methode der Home-Interface gibt eine Instanz zurück, aber legt nicht unbedingt eine neue an

# Stateless Session Beans

- “Conversational State” (z.B. Werte der Instanz-Variablen) wird zwischen Methodenaufrufen nicht unbedingt erhalten
- Methodenaufrufe müssen also “atomar” sein
- Instance-Pooling leicht und effizient
- Im PetStore: Customer, Catalog

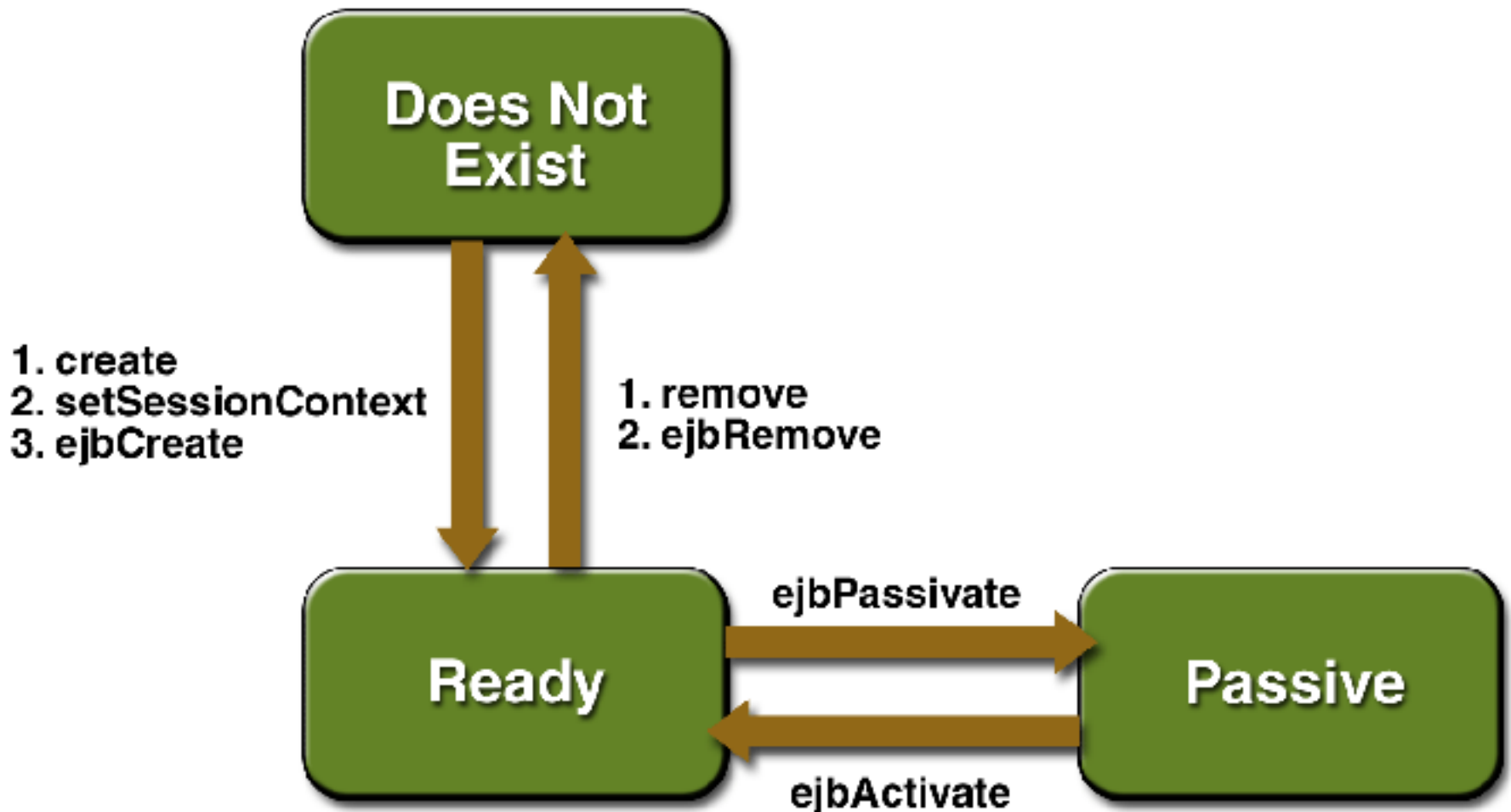
# Stateless Session Beans: Lebenszyklus



# Stateful Session Beans

- “Conversational State” wird solange erhalten, wie ein Client dieselbe Bean-Instanz einsetzt
- Vorsicht: Kein Instance-Pooling, Speicherbedarf kann hoch werden
- Im PetStore: Shopping Client Controller

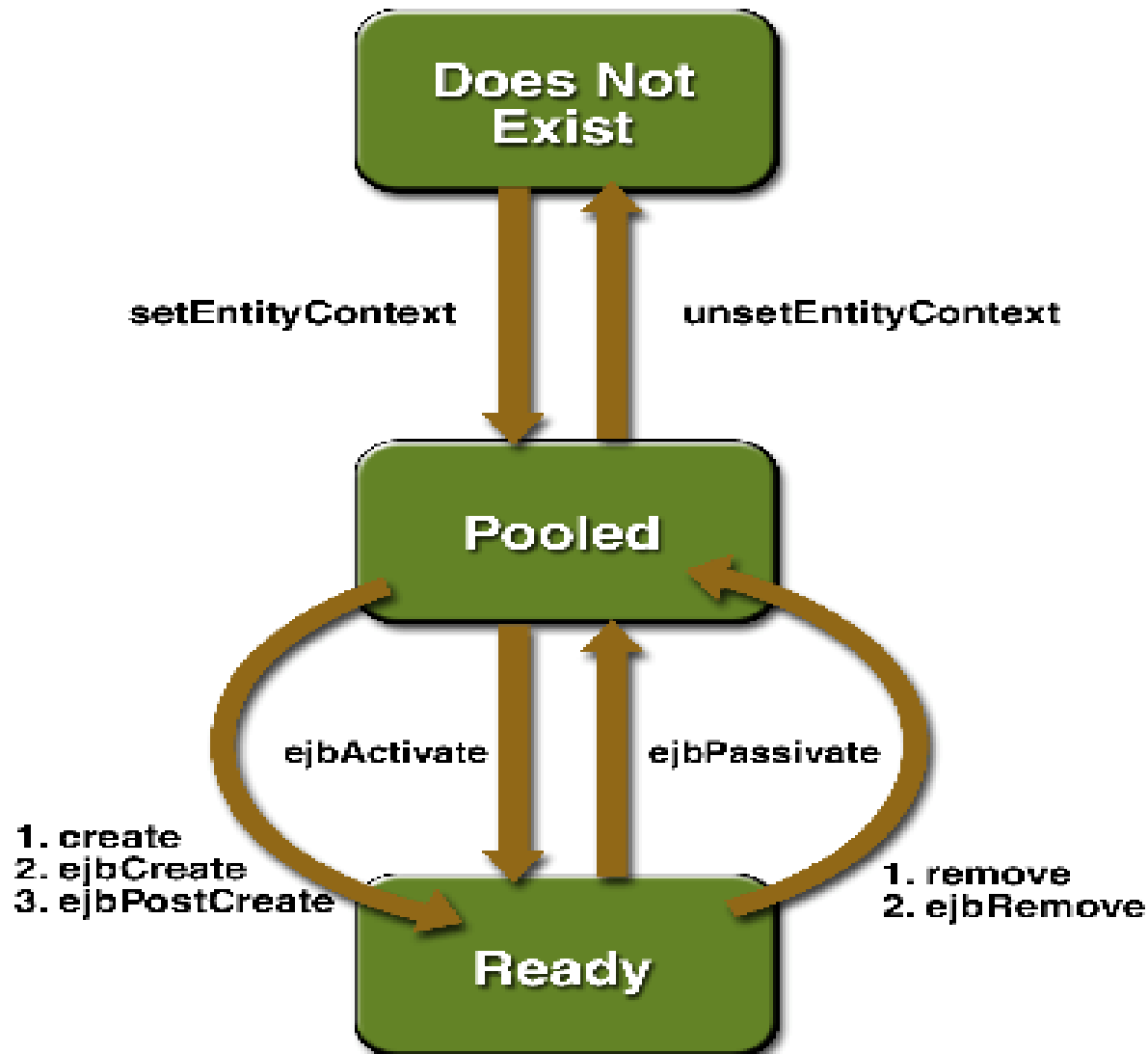
# Stateful Session Beans: Lebenszyklus



# Entity Beans

- Stehen konzeptuell für “Dinge” und entsprechen **i.d.R.** Zeilen in Tabellen
- Haben eine Primary Key Klasse (fast immer `java.lang.Integer`)
- Home-Interface hat immer `findByPrimaryKey()`
- Im PetStore: Account, Order, Inventory, SignOn

# Entity Beans: Lebenszyklus



# Container-Managed Persistence (1)

- Container verwaltet:
  - Verbindung zwischen Instanz-Variablen der EJB-Klasse und DB-Spalten
  - Suchverfahren für `findByXXX()`-Methoden der Home-Interface
- Im Deployment-Deskriptor:

```
<cmp-field><field-name>description</field-name></cmp-field>  
<cmp-field><field-name>price</field-name></cmp-field>
```

# Container-Managed Persistence (2)

- Deklaration für die Verbindung zur DB ist abhängig vom AppServer
  - Bei S1AS in `XXX-ias-cmp.xml`, wobei XXX ein selbst gewählter Name ist
- Kein Beispiel im PetStore
  - Aber im iAS Sample App “J2EE Developer's Guide”, ProductEJB

# Bean-Managed Persistence

- Entwickler schreibt eigenen Code für die Interaktion mit einer DB
- Z.B. eigener Code für JDBC
  - Am besten in Standard-Klassen verkapselt (DAOs == Data Access Objects)
- Oder Code, der Inhalte andere Entity Beans zusammensetzt
- PetStore: Customer, Inventory (DAO)

# EJBs: Transaktionen

- Transaction Scope: Beans, die während eines Methodenaufrufs an einer Transaktion beteiligt sind
- Transaction Attribute im Deployment Descriptor legt die Verbindungen zwischen Methodenaufrufen bzgl. Transaktionen fest
- Bewirkt, ob Exceptions zu Rollback führen

# Transaction Attributes

- Not Supported
- Supports
- Required
- Requires New
- Mandatory
- Never
- Bean Managed

# Best Practices für Enterprise Java Beans (1)

- Remote-Aufrufe minimieren, da das Netzwerk zum Engpass wird
  - Value Object Design Pattern:  
Leichtgewichtige Objekte mit allen Feldern einer EJB gehen über die Leitung
- Granularität von Entity Beans soll nicht zu fein sein
  - DAOs bei Objekten mit wenigen Feldern

# Best Practices für Enterprise Java Beans (2)

- Entity Beans sollen häufig wieder verwendet werden
  - Sonst DAOs verwenden
- Keine Business Logik in Entity Beans: Klienten sollen nur auf Session Beans zugreifen
  - Session Facade Design Pattern
- Wo möglich, Stateless Session Beans

# EJBs: Referenzen

- Richard Monson-Haefel, Enterprise Java Beans, 3<sup>rd</sup> Edition
  - <http://www.oreilly.com/catalog/entjbeans3/>
- Roman et al., Mastering Enterprise Java Beans, 2<sup>nd</sup> Edition
  - <http://www.theserverside.com/books/masteringEJB/index.jsp>
- Alur et al., Core J2EE Patterns,  
Kapiteln zur Business- und  
Integrations-schicht