
Software
Architectures
for Enterprises
SA4E
Peter Sturm
Universität Trier
Winter 2024


5. JavaBeans und EJBs
2

5.1 JavaBeans

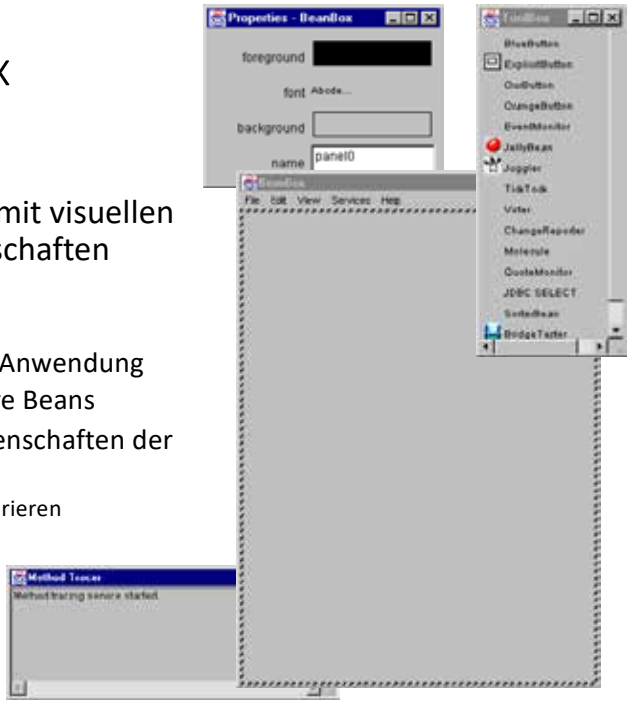
3

JavaBeans

- Komponentenbasierte Programmierung in Java
- JavaBeans sind Komponenten
  - Verfügen über Zugriffsklassen
  - Namenskonventionen bei den Methoden
  - Konfigurierbar über Dialogboxen
  - Neuen Programmiermethoden (Visual Programming)

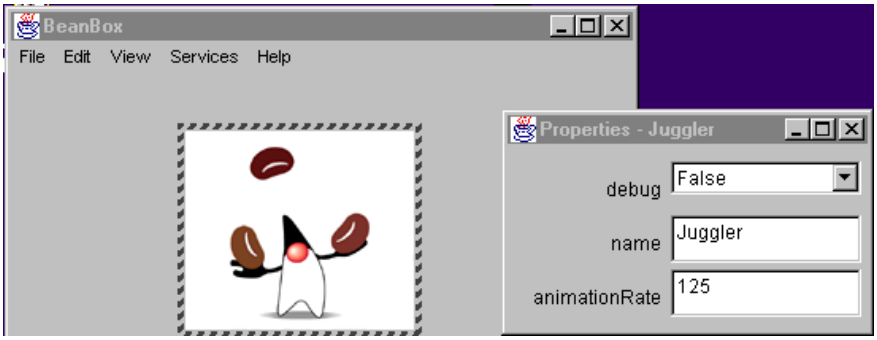
Die BeanBox

- Prototyp einer IDE mit visuellen Programmiereigenschaften
- 4 Fenster
  - BeanBox: Aktuelle Anwendung
  - ToolBox: Verfügbare Beans
  - PropertiesBox: Eigenschaften der aktuellen Bean
    - Lesen und Konfigurieren
  - MessageTracer



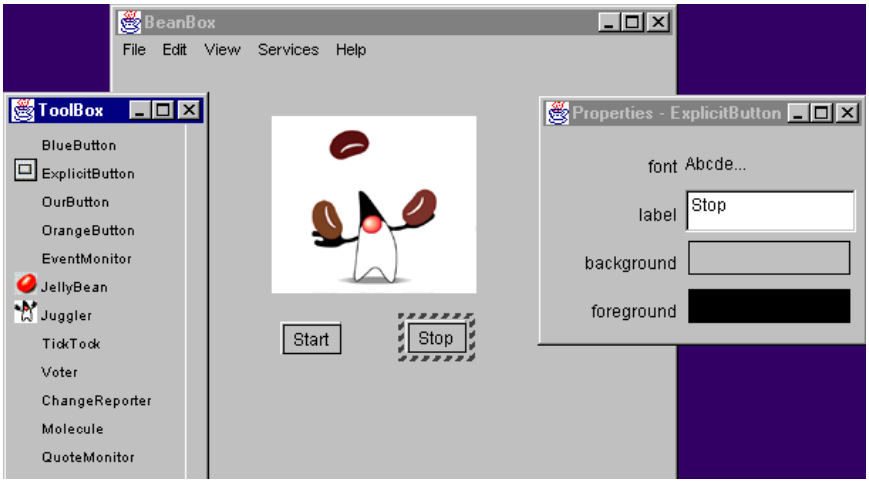
Beispiel: JugglerBean (1)

- JugglerBean aus Toolbox auswählen und plazieren



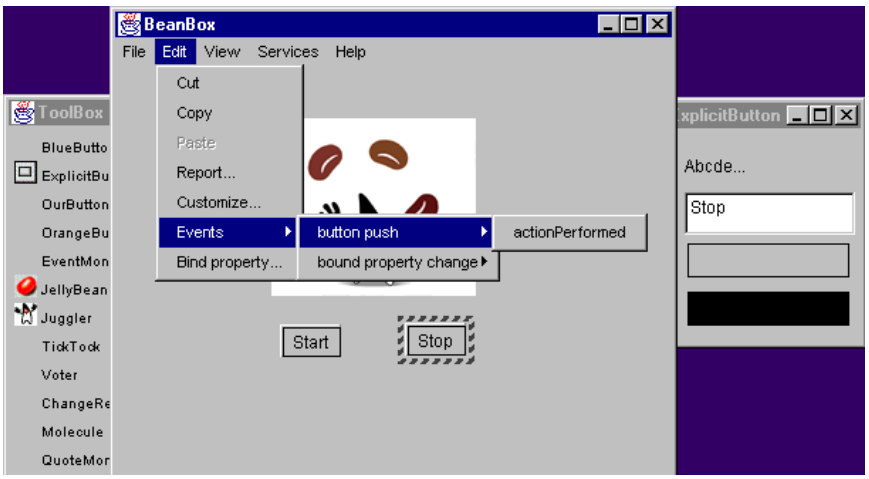
JugglerBean (2)

- 2 ExplicitButton mit „Start“ und „Stop“ hinzufügen



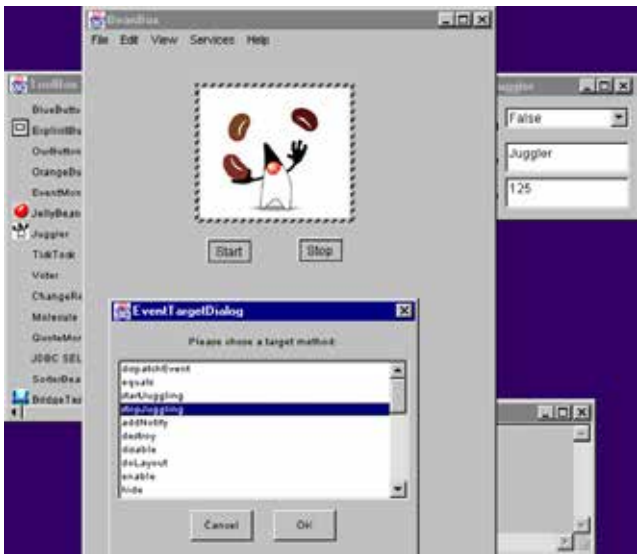
JugglerBean (3)

- „Stop“: Edit->Events->button push->actionPerformed



JugglerBean (4)

- ... mit JugglerBean verknüpfen



JugglerBean (5)

- Automatisch generierter Code

```
1  package tmp.sunw.beanbox;
2
3  import sunw.demo.juggler.Juggler;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6
7  public class ___Hookup_1653fe1423
8      implements java.awt.event.ActionListener, java.io.Serializable {
9
10     public void setTarget(sunw.demo.juggler.Juggler t) {
11         target = t;
12     }
13
14     public void actionPerformed (java.awt.event.ActionEvent arg0) {
15         target.stopJuggling(arg0);
16     }
17
18     private sunw.demo.juggler.Juggler target;
19 }
```

BeanProperties

- BeanBox erkennt Properties
  - Namensstruktur der Zugriffsfunktionen
- BeanProperty X
  - Verwendung sogenannter Decapitalization:
    - aus `setMeineProp` bzw. `getMeineProp` wird `meineProp`
- Angabe der Zugriffsfunktionen:
  - `public X getX ()`
    - Ausnahme: `public boolean isX ()`
  - `public void setX ( X x )`
- Nurleseeigenschaften: Keine `setX()`-Methode

Property-Arten

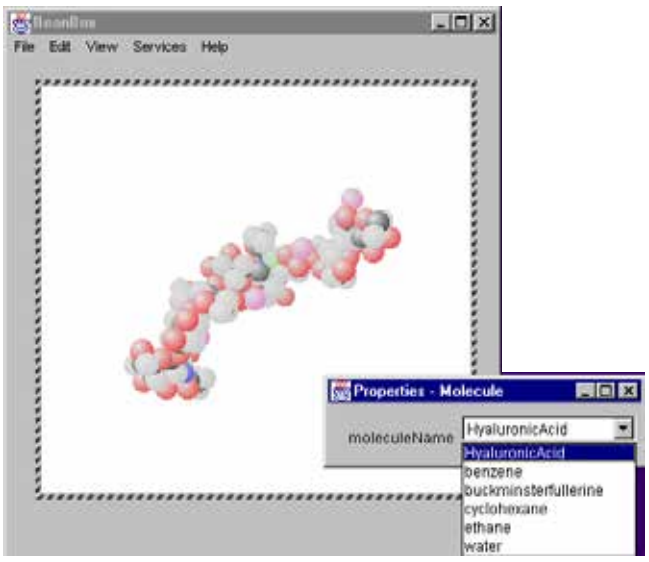
- Simple Property
  - Variable speichert einen einfachen Wert
- Indexed Property
  - Speicherung eines Feldes
  - Zugriffsfunktionen
    - `X[] getX ()`
    - `void setX ( X[] x)`
    - `X getX ( int i )`
      `void setX ( int i, X x )`

Property-Arten (contd.)

- Bound Property
  - Listener werden über Änderungen informiert
  - Zusätzlicher Implementierungsaufwand
    - Bean muß bei Änderung PropertyChange-Event senden
    - Verwaltung aller Listener:
      - `void addPropertyChangeListener(…)`
      - `void removePropertyChangeListener (…)`
    - Convenience-Klasse PropertyChangeSupport vorhanden
- Constraint Property
  - Bound Property mit Vetorecht der Listener

Property-Editoren

- BeanBox stellt für Grundtypen Editoren zur Verfügung
- Editoren für anwendungs-spezifische Methoden integrierbar



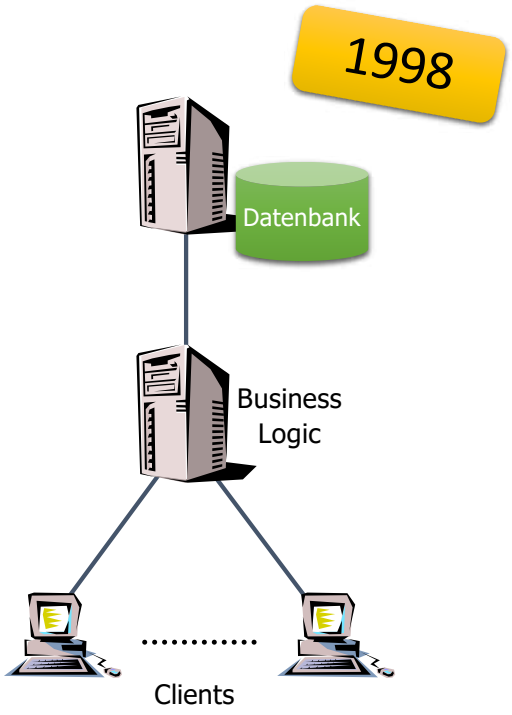
Alternativen

- BeanBox ist veraltet ☹
  - Schöne Demonstration wesentlicher Bean-Eigenschaften
- „Bean Builder“ auch veraltet ☹
- Primärer Einsatz: IDEs
  - JBuilder
  - NetBeans
  - Eclipse
  - …

5.2 Enterprise JavaBeans (EJB)

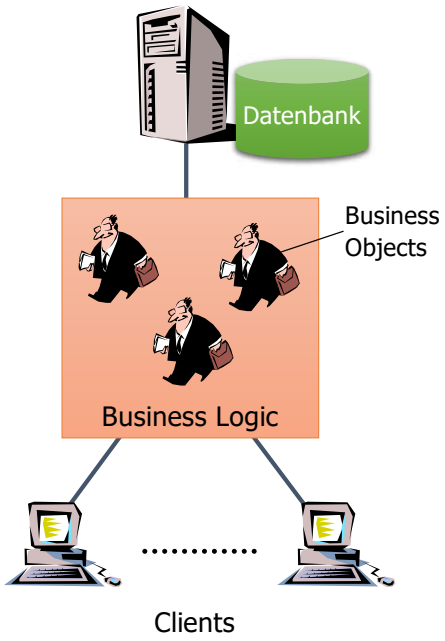
Motivation

- Bean = Komponente
- Zielgruppe
  - Kommerzielle Anwendungen
  - E-Commerce
- Evolution früherer
  - Client/Server-Systeme
  - Transaktionsmonitore
- 3-Tier-Applikationen
  - Client
  - Bussiness Application
  - Database



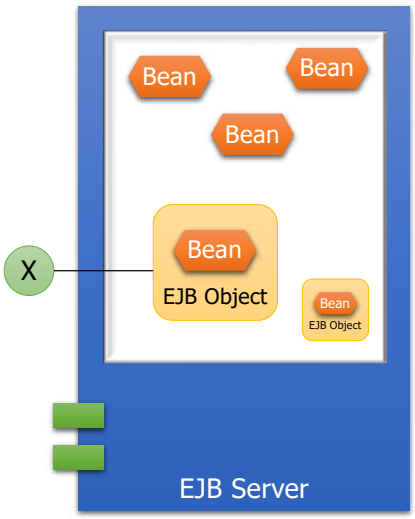
Business Objects

- Komponenten-basierte Anwendung
- Varianten
  - Eigenständige Komponenten
  - Wrapper für Legacy Software
  - Client-Transaktionen
- Laufzeitumgebung
  - EJB Server



Aufbau des EJB-Server

- Laufzeitumgebung
  - Namensverwaltung
  - Anbindung an DB
  - Nachrichtenkommunikation
- EJB Object = Bean Wrapper
  - Indirektionsstufe
  - Zugriff über Reflection
- Beantypen
  - Entity Bean
    - Zustand, Persistent
  - Session Bean
    - Transient
    - Stateless, Stateful
  - Message-Driven Beans



Aufgaben des Servers

- Resource Management
  - Instance Pooling: Beans vorinstanziiert
  - Instance Swapping: Wechsel der EJB-Object-Bindung
  - Activation: Stilllegen und Reaktivieren von Beans
- Primary Services
  - Concurrency: Single-Threaded Bean-Implementierung
  - Transactions: ACID-Prinzip
  - Persistence: Container-Managed, Bean-Managed
  - Distribution: RMI over JRMP oder RMI-IIOP
  - Naming: JNDI unterstützt LDAP, NIS+, DNS, ...
  - Security: Methodenbasierte Zugriffskontrolle

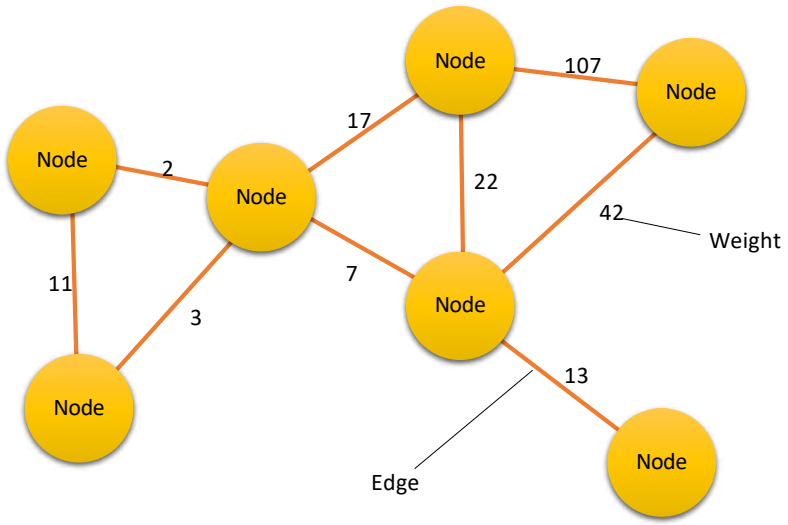
Entity Beans

- Repräsentieren zustandsbehaftete Anwendungsobjekte
  - „Can be expressed as nouns in the business model“ (Burke)
- Persistente Speicherung in Datenbank
  - Abbildung über Annotationen aus Java Persistence Spezifikation
- Objekt-relationales Mapping

Entity Beans

- Keine Komponente im EJB-Sinn
  - sondern ein „POJO“ (Plain old Java Object) -> Klasse
- Mapping
  - Fields über Annotationen auf Spalten einer DB-Realtion
  - Alternativ über Map-File (XML)
  - besitzt einen Primärschlüssel
- Wird von EntityManager verwaltet

Beispiel



Beispiel Node

```
package syssoft.ejb_graph;

import javax.persistence.*;

@Entity
@Table(name="GRAPH_NODES")
public class Node implements java.io.Serializable {
     private static final long serialVersionUID = 42;
     private int id;
     private String name;

     @Id
     @Column(name="ID")
     public int getId() { return id; }
     public void setId ( int id ) { this.id = id; }

     @Column(name="NAME")
     public String getName() { return name; }
     public void setName ( String name ) { this.name = name; }
}
```

Beispiel Edge

```
package syssoft.ejb_graph;

import javax.persistence.*;

@Entity
@Table(name="GRAPH_EDGES")
public class Edge implements java.io.Serializable {
	private static final long serialVersionUID = 1L;
	private int id;
	private int weight;
	private int l;
	private int r;

	@Id
	@Column(name="ID")
	@GeneratedValue
	public int getId() { return id; }
	public void setId ( int id ) { this.id = id; }

	@Column(name="WEIGHT")
	public int getWeight() { return weight; }
	public void setWeight ( int weight ) { this.weight = weight; }

	@Column(name="LEFT")
	public int getLeft() { return l; }
	public void setLeft ( int l ) { this.l = l; }

	@Column(name="RIGHT")
	public int getRight() { return r; }
	public void setRight ( int r ) { this.r = r; }

}
```

Abbildung auf Datenbank

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
     <persistence-unit name="syssoft_graphs">
          <jta-data-source>java:/DefaultDS</jta-data-source>
          <properties>
               <property name="hibernate.hbm2ddl.auto“
                    value="create-drop"/>
          </properties>
     </persistence-unit>
</persistence>
```

- DefaultDS = Standard-DB des jeweiligen Application Servers
- hibernate.hbm2ddl.auto
  - Automatische Abbildung der EntityBeans auf DB-Tabellen
- Create-drop
  - Tabellen beim Deploy erzeugen und beim Remove wieder löschen

Stateless Session Bean

- Remote-Interface definiert Funktionalität
- Bean-Klasse implementiert Remote-Interface
- PersistenceContext realisiert Zugang zu Entity Beans

Interface GraphRemote

```
package syssoft.ejb_graph;

import javax.ejb.*;

@Remote
public interface GraphRemote {

     public void createNode ( Node n );
     public Node findNode ( int id );

     public void connectNodes (
          int left_node,
          int right_node,
          int weight
     );
}
```

Stateless Session Bean Graph

```
package syssoft.ejb_graph;

import javax.ejb.*;
import javax.persistence.*;

@Stateless
public class Graph implements GraphRemote {
     @PersistenceContext(unitName="syssoft_graphs") private EntityManager em;

     public void createNode ( Node n ) {
          em.persist(n);
     }

     public Node findNode ( int id ) {
          return em.find(Node.class, id);
     }

     public void connectNodes ( int left, int right, int weight  ) {
          Edge e = new Edge();
          e.setWeight(weight);
          e.setLeft(left);
          e.setRight(right);
          em.persist(e);
     }
```

Stateful Session Beans

- Speichern „conversational state“
- Feste Bindung zu jeweils einem Client
- Warenkorb etc.

Message Driven Beans

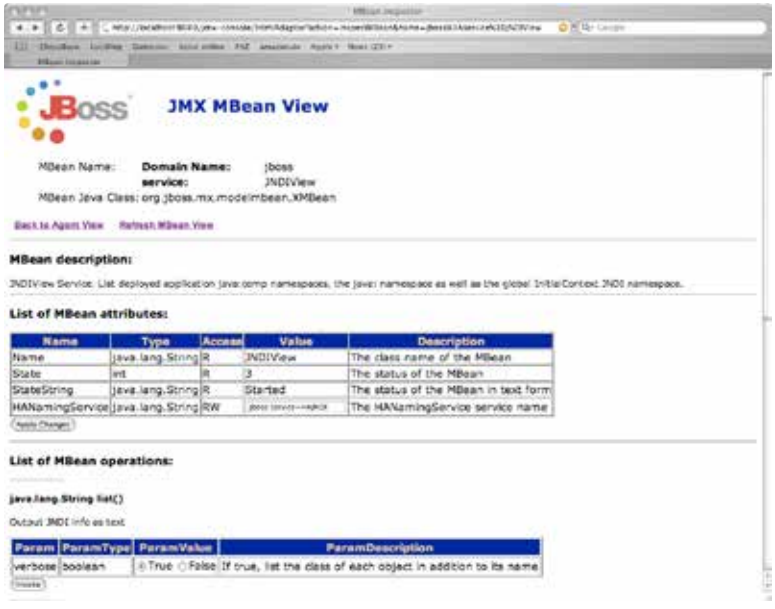
- Asynchrone Kommunikation
  - Client erwartet keine Antwort
- Lebenszyklus der Bean unabhängig von Clients
  - Implementiert kein Remote Interface
  - onMessage (Message m)
- Unterstützung für verschiedene Messaging Systeme

Mögliche Umsetzung

- Application Server
  - JBoss, Version 4.0.5
  - jems-Installer für EJB 3.0
  - Standardinstallation
  - Wird später nochmal für Webservices verwendet ;-)
  - Sei JBOSS_HOME das gewählte Installationsverzeichnis
- Starten mit JBOSS_HOME/bin/run.sh
- Webzugang über http://localhost:8080
- Verwendung der Default-DB Hypersonic SQL

Webzugang

MBean JNDIView

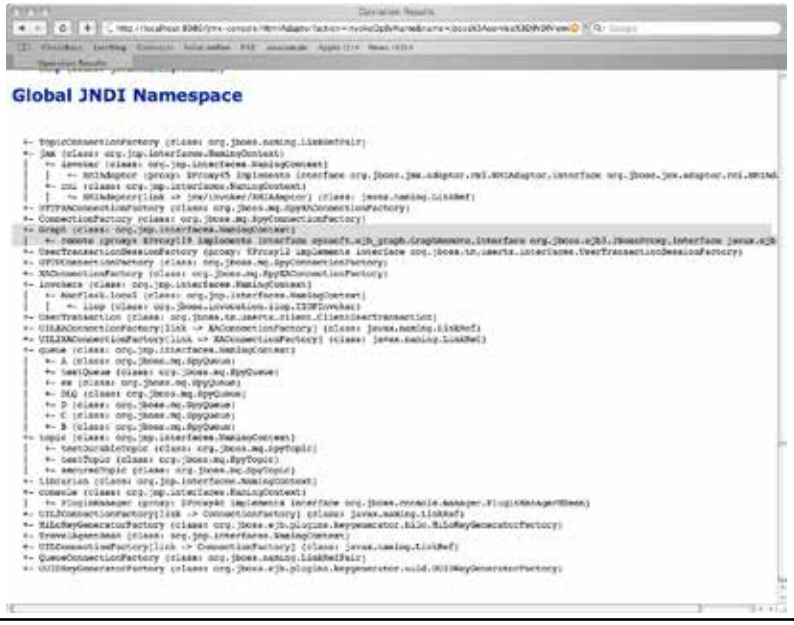


Deployment

- JAR-File in den Ordner Deploy kopieren
  - JBoss beobachtet das Verzeichnis kontinuierlich
  - Einfügen heißt Deploy
  - Löschen heißt Remove

```
META-INF/MANIFEST.MF
syssoft/
syssoft/ejb_graph/
syssoft/ejb_graph/GraphRemote.class
syssoft/ejb_graph/Edge.class
syssoft/ejb_graph/Graph.class
syssoft/ejb_graph/Node.class
META-INF/
META-INF/persistence.xml
```

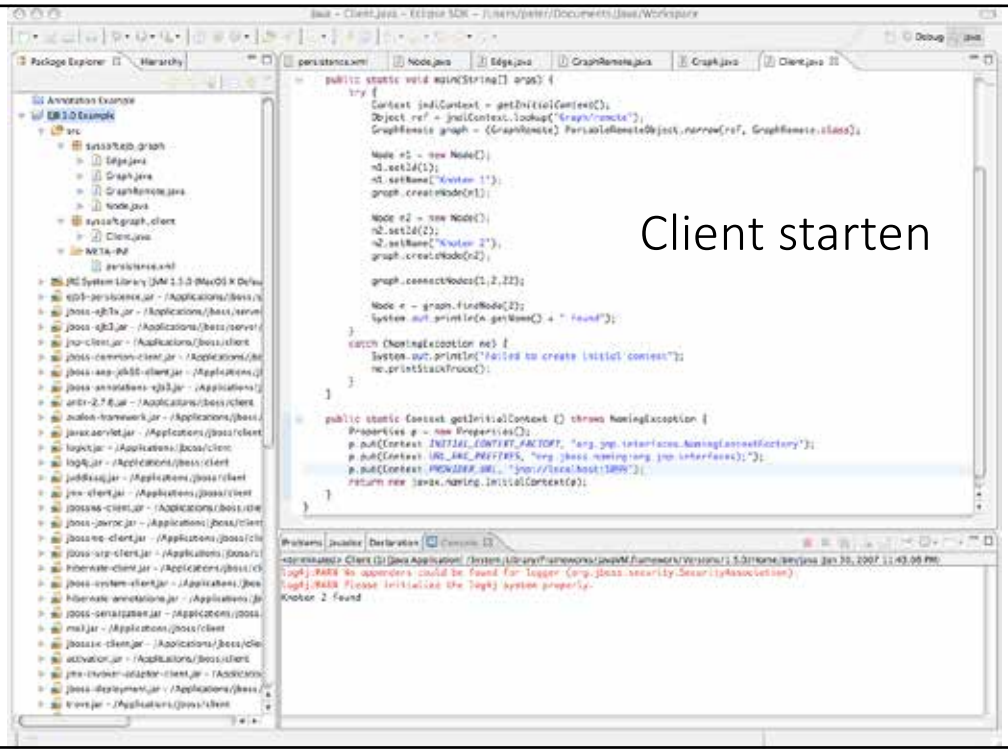
Erfolgreiches Deployment



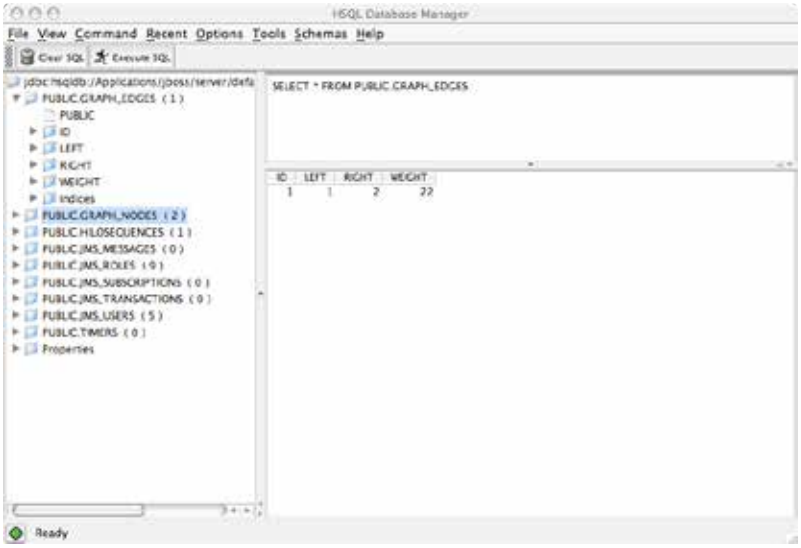
Client

```
package syssoft.graph_client;

import syssoft.ejb_graph.*;

import javax.naming.*;
import javax.rmi.*;
import java.util.*;

public class Client {

	public static void main(String[] args) {
		try {
			Context jndiContext = getInitialContext();
			Object ref = jndiContext.lookup("Graph/remote");
			GraphRemote graph = (GraphRemote) PortableRemoteObject.narrow(ref, GraphRemote.class);

			Node n1 = new Node();
			n1.setId(1);
			n1.setName("Knoten 1");
			graph.createNode(n1);

			[…]

			Node n = graph.findNode(2);
			System.out.println(n.getName() + " found");
		}
		catch (NamingException ne) {
			System.out.println("Failed to create initial context");
			ne.printStackTrace();
		}
	}

	public static Context getInitialContext () throws NamingException {
		Properties p = new Properties();
		p.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
		p.put(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces);");
		p.put(Context.PROVIDER_URL, "jnp://localhost:1099");
		return new javax.naming.InitialContext(p);
	}
}
```

Client starten

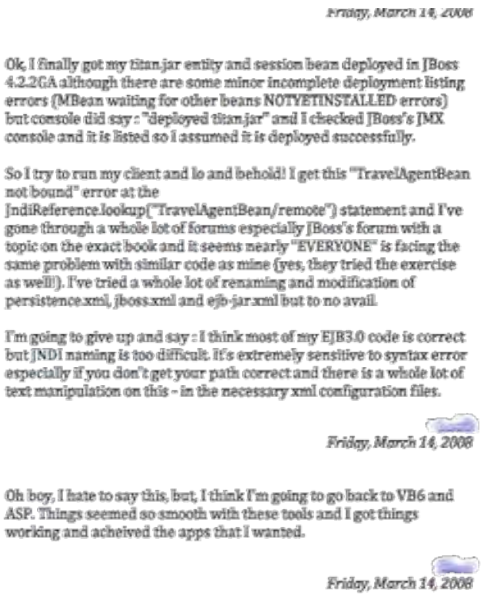


Datenbankinhalt



Video

Andere Meinungen dazu



Friday, March 14, 2008

Ok, I finally got my titan.jar entity and session bean deployed in JBoss 4.2.2GA although there are some minor incomplete deployment listing errors (MBean waiting for other beans NOTYETINSTALLED errors) but console did say : "deployed titan.jar" and I checked JBoss's JMX console and it is listed so I assumed it is deployed successfully.

So I try to run my client and lo and behold! I get this "TravelAgentBean not bound" error at the InitialReference.lookup("TravelAgentBean/remote") statement and I've gone through a whole lot of forums especially JBoss's forum with a topic on the exact book and it seems nearly "EVERYONE" is facing the same problem with similar code as mine (yes, they tried the exercise as well!). I've tried a whole lot of renaming and modification of persistence.xml, jboss.xml and ejb-jar.xml but to no avail.

I'm going to give up and say : I think most of my EJB3.0 code is correct but JNDI naming is too difficult. It's extremely sensitive to syntax error especially if you don't get your path correct and there is a whole lot of text manipulation on this - in the necessary xml configuration files.

Friday, March 14, 2008

Oh boy, I hate to say this, but, I think I'm going to go back to VB6 and ASP. Things seemed so smooth with these tools and I got things working and acheived the apps that I wanted.

Friday, March 14, 2008

- Ich bin nicht allein ☺

Literatur

- Bruce Eckel
  Thinking in Java
  4. Auflage, Prentice Hall
- Bill Burke, Richard Monson-Haefel
  Enterprise JavaBeans 3.0
  5. Auflage, O'Reilly