Präsentation
Großer Beleg

Aspektorientierte Programmierung und Enterprise JavaBeans im Vergleich

Vorstellung des Belegthemas

- Entwicklung einer Umgebung mit AOP
  - Erweiterung der Funktionalität einer Anwendungsgruppe
- Idee – Nachbildung eines EJB Containers mit AOP
  - Test auf Eignung und Flexibilität
- Nachbildung der Funktionalität eines Entity Beans
  - Zugriff auf das verteilte Entity Bean
  - Persistenzhaltung in der Datenbank
- Vergleich der Lösungen mit AOP und EJB
  - Realisierbarkeit mit AOP
  - Bringt AOP Vorteile bei Gestaltung einer Umgebung?

Gliederung

1. Einführung in AOP
2. Funktionalität eines EJB Entity Beans
3. Umsetzung einer Umgebung mit AOP
4. Vergleich der Lösungen mit AOP und EJB
5. Zusammenfassung

Gliederung

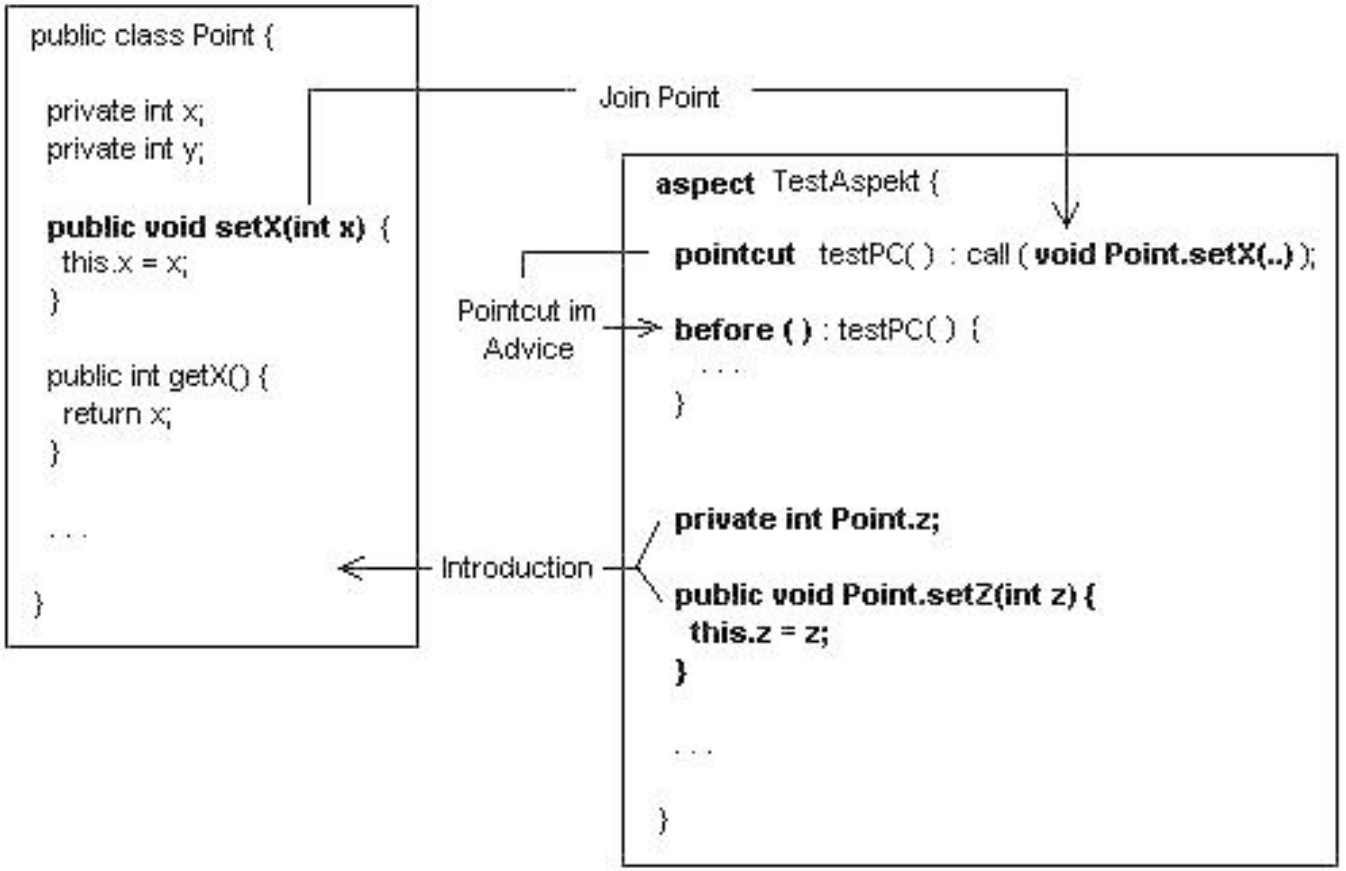
- **Einführung in AOP**

2. Funktionalität eines EJB Entity Beans
3. Umsetzung einer Umgebung mit AOP
4. Vergleich der Lösungen mit AOP und EJB
5. Zusammenfassung

Einführung in AOP

- AOP für jede Programmiersprache
- z.B. Objektorientierte Programmierung
  - Kapselung von Funktionseinheiten in Objekten
  - durch Zusammenarbeit und Kommunikation zusätzliche Funktionalität = „Crosscutts“
- AOP: Kapselung der Crosscuts in modularer Form in Aspekten
- Zugriff auf Klassen über Join Points
- Konstrukte der AOP:
  - Aspekt, Pointcut, Advice, Introduction

AOP – Konstrukte in AspectJ



Gliederung

1. Einführung in AOP
   - ➢ **Funktionalität eines EJB Entity Beans**
3. Umsetzung einer Umgebung mit AOP
4. Vergleich der Lösungen mit AOP und EJB
5. Zusammenfassung

Funktionalität eines EJB Entity Beans

- serverseitige Objektabbildung persistenter Daten
- Entity Bean Klasse / Remote und RemoteHome Schnittstellen (keine Local und LocalHome)
- Remote Schnittstelle:
  - Geschäftsmethoden (`get / set`)
- RemoteHome Schnittstelle:
  - Lebenszyklus (`create`–, `find`– und `home` – Methoden)
- Entity Bean Klasse:
  - Implementierung der Methoden der Schnittstellen
  - `select` – Methoden

Gliederung

1. Einführung in AOP
2. Funktionalität eines EJB Entity Beans
   - **Umsetzung einer Umgebung mit AOP**
     - 3.1. Anforderungen an eine AOP – Umgebung
     - 3.2. Realisierbarkeit einer Umgebung mit AOP
     - 3.3. Vorstellung der entwickelten Umgebung für ein Entity Bean
4. Vergleich der Lösungen mit AOP und EJB
5. Zusammenfassung

Anforderungen an eine AOP – Umgebung

- Gültigkeit für eine Vielzahl verschiedener Anwendungen (z.B. Entity Bean)
- Modularisierung:
  - Unterteilung in Funktionseinheiten
  - beliebige Kombination der Moduleinheiten
- Anpassbarkeit an Anforderungen

  (z.B. lokaler und verteilter Zugriff)

Realisierbarkeit einer Umgebung mit AOP

- über Join Point:
  - Zugriff auf Klassendeklaration (z.B. Attribute)
  - Laufzeitinformationen (Parameter – neue Attributwerte)

  → Verarbeitung der Informationen

  → Umgebung realisierbar
- Konventionen für allgemein gültige Umgebung, z.B.:
  - persistente Attribute: `private`
  - Namenskonventionen:
    - Endungen: `Entity`, `Remote`, `RemoteHome`
    - gleicher Name: (`PassengerEntity`, `PassengerRemote`, `PassengerRemoteHome`)
    - Methoden beginnend mit: `set, get, create` und `find`
    - Primärschlüssel Endung: `_PK` (z.B. `id_PK`)

Vorstellung der entwickelten Umgebung für ein Entity Bean

Funktionseinheiten der AOP – Umgebung:

(1) Zugriff auf das Entity Bean (verteilt/lokal)

(2) Datenbankzugriff

(3) Objektverwaltung im Server

(4) Locking (Blockierung) der Objekte

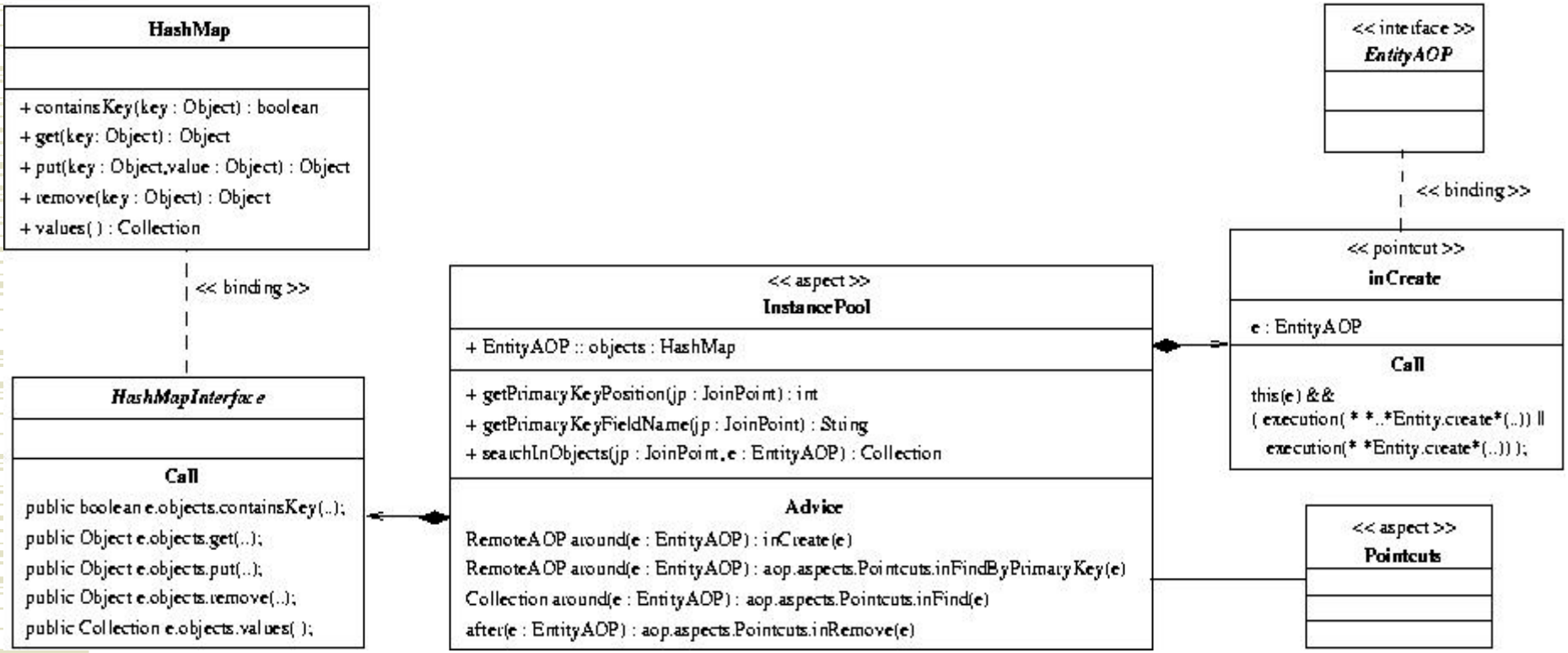
Objektverwaltung im Server

- im Server nur ein Objekt je Eintrag in Datenbank
- Aspekt: `InstancePool`
- Zugriff bei den Methoden:
  `create`, `find`, `remove`

<u>Verbesserungsvorschläge:</u>

- *anlehnend an EJB: Objekte ohne Werte halten*
  - *bessere Performanz*
- *Freigabe von Objekten*

Klassendiagramm: InstancePool



Gliederung

1. Einführung in AOP
2. Funktionalität eines EJB Entity Beans
3. Umsetzung einer Umgebung mit AOP
   - **Vergleich der Lösungen mit AOP und EJB**
     - 4.1. Anpassungsaufwand einer Anwendung
     - 4.2. Anpassungsmöglichkeiten der Umgebung
5. Zusammenfassung

Vergleich der Lösungen

- EJB – Funktionalität mit AOP nachbildbar
- Ziel: eigenständig funktionsfähige Anwendung
  - in AOP bessere Möglichkeiten
  - mit AspectJ Eingriff in existierenden Programmverlauf
  - der original Programmverlauf bleibt erhalten

Anpassungsaufwand einer Anwendung (I.)

EJB:

- Server – Anwendung
  - keine Attribute
  - keine Konstruktoren
  - abstrakte `get`– und `set` – Methoden
  - keine `find` – Methoden im Entity Bean
  - XML Deployment Descriptor
  - Ableitung von EJB Schnittstellen und Klassen
  - nicht eigenständig funktionsfähig
- Client – Anwendung
  - verteilten Zugriff beachten
  - Verbindungsaufbau

Anpassungsaufwand einer Anwendung (II.)

AOP:

- Server – Anwendung
  - in Anlehnung an den Aufbau eines Java Beans
  - zusätzlich `create-`, `find-`, `remove-` und `release` – Methoden
  - Implementierung der `Remote-` und `RemoteHome` – Schnittstellen
  - Default – Konstruktor gefordert
  - Initialisierung im `Server` – Aspekt
  - Namenskonventionen (z.B. ...`Entity` , ...`Remote`, ...`RemoteHome`)
  - viele Konventionen aber Anwendung eigenständig funktionsfähig
- Client – Anwendung
  - `getHomeRemote` – Methode
  - Angabe der Klasse / des Packages

Anpassungsmöglichkeiten der Umgebungen

- Ziel: Anpassung der Umgebung an gestellte Anforderungen
- EJB in Grenzen möglich
  - Deaktivierung von Container Management bei Persistenzhaltung und Transaktionsverwaltung
- AOP besser durch Modularisierung
  - wiederverwendbare und kombinierbare Module

Gliederung

1. Einführung in AOP
2. Funktionalität eines EJB Entity Beans
3. Umsetzung einer Umgebung mit AOP
4. Vergleich der Lösungen mit AOP und EJB
   - **Zusammenfassung**

Zusammenfassung

- Umgebung realisierbar / Lösung nicht vollständig
- AOP – Umgebung leicht erweiterbar
- unabhängigere Gestaltung der Anwendung
  - durch Eingriff in existierende Klassenstruktur
  - erhöht Wiederverwendbarkeit der Anwendung
- anderer Ansatz:
  - AOP zur Erweiterung von EJB
  - z.B. Ableitung von EJB – Schnittstellen und –Klassen

ENDE

Vielen Dank für ihre Aufmerksamkeit!

Anhang

- **Weitere Module der AOP – Umgebung**
- **Besonderheiten der Modularisierung**

(1.) Zugriff auf das Entity Bean

Server:

- im `Server` – Aspekt Bindung der Objekte

Client:

- `LocalClient` oder `RemoteClient` – Aspekt
- Zugriff Verborgen vor Clientanwendung
- Einführung einer Methode die Zugriff realisiert
- `getHomeRemote(„EntityBean“)`

(2.) Datenbankzugriff

- Zugriff bei den Methoden:
  `create`, `find`, `set*`, `get*`, `remove`
- Datenbankanfragen
  - kein XML
  - Informationen über Join Points
- Datenbankzugriff
- Auswertung der Ergebnisse

(4.) Locking der Objekte

- Ansatz für Transaktionsverwaltung
- nur ein Client hat Zugriff auf ein Objekt
- Blockierung:
  - `create` und `find`
- Aufhebung der Blockierung:
  - `release`

Probleme:

- *keine Nutzerverwaltung*
  - *Client mit Zugriff nicht registriert*
  - *z.B. besitzt Zugriff und erneuter Aufruf über `find` – Methode*

Besonderheiten der Modularisierung

Abhängigkeit zwischen den Modulen

a) Reduzierung des Funktionsumfangs
   - Objektverwaltung / Datenbankzugriff und nur eine Serveranwendung
   - nur ein Objekt im Server
   - bei Änderung von Einträgen aktueller Wert im Objekt
   - keine Aktualisierung bei Aufruf einer `get` – Methode

b) Notwendigkeit von Modulen
   - Locking nur sinnvoll bei Objektverwaltung
   - ABER – Objektverwaltung beliebig

Vergleich der Lösungen (II.)

- manuelle Übertragung von Schnittstellen und Stub in Clientumgebung
- in EJB durch Referenzimplementierung
- gleicher zeitlicher Aufwand
- Arbeitsschritte bei Änderungen in AOP – Umgebung geringerer

Vergleich der Lösungen (III.)

Änderungen (z.B. neues Attribut):

- EJB – Container
  - Änderungen in Entity Bean Klasse und Remote Schnittstelle
  - erneute Kompilierung
  - Aktualisierung im XML Deployment Descriptor
  - Aktualisierung im Deployment Tool
  - Start Deployment – Vorgang
  - Clientanwendung auf Clientumgebung übertragen
- AOP – Umgebung
  - Änderungen in Entity Bean Klasse und Remote Schnittstelle
  - erneute Kompilierung (AspectJ und RMI Compiler)
  - Schnittstelle und Stub auf Clientumgebung übertragen