JavaOne℠
Sun's 2000 Worldwide Java Developer Conference℠

EJB Design Strategies and Performance Optimizations

Ed Roman
CEO, The Middleware Company
An EJB Training and Consulting Company
http://www.middleware-company.com

Overview of Talk

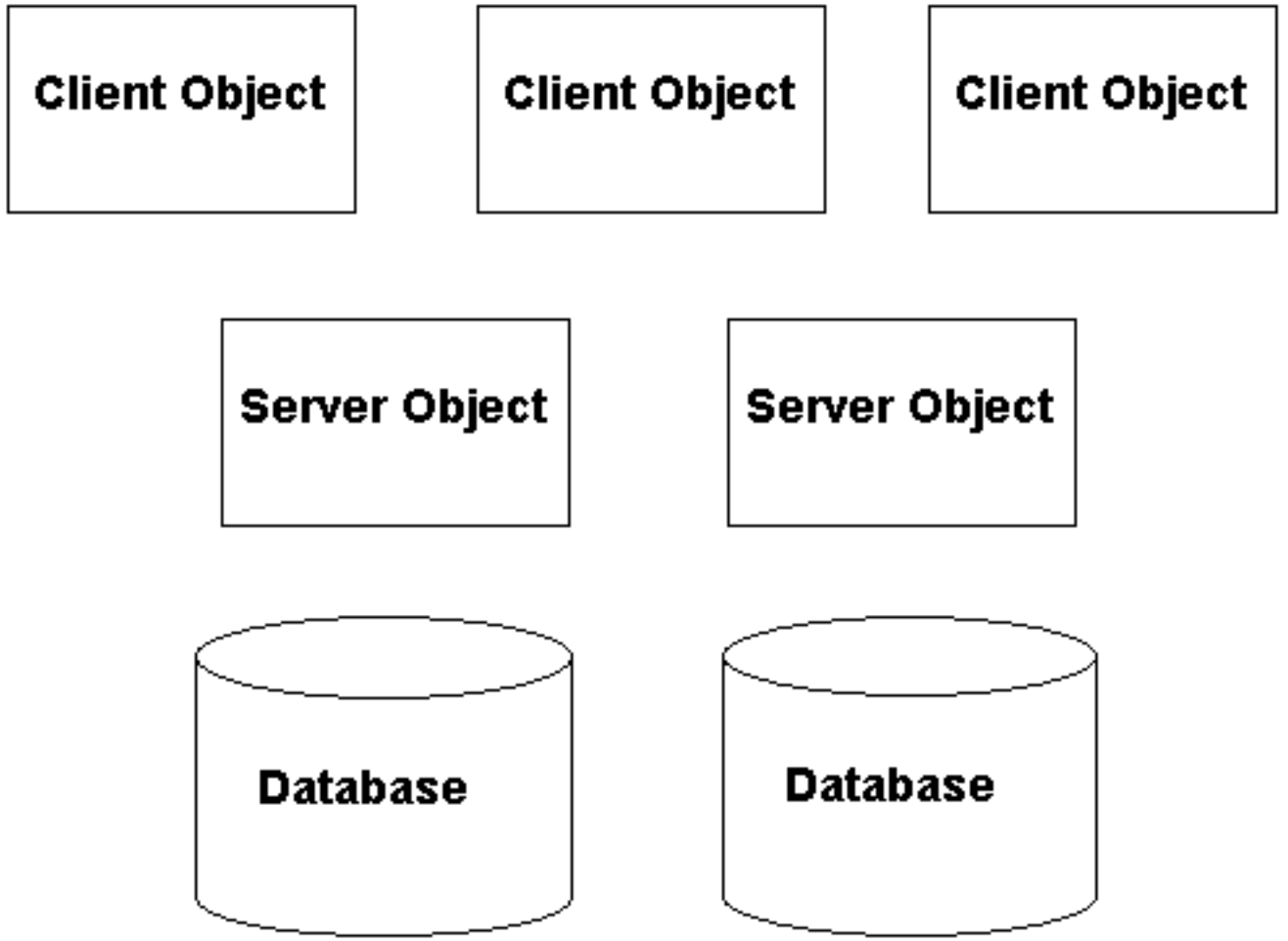
- **Discussion of advanced EJB topics:**
  - Part 1: Properly using transactions for optimizing code and guaranteeing safety
  - 15 minute break
  - Part 2: EJB Design Patterns
- **Intended for Java programmers with some EJB knowledge**
- **Basics of EJB are *not* covered**
- **Information is extracted from our EJB course that we teach at client sites**

About the speaker, Ed Roman…

- Author of book, "Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition"
- Author of Sun whitepaper, "Comparing EJB with COM+"
- Had debate with Roger Sessions about EJB vs. COM+
- CEO, The Middleware Company – EJB Training and Consulting company

Part I: Properly using transactions for optimizing code and guaranteeing safety

Group discussion: what could go wrong here?



What's wrong with this code?

```
try {
   // Withdraw funds from account 1
}
catch (Exception e) {
   // If an error occurred, do not proceed.

   return;
}
try {
   // Otherwise, deposit funds into account 2
}
catch (Exception e) {
   // If an error occurred, do not proceed,
   // and redeposit the funds back into account 1.

   return;
}
```

Transactions and the ACID Properties

- **As we have seen, exceptions are not enough for enterprise computing**
  - Code is non-deterministic
- ***Transactions* guarantee determinism**
- **Transactions give you four virtues, called the ACID properties:**
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- **Let's explore what these values *really* are**

Atomicity ACID Property

- **"All or nothing" operations**
- **You get atomicity from the database**
  - It undoes updates automatically upon failure
- **Example:**
  - Withdraw from one bank, deposit into another
  - Want both or neither to succeed
  - Database will undo deposit or withdraw if either fails

Consistency ACID Property

- **System is always consistent based upon state invariants**
  - But: You don't get consistency for free!
- **Transactions give you an opportunity to write code that checks the system state.**
  - Example: You can write bean code that throws an exception if balance is < 0.

Isolation ACID Property

- **Operations on shared data are isolated.**
- **When a transaction begins, your 'umbrella' opens.**
  - Now you have an isolated, tunnel view of the database
- **When the transaction ends, your 'umbrella' closes**
  - You are no longer safe.
- **You choose how isolated you are from others**
- **Challenge: balance safety and concurrency.**

Safety — Concurrency

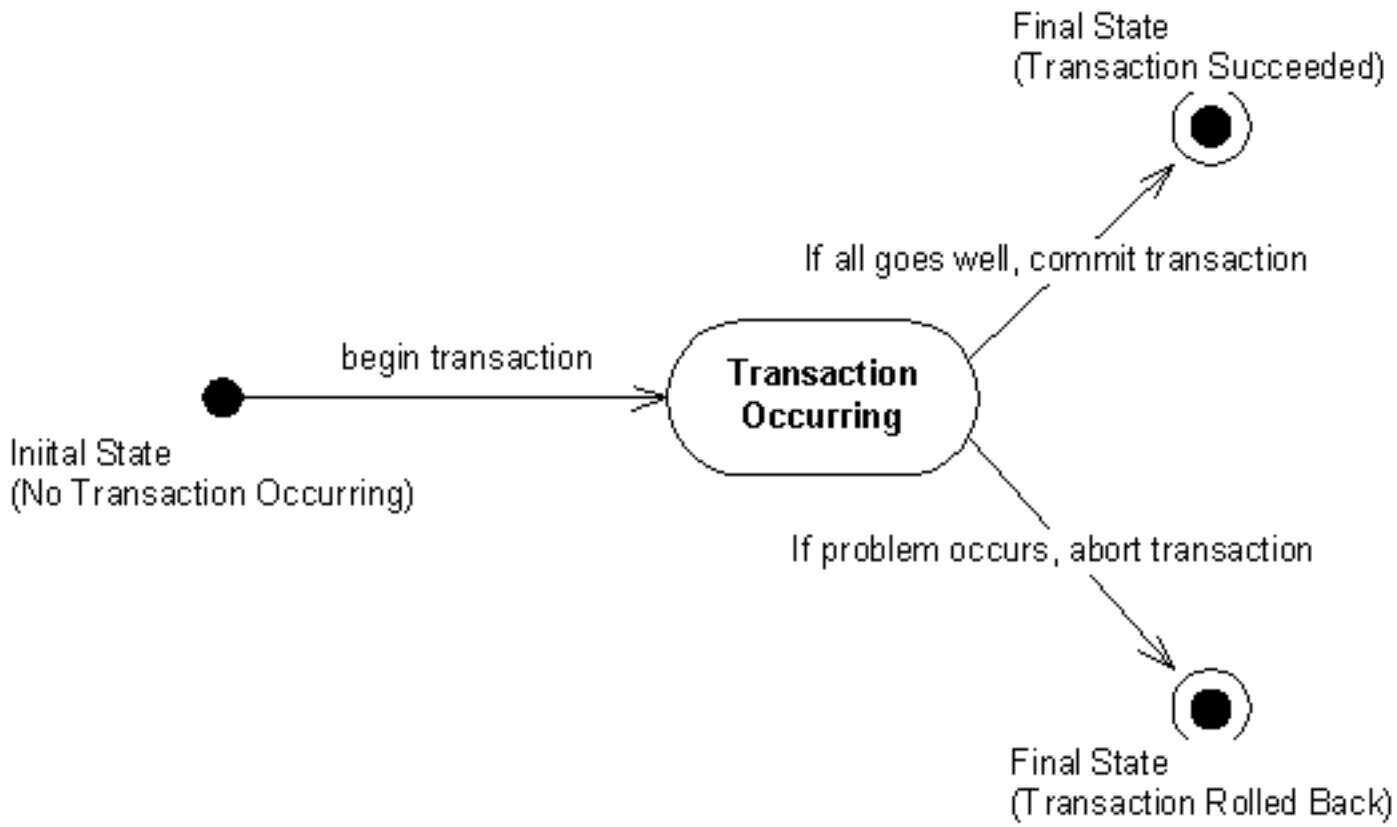
Durability ACID Property

- Catastrophic failure is recoverable.
- If a database crashes, it will reboot and fix itself automatically.

Transaction types

- ***Flat transactions*** **are the most common transaction type**
- **Supported by all databases**
- **Only transaction type supported universally by EJB**

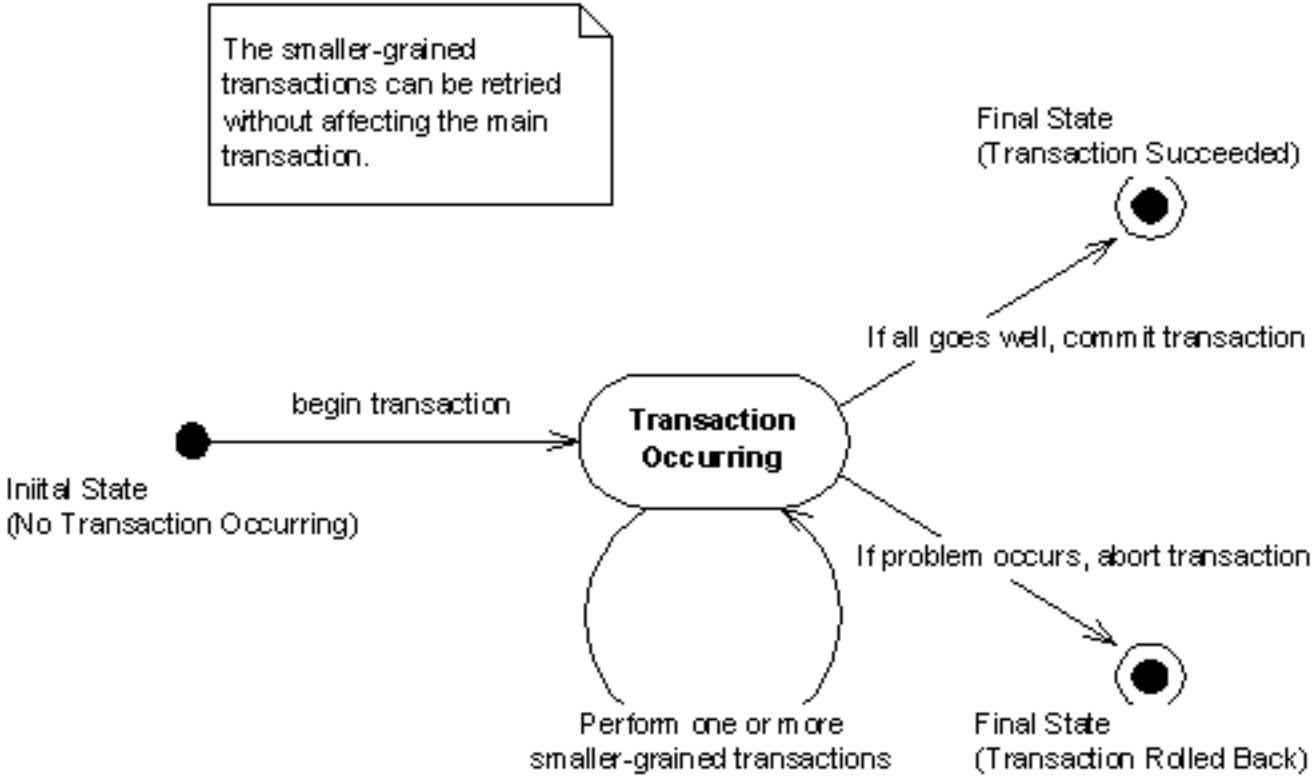
Flat transactions



Nested transactions

- **Unit of work nested in other unit of work**
  - Motivation: Trip-planning problem
  - 1) You buy a ticket from NYC to Los Angeles
  - 2) You buy a ticket from Los Angeles to Japan
  - 3) You buy a ticket from Japan to London, but the flight is booked.
- **Under a flat transaction, this would cause a rollback of all tickets purchased.**
  - But: We might not want this! What if we chose another airline?
- **Nested transactions allow us to retry nested units of work.**
  - But: What if the nested unit of work cannot be made to succeed?
  - Then ultimately the outer unit of work will fail.

Nested transactions



How to design transactional conversations

- **Motivation: A stateful session bean is a resource too.**
  - Has in-memory state that needs to rollback in case of failure.
- **Can participate in transactions like a database by implementing SessionSynchronization:**

```
public interface javax.ejb.SessionSynchronization {
    public void afterBegin();
    public void beforeCompletion();
    public void afterCompletion(boolean);
 }
```

Uses of SessionSynchronization

- **2 uses of *SessionSynchronization*:**
  1. Alerts your bean to transaction failure
     - Enables you to act as a transactional resource
     - You can undo state changes like a DBMS
  2. Alerts you to transaction boundaries
     - Enables you to cache database data for performance
- **What the methods do:**
  - ***afterBegin():*** Transaction has just started. Read in data to cache.
  - ***beforeCompletion():*** Transaction has ended. Write out cached data.
  - ***afterCompletion():*** Indicates whether to rollback.

Code for Designing Transactional Conversations

```
public class CountBean implements SessionBean, SessionSynchronization {

    public int val;
    public int oldVal;

    public void ejbCreate(int val) throws CreateException {
        this.val=val;
        this.oldVal=val;
    }

    public int count() { return ++val; }

    public void afterCompletion(boolean b) { if (b == false) val = oldVal; }

    public void afterBegin() { oldVal = val;}
    public void beforeCompletion() {}
    public void ejbRemove() {}
    public void ejbActivate() {        }
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext ctx) {}
}
```

Dooming Transactions

- As you may know, you can call EJBContext.setRollbackOnly() to force transaction rollback

```
public class MyBean implements javax.ejb.EntityBean {

   private EntityContext ctx  = null;

   public void setEntityContext(EntityContext ctx) {
        this.ctx = ctx;
   }
   ...

   public void foo() throws Exception {

        // something bad happened..
        ctx.setRollbackOnly();
   }
}
```

The problem with dooming transactions

- Doomed transactions decreases scalability
- If transaction is doomed, why perform compute-intensive operations?
- Need a way to detect doomed transactions

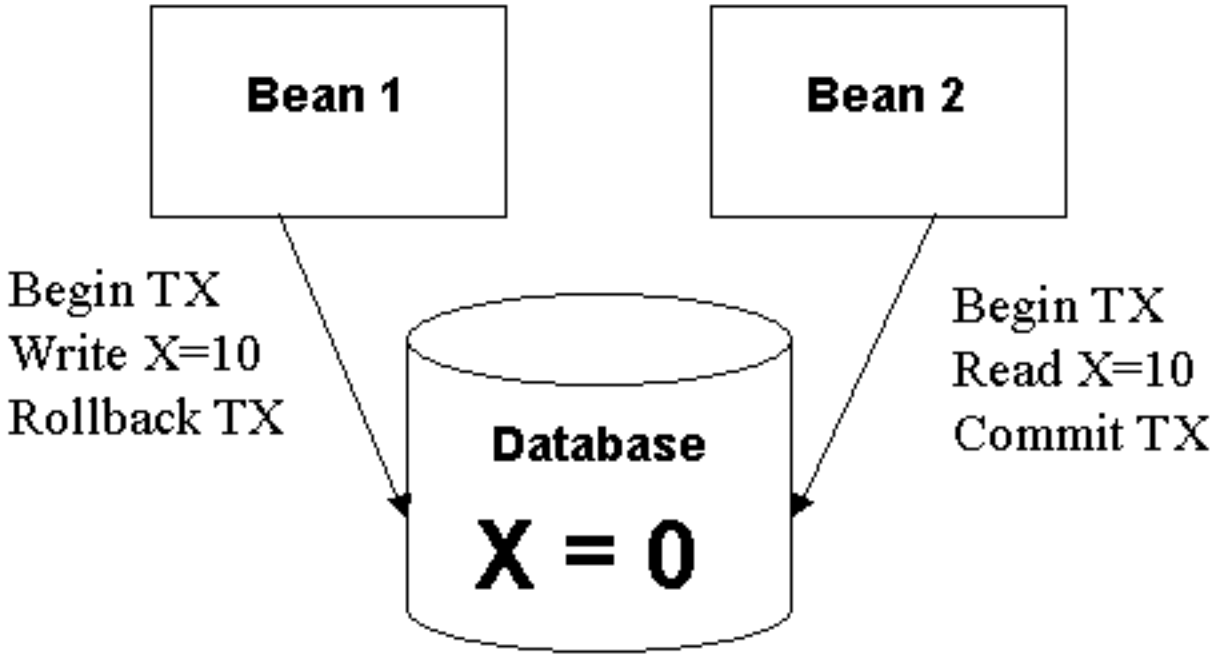
How to detect doomed transactions for scalability

```
public class MyBean implements javax.ejb.EntityBean {

   private EntityContext ctx  = null;

   public void setEntityContext(EntityContext ctx) {
        this.ctx = ctx;
   }

   ...

   public void performComplexOperation() throws Exception {

        if (ctx.getRollbackOnly()) { return; }

        else {
                // perform complex operation
        }
   }
}
```

Isolation - The "I" in ACID

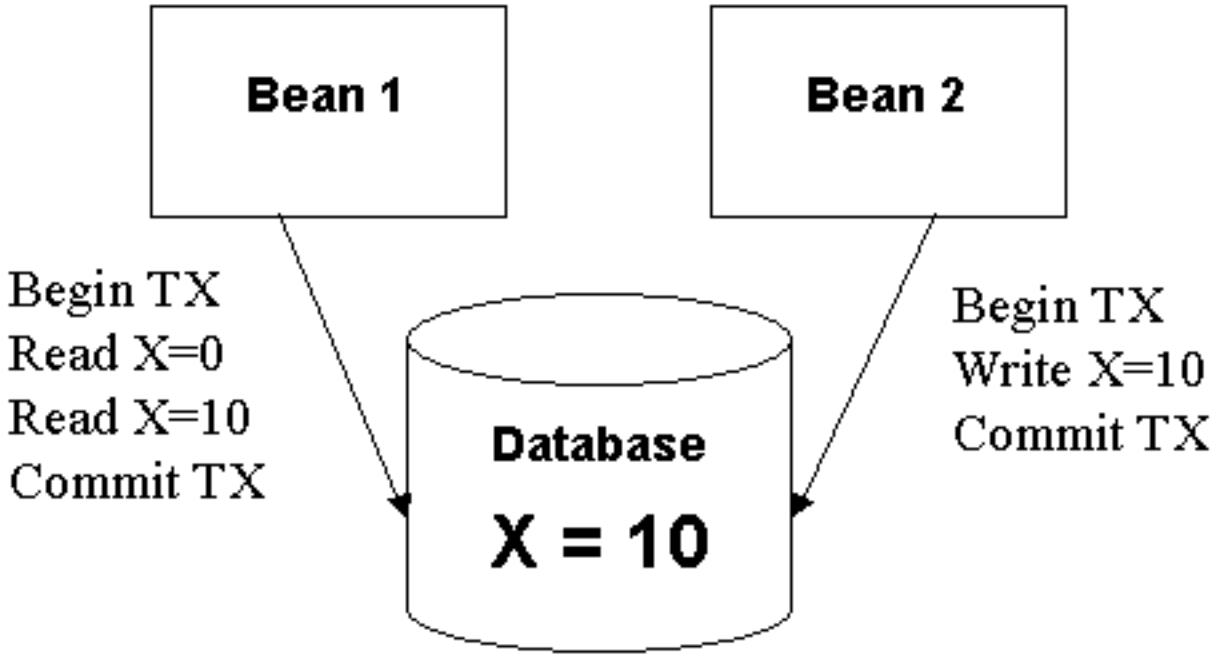
- Problem: Need to share data safely
- Isolation is the tradeoff between concurrency and safety
- EJB 1.0: EJB Isolation Levels put you in control
- EJB 1.1: No EJB isolation levels. Use JDBC.
- Let's discuss how to properly use isolation
- We begin by investigating the types of transactional problems

The Dirty Read Problem



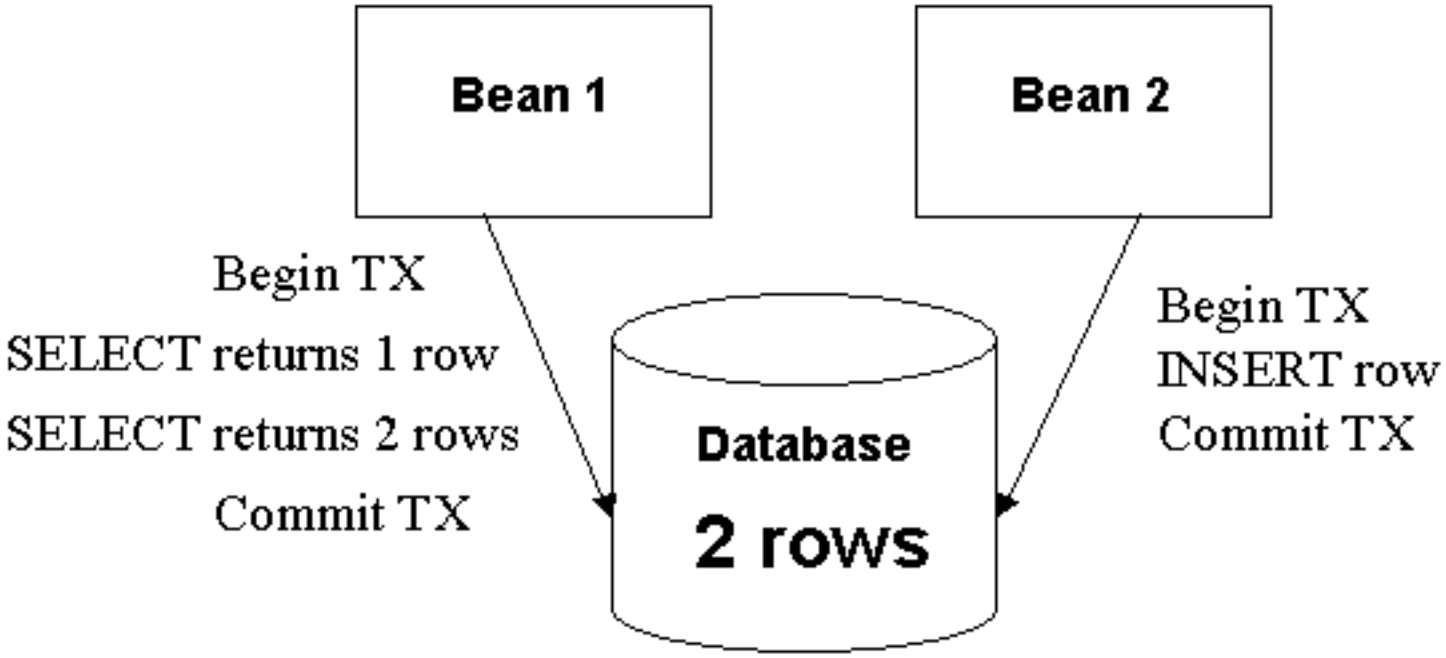
- TRANSACTION_READ_UNCOMMITTED isolation level allows for this problem to happen
- TRANSACTION_READ_COMMITTED isolation level solves this problem

The Unrepeatable Read Problem



- TRANSACTION_READ_COMMITTED isolation level allows for this problem to happen
- TRANSACTION_REPEATABLE_READ isolation level solves this problem

The Phantom Problem



- TRANSACTION_REPEATABLE_READ isolation level allows for this problem to happen
- TRANSACTION_SERIALIZABLE isolation level solves this problem

Isolation Summary

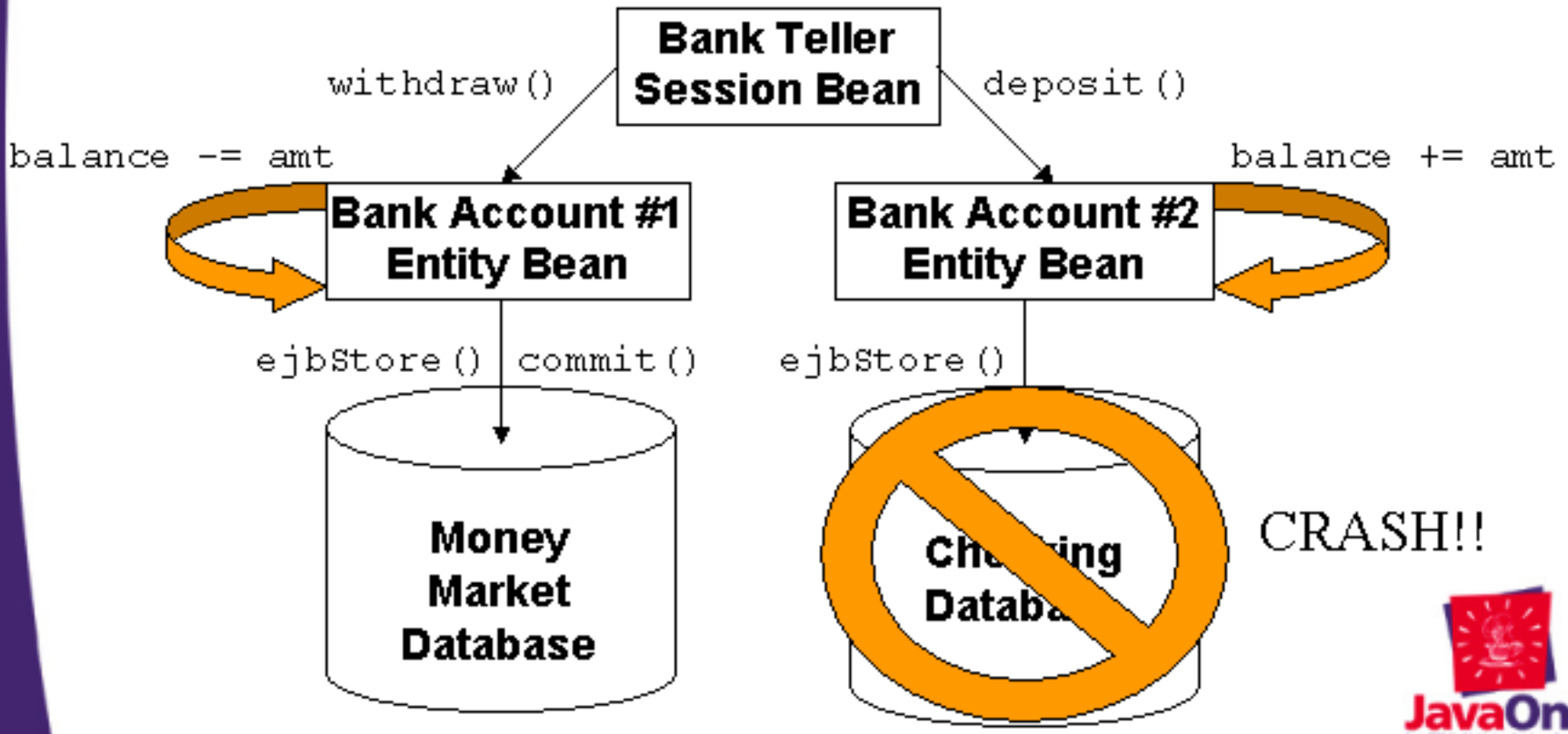
Isolation Level	Dirty Reads?	Unrepeatable Reads?	Phantom Reads?
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Isolation Summary

- **READ_UNCOMMITTED (fastest)**
  - No data sharing, non mission-critical apps.
- **READ_COMMITTED (fast)**
  - Rough Reporting tools. Data should be committed. Only need a snapshot.
- **REPEATABLE_READ (medium)**
  - Mission-critical, high data sharing. You have not performed a query, so phantoms are OK, so long as the data you're working with isn't modified.
- **SERIALIZABLE (slow)**
  - Mission-critical, high data sharing. You have performed a query, and you'd like to know about any new phantom rows.

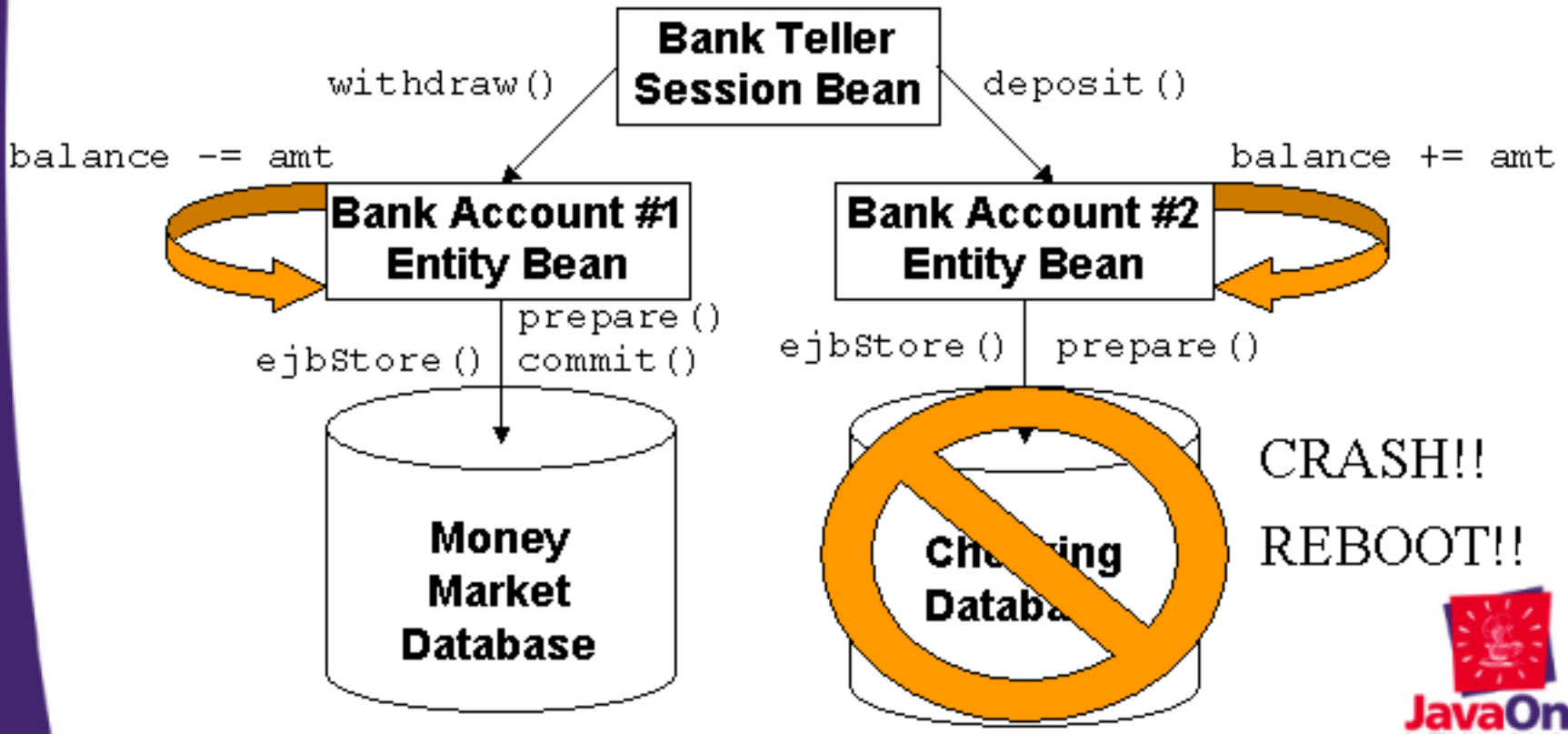
Distributed Transactions

- **Motivation: Have 2 data sources in 1 transaction**
  - Example: Database and Message Queue
- **Normal commit won't work.**



The 2-phase commit solution

- Idea: split commit into 2 phases
  - Phase 1 – Database operations are complete
  - Phase 2 – Commit or rollback operations



Distributed Transactions

- **Server-side only**
- **Keep them short**
- **Choose the optimal isolation level**
- **Avoid 2PC if possible**
- **Use declarative transactions when possible**
- **No on-the-wire transaction context propagation**
  - Means no interoperability until EJB 2.0 is adopted

More information on transactions

- **Jim Gray's Transaction Processing book**
- **Ullman's classic database book**
- **EJB Specification**
- **TheServerSide.com**
  - An open EJB/J2EE discussion community
  - Free electronic copy of my EJB book for download
    - There's more about transactions in that book

Part 2 Preview: EJB Design Patterns

- Properly using sessions and entities
- Value objects
- Many-many relationships
- Unique primary key generation tactics
- Networked singletons
- Lazy-loading beans
- UML aggregation vs. composition
- Entity bean database synchronization
- Dependent objects

Questions

15 minute break

Part II: EJB Design Patterns

Speaker:

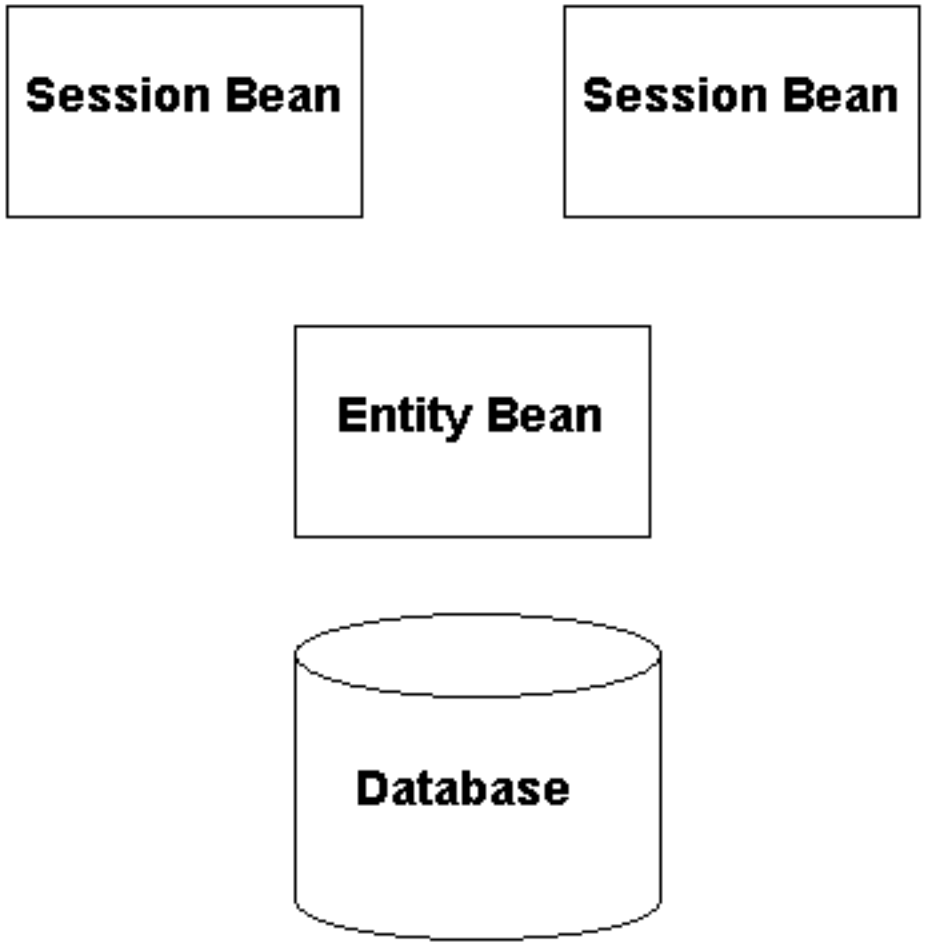
Ed Roman

CEO, The Middleware Company

An EJB Training and Consulting Company

http://www.middleware-company.com

Pattern I
Group discussion: Why wrap entities with sessions?



Pattern II: Value objects

- **Motivation:**
  - You have a three-tier deployment
  - You don't want your web server to hit the application server whenever you need data
- **Solution:**
  - Marshal the entity bean state out of the bean, into a Java class, and pass it across the network
  - Can access the 'local' class faster than entity bean

Your basic value object code

```
public class ProductInfo implements java.io.Serializable {
    public String SKU;
    public String description;
}

public class Product implements javax.ejb.EntityBean {

     public String SKU;
     public String description;
     ...

    public ProductInfo getProductInfo() {
          ProductInfo info = new ProductInfo();
          info.SKU = this.SKU;
          info.description = this.description;
          return info;
    }

    public void setProductInfo(ProductInfo info) {
          this.SKU = info.SKU;
          this.description = info.description;
    }
}
```

A problem with value objects is duplication of code!

```
public class ProductInfo implements java.io.Serializable {
    public String SKU;
    public String description;
}

public class Product implements javax.ejb.EntityBean {

     public String SKU;
     public String description;
     ...

    public ProductInfo getProductInfo() {
          ProductInfo info = new ProductInfo();
          info.SKU = this.SKU;
          info.description = this.description;
          return info;
    }

    public void setProductInfo(ProductInfo info) {
          this.SKU = info.SKU;
          this.description = info.description;
    }
}
```

Solution: use inheritance

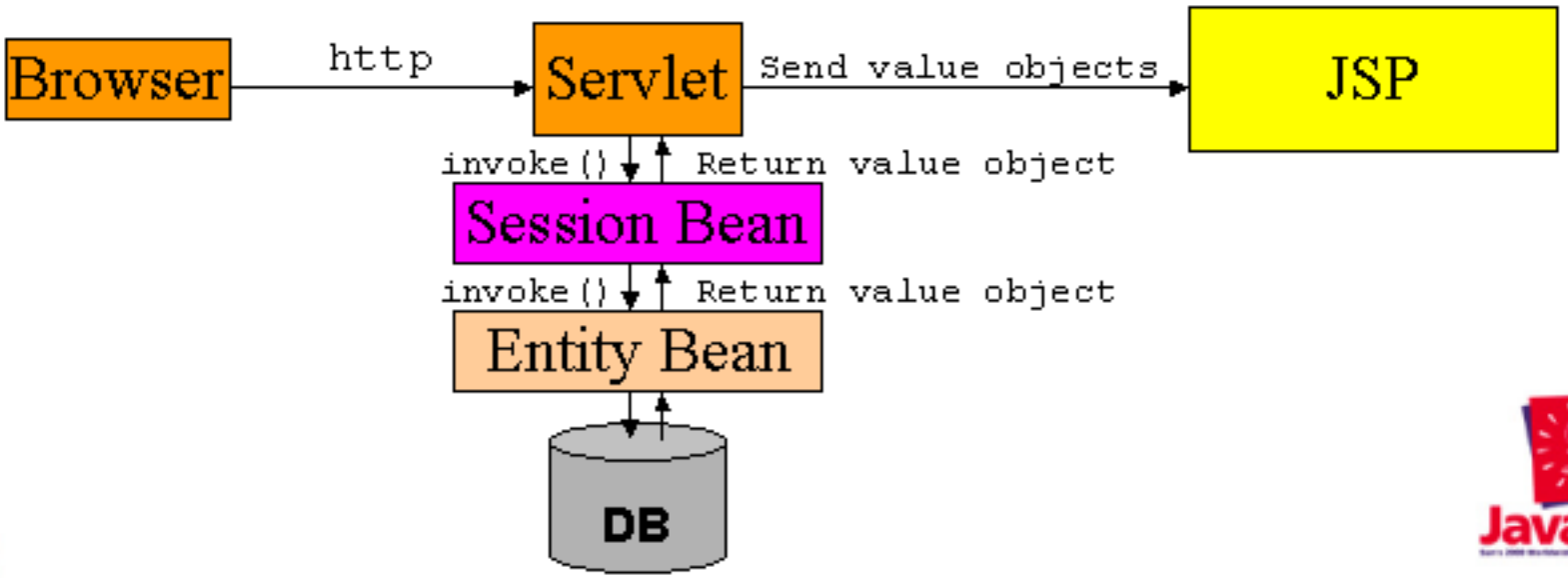
```
public class ProductInfo implements java.io.Serializable {
    public String SKU;
    public String description;
}

public class Product extends ProductInfo implements javax.ejb.EntityBean
    {

    // All my fields are inherited from the value object!

    public ProductInfo getProductInfo() {
          ProductInfo info = new ProductInfo();
          info.SKU = this.SKU;
          info.description = this.description;
          return info;
    }

    public void setProductInfo(ProductInfo info) {
          this.SKU = info.SKU;
          this.description = info.description;
    }
}
```

Another problem value objects solve is input validation

- **What is Input validation?**
  - Checking the user's zipcode
  - Could be submitted over the web, applet, XML stream, or any other way
- **Problem: where do I validate?**
  - If I validate in the web server, I lose validation if I attach a different type of client
- **Solution: put validation in the value object's get/set methods**
  - Entity bean inherits these methods and gets validation checking
  - Value objects get marshaled to web server so checking can happen there too

JSP: yet another use of value objects

- **The problem:**
  - How do I access my database data from a JSP?
  - I want my non-programming web designer to maintain the JSP
- **Solution: use value objects**
  - Marshal the value objects from servlet to JSP
  - Value objects conform to JavaBeans spec due to get/set methods
  - Web designer can learn to access JavaBeans via tags



One problem with this pattern

- **Container-managed entity bean fields are public**
- **So EJB server can inspect fields**
- **Therefore, the value object fields are public**
- **Not so nice. Violates encapsulation.**

```
public class ProductInfo implements java.io.Serializable {
   public String SKU;
   public String description;
}

public class Product extends ProductInfo implements
   javax.ejb.EntityBean {

   // All my fields are inherited from the value object!

   ...
}
```

Pattern III: Many-many relationships

- **Motivation: You have two entity beans**
  - Entity Bean 1: Course (models a course a student is enrolled in)
  - Entity Bean 2: Student (models a student in a college)
  - Inherent Many-to-many relationship
- **How do we implement a query for "find all the courses that a particular student is in"?**
  - Mechanism #1: Add findByStudent() on Course home.
    - Not a good solution, since student is not a dependent on Course.
  - Mechanism #2: Write session bean to perform the query via JDBC.
    - Not a good solution, since it invalidates caching, and adds extra complexity.
  - Mechanism #3: Make a new entity bean, called Enrollment.
    - Good solution. The Enrollment models the JOIN relationship between students and courses

Pattern IV: Unique Primary Key Generation Tactics

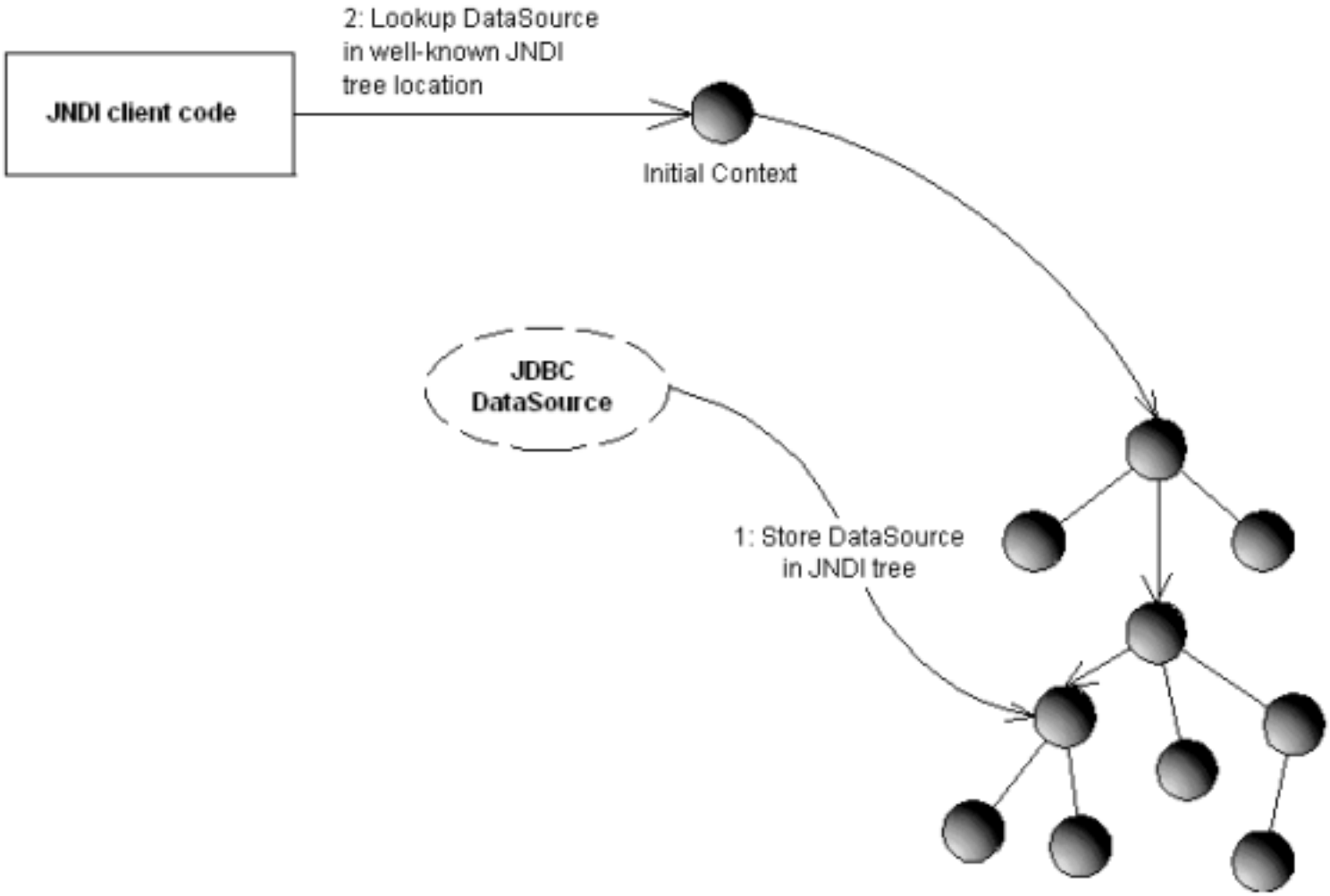
- **Mechanism #1: Use a static in bean**
  - Unfortunately, can't use statics in EJB. EJB 2.0 will via home objects.
- **Mechanism #2: Use your database's build-in counter**
  - Proprietary, non-portable, not always there
- **Mechanism #3: Make an Entity Bean that increments its value**
  - Requires serializable transactions (slow)
- **Mechanism #4: Networked RMI object singleton**
  - Requires a JNDI lookup and a networked call. Also, if you start incrementing at the current time, then daylight savings time will kill you.
- **Mechanism #5: Use a global time service (such as CORBA time service)**
  - Requires native code. Makes your beans OS-dependent. Must be careful not to generate two primary keys in the same millisecond.
- **Mechanism #6: GUIDs in Java**
  - Challenge is to find an algorithm in Java. Java cannot access your network card's MAC address, nor is System.currentTimeMillis() precise enough.
- **Mechanism #7: GUIDs in native code**
  - Using JNI makes your beans OS-dependent.

The RMI-based Unique Key Generator

- **No statics in EJB**
  - Need to still have singleton, but networked singleton instead
- **Solution: Use JNDI and RMI together**
  - Bind RMI remote object to JNDI tree via RMI registry JNDI SPI
  - Clients lookup singleton via JNDI

```
public class uniqueGen extends UnicastRemoteObject
   implements Remote {

   private static long uniqueID = System.currentTimeMillis();

   public static synchronized long getUnique()
   throws RemoteException {
        return uniqueID++;
   }
}
```

Looking up the unique key generator



Pattern V: Lazy-loading beans

- **Assume we have a graph of related entity beans**
  - An Order has a Customer, which has an Address, ...
- **Problem: ejbLoad() on Order Bean loads the whole graph**
  - But you might not need the whole graph!
- **Solution: Lazy-load your beans**
  - Idea: load your bean 'just in time' when someone really wants it

How to lazy-load beans manually

```
public class OrderBean implements javax.ejb.EntityBean {

public Customer customer;

// Load nothing in these two methods
public void ejbLoad() {}
public void ejbStore {}

public void getCustomerName{} {
    loadCustomer();
    return customer.getName();
}

// Perform a JIT lazy load of Customer
private void loadCustomer() {
    if (customer != null) {
          Context initCtx = new
          InitialContext(ctx.getEnvironment());
          home = (CustomerHome) initCtx.lookup("CustomerHome");
          customer = home.findByOrder(orderID);
    }
}
}
```

How to lazy-load beans automatically

- **Hopefully, a future EJB spec will standardize on lazy-loading semantics**
- **For now, Rickard Oberg, an EJB zealot and a friend of mine, wrote a package called SmartProxies**
  - This 'smart stub' wraps the real stub
  - Smart stub loads the object just-in-time when it is accessed
- **Check it out at:**

www-und.ida.liu.se/~ricob684/java/smartproxy/docs/index.html

Pattern VI: Understanding UML aggregation vs. composition

- **Composition (strong) relationship: entity bean A owns entity bean B.**
  - Deleing A deletes B. Shared lifecycle.
- **Aggregation (weak) relationship: entity bean A uses entity bean B.**
  - Deleting A does not delete B. Separate lifecycles.

UML composition code

```
public class OrderBean implements EntityBean {

   private EntityContext ctx = null;
   public void setEntityContext(EntityContext ctx) {this.ctx = ctx;}
   ...

// Delete order means manual cascading delete to line-items
public void ejbRemove() throws Exception {
        Context initCtx = new InitialContext(ctx.getEnvironment());
        OrderLineItemHome home = (OrderLineItemHome)
        initCtx.lookup("OrderLineItemHome");
        Enumeration e = home.findByOrder(orderID);
        while (e.hasMoreElements()) {
                OrderLineItem li = (OrderLineItem) e.nextElement();
                li.remove();
        }
}
```

UML aggregation code

```
public class StudentBean implements EntityBean {
       Aggregation

   ...

// Do not delete the Course this student was enrolled in
public void ejbRemove() throws RemoveException {}
}
```

Pattern VII: Optimizing store operations

- **Problem: ejbStore() is expensive. And not always necessary.**
- **Workaround #1:**
  - Your bean tells the application server whether it's been modified
  - For example, BEA WebLogic has this - write an isModified() method that returns true or false
  - Disadvantage: You need to write this code yourself
- **Workaround #2:**
  - EJB server inspects container-managed fields and makes the determination
  - Disadvantage: Doesn't work w/ bean-managed persistence (fields are private)

Pattern VIII: Dependent Object

One solution for fine-grained data:

Use regular Java objects (also called dependent objects):

```
public class OrderBean implements javax.ejb.EntityBean {
    // container-managed vector of OrderLineItem objects
public Vector orderLineItems;

...
}

public class OrderLineItem {
   public String productName;
   public long basePrice;
   public int quantity;
}
```

Pattern VIII: Dependent Object

One solution for fine-grained data:

Use regular Java objects (also called dependent objects):

```
public class OrderBean implements javax.ejb.EntityBean {
    // container-managed vector of OrderLineItem objects
public Vector orderLineItems;

...
}

public class OrderLineItem {
   public String productName;
   public long basePrice;
   public int quantity;
}
```

Pattern VIII: Dependent Object

- **Better solution: What if a deployment descriptor flag marked a bean as a 'local' bean or a 'remote' bean?**
  - Local beans are accessed directly. No RMI, serialization of parameters, or middleware
  - Remote beans are accessed in a heavyweight fashion as usual
  - Could demarcate such differences in the deployment descriptor
  - Easy to 'flip a switch' to disengage local beans
  - Maybe this will come in the future…

For more information…

- http://www.TheServerSide.com has an EJB design patterns repository
- Check out the wiki web patterns repository at http://www.c2.com

Did you like this talk? How would you like to give it yourself?

- All employees of The Middleware Company are speakers.
- That's one value of working with us – we offer smart developers the opportunity to become recognized experts
- This is part of our on-site training course offering:
  - Java
  - EJB
  - J2EE
  - XML
- We also 'jumpstart' projects. Get them off the ground quickly and reliably with architecture and mentoring
- Custom consulting available
- See us at http://www.middleware-company.com
- Talk to us at info@middleware-company.com

Questions

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠