Using Apache OJB (to get Beer)

PhillyJUG

March 30, 2004

Brian McCallister

Transparent Object/Relational Mapping

- Persists Plain Old Java Objects
- Tracks What You Change For You
- Works in Any Container
- Manages Database Resources
- Caches Intelligently
- Is Hard to do Well

Apache OJB

(It does Transparent O/R Mapping)

OJB Client Interfaces

- Persistence Broker (PB)
- ODMG
- Object Transaction Manager (OTM)
- JDO

OJB Client Interfaces

- Persistence Broker
- ODMG
- Object Transaction Manager
- JDO

Persistence Broker

- Lowest Level
- Most Flexible
- Query By Criteria
- Query by SQL
- Everything Else is Implemented on PB
- Simplest API

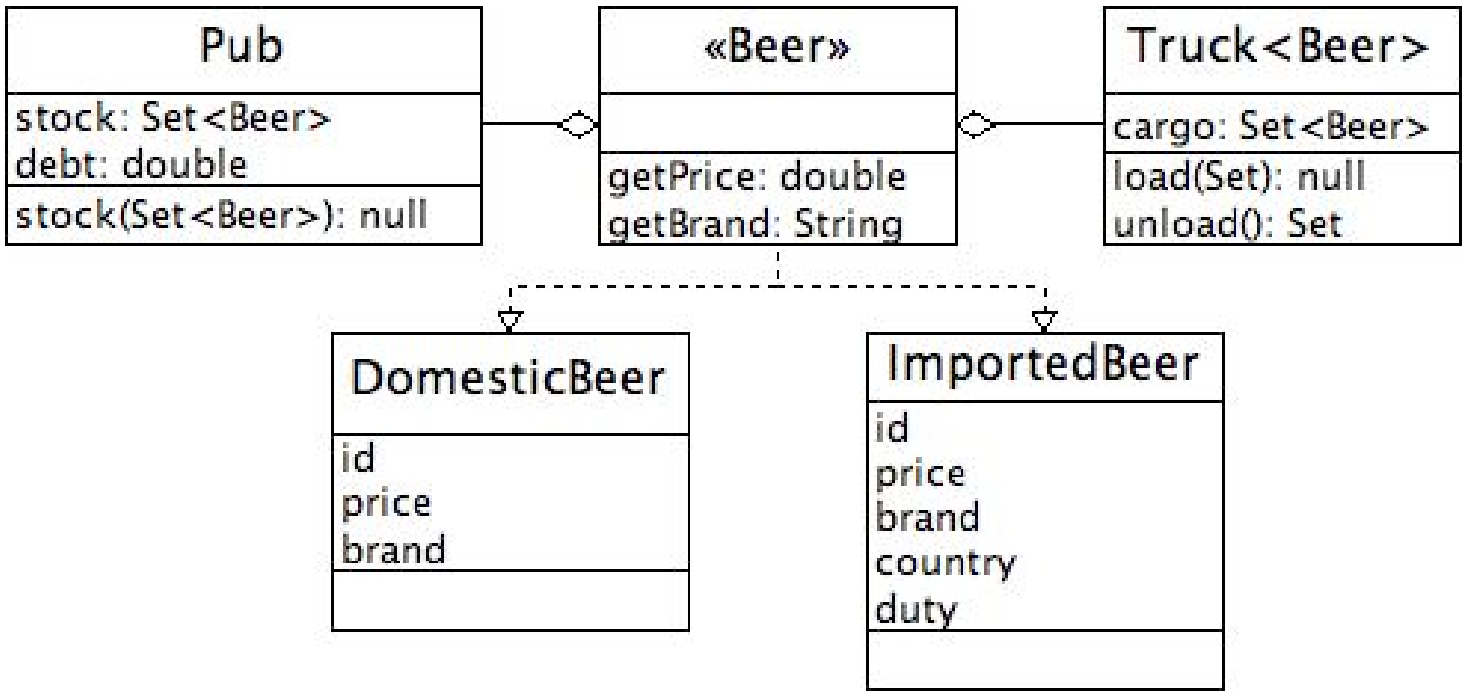
Object Transaction Manager

- Object Transactions
- Object Locking
- Automatic Dirtying
- Query By Criteria
- Query by OQL

Beer

(Let’s see some code)

Beer Delivery Domain Model



CRUD

(literally, turns out the pub has no beer)

PersistenceBroker - Create

```
public static Pub makeMyPub(String name) {
   Pub pub = new Pub(name);
   PersistenceBroker broker =
       PersistenceBrokerFactory.createPersistenceBroker(key);
   broker.store(pub);
   broker.close();
   return pub;
}
```

PersistenceBroker - Create

```
public static Pub makeMyPub(String name) {
   Pub pub = new Pub(name);
   PersistenceBroker broker =
        PersistenceBrokerFactory.createPersistenceBroker(key);
   broker.store(pub);
   broker.close();
   return pub;
}
```

PersistenceBroker - Create

```
public static Pub makeMyPub(String name) {
   Pub pub = new Pub(name);
   PersistenceBroker broker =
       PersistenceBrokerFactory.createPersistenceBroker(key);
   broker.store(pub);
   broker.close();
   return pub;
}
```

OTM - Create

```
public static Truck buyTheTruck() throws LockingException {
   Truck<Beer> truck = new Truck<Beer>();
   OTMConnection conn=kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   conn.makePersistent(truck);
   tx.commit();
   conn.close();
   return truck;
}
```

OTM - Create

```
public static Truck buyTheTruck() throws LockingException {
   Truck<Beer> truck = new Truck<Beer>();
   OTMConnection conn=kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   conn.makePersistent(truck);
   tx.commit();
   conn.close();
   return truck;
}
```

OTM - Create

```
public static Truck buyTheTruck() throws LockingException {
   Truck<Beer> truck = new Truck<Beer>();
   OTMConnection conn=kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
  tx.begin();
  conn.makePersistent(truck);
  tx.commit();
  conn.close();
  return truck;
}
```

OTM - Create Dependent Objects

```
public static void fillHerUp(Truck truck) throws
       LockingException {
   Beer schlitz = new DomesticBeer();
   Beer more_schlitz = new DomesticBeer();
   Beer good_beer = new ImportedBeer("Caffreys 70");
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   Identity id = conn.getIdentity(truck);
   Truck tx_truck = (Truck) conn.getObjectByIdentity(id);
   tx_truck.load(new Beer[]{schlitz,
                            more_schlitz,
                            good_beer});
   tx.commit();
   conn.close();
}
```

OTM - Create Dependent Objects

```
public static void fillHerUp(Truck truck) throws
      LockingException {
   Beer schlitz = new DomesticBeer();
   Beer more_schlitz = new DomesticBeer();
   Beer good_beer = new ImportedBeer("Caffreys 70");
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   Identity id = conn.getIdentity(truck);
   Truck tx_truck = (Truck) conn.getObjectByIdentity(id);
   tx_truck.load(new Beer[]{schlitz,
                            more_schlitz,
                            good_beer});
   tx.commit();
   conn.close();
}
```

OTM - Create Dependent Objects

```
public static void fillHerUp(Truck truck) throws
        LockingException {
    Beer schlitz = new DomesticBeer();
    Beer more_schlitz = new DomesticBeer();
    Beer good_beer = new ImportedBeer("Caffreys 70");
    OTMConnection conn = kit.acquireConnection(key);
    Transaction tx = kit.getTransaction(conn);
    tx.begin();
    Identity id = conn.getIdentity(truck);
    Truck tx_truck = (Truck) conn.getObjectByIdentity(id);
    tx_truck.load(new Beer[]{schlitz,
                             more_schlitz,
                             good_beer});
    tx.commit();
    conn.close();
}
```

Truck.java

```
public class Truck <Cargo> {
    private Integer id;
    private Set<Cargo> cargo = new HashSet<Cargo>();

    public void load(Cargo[] items) {
        for (int i = 0; i < items.length; i++) {
            this.cargo.add(items[i]);
        }
    }
    public Set<Cargo> unload() {
        HashSet<Cargo> set = null;
        synchronized (cargo) {
            set = new HashSet<Cargo>(this.cargo);
            this.cargo.clear();
        }
        return set;
    }
}
```

Truck.java

```
public class Truck <Cargo> {
    private Integer id;
    private Set<Cargo> cargo = new HashSet<Cargo>();

    public void load(Cargo[] items) {
        for (int i = 0; i < items.length; i++) {
            this.cargo.add(items[i]);
        }
    }
    public Set<Cargo> unload() {
        HashSet<Cargo> set = null;
        synchronized (cargo) {
            set = new HashSet<Cargo>(this.cargo);
            this.cargo.clear();
        }
        return set;
    }
}
```

OTM - Read

```
public static Collection findTheCaffreys() {
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   Criteria criteria = new Criteria();
   criteria.addLike("brand", "Caffreys%");
   Query query = QueryFactory.newQuery(Beer.class,criteria);
   Collection beers = conn.getCollectionByQuery(query);
   tx.commit();
   conn.close();
   return beers;
}
```

OTM - Read

```
public static Collection findTheCaffreys() {
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   Criteria criteria = new Criteria();
   criteria.addLike("brand", "Caffreys%");
   Query query = QueryFactory.newQuery(Beer.class,criteria);
   Collection beers = conn.getCollectionByQuery(query);
   tx.commit();
   conn.close();
   return beers;
}
```

OTM - Read Again

```
public static Iterator findTheCaffreysAgain()
       throws Exception {
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   OQLQuery query = conn.newOQLQuery();
   query.create("select beer from Beer " +
                "where brand like $1");
   query.bind(new Object[] {"Caffreys%"});
   Iterator beers = conn.getIteratorByOQLQuery(query);
   tx.commit();
   conn.close();
   return beers;
}
```

OTM - Read Again

```
public static Iterator findTheCaffreysAgain()
      throws Exception {
   OTMConnection conn = kit.acquireConnection(key);
   Transaction tx = kit.getTransaction(conn);
   tx.begin();
   OQLQuery query = conn.newOQLQuery();
   query.create("select beer from Beer " +
                "where brand like $1");
   query.bind(new Object[] {"Caffreys%"});
   Iterator beers = conn.getIteratorByOQLQuery(query);
   tx.commit();
   conn.close();
   return beers;
}
```

Persistence Broker - Read

```
public static Collection findTheSchlitz() {
   PersistenceBroker broker =
      PersistenceBrokerFactory.createPersistenceBroker(key);
   Criteria criteria = new Criteria();
   criteria.addLike("brand", "Schlitz");
   Query query = QueryFactory.newQuery(Beer.class,
                                       criteria);
   Collection beers = broker.getCollectionByQuery(query);
   broker.close();
   return beers;
}
```

Persistence Broker - Read

```
public static Collection findTheSchlitz() {
   PersistenceBroker broker =
      PersistenceBrokerFactory.createPersistenceBroker(key);
   Criteria criteria = new Criteria();
   criteria.addLike("brand", "Schlitz");
   Query query = QueryFactory.newQuery(Beer.class,
                                       criteria);
   Collection beers = broker.getCollectionByQuery(query);
   broker.close();
   return beers;
}
```

OTM - Update

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,
                                       LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

OTM - Update

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,
                                       LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

OTM - Update

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,
                                   LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

PersistenceBroker - Delete

```
public static void junkThePubs() {
   PersistenceBroker broker =
   PersistenceBrokerFactory.createPersistenceBroker(key);
   Query query = QueryFactory.newQuery(Truck.class,
                                       new Criteria());
   broker.beginTransaction();
   broker.deleteByQuery(query);
   broker.commitTransaction();
   broker.close();
}
```

PersistenceBroker - Delete

```
public static void junkThePubs() {
   PersistenceBroker broker =
   PersistenceBrokerFactory.createPersistenceBroker(key);
   Query query = QueryFactory.newQuery(Truck.class,
                                       new Criteria());
   broker.beginTransaction();
   broker.deleteByQuery(query);
   broker.commitTransaction();
   broker.close();
}
```

The Pub Has Beer

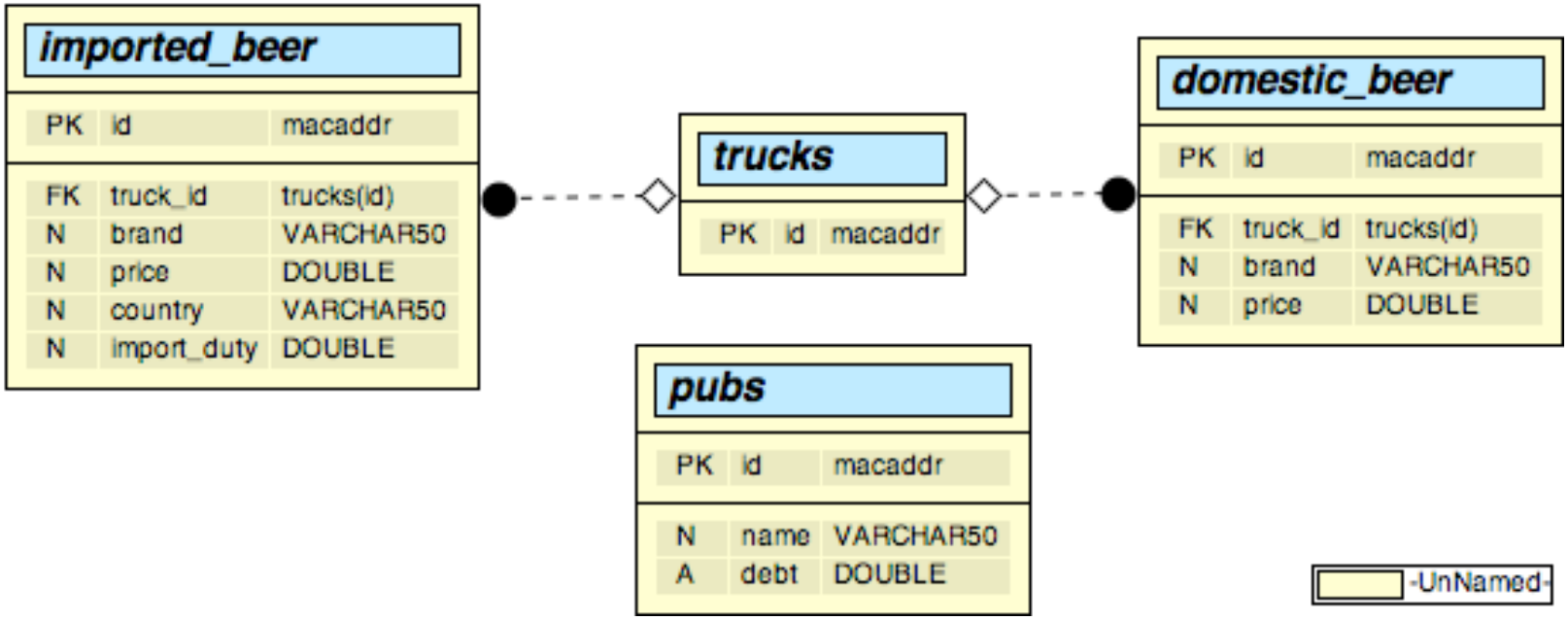
(but we only care about the delivery)

(well, and the beer)

How This All Works

(Mapping)

A Schema



(It's not well normalized, this is a demo. Thomas normalized it better for his demo. His model would work just as well for our objects though -- bug me for the mapping in Q/A if it is important to you. ;-)

Simple Properties

```
<class-descriptor class="DomesticBeer”
                  table="DOMESTIC_BEER">
  <field-descriptor name="id”
                    column="ID"
                    primarykey="true"
                    autoincrement="true” />
  <field-descriptor name="brand”
                    column="BRAND"
                    length="50” />
  <field-descriptor name="price” column="PRICE"/>

  <field-descriptor name="TRUCK_ID”
                    column="TRUCK_ID"
                    access="anonymous"/>
</class-descriptor>
```

DomesticBeer.java

```
public class DomesticBeer implements Beer {
   private Integer id;
   private String brand;
   private Double price;

   public DomesticBeer() {
       this("Schlitz", 0.50);
   }
   public double getPrice() {
       return price.doubleValue();
   }
   public String getBrand() { return brand; }
}
```

DomesticBeer.java

```
public class DomesticBeer implements Beer {
   private Integer id;
   private String brand;
   private Double price;

   public DomesticBeer() {
       this("Schlitz", 0.50);
   }
   public double getPrice() {
       return price.doubleValue();
   }
   public String getBrand() { return brand; }
}
```

Collections

```
<class-descriptor class="Truck" table="TRUCKS">
   <field-descriptor name="id"
                     column="ID"
                     primarykey="true"
                     autoincrement="true"/>
   <collection-descriptor
             name="cargo"
             element-class-ref="Beer"
             otm-dependent="true">
        <inverse-foreignkey field-ref="TRUCK_ID"/>
   </collection-descriptor>
</class-descriptor>
```

Collections

```
<class-descriptor class="Truck" table="TRUCKS">
  <field-descriptor name="id"
                    column="ID"
                    primarykey="true"
                    autoincrement="true"/>
  <collection-descriptor
            name="cargo"
            element-class-ref="Beer"
            otm-dependent="true">
      <inverse-foreignkey field-ref="TRUCK_ID"/>
  </collection-descriptor>
</class-descriptor>
```

Anonymous Fields

```
<class-descriptor class="DomesticBeer”
                  table="DOMESTIC_BEER">
  <field-descriptor name="id”
                    column="ID"
                    primarykey="true"
                    autoincrement="true” />
  <field-descriptor name="TRUCK_ID”
                    column="TRUCK_ID"
                    access="anonymous"/>
  <field-descriptor name="brand”
                    column="BRAND"
                    length="50” />
  <field-descriptor name="price” column="PRICE"/>
</class-descriptor>
```

Anonymous Fields

```
<class-descriptor class="DomesticBeer"
                  table="DOMESTIC_BEER">
  <field-descriptor name="id"
                    column="ID"
                    primarykey="true"
                    autoincrement="true" />
  <field-descriptor name="TRUCK_ID"
                    column="TRUCK_ID"
                    access="anonymous"/>
  <field-descriptor name="brand"
                    column="BRAND"
                    length="50" />
  <field-descriptor name="price" column="PRICE"/>
</class-descriptor>
```

DomesticBeer.java

```
public class DomesticBeer implements Beer {
   private Integer id;
   private String brand;
   private Double price;

   public DomesticBeer() {
       this("Schlitz", 0.50);
   }
   public double getPrice() {
       return price.doubleValue();
   }
   public String getBrand() { return brand; }
}
```

(notice that there is no TRUCK_ID field)

Extents & Polymorphism
Beer.java

```
public interface Beer {

        public double getPrice();
        public String getBrand();

        /* bmc - removed, too controversial
         * public boolean lessFilling();
         * public boolean tastesGreat();
         */
}
```

Extents & Polymorphism Mapping

```
<class-descriptor class="Beer">
   <extent-class class-ref="DomesticBeer"/>
   <extent-class class-ref="ImportedBeer"/>
</class-descriptor>
```

Transactions

(the data oriented kind)

Database Transactions

- OJB Uses Database Transactions
- PersistenceBroker Tx == RDBMS Tx
- Manually Demarcated
  - startTransaction()
  - commitTransaction()
  - rollbackTransaction()
  - checkPointTransaction()

JTA Transactions

- OJB Uses JTA Transactions
- PersistenceBroker Tx == JTA Tx
- Uses Containers Transaction Demarcation
  - Treat just like JTA anywhere else

Object Transactions

- Implemented in OJB
- (Object Transaction Manager)
- Wraps PersistenceBroker Transaction
- Provides Object Locking
- Even Across different JVM's!
- Automatic Dirtying
- Manually Demarcated
- Even provides rollbacks on objects!

When to Use OJB

(when not to as well)

When to Use OJB…

- With a Domain Model
- Complex Mapping Requirements
- In J2SE Applications
- In Servlet Applications
- In EJB Applications
- When you want to be more productive
- When you want to be more flexible

When Not to Use OJB

- Really Simple Applications
  - Commons-DbUtils, iBatis, JDBC
- Tabular Reporting Applications
  - Table -> Object -> Table (hmmm)
- Just Want Direct Data -> Object
  - Apache Torque

How to Use OJB

(good practices)

Caching

- Everyone Underestimates Caching
- Single App, Dedicated Database
- Clustered App, Dedicated Database
- Multiple Apps, Shared Cache
- Multiple Apps, Cannot Share Cache
  - Database as Integration Layer

Caching - Queries and Cache

- Query By Identity
  - Can draw from cache

Query By Identity

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,
   LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

Caching - Queries and Cache

- Query By Identity
  - Can draw from cache
- Query By Criteria (or OQL)
  - Must Hit Database
  - Can Avoid Materializing if Object Cached

Query By Criteria

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,

   LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

Caching - Queries and Cache

- Query By Identity
  - Can draw from cache
- Query By Criteria (or OQL)
  - Must Hit Database
  - Can Avoid Materializing if Object Cached
- Choose Cache Implementation Carefully!
  - **Big** Performance Boost from Good Caching
  - Avoid Dirty Data in Cache

Modify Transactional Objects

1. You have an Object (Pub)
2. Start a transaction
3. Query by Identity for the Same Object
4. Modify the Transactional Object
5. Commit the Transaction

(same thing in Hibernate)

Update Transactional Objects

```
tx.begin();
Query query = QueryFactory.newQuery(Truck.class,
                                    new Criteria());
Collection all_trucks = conn.getCollectionByQuery(query,
                                   LockType.WRITE_LOCK);

Identity pub_id = conn.getIdentity(pub);
Pub tx_pub = (Pub)conn.getObjectByIdentity(pub_id,
   LockType.WRITE_LOCK);

Iterator itty = all_trucks.iterator();
while(itty .hasNext()) {
   Truck<Beer> truck = (Truck<Beer>) itty.next();
   tx_pub.addStock(truck.unload());
}
tx.commit();
```

Thread Local Transactions

- Store Connection in a ThreadLocal
- Use J2EE Transaction Rules in J2SE
- Allows CMT in J2SE

(ODMG Does This By Default)

(Spring Does This **Really** Nicely)

Gotchas

- Fields are Queried, not Properties

ImportedBeer.java

```
public class ImportedBeer implements Beer {
   private Double price;
   private Double duty;

   public double getPrice() {
       return price.doubleValue() + duty.doubleValue();
   }

   /* More Beer Implementation Stuff */
}
```

ImportedBeer.java

```
public class ImportedBeer implements Beer {
   private Double price;
   private Double duty;

   public double getPrice() {
        return price.doubleValue() + duty.doubleValue();
   }

   /* More Beer Implementation Stuff */
}
```

Field

Property

Gotchas

- Fields are Queried, not Properties
- Watch Your Caching
- Entity Identity, Java Identity, Equality
- Benchmark and Profile Proxy Options
- Primary Keys Unique Across Extent
- Use Object wrappers for primitives

Questions?

(I can attempt to answer)

More Information

http://db.apache.org/ojb/

http://kasparov.skife.org/ojb-phillyjug.pdf
http://kasparov.skife.org/ojb-phillyjug.tar.gz

ojb-user-subscribe@db.apache.org

Other Great (Free) Tools

- Apache Torque
  - http://db.apache.org/torque/
- Hibernate
  - http://www.hibernate.org/
- Jakarta Commons-DbUtils
  - http://jakarta.apache.org/commons/dbutils/
- iBatis
  - http://www.ibatis.com/
- TJDO
  - http://tjdo.sourceforge.net/
- Spring Framework
  - http://www.springframework.org/

Thank You!

(Audience, eXceed education, and PhillyJUG)