



Welcome to Hibernate

Object-Relational Mapping for Java

Rod Cope

Agenda

- **Introductions**
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Rod Cope

- Founder and CTO of OpenLogic
 - rod.cope@openlogic.com
- 9 years Java experience
 - Sun Certified Java Architect and Programmer
 - J2SE, J2EE, J2ME, Swing, Security, Open Source, etc.
 - GE, IBM, Ericsson, Manugistics, Digital Thoughts, etc.
- 2 years of Hibernate
- 3 years of BlueGlue
- Writing book for O'Reilly on "Groovy"
- Speaks at No Fluff Just Stuff, JavaOne, O'Reilly Open Source Convention, JUGs, etc.

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Object-Relational Mapping

- Java objects ↔ database tables
- JDBC is tedious and error-prone
- Features of modern ORM facilities
 - Transparent persistence (POJO/JavaBeans)
 - Transitive persistence
 - Persistent/transient instances
 - Automatic dirty checking
 - Inheritance mapping
 - Lazy fetching
 - Outer join fetching
 - Runtime SQL generation
 - Development tools (IDE plug-ins, roundtrip tools, etc.)

Why ORM tools?

- Natural programming model
- Minimize code length
 - Faster to write
 - Easier to read and maintain
- Code can tested and run outside of any containers
- Classes may be reused in non-persistent context
- Minimize database trips with smart fetching strategies
- Opportunities for aggressive caching
- Structural mapping more robust when object/data model changes

Agenda

- Introductions
- Object-Relational Mapping
- **Hibernate**
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Hibernate

- ORM tool created in 2001 by Gavin King
- Open Source, released under LGPL
- Version 2.1.8 is production-ready, 3.0 is in active beta
- Over 15,000 downloads per month
- Persistence engine for JBoss
- Works standalone or with most app servers, databases, caching tools, and other Open Source components
- Extremely good documentation and community support
- Not standards-based (i.e., not a JDO implementation)

Hibernate (cont.)

- Powerful, ultra-high performance object-relational persistence and query service for Java
- Supports inheritance, polymorphism, composition, and Java collections
- Long transactions and optimistic locking
- Runtime bytecode and SQL generation
 - No base classes or interfaces to implement
 - Essentially database independent
- Transparent persistence and detached objects
 - No ORM dependencies
 - No DTO's needed for multi-tiered architecture!

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- **Example**
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Example (Model)

- Simple Auction Model
- An auction has items
- Items have bids
- An item may have a winning bid



Example (Java class)

- Default constructor
- Get/set pairs
- Collection property is an interface type
- Identifier property

```
public class AuctionItem {  
    private Long id;  
    private Set bids;  
    private Bid successfulBid;  
    private String description;  
    public Long getId() {  
        return id;  
    }  
    private void setId(Long id) {  
        this.id = id;  
    }  
    public String getDescription() {  
        return description;  
    }  
    public void setDescription(String desc) {  
        this.description = desc;  
    }  
    ...  
}
```

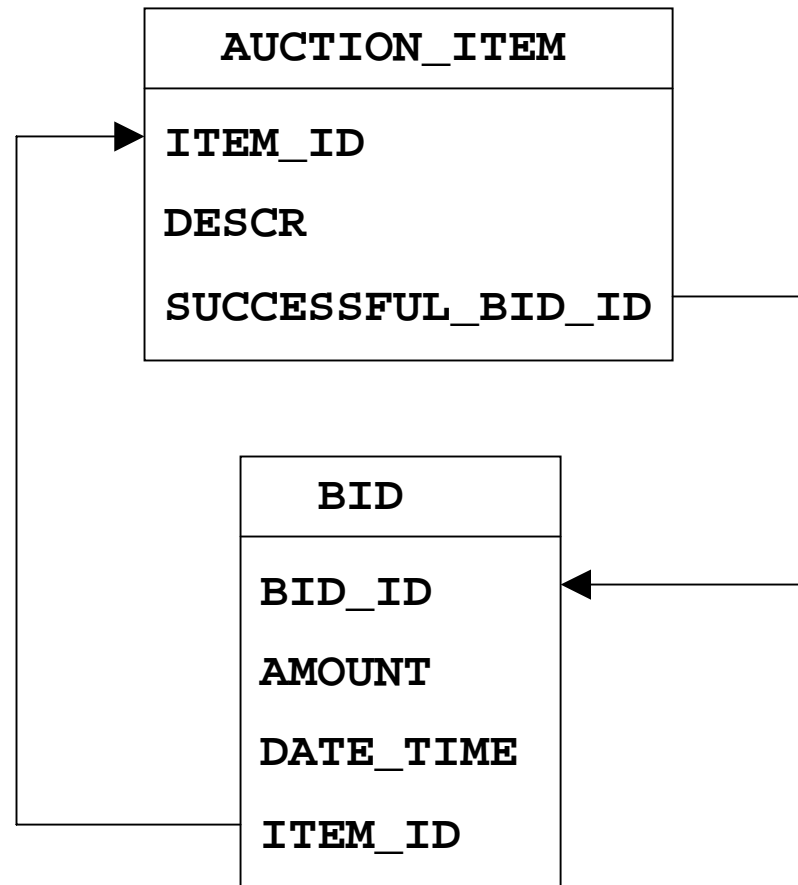
Example (XML Mapping File)

- Readable metadata
- Column / table mappings
- Surrogate key generation strategy
- Collection metadata
- Fetching strategies

```
<class name="AuctionItem" table="AUCTION_ITEM">
  <id name="id" column="ITEM_ID">
    <generator class="native"/>
  </id>
  <property name="description" column="DESCR"/>
  <many-to-one name="successfulBid"
    column="SUCCESSFUL_BID_ID"/>
  <set name="bids" cascade="all" lazy="true">
    <key column="ITEM_ID"/>
    <one-to-many class="Bid"/>
  </set>
</class>
```

Example (Database)

- Straightforward schema
- Simple foreign keys
- Multiple relationships per table
- All data types supported



Example: Automatic Dirty Checking

- Retrieve and update an item

```
Session session = sessionFactory.openSession();
```

```
Transaction tx = session.beginTransaction();
```

```
AuctionItem item =
```

```
    (AuctionItem) session.get(AuctionItem.class, itemId);
```

```
item.setDescription(newDescription);
```

```
tx.commit();
```

```
session.close();
```

Example: Transitive Persistence

- Retrieve an auction item and add a persistent bid to it

```
Bid bid = new Bid();  
bid.setAmount(bidAmount);
```

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();
```

```
AuctionItem item =  
    (AuctionItem) session.get(AuctionItem.class, itemId);
```

```
bid.setItem(item);  
item.getBids().add(bid);
```

```
tx.commit();  
session.close();
```

Example: Detach and Update

- Change an item outside of a transaction, then make the change persistent

```
// initially - in a SessionBean
Session s1 = sessionFactory.openSession();
Transaction tx = s1.beginTransaction();
AuctionItem item = (AuctionItem) s1.get(AuctionItem.class, itemId);
tx.commit();
s1.close();

// later - in a servlet/action
Bid bid = new Bid();
bid.setAmount(bidAmount);
bid.setItem(item);
item.getBids().add(bid);

// later still - back in a SessionBean
Session s2 = sessionFactory.openSession();
Transaction tx2 = s2.beginTransaction();
s2.update(item);
tx2.commit();
s2.close();
```

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Configuration (Mapping Files)

- Initialization code (only runs once per app)

```
config = new Configuration()  
        .addClass(AuctionItem.class)  
        .addClass(Bid.class);  
sessionFactory = config.buildSessionFactory();
```

- Hibernate will look in classpath for these mapping files:
 - AuctionItem.hbm.xml
 - Bid.hbm.xml
- There are other (better) ways to configure Hibernate

Configuration (Hibernate)

- Dozens of configuration parameters in `hibernate.properties`
 - `hibernate.connection.driver_class = org.postgresql.Driver`
 - `hibernate.connection.url = jdbc:postgresql://localhost/db`
 - `hibernate.connection.username = user`
 - `hibernate.connection.password = secret`
 - `hibernate.dialect=net.sf.hibernate.dialect.PostgreSQLDialect`
- Lots more properties for:
 - Using datasources (with application servers)
 - Caching control, outer join control and other performance tweaks
 - Connection limits
 - Transaction management
 - Logging and debugging
- Use `hibernate.cfg.xml` to encapsulate setup from code
 - `sf = new Configuration().configure().buildSessionFactory();`

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- **Hibernate Query Language**
- Advanced Topics
- BlueGlue
- Conclusion

Hibernate Query Language (HQL)

- Like an object-oriented flavor of SQL
 - Classes and properties instead of tables and columns
 - Polymorphism
 - Associations
 - *Much* less verbose than SQL
- Full support for relational operations
 - Inner/outer/full joins, cartesian products
 - Projection
 - Aggregation (max, avg) and grouping
 - Ordering
 - Subqueries
 - SQL function calls

HQL Samples

- The simplest query, "from AuctionItem"
 - Uses automatically-generated SQL via JDBC
 - Could span multiple tables, involve joins, filters, etc.
 - Constructs AuctionItem object for each entry in the ResultSet
- Usage in Java code

```
List allItems = session.find( "from AuctionItem" );
```

or:

```
Iterator iter = session.iterate( "from AuctionItem" );
```

HQL Samples (cont.)

- To get:
 - All AuctionItem's with a description that starts with "hib" and at least one bid over \$100, sorted by highest bid
- Use this query:

```
select item
from AuctionItem item
    join item.bids bid
where item.description like 'hib%'
    and bid.amount > 100
order by bid.amount desc
```

HQL Samples (cont.)

- To get:
 - All Customers currently shopping in a store in either Melbourne or Sydney where every item in their basket is a "widget"
- Use a more interesting query in HQL:

```
select cust
from Product prod,
     Store store inner join store.customers cust
where prod.name = 'widget' and
     store.location.name in ('Melbourne','Sydney') and
     prod = all elements(cust.currentOrder.lineItems)
```

HQL Samples (cont.)

- Or use a nasty query in SQL:

```
SELECT cust.name, cust.address, cust.phone, cust.id,  
       cust.current_order  
FROM customers cust, stores store, locations loc,  
     store_customers sc, product prod  
WHERE prod.name = 'widget' AND  
       store.loc_id = loc.id AND  
       loc.name IN ( 'Melbourne', 'Sydney' ) AND  
       sc.store_id = store.id AND  
       sc.cust_id = cust.id AND  
       prod.id = ALL( SELECT item.prod_id  
                     FROM line_items item, orders o  
                     WHERE item.order_id = o.id AND  
                           cust.current_order = o.id )
```

Criteria Query

- HQL Query

```
from AuctionItem item
    left join fetch item.bids
where item.description like :description and
    item.successfulBid.amount > :minAmount
```

- Criteria Query

```
List auctionItems =
    session.createCriteria(AuctionItem.class)
        .setFetchMode("bids", FetchMode.EAGER)
        .add( Expression.like("description", description) )
        .createCriteria("successfulBid")
            .add( Expression.gt("amount", minAmount) )
        .list();
```

Example Query

- HQL Query

```
from AuctionItem item
    left join fetch item.bids
where item.description like 'hib%' and
    item.successfulBid.amount = 1.0
```

- Example Query

```
AuctionItem item = new AuctionItem();
item.setDescription("hib");
Bid bid = new Bid();
bid.setAmount(1.0);
List auctionItems =
    session.createCriteria(AuctionItem.class)
        .add( Example.create(item).enableLike(MatchMode.START) )
        .createCriteria("successfulBid")
            .add( Example.create(bid) )
        .list();
```

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- **Advanced Topics**
- BlueGlue
- Conclusion

Advanced Topics (Outer Join)

- Outer Join Fetching (Mapping)

```
<class name="AuctionItem" table="AUCTION_ITEM">
  <id name="id" column="ITEM_ID">
    <generator class="native"/>
  </id>
  <property name="description" column="DESC"/>
  <many-to-one name="successfulBid"
    outer-join="true"
    column="SUCCESSFUL_BID_ID"/>
  <set name="bids"
    cascade="all"
    outer-join="true">
    <key column="ITEM_ID"/>
    <one-to-many class="Bid"/>
  </set>
</class>
```

Advanced Topics (Outer Join 2)

- Outer Join Fetching (SQL)

```
AuctionItem item = (AuctionItem)s.get(AuctionItem.class,itemId);
```

```
SELECT ...  
FROM AUCTION_ITEM ITEM  
LEFT OUTER JOIN BID BID1 ON  
    BID1.ITEM_ID = ITEM.ITEM_ID  
LEFT OUTER JOIN BID BID2 ON  
    BID2.BID_ID = ITEM.SUCCESSFUL_BID  
WHERE ITEM.ITEM_ID = ?
```

Advanced Topics (Components)

- **Components**
 - Part of parent object
 - Reside in same relational table
 - Support object-oriented programming

Advanced Topics (Components)

- Address class
 - street, city, and zipCode properties
 - STREET, CITY, and ZIP_CODE columns of PERSON and COMPANY tables
 - Mutable

```
<class name="Person" table="PERSON">
...
  <component name="address">
    <property name="street" column="STREET"/>
    <property name="city" column="CITY"/>
    <property name="zipCode" column="ZIP_CODE"/>
  </component>
</class>
```

Advanced Topics (Native SQL)

- Issue native SQL queries

```
Query sqlQuery = sess.createSQLQuery(
    "select {cat.*} from cats {cat}",
    "cat", Cat.class);
sqlQuery.setMaxResults(50);
List cats = sqlQuery.list();
```

- Parameters to `createSQLQuery`
 - SQL query
 - Table alias (or `String[]` of table aliases)
 - Class to instantiate per row (or `Class[]` of classes)
- Use any SQL your database can handle
 - Still provides benefit of JDBC-free SQL
 - Hibernate automatically creates instances for you

Hibernate Tools

- **Ant**
 - Tasks available for most tools below
- **SchemaExport**
 - Create DDL from mapping files
- **SchemaUpdate**
 - Update schema from mapping files
- **CodeGenerator**
 - Create Java from mapping files
- **MapGenerator/XDoclet**
 - Create mapping files from Java
- **Middlegen**
 - Create Java and mapping files from existing database
- **Hibernate Console**
 - View object graphs, edit properties, etc.
- **Eclipse Plug-ins**
 - Visually create mapping files and test queries

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- **BlueGlue**
- Conclusion

BlueGlue

- Point-and-click GUI that automatically installs, configures, integrates, deploys, and tests up to 100 Open Source projects for Java and LAMP developers
- Includes Hibernate and many sample apps
 - Hibernate-XDoclet: Hibernate w/auto-generated mappings
 - Hibernate-Middlegen: Database generates Hibernate code
 - Hibernate-AspectJ: Simple Hibernate usage via AspectJ power
 - Hibernate-JBoss-AspectJ: Use Hibernate with AspectJ in a J2EE environment to make life with EJB's and persistence much easier
 - Hibernate also used in Spring-JSF, Spring-Struts, Spring-Tapestry, Spring-WebWork, and Spring-MVC samples

Hibernate Usage in BlueGlue

- `// Outside of J2EE: No tables, exceptions, tx, or Hibernate`
`address = new Address("123 Elm", "Denver", "CO", "80132");`
`cust = new Customer("jd@some.org", "John", "Doe", address);`
`cust.persist();`
- `// SessionBean: No trace of tables, exceptions, tx, etc.`
`public List findByLastName(String lastName)`
`{`
 `Query query = getNamedQuery("find-by-last-name");`
 `query.setString("lastName", lastName);`

 `return query.list();`
`}`

Agenda

- Introductions
- Object-Relational Mapping
- Hibernate
- Example
- Configuration
- Hibernate Query Language
- Advanced Topics
- BlueGlue
- Conclusion

Conclusion

- Good
 - Hibernate is fast, easy to learn and use, and stable
 - Excellent documentation and community support
 - Broad mapping support (1-1, many-1, 1-many, many-many, inheritance schemes, composition, etc.)
 - Pluggable (database, caching, identity strategies)
 - Works inside and outside containers (J2EE, Spring)
 - Plays nice in clusters

Conclusion (cont.)

- Bad
 - Still relatively young – permanent staying power?
 - Proprietary in that it doesn't implement a standard
 - Mapping files can become very complex

Conclusion (cont.)

- Ugly?
 - It does not follow a standard, such as JDO
 - Open Source with beneficent dictator
 - JBoss affiliation

References and Links

- Hibernate Home Page
 - <http://www.hibernate.org>
- Download (current stable version is 2.1.8)
 - <http://sf.net/projects/hibernate/>
- Introduction to Hibernate (The Server Side)
 - <http://www.theserverside.com/articles/content/Hibernate/IntroductionToHibernate.pdf>
- Help with Hibernate mapping files
 - <http://www.xylax.net/hibernate/>
- Hibernate in Action (Manning)
 - <http://www.manning.com/bauer>
- Hibernate: A Developer's Notebook (O'Reilly)
 - <http://www.oreilly.com/catalog/hibernate/>

Any Questions?