

Introduction to Hibernate

Christopher Bartling
Identix, Inc.

Topics to be covered

- Introductions
- Object-relational mapping (ORM) basics
- Basic Hibernate
- Discussion

Who am I?

- Employed at Identix, Inc. as a software engineer.
- Build facial and fingerprint biometric security applications and infrastructure.
- Develop solutions utilizing both Java/J2EE and .NET platforms.
 - Server-side work has been Java-based.
 - Application development has been database-centric.
- We have been using Hibernate 1.2.x and 2.x since February 2003.

Making a case for ORM

- A major part of any enterprise application development project is the persistence layer.
- Data lives in the relational database
 - A fact that will not be changing any time soon!
- We want to work with objects having behavior, not rows and columns of data.
- *Object-relational impedance mismatch.*

Transparent persistence

- GOAL: We want to directly manipulate data contained in a relational database using an object programming language (Java).
- SOLUTION: Use a mapping layer to map between the object paradigm and the relational paradigm.
- Use *object caching techniques* to optimize the mapping solution for performance.

Transparent vs. Call-level

- IEEE Computer article comparing transparent persistence (ODMG Java Binding) to call-level (JDBC) persistence.
- For this exercise, 496 lines of code were needed using the ODMG Java Binding compared to 1,923 lines of code using JDBC.
- *Development time can be significantly reduced through the use of transparent persistence provided by object-relational mapping frameworks.*

What is Hibernate?

- Object/relational mapping framework for Java.
- Licensed under the Lesser GPL.
 - Can be used in commercial products.
- Build persistent objects following common Java idioms:
 - Association
 - Inheritance
 - Polymorphism
 - Composition
 - Collections API for “many” relationships.

Why use Hibernate?

- Focus on domain object modeling.
- Performance.
 - High performance object caching.
 - Configurable materialization strategies.
- Sophisticated query facilities
 - Hibernate Query Language (HQL)
 - Criteria API and Query by Criteria
 - Query by Example.
- *NIH* -- Not invented here.

Tool support

- XDoclet
 - Code generator using javadoc tags to add attributes to source code, generating Hibernate mappings and database definition language from the source code.
- Middlegen
 - Code generator which generates Hibernate mappings and objects from an existing database model.
- AndroMDA
 - Code generation framework that follows the *model driven architecture* (MDA) paradigm.
- Spring Framework
 - J2EE framework with Hibernate support.

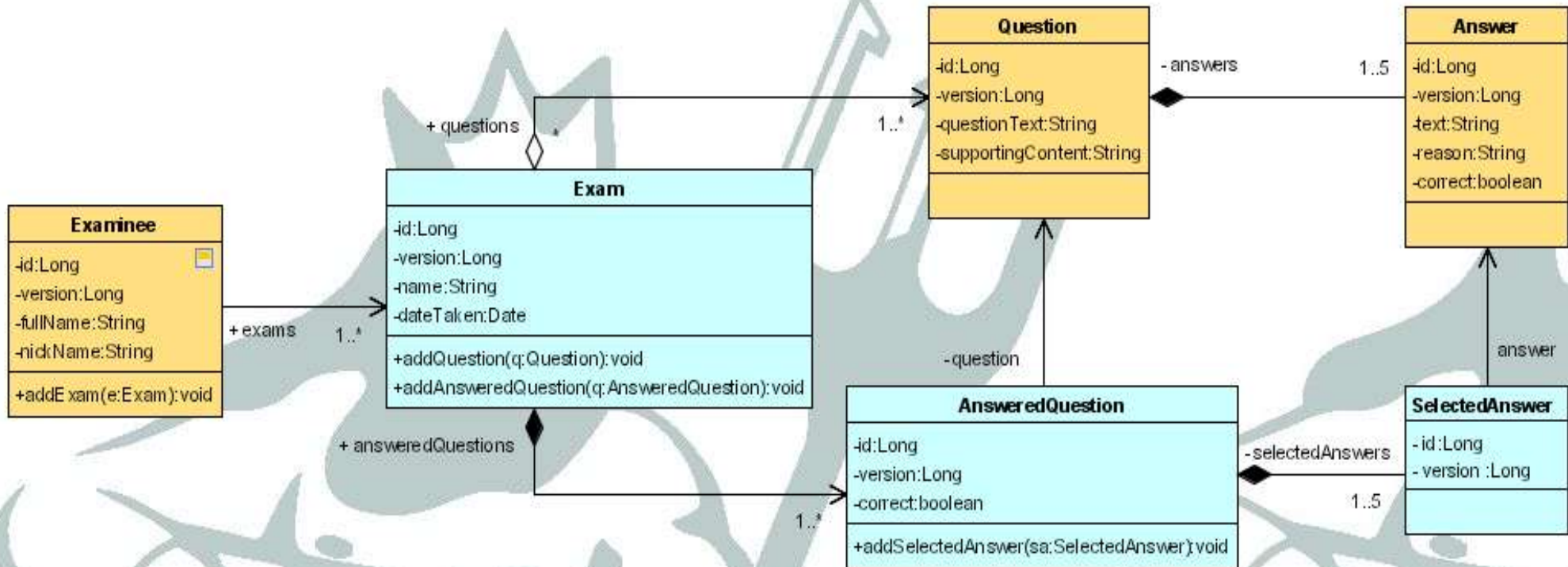
The Quizzer example

- A domain object model of potential quiz application.
- Will be used to demonstrate several topics within this presentation.
- Packaged as a JAR file containing domain objects, data access objects, and JUnit test cases.
- Shell scripts (Linux) or batch files (Windows) in the *build* directory can be use to build and run each example.

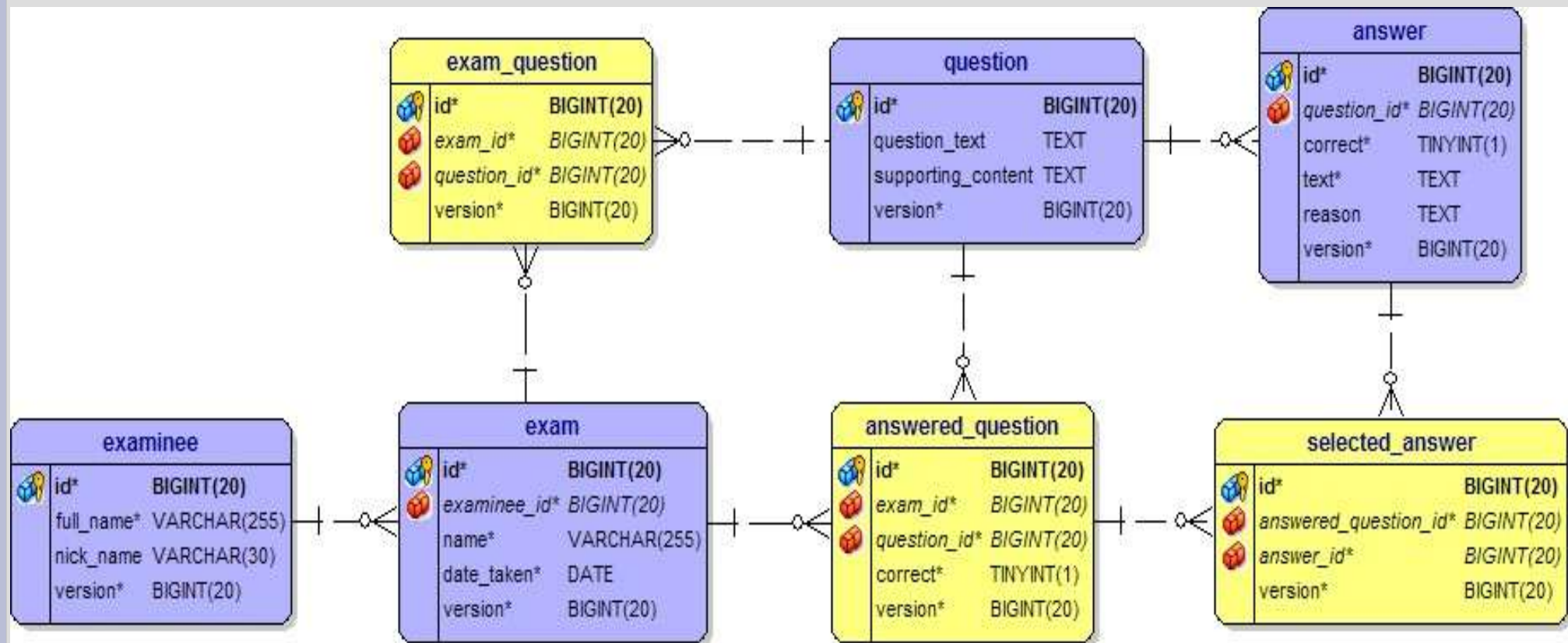
Quizzer requirements

- Questions and associated answers exist independently of any exam.
- An exam is built by associating n number of questions to the exam.
- Each exam can be composed of different questions.
- Each exam is associated with only one examinee.
- An examinee can have 0 or more exams.
- Answered questions and selected answers are created when the examinee takes the exam.
- Each answered question can have 1 to 5 selected answers.

Quizzer object model



Entity-relationship diagram



Introduction to Hibernate

- Presentation and examples based on Hibernate 2.1.1.
- Domain objects defined in Java as Plain Old Java Objects (POJOs).
- Mapping file
- Configuration file
- Hibernate runtime

Using Hibernate

- Define persistent domain object in Java
- Define database schema in SQL.
- Define or generate Hibernate mapping file mapping
 - Classes to tables
 - Object attributes to table columns
 - Object relationships to table joins
- Build Hibernate configuration file.
- Create/update DAOs for new domain object.
- Create JUnit test cases to test DAO.
 - Use Hibernate classes to fetch, insert, update, and delete objects to/from database.
 - Load database with lots of test data. Is list fetching performant?

Hibernate and data modeling

- My experience with Hibernate has only been with new database schemas.
 - Use synthetic primary keys (AUTO_INCREMENT in MySQL and sequences in Oracle).
 - Use versioning to support optimistic concurrency/locking.
 - We have had the luxury of changing our data model to better fit how Hibernate works.
 - Our object and data models are not large.

Define persistent domain object

- Hibernate can persist POJOs which follow JavaBeans specification.
 - No arg constructor.
 - Getter/setter methods for mapped attributes.
- No requirement to inherit from a persistent base class or interface.
- Cross-cutting persistence features are *declaratively* maintained within the Hibernate configuration file and mapping files.
 - Object caching
 - Relationship materialization strategies

Hibernate mapping file

- Defined by a DTD.
- Mapping file is XML.
- Can be generated from the Java source using XDoclet.
 - *Much easier to keep in sync if generated.*
- Defines identifier generation, versioning, persistent properties, and object relationships and the mapping of these to the database.

Mapping classes to tables

- Use the *class* element.

```
<class name="quizzer.domain.Answer"  
       table="answer"  
       dynamic-update="false"  
       dynamic-insert="false">
```

...

```
</class>
```

Mapping classes to tables

- XDoclet usage:

```
/**
 * @hibernate.class table="answer"
 */
public class Answer
    implements Serializable, Comparable {
    ...
}
```

Mapping object properties

- Use the *property* element.

```
<property
  name="reason"
  type="java.lang.String"
  update="true"
  insert="true"
  column="reason"
  not-null="true" />
```

Mapping object properties

- XDoclet usage:

```
/**
 * @hibernate.property    column="reason"
 *                       not-null="true"
 */
public String getReason() {
    return this.reason;
}
```

Generating object identifiers

- Use the *id* element:

```
<id name="id"
    column="id"
    type="java.lang.Long"
    unsaved-value="null">
    <generator class="native" />
</id>
```

Generating object identifiers

- XDoclet usage:

```
/**
 * @hibernate.id generator-class="native"
 *             unsaved-value="null"
 */
public Long getId() {
    return this.id;
}
```

Mapping relationships

- Number of different ways to map relationships in Hibernate, depending on the relationship multiplicity.
 - **many-to-one**: ordinary reference to another persistent object. Child object lifetime is *independent* of parent.
 - **one-to-many**: a collection of references to another class of mapped objects.
 - **one-to-one**: reference to another persistent object. Child object lifetime is *dependent* on parent lifetime.
 - **many-to-many**: collection of entities with its own table.
 - Intervening collection (association) table needed.
- Table subset mapping via **component** and **dynamic-component** elements.

Transitive persistence

- *Persistence-by-reachability*
- Persisting any transient, persistence-capable objects that can be reached transitively from any persistent object.
- Object's lifetime determined by its reachability from a designated set of root persistent objects.
- Transitive persistence is not the default behavior in Hibernate, but it can be enabled.

Mapping collections

- Mappings include `<set>`, `<list>`, `<map>`, `<bag>`, `<array>`, `<primitive-array>`
- Correlation to many of the Java Collections interfaces:
 - `java.util.Map`
 - `java.util.Set`
 - `java.util.SortedMap`
`java.util.SortedSet`
 - `java.util.List`
 - `java.util.Collection`
- ***BEST PRACTICE:** Declare collection properties using interface types in your mapped objects.*

Mapping inheritance

- **discriminator**: *table-per-class-hierarchy* strategy
 - Use a discriminator column in the mapped table.
 - Use **subclass** nested element to define each subclass and the distinct discriminator value.
- **joined-subclass**: *table-per-subclass* strategy
 - Foreign key relationship exists between common table and subclass table.

Using XDoclet and Ant

- Run Example 1.
- Live example of using XDoclet and Ant to build the Hibernate mapping files automatically from javadoc comments in the POJO source files.
- New **.hbm.xml* files will be created in the *source/generated* directory of the examples project area.
- **BEST PRACTICE:** Use XDoclet and Ant to generate the Hibernate mapping files.

Hibernate Query Language

- Very similar to SQL (intentional).
- Understands inheritance, polymorphism, associations, aggregations, compositions.
 - Selection: *from*, *as*
 - Associations and joins: *inner join*, *left outer join*, *right outer join*, *full join*
 - Projection: *select*, *elements*
 - Constraints: *where*
 - Other constructs: aggregate functions, expressions, order by clauses, group by clauses, polymorphic selections, subqueries
- HQL is much less verbose than SQL.
- Run Example 6 for demonstration.

HQL examples

- `from quizzer.domain.Question as question`
- `from java.lang.Object as object`
- `from quizzer.domain.Question as question
order by question.questionText`
- `select elements(question.answers)
from quizzer.domain.Question as question`
- `select count(question)
from quizzer.domain.Question as question`
- `select question.questionText
from quizzer.domain.Question as question`

Why another query language?

- HQL seems like SQL, but wasn't the whole idea of ORM to reduce the need for the query language?
- The Hibernate Criteria API may solve this problem...

Criteria Queries

- New in Hibernate 2.1.
 - In its current manifestation, not as powerful as the HQL query facilities.
 - Does not currently support projection or aggregation.
- The ***net.sf.hibernate.Criteria*** interface represents a query against a persistent class.
- Use ***net.sf.hibernate.expression.Criterion*** implementations to narrow your search.
- Use the Hibernate Session to create these objects.
- This API looks like a more dynamic querying facility than the more mature HQL.

Criteria examples

- Run Example 7.
- The `TEST_CriteriaQueries` JUnit test case contains examples of Criteria queries.
- Examples of using criteria, expressions, and results ordering.
- Examples of using the Query by Example using the ***net.sf.hibernate.expression.Example*** interface.
 - Use a persistent object to specify criterion attributes for the Criteria query.

Criteria query examples

```
Criteria criteria = session.createCriteria(Question.class);  
List objects = criteria.list();
```

```
Criteria criteria = session.createCriteria(Question.class);  
criteria.add(Expression.like("questionText", "%kiwi%"));  
List objects = criteria.list();
```

```
Criteria criteria = session.createCriteria(Examinee.class);  
Examinee examinee = new Examinee();  
examinee.setNickName("Chris");  
Example examineeExample = Example.create(examinee);  
criteria.add(examineeExample);  
List objects = criteria.list();
```

Architecting with Hibernate

- Presentation layer (web or rich client)
- Service layer
 - Higher level business logic lives here.
 - Commands and Struts Actions.
- DAO layer
 - Hibernate lives here.
 - Domain objects know nothing of Hibernate.
- DB layer
- Domain objects migrate between the top three layers.
 - Detaching POJOs from Session is key.

Selling Hibernate to management

- Reduces development effort significantly for database-centric applications.
- Promotes domain object modelling and localization of database access code.
- Excellent support and documentation.
 - JBoss Group offers support contracts for Hibernate.
- Open source and free for all uses.
 - LGPL is commercial software friendly.
 - Open source seems to be creating de facto standards in today's world.
 - ODMG is dead--long live Hibernate!!

Hibernate Alternatives

- Object relational mapping tool comparison Wiki...
 - <http://c2.com/cgi-bin/wiki?ObjectRelationalToolComparison>

Tools used

- XDoclet 1.2b4
- Ant 1.6.0
- Hibernate 2.1.1
- MySQL 4.0.17
- J/Connector 3.0.8 JDBC driver
- IntelliJ IDEA 3.0.5
- AspectJ 1.1.0
- OpenOffice 1.1.0
- Poseidon for UML 2.1.2 (via Web Start)
- DbVisualizer 4.0.2
- CVS, TortoiseCVS, SmartCVS
- Red Hat Linux 9.0 and Windows XP Home

References

- *Threading lightly, Part 3: Sometimes it's best not to share.*
<http://www-106.ibm.com/developerworks/java/library/j-threads3.html>
- *Orthogonal Persistence for the Java Platform.*
http://research.sun.com/research/forest/COM.Sun.Labs.Forest.doc.eth_sep99.slides_pdf.pdf
- *ObjectStore Java API User Guide.* http://tinman.cs.gsu.edu/~raj/osjidoc/apiug/1_intro.htm
- *Core J2EE Patterns – Intercepting Filter.*
<http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html>
- *Follow the Chain of Responsibility.*
<http://www.javaworld.com/javaworld/jw-08-2003/jw-0829-designpatterns.html>
- *Transparent persistence vs. JDBC call-level interfaces.*
http://www.service-architecture.com/object-relational-mapping/articles/transparent_persistence_vs_jdbc
- Laddad, Ramnivas. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co.. 2003. pp. 256-260.
- http://www.hibernate.org/hib_docs/online/javapolis2003_presentation/hibernate_javapolis2003.pdf

Contact information

- Email: chris@bartling.net
- Website: <http://www.bartling.net>