

## Introduction to Hibernate

HADN Ch. 1-4

MSIS 531 – Spring 2006

### Learning Goals

2

- Hibernate overview; purpose, configuration, etc.
- Using mapping documents
- Generating Java classes
- Generating database DDL
- Connecting to different platforms
- Persisting and querying objects
- Mapping, persisting, and retrieving collections
- Other topics
  - Bidirectional associations
  - Simple collections

## Hibernate Overview

- Problem: Object/Relational Mapping (ORM)
  - How to cleanly connect Java objects to RDBMS tables?
  - EJB provides a solution, but at the cost of some complexity
    - Hibernate's newest version is supposedly the basis for EJB3
  - As we've seen, JDBC allows connectivity, but developer owns some hard problems
- Hibernate: lightweight, database-independent ORM solution
  - Abstraction of database persistence, in the same way we've seen for model/view/controller

## Hibernate Configuration

- Hibernate setup just requires adding a group of Java classes to the project
- Configuration methods shown in the book are different than those for NetBeans
- For the initial Hibernate content, we'll follow the book and run the examples from the command line
  - Gives a point for future reference
  - Makes the intention of each section more obvious
  - Allows easier access to Ant build tool
- Our goal as developers is to ignore this configuration for the most part; it's a one-time charge

## Aside: Java “Ant” Build Tool

- Anyone who’s used the unix “make” tool, or variants, has felt the pain of the developers who built Ant
- Ant goal: Java-based equivalent of make
  - Transparent creation of build artifacts (code, documentation, etc.), only building what has changed
  - Production delivery, e.g. .war files for Tomcat
- Ant has become an integral part of nearly all Java development
  - Tightly integrated into NetBeans, other IDEs
  - The HADN examples show all build work is possible from the command line, using vi/notepad as the IDE

## Hibernate Mapping Documents

- The mapping document is the blueprint for both Java classes and RDBMS schema creation
  - A metadata view of a class and a table, from a single source
- All information about table structure must be included
  - Primary key, table relationships, etc.
- A key development artifact, obviously
- Question: what happens when business requirements change and downstream artifacts have been created?
  - We’ll see this is a question that is partially, but not completely, addressed

Example: Track.hbm.xml

## Generating Java Classes

- Hibernate includes a built-in Ant task, Hbm2JavaTask, that generates Java classes for each Hibernate mapping document
  - Standard JavaBean classes, with some Hibernate-specific additions, are produced
- Key issue: object/table row “sameness”
  - Java identity: two objects are equal if they refer to the same memory location
  - Java equality: two objects are equal if the “equals” method returns true
  - RDBMS identity: two table rows are identical if they have the same primary key value

Example: Ch. 2 Ant “codegen” task

## Generating RDBMS Tables

- Generating database tables is equally straightforward, using the SchemaExportTask
  - Goal: produce database-dependent DDL
  - Hibernate does all the heavy lifting, a small amazing thing
  - Configuration stored in hibernate.properties
- Caution noted: project as configured drops and recreates schema tables
  - Potential data loss possible
- Java code must be available to schema generator for inspection (Java term: “introspection”)
  - So Ant “schema” depends on “codegen”

Example: Ch. 2 Ant “schema” task

## Switching Platforms

- Delivery to other platforms is straightforward
- All popular RDBMS “dialects” supported
  - Open source and proprietary
- Project changes to switch from MySQL to Oracle:
  - Make Oracle JDBC driver available to Ant and NetBeans
  - Modify hibernate.properties to use Oracle params
  - Rerun “schema” task
- Compare the level of effort with that required to recode for another platform using JDBC
  - Significant vendor knowledge required: Hibernate owns this problem

Example: Ch. 2 Ant “schema” task (Oracle)

## Persisting Data

- Now for the fun part: actually using the tables
- The CreateTest class exercises the full range of Hibernate functionality to do something simple: add a few rows of data
  - We’ll want to replace the user interface
- Note the Configuration and SessionFactory calls
  - These give access to the runtime Hibernate environment
- The record additions are straightforward
  - Wrapped in a transaction
  - Try/catch block is used to catch an error and roll back
  - Session is closed in finally block

Example: Ch. 3 Ant “ctest” task

## Selecting Data

- Now, as the book notes, we throw the lever in reverse
  - Populate Java objects from database tables
- Program structure nearly identical to previous example
  - Trick used to make classes easy to find (one mapping document per Java class)
  - No transaction required, since we're not saving anything
  - Session.find() method used to query tables (see tracksNoLongerThan() method)
  - ListIterator used similarly to RecordSet in .NET
    - Note cast to Track before getTitle(), getPlayTime() called

Example: Ch. 3 Ant "qtest" task

## Mapping Collections

- Collections are just DBMS relationships
- Schema complicated by relationship (M:M) between artists and the music tracks they play on
- Key question: where is the association stored from a Java perspective?
  - Initially, as a collection of Artist objects associated with Tracks
  - Hibernate implements this (transparently) as an intersection table, which is exactly what we want
- First example just populates some artist/track data

Example: Ch. 4 Ant "ctest" task

## Retrieving Collections

- No surprise: changes to Ch. 3 QueryTest class are minor
  - Need helper method to list all artists on a track
  - Make Artist class available to Hibernate config
  - Call helper class
- Not much additional work to handle a significantly more complicated situation
- Note no mention of intersection table (TRACK\_ARTISTS)
  - As modifications are made to the two intersected tables, this one is updated automatically

Example: Ch. 4 Ant “qtest” task

## Bidirectional Relationships

- In previous example, we added artists to tracks
  - Hibernate is aware (via the mapping document) of the inverse relationship
    - Persistence is automatic
  - Example shows it in use
- At issue: the mapping docs make distinction between “primary” and “inverse” objects
  - Relationship changes made to inverse object (Artist in this case) would not be persisted
  - Reflects reality (normally one side or the other is “more important”) but must be kept in mind

Example: Ch. 4 Ant “qtest2” task

## Simple Collections

- Associations to simple values are easy, as are 1:M relationships
  - TRACK\_COMMENTS table is added with a “<set>” entry in the TRACK table
  - Hibernate automatically generates the table, foreign key values
- Remarkably little work for the value added

Example: Review Track mapping doc