Walk-through of a Simple Hibernate Example

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/hibernate.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Spring & Hibernate training, see courses at http://courses.coreservlets.com/.

Taught by the experts that brought you this tutorial.
Available at public venues, or customized versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics
• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details
Topics in This Section

• Creating a simple, but full, end to end Hibernate Application
Building a Hibernate Application

1. Define the domain model
2. Setup your Hibernate configuration
   - hibernate.cfg.xml
3. Create the domain object mapping files 
   - <domain_object>.hbm.xml
4. Make Hibernate aware of the mapping files 
   - Update the hibernate.cfg.xml with list of mapping files
5. Implement a HibernateUtil class 
   - Usually taken from the Hibernate documentation
6. Write your code
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <!-- All Hibernate configuration files validate against their appropriate xml DTDs -->
    <session-factory>
        <!-- Configure Hibernate here, particularly the session-factory -->
    </session-factory>
</hibernate-configuration>
hibernate.cfg.xml

...<session-factory>
    <property name="hibernate.connection.driver_class">
        oracle.jdbc.driver.OracleDriver
    </property>

    <property name="hibernate.connection.url">
        jdbc:oracle:thin:@localhost:1521:XE
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.password">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="hibernate.connection.username">
        lecture2
    </property>

    <property name="dialect">
        org.hibernate.dialect.Oracle10gDialect
    </property>

    <property name="connection.pool_size">
        1
    </property>

    <property name="current_session_context_class">
        thread
    </property>

    <property name="show_sql">
        true
    </property>

    <property name="format_sql">
        false
    </property>

</session-factory>
...
Configuring Hibernate

- There are multiple ways to configure Hibernate, and an application can leverage multiple methods at once.
- Hibernate will look for and use configuration properties in the following order:
  - hibernate.properties (when 'new Configuration()' is called)
  - hibernate.cfg.xml (when 'configure()' is called on Configuration)
  - Programatic Configuration Settings

```java
SessionFactory sessionFactory =
    new Configuration()
    .configure("hibernate.cfg.xml")
    .setProperty(Environment.DefaultSchema, "MY_SCHEMA");
```

Object Mapping Files

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/
  hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class>
        . . .
    </class>
</hibernate-mapping>
```

- DTD for the object mapping files
- Describe your class attributes here.
Account Object / Table

**Account**
- accountId : long
- accountType : String
- creationDate : Date
- balance : double

**ACCOUNT TABLE**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Default</th>
<th>Primary Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT_ID</td>
<td>NUMBER</td>
<td>No</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>ACCOUNT_TYPE</td>
<td>VARCHAR(200)</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP(6)</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BALANCE</td>
<td>NUMBER</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Account.hbm.xml Mapping File

```xml
...<class name="courses.hibernate.vo.Account" table="ACCOUNT">

  <id name="accountId" column="ACCOUNT_ID">
    <generator class="native"/>
  </id>

  <property name="creationDate" column="CREATION_DATE" type="timestamp" update="false"/>
  <property name="accountType" column="ACCOUNT_TYPE" type="string" update="false"/>
  <property name="balance" column="BALANCE" type="double"/>

</class>
...
```

Mapping file named after Java Object
Hibernate ID Generators

• Native:
  – Leverages underlying database method for generating ID (sequence, identity, etc…)

• Increment:
  – Automatically reads max value of identity column and increments by 1

• UUID:
  – Universally unique identifier combining IP & Date (128-bit)

• Many more…

Identify Mapping Files in the hibernate.cfg.xml

...  
<property name="dialect">
  org.hibernate.dialect.Oracle10gDialect
</property>

<property name="connection.pool_size">1</property>

<property name="current_session_context_class">
  thread
</property>

<property name="show_sql">true</property>
<property name="format_sql">false</property>

<mapping resource="Account.hbm.xml"/>

</session-factory>
...
**HibernateUtil**

- **Convenience class to handle building and obtaining the Hibernate SessionFactory**
  - Use recommended by the Hibernate org
- **SessionFactory is thread-safe**
  - Singleton for the entire application
- **Used to build Hibernate ‘Sessions’**
  - Hibernate Sessions are NOT thread safe
  - One per thread of execution

```java
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```
Common Methods of Session API

• Hibernate Session
  - `session.saveOrUpdate()`
  - `session.get()`
  - `session.delete()`

• What about just plain save?
  - It’s there, but not typically used
  - `session.save()`

Account DAO – `saveOrUpdate()`

```java
public void saveOrUpdateAccount(Account account) {
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();

    session.saveOrUpdate(account);
}
```

Remember the number of LOC needed to do this with JDBC?
JDBC Example – Create Account

```java
public Account createAccount(Account account) {
    Connection connection = null;
    PreparedStatement getAccountIdStatement = null;
    PreparedStatement createAccountStatement = null;
    ResultSet resultSet = null;
    long accountId=0;
    try {
        Connection connection = DriverManager.getConnection("jdbc:oracle:thin:lecture1/password@localhost:1521:XE");
        connection.setAutoCommit(false);
        getAccountIdStatement = connection.prepareStatement("SELECT ACCOUNT_ID_SEQ.NEXTVAL FROM DUAL");
        resultSet = getAccountIdStatement.executeQuery();
        resultSet.next();
        accountId = resultSet.getLong(1);
        createAccountStatement = connection.prepareStatement(AccountDAOConstants.CREATE_ACCOUNT);
        createAccountStatement.setLong(1, accountId);
        createAccountStatement.setString(2,account.getAccountType());
        createAccountStatement.setDouble(3, account.getBalance());
        createAccountStatement.execute();
        connection.commit();
    } catch (SQLException e) {
        try{
            connection.rollback();
        }catch(SQLException e1){// log error}
    } catch(SQLException e1){// log error}
    throw new RuntimeException(e);
} finally {
    try {
        if (resultSet != null)
            resultSet.close();
        if (getAccountIdStatement!= null)
            getAccountIdStatement.close();
        if (createAccountStatement!= null)
            createAccountStatement.close();
        if (connection != null)
            connection.close();
    } catch (SQLException e) {// log error}
}
```

Account DAO – get()

```java
public Account getAccount(long accountId) {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    Account account = (Account)session.get(Account.class,accountId);
    return account;
}
```
Account DAO – delete()

```java
public void deleteAccount(Account account) {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.delete(account);
}
```

Testing with JUnit

- JUnit is an open source framework to perform testing against units of code.
  - A single test class contains several test methods
  - Provides helper methods to make ‘assertions’ of expected results
  - Common to have multiple test classes for an application

- Using JUnit
  1. Download the jar from JUnit.org
  2. Add downloaded jar to project classpath
  3. Create a class to house your test methods, naming it anything you like (typically identifying it as a test class)
  4. Implement test methods, naming them anything you like and marking each with the @Test annotation at the method level
  5. Call the code to be tested passing in known variables and based on expected behavior, use ‘assert’ helper methods provided by JUnit to verify correctness
     ```java
     Assert.assertTrue(account.getAccountId() == 0);
     ```
JUnit and Eclipse

• JUnit comes with most Eclipse downloads

Right click on project and select properties
Under the Libraries tab, select ‘Add Library’
Select JUnit

Running JUnit in Eclipse

Right click on your test class
Select ‘Run As’ from menu
Select ‘JUnit Test’ from menu
@Test
public void testCreateAccount() {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Account account = new Account();

    // no need to set id, Hibernate will do it for us
    account.setAccountType(Account.ACCOUNT_TYPE_SAVINGS);
    account.setCreationDate(new Date());
    account.setBalance(1000L);

    // confirm that there is no accountId set
    Assert.assertTrue(account.getAccountId() == 0);
    ...

    // save the account
    AccountService accountService = new AccountService();
    accountService.saveOrUpdateAccount(account);

    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();

    System.out.println(account);

    // check that ID was set after the hbm session
    Assert.assertTrue(account.getAccountId() > 0);
}

Handling Transactions

- Why am I starting/ending my transactions in my test case?
  - In order to take advantage of certain Hibernate features, the Hibernate org recommends you close your transactions as late as possible. For test cases, this means in the tests themselves
  - Later we’ll discuss suggested ways of handling this within applications

Test Create - Output

31 [main] INFO org.hibernate.cfg.Environment - Hibernate 3.3.0.SP1
31 [main] INFO org.hibernate.cfg.Environment - hibernate.properties not found
47 [main] INFO org.hibernate.cfg.Environment - Bytecode provider name: javassist
47 [main] INFO org.hibernate.cfg.Environment - using JDK 1.4 java.sql.Timestamp handling
125 [main] INFO org.hibernate.cfg.Configuration - configuring from resource: /hibernate.cfg.xml
125 [main] INFO org.hibernate.cfg.Configuration - Configuration resource: /hibernate.cfg.xml
375 [main] INFO org.hibernate.cfg.Configuration - Configured SessionFactory: null
453 [main] INFO org.hibernate.connection.DriverManagerConnectionProvider - Using Hibernate built-in connection pool (not for production use!)
453 [main] INFO org.hibernate.connection.DriverManagerConnectionProvider - Hibernate connection pool size: 1
453 [main] INFO org.hibernate.connection.DriverManagerConnectionProvider - autocommit mode: false
Test Create – Output

469 [main] INFO org.hibernate.connection.DriverManagerConnectionProvider - connection properties: {user=lecture2, password=****}
750 [main] INFO org.hibernate.cfg.SettingsFactory - JDBC driver: Oracle JDBC driver, version: 10.2.0.1.0XE
797 [main] INFO org.hibernate.transaction.TransactionFactoryFactory - Using default transaction strategy (direct JDBC transactions)
797 [main] INFO org.hibernate.transaction.TransactionManagerLookupFactory - No TransactionManagerLookup configured (in JTA environment, use of read-write or transactional second-level cache is not recommended)
797 [main] INFO org.hibernate.cfg.SettingsFactory - Automatic flush during beforeCompletion(): disabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - Automatic session close at end of transaction: disabled

Test Create – Output

797 [main] INFO org.hibernate.cfg.SettingsFactory - JDBC batch size: 15
797 [main] INFO org.hibernate.cfg.SettingsFactory - JDBC batch updates for versioned data: disabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - Scrollable result sets: enabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - JDBC3 getGeneratedKeys(): disabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - Connection release mode: auto
797 [main] INFO org.hibernate.cfg.SettingsFactory - Default batch fetch size: 1
797 [main] INFO org.hibernate.cfg.SettingsFactory - Generate SQL with comments: disabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - Order SQL updates by primary key: disabled
797 [main] INFO org.hibernate.cfg.SettingsFactory - Order SQL inserts for batching: disabled
797 [main] INFO org.hibernate.hql.ast.ASTQueryTranslatorFactory - Using ASTQueryTranslatorFactory
Test Create – Output

Hibernate: select hibernate_sequence.nextval from dual

Hibernate: insert into ACCOUNT (CREATION_DATE, ACCOUNT_TYPE, BALANCE, ACCOUNT_ID) values (?, ?, ?, ?)

1453 [main] INFO org.hibernate.impl.SessionFactoryImpl - closing

    var account =
        ----ACCOUNT----
    accountId=1
    accountType=SAVINGS
    creationDate=Sat Sep 13 21:53:01 EDT 2008
    balance=1000.0
    ----ACCOUNT----
@Test
public void testGetAccount(){
    Account account = createAccount(); // create account to get
    Session session = HibernateUtil.getSessionFactory().
        getCurrentSession();
    session.beginTransaction();
    AccountService accountService = new AccountService();
    Account anotherCopy = accountService.
        getAccount(account.getAccountId());
    System.out.println(account);
    // make sure these are two separate instances
    Assert.assertTrue(account != anotherCopy);
    System.out.println("var anotherCopy = "
        + anotherCopy);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
}
@Test
public void testUpdateAccountBalance() {
    // create account to update
    Account account = createAccount();

    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    AccountService accountService = new AccountService();
    account.setBalance(2000);
    accountService.saveOrUpdateAccount(account);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();

    ...

    Session session2 = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session2.beginTransaction();
    Account anotherCopy = accountService
        .getAccount(account.getAccountId());
    System.out.println(anotherCopy);
    System.out.println(anotherCopy);
    // make sure the one we just pulled back from the
    // database has the updated balance
    Assert.assertTrue(anotherCopy.getBalance() == 2000);
    session2.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
}
Test Update Balance - Output

Hibernate: update ACCOUNT set BALANCE=? where ACCOUNT_ID=?

Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_,
    account0_.CREATION_DATE as CREATION2_0_0_,
    account0_.ACCOUNT_TYPE as ACCOUNT3_0_0_,
    account0_.BALANCE as BALANCE0_0_ from ACCOUNT
account0_ where account0_.ACCOUNT_ID=?

var anotherCopy =
----ACCOUNT----
accountId=22
accountType=SAVINGS
creationDate=2008-09-13 22:56:42.296
balance=2000.0
----ACCOUNT----

Test Delete

@Test
public void testDeleteAccount() {
    // create an account to delete
    Account account = createAccount();
    Session session = HibernateUtil.getSessionFactory().
        getCurrentSession();
    session.beginTransaction();
    AccountService accountService = new AccountService();
    accountService.deleteAccount(account);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
    ...
}
Test Delete

```java
... 
Session session2 = HibernateUtil.getSessionFactory().
    getCurrentSession();
session2.beginTransaction();

// try to get the account again -- should be null
Account anotherCopy = accountService
    .getAccount(accountService.getAccount(accountId));

System.out.println("var anotherCopy = "
    + anotherCopy);

Assert.assertNull(anotherCopy);

session2.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
}
```

Test Delete - Output

Hibernate: delete from ACCOUNT where ACCOUNT_ID=5
Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_,
    account0_.CREATION_DATE as CREATION2_0_0_,
    account0_.ACCOUNT_TYPE as ACCOUNT3_0_0_,
    account0_.BALANCE as BALANCE0_0_ from ACCOUNT
    account0_ where account0_.ACCOUNT_ID=5

var anotherCopy = null
Remember “update=false”? 

```xml
...<class name="courses.hibernate.vo.Account" table="ACCOUNT">
    <id name="accountId" column="ACCOUNT_ID">
        <generator class="native"/>
    </id>

    <property name="creationDate" column="CREATION_DATE"
        type="timestamp" update="false"/>

    <property name="accountType" column="ACCOUNT_TYPE"
        type="string" update="false"/>

    <property name="balance" column="BALANCE"
        type="double"/>
</class>
...
```

Test Update Account Type

```java
@Test
public void testUpdateAccountType() {
    // create account to update
    Account account = createAccount();

    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    AccountService accountService = new AccountService();
    account.setAccountType(Account.ACCOUNT_TYPE_CHECKING);
    accountService.saveOrUpdateAccount(account);
    session.getTransaction().commit();
    HibernateUtil.getSessionFactory().close();
    ...
```
Test Update Account Type

```java
... 

Session session2 = HibernateUtil.getSessionFactory().getCurrentSession();
session2.beginTransaction();

Account anotherCopy = accountService.getAccount(account.getAccountId());

System.out.println(anotherCopy);

// make sure the one we just pulled back from
// the database DOES NOT HAVE the updated type
Assert.assertFalse(anotherCopy.getAccountType().equals(Account.ACCOUNT_TYPE_CHECKING));

session2.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
}
```

Test Update Account Type - Output

Hibernate: insert into ACCOUNT (CREATION_DATE, ACCOUNT_TYPE, BALANCE, ACCOUNT_ID) values (?, ?, ?, ?)

Hibernate: select account0_.ACCOUNT_ID as ACCOUNT1_0_0_, account0_.CREATION_DATE as CREATION2_0_0_, account0_.ACCOUNT_TYPE as ACCOUNT3_0_0_, account0_.BALANCE as BALANCE0_0_ from ACCOUNT account0_ where account0_.ACCOUNT_ID=?

var anotherCopy =
----ACCOUNT----
accountId=82
accountId=82
accountType=SAVINGS
creationDate=2008-09-28 22:04:43.718
balance=1000
----ACCOUNT----
Wrap-up

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Preview of Next Sections

• Associations and Collections
• Realizing relationships in Java and the database
• Use Hibernate to help bridge the gap between the two
Summary

• End to end Hibernate Application
  – Configuration
    • hibernate.cfg.xml
      <property name="hibernate.connection.url">
        jdbc:oracle:thin:@localhost:1521:XE
      </property>
  – Object mapping files
    • Account.hbm.xml
      <class name="courses.hibernate.vo.Account" table="ACCOUNT">
        <property name="accountType"
          column="ACCOUNT_TYPE"
          type="string"
          update="false"/>
      </class>

– HibernateUtil to handle Session
  static {
    sessionFactory = newConfiguration()
      .configure().buildSessionFactory();
  }

  public static SessionFactory
    getSessionFactory() {
    return sessionFactory;
  }

– Writing the implementation
  Session session =
    HibernateUtil.getSessionFactory()
      .getCurrentSession();
  session.saveOrUpdate(account);
Questions?

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.