

HIBERNATE

Relational Persistence for Java and .NET

Presentation by Rodney Beede

CSCI 5448 – Object Oriented Analysis and Design
University of Colorado, 2011-11-04

[contactme@nospam@rodneybeede\[D0t\]com](mailto:contactme@nospam@rodneybeede[D0t]com)

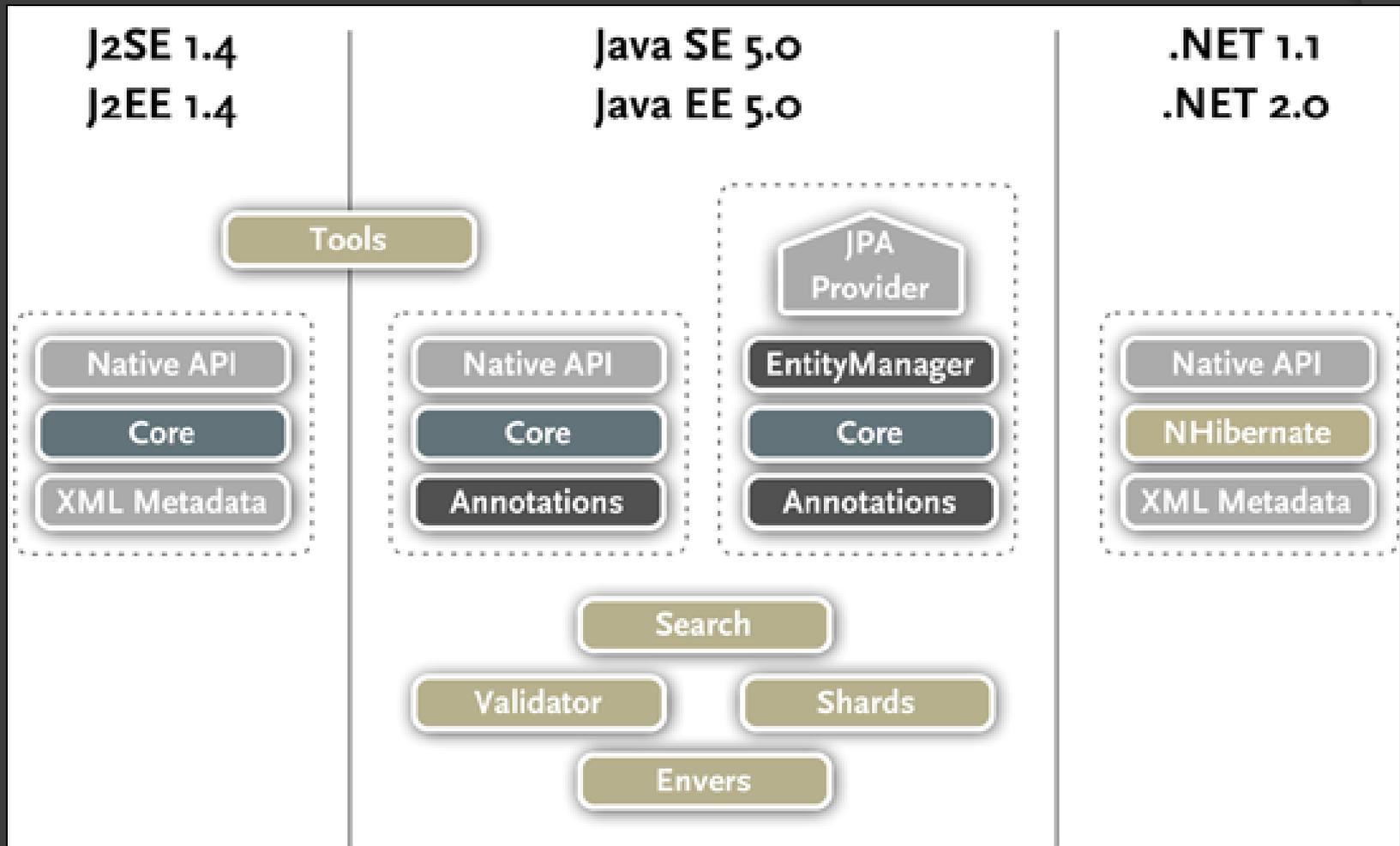
What is Hibernate?

- ◎ <http://www.hibernate.org/>
 - Hosted by the JBoss Community
- ◎ Their Definition:
 - “Hibernate is a high-performance Object/Relational persistence and query service”
- ◎ Traditional (historical) use
 - Mapping Java objects to relational databases
- ◎ Today
 - Collection of projects/frameworks for extended use of POJO (plain old Java objects)

Why use Hibernate?

- Simpler data persistence
 - Automatically handles mapping SQL to Object and vice versa
 - Automatic creation of database schemas
 - Automatic updating of database schemas
 - Add a field to an object; Hibernate converts your existing database for you.
 - Provides search functionality
- Simpler database management
 - No JDBC code or SQL code needed
 - Yet you can still use SQL if you want
 - Easy to swap out database engines by a simple configuration change
 - No need to create the schema on the new database
- It's free
 - LGPL (use in open or closed source project)
 - Open source and standards = no vendor lock-in

Hibernate Stack



Hibernate Projects

- ◎ Core
 - Provides the core functionality of object relational mapping and persistence
- ◎ Shards
 - Provides for horizontal partitioning of Core so you can put object data in multiple databases
- ◎ Search
 - Combines Apache Lucene full-text search engine with Core. Extends the basic query capabilities of Core.
- ◎ Tools
 - Eclipse plug-ins to facilitate
 - Creating Hibernate mapping files
 - Database connection configurations
 - Reverse engineering existing databases into Java class code

Hibernate Projects (2)

- ◎ Validator
 - [JSR 303 - Bean Validation](#)
 - Standardized way of annotating JavaBean fields with value constraints
 - @NotNull on a bean field also gets set in the database schema
- ◎ Metamodel Generator
 - Auto-creation of Criteria classes for use in Java EE 6 Criteria API
 - Non-string based API for object-based queries
- ◎ OGM (Object/Grid Mapper)
 - Allows use of NoSQL data stores versus SQL relational
- ◎ Envers
 - Automatic revision history of persistent classes

This presentation will focus on the Core project.

Getting Hibernate

- ◎ Hibernate 3.6
 - Latest stable (4.0 is out though)
- ◎ Easiest method is Maven
 - Alternative is “release bundles”
 - <http://www.hibernate.org/downloads.html>
 - Sample code has complete example
 - Next slides show snippets for pom.xml
- ◎ See [Hibernate Getting Started Guide](#) also

Maven – pom.xml - dependencies

```
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-core</artifactId>  
  <version>3.6.8.Final</version>  
</dependency>
```

- Easier compared to version 3.3.x

Maven – pom.xml - repositories

```
<repository>
  <id>jboss-public-repository-group</id>
  <name>JBoss Public Maven Repository Group</name>
  <url>https://repository.jboss.org/nexus/content/groups/public-
jboss/</url>
  <layout>default</layout>
  <releases>
    <enabled>true</enabled>
    <updatePolicy>never</updatePolicy>
  </releases>
  <snapshots>
    <enabled>true</enabled>
    <updatePolicy>never</updatePolicy>
  </snapshots>
</repository>
```

Hibernate Class Entities

- ⦿ Class attributes
 - Hibernate uses reflection to populate
 - Can be private or whatever
- ⦿ Class requirements
 - Default constructor (private or whatever)
 - “However, package or public visibility is required for runtime proxy generation and efficient data retrieval without bytecode instrumentation.”
- ⦿ JavaBean pattern common
 - Not required though but easier
- ⦿ 3 methods of serialization definition
 - Following slides

Hibernate Annotation Mappings

- ⦿ Annotations in code
 - Beginning of class
 - Indicate class is Entity
 - Class doesn't have to implement `java.lang.Serializable`
 - Define database table
 - Define which attributes to map to columns
 - Supports auto-increment IDs too
 - Can dictate value restrictions (not null, etc)
 - Can dictate value storage type
- ⦿ Existed before JPA standard (later slides)
- ⦿ Doesn't require a separate `hbm.xml` mapping file (discussed later)
 - But is tied to code

Hibernate Annotation Example

```
@Entity
@Table( name = "EVENTS" )
public class Event {
    private Long id;
    ...
    @Id
    @GeneratedValue(generator="increment"
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() { return id; }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
    ...
}
```

Hibernate Annotation Example (2)

```
@Entity
```

```
@Table( name = "EVENTS" )
```

```
public class Event {  
    private Long id;
```

```
...
```

```
    @Id
```

```
    @GeneratedValue(generator="increment"
```

```
    @GenericGenerator(name="increment", strategy = "increment")
```

```
    public Long getId() { return id; }
```

```
    @Temporal(TemporalType.TIMESTAMP)
```

```
    @Column(name = "EVENT_DATE")
```

```
    public Date getDate() { return date; }
```

```
    public void setDate(Date date) { this.date = date; }
```

```
...
```

```
}
```

Tells hibernate this goes into the
EVENTS table

Hibernate Annotation Example (3)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
    private Long id;
```

...

```
    @Id
    @GeneratedValue(generator="increment"
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() { return id; }
```

```
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
```

...

```
}
```

Tells hibernate that this is an auto generated field for the database

Hibernate Annotation Example (4)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
    private Long id;
    ...
    @Id
    @GeneratedValue(generator="increment"
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() { return id; }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EVENT_DATE")
    public Date getDate() { return date; }

    public void setDate(Date date) { this.date = date; }
    ...
}
```

Note that you don't need any annotations on the actual private fields or setters if you use the standard JavaBean pattern. The getter defines it.

Hibernate Annotation Example (5)

```
@Entity
@Table( name = "EVENTS" )
public class Event {
...
    private String title;

    public String getTitle() { return title; }

    public void setTitle(String title) { this.title = title; }
...
}
```

Also note that this is automatically stored with a column name of "title" so we didn't have to add any annotations

JPA Annotation

- ⦿ Became standard
 - Came after Hibernate annotations
- ⦿ Works almost like Hibernate annotations
 - Requires “META-INF/persistence.xml” file
 - Defines data source configuration
 - HibernatePersistence provider
 - Auto-detects any annotated classes
 - Auto-detects any hbm.xml class mapping files
 - (later slides)
 - Allows explicit class loading for mapping
- ⦿ Annotation syntax
 - Same as Hibernate
 - Hibernate has a few extensions (see docs)

JPA 2 XML

- ◎ JPA 2 XML
 - like Hibernate's hbm.xml discussed later
- ◎ Separate from code unlike in-line annotations

```
01. <?xml version="1.0" encoding="UTF-8" ?>
02. <entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
03.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04.     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm ;
05.     http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
06.     version="1.0">
07.     <description>My First JPA XML Application</description>
08.     <package>entity</package>
09.     <entity class="entity.Employee" name="Employee">
10.         <table name="EMPLOYEE" />
11.         <attributes>
12.             <id name="empId">
13.                 <generated-value strategy="TABLE" />
14.             </id>
15.             <basic name="empName">
16.                 <column name="EMP_NAME" length="100" />
17.             </basic>
18.             <basic name="empSalary">
19.             </basic>
20.         </attributes>
21.     </entity>
22. </entity-mappings>
```

Hibernate *.hbm.xml Mappings

- ⦿ For each class
 - Define ClassName.hbm.xml
 - ClassName not required convention
 - Usually stored in same package
 - I like the convention of
 - src/main/java (actual .java source)
 - src/main/resources (any configuration files, et cetera)
 - src/main/hibernate (I put all of my .hbm.xml here)
 - Matching folder/package structure to src/main/java
 - Optionally multiple classes in one .hbm.xml possible
 - Not common convention though
- ⦿ Becoming legacy in favor of JPA 2 XML or Annotations

Hibernate *.hbm.xml Mappings (2)

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC  
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
```

```
    <class name="Event" table="EVENTS">  
        <id name="id" column="EVENT_ID">  
            <generator class="increment"/>  
        </id>  
        <property name="date" type="timestamp" column="EVENT_DATE"/>  
        <property name="title"/>  
    </class>
```

```
</hibernate-mapping>
```

Hibernate *.hbm.xml Mappings (3)

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC  
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
```

Define what package
for the class(es)

```
  <class name="Event" table="EVENTS">  
    <id name="id" column="EVENT_ID">  
      <generator class="increment"/>  
    </id>  
    <property name="date" type="timestamp" column="EVENT_DATE"/>  
    <property name="title"/>  
  </class>
```

```
</hibernate-mapping>
```

Hibernate *.hbm.xml Mappings (4)

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC  
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
```

```
<class name="Event" table="EVENTS">  
    <id name="id" column="EVENT_ID">  
        <generator class="increment"/>  
    </id>  
    <property name="date" type="timestamp" column="EVENT_DATE"/>  
    <property name="title"/>  
</class>
```

```
</hibernate-mapping>
```

The class name and
data store table

Hibernate *.hbm.xml Mappings (5)

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC  
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
```

```
<class name="Event" table="EVENTS">  
    <id name="id" column="EVENT_ID">  
        <generator class="increment"/>  
    </id>  
    <property name="date" type="timestamp" column="EVENT_DATE"/>  
    <property name="title"/>  
</class>
```

```
</hibernate-mapping>
```

The properties of the class to save (getAttribute() or class.attribute style)

Hibernate Configuration

- ⦿ Need to define
 - Data source (database) connection
 - Engine, URI, credentials, etc
 - Pooling mechanism (Hibernate provides C3P0)
 - Mapping XML definitions or classes with annotations
 - You specify which ones to actually activate
 - Also allows for multiple databases and mappings
 - **Many IDE and Ant tools exist to auto-create .hbm.xml and hibernate.cfg.xml/persistence.xml**
- ⦿ JPA
 - Provides some auto-discovery at startup
 - Limited to jar files
- ⦿ Hibernate
 - Has XML or .properties (not used much)
- ⦿ Approaches
 - XML configuration file
 - Programmatic

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0/EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property name="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1;MVCC=TRUE</pr
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.H2Dialect</property>

        <!-- Disable the second-level cache -->
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">>true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <!-- Names the annotated entity class -->
        <mapping class="org.hibernate.tutorial.annotations.Event"/>

    </session-factory>

</hibernate-configuration>
```

Usually placed in root of the classpath (/hibernate.cfg.xml) for auto-detection.

You can also programmatically load it in code.

hibernate.cfg.xml (2)

```
<!-- Database connection settings -->
<property
name="connection.driver_class">org.h2.Driver</property>
<property
name="connection.url">jdbc:h2:mem:db1;DB_CLOSE_DELAY=-
1;MVCC=TRUE</property>
<property name="connection.username">sa</property>
<property name="connection.password"></property>
```

- ⦿ You define the database settings in this manner
- ⦿ There are third party library extensions that allow the password inside hibernate.cfg.xml to be encrypted as well
 - Or you could use programmatic configuration instead
- ⦿ You can't use system/environment variables inside the config

hibernate.cfg.xml (3)

```
<!-- SQL dialect -->
```

```
<property name="dialect">org.hibernate.dialect.H2Dialect</property>
```

- ◉ Depending on the database/data source you use you need to let Hibernate know how to talk to it. Hibernate supports many data sources.

```
<!-- Drop and re-create the database schema on startup -->
```

```
<property name="hbm2ddl.auto">create</property>
```

- ◉ You can have Hibernate:
 - “validate” - Exists and has expected tables/columns; don't touch data/schema
 - “update” – Create if needed and update tables/columns if class has new fields
 - “create” – Drop any existing and create new (only at start of session)
 - “create-drop” – Same as create but also drop at end of session

Caveats of hbm2ddl.auto

- ⦿ Some people don't recommend
 - “update” can mess up
 - Doesn't remove renamed columns/attributes
 - Makes new one by default unless you update mapping
 - Migrations not perfect
 - Suppose add not null constraint after the fact
 - Existing nulls would blow up
 - Not recommended for production use anyway
- ⦿ Hibernate class tools like
 - SchemaExport and SchemaUpdate
 - Useful for getting auto-generated SQL migration/creation scripts
- ⦿ When set to “create” or “create-drop”
 - hbm2ddl.import_file – allow specifying */path/somefile.sql* to run
 - (used to be hard coded “/import.sql”)
 - Needs to be in classpath

hibernate.cfg.xml (4)

<!-- Names the annotated entity class -->

```
<mapping  
  class="org.hibernate.tutorial.annotations.Event"/>
```

- ⦿ For each class with annotations you provide a <mapping/> entry
- ⦿ For each .hbm.xml it looks like this instead
 - <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
 - Event.hbm.xml is in the classpath
 - No *.hbm.xml is possible, sorry. See persistence.xml for wildcards or programmatic configuration.

persistence.xml

- ◎ JPA 2 XML method
 - Similar idea to hibernate.cfg.xml
- ◎ Additions
 - Allows defining EJB Persistence provider
 - Usually HibernatePersistence suffices
 - jar-file option
 - Allows auto-inclusion of any classes with annotations
 - No need for manual mapping like hibernate.cfg.xml
 - Also auto-includes any .hbm.xml files

Programmatic Configuration

```
Configuration cfg = new Configuration();  
cfg.addResource("Item.hbm.xml");  
cfg.addResource("Bid.hbm.xml");
```

- ⦿ Assumes the .hbm.xml are in the classpath
 - This example assumes the root

Programmatic Configuration (2)

```
Configuration cfg = new Configuration();  
cfg.addClass(org.hibernate.auction.Item.class);  
cfg.addClass(org.hibernate.auction.Bid.class);
```

- ⦿ Translates to
“/org/hibernate/auction/Item.hbm.xml”
 - Again in classpath
 - Avoids hardcoded filenames (less work than updating hibernate.cfg.xml)
- ⦿ If class is annotated uses annotation versus xml

Programmatic Configuration (3)

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class)
    .setProperty("hibernate.dialect",
        "org.hibernate.dialect.MySQLInnoDBDialect")
    .setProperty("hibernate.connection.datasource",
        "java:comp/env/jdbc/test")
    .setProperty("hibernate.order_updates", "true");
```

- ⦿ Database settings can be configured as well
 - Note the “hibernate.” prefix on the options this time

Programmatic Configuration (4)

```
Configuration conf = new Configuration();  
conf.configure().setProperty("hibernate.connection.url",  
    "jdbc:mysql://"+host+"/"+dbname);
```

```
conf.setProperty("hibernate.connection.username",  
    user);
```

```
conf.setProperty("hibernate.connection.password",  
    password);
```

- ⦿ Load /hibernate.cfg.xml as defaults
 - Mappings still defined inside XML
- ⦿ Override database settings with run-time values

Mapping & Configuration Summary

- ◎ Each Java class
 - Annotations or XML mapping file (.hbm.xml)
 - or JPA persistence orm.xml
 - You can mix annotations and XML mappings
- ◎ Hibernate XML Configuration
 - Each class/.hbm.xml gets <mapping/> entry
 - Configure database
 - hibernate.cfg.xml
 - or for JPA use persistence.xml
 - Alternative
 - Programmatic configuration
 - Hybrid approach

Initializing Hibernate

```
try {  
    // Create the SessionFactory from hibernate.cfg.xml  
    return new Configuration().configure().buildSessionFactory();  
} catch (Throwable ex) {  
    // Make sure you log the exception, as it might be swallowed  
    System.err.println("Initial SessionFactory creation failed." + ex);  
    throw new ExceptionInInitializerError(ex);  
}
```

- ⦿ SessionFactory returned is used later

Code from documentation at http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#tutorial-firstapp-helpers

Saving Data

```
Session session =  
    HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();
```

```
Event theEvent = new Event();  
theEvent.setTitle(title);  
theEvent.setDate(theDate);
```

```
session.save(theEvent);
```

```
session.getTransaction().commit();
```

- ⦿ Just start a transaction similar to how you do in databases

Loading Data

```
Session session =  
    HibernateUtil.getSessionFactory().getCurrentSession()  
    n();  
session.beginTransaction(); // important even for query  
List result = session.createQuery("from  
    Event").list();  
session.getTransaction().commit();  
return result;
```

- Note the “from Event” which is SQL like
 - Known as HQL
 - More complex queries possible
 - See <http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/queryhql.html>

Querying Data

- ◎ Remember don't use string concatenation to form queries
 - Bad: "from Users where id=" + paramUserId
 - Why? SQL Injection Vulnerabilities
 - paramUserId = "1 OR 1=1"
- ◎ Use Queries instead
 - List cats = sess.createCriteria(Cat.class)
.add(Restrictions.like("name", "Fritz%"))
.add(Restrictions.between("weight", minWeight, max))
.list();
 - http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#querycriteria
 - List cats = session.createQuery(
"from Cat as cat where cat.birthdate < ?")
.setDate(0, date)
.list();
 - http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#objectstate-querying

Closing Connection

```
sessionFactory.close();
```

- ◎ Closes resources
 - Connection pools
 - Caches
 - Etc.

Session and Thread Safety

- ◎ `org.hibernate.Session`
 - Don't use the deprecated class `org.hibernate.classic.Session`
- ◎ **NOT** thread safe
 - Each thread should call `SessionFactory.getCurrentSession()`
 - Versus `.openSession()` which has
 - No automatic close management
 - No automatic association to a thread

Sample Code

◎ Sample1

- Creates new database (if needed)
 - Located in “../SampleDatabase/Database”
 - Parent of current directory
 - HSQLDB database engine
- Populates database with data
- Queries database for data
- Outputs results

Sample 2

- ⦿ Reads existing database
 - Same database from Sample 1
 - Opens in “update” schema mode
- ⦿ Has MODIFIED “Sample” class
 - Added an attribute
- ⦿ Adds a new single entry
 - Will have additional attribute
- ⦿ Outputs found objects

Schema Updating

- ⦿ View updated database schema
 - Run Sample 1 first
 - View table schema with command below
 - Run Sample 2 next
 - View table schema with command below
 - Note the addition of another field in the table

- ⦿ On command line

```
java -cp /path/to/hsqldb-2.2.4.jar  
org.hsqldb.util.DatabaseManager -user sa -url  
"jdbc:hsqldb:file:./SampleDatabase/Database"
```

- All one line
- Hsqldb jar probably in
%USERPROFILE%\m2\repository\org\hsqldb\hsqldb\2.2.4\hsqldb-
2.2.4.jar
- SampleDatabase path may need updated unless you are in the parent
folder of the two sample code directories

References

⦿ References

- Hibernate.org
- http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html_single/
- http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/