First Hibernate example

This tutorial shows a simple example using Hibernate. We will create a simple Java application, showing how Hibernate works.

General

Author: Sebastian Hennebrueder

Date:

First edition 2005

updated February, 9th 2008

Used software and frameworks

Eclipse 3.x

MyEclipse or alternatively Hibernate Tools are recommended but optional

Hibernate 3.x (I used 3.3)

Source code: http://www.laliluna.de/download/first-hibernate-example-tutorial.zip

The sources does not include the libraries. Download the libraries from hibernate.org and your database driver and add them to the project as explained below! The example must be configured to work with your database settings! Read the tutorial.

PDF version of the tutorial: http://www.laliluna.de/download/first-hibernate-example-tutorial-en.pdf

Old PDF version with Hibernate 2: http://www.laliluna.de/download/first-hibernate-2-example-tutorial-en.pdf

Short introduction

Hibernate is a solution for object relational mapping and a persistence management solution or persistent layer. This is probably not understandable for anybody learning Hibernate.

What you can imagine is probably that you have your application with some functions (business logic) and you want to save data in a database. When you use Java all the business logic normally works with objects of different class types. Your database tables are not at all objects.

Hibernate provides a solution to map database tables to a class. It copies one row of the database data to a class. In the other direction it supports to save objects to the database. In this process the object is transformed to one or more tables.
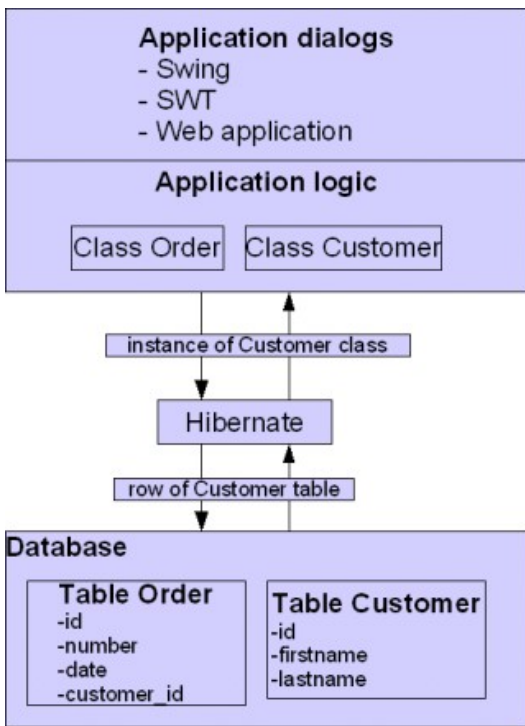
Saving data to a storage is called persistence. And the copying of tables to objects and vice versa is called object relational mapping.

Create a Java Project

Using Eclipse press the keys *Ctrl+n (Strg+n)* to create a new project. Select a Java project. We will call it FirstHibernateExample.

Prepare the project for Hibernate using MyEclipse

When you are using MyEclipse, right click on your project in the package explorer and choose *Add Hibernate capabilities.*
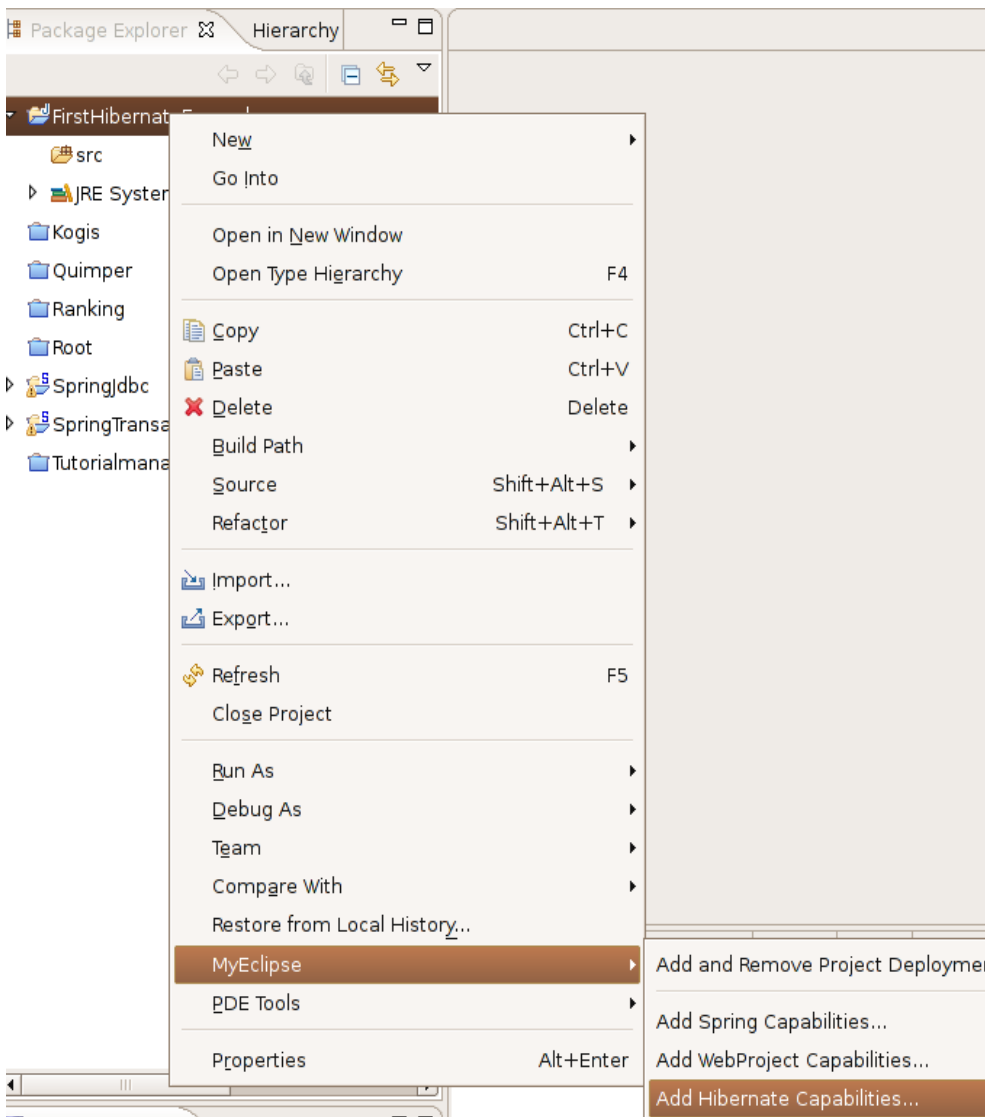
Continue the wizard and create a new *hibernate.cfg.xml* in the *src* directory.

In the last step you can create a Hibernate SessionFactory. I prefer to create my own. You can find it below.

Prepare the project for Hibernate for anybody

If you do not use MyEclipse, you can use the ant build to retrieve the libs using Ivy or alternatively just download Hibernate from the website http://www.hibernate.org/

You will need at least Hibernate Core. If you want to use Annotations you need Hibernate Annotations and if you want to use Hibernate Search, you need to download Hibernate Search as well.

Extract the file. Hibernate comes with a long list of libraries. You do not need all of them. There is a REAME file in the lib directory explaining what is required. Open your project properties, select "Java Build Path", click on "Add External Jars" and add the libaries shown below to your project path.

The following list includes the libraries to use Hibernate with XML or annotations, EHCache, PostgreSQL and logging over log4j.

- antlr.jar

- backport-util-concurrent.jar

- hibernate-commons-annotations.jar

- hibernate-annotations.jar

- postgresql.jar

- hibernate-ehcache.jar

- log4j.jar

- ejb3-persistence.jar

- slf4j-log4j12.jar

- slf4j-api.jar

- javassist.jar

- commons-collections.jar

- dom4j.jar

- lucene-core.jar

- commons-logging.jar

- hibernate-search.jar

- jta.jar

- hibernate-core.jar

- ehcache.jar

- xml-apis.jar

Create a SessionFactory

A session factory is important for Hibernate. It implements a design pattern, that ensures that only one instance of the session is used per thread. You should only get your Hibernate session from this factory.

Create a class named InitSessionFactory in the package de.laliluna.hibernate and add the source code below.

```
/**
 *
 * @author Sebastian Hennebrueder
 * created Feb 22, 2006
 * copyright 2006 by http://www.laliluna.de
 */
package de.laliluna.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

/**
 * @author hennebrueder This class garanties that only one single
SessionFactory
 *          is instanciated and that the configuration is done thread safe
as
 *          singleton. Actually it only wraps the Hibernate SessionFactory.
 *          You are free to use any kind of JTA or Thread
transactionFactories.
 */
public class SessionFactoryUtil {

  /** The single instance of hibernate SessionFactory */
  private static org.hibernate.SessionFactory sessionFactory;
```

```
  /**
   * disable contructor to guaranty a single instance
   */
  private SessionFactoryUtil() {
  }

  static{
// Annotation and XML
     sessionFactory = new
AnnotationConfiguration().configure().buildSessionFactory();
// XML only
//     sessionFactory = new
Configuration().configure().buildSessionFactory();
  }

  public static SessionFactory getInstance() {
     return sessionFactory;
  }

  /**
   * Opens a session and will not bind it to a session context
   * @return the session
   */
  public Session openSession() {
     return sessionFactory.openSession();
  }

  /**
   * Returns a session from the session context. If there is no session in
the context it opens a session,
   * stores it in the context and returns it.
   * This factory is intended to be used with a hibernate.cfg.xml
   * including the following property <property
   * name="current_session_context_class">thread</property> This would
return
   * the current open session or if this does not exist, will create a new
   * session
   *
   * @return the session
   */
  public Session getCurrentSession() {
     return sessionFactory.getCurrentSession();
  }

  /**
   * closes the session factory
   */
  public static void close(){
     if (sessionFactory != null)
        sessionFactory.close();
     sessionFactory = null;

  }
}
```

Configuring Log4J

As you can see above we added the log4j library. This library does like a configuration file in the

source directory or it welcomes you with the following error.

```
log4j:WARN No appenders could be found for logger (TestClient).
log4j:WARN Please initialize the log4j system properly.
```

Create a file named log4j.properties in the root directory and insert the following:

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=debug, stdout

log4j.logger.org.hibernate=info
#log4j.logger.org.hibernate=debug

### log HQL query parser activity
#log4j.logger.org.hibernate.hql.ast.AST=debug

### log just the SQL
log4j.logger.org.hibernate.SQL=debug

### log JDBC bind parameters ###
log4j.logger.org.hibernate.type=info

### log schema export/update ###
log4j.logger.org.hibernate.tool.hbm2ddl=info

### log HQL parse trees
#log4j.logger.org.hibernate.hql=debug

### log cache activity ###
log4j.logger.org.hibernate.cache=info

### log transaction activity
#log4j.logger.org.hibernate.transaction=debug

### log JDBC resource acquisition
#log4j.logger.org.hibernate.jdbc=debug

### enable the following line if you want to track down connection ###
### leakages when using DriverManagerConnectionProvider ###
#log4j.logger.org.hibernate.connection.DriverManagerConnectionProvider=tra
ce
```

Add the database driver

Even Hibernate needs a database driver to access a database. Open the project properties, click on "Java Build Path", select "Add External Jars" and add your database driver. If you use PostgreSQL you can find your database driver on http://jdbc.postgresql.org if you use MySQL have a look here http://www.mysql.de/products/connector/j

Create database and tables.

Create a database with MySql or PostgreSQL or anything you like. Call it "firsthibernate".

Using PostgreSql use the following script to create your table:

```
CREATE TABLE "public"."honey" (
  id SERIAL,
  name text,
  taste text,
  PRIMARY KEY(id)
);
```

Using MySql use the following script:

```
CREATE TABLE `honey` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(250) default NULL,
  `taste` varchar(250) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Create the class

Create a new class named "Honey" in the package "de.laliluna.example". Add three fields id, name and taste and generate (Context menu -> Source -> Generate Getter and Setter) or type the getters and setters for the fields. Then create an empty constructor.

```
package de.laliluna.example;

public class Honey {
  private Integer id;
  private String name;
  private String taste;

  public Honey(){

  }

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getTaste() {
    return taste;
  }

  public void setTaste(String taste) {
    this.taste = taste;
  }
```

```
  public String toString() {
     return "Honey: "+getId()+" Name: "+getName()+" Taste: "+getTaste();
  }
}
```

Create the Hibernate configuration

Create a new file named "hibernate.cfg.xml" in your root directory if it is not already created.

Insert the following in your hibernate file. Do not forget to change the username and the password to suit your database configuration.

PostgreSQL Version:

```
<?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>
<session-factory>
 <property
name="connection.url">jdbc:postgresql://localhost/firsthibernate</property
>
 <property name="connection.username">postgres</property>
 <property name="connection.driver_class">org.postgresql.Driver</property>
 <property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
 <property name="connection.password">p</property>
 <property
name="transaction.factory_class">org.hibernate.transaction.JDBCTransaction
Factory</property>
    <!--  thread is the short name for
      org.hibernate.context.ThreadLocalSessionContext
      and let Hibernate bind the session automatically to the thread
    -->
    <property name="current_session_context_class">thread</property>
    <!-- this will show us all sql statements -->
    <property name="hibernate.show_sql">true</property>
 <!-- mapping files -->
 <mapping resource="de/laliluna/example/Honey.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

MySQL Version:

```
<?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>
<session-factory>
 <property
name="connection.url">jdbc:mysql://localhost/firsthibernate</property>
 <property name="connection.username">root</property>
 <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
 <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```xml
 <property name="connection.password">r</property>
 <property
name="transaction.factory_class">org.hibernate.transaction.JDBCTransaction
Factory</property>
    <!--  thread is the short name for
      org.hibernate.context.ThreadLocalSessionContext
      and let Hibernate bind the session automatically to the thread
    -->
    <property name="current_session_context_class">thread</property>
    <!-- this will show us all sql statements -->
    <property name="hibernate.show_sql">true</property>


 <!-- mapping files -->
 <mapping resource="de/laliluna/example/Honey.hbm.xml" />

</session-factory>
</hibernate-configuration>
```

This file includes the configuration of the database in our case a PostgreSQL database and all mapping files. In our case it is only the file Honey.hbm.xml. The tag

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

configures the dialect. Change this to fit your database. Have a look in the chapter "SQL Dialects" of the Hibernate reference to find the dialect for your database.

XML Mapping

You can use XML or annotations to define, how to map your class attributes to a database table.

Create the Honey.hbm.xml in the package de.laliluna.example and change it to the following:

PostgreSQL Version:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
 <class name="de.laliluna.example.Honey" table="honey">
  <id name="id" column="id" >
  <generator class="sequence">
     <param name="sequence">honey_id_seq</param>
   </generator>

  </id>

  <property name="name" column="fooname" />
  <property name="taste" column="bartaste" />
 </class>
</hibernate-mapping>
```

MySQL Version:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
 <class name="de.laliluna.example.Honey" table="honey">
```

```
  <id name="id" column="id" >
    <generator class="increment"/>
  </id>
  <property name="name" column="fooname" />
  <property name="taste" column="bartaste" />
 </class>
</hibernate-mapping>
```

In this file the mapping from our class Honey to the database table honey is configured.

Annotation based mapping

If you use annotation, you can define the mapping in the class.

```
package de.laliluna.example;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;

@Entity
public class Honey {

 @Id
 @GeneratedValue
 private Integer id;

 private String name;

 private String taste;
// ......... snip .........
```

Create a Test Client

Create a Java Class "TestClient" in the package "de.laliluna.example".

Add the following source code. It includes methods to create entries in the database, to update and to list them.

```
package de.laliluna.example;

import java.util.Iterator;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import de.laliluna.hibernate.SessionFactoryUtil;

public class TestExample {

   final static Logger logger = LoggerFactory.getLogger(TestExample.class);

   /**
```

```java
    * @param args
    */
   public static void main(String[] args) {
     Honey forestHoney = new Honey();
     forestHoney.setName("forest honey");
     forestHoney.setTaste("very sweet");
     Honey countryHoney = new Honey();
     countryHoney.setName("country honey");
     countryHoney.setTaste("tasty");
     createHoney(forestHoney);
     createHoney(countryHoney);
     // our instances have a primary key now:
     logger.debug("{}", forestHoney);
     logger.debug("{}", countryHoney);
     listHoney();
     deleteHoney(countryHoney);
     listHoney();
     forestHoney.setName("Norther Forest Honey");
     updateHoney(forestHoney);

   }

   private static void listHoney() {
     Transaction tx = null;
     Session session =
SessionFactoryUtil.getInstance().getCurrentSession();
     try {
       tx = session.beginTransaction();
       List honeys = session.createQuery("select h from Honey as h")
               .list();
       for (Iterator iter = honeys.iterator(); iter.hasNext();) {
         Honey element = (Honey) iter.next();
         logger.debug("{}", element);
       }
       tx.commit();
     } catch (RuntimeException e) {
       if (tx != null && tx.isActive()) {
         try {
// Second try catch as the rollback could fail as well
           tx.rollback();
         } catch (HibernateException e1) {
           logger.debug("Error rolling back transaction");
         }
// throw again the first exception
         throw e;
       }


     }
   }

   private static void deleteHoney(Honey honey) {
     Transaction tx = null;
     Session session =
SessionFactoryUtil.getInstance().getCurrentSession();
     try {
       tx = session.beginTransaction();
       session.delete(honey);
       tx.commit();
     } catch (RuntimeException e) {
       if (tx != null && tx.isActive()) {
```

```
        try {
// Second try catch as the rollback could fail as well
            tx.rollback();
        } catch (HibernateException e1) {
            logger.debug("Error rolling back transaction");
        }
// throw again the first exception
        throw e;
      }
    }
  }

  private static void createHoney(Honey honey) {
    Transaction tx = null;
    Session session =
SessionFactoryUtil.getInstance().getCurrentSession();
    try {
      tx = session.beginTransaction();
      session.save(honey);
      tx.commit();
    } catch (RuntimeException e) {
      if (tx != null && tx.isActive()) {
        try {
// Second try catch as the rollback could fail as well
          tx.rollback();
        } catch (HibernateException e1) {
          logger.debug("Error rolling back transaction");
        }
// throw again the first exception
        throw e;
      }
    }
  }

  private static void updateHoney(Honey honey) {
    Transaction tx = null;
    Session session =
SessionFactoryUtil.getInstance().getCurrentSession();
    try {
      tx = session.beginTransaction();
      session.update(honey);
      tx.commit();
    } catch (RuntimeException e) {
      if (tx != null && tx.isActive()) {
        try {
// Second try catch as the rollback could fail as well
          tx.rollback();
        } catch (HibernateException e1) {
          logger.debug("Error rolling back transaction");
        }
// throw again the first exception
        throw e;
      }
    }
  }
}
```

Congratulations. You have finished your first steps in the Hibernate world.

We wanted to give you a fast entry in the Hibernate world. There are many more complex topics

and better implementation to use Hibernate. For example, the opening and closing of the session in each method is not a good practice. A session can be reused during which saves a lot of time.

If you want to learn more about best practices have a look at my eBook or my Hibernate training offers.

Copyright and disclaimer