First Hibernate example

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

Generals

Author:
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html Tutorials for Struts, EJB, xdoclet, JSF, JSP and eclipse.
Date:
February, 25th 2005

PDF Version of the tutorial:

http://www.laliluna.de/assets/tutorials/first-hibernate-example-tutorial-en.pdf

Sources
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.

http://www.laliluna.de/assets/tutorials/first-hibernate-example-tutorial.zip

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 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.

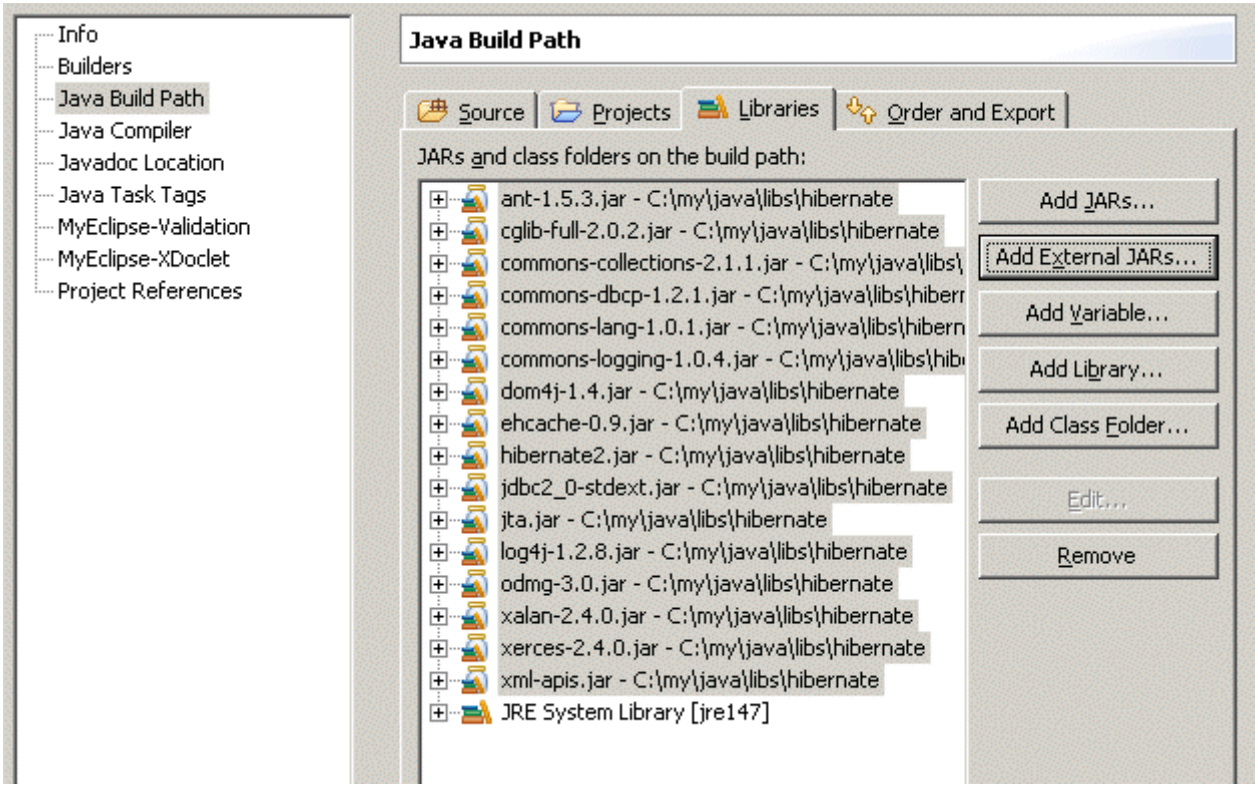
Create a Java Project

Using Eclipse press the keys “Strg + n” to create a new project. Select a Java project. We will call it FirstHibernateExample.

Add Hibernate libraries

Download Hibernate from the following website http://www.hibernate.org/

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.

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:

```
# Set root logger level to DEBUG and its only appender to A1.
log4j.rootLogger=info, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

log4j.logger.org.apache.commons.digester=INFO
```

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.

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 NOT NULL UNIQUE,
  "name" VARCHAR,
  "taste" VARCHAR,
  PRIMARY KEY("id")
) WITH OIDS;
```

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 the getters and setters for the fields. Then create an empty constructor.

```
package de.laliluna.example;

/**
 * @author laliluna
 *
 */
public class Honey {
  private Integer id;
  private String name;
  private String taste;

  public Honey(){

  }

  /**
   * @return Returns the id.
   */
  public Integer getId() {
     return id;
  }
  /**
   * @param id The id to set.
   */
  public void setId(Integer id) {
     this.id = id;
  }
  /**
   * @return Returns the name.
   */
  public String getName() {
     return name;
  }
  /**
   * @param name The name to set.
   */
  public void setName(String name) {
     this.name = name;
  }
  /**
   * @return Returns the taste.
   */
  public String getTaste() {
     return taste;
  }
```

```
  /**
   * @param taste The taste to set.
   */
  public void setTaste(String taste) {
     this.taste = taste;
  }
}
```

Create the mapping files

Create a new file named “hibernate.cfg.xml” in your root directory.

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 2.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<!-- DO NOT EDIT: This is a generated file that is synchronized -->
<!-- by MyEclipse Hibernate tool integration.                   -->
<hibernate-configuration>

    <session-factory>
        <!-- properties -->
        <property name="connection.username">postgres</property>
        <property
name="connection.url">jdbc:postgresql://localhost:5432/firsthibernate</property>
        <property
name="dialect">net.sf.hibernate.dialect.PostgreSQLDialect</property>
        <property name="connection.password">postgres</property>
        <property
name="connection.driver_class">org.postgresql.Driver</property>

        <!-- mapping files -->
        <mapping resource="de/lalilуna/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 2.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<!-- DO NOT EDIT: This is a generated file that is synchronized -->
<!-- by MyEclipse Hibernate tool integration.                   -->
<hibernate-configuration>

    <session-factory>
        <!-- properties -->
                <property name="connection.username">root</property>
        <property
name="connection.url">jdbc:mysql://localhost/firsthibernate</property>
        <property
name="dialect">net.sf.hibernate.dialect.MySQLDialect</property>
        <property name="connection.password">mysql</property>
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</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 `<property name="dialect">net.sf.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.

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

PostgreSQL Version:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
                            "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
                            "http://hibernate.sourceforge.net/hibernate-mapping-
2.0.dtd" >

<hibernate-mapping package="de.laliluna.example">
  <class name="Honey" table="honey">
     <id name="id" column="id" type="java.lang.Integer">
        <generator class="sequence">
          <param name="sequence">honey_id_seq</param>
        </generator>
     </id>
     <property name="name" column="name" type="java.lang.String" />
     <property name="taste" column="taste" type="java.lang.String" />
  </class>
</hibernate-mapping>
```

MySQL Version:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
                            "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
                            "http://hibernate.sourceforge.net/hibernate-mapping-
2.0.dtd" >

<hibernate-mapping package="de.laliluna.example">
          <class name="Honey" table="honey">
                    <id name="id" column="id" type="java.lang.Integer">
                              <!-- postgre:
                              <generator class="sequence">
                                        <param
name="sequence">honey_id_seq</param>
                              </generator>
                              -->
                              <generator class="native"/>
                    </id>
                    <property name="name" column="name" type="java.lang.String" /
>
                    <property name="taste" column="taste" type="java.lang.String"
/>
          </class>
</hibernate-mapping>
```

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

Create a SessionFactory

A session factory is important for Hibernate. It should implement 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 HibernateSessionFactory in the package de.laliluna.example and add the source code below.

```
package de.laliluna.example;

import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Session;
import net.sf.hibernate.cfg.Configuration;

/**
 * Configures and provides access to Hibernate sessions, tied to the
 * current thread of execution.  Follows the Thread Local Session
 * pattern, see {@link http://hibernate.org/42.html}.
 */
public class HibernateSessionFactory {

    /**
     * Location of hibernate.cfg.xml file.
     * NOTICE: Location should be on the classpath as Hibernate uses
     * #resourceAsStream style lookup for its configuration file. That
     * is place the config file in a Java package - the default location
     * is the default Java package.<br><br>
     * Examples: <br>
     * <code>CONFIG_FILE_LOCATION = "/hibernate.conf.xml".
     * CONFIG_FILE_LOCATION = "/com/foo/bar/myhiberstuff.conf.xml".</code>
     */
    private static String CONFIG_FILE_LOCATION = "/hibernate.cfg.xml";

    /** Holds a single instance of Session */
    private static final ThreadLocal threadLocal = new ThreadLocal();

    /** The single instance of hibernate configuration */
    private static final Configuration cfg = new Configuration();

    /** The single instance of hibernate SessionFactory */
    private static net.sf.hibernate.SessionFactory sessionFactory;

    /**
     * Returns the ThreadLocal Session instance.  Lazy initialize
     * the <code>SessionFactory</code> if needed.
     *
     *  @return Session
     *  @throws HibernateException
     */
    public static Session currentSession() throws HibernateException {
        Session session = (Session) threadLocal.get();

        if (session == null || ! session.isConnected()) {
            if (sessionFactory == null) {
                try {
                    cfg.configure(CONFIG_FILE_LOCATION);
                    sessionFactory = cfg.buildSessionFactory();
                }
                catch (Exception e) {
                    System.err.println("%%%% Error Creating SessionFactory %%%
%");
                    e.printStackTrace();
                }
            }
            session = sessionFactory.openSession();
            threadLocal.set(session);
        }

        return session;
    }
```

```
    /**
     *  Close the single hibernate session instance.
     *
     *  @throws HibernateException
     */
    public static void closeSession() throws HibernateException {
        Session session = (Session) threadLocal.get();
        threadLocal.set(null);

        if (session != null) {
            session.close();
        }
    }

    /**
     * Default constructor.
     */
    private HibernateSessionFactory() {
    }

}
```

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.apache.log4j.Logger;

import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Session;
import net.sf.hibernate.Transaction;

/**
 * @author laliluna
 *
 */
public class TestClient {

  public static void main(String[] args) {

     Integer primaryKey = createHoney();
     System.out.println("primary key is " + primaryKey);
     updateHoney(primaryKey);
     listHoney();
  }

  private static Integer createHoney() {
     Session session = null;
     Transaction tx = null;
     Logger log = Logger.getLogger("TestClient");

     log.info("creating honey");

     // [laliluna] our returned primary key
     Integer id = null;

     try {
```

```
      // [lalilуna] get the session from the factory
      session = HibernateSessionFactory.currentSession();

      // [lalilуna] always start a transaction before doing something
      // (even reading) from the database
      tx = session.beginTransaction();

      // [lalilуna] create a new object
      Honey honey = new Honey();
      honey.setName("Sebastian's favourite honey");
      honey.setTaste("sweet");

      // [lalilуna] save it to the database, Hibernate returns your object
      // with the primary key field updated!
      id = (Integer) session.save(honey);

      // [lalilуna] commit your transaction or nothing is wrote to the db
      tx.commit();

      // [lalilуna] clean up (close the session)
      session.close();

   } catch (HibernateException e) {

      // [lalilуna] when an error occured, try to rollback your
      // transaction
      if (tx != null)
        try {
                tx.rollback();
        } catch (HibernateException e1) {
                log.warn("rollback not successful");
        }
      /*
       * [lalilуna] close your session after an exception!! Your session
       * is in an undefined and unstable situation so throw it away!
       *
       */
      if (session != null)
        try {
                session.close();
        } catch (HibernateException e2) {
                log.warn("session close not successful");
        }
   }

   return id;
}

private static void updateHoney(Integer primaryKey) {
   Session session = null;
   Transaction tx = null;
   Logger log = Logger.getLogger("TestClient");

   log.info("updating honey");

   try {
      // [lalilуna] get the session from the factory
      session = HibernateSessionFactory.currentSession();

      // [lalilуna] always start a transaction before doing something
      // (even reading) from the database
      tx = session.beginTransaction();

      // [lalilуna] load object from the database
      Honey honey = (Honey) session.get(Honey.class, primaryKey);

      // [lalilуna] honey is null when no object was found
      if (honey != null) {
        honey.setName("Sascha's favourite honey");
```

```
        honey.setTaste("very sweet");
      }
      session.flush();
      // [lalilуna] commit your transaction or nothing is wrote to the db
      tx.commit();

      // [lalilуna] clean up (close the session)
      session.close();

   } catch (HibernateException e) {

      // [lalilуna] when an error occured, try to rollback your
      // transaction
      if (tx != null)
        try {
                  tx.rollback();
        } catch (HibernateException e1) {
                  log.warn("rollback not successful");
        }
      /*
       * [lalilуna] close your session after an exception!! Your session
       * is in an undefined and unstable situation so throw it away!
       *
       */
      if (session != null)
        try {
                  session.close();
        } catch (HibernateException e2) {
                  log.warn("session close not successful");
        }
   }

}

private static void listHoney() {
   Session session = null;
   Transaction tx = null;
   Logger log = Logger.getLogger("TestClient");

   log.info("listing honey");

   try {
      // [lalilуna] get the session from the factory
      session = HibernateSessionFactory.currentSession();

      // [lalilуna] always start a transaction before doing something
      // (even reading) from the database
      tx = session.beginTransaction();

      List honeys = session.find("select from Honey");

      for (Iterator iter = honeys.iterator(); iter.hasNext();) {
        Honey honey = (Honey) iter.next();
        System.out.println("Id " + honey.getId() + " Name "
                           + honey.getName());
      }

      // [lalilуna] commit your transaction or nothing is wrote to the db
      tx.commit();

      // [lalilуna] clean up (close the session)
      session.close();

   } catch (HibernateException e) {

      // [lalilуna] when an error occured, try to rollback your
      // transaction
      if (tx != null)
        try {
```

```
                    tx.rollback();
            } catch (HibernateException e1) {
                    log.warn("rollback not successful");
            }
        /*
         * [laliluna] close your session after an exception!! Your session
         * is in an undefined and unstable situation so throw it away!
         *
         */
        if (session != null)
          try {
                    session.close();
          } catch (HibernateException e2) {
                    log.warn("session close not successful");
          }
      }

  }
}
```

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