Sunday, March 20, 2011

Hibernate Tutorial for Beginners

Hibernate is an Object/ Relational Mapping solution for Java environments. It reduces the development cost by reducing paradigm mismatch between how data is represented in objects versus relational databases.

Hibernate takes care of the mapping from Java classes to database tables and provides data query and retrieval facilities. This tutorial is designed for the beginners of Hibernate and expects you to have some knowledge about Java and SQL. I'm going to use MySQL database and Eclipse IDE in this tutorial.

By following this tutorial you will get the knowledge of;

1. Setup the Development Environment for Hibernate
2. Create POJO Classes
3. Create Mapping Files
4. Hibernate Configuration
5. Create Startup Helper Class
6. Create Test Class to Load and Store Objects

Let's start.

1. Setup the Development Environment for Hibernate

First you need to set up the development environment. Following things are needed if you want to try Hibernate with MySQL using Eclipse IDE.

1. Hibernate distribution bundle – Download
2. MySQL installed in your Computer – Download
3. JDBC driver for MySQL – Download
4. slf4j logging backend – Download
5. Eclipse IDE – Download

1.1 Setup the Database

I'm going to use simple database named "userdata" which has two tables named "users" and "tasks". Figure 1 illustrates the structure of the database and there "id" column is the Primary Key of both tables and it is set to AUTO_INCREMENT.

users

Field	Type
id	int(10)
first_name	varchar(100)
last_name	varchar(100)

tasks

Field	Type
id	int(10)
user_id	int(10)
title	varchar(100)
description	varchar(100)

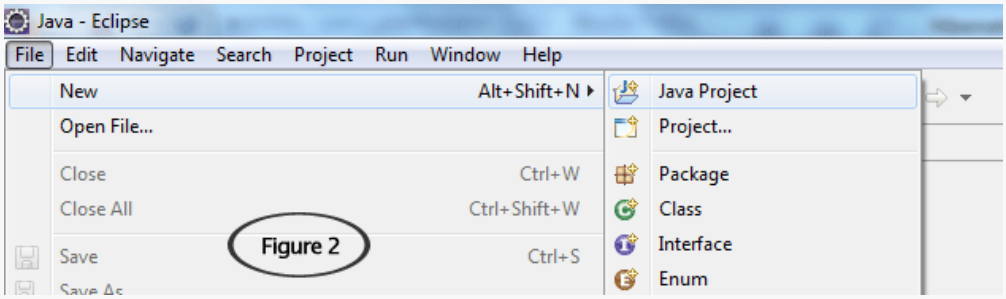
Figure 1

1.2 Create New Project

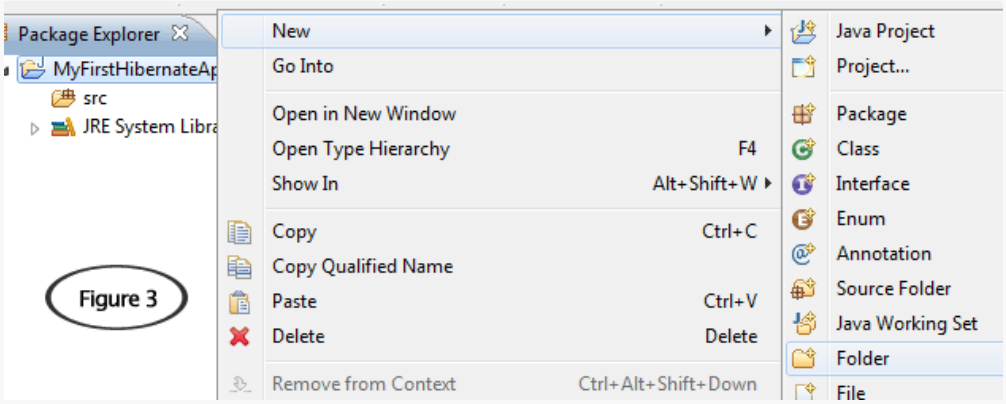
All Posts
▼ 2011 (2)
► June (1)
▼ March (1)
Hibernate Tutorial for Beginners
► 2010 (2)

In this tutorial I'm going to show how to Retrieve, Add, Delete and Update data in a MySQL database. There I'll create simple Java Project and you can use this tutorial to create web applications too.

First create new project in Eclipse using appropriate name, I'll give "MyFirstHibernateApp".



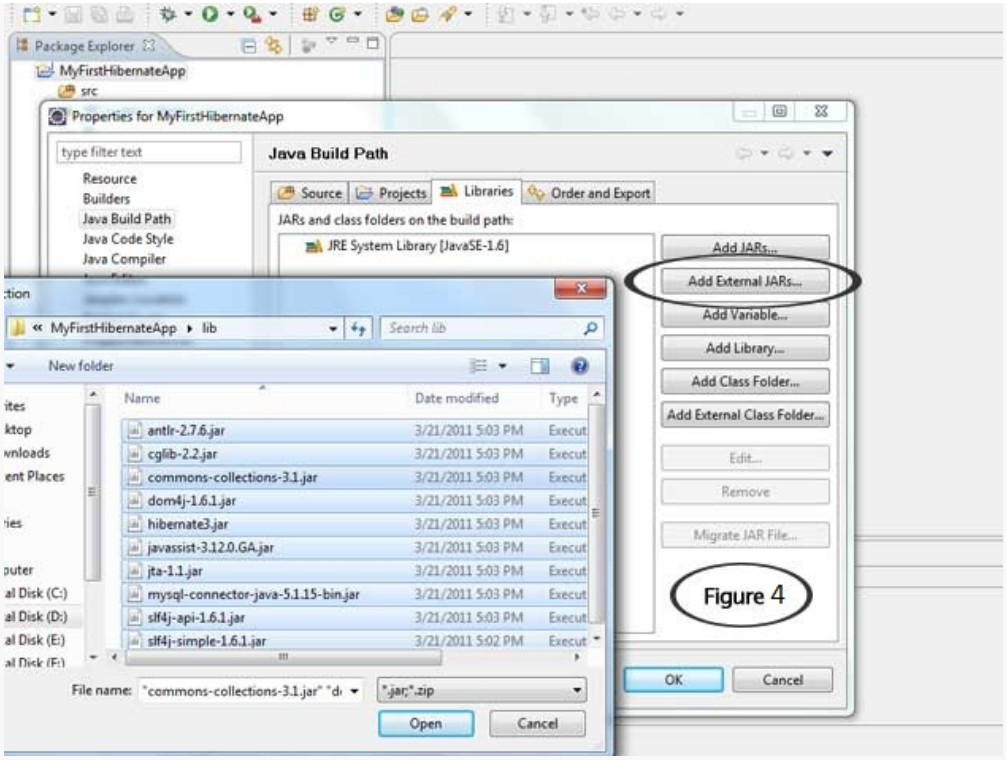
You need to manually account for all the needed dependencies, so let's create new folder called "lib" inside the "MyFirstHibernateApp" by right clicking on it as shown in following Figure 3.



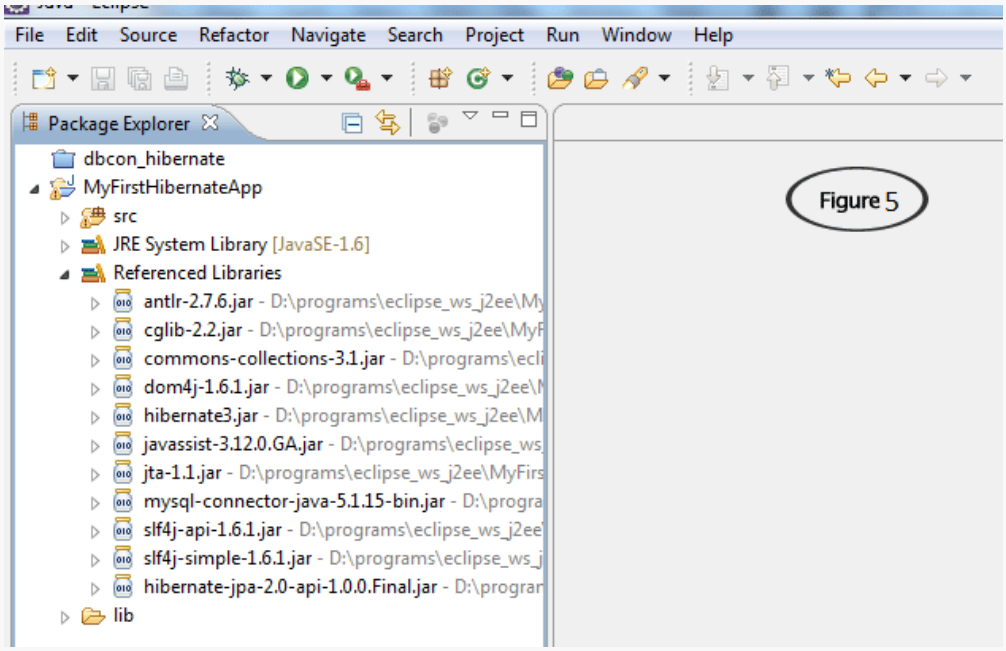
Now copy following files into that folder.

1. hibernate3.jar (Can be found in distribution bundle)
2. All artifacts in the lib/required directory in distribution bundle
3. All artifacts in the lib/jpa directory in distribution bundle
4. All files from either the lib/bytecode/cglib or lib/bytecode/javassist directory
5. One of the slf4j logging (I'll use slf4j-simple-1.6.1.jar found inside slf4j-1.6.1.zip)
6. mysql-connector-java-5.1.15-bin.jar (JDBC driver for MySQL)

Then you can configure the built path. To do that, right click on project "MyFirstHibernateApp" in Package Explorer and select Build Path --> Configure Build Path... Then go to Libraries tab and browse lib folder you created earlier. Then add external jars by selecting all jar files that you copied to lib folder in one of previous step. (Check Figure 4).



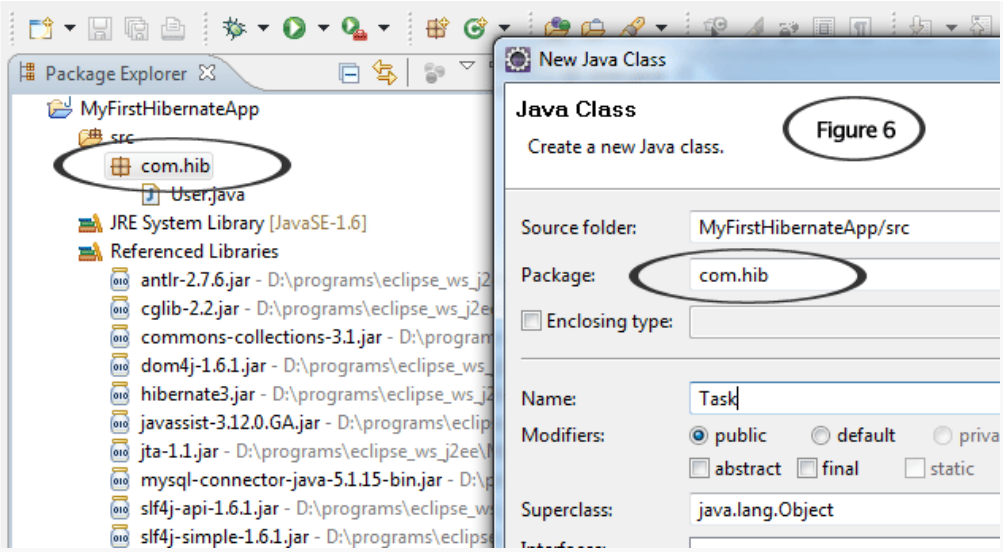
Now you will be able to see the referenced libraries as follow (Figure 5).



2. Create POJO Classes

Next, I'll create two classes that represent the user and task we want to store in the database. Those are simple JavaBean classes with some properties which uses standard JavaBean naming conventions for property getter and setter methods and private visibility for the fields.

To create new class right click on "MyFirstHibernateApp" and select New --> Class. Then give appropriate package name, I'll give "com.hib" as figure 6. You can generate getter and setters easily by selecting variable declaration and selecting Source --> Generate Getters and Setters...



User Class

```
1  package com.hib;
2
3  public class User {
4   private Integer id;
5   private String firstName;
6   private String lastName;
7
8   public Integer getId() {
9    return id;
10  }
11  public void setId(Integer id) {
12   this.id = id;
13  }
14  public String getFirstName() {
15   return firstName;
16  }
17  public void setFirstName(String firstName) {
18   this.firstName = firstName;
19  }
20  public String getLastName() {
21   return lastName;
22  }
23  public void setLastName(String lastName) {
24   this.lastName = lastName;
25  }
26 }
```

Task Class

```
1  package com.hib;
2
3  public class Task {
4
5   private Integer id;
6   private Integer userID;
7   private String title;
8   private String description;
9
10  public Integer getId() {
11   return id;
12  }
13  public void setId(Integer id) {
14   this.id = id;
15  }
16  public Integer getUserID() {
17   return userID;
18  }
19  public void setUserID(Integer userID) {
20   this.userID = userID;
21  }
22  public String getTitle() {
23   return title;
24  }
25  public void setTitle(String title) {
26   this.title = title;
27  }
28  public String getDescription() {
29   return description;
30  }
31  public void setDescription(String description) {
```

```
    this.description = description;
  }
}
```

The id property holds a unique identifier value for a particular user and task. All persistent entity classes will need such an identifier property if you want to use the full feature set of Hibernate.

3. Create Mapping Files

Hibernate needs to know how to load and store objects of the persistent class. This is where the Hibernate mapping file comes into play. The mapping file tells Hibernate what table in the database it has to access, and what columns in that table it should use.

Right click on the "src" folder and select New --> File and paste following code there. Save it as user.hbm.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 name="com.hib.User" table="users" >
  <id name="id" type="int" column="id" >
   <generator class="native"/>
  </id>

  <property name="firstName">
   <column name="first_name" />
  </property>
  <property name="lastName">
   <column name="last_name"/>
  </property>
 </class>
</hibernate-mapping>
```

Tasks.java should also have a mapping file, so create another file and paste following code. Save it as task.hbm.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 name="com.hib.Task" table="tasks">
  <id name="id" type="int" column="id" >
   <generator class="native"/>
  </id>

  <property name="userID">
   <column name="user_id" />
  </property>
  <property name="title">
   <column name="title" />
  </property>
  <property name="description">
   <column name="description"/>
  </property>
 </class>
</hibernate-mapping>
```

In the above mapping file you can see the mapping of the unique identifier property to the tables primary key. As we do not want to care about handling this identifier, we configure Hibernate's identifier generation strategy for a surrogate primary key column by giving class="native". If it is not AUTO_INCREMENT one we should use class="assign".

The id element is the declaration of the identifier property. The name="id" mapping attribute declares the name of the JavaBean property and tells Hibernate to use the getId() and setId() methods to access the property. The column attribute tells Hibernate which column of the "users" and "tasks" tables holds the primary key value. Similar to the id element, the name attribute of the property element tells Hibernate which getter and setter methods to use.

4. Hibernate Configuration

Now you have the persistent cla its mapping file in place. Let's configure Hibernate.

Right click on the "src" folder and select New --> File and paste following code there. Save it as hibernate.cfg.xml. Here you have to give the username and password according to your MySQL account. In my case username is root and there is no password.

```
<?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">com.mysql.jdbc.Drive
  <property name="connection.url">jdbc:mysql://localhost/userda
  <property name="connection.username">root</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.MySQLDialect</

  <!-- Enable Hibernate's automatic session context management
  <property name="current_session_context_class">thread</proper

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

  <!-- 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">update</property>

  <!-- Mapping files -->
  <mapping resource="user.hbm.xml"/>
  <mapping resource="task.hbm.xml"/>

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

5. Create Startup Helper Class

You have to startup Hibernate by creating a global org.hibernate.SessionFactory object and storing it somewhere for easy access in your application code. A org.hibernate.SessionFactory is used to obtain org.hibernate.Session instances. A org.hibernate.Session represents a single-threaded unit of work. The org.hibernate.SessionFactory is a thread-safe global object that is instantiated once. We will create a HibernateUtil helper class that takes care of startup and makes accessing the org.hibernate.SessionFactory more convenient.

To create new class right click on "com.hib" package and select New --> Class and give Name as HibernateUtil.java and paste the following code in class file.

```
package com.hib;

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

public class HibernateUtil {
 private static final SessionFactory sessionFactory = buildSess
 private static SessionFactory buildSessionFactory() {
  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."
   throw new ExceptionInInitializerError(ex);
  }
 }
```

```
 public static SessionFactory getSessionFactory() {
  return sessionFactory;
 }
}
```

6. Create Test Class to Load and Store Objects

Now it's time to do some real work using Hibernate. Following test class illustrate how we can Add, Retrieve, Update and Delete data in a MySQL database.

To create new class right click on "com.hib" package and select New --> Class and give Name as Test.java and paste the following code in class file.

```
package com.hib;

import java.util.Iterator;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class Test {

 /**
  * @param args
  */
 public static void main(String[] args) {

  Test tst = new Test();

  /**
   * adding records
   */
  tst.addUser("Saranga", "Rath");
  tst.addUser("Isuru", "Sampath");
  tst.addUser("Saranga", "Jaya");
  tst.addUser("Prasanna", "Milinda");

  tst.addTask(1, "Call", "Call Pubudu at 5 PM");
  tst.addTask(1, "Shopping", "Buy some foods for Kity");
  tst.addTask(2, "Email", "Send birthday wish to Pubudu");
  tst.addTask(2, "SMS", "Send message to Dad");
  tst.addTask(2, "Office", "Give a call to Boss");

  /**
   *  retrieving data
   */
  tst.getFullName("Saranga");

  /**
   * full updating records
   */
  User user = new User();
  user.setId(1);
  user.setFirstName("Saranga");
  user.setLastName("Rathnayake");
  tst.updateUser(user);

  /**
   * partial updating records
   */
  tst.updateLastName(3, "Jayamaha");

  /**
   * deleting records
   */
  User user1 = new User();
  user1.setId(4);
  tst.deleteUser(user1);
 }

 private void addUser(String firstName, String lastName) {

  Transaction trns = null;
  Session session = HibernateUtil.getSessionFactory().openSess
  try {
   trns = session.beginTransaction();

   User user = new User();

   user.setFirstName(firstName);
   user.setLastName(lastName);

   session.save(user);
```

```
71
72          session.getTransaction().commit();
73         } catch (RuntimeException e) {
74          if(trns != null){
75           trns.rollback();
76          }
77          e.printStackTrace();
78         } finally{
79          session.flush();
80          session.close();
81         }
82        }
83
84        private void addTask(int userID, String title, String descrip
85
86         Transaction trns = null;
87         Session session = HibernateUtil.getSessionFactory().openSess
88         try {
89          trns = session.beginTransaction();
90
91          Task task = new Task();
92
93          task.setUserID(userID);
94          task.setTitle(title);
95          task.setDescription(description);
96
97          session.save(task);
98
99          session.getTransaction().commit();
100        } catch (RuntimeException e) {
101         if(trns != null){
102          trns.rollback();
103         }
104         e.printStackTrace();
105        } finally{
106         session.flush();
107         session.close();
108        }
109       }
110
111       private void updateLastName(int id, String lastName) {
112        Transaction trns = null;
113        Session session = HibernateUtil.getSessionFactory().openSess
114        try {
115         trns = session.beginTransaction();
116         String hqlUpdate = "update User u set u.lastName = :newLastN
117         int updatedEntities = session.createQuery( hqlUpdate )
118         .setString( "newLastName", lastName )
119         .setInteger( "oldId", id )
120         .executeUpdate();
121
122         trns.commit();
123        } catch (RuntimeException e) {
124         if(trns != null){
125          trns.rollback();
126         }
127         e.printStackTrace();
128        } finally{
129         session.flush();
130         session.close();
131        }
132
133       }
134
135       private void updateUser(User user) {
136        Transaction trns = null;
137        Session session = HibernateUtil.getSessionFactory().openSess
138        try {
139         trns = session.beginTransaction();
140
141         session.update(user);
142
143         session.getTransaction().commit();
144        } catch (RuntimeException e) {
145         if(trns != null){
146          trns.rollback();
147         }
148         e.printStackTrace();
149        } finally{
150         session.flush();
151         session.close();
152        }
153       }
154
155       private void getFullName(String firstName) {
```

```
    Transaction trns = null;
    Session session = HibernateUtil.getSessionFactory().openSess
    try {
     trns = session.beginTransaction();
     List<User> users = session.createQuery("from User as u where
     .setString( "firstName", firstName )
     .list();
     for (Iterator<User> iter = users.iterator(); iter.hasNext()
      User user = iter.next();
      System.out.println(user.getFirstName() +" " + user.getLastN
     }
     trns.commit();
    } catch (RuntimeException e) {
     if(trns != null){
      trns.rollback();
     }
     e.printStackTrace();
    } finally{
     session.flush();
     session.close();
    }
   }

   private void deleteUser(User user) {
    Transaction trns = null;
    Session session = HibernateUtil.getSessionFactory().openSess
    try {
     trns = session.beginTransaction();

     session.delete(user);

     session.getTransaction().commit();
    } catch (RuntimeException e) {
     if(trns != null){
      trns.rollback();
     }
     e.printStackTrace();
    } finally{
     session.flush();
     session.close();
    }
   }
  }
```

If you have setup the database and followed the above steps correctly you should get following output.

```
Hibernate: insert into users (first_name, last_name) values (?
Hibernate: insert into users (first_name, last_name) values (?,
Hibernate: insert into users (first_name, last_name) values (?,
Hibernate: insert into users (first_name, last_name) values (?,
Hibernate: insert into tasks (user_id, title, description) value
Hibernate: insert into tasks (user_id, title, description) value
Hibernate: insert into tasks (user_id, title, description) value
Hibernate: insert into tasks (user_id, title, description) value
Hibernate: insert into tasks (user_id, title, description) value
Hibernate: select user0_.id as id0_, user0_.first_name as first1
Saranga Rath
Saranga Jaya
Hibernate: update users set first_name=?, last_name=? where id=?
Hibernate: update users set last_name=? where id=?
Hibernate: delete from users where id=?
```

Hibernate use Hibernate Query Language (HQL) which is powerful than SQL and fully object-oriented. You can get good knowledge about HQL by following HQL Documentation.

I have included the source files of this tutorial with the database. You can download it from here (Password : sara).

You might also like:

- Top 15 HTML Best Practices
- Create Web Service Using Apache Axis2 and Eclipse IDE
- Welcome

Posted by Saranga Rathnayaka at 9:20 PM

Labels: Hibernate, Java

65 comments:

Anonymous March 26, 2011 at 7:03 PM

Nice Article.....

I'm novel to Hibernate & your post gain me lot of practical knowledge in Hibernate...

I found another article named **Introduction to Hibernate** which gain the theoritical knowlegde for begginers.....

Reply

Adil Mukarram April 3, 2011 at 5:44 PM

Great Tutorial.
Searched internet for two weeks but was unable to run hibernate using other tutorials but your tutorial got me up and running in just an hour . Thanks
really great tutorial for beginners . I'd highly appreciate to see a post from you for "Integrating Jasypt with Hibernate 3" .

Reply

Saranga April 5, 2011 at 12:14 PM

@ Anonymous,
Thank you for sharing the article.

@ Adil Mukarram,
Thank you very much for your comment. We will try to create a tutorial for that.

Reply

Pondu April 12, 2011 at 10:07 AM

nice and easy to learn for beginners

Reply

babarathotmail April 20, 2011 at 5:12 PM

thanX a Lot for such nice article. It helped me a lot. I searched for days but didn't find any good tutorial using hib with eclipse. :)

Reply

gad.dan April 23, 2011 at 5:22 PM

Very good tutorial!!! I've been looking for something like that for sometime..

For beginners like me I would also add the stage of building the MySQL database and tables:

1. Connect to your account at MySQL.
2. Enter "CREATE DATABASE userdata;"
3. Enter "USE userdata;"
4. Enter "CREATE TABLE users(id int(10) AUTO_INCREMENT, first_name varchar(100), la st_name varchar(100), primary key(id));
5. ENTER "CREATE TABLE tasks(id int(10) AUTO_INCREMENT, user_id int(10), title varc har(100), description varchar(100), primary key(id));"

Also, if you need a password to access your MySQL database, you should provide it to the hibernate.cfg.xml file at the line HereShouldComeYourPassWord

Reply

Anonymous April 28, 2011 at 8:36 PM

Like++. Kudos to you Saranga. The clearest tutorial out there by far. Why couldn't the others make it this simple?