Chapter

3

SECTION II: GETTING STARTED

Writing The First Application

Hibernate when compared to other Java persistence solutions, is quite easy. In fact, it is considered the de-facto ORM library for most of the organizations today. Any project that requires database interaction have started looking at Hibernate than considering the traditional approach i.e. JDBC. This, therefore, saves a huge amount of time revolving around unnecessary chores.

However, to make this more convincible, let's look at a practical implementation and work on it.

This is the only compelling reason why this chapter has been introduced. This chapter aims at convincing the readers/developers about how easy it is, to begin using Hibernate.

This chapter does not cover the basics or a detailed explanation of Hibernate configurations or API. Those topics will be covered in the chapters that follow. This Chapter leads through the building of a small example called **GuestBook** that uses Hibernate.

To get the first Hibernate application to work, the following needs to be setup:

- Database
- Mapping files
- Configuration
- Plain old Java objects [POJOs]

Once all this is in place, the application logic needs to be written which uses the Hibernate session to actually do something.

Application Requirement Specifications

The application [example] to be built is called **GuestBook**. This application should be capable of accepting and displaying visitor's comments.

To achieve this, it should provide a user interface that accepts visitor's name and message/comments.

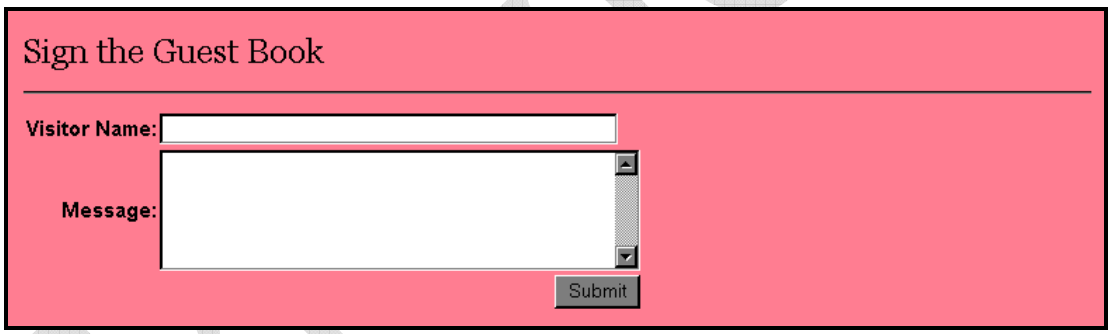


Diagram 3.1: GuestBook data entry form

After such information is captured and stored, other visitors to the application should be able to view all the available comments as shown in diagram 3.2.

This user interface displays the visitor's name along with the message and the date when the message was keyed in. It should also provide a link to sign the GuestBook which when clicked should display the GuestBook data entry form as shown in diagram 3.1.

Diagram 3.2: View GuestBook

Where Does Hibernate Fit

Hibernate can be invoked from a Java application either directly or via another framework such as Struts or Spring and so on.

Hibernate's API makes it easy for these frameworks to support Hibernate in one way or another.

Frameworks such as Spring / Struts provide excellent Hibernate integration including generic support for persistence objects, a generic set of persistence exceptions and transaction management.

Section V: Application Development With Hibernate explains how Hibernate can be configured and integrated with such frameworks.

Hibernate can also be invoked from:

- A Swing application
- A Servlet
- A Portlet
- A JSP
- Any other kind of Java application that has access to a database

Typically, Hibernate is used to create a Data Access Layer for an application.

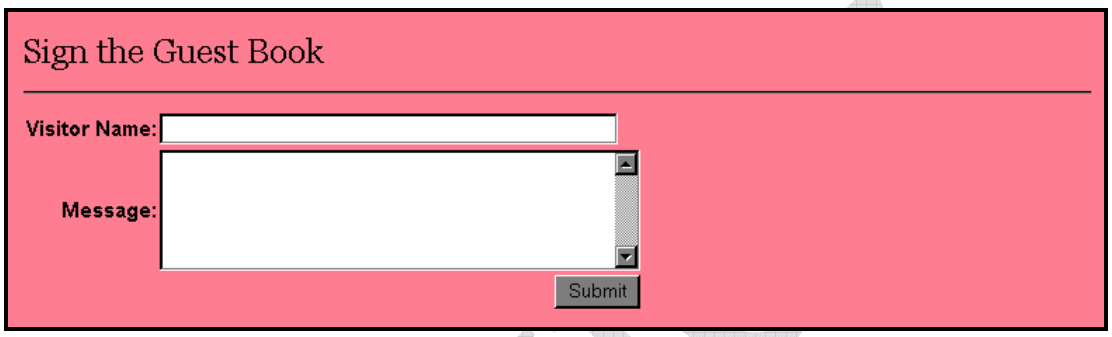
The most typical workflow would be:

- Define the configuration details. These details are then represented by a Configuration object
- Create a **SessionFactory** object from the Configuration object

- Instantiate the Session object through which the application accesses Hibernate's representation of the database

From this application's [GuestBook] point of view, Hibernate will be used as follows:

- The user invokes the application
- The "Sign the Guest Book" data entry form is served to allow capturing the visitor's name and comments



- The user keys in the details and clicks **Submit**

After such information is captured and the form is submitted, the server-side script in this case a JSP [GuestBookView.jsp] takes charge.

This script invokes Hibernate as follows:

- Creates a **SessionFactory** object from the Configuration object
- Instantiate the **Session** object
- Uses the **save()** method of the instantiated Session object to save the captured data to the configured database
- Uses the **createQuery()** method of the instantiated Session object to query the configured database and fetch all the entries to display them

This user interface displays the visitor's name along with the message and the date when the message was keyed in.

Let's begin!

Software Requirements

From the application development perspective, the following software will be required on the development machine**:**

- **J**ava **D**evelopment **K**it
- NetBeans IDE [The development IDE]
- MySQL Community Server [The database server]
- Hibernate 3 [The ORM tool]

Downloading Hibernate

Hibernate can be downloaded from**:**
http://www.hibernate.org/Download/DownloadOverview

From the download page that appears, choose to download the current latest release of **Hibernate Core**. At the time of writing this book the latest version that was available for download is **3.3.1.GA** [available in the Book's accompanying CDROM].

REMINDER

The above mentioned software setups are available in this book's accompanying CDROM.

Library Files

To integrate Hibernate with a Java application, a few Java libraries [**.**JAR] are required**:**

- **JDBC driver:** This will be specific to a relational database to be used. In this case MySQL is used as the database of choice, hence, the database specific JDBC driver file will be **MySQL Connector/J 5.1.7** [can be downloaded from http**:**//www**.**mysql**.**com also available in the Book's accompanying CDROM]

HINT

Hibernate does not include any database specific JDBC drivers. These must be obtained separately. Typically, the database provider offers them, as a separate downloads or bundled with the database installation.

- **Hibernate Library Files**
  These include**:**
  - From the **hibernate-distribution-3.3.1.GA** directory
    - hibernate3.jar
  - From the **hibernate-distribution-3.3.1.GA → lib → required** directory
    - antlr-2.7.6.jar
    - jta-1.1.jar
    - javassist-3.4.GA.jar
    - commons-collections-3.1.jar
    - dom4j-1.6.1.jar
  - From the **hibernate-distribution-3.3.1.GA → lib → bytecode → cglib** directory
    - hibernate-cglib-repack-2.1_3.jar
  - From the **hibernate-annotations-3.4.0.GA → lib** directory
    - slf4j-api.jar
    - slf4j-log4j12.jar
  - From the **hibernate-annotations-3.4.0.GA → lib → test** directory
    - log4j.jar

These files are available in the lib directory of the Hibernate Core and Hibernate Annotations download.

REMINDER

The Hibernate Core lib directory holds several optional library files. These libraries provide connection pools, additional caching functionality and the JCA API. The purpose of each library is detailed in the README file, which also states which libraries are optional and which are required.

The Application Development Approach

This application will be built using **JSP**.

The **data entry form** that captures the data will be called GuestBookEntry**.**jsp and the page that will fetch and display the entries will be called GuestBookView**.**jsp.

The **captured data** will be stored in a **table** called **GuestBook** under the **MySQL** database server.

In the Java application, the **POJO** that will represent the GuestBook database table will be called myApp**.**Guestbook**.**java.

Just to make this simple, the application development will be carried out/demonstrated using a development IDE called NetBeans IDE 6**.**5. Ensure that this IDE [available in the Book's accompanying CDROM] is installed on the development machine prior proceeding further.

Refer to *Appendix A: Installing The NetBeans IDE* for the installation steps.

The following are the steps that will help build this application.

1. Create the database schema
2. Create the Web Application using the NetBeans IDE
3. Add the Java libraries [Hibernate and JDBC driver] to the application
4. Create a POJO to represent the table in the database schema
5. Create a Hibernate [XML] configuration file that points to the database server [MySQL]
6. Create a Hibernate [XML] mapping files for the POJO that maps the JavaBean properties to the columns in the table
7. Add the Hibernate mapping file as a reference to the Hibernate configuration file
8. Create JSPs
    a. Create a Hibernate **Configuration object** that references the XML configuration file
    b. Build a Hibernate **SessionFactory** object from the Configuration object
    c. Finally, retrieve Hibernate **Session objects** from the SessionFactory and write the data access logic for the application to allow the operations CREATE and RETRIEVE

Creating Database And Tables In MySQL

Since MySQL is the database server of choice, ensure that the MySQL database engine [available in the Book's accompanying CDROM] is installed on the development machine prior proceeding further. This can also be downloaded from the website http://www.mysql.com/download.

Login to the MySQL database using a valid username and password. The pre-created default user called **root** can also be used.

Create the database named GuestBook**:**

```
CREATE DATABASE GuestBook;
```

Switch to the database GuestBook**:**

```
USE GuestBook;
```

Create the table named GuestBook**:**

```
CREATE TABLE GuestBook(
  VisitorNo Int PRIMARY KEY AUTO_INCREMENT,
  VisitorName varchar(50),
  Message varchar(100),
  MessageDate varchar(40));
```

Creating A Web Application

Since NetBeans is the IDE of choice throughout this book. Use it to create a new Web Application Project called **GuestBook**.

Run the NetBeans IDE and create a new **Web Application** project, as shown in diagram 3.3.1.

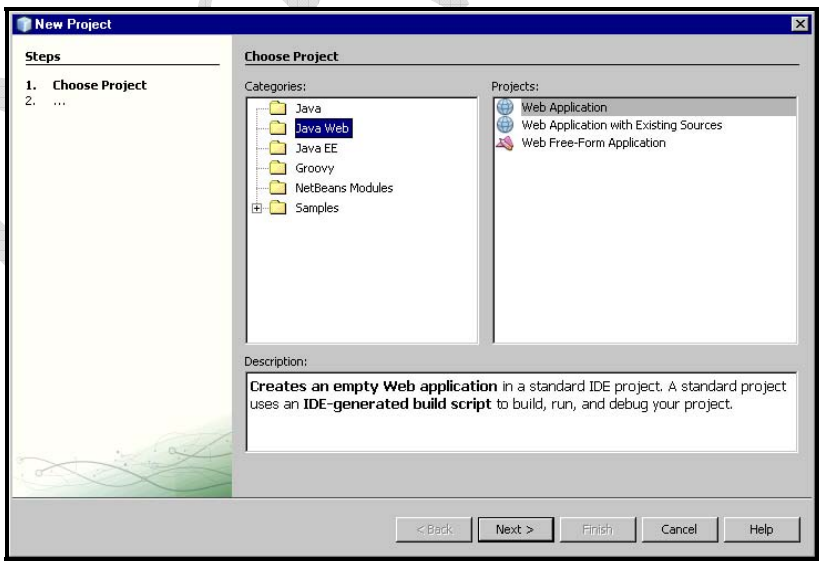


Diagram 3.3.1: New Project

Click  . Do not choose a framework, in the Frameworks dialog box.

Click  .

The GuestBook application is created in the NetBeans IDE as shown in diagram 3.3.4.

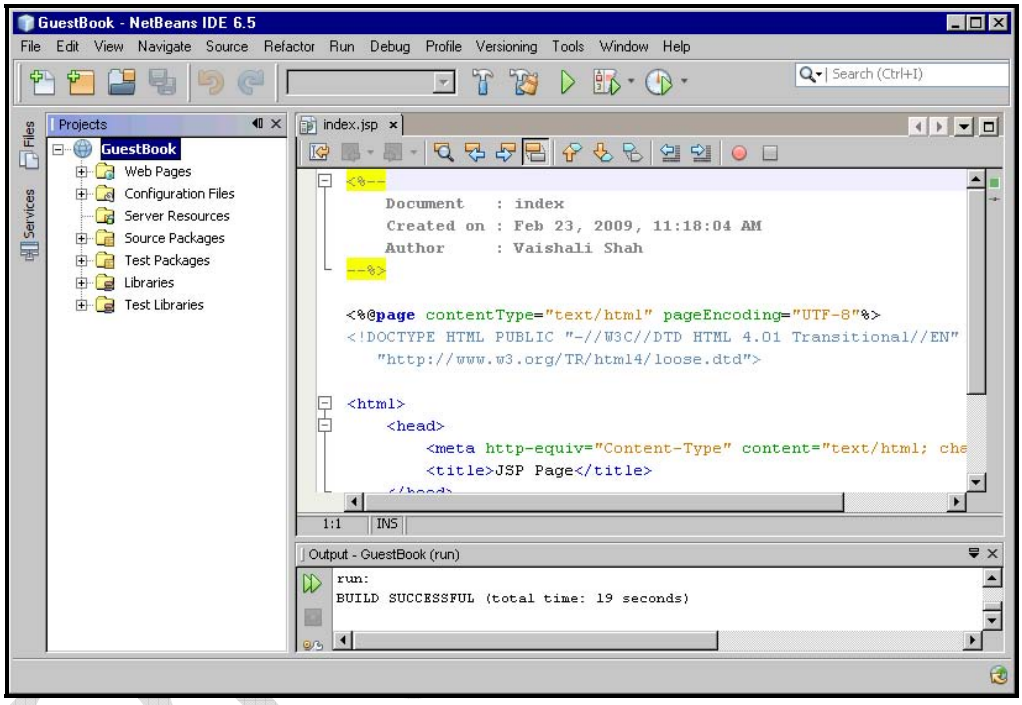


Diagram 3.3.4: GuestBook in NetBeans IDE

Once the NetBeans IDE brings up the GuestBook application, the next step is to add the required library files [JDBC driver and Hibernate] to the GuestBook application.

Adding The Required Library Files

*It's a good practice to manually create a dedicated **lib** folder with all the required library files in the project folder and then using NetBeans add libraries from this folder as the source.*

To do so,

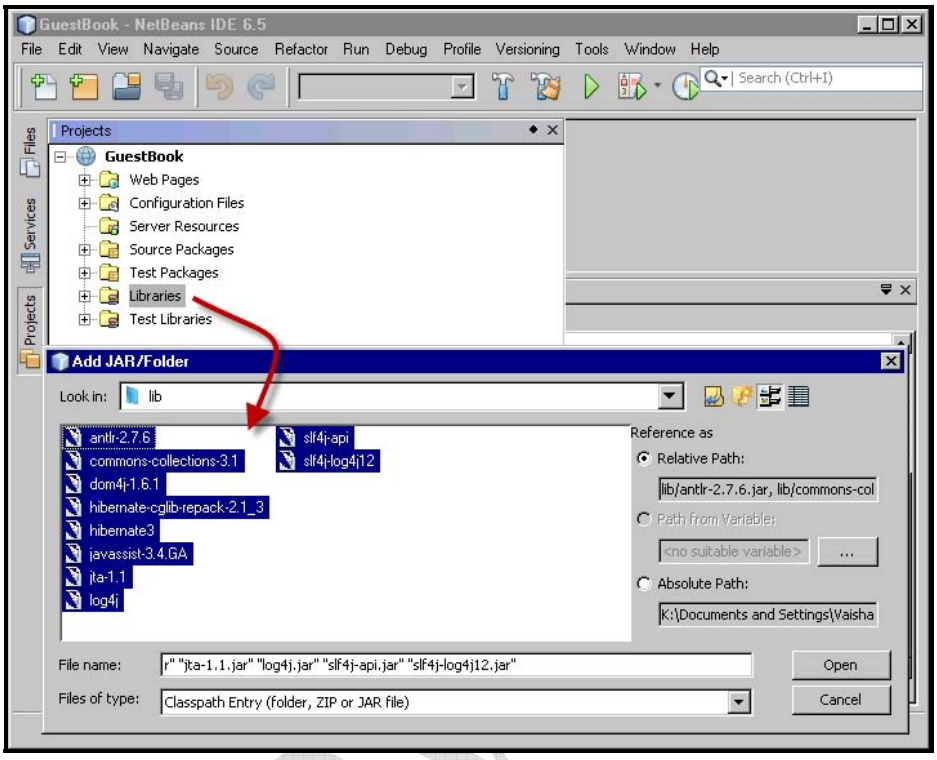


Diagram 3.4.2: Add Jar/Folder dialog box

After adding the **required JAR files**, the **Libraries** directory now appears as shown in diagram 3.4.3**:**

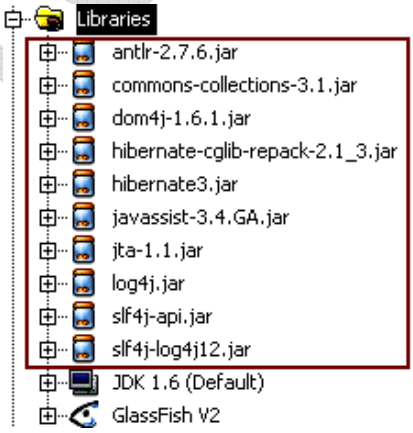


Diagram 3.4.3: Libraries folder

JDBC Driver For MySQL

MySQL provides connectivity to client applications developed in the Java EE 5 via a JDBC driver named **MySQL Connector/J**.

MySQL Connector/J is a native Java driver that converts JDBC calls into the network protocol used by the MySQL database. MySQL Connector/J is a Type 4 driver, which means that MySQL Connector is pure Java code spec and communicates directly with the MySQL server using the MySQL protocol.

MySQL Connector/J allows the developers working with Java EE 5, to build applications, which interact with MySQL and connect all corporate data even in a heterogeneous environment.

Visit the site http://www.mysql.com to download the MySQL Connector/J JDBC Driver.

At the time of writing this book the latest version of the **MySQL Connector/J** was **5.1.7** [available in this Book's accompanying CDROM].

After it is downloaded, using any unzip utility such as Winzip unzip the contents of the zip file.

Copy the **mysql-connector-java-X.X.X-bin.jar** library file to the **lib** directory created earlier under <Drive>:\NetBeansProjects\GuestBook to store the JDBC library file.

Using NetBeans IDE add this library file to the project. Right-click on the **Libraries** directory, click the **Add JAR/Folder...** menu item.

Clicking the **Add JAR/Folder** file displays the dialog box to choose the JAR files as shown in diagram 3.4.4. Browse to the **lib** directory and select **mysql-connector-java-X.X.X-bin.jar** JAR file to add to the project.

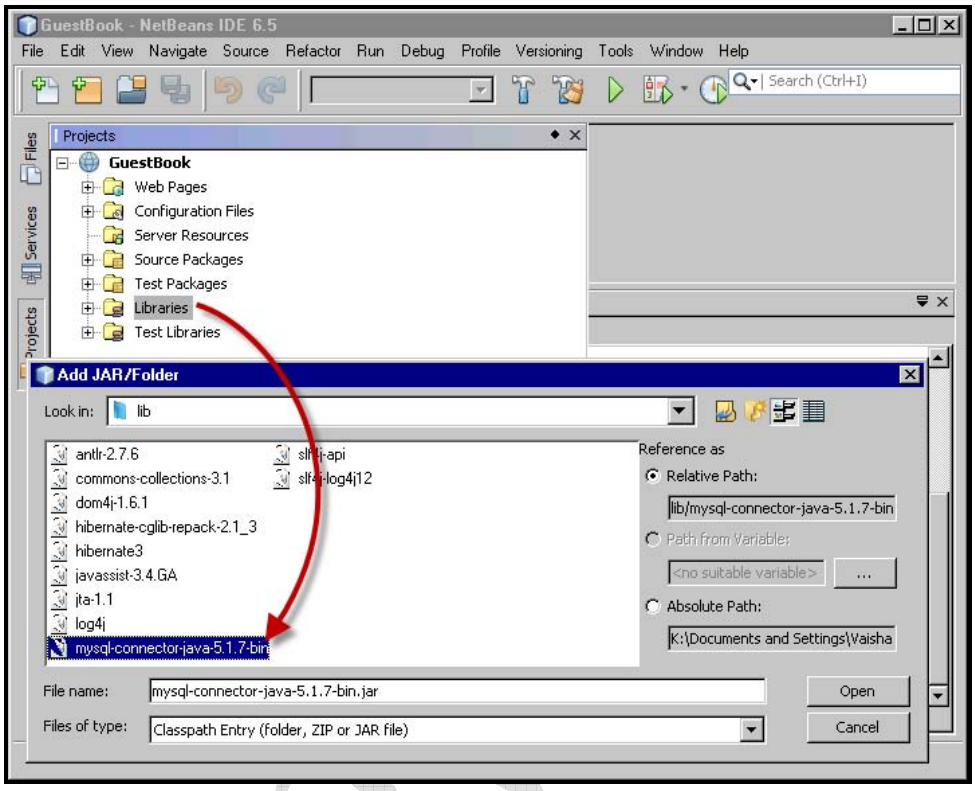


Diagram 3.4.4: Add Jar/Folder dialog box

This adds the JDBC driver file [mysql-connector JAR file] to the project.

Now, let's move to the application development area.

Creating A JavaBean Class

To hold the captured data in a structured manner, a bean class is required. This class should expose the following properties:

Property Name	To Store
visitorNo	The Primary Key value
visitorName	Visitor's Name
message	Message that the visitor enters
messageDate	The date/time on which the message was entered

The class should have a parameterized constructor that allows setting captured values to these properties.

The primary purpose of having such a class is to hold individual guestbook entry as and when they are captured.

The following are the steps to create the Bean class using the NetBeans IDE**:**

1. Right click **Sources Package** directory, select **New → Java Class…** as shown in diagram 3.5.1

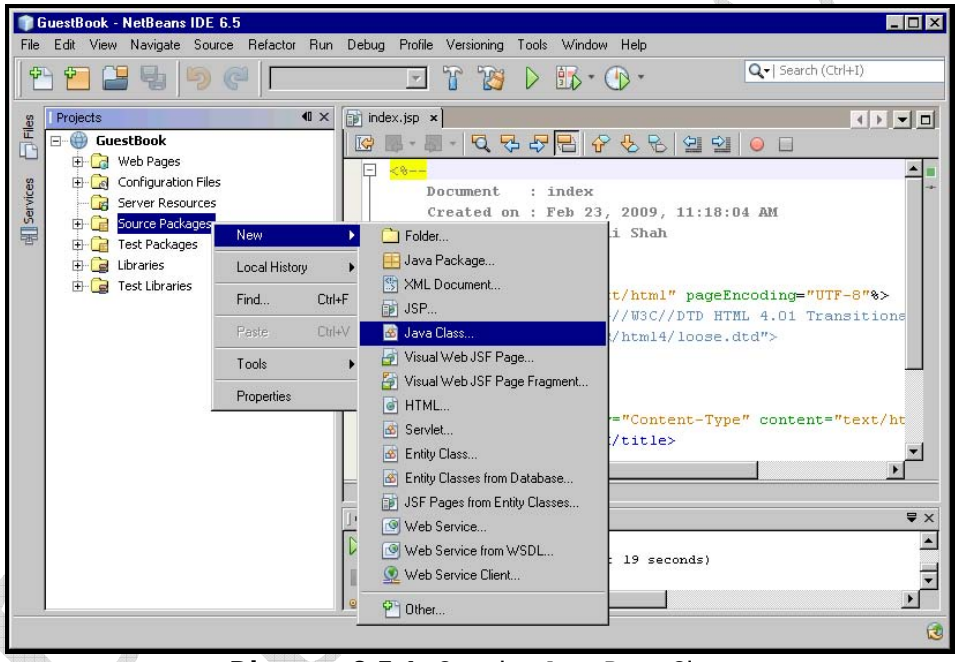


Diagram 3.5.1: Creating Java Bean Class

2. Enter **Guestbook** in the Class Name textbox and enter **myApp** in the Package textbox as shown in diagram 3.5.2

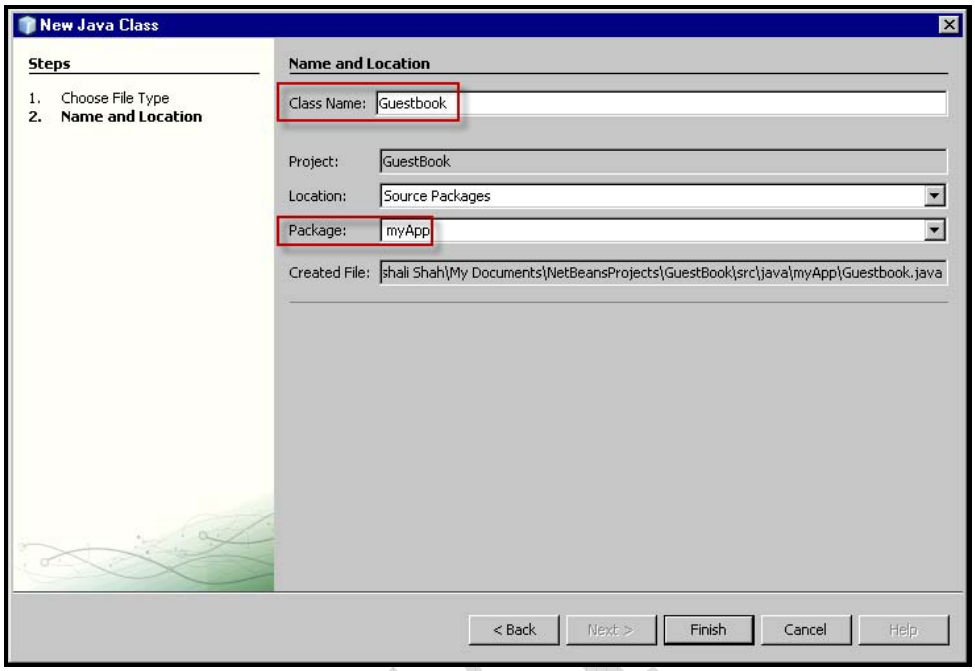


Diagram 3.5.2: Naming the Java class file

3. Click **Finish**

This creates the bean class named Guestbook.java under the package called **myApp**.

Guestbook.java [Code Spec]

Edit the **Guestbook.java** file with the following contents.

```
1  package myApp;
2
3  public class Guestbook implements java.io.Serializable {
4      private Integer visitorNo;
5      private String visitorName;
6      private String message;
7      private String messageDate;
8
9      public Guestbook() {
```

Creating Hibernate Configuration File

Hibernate uses the **hibernate.cfg.xml** file to create the connection and setup the required environment. This file is used to provide the information which is necessary for making database connections.

The **hibernate.cfg.xml** configuration file defines information such as**:**

- The database connection
- Resource mappings

To do so, right-click the **Source Packages** folder. Select **New → Other…** as shown in diagram 3.6.1.

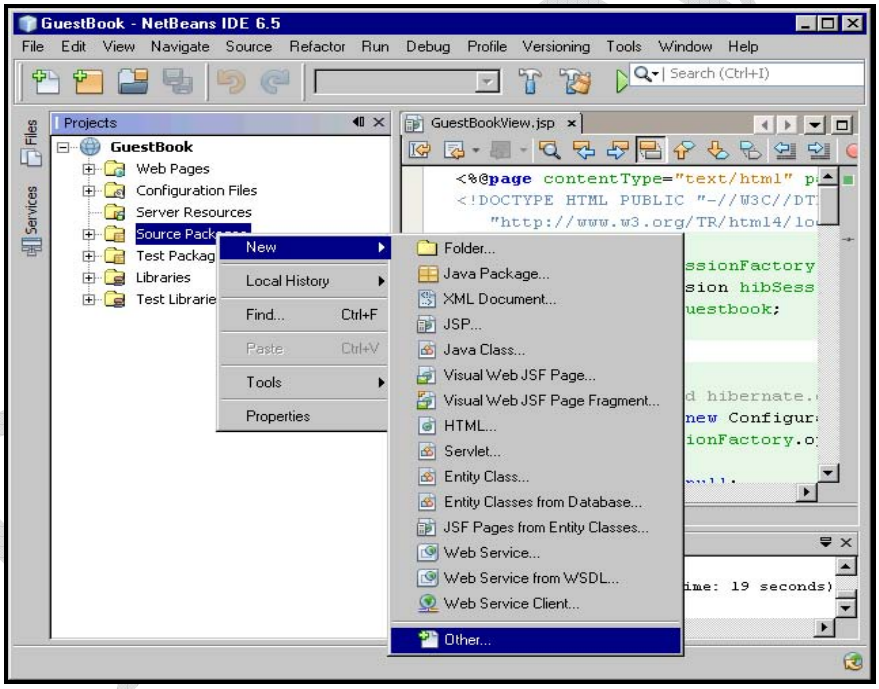


Diagram 3.6.1: Creating hibernate.cfg.xml file

New File dialog appears. Select **XML** from the **Categories** list and select **XML Document** from the **File Types** as shown in diagram 3.6.2.

connection.username: Is the username used to connect to the database

- **connection.password:** Is the password used to authenticate the username

The connection properties are common to any Java developer who has worked with JDBC in the past.

- **dialect:** Is the name of the SQL dialect for the database

The **dialect** property informs the Hibernate framework whether the given database supports identity columns, altering relational tables and unique indexes, among other database specific details.

HINT

Hibernate ships with more than 20 SQL dialects supporting each of the major database vendors including Oracle, DB2, MySQL and PostgreSQL.

Creating Hibernate Mapping File

Mapping definitions [also called mapping documents] are used to provide Hibernate with information to persist objects to a relational database. The mapping files also provide support features such as creating the database schema from a collection of mapping files.

In Hibernate, mapping a bean to a relational database is done by creating a mapping file in XML.

In this application, a Guestbook.hbm.xml file will be created that holds mappings between the database, table and column names to the properties in the **Guestbook** bean [created earlier].

This provides a one-to-one correspondence between a bean to be mapped and the Hibernate configuration file.

The naming convention for mapping files is to use the name of the persistent class with the **.hbm.xml** extension. In the Guestbook application, the persistent class is named Guestbook. The mapping file for the Guestbook class is thus named Guestbook.hbm.xml.

To do so, right-click the **myApp** directory. Select **New → Other…** as shown in diagram 3.7.1.

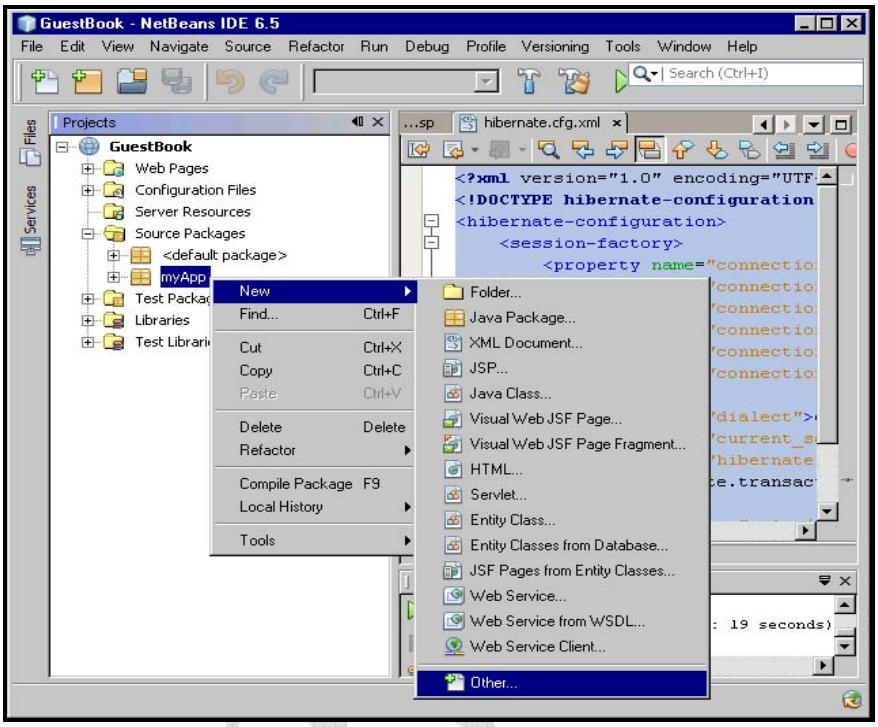
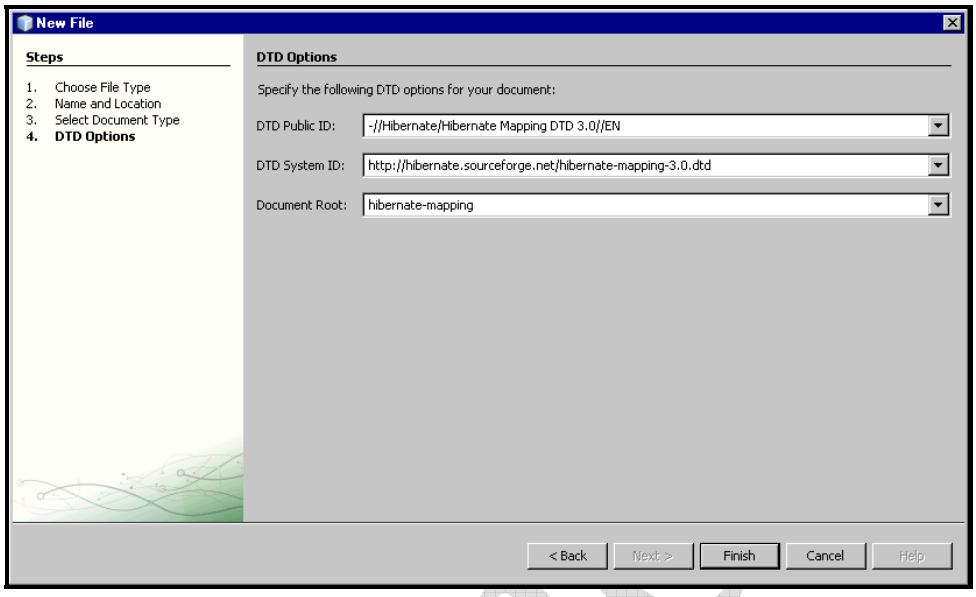


Diagram 3.7.1: Creating GuestBook.hbm.xml file

New File dialog appears. Select **XML** from the **Categories** list and select **XML Document** from the **File Types** as shown in diagram 3.7.2.



```
18          </property>
19      </class>
20  </hibernate-mapping>
```

Explanation:

This mapping document informs the following to the Hibernate**:**

- The **Guestbook** class is to be persisted to the **guestbook** table
- The identifier property **visitorNo** maps to a column named **VisitorNo**
  - The Primary Key value will be generated by MySQL [generator class="native"]
- The text properties **visitorNo**, **visitorName**, **message**, **messageDate** map to columns **VisitorNo**, **VisitorName**, **Message**, **MessageDate** respectively

Adding A Mapping Resource

Before a session factory is created, Hibernate must be informed about the mapping files that define how the Java classes relate to the database tables.

Now that the mapping file is available, the same can be informed to the Hibernate framework. This can be done by adding a <mapping> tag to the **hibernate.cfg.xml** file [created earlier].

Edit the **hibernate.cfg.xml** file to hold a mapping resource as shown below**:**

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC '-//Hibernate/Hibernate Configuration DTD//EN'
   'http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd'>
3  <hibernate-configuration>
4      <session-factory>
5          <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
6          <property name="connection.url">jdbc:mysql://localhost:3306/GuestBook</property>
7          <property name="connection.username">root</property>
8          <property name="connection.password">12345678</property>
9          <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
10         <mapping resource="myApp/Guestbook.hbm.xml"/>   <-- Add this
11     </session-factory>
12 </hibernate-configuration>
```

Explanation:

Hibernate also needs to know the location and names of the mapping files describing the persistent classes. The mapping element provides the name of each mapping file as well as its location relative to the application classpath. Mapping file **Guestbook.hbm.xml** is included in the configuration file.

Creating JSPs

Before creating the JSP, let's create a directory to hold JSP.

The following are the steps to create the directory**:**

1. Right click **Web Pages** directory, select **New → Folder…** as shown in diagram 3.8.1

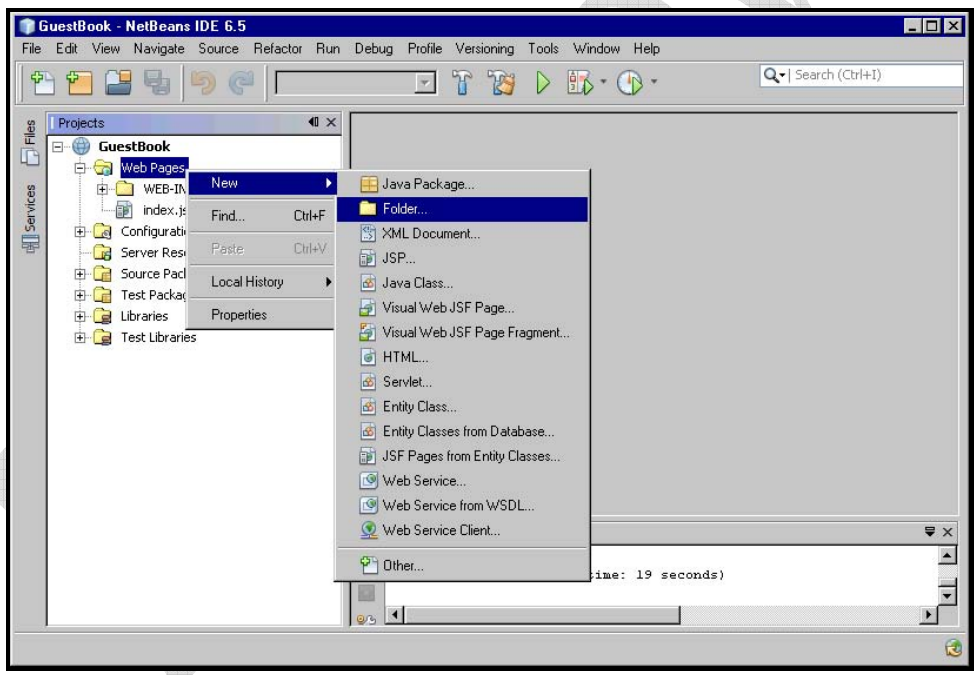


Diagram 3.8.1: Creating Folder

2. Enter the name **JSP** in the **Folder Name** textbox as shown in diagram 3.8.2

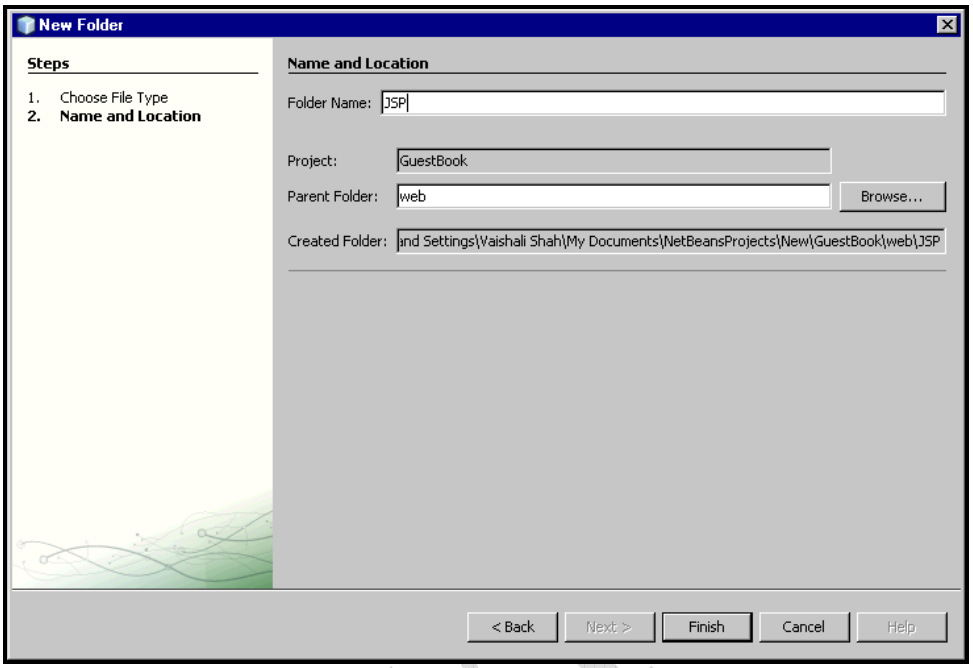


Diagram 3.8.2: Naming the folder

3. Click **Finish**

This creates the directory named **JSP** under **Web Pages**.

The following are the steps to create the JSP file**:**

1. Right click **JSP** directory, select **New → JSP…** as shown in diagram 3.8.3

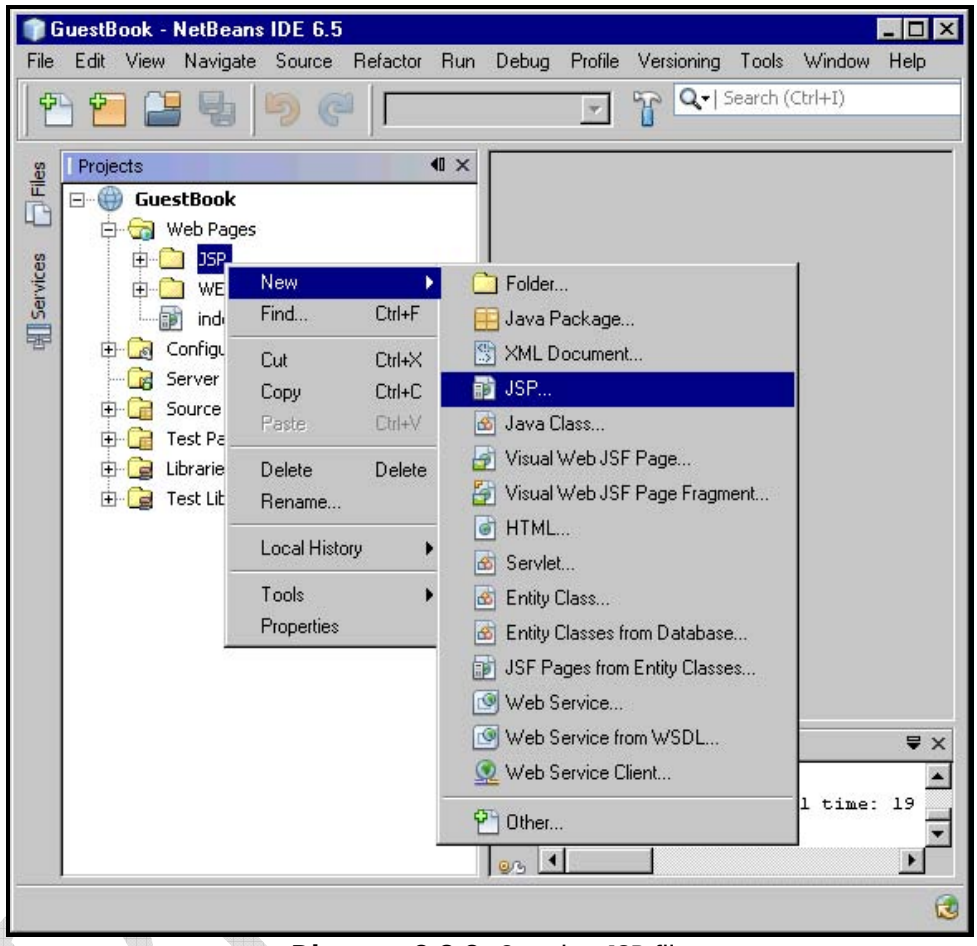


Diagram 3.8.3: Creating JSP file

2. Enter the name **GuestBookEntry** in the **JSP File Name** textbox as shown in diagram 3.8.4

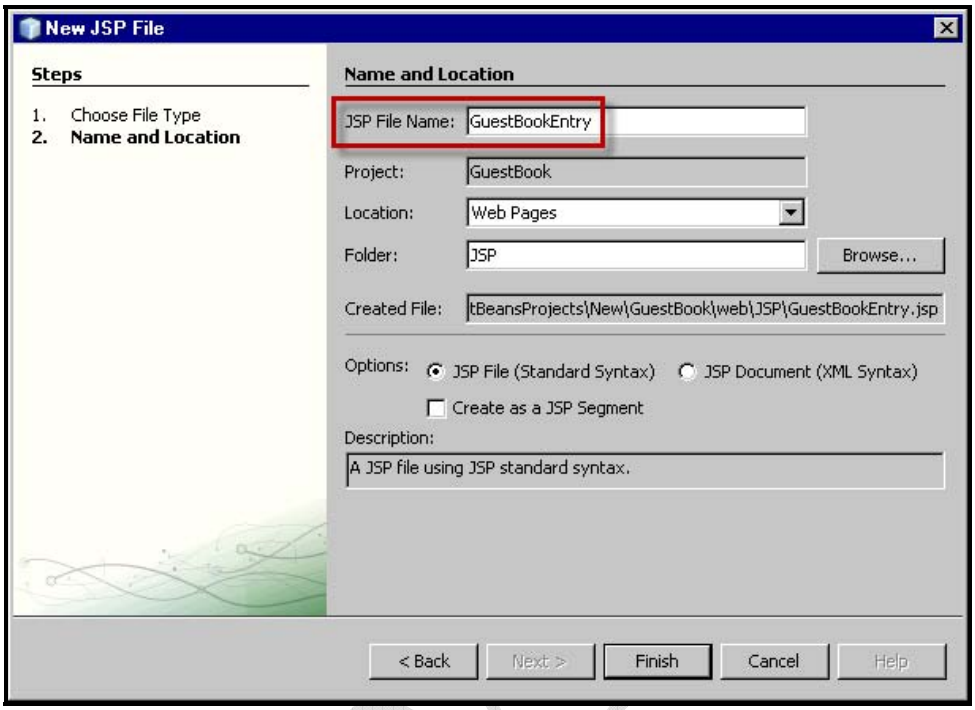


Diagram 3.8.4: Naming the JSP file

3. Click **Finish**

This creates the JSP named GuestBookEntry.jsp under the JSP directory created earlier.

GuestBookEntry.jsp [Code Spec]

Edit the **GuestBookEntry.jsp** file with the following contents.

```
1  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
3  <html>
4     <head>
5        <meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-1">
6        <title>Guest Book</title>
7     </head>
8     <body bgcolor="pink">
9        <table border="0" cellpadding="0" cellspacing="0" align="center" width="760">
```

```
45                      </tr>
46                      <tr>
47                          <td colspan="2" align="right">
48                              <input type="submit" name="btnSubmit" value="Submit" />
49                          </td>
50                      </tr>
51                  </table>
52              </form>
53          </td>
54      </tr>
55    </table>
56  </body>
57 </html>
```

Explanation:

This is the data entry form that allows capturing the visitor's name and comments and submits the data to another server-side script called GuestBookView.jsp for further processing.

GuestBookView.jsp [Code Spec]

Using NetBeans create one more JSP called GuestBookView.jsp using the same steps as shown earlier.

Edit the **GuestBookView.jsp** file with the following contents.

```
1  <%@page contentType="text/html" pageEncoding="UTF-8"
   import="org.hibernate.SessionFactory, org.hibernate.cfg.Configuration,
   org.hibernate.Session, org.hibernate.Transaction, java.util.List, java.util.Iterator,
   myApp.Guestbook" %>
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3    "http://www.w3.org/TR/html4/loose.dtd">
4  <%!
5      SessionFactory sessionFactory;
6      org.hibernate.Session hibSession;
7      List<Guestbook> guestbook;
8  %>
9
10 <%
11     sessionFactory = new Configuration().configure().buildSessionFactory();
12     hibSession = sessionFactory.openSession();
```

```
88                      </td>
89                    </tr>
90                  <%
91                    }
92                  %>
93                  </table>
94                </td>
95              </tr>
96            </table>
97          </body>
98        </html>
```

Explanation:

This is a server-side script that saves the captured data, fetches all the available entries and displays them.

Imports

```
1  <%@page contentType="text/html" pageEncoding="UTF-8"
   import="org.hibernate.SessionFactory, org.hibernate.cfg.Configuration,
   org.hibernate.Session, org.hibernate.Transaction, java.util.List, java.util.Iterator,
   myApp.Guestbook" %>
```

To allow all this, the following interfaces/classes are imported:

- org.hibernate.SessionFactory: Allows creating sessions. The SessionFactory caches generate SQL statements and other mapping metadata that Hibernate uses at runtime
- org.hibernate.cfg.Configuration: Is used to configure and bootstrap Hibernate. It is meant only as a initialization-time object. The application uses a Configuration instance to specify the location of mapping documents and Hibernate-specific properties and then create the SessionFactory
- org.hibernate.Session: Is the main runtime interface between a Java application and Hibernate. This is the central API class abstracting the notion of a persistence service. The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes
- org.hibernate.Transaction: Is a package, which abstracts the underlying transaction mechanism [JTA or JDBC] and provides strategies for obtaining application server TransactionManagers
- java.util.List: Is an ordered collection. The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index [position in the list] and search for elements in the list

- java.util.Iterator: Is used to sequence over a collection of objects. Iterator allows the caller to remove elements from the underlying collection during the iteration with well-defined semantics

Variable Declaration

The following variables are declared:

```
4  <%!
5      SessionFactory sessionFactory;
6      org.hibernate.Session hibSession;
7      List<Guestbook> guestbook;
8  %>
```

- To represent the connections to the database, variable named **sessionFactory** is declared
- To create an instance of Session, variable named **hibSession** is declared
- A list of type **Guestbook** is created. This will hold the entries for view purposes

Application Logic

Creating A Configuration Instance And Building A Session Factory

```
11      sessionFactory = new Configuration().configure().buildSessionFactory();
```

An instance of the Configuration interface is created. Using it's **configure()** method, the session factory is built. This step indicates Hibernate to load the **hibernate.cfg.xml** file.

HINT

The default name of the configuration file is **hibernate.cfg.xml**. If this is changed, the new file name needs to be passed explicitly as an argument to the **configure()** method.

The configure() method returns an instance of Configuration, which can be used to obtain a Hibernate SessionFactory instance by calling the **buildSessionFactory()** method.

The **buildSessionFactory()** method decides about various SQL statements that must be used to access the data and creates the **SessionFactory** object, which is stored in the private variable declared earlier.

Obtaining A Session Object

After obtaining the SessionFactory, Hibernate org.hibernate.Session objects can be retrieved.

```
12      hibSession = sessionFactory.openSession();
```

The **openSession()** method of the SesisonFactory interface creates an instance of Session. This instance represents the primary interface to the Hibernate framework

HINT

All the persistence operations are performed using Session objects.

A typical application will usually have:

- A single **Configuration** object, which will only be used in initialization
- One **SessionFactory** object that will exist throughout the life cycle of the application

The application will typically ask this SessionFactory object for a Session, retrieve an object, make the desired property changes and then persist it, all within one session. Finally, the application will close the Session object

Instantiating A Transaction

```
13      Transaction tx = null;
```

An empty instance of **Transaction** is created

Instantiating The JavaBean class [POJO]

```
14      Guestbook gb = new Guestbook();
```

An object of the GuestBook **JavaBean** class is created

Persisting Data

To insert the captured data:

```
18          tx = hibSession.beginTransaction();
```

- An instance of the Transaction class is created by invoking the **beginTransaction()** method of the Session interface

```
20          String guest = request.getParameter("guest");
21          String message = request.getParameter("message");
```

- The data captured is stored in variables via the getParameter() method

```
22        String messageDate = new java.util.Date().toString();
```

- The current date is set when the visitor entered data in the Guest Book

```
23        gb.setVisitorName(guest);
24        gb.setMessage(message);
25        gb.setMessageDate(messageDate);
```

- The variable values are then set in the Guestbook list via the setter methods of the JavaBean class

```
27        hibSession.save(gb);
```

- The session interface then saves the Guestbook list by invoking the **save()** method. The save() method of the session object allows saving the information to the database table. When an object is passed to the **save()** method, Hibernate reads the state of the variables of that object and executes the required SQL query
- Hibernate would automatically create and fire the INSERT query**:**

```
INSERT INTO GuestBook (VisitorName, Message, MessageDate)
  VALUES (visitorName, message, messageDate);
```

```
28        tx.commit();
```

- Finally, the transaction is committed

```
31        catch (RuntimeException e) {
32            if(tx != null) tx.rollback();
33            throw e;
34        }
```

- In case of errors, if any, it is determined if the transaction object is empty. If not, then the transaction is rolled backed

Retrieving Data

To view the data captured via the GuestBookView.jsp file**:**

```
38        hibSession.beginTransaction();
```

- An instance of the Transaction class is created by invoking the **beginTransaction()** method of the Session interface

```
39        guestbook = hibSession.createQuery("from Guestbook").list();
```

- Using **createQuery()** method of the Session object the persistent objects are retrieved. HQL statements are object-oriented, meaning that the query on object properties instead of database table and column names. The createQuery() method returns a collection of all **GuestBook** instances
- Hibernate would automatically create and fire the SELECT query:

```
SELECT * FROM GuestBook;
```

```
42      catch (RuntimeException e) {
43          throw e;
44      }
```

- In case of errors, if any, is trap in the catch block

```
46      hibSession.close();
```

- It is a good practice to close the Session when all the work for a transaction is completed

Displaying Retrieved Data

The displaying of the records fetched is handled by:

```
77              <%
78                  Iterator iterator = guestbook.iterator();
79                  while (iterator.hasNext()) {
80                      Guestbook objGb = (Guestbook) iterator.next();
81              %>
82                  <tr>
83                      <td style="font:12px/16px Georgia; color:#786e4e;">
84                          On <%=objGb.getMessageDate()%>,<br />
85                          <b><%=objGb.getVisitorName()%>:</b>
86                          <%=objGb.getMessage()%>
87                          <br /><br />
88                      </td>
89                  </tr>
90              <%
91                  }
92              %>
```

Here, an iterator is used to traverse through the List object called guestbook. This list object was populated earlier by:

```
39          guestbook = hibSession.createQuery("from Guestbook").list();
```

Using <TABLE>, <TR> and <TD> the elements of the List object are placed.

Editing The web.xml File

In NetBeans, by default, the web.xml file uses the index.jsp file as the welcome file i.e. whenever the application is run the web.xml file will display the index.jsp file.

This file needs to be edited to invoke the application's data entry form [GuestBookEntry.jsp] every time it's invoked.

Edit the web.xml file with following contents:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
3      <session-config>
4          <session-timeout>
5              30
6          </session-timeout>
7      </session-config>
8      <welcome-file-list>
9          <welcome-file>index.jsp</welcome-file>
9          <welcome-file>JSP/GuestBookEntry.jsp</welcome-file>
10     </welcome-file-list>
11 </web-app>
```

Replace this with this

Running The GuestBook Application

Now that the application is ready, let's run this application [source code available on this Book's accompanying CDROM].

Begin by **building** the project, using the NetBeans IDE.

To do so, right click the **GuestBook project** and select the **Build** menu item as shown in diagram 3.9.1.

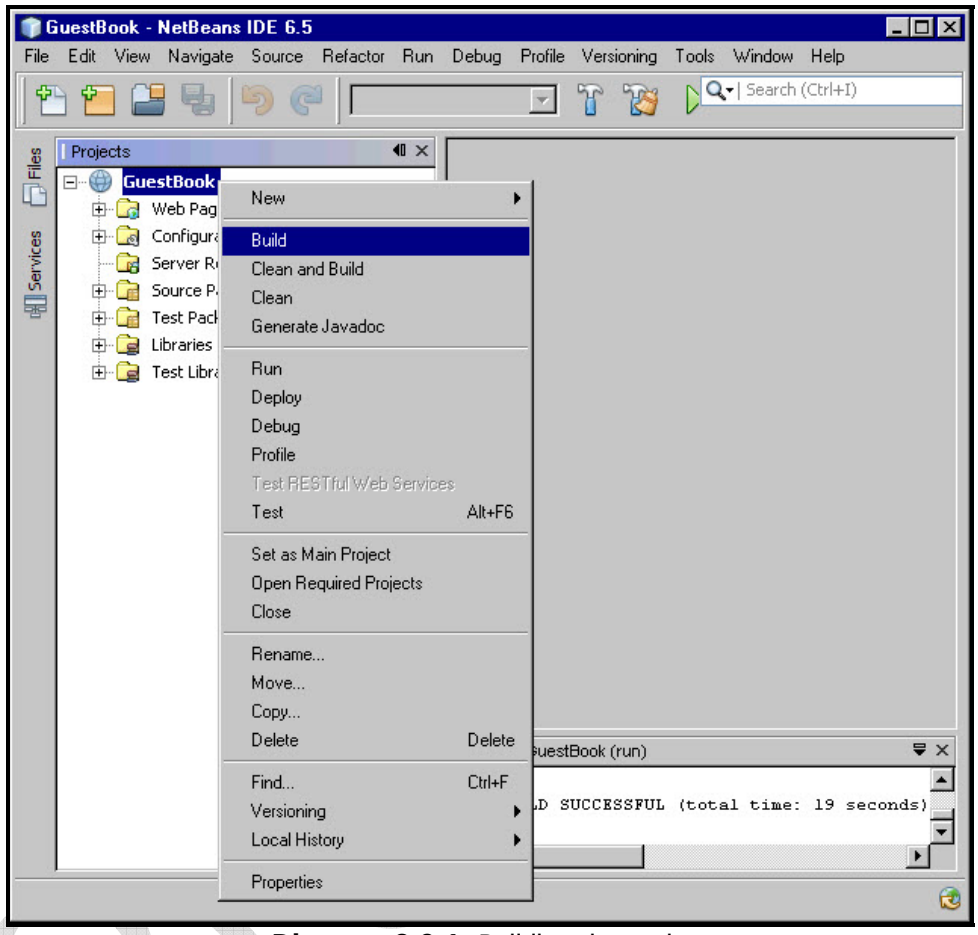


Diagram 3.9.1: Building the project

Then run the application by right clicking the **GuestBook project** and selecting the **Run** menu item as shown in diagram 3.9.2.

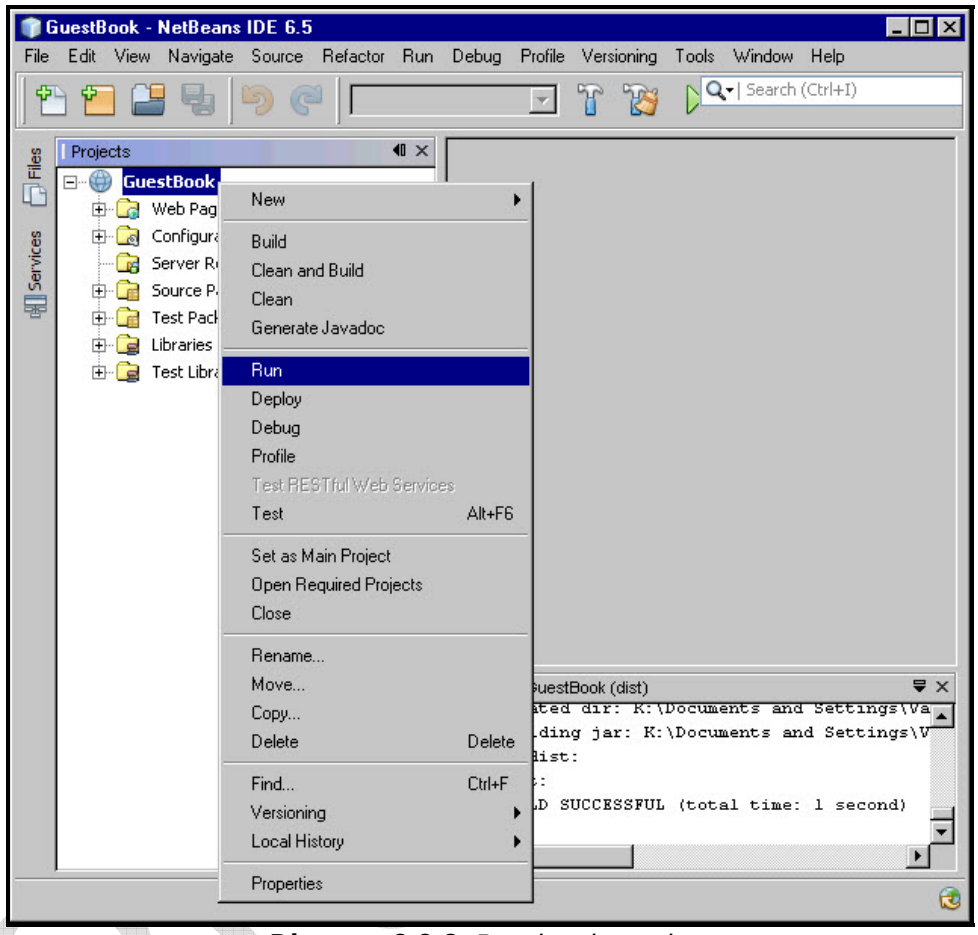


Diagram 3.9.2: Running the project

The GuestBookEntry.jsp page is served in the Web browser as shown in diagram 3.9.3.

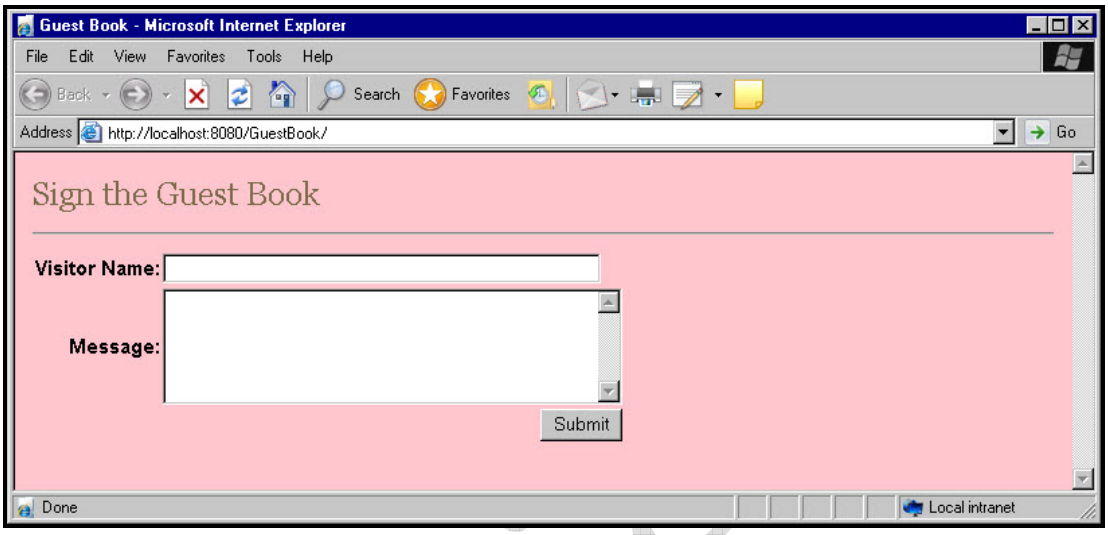


Diagram 3.9.3: The application run in the Web browser

Enter the name and the comments in the Name and Message fields as shown in diagram 3.9.4.

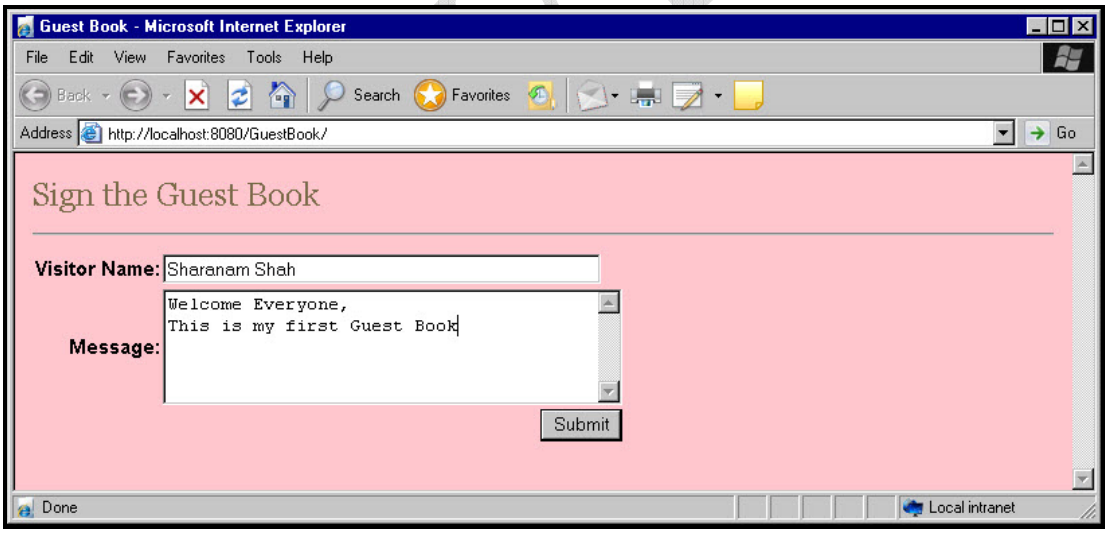


Diagram 3.9.4: Entering data

Click Submit. This displays the already existing messages [entered by others who visited the site before] along with the newly added message.

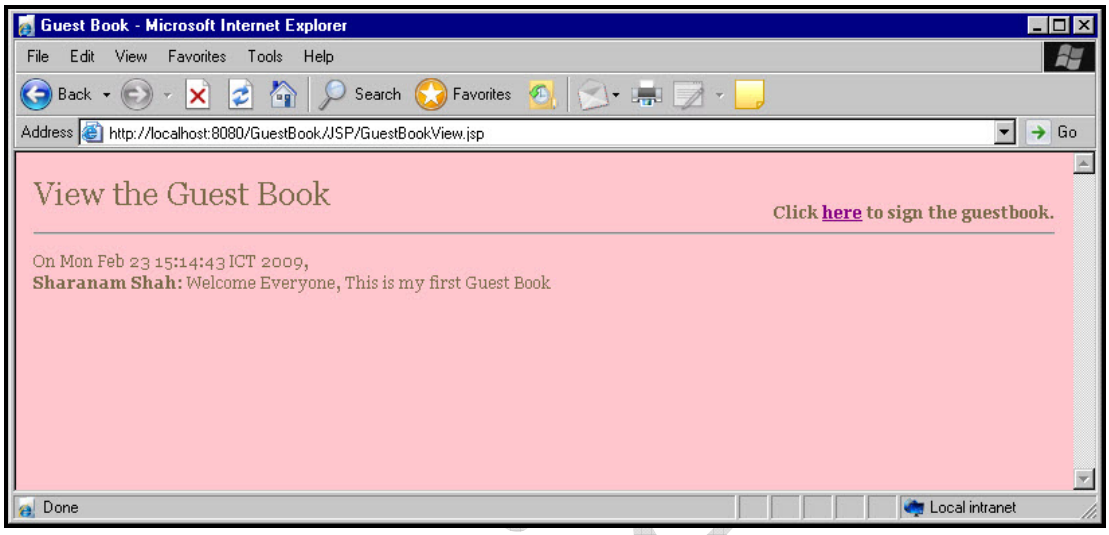


Diagram 3.9.5: Viewing data

When the GuestBookView.jsp page appears, the data entered by the user is stored in the MySQL database table named GuestBook that was created earlier.

To ensure that this was done successfully, open MySQL command line utility and query the table to view the following output:

```
+-----------+---------------+-----------------------------------------+---------------------------+
| VisitorNo | VisitorName   | Message                                 | MessageDate               |
+-----------+---------------+-----------------------------------------+---------------------------+
|         1 | Sharanam Shah | Welcome Everyone, This is my first Guest Book | Mon Feb 23 15:14:43 ICT 2009 |
+-----------+---------------+-----------------------------------------+---------------------------+
1 row in set (0.00 sec)
```

Click **here** link available on the top right corner of the page to go back to the Guest Book data entry form.

This chapter dealt with building a web application and integrating Hibernate into that application that exemplifies the core Hibernate concepts discussed in the first two chapters.

The next chapter demonstrates building the same application using Hibernate plugins [reverse engineering from database tables] available in the NetBeans IDE. This approach helps reduce a lot of manual code spec.

The Book CDROM holds the complete application source code built using the NetBeans IDE for the following application**:**

- GuestBook_Chap03

This can be directly used by making appropriate changes [username/password] to the configuration file.