Appendix

The relationship between the CLASSPATH and packages is explained. When complex applications are developed that require the use of many additional packages, it is necessary to understand how Java finds packages at runtime.

With any complex software package, there can be difficulties using it. There can be conflicts between separate packages that lead to memory leaks. Some of these problems can be avoided by moving the packages to another location. Some need to be modified. Two packages need to have methods called to avoid memory leaks and errors. The details of memory leaks are explained in this Appendix.

Hibernate can be used without any knowledge of SQL. For those that still want to see what Hibernate is doing on the database server, simple commands for the MySQL database server have been explained. Using these commands it will be possible to log onto the server and list the contents of the tables that have been created by Hibernate.

The remainder of the Appendix lists the contents of the auxiliary classes that were used in the book. The helper classes use many standard Java techniques to implement some of the features in the book. A detailed explanation of these techniques belongs in a book on Java. Hopefully, the contents of these files are easy to understand. The Javadoc statements have been removed from these files, but can be accessed at http://bytesizebook.com/book/doc.

A.1 Classpath and Packages

When using Java, it is important to understand the concepts of the classpath and packages. The two concepts are intertwined; one will not make any sense until the other is understood. When Java looks for packages, it searches the classpath.

A.1.1 Usual Suspects

There is a great scene at the end of the movie *Casablanca*. Humphrey Bogart has just killed the German Commander in front of the Chief of Police. The Chief then calls his office and informs them that the Commander has been murdered and that they should round up the usual suspects.

For Java, the CLASSPATH variable is a list of the usual suspects. When Java wants to find a class file, it searches through all of the directories that are listed in the CLASSPATH. In order to have Java look in new places, just add more paths to the CLASSPATH variable.

For example, suppose the CLASSPATH contains

1. `/myData`
2. `/myFiles`
3. `/myStuff`

Java will check the following paths to find a class file named `myFile.class`.

1. `/myData/myFile.class`
2. `/myFiles/myFile.class`
3. `/myStuff/myFile.class`

There are also system paths that are searched that are not listed in the CLASSPATH.

A.1.2 What Is a Package?

The simplest definition of a package is a folder that contains Java class files. However, packages do more than that. They also indicate where a Java class can be found. Essentially, packages allow for an extension to the CLASSPATH list, without adding new paths to it.

If the class file `myFile.class` was in a package named `jbond007`, then Java will check the following paths to find the class file.

1. `/myData/jbond007/myFile.class`
2. `/myFiles/jbond007/myFile.class`
3. `/myStuff/jbond007/myFile.class`

If the class file `myFile.class` was in a package named `agents.jbond007`, then Java will check the following paths to find the class file.

1. `/myData/agents/jbond007/myFile.class`
2. `/myFiles/agents/jbond007/myFile.class`
3. `/myStuff/agents/jbond007/myFile.class`

Every section of the package name corresponds to a subdirectory in the file system. The first part of the package name corresponds to a directory on the file system that must be a subdirectory of a path in the CLASSPATH variable.

A.2 JAR File Problems

Java loads *.class* files when they are first accessed. At different times, Java uses different class loaders to create the class that is defined in the *.class* file. When a class is created, its information is stored in a separate part of memory that is never

garbage-collected by the JVM. The memory can be released when the class loader is removed; however, if there is still a reference from a different class loader to any object that was created by the class loader being removed, then none of the class definitions that were loaded by the class loader will be released. This is known as a memory leak.

Each web application has its own class loader. This is what makes it possible to restart a web application without restarting the entire JVM. There are class loaders for running the JVM and there is a class loader for the web application. When the web application is restarted, the class loader is reinitialised. As long as there are no active references to the objects created by the web application's class loader, all of the permanent memory that was allocated by the class loader will be released. If there is still an active reference, then none of the class information will be released and there will be a memory leak.

Other memory leaks can be caused by the programmer. Many packages have methods that should be called in order to release all the reources that are being used by the package. This is analogous to closing a file: there is a method to close the file, but the programmer must call it. If the file is not closed, then some data might not be written to the file. In the same way, the methods that exist for releasing resources must be called by the programmer, or the application will have a memory leak.

There are two types of memory leaks: catastrophic and one-timers. Catastrophic errors can cause the servlet engine to crash. The one-timers will usually not cause the servlet engine to crash; they will only force the engine to hold onto more resources than it needs.

If a web application with a catastrophic leak is reloaded enough times, then it will crash the servlet engine. Each time it is reloaded, it maintains a reference to the previous class loader. All the class data from the old class loader cannot be released and then a new class loader is created. If it is reloaded ten times, then there will be ten copies of the class data. If it is reloaded enough times, all of the memory in the servlet engine will be used and the engine will crash.

The one-timers only hold onto extra memory during the first reload; they do not leak more memory on subsequent reloads. The one-timers are usually caused by a static reference to an object. When the static reference is loaded by one class loader and the object is loaded by a different class loader, then there is a memory leak. Often, these leaks can be removed by using a weak reference to the object.

As of the publication of this book, Tomcat 6 is available as well as Hibernate 3.2.4. These packages have removed all of the catastrophic memory leaks. However, it is still up to the programmer to call all the necessary methods to release resources; otherwise, there will be catastrophic memory leaks.

A.2.1 Hibernate

Hibernate can cause a catastrophic memory leak, if the programmer does not release Hibernate's resources. There is a method in the Hibernate package that must be called before the web application closes: `closeFactory()`. If this is not called, then every time the web application is reloaded, the previous class loader and all of its memory will not be released.

The call to this method has been encapsulated in the `closeHibernate` method of the `HibernateHelper` class.

A.2.2 MySQL Driver

The MySql driver is used in this book, but this section applies to any SQL driver that is used. Once a driver has been registered with Java, then it must be deregistered before the web application is stopped; otherwise, there will be a catastrophic memory leak.

One technique is to deregister all of the drivers. An enumeration of all of the drivers can be obtained from the `DriverManager` class. Loop through each driver and deregister it.

```
try {
  Enumeration<Driver> enumer = DriverManager.getDrivers();
  while (enumer.hasMoreElements()) {
    DriverManager.deregisterDriver(enumer.nextElement());
  }
} catch (java.sql.SQLException se) {
  se.printStackTrace();
}
```

A.2.3 Hibernate Annotations

This section is not about a memory leak, but it is a warning to developers who have used earlier versions of Hibernate. In earlier versions, a multiple-valued bean property was annotated with `@OneToMany` to indicate that it will have a separate table created for it in the database. This table was created by Hibernate, without having to be defined. In Hibernate 3.2, this annotation is only to be used for entities that have been defined by the programmer, an implicit table cannot be created by using this annotation. The `@CollectionOfElements` annotation should be used, instead, to indicate that Hibernate should create an implicit, related table in the database.

A.3 MySQL

Although Hibernate eliminates the necessity of knowing SQL, sometimes curiosity gets the better of us and we want to see what Hibernate is doing. An example will be presented that demonstrates how to issue a few SQL commands in the MySQL database server to see the structure of the tables that Hibernate has created. A command will also be supplied that shows all the records in a table. These commands are very simple SQL commands. Only the bare minimum of statements will be introduced.

Additional commands will be supplied that are specific to the MySQL database. These additional commands are needed in order to log onto the server. The MySQL server is a free relational database. The source code can be obtained at http://mysql.org.

Two essential pieces of information are needed to log onto any server: username and password.

To access the MySQL database, issue the following command. This command will connect to the MySQL server that is installed on the local machine. Enter your password after MySQL prompts you for it.

```
mysql -u username -p <Enter Key>
password
```

View all databases in the server with the following command. Don't forget the ; at the end.

```
show databases;
```

Select a database named *db_name* with the following command. This is the only command that does not need a ; at the end.

```
use db_name
```

Once a database has been chosen, the names of all the tables can be displayed.

```
show tables;
```

There won't be any tables until you create a servlet that saves data to a database. Once you have a table, you can view the structure of a table named *name-of-table* with the following command.

```
describe name-of-table;
```

Use the select statement to see all the records in the table.

```
select * from name-of-table;
```

Exit MySQL with the following command.

```
exit;
```

These are the only commands that are needed to see what Hibernate has done.

A.4 Auxiliary Classes

Many classes that were used in the book were not explained in complete detail. The parts that were relevant to the theme of the book were covered, but the remaining parts of these classes were not covered. Most of the uncovered details deal with annotations, enumerations, error handling and reflection. A complete explanation of the code from these files belongs in a Java programming text.

A.4.1 Annotations

In addition to the annotations that are used by Hibernate, two other annotations were used in the book. `ButtonMethod` was used to annotate a method that does the processing associated with a button on a form. `SetByAttribute` was used on bean property accessors to indicate if the property is initialised by including an extra attribute in the definition of the form element in the JSP.

ButtonMethod

Listing A.1 is the `ButtonMethod` annotation. Place this annotation before a method that does the processing associated with a button in a form. The method that is annotated should have no parameters and should return a string.

```
package shared;

import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface ButtonMethod {
    String buttonName() default "";
    boolean isDefault() default false;
}
```

Listing A.1 The `ButtonMethod` annotation.

SetByAttribute

Listing A.2 is the `SetByAttribute` annotation. Place the annotation before the accessor of a bean property that is associated with a form element that is a button group or a select list. These types of form elements are initialised by including an extra attribute in the element. The type of the additional attribute is defined in the `AttributeType` parameter of the annotation.

```
package shared;

import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface SetByAttribute {
     AttributeType type();
}
```

Listing A.2 The `SetByAttribute` annotation.

A.4.2 Cookie Utility

The cookies that are retrieved in a servlet cannot be accessed randomly; it is necessary to do a linear search each time a cookie is needed. To facilitate this task, a

utility class of static methods was created that will perform such linear searches. A cookie can be retrieved by name; either the cookie or the cookie's value can be returned (Listing A.3).

```
package shared;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;

public class CookieUtil {

  static
  public Cookie findCookie(HttpServletRequest request,
                           String name)
  {
    if (request.getCookies() == null ) return null;
    for(Cookie cookie : request.getCookies()) {
      if (cookie.getName().equals(name)) return cookie;
    }
    return null;
  }

  static
  public String findCookieValue(HttpServletRequest request,
                                String name)
  {
    Cookie cookie = findCookie(request, name);
    if (cookie != null) {
      return cookie.getValue();
    }
    return null;
  }

}
```

Listing A.3 The `CookieUtil` class.

A.4.3 Enumerations

Enumerations are new in Java 1.5. They are an excellent way to create self-documenting code. The `SessionData` enumeration is used to indicate if the data that is already in the session should be read or ignored. The `AttributeType` enumeration is used by the `SetByAttribute` annotation to indicate the type of the attribute that is used to initialise a form element.

Session Data

Listing A.4 is the `SessionData` enumeration, which is defined in the `HelperBaseCh4` class. It is used as a parameter to the `addHelperToSession` method. While this parameter could have been implemented with a boolean variable, it is clearer to understand what `SessionData.READ` means than to remember what `true` means in this context.

```
protected enum SessionData { READ, IGNORE };
```

Listing A.4 The `SessionData` enumeration from `HelperBaseCh4`.

Attribute Type

Listing A.5 is the `AttributeType` enumeration. This is used in the `SetByAttribute` annotation. Certain form elements are initialised by adding an attribute to the form element. This form element attribute is named either `checked or selected`. This enumeration encapsulates these values.

```
package shared;

public enum AttributeType { CHECKED, SELECTED }
```

Listing A.5 The `AttributeType` enumeration.

A.4.4 Helper Base

The helper base classes have been used since Chapter Four. The subsequent helper base classes from each chapter extend the helper base from the previous chapter. Most of the details of the classes were covered in the book, but not every version of an overloaded method was covered. Reflection was also used in several places to simplify the work of the controller helper; the details of using reflection should be covered in a Java programming text.

Helper Base Chapter Four

This base class has member variables for the request, response and logger. These are initialised when the base class is constructed. There are also default implementations of the `doGet` and `doPost` methods that only display an error. The session is used to store data, the population of a bean is automated and reflection is used to execute a method that is associated with a button. Listing A.6 is the helper base class from Chapter Four.

```
package shared;

import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;

public abstract class HelperBaseCh4 {

protected enum SessionData { READ, IGNORE };
```

```
  private Method methodDefault = null;

  protected HttpServletRequest request;
  protected HttpServletResponse response;
  protected Logger logger;

  public HelperBaseCh4(HttpServletRequest request,
     HttpServletResponse response) {
  this.request = request;
  this.response = response;
  logger = Logger.getLogger("bytesizebook.webdev");
  logger.setLevel(Level.DEBUG);
  }

  protected void doGet()
  throws ServletException, IOException {
    response.getWriter()
       .print("The doGet method must be overridden" +
              " in the class that extends HelperBase.");
  }

  protected void doPost()
  throws ServletException, IOException {
    response.getWriter()
      .print("The doPost method must be overridden" +
             " in the class that extends HelperBase.");
  }

  protected abstract void copyFromSession(Object helper);

  public void addHelperToSession(String name,
                                 SessionData state) {
    if (SessionData.READ == state) {
      Object sessionObj =
         request.getSession().getAttribute(name);
      if (sessionObj != null) {
        copyFromSession(sessionObj);
      }
    }
    request.getSession().setAttribute(name, this);
  }

  public void addHelperToSession(String name,
                                 boolean checkSession) {
    if (checkSession) {
      Object sessionObj =
         request.getSession().getAttribute(name);
      if (sessionObj != null) {
        copyFromSession(sessionObj);
      }
    }
    request.getSession().setAttribute(name, this);
  }
```

```
protected String executeButtonMethod()
throws ServletException, IOException {
  String result = "";
  methodDefault = null;
  Class clazz = this.getClass();
  Class enclosingClass = clazz.getEnclosingClass();
  while (enclosingClass != null) {
    clazz = this.getClass();
    enclosingClass = clazz.getEnclosingClass();
  }

  try {
    result = executeButtonMethod(clazz, true);
  } catch (Exception ex) {
    writeError(request, response,
               "Button Method Error", ex);
    return "";
  }

  return result;
}

protected
String executeButtonMethod(Class clazz,
                           boolean searchForDefault)
throws IllegalAccessException, InvocationTargetException
{
  String result = "";
  Method [] methods =  clazz.getDeclaredMethods();
  for(Method method : methods) {
    ButtonMethod annotation =
        method.getAnnotation(ButtonMethod.class);
    if (annotation != null) {
      if (searchForDefault && annotation.isDefault())
      {
        methodDefault = method;
      }
      if (request.getParameter(annotation.buttonName())
              != null)
      {
        result = invokeButtonMethod(method);
        break;
      }
    }
  }
  if (result.equals("")) {
    Class superClass = clazz.getSuperclass();
    if (superClass != null) {
      result =
          executeButtonMethod(superClass,
                              methodDefault == null);
    }
    if (result.equals("")) {
      if (methodDefault != null) {
        result = invokeButtonMethod(methodDefault);
      } else {
```

```
        logger.error(
               "(executeButtonMethod) No default method " +
               "was specified, but one was needed.");
        result = "No default method was specified,.";
      }
    }
  }
  return result;
}

protected String invokeButtonMethod(Method buttonMethod)
throws IllegalAccessException, InvocationTargetException
{
  String resultInvoke = "Could not invoke method";
  try{
    resultInvoke =
        (String) buttonMethod.invoke(this,
                                     (Object[]) null);
  } catch (IllegalAccessException iae) {
    logger.error("(invoke) Button method is not public.",
                 iae);
    throw iae;
  } catch (InvocationTargetException ite) {
    logger.error("(invoke) Button method exception",
                 ite);
    throw ite;
  }
  return resultInvoke;
}

public void fillBeanFromRequest(Object data) {
  try {
    org.apache.commons.beanutils.BeanUtils.
        populate(data, request.getParameterMap());
  } catch (IllegalAccessException iae) {
    logger.error("Populate - Illegal Access.", iae);
  } catch (InvocationTargetException ite) {
    logger.error("Populate - Invocation Target.", ite);
  }
}

public void populateThrow(Object data)
throws IOException, ServletException {
  try {
    org.apache.commons.beanutils.BeanUtils.
        populate(data, request.getParameterMap());
  } catch (IllegalAccessException iae) {
    logger.error("Populate - Illegal Access.", iae);
    writeError(request, response,
               "Populate - Illegal Access.", iae);
  } catch (InvocationTargetException ite) {
    logger.error("Populate - Invocation Target.", ite);
    writeError(request, response,
               "Populate - Invocation Target.", ite);
  }
}
```

```
  static public void writeError(
      javax.servlet.http.HttpServletRequest request,
      javax.servlet.http.HttpServletResponse response,
      String title,
      Exception ex)
      throws IOException, ServletException
  {
    java.io.PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    out.println("<html>");
    out.println("  <head>");
    out.println("    <title>" + title + "</title>");
    out.println("  </head>");
    out.println("  <body>");
    out.println("<h2>" + title + "</h2>");
    if (ex.getMessage() != null)
      out.println("    <h3>" + ex.getMessage()+ "</h3>");
    if (ex.getCause() != null)
      out.println("    <h4>" + ex.getCause()+ "</h4>");
    StackTraceElement[] trace = ex.getStackTrace();
    if (trace != null && trace.length > 0)
      out.print("<pre>");
    ex.printStackTrace(out);
    out.println("</pre>");
    out.println("  </body>");
    out.println("</html>");
    out.close();
  }
}
```

Listing A.6 The helper base class from Chapter Four.

Helper Base Chapter Five

The Chapter Five helper base class adds the enhanced interface for accessing the Hibernate validation messages (Listing A.7).

```
package shared;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.hibernate.validator.ClassValidator;
import org.hibernate.validator.InvalidValue;

public abstract class HelperBaseCh5 extends HelperBaseCh4 {

   public HelperBaseCh5(HttpServletRequest request,
          HttpServletResponse response) {
       super(request, response);
   }

   java.util.Map<String, String> errorMap =
         new java.util.HashMap<String, String>();
```

```
    public void setErrors(Object data) {
        InvalidValue[] validationMessages;
        ClassValidator requestValidator =
               new ClassValidator(data.getClass());
        validationMessages =
                requestValidator.getInvalidValues(data);

        errorMap.clear();
        if (validationMessages.length != 0) {
            for(InvalidValue msg : validationMessages) {
                errorMap.put(msg.getPropertyName(),
                             msg.getMessage());
            }
        }
      }

    public boolean isValid(Object data) {
        setErrors(data);
        return errorMap.isEmpty();
    }

    public java.util.Map getErrors() {
        return errorMap;
    }

    public boolean isValidProperty(String name) {
        String msg = errorMap.get(name);
        return msg == null || msg.equals("");
    }

}
```

Listing A.7 The helper base class from Chapter Five.

Helper Base Chapter Six

The Chapter Six helper base class adds the maps for initialising complex form elements (Listing A.8).

```
package shared;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public abstract class HelperBaseCh6 extends HelperBaseCh5 {

  protected Map<String, Map<String, String>> checked =
      new HashMap<String, Map<String, String>>();
```

```
protected Map<String, Map<String, String>> selected =
    new HashMap<String, Map<String, String>>();


public HelperBaseCh6(HttpServletRequest request,
    HttpServletResponse response) {
  super(request, response);
}

protected void setCheckedAndSelected(Object data) {
  setCheckedAndSelected(data, data.getClass());
}

protected void setCheckedAndSelected(Object data,
    Class clazz) {
  Method [] allMethods = clazz.getDeclaredMethods();
  Method methodDefault = null;
  for (Method method : allMethods) {
    SetByAttribute propAnnotation = method
        .getAnnotation(SetByAttribute.class);
    if (propAnnotation!=null) {
      String property = method.getName();
      java.util.regex.Pattern pattern
         = java.util.regex.Pattern.compile("get(.+)");
      java.util.regex.Matcher matcher
         = pattern.matcher(property);
      int index = property.indexOf("get");
      if (!matcher.matches()) {
        logger.error(property + " must be an accessor.");
      } else {
        property = matcher.group(1);
        property = property.substring(0,1).toLowerCase()
                      + property.substring(1);
        clearProperty(property,
                      propAnnotation.type());
        if (method.getReturnType().isArray()) {
          Object[] result =
              (Object[]) invokeGetter(data, method);
          if (result != null) {
            for(Object obj: result) {
              addChoice(property, obj.toString(),
                        (AttributeType)propAnnotation.type());
            }
          }
        } else {
          Object result = invokeGetter(data, method);
          if (result != null) {
            addChoice(property, result.toString(),
                 (AttributeType)propAnnotation.type());
          }
        }
      }
    }
  }
  Class parentClass = clazz.getSuperclass();
```

```
    if (parentClass != null) {
      setCheckedAndSelected(data, parentClass);
    }
  }

  protected Object invokeGetter(Object obj, Method method)
  {
    Object result = null;
    try{
      result =  method.invoke(obj, (Object[]) null);
    } catch (IllegalAccessException iae) {
      logger.error("(invoke) Accessor needs public access",
                   iae);
    } catch (InvocationTargetException ite) {
      logger.error("(invoke) Accessor threw an exception",
                   ite);
    }
    return result;
  }

  public Map getChecked() {
    return checked;
  }

  public Map getSelected() {
    return selected;
  }

  public void addChecked(String group, String item) {
    if (checked.get(group) == null) {
      checked.put(group,
                  new HashMap<String, String>());
    }
    checked.get(group).put(item, "checked");
  }

  public void addSelected(String list, String item) {
    if (selected.get(list) == null) {
      selected.put(list,
                   new HashMap<String, String>());
    }
    selected.get(list).put(item, "selected");
  }

  public void addChoice(String list,
      String item,
      AttributeType type) {
    if (type == null ) return;
    if (AttributeType.CHECKED == type) {
      addChecked(list, item);
    }
    if (AttributeType.SELECTED == type) {
      addSelected(list, item);
    }
  }
```

```
  public void clearProperty(String property,
      AttributeType type) {
    Map<String, String> propMap;
    if (AttributeType.CHECKED == type) {
      propMap = checked.get(property);
      if (propMap != null) {
        propMap.clear();
      }
    } else if (AttributeType.SELECTED == type) {
      propMap = selected.get(property);
      if (propMap != null) {
        propMap.clear();
      }
    }
  }

  public void clearMaps() {
    checked.clear();
    selected.clear();
  }

}
```

Listing A.8 The helper base class from Chapter Six.

A.4.5 Hibernate Helper

The Hibernate helper class encapsulates access to the Hibernate methods. Hibernate uses transactions and session to access the database. The details of using these are placed in the Hibernate helper methods.

Saving Data

The `saveOrUpdate` method is used to add new objects in, or update existing objects to the database. Hibernate uses sessions and transactions to interact with the database. The details of sessions and transactions are outside the scope of this text.

```
public void updateDB(Object obj) {
  Session session = null;
  try {
    session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    session.saveOrUpdate(obj);

    tx.commit();
  } finally {
    if (session != null) session.close();
  }
}
```

```
static
public void updateDB(java.util.List list) {

  Session session = sessionFactory.openSession();
  Transaction tx = session.beginTransaction();

  for(Object obj : list) {
    session.saveOrUpdate(obj);
  }

  tx.commit();
  session.close();
}
```

Retrieving Data

Retrieving data is accomplished by using a `Criteria` object. This method returns all the records from the database. If a key and value are sent to the method, then only the records that match the value for that key will be returned. This method could be extended to include additional criteria, so that a more complicated search could be performed.

```
public java.util.List getListData(
    Class classBean, String strKey, Object value)
{
  java.util.List result = new java.util.ArrayList();

  Session session = sessionFactory.openSession();
  Transaction tx = session.beginTransaction();

  Criteria criteria =
      session.createCriteria(classBean);
  if (strKey != null) {
    criteria.add(Restrictions.like(strKey, value));
  }
  result = criteria.list();

  tx.commit();
  session.close();
  return result;
}
```

Removing Data

Once a record has been found, Hibernate will remember that it was retrieved from the database. By sending the same object to the remove method, it will be deleted from the database.

```
public void removeDB(Object obj) {
  Session session = null;
  try {
    session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
```

```
    session.delete(obj);

    tx.commit();
  } finally {
    if (session != null) session.close();
  }
}
```

Hibernate Helper Class

Listing A.9 is the complete Hibernate helper class.

```
package shared;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;
import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.criterion.Restrictions;

public class HibernateHelper {

  static protected Logger log =
     Logger.getLogger("bytesizebook.webdev");

  static protected List<Class> listClasses = new
ArrayList<Class>();
  static protected SessionFactory sessionFactory;
  static protected Exception lastError;

  static
  public void initSessionFactory(Properties props,
     Class... mappings) {
    if (addMappings(listClasses, mappings)) {
      closeFactory(sessionFactory);
      sessionFactory = createFactory(props, listClasses);
    }
  }
```

```
  static
  public void initSessionFactory(Class... mappings) {
    initSessionFactory(null, mappings);
  }

  static
  public void createTable(Properties props,
      Class... mappings) {
    List<Class> tempList = new ArrayList<Class>();
    SessionFactory tempFactory = null;

    addMappings(tempList, mappings);
    if (props == null ) props = new Properties();
    props.setProperty(Environment.HBM2DDL_AUTO, "create");
    tempFactory = createFactory(props, tempList);
    closeFactory(tempFactory);
  }

  static
  public void createTable(Class... mappings) {
    createTable(null, mappings);
  }

  static
  protected boolean addMappings(List<Class> list, Class...
mappings) {
    boolean bNewClass = false;
    for (Class mapping : mappings) {
      if (!list.contains(mapping)) {
        list.add(mapping);
        bNewClass = true;
      }
    }
    return bNewClass;
  }

  static
  protected SessionFactory createFactory(
                           Properties props,
                           List<Class> list) {
    SessionFactory factory = null;
    AnnotationConfiguration cfg =
        new AnnotationConfiguration();
    try {
      if (props != null) cfg.addProperties(props);
      configureFromFile(cfg);
      for (Class mapping : list) {
        cfg.addAnnotatedClass(mapping);
      }
      factory = buildFactory(cfg);
      testConnection(factory);
    } catch (Exception ex) {
      log.error("SessionFactory creation failed.", ex);
      lastError = ex;
      closeFactory(factory);
```

```
    factory = null;
  }
  return factory;
}

static
protected void configureFromFile(Configuration cfg)
throws Exception {
  try {
    cfg.configure();
  } catch (HibernateException ex) {
    if (ex.getMessage().equals(
        "/hibernate.cfg.xml not found")) {
      log.warn(ex.getMessage());
    } else {
      log.error("Error in hibernate " +
          "configuration file.", ex);
      throw ex;
    }
  }
}

static
protected SessionFactory buildFactory(Configuration cfg)
throws Exception
{
  SessionFactory factory = null;
  try {
    factory = cfg.buildSessionFactory();
  } catch (Exception ex) {
    closeFactory(factory);
    factory = null;
    throw ex;
  }
  return factory;
}

static
protected void testConnection(SessionFactory factory)
throws Exception {
  Session session = null;
  try {
    session = factory.openSession();
    Transaction tran = session.beginTransaction();

    someDatabaseProcess();

    tran.commit();
    session.close();
  } catch (Exception ex) {
    log.error("Database transaction failed.", ex);
    if (session != null) session.close();
    throw ex;
  }
}
```

```
  static
  protected void someDatabaseProcess() {}

  static
  public void closeFactory(SessionFactory factory) {
    if (factory != null) {
      factory.close();
    }
  }

  static
  public void closeFactory() {
    closeFactory(sessionFactory);
  }

  static
  public Exception getLastError() {
    return lastError;
  }

  static
  public void updateDB(Object obj) {
    Session session = null;
    try {
      session = sessionFactory.openSession();
      Transaction tx = session.beginTransaction();

      session.saveOrUpdate(obj);

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

  static
  public void updateDB(java.util.List list) {

    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    for(Object obj : list) {
      session.saveOrUpdate(obj);
    }

    tx.commit();
    session.close();
  }

  static
  public void saveDB(Object obj) {
    Session session = null;
    try {
      session = sessionFactory.openSession();
      Transaction tx = session.beginTransaction();
```

```
    session.save(obj);

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

static
public void removeDB(Object obj) {
  Session session = null;
  try {
    session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    session.delete(obj);

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

static
public java.util.List getListData(
   Class classBean, String strKey, Object value)
{
  java.util.List result = new java.util.ArrayList();

  Session session = sessionFactory.openSession();
  Transaction tx = session.beginTransaction();

  Criteria criteria =
     session.createCriteria(classBean);
  if (strKey != null) {
    criteria.add(Restrictions.like(strKey, value));
  }
  result = criteria.list();

  tx.commit();
  session.close();
  return result;
}

static
public java.util.List getListData(
   Class classBean) {
  return getListData(classBean, null, null);
}

static
public Object getFirstMatch(
   Class classBean, String strKey, Object value) {
  java.util.List records =
     getListData(classBean, strKey, value);
```

```
    if (records != null && records.size() > 0) {
      return records.get(0);
    }
    return null;
  }

  static
  public Object getFirstMatch(
     Object data, String strKey, Object value) {
    return getFirstMatch(data.getClass(),
       strKey, value);
  }

  static
  public Object getKeyData(Class beanClass, long itemId) {
    Object data = null;
    Session session = sessionFactory.openSession();

    data = session.get(beanClass, itemId);

    session.close();

    return data;
  }

  static
  public boolean isSessionOpen() {
    return sessionFactory != null;
  }

  static
  public boolean testDB(HttpServletResponse response)
  throws IOException, ServletException {
    if (!isSessionOpen()) {
      writeError(response);
    }
    return isSessionOpen();
  }

  static
  public void writeError(HttpServletResponse response,
                         String title,
                         Exception ex)
  throws java.io.IOException, ServletException
  {
    java.io.PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    out.println("<html>");
    out.println("  <head>");
    out.println("    <title>" + title + "</title>");
    out.println("  </head>");
    out.println("  <body>");
    out.println("<h2>" + title + "</h2>");
    if (ex != null) {
      if (ex.getMessage() != null) {
```

```
        out.println(
            "     <h3>" + ex.getMessage() + "</h3>");
      }
      if (ex.getCause() != null) {
        out.println(
            "     <h4>" + ex.getCause() + "</h4>");
      }
      StackTraceElement[] trace = ex.getStackTrace();
      if (trace != null && trace.length > 0) {
        out.print("<pre>");
        ex.printStackTrace(out);
        out.println("</pre>");
      }
    } else {
      out.println("Hibernate must be initialized");
    }
    out.println("   </body>");
    out.println("</html>");
    out.close();
  }

  static
  public void writeError(HttpServletResponse response)
  throws java.io.IOException, ServletException {
    writeError(response,
        "Hibernate Initialization Error",
        lastError);
  }
}
```

Listing A.9 The Hibernate helper class.

A.4.6 InitLog4j Servlet

A servlet is used to configure Log4j. Listing A.10 is the complete InitLog4j servlet.

```
package shared;

import java.io.IOException;
import javax.servlet.http.HttpServlet;
import org.apache.log4j.FileAppender;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.PatternLayout;
import org.apache.log4j.RollingFileAppender;

public class InitLog4j extends HttpServlet {

  private static final String logPath =
      "/WEB-INF/logs/error.log";

  public void init() {
    FileAppender appender = getAppender(logPath);
    if (appender == null ) return;
    initLogger(null, appender, Level.ERROR);
```

```
    initLogger("org.apache.commmons.beanutils",
                appender, Level.DEBUG);
  }

  private FileAppender getAppender(String fileName) {
    RollingFileAppender appender = null;
    try {
      appender = new RollingFileAppender(
          new PatternLayout("%-5p %c %t%n%29d - %m%n"),
          getServletContext().getRealPath(fileName),
          true);
      appender.setMaxBackupIndex(5);
      appender.setMaxFileSize("1MB");
    } catch (IOException ex) {
      System.out.println(
          "Could not create appender for "
          + fileName + ":"
          + ex.getMessage());
    }
    return appender;
  }

  private void initLogger(String name,
                          FileAppender appender,
                          Level level)
  {
    Logger logger;
    if (name == null) {
      logger = Logger.getRootLogger();
    } else {
      logger = Logger.getLogger(name);
    }
    logger.setLevel(level);
    logger.addAppender(appender);
    logger.info("Starting " + logger.getName());
  }
}
```

Listing A.10 The `InitLog4j` servlet.

A.4.7 PersistentBase Class

It is a repetitive task to make a bean into a persistent bean. To facilitate this task, a base class was created that contains the definition of a primary key (Listing A.11). By extending this class, a primary key will be added to the bean.

```
package shared;

import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;

@MappedSuperclass
public class PersistentBase {
```

```
    protected Long id;

    @Id
    @GeneratedValue
    public Long getId() { return id; }

    protected void setId(Long id) { this.id = id; }


    public PersistentBase() {
    }

}
```

Listing A.11 The `PersistentBase` class.

It is recommended that such a primary key is used, even if there is a natural primary key in the data. This is because the primary key plays a fundamental role in how Hibernate maintains state.

A.4.8 Webapp Listener

In order to be sure that Hibernate is closed before the web application is stopped, a reference to a context listener was added to the *web.xml* file. Listing A.12 contains the source code for this listener.

```
package shared;

import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Enumeration;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class WebappListener implements ServletContextListener
{
  public void contextInitialized(ServletContextEvent sce)
  {}

  public void contextDestroyed(ServletContextEvent sce)
  {
    try {
      Enumeration<Driver> enumer = DriverManager.getDrivers();
      while (enumer.hasMoreElements()) {
        DriverManager.deregisterDriver(enumer.nextElement());
      }
    } catch (java.sql.SQLException se) {
      se.printStackTrace();
    }
    shared.HibernateHelper.closeFactory();
  }
}
```

Listing A.12 The `WebappListener` class that is used to close Hibernate.

Glossary

CSS – Cascading Style Sheets
EL – Expression Language
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
IDE – Integrated Development Environment
JAR – Java Archive
JSP – Java Server Page
JSTL – Java Standard Template Library
JVM – Java Virtual Machine
MIME – Multipurpose Internet Mail Extensions
MVC – Model, View, Controller
SQL – Structured Query Language
URL – Uniform Resource Locator
W3C – WWW Consortium
WAR – Web Archive

References

Additional Resources

Books

Bauer C, King G, (2005) Hibernate in Action, Manning, Greenwich.
Chopra V, Galbraith B, Li S, Wiggers C, Bakore A, Bhattacharjee D, et al., (2002) Professional Apache Tomcat, Wrox, Birmingham.
Hall M, (2002) More Servlets and Java Server Pages, Sun, Palo Alto.
Hall M, Brown L, (2004) Core Servlets and Java Server Pages, Second Edition, Sun, Santa Clara.
Ragget D, Lam J, Alexander I, Kmiec M, (1998) Raggett on HTML 4, Second Edition, Addison-Wesley, Harlow.
Stein L, (1997) How to Set Up and Maintain a Web Site, Second Edition, Addison-Wesley, Reading.

Web Sites

Hibernate, http://www.hibernate.org/247.html
Hibernate Annotations, http://www.hibernate.org/hib_docs/annotations/api/index.html?org/hibernate/annotations/package-summary.html
Hibernate Validator, http://www.hibernate.org/hib_docs/annotations/reference/en/html/validator.html
Jakarta Commons, http://jakarta.apache.org/commons/
Java, http://sun.comJava
Log4j, http://logging.apache.org/log4j/docs/
Memory Leaks and Class Loaders, Frank Kieviet Blog,
http://blogs.sun.com/fkieviet/entry/classloader_leaks_the_dreaded_java
NetBeans, http://netbeans.org
Regular Expressions, http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html
Tomcat, http://tomcat.apache.org/

Index