

Chapter 1, Page 3, Example 1-2, Add at top of code:

```
import java.util.*;
```

Chapter 1, Page 4, Exercise 1-3, Add at top of code:

```
import java.util.*;
```

Chapter 1, Page 7, the sentence right after Item 3. in the ordered list, change

<We'll rename the RentABike to ArrayListRentABike (Example 1-4)> to

<We'll rename the RentABike file, class definition and constructor to
ArrayListRentABike (Example 1-4)>

Chapter 1, Page 7, Exercise 1-4, Change caption from:

<ArrayListRentABike.java> to

<ArrayListRentABike.java (renamed from RentABike.java)>

Chapter 1, Page 7, Exercise 1-4, Change the entire code sample to:

```
import java.util.*;
```

```
public class ArrayListRentABike implements RentABike {  
    private String storeName;  
    final List bikes = new ArrayList();
```

```
    public ArrayListRentABike() { initBikes(); }
```

```
    public ArrayListRentABike(String storeName) {  
        this.storeName = storeName;  
        initBikes();
```

```
}
```

```
    public void initBikes() {  
        bikes.add(new Bike("Shimano", "Roadmaster", 20,  
            "11111", 15,  
            "Fair"));  
        bikes.add(new Bike("Cannondale", "F2000 XTR",  
            18, "22222", 12,  
            "Excellent"));  
        bikes.add(new Bike("Trek", "6000", 19, "33333",
```

```
12.4,  
"Fair"));  
}
```

```
public String toString() { return "RentABike: " + storeName; }
```

```
public List getBikes() { return bikes; }
```

```
public Bike getBike(String serialNo) {  
    Iterator iter = bikes.iterator();  
    while(iter.hasNext()) {
```

```
        Bike bike = (Bike)iter.next();  
        if(serialNo.equals(bike.getSerialNo())) return bike;  
    }
```

```
    return null;  
}
```

Chapter 1, Page 7, Exercise 1-5, Add at top of code:

```
import java.util.*;
```

Chapter 1, Page 7, Exercise 1-6, Add at top of code:

```
import java.util.*;
```

Chapter 1, Page 11, the sentence right after the bulleted list, change from:

<Finally, you'll need an ant build script.> to
<Finally, you'll need an ant build script, which we'll place in the root folder of our project.>

Chapter 1, Page 11, Example 1-8, change the line:

<property name="class.dir" value="\${war.dir}/classes"/> to
<property name="class.dir" value="\${war.dir}/WEB-INF/classes/">

Chapter 1, Page 11, output listing at bottom of page, change the line:

```
<[mkdir] Created dir: C:\RentABikeApp\war\classes> to  
<[mkdir] Created dir: C:\RentABikeApp\war\WEB-INF\classes>
```

Chapter 1, Page 11, output listing at bottom of page, change the line:

```
<[javac] Compiling 5 source files to C:\RentABikeApp\war\classes> to  
<[javac] Compiling 5 source files to C:\RentABikeApp\war\WEB-INF\classes>
```

Chapter 1, Page 13, last sentence of first paragraph, change the sentence:

```
<We used version 1.1.> to  
<We used version 1.1. You will need to add a new folder to your project, war\WEB-  
INF\lib, and put the Spring libraries there (everything in the \dist folder of your Spring  
distribution).>
```

Chapter 1, Page 13, right before Example 1-9, change the sentence:

```
<We'll simply replace our assembler with a Spring version, and provide a configuration  
file.>  
to  
<We'll simply replace our assembler with a Spring version, and provide a configuration  
file which will go in the \war\WEB-INF folder.>
```

Chapter 1, Page 16, Example 1-12, replace with:

```
<property name="test.class.dir" value="{test.dir}/classes"/>  
<path id="bikestore.class.path">  
  <fileset dir="{spring.dir}/dist">  
    <include name="*.jar" /></fileset> <pathelement  
      location="{spring.dir}/lib/jakarta-commons/commons-logging.jar"/><pathelement  
      location="{spring.dir}/lib/log4j/log4j-1.2.8.jar"/><pathelement  
      location="{spring.dir}/lib/j2ee/servlet.jar"/><dirset dir="{basedir}"/><dirset  
      dir="{class.dir}"/>  
</path>  
<path id="junit.class.path"><path  
  refid="bikestore.class.path"/><pathelem  
  ent location="PATH_TO_JUNIT"/>  
</path>  
<target name="compile.test" depends="init" description="Compiles all unit test source">  
  
  <javac srcdir="{test.dir}"
```

```
    destdir="${test.class.dir}"
    classpathref="junit.class.path"/>
```

```
</target>
```

Chapter 2, page 19, Example 2-2: there are two bolded lines above the label for the example. They should be included as part of the example, not appear above the label.

Chapter 2, page 19, sentence right before Example 2-2, change from:

<Here's the hardwired JSP:> to

<Notice that from here on out, we've moved our code into a package,com.springbook. The source files for the domain classes should move into the src\com\springbook folder as well. Here's the hardwired JSP:>

Chapter 2, Page 22, Paragraph above Example 2-6, replace whole paragraph with:

For now, the view will be a thin JSP layer that lets you pick a bike from a list, then remove or edit it. You'll use another screen to add or edit bike information. We aren't focusing on formatting, since we're primarily interested in Spring's role. You'll use standard JSTL (available at <http://java.sun.com/products/jsp/jstl/>, or packaged with Spring). Add standard.jar,jstl.jar, c.tld and fmt.tld to your war\WEB-INF\lib folder. You'll link them through an include.jsp page which contains just the following two lines:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core"%>
```

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
```

Chapter 2, Page 22, Example 2-6: change label from:

```
<editBike.jsp>
```

```
to <bikes.jsp>
```

example 2-11: requires taglib

```
<taglib>
  <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
  <taglib-location>/WEB-INF/lib/c.tld</taglib-location>

</taglib>
```

Chapter 2, Page 25, Example 2-8, Add to top of code:

```
package com.springbook;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
```

Chapter 2, Page 25, Example 2-9, Add to top of code:

```
package com.springbook;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

Chapter 2, Page 26, Example 2-10, Add to top of code:

```
package com.springbook;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

Chapter 2, Page 26, Example 2-10, Change the line:

```
<new Integer(request.getParameter("frame")).intValue(),> to  
<Integer.parseInt(request.getParameter("frame")),>
```

Chapter 2, Page 26, Example 2-10, Change the line:

```
<new Double(request.getParameter("weight")).doubleValue(),> to  
<Double.parseDouble(request.getParameter("weight")),>
```

Chapter 2, Page 26, Paragraph just above Example 2-11, replace whole paragraph with:

Within the context, you'll wire together the model, facade and UI. You'll want to specify the controller and view objects. You'll also need to configure a special object, called a dispatcher, within the configuration. First, you'll need to register the top-level dispatcher and your new taglibs with Tomcat in web.xml (Example 2-11).

Chapter 2, Page 26, Example 2-11, replace whole example with:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application  
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">  
<web-app>  
<taglib><taglib-uri>http://java.sun.com/jstl/core</taglib-  
uri><taglib-location>/WEB-INF/lib/c.tld</taglib-location>  
</taglib>  
<servlet> <servlet-name>rentaBikeApp</servlet-  
name><servlet-class>  
    org.springframework.web.servlet.DispatcherServlet  
</servlet-class>  
<load-on-startup>1</load-on-startup>  
  
</servlet>  
<servlet-mapping><servlet-
```

```
name>rentaBikeApp</servlet-name><url-  
pattern>*.bikes</url-pattern>  
</servlet-mapping></web-app>
```

Chapter 2, Page 27, Paragraph above Example 2-12, Change sentence from:

<Next, you're going to want to build a Spring context, called rentabikeApp-servlet.xml that has your controllers and view now.> to

<Next, build a Spring context in \war\WEB-INF\ called rentabikeApp-servlet.xml that has your controllers and view in it.>

Chapter 2, Page 27, Example 2-12, change label to:
rentabikeApp-servlet.xml

Chapter 2, Page 28, Figure 2-2:

the URL in the figure should be pointing to bikes.bikes, not bikes.htm.

Chapter 2, Page 30, Example 2-13, change label to:
rentabikeApp-servlet.xml

Chapter 2, Page 34, First sentence, change from:

<Finally, you'll need to configure the changes in the context [Example 2-17].>
to

<Next, modify the context as in Example 2-17; also, in the urlMapping, point /
editBike.bikes to editBikeForm instead of editBikeController.>

Chapter 2, Page 34, Example 2-17, change label to:
rentaBikeApp-servlet.xml, editBike.bikes should point to editBikeForm

Chapter 2, Page 34, right after Example 2-17, need to add paragraph and example:

Next, you'll need to add the CRUD functionality to `ArrayListRentABike` that you specified earlier in the `RentABike` interface.

Example 2-18 `ArrayListRentABike.java`

```
package com.springbook;
import java.util.*;
```

```
public class ArrayListRentABike implements RentABike {
    private String storeName;
    final List bikes = new ArrayList();
```

```
public void setStoreName(String name) {
    this.storeName = name;
}
```

```
public String getStoreName() {
    return storeName;
}
```

```
private void initBikes() {
    bikes.add(new Bike("Shimano",
        "Roadmaster", 20, "11111", 15,
        "Fair"));
    bikes.add(new Bike("Cannondale", "F2000
        XTR", 18, "22222", 12,
        "Excellent"));
    bikes.add(new Bike("Trek", "6000", 19,
        "33333", 12.4,
        "Fair"));
```

```
}
```

```
public ArrayListRentABike() {  
    initBikes();  
}
```

```
public ArrayListRentABike(String storeName) {  
    this.storeName = storeName;  
    initBikes();  
}
```

```
public String toString() { return "com.springbook.RentABike: " + storeName; }
```

```
public List getBikes() { return bikes; }
```

```
public Bike getBike(String serialNo) {  
    Iterator iter = bikes.iterator();  
    while(iter.hasNext()) {  
  
        Bike bike = (Bike)iter.next();  
        if(serialNo.equals(bike.getSerialNo())) return bike;  
    }  
    return null;  
}
```

```
public void saveBike(Bike bike) {  
    deletelfContains(bike);  
    bikes.add(bike);  
}
```

```
public void deleteBike(Bike bike) {
    deletelfContains(bike);
}
```

```
private void deletelfContains(Bike bike) {
    Iterator iter = bikes.iterator();
    while(iter.hasNext()) {
```

```
        Bike comp = (Bike)iter.next();
```

```
        if(comp.getManufacturer().equals(bike.getManufacturer()) &&
```

```
        comp.getModel().equals(bike.getModel())) {bikes.remove(comp);return;
        }}}
```

Chapter 2, Page 34, Right after previous addition, add paragraph and example:

Finally, you'll need to add the Spring taglib to the web.xml file so that your new JSP tags will work.

Example 2-19. web.xml

```
<taglib><taglib-uri>/spring</taglib-uri><taglib-location>/WEB-INF/lib/spring.tld</taglib-location>
</taglib>
```

Chapter 2, Examples 2-18 and 2-19, renumber as 2-20 and 2-21

Chapter 4, Page 60, First paragraph after the bulleted list, add this sentence as the first sentence in the paragraph:

Instead, you'll create a file with your database schema and data and store it in the project's /db folder, which you created earlier.

CHAPTER 3:

Page 37, last paragraph, change sentence from:

<Keith Donald's Spring rich client project brings to users a more sophisticated swing user interfaces to Spring.>

to

<Keith Donald's Spring rich client project brings more sophisticated Swing user interfaces to Spring.>

Page 41, paragraph right before example 3-3, change first sentence from:

<You'll need to define stub for the scheduling part of the application.>

to

<You'll need to define a stub for the scheduling part of the application.>

Chapter 4, Page 60, right after Example 4-2, add the sentence:

After creating the tables, you should assign all permissions to your account so that your code can access the data.

Chapter 4, Page 61, Example 4-3, change entire example to:

```
public void testJDBC() throws Exception {
    try {
        System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection("jdbc:mysql://
localhost/bikestore");
    } catch (Exception ex) {

        fail("Failed to open connection: " + ex.getMessage());
    }
    assertTrue(true);
}
```

Chapter 4, Page 61, paragraph right after Example 4-3, change first sentence from:

<Finally, Example 4-4 creates some sample data.> to

<Example 4-4 creates some sample data.>

Chapter 4, Page 61, right after Example 4-4, add the following paragraph and example:

Finally, you'll modify your Ant build script to use the rentabike.sql file to create your database. To do so, you'll have to provide a couple of new properties, as well as a classpath that points to the MySQL driver in your /lib folder.

Example 4-5 build.xml

```
<property name="database.url" value="jdbc:mysql://localhost/bikestore"/>
```

```
<property name="database.username" value="bikestore"/>
```

```
<path id="mysql.class.path">
```

```
  <pathelement location="${war.dir}/WEB-INF/lib/mysql-  
connector-java-3.0.14-production-bin.jar"/>
```

```
</path>
```

```
<target name="create.tables">
```

```
  <sql driver="com.mysql.jdbc.Driver"
```

```
    url="${database.url}"
```

```
    userid="${database.username}"
```

```
    password="">
```

```
  <classpath>
```

```
    <path refid="mysql.class.path"/>
```

```
  </classpath>
```

```
    <fileset dir="${db.dir}">
```

```
<include name="rentabike.sql"/>
  </fileset>
</sql>
</target>
```

Chapter 4, Page 64, Example 4-6, change label to:
rentabikeApp-servlet.xml

Chapter 4, Page 67 and 68, Example 4-7, delete both lines that read:
template.setDataSource(this.dataSource);

Chapter 4, Page 67, second paragraph, change the sentence:
<JDBC's clob handling does not match VARCHAR handling at all.> to
<JDBC's CLOB handling does not match VARCHAR handling at all.>

Chapter 4, Page 70, After example 4-9, add the following paragraph and figures:

At this point, you will also add two new domain classes that represent customers and reservations.

Example 4-10

```
Customer.javapackage
com.springbook;import
java.util.Set;
```

```
public class Customer {private int
  custId; private String
  firstName;private String
  lastName; private Set
  reservations;
```

```

public Set getReservations() { return reservations; }

public void setReservations(Set reservations) { this.reservations
=reservations; }

public int getCustId() { return custId; }

public void setCustId(int custId) { this.custId = custId; }

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public Customer(int custId, String firstName, String lastName) {
    this.custId = custId;
        this.firstName = firstName;
        this.lastName = lastName;
}

public Customer() {}

public String toString() {
    return "Customer : " +
        "custId -- " + custId +
            "\n: firstName -- " + firstName +
                "\n: lastName -- " + lastName +
                    ".\n";
}}

```

Example 4-11 Reservation.java

```
package com.springbook;
```

```
import java.util.Date;
```

```
public class Reservation {
```

```
    private int reservationId;
```

```
    private Date reservationDate;
```

```
    private Bike bike;
```

```
    private Customer customer;
```

```
    public Reservation() {}
```

```
    public int getReservationId() { return reservationId; }
```

```
    public void setReservationId(int reservationId) { this.reservationId =  
reservationId; }
```

```
    public Date getReservationDate() { return reservationDate; }
```

```
    public void setReservationDate(Date reservationDate) {  
this.reservationDate = reservationDate; }
```

```
    public Bike getBike() { return bike; }
```

```
    public void setBike(Bike bike) { this.bike = bike; }
```

```
    public Customer getCustomer() { return customer; }
```

```
    public void setCustomer(Customer customer) { this.customer = customer; }
```

```
    public Reservation(int id, Bike bike, Customer customer, Date date) {
```

```

    this.reservationId = id;
    this.bike = bike;
    this.customer = customer;

    this.reservationDate = date;
}

public String toString() {
    return "Reservation : " +
        "reservationId -- " + reservationId +
        "\n: reservationDate -- " + reservationDate +
        "\n: bike -- " + bike +
        "\n: customer -- " + customer +
        ".\n";
}}

```

Chapter 4, figures after 4-9, rename to figure name + 2 (4-9 becomes 4-11, 4-10 become 4-12, etc.)

Chapter 4, Page 72, Example 4-10, change the line:

```

<MockControl controlConnection = MockControl.createControl (Connection.class);> to
<MockControl controlConnection = MockControl.createNiceControl
(Connection.class);>

```

Chapter 5, Page 77, Last paragraph, change last sentence from:

```

<Just unzip the download and make the jars available on your project's
classpath (in our case, we placed.ibatis-sqlmap.jar,.ibatis-dao.jar and.ibatis-
common.jar in the /lib folder of our project directory.> to

```

```

<Place the IBATIS provided jars (.ibatis-sqlmap.jar,.ibatis-dao.jar and.ibatis-
common.jar) as well as the Spring-provided jdom.jar (in Spring's /lib folder) in
the /war/WEB-INF/lib folder of your project directory.>

```

Chapter 5, Page 77, change the URL in the last paragraph from:

<http://www.ibatis.com/common/sqlmaps.html> to

<http://www.ibatis.com>

Page 79, near example 5-2, either right before or right after it, add the following two examples. You can add this sentence before them all: "Also included here are the mappings for Customer and Reservation, as well."

Customer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE sql-map
```

```
  PUBLIC "-//IBATIS.com//DTD SQL Map 1.0//EN"
```

```
  "http://www.ibatis.com/dtd/sql-map.dtd">
```

```
<sql-map name="Customer">
```

```
  <result-map name="result" class="com.springbook.Customer">
```

```
    <property name="custId" column="custId" columnIndex="1"/>
```

```
    <property name="firstName" column="firstname" columnIndex="2"/>
```

```
    <property name="lastName" column="lastname" columnIndex="3"/>
```

```
  </result-map>
```

```
  <mapped-statement name="getCustomers" result-map="result">
```

```
    select
```

```
      custId,
```

```
      firstName,
```

```
      lastname
```

```
    from customers
```

```
  </mapped-statement>
```

```
  <mapped-statement name="getCustomer" result-map="result">
```

```
    select
```

```
      custId,
```

```
      firstName,
```

```
      lastname
```

```
    from customers
```

```
    where custId = #value#
```

```
  </mapped-statement>
```

```
</sql-map>
```

Reservation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE sql-map
```

```
  PUBLIC "-//IBATIS.com//DTD SQL Map 1.0//EN"
```

```
  "http://www.ibatis.com/dtd/sql-map.dtd">
```

```
<sql-map name="Reservation">
```

```
  <result-map name="result" class="com.springbook.Reservation">
```

```
    <property name="reservationId" column="resId" columnIndex="1"/>
```

```
    <property name="bike" column="bikeId" columnIndex="2"/>
```

```
    <property name="customer" column="custId" columnIndex="3"/>
```

```
    <property name="reservationDate" column="resDate" columnIndex="4"/>
```

```
  </result-map>
```

```
  <mapped-statement name="getReservations" result-map="result">
```

```
    select
```

```
      resId,
```

```
      bikeId,
```

```
      custId,
```

```
      resDate
```

```
    from reservations
```

```
  </mapped-statement>
```

```
  <mapped-statement name="getReservationsForCustomer" result-  
map="result">
```

```
    select
```

```
      resId,
```

```
      bikeId,
```

```
      custId,
```

```
      resDate
```

```
    from reservations
```

```
    where custId = #value#
```

```
  </mapped-statement>
```

```
  <mapped-statement name="getReservationsForBike" result-map="result">
```

```
    select
```

```
      resId,
```

```
      bikeId,
```

```
      custId,
```

```
      resDate
```

```
    from reservations
```

```
    where bikeId = #value#
```

```
</mapped-statement>
```

```
<mapped-statement name="getReservationsForDate" result-map="result">
```

```
  select
    resId,
    bikeId,
    custId,
    resDate
  from reservations
  where resDate = #value#
```

```
</mapped-statement>
```

```
<mapped-statement name="getReservation" result-map="result">
```

```
  select
    resId,
    bikeId,
    custId,
    resDate
  from reservations
  where resId = #value#
```

```
</mapped-statement>
```

```
</sql-map>
```

Page 81, Example 5-4, change line from:

```
<value>classpath:/ibatis.config</value>
```

to

```
<value>/WEB-INF/ibatis.config</value>
```

Page 81, New example right after 5-4, add this file:

Example 5-5 ibatis.config

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE sql-map-config
```

```
  PUBLIC "-//iBATIS.com//DTD SQL Map Config 1.0//EN"
```

```
  "http://www.ibatis.com/dtd/sql-map-config.dtd">
```

```
<sql-map-config>
```

```
  <sql-map resource="Bike.xml" />
```

```
  <sql-map resource="Customer.xml" />
```

```
  <sql-map resource="Reservation.xml" />
```

```
</sql-map-config>
```

Chapter 5, Page 81, Example 5-4, change label to:

rentabikeApp-servlet.xml

Chapter 5, Page 84, The sentence right after Example 5-5, change from:

<Remember to make the appropriate jar files available to Ant (kodo-jdo.jar, jdo-1.0.1.jar and jakarta-commons-lang-1.0.1.jar).> to

<You'll also add path elements to your Ant build file for kodo-jdo.jar, jdo-1.0.1.jar and jakarta-commons-lang-1.0.1.jar.>

Chapter 5, Page 88, Example 5-10, change label to:

rentabikeApp-servlet.xml

Chapter 5, Page 90, Paragraph just under "How Do I Do That?", replace paragraph with:

Now that you've configured Spring with an ORM, you know the basic flow, with a persistent domain model and a facade. Spring ships with Hibernate dependencies, you can just copy them from Spring's /dist folder to your /lib folder: hibernate2.jar, aopalliance.jar, cglib-full-2.0.2.jar, dom4j.jar, ehcache-1.1.jar, and odmng.jar.

Chapter 5, Page 91, Example 5-14, change label to:

rentabikeApp-servlet.xml

Chapter 5, Page 91, Example 5-14, second line of code:

add a "<" as the first character of the line

Chapter 6, Page 100, Right after Example 6-1, add this paragraph and new

Examples:

The logger uses a new persistent class called LogEvent, whose class file and mapping file are shown in Example 6-2 and 6-3.

Example 6-2 LogEvent.java

```
package com.springbook;  
import java.util.Date;
```

```
public class LogEvent {  
    private int eventId;  
    private String methodName;  
    private Date dateTime;  
    private String message;  
  
    public LogEvent() {}  
    public LogEvent(String methodName, Date date, String message) {  
        this.eventId = -1;  
        this.methodName = methodName;  
        this.dateTime = date;  
        this.message = message;  
    }  
  
    public LogEvent(String methodName, Date date) { this(methodName, date,  
    "");}  
  
    public String getMessage() { return message; }  
  
    public void setMessage(String message) { this.message = message; }  
  
    public int getEventId() { return eventId; }  
}
```

```

public void setEventId(int eventId) { this.eventId = eventId; }

public String getMethodName() { return methodName; }

public void setMethodName(String methodName) { this.methodName =
methodName; }

public Date getDateTime() { return dateTime; }

public void setDateTime(Date dateTime) { this.dateTime = dateTime; }
}

```

Example 6-3 LogEvent.hbm.xml

```

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

  <class name="com.springbook.LogEvent" table="eventlog">
    <id name="eventId" column="eventId" type="java.lang.Integer"
    unsaved-
    value="-1">

      <generator class="native"></generator>
    </id>
    <property name="methodName" column="methodname" type="string"/>
    <property name="dateTime" column="datetime" type="date"/>
    <property name="message" column="message" type="string"/>
  </class>
</hibernate-mapping>

```

Chapter 6, Page 102, Example 6-3, change label to:

rentabikeApp-servlet.xml

Chapter 6, Page 103, Example 6-3, at end of example:

at the end of the example is an XML element that starts with <bean id="datasource"... This entire section either needs to come before the </beans> tag just above it, or be deleted.

Chapter 6, Page 105, Example 6-5, change label to:

rentabikeApp-servlet.xml

Chapter 6, Page 107, Example 6-8, change label to:

rentabikeApp-servlet.xml

Chapter 6, Page 109, Example 6-11, change label to:

rentabikeApp-servlet.xml

Chapter 6, Page 111, Example 6-16, change label to:

MockInterceptorTest.java

Chapter 6, Page 112, Example 6-18, change label to:

MockInterceptorTest.java

Chapter 6, Page 113, Example 6-18, in the code:

in the section of code that starts with the comment "// and once after", remove the line of code that reads:

```
sessionControl.setMatcher(MockControl.ALWAYS_MATCHER);"
```

Chapter 7, Page 121, Paragraph right under "Transactions on Multiple Databases", replace whole paragraph with:

If you have to refactor a simple application to use multiple resources, Spring's pluggable transaction strategies can save you a whole lot of effort. In this example, you're going to use JTA transactions to span multiple databases. For this example to work, your application **must** be running in a JTA-aware J2EE container.

Chapter 7, page 123, Example 7-6, bottom of code, change the lines:

```
<public MonetaryTransaction() {
```

```
}>
```

to

```
<public MonetaryTransaction() {
```

```
this(0.0, 0);
```

```
}>
```

Chapter 7, page 127, paragraph after How Do I Do That, in middle of paragraph:

where it says "copy acegi-securiyt-catalina-server.jar", the word security is misspelled.

Chapter 7, page 129, first sentence after Example 7-16, change from:

```
<The class called DAOAuthenticationProvider ACEGI class implements...>
```

to

```
<The DAOAuthenticationProvider ACEGI class implements...>
```

Chapter 7, page 130, Paragraph after example 7-18:

in the middle of the paragraph, change ".htm" to ".bikes".

Chapter 7, page 131, Example 7-19:

change ".htm" to ".bikes"

Chapter 7, Page 123, Example 7-8, change label to:

```
rentabikeApp-servlet.xml
```

Chapter 7, Page 126, Example 7-13, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 128, Example 7-15, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 129, Example 7-16, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 129, Example 7-17, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 130, Example 7-18, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 130, Example 7-18, in middle of code, change line from:
`\A.*/*.htm\Z=ROLE_USER`
to
`\A.*/*.bikes\Z=ROLE_USER`

Chapter 7, Page 131, Example 7-19, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 131, Example 7-20, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 131, Paragraph before Example 7-20, replace whole paragraph with:

Finally, configure your servlet filters themselves (Example7-20). You need three: a SecurityEnforcementFilter, an AutoIntegrationFilter and anAuthentication-ProcessingFilter. The enforcement filter uses FilterSecurity- Interceptor to determine if access to a resource has been granted, and the AuthenticationProcessingFilter redirects unauthenticated users to the login page.

The SecurityEnforcementFilter also needs access to a configured AuthenticationProcessingFilterEntryPoint, which simply stores the URL for the login page and whether or not it requires HTTPS.

Chapter 7, Page 132, Example 7-20, at bottom of code, add this line as last line of example:

```
<bean  
id="autoIntegrationFilter"class="net.sf.acegisecurity.ui.AutoInt  
egrationFilter"></bean>
```

Chapter 7, Page 135, Example 7-23, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 135, Example 7-24, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 136, Example 7-25, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 136, Example 7-26, change label to:
rentabikeApp-servlet.xml

Chapter 7, Page 140, Example 7-30, change label to:
rentabikeApp-servlet.xml

CHAPTER 8

Page 142, change <flows all over the world> **to** <flows across the United States>

Change <It would be much more useful to send an email message to you when something broke. Let's use Spring's email abstraction to do that.>
to

<An email message would be more useful. Let's use Spring's email abstraction to send an email message to a user when something breaks.>

Page 143, bottom of page: strike <NOTE TO PROD: NoteWarning was here.>

Page 144: MAJOR CHANGE

The order of the statements in example 8-3 is wrong. The get and set methods must appear inside of the class definition. So...change

-----Incorrect version -----

```
public MailSender getMailSender( ) {
return mailSender;
}
public void setMailSender(MailSender mailSender) {
this.mailSender = mailSender;
}
public SimpleMailMessage getMailMessage( ) {
return mailMessage;
}
public void setMailMessage(SimpleMailMessage mailMessage) {
this.mailMessage = mailMessage;
}

public class ExceptionInterceptor implements ThrowsAdvice {
public void afterThrowing(Method m, Object[] args,
Object target, Exception ex) {
```

TO ----- CORRECT VERSION -----

```
public class ExceptionInterceptor implements ThrowsAdvice {
private MailSender mailSender;
private SimpleMailMessage mailMessage;

public SimpleMailMessage getMailMessage() {
return mailMessage;
}
```

```
public void setMailMessage(SimpleMailMessage mailMessage) {
    this.mailMessage = mailMessage;
}
```

```
public MailSender getMailSender() {
    return mailSender;
}
```

```
public void setMailSender(MailSender mailSender) {
    this.mailSender = mailSender;
}
```

-----END CORRECT VERSION -----

Page 145. Change Example 8-4 continued label <rentABike-servlet.xml> to <rentabikeApp-servlet.xml>

Page 145. Paragraph right after Example 8-4. Change paragraph to read: Now, do something that will cause an exception. For example, shut down the database while the application is running. The JDBC layer should fire an exception next time you try to access the database, which in turn should fire an email message.

Change <rentABikeTarget> to <rentaBikeTarget> in both places

Page 146. Example 8-7. Change label <RentABike-servlet.xml> to <rentabikeApp-servlet.xml>

Page 148. At the end of the last paragraph ending <the JMS connection, as in Example 8-9.> Add <Point the brokerURL to the MQ machine and port.>

Page 148. Change the code fragment <vm://localhost> to <<tcp://yourMqHost:yourMqPort>>

Chapter 8, Page 143, Example 8-1, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 143, Example 8-2, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 144, Example 8-4, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 146, Example 8-5, add at bottom of example:

```
<servlet-mapping>  
  <servlet-name>remoting</servlet-name>  
  <url-pattern>/remoting/*</url-pattern>  
  
</servlet-mapping>
```

Chapter 8, Page 146, Example 8-7, change line from:

```
<value>http://YOUR_HOST:8080/RentABike</value>
```

to

```
<value>http://YOUR_HOST:8080/rentaBike/remoting/RentABike</value>
```

Chapter 8, Page 148, Example 8-9, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 149, Example 8-10, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 149, Example 8-11, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 150, Example 8-14, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 150, Sentence right before Example 8-15, change from:

<You can build a standalone consumer (Example8-15), so you can watch objects as the application creates them (heavily influenced by the sample receiver in theActiveMQ download).> to

<Example 8-15 is a standalone consumer that requires a JNDI-aware container to find and observe the queue. Alternatively, you could use the ConsumerTool in ActiveMQ's /example folder to watch the messages as they are created.>

Chapter 8, Page 153, Example 8-16, at bottom of code:

add a final "}"

Chapter 8, Page 153, Example 8-17, change label to:

rentabikeApp-servlet.xml

Chapter 8, Page 154, Example 8-17, change line of code from:

<bean id="rentaBikeTarget" class="com.springbook.HibRentABike"> to

<bean id="rentaBikeTarget" class="com.springbook.JMSTestRentABike">

Chapter 8, Page 15, Example 8-18, change line of code from:

<assertEquals("La Bruja", b.getManufacturer());> to

<assertEquals("La Bruja", b.getModel());>