Value Beans

Value beans are useful even without EJB. They are handy for capturing form input, because the <jsp:setProperty> JSP action automatically sets all properties with names corresponding to request parameter names, as described in Chapter 8. In addition, the <jsp:getProperty> action and the JSP EL let you include the property values in the response without using scripting elements.

Another benefit of value beans is that they can be used to minimize expensive database accesses for entities that rarely change their value. By placing a value bean in the application scope, all users of your application can use the cached value instead. Example 20-1 shows the source code for the ProductBean used in Chapter 10 to represent products in an online shopping application. This is a pure value bean, with only property accessor methods, that can represent data retrieved from a database.

Example 20-1. ProductBean

```
package com.ora.jsp.beans.shopping;

import java.io.*;

public class ProductBean implements Serializable {
    private String id;
    private String name;
    private String descr;
    private float price;

    public String getId( ) {
        return id;
    }

    public String getName( ) {
        return name;
    }

    public String getDescr( ) {
        return descr;
    }

    public float getPrice( ) {
        return price;
    }

    void setId(String id) {
        this.id = id;
    }

    void setName(String name) {
        this.name = name;
    }

    void setDescr(String descr) {
```

Example 20-1. ProductBean (continued)

```
        this.descr = descr;
    }

    void setPrice(float price) {
        this.price = price;
    }
}
```

This bean is created and initialized by the single instance of the CatalogBean. All setter methods have package accessibility, while the getter methods are public. Using package accessibility for the setter methods ensures that only the CatalogBean can set the property values. For instance, a JSP page can read the product information but not change the price.

Another example of a value bean is the UserInfoBean introduced in Chapter 8. Part of this bean is shown in Example 20-2. Besides encapsulating the property values of the entity it represents, it also provides methods for validation of the data.

Example 20-2. Part of the UserInfoBean

```
package com.ora.jsp.beans.userinfo;

import java.io.*;
import java.util.*;
import com.ora.jsp.util.*;

public class UserInfoBean implements Serializable {
    // Validation constants
    private static String DATE_FORMAT_PATTERN = "yyyy-MM-dd";
    private static String[] GENDER_LIST = {"m", "f"};
    private static String[] FOOD_LIST = {"z", "p", "c"};
    private static int MIN_LUCKY_NUMBER = 1;
    private static int MAX_LUCKY_NUMBER = 100;

    // Properties
    private String birthDate;
    private String emailAddr;
    private String[] food;
    private String luckyNumber;
    private String gender;
    private String userName;

    public String getBirthDate() {
        return (birthDate == null ? "" : birthDate);
    }

    public void setBirthDate(String birthDate) {
        this.birthDate = birthDate;
    }

    public boolean isBirthDateValid() {
```

Example 20-2. Part of the UserInfoBean (continued)

```
        boolean isValid = false;
        if (birthDate != null &&
            StringFormat.isValidDate(birthDate, DATE_FORMAT_PATTERN)) {
            isValid = true;
        }
        return isValid;
    }
    ...
```

In addition to the setter and getter methods for the birthDate property, the UserInfoBean includes a separate method for validation. It follows the naming conventions for a Boolean getter method, so it can be used in an EL expression to test if the value is valid. The getter method returns an empty string in case the property is not set. Without this code, a <jsp:getProperty> action adds the string null to the response. The JSTL <c:out> action and EL expressions, on the other hand, automatically convert a null value to the empty string, so this type of code is not needed in an application that always uses <c:out> or the EL for adding bean property values to the response.

This type of getter, setter, and validation method combo represents all UserInfoBean properties. In addition, the bean includes other validation and test methods, all of them posing as Boolean read-only getter methods. They are shown in Example 20-3.

Example 20-3. Validation and test methods

```
public boolean isValid() {
    return isBirthDateValid() && isEmailAddrValid() &&
        isFoodValid() && isLuckyNumberValid() &&
        isGenderValid() && isUserNameValid();
}

public boolean isPizzaSelected() {
    return isFoodTypeSelected("z");
}

public boolean isPastaSelected() {
    return isFoodTypeSelected("p");
}

public boolean isChineseSelected() {
    return isFoodTypeSelected("c");
}

private boolean isFoodTypeSelected(String foodType) {
    if (food == null) {
        return false;
    }
    boolean selected = false;
    for (int i = 0; i < food.length; i++) {
        if (food[i].equals(foodType)) {
```