The last piece of the request-handling framework deals with how the view adapts the data stored in the model for presentation. The View Helper pattern describes the ways this can be done using custom tags and JavaBeans to adapt the model to the presentation inside the view. This chapter will discuss these methods and build a few useful view helpers that you can add to your own toolkit.

Defining the Pattern

The View Helper pattern specifies that you use helpers to adapt model data to the presentation layer of an application. Typically, the presentation layer contains several JSP pages. These pages consist of HTML and images used to present content to the user. A problem arises, however, when these pages need to display dynamic content stored in the model. You'd probably like to avoid embedding Java code within these pages to display model data, so you need to employ some helpers to do the work for you.

Recall from the previous chapter that the controller servlet attaches the model to the request as an attribute. To get at this model from within the JSP page, you have three choices. You could embed Java code as JSP scriptlets, you could use the EL, or you could use a helper to extract the data for you. In keeping with the idea of separating presentation from business logic, it makes sense to employ several helpers to adapt the model to the presentation rather than clutter the presentation code with Java code (see Figure 8-1).

Figure 8-1. View Helper pattern

A. Patzer, Foundations of JSP Design Patterns
© Andrew Patzer, Matthew Moodie 2004
As you can imagine, helpers make it a lot easier for page designers to develop the presentation layer by replacing Java code with simple-to-use helpers. Of course, developers can accomplish this only if they publish a catalog of the helpers and describe how to use them. However, if the page designers develop their pages before the application designer is ready to give them a model for the helpers with which to work, then there's a problem. A useful technique to address this problem is to code into the helper a set of dummy data to display when no model is present. An alternate approach would be to provide a dummy model with which to work. Either way, the page designer shouldn't be held up while waiting for the developers.

Using helpers has the following advantages:

- Presentation components are standardized, creating a consistent look-and-feel for the application.
- Java code is abstracted away from page designers, giving them an easy-to-use set of helpers to access the model.
- You can create helpers to display dummy data if no model exists, thus letting page designers continue development regardless of the readiness of the application.
- Helpers provide a clean separation between presentation and business data by acting as intermediaries between the two.

## Implementing View Helper Pattern Strategies

When developing helpers for JSP pages, you have two choices. You could use either JavaBeans or custom tags. Which you choose really depends on the type of data you're trying to adapt. Typically, you use JavaBeans to extract individual pieces of data, and custom tags are better suited for working with sets of data. However, it's important to point out that you can use either for both types of data.

### Implementing the JavaBean Helper Strategy

You can implement helpers as JavaBeans within a JSP page. These helpers are typically easy to use when extracting and formatting single text items. The built-in JSP tags that enable you to work with JavaBeans are simple and intuitive to use. Using JavaBeans, as you may recall from Chapter 3, involves simply declaring the bean and referencing it later using the special tags as follows:

```jsp
declare bean
<%-- Declare bean --%>
<jsp:useBean id="myBean" class="jspBook.util.myBean"/>

<get first name from bean
<%-- Get first name from bean --%>
Hello <jsp:getProperty name="myBean" property="firstName"/>
, welcome to Acme Products' online store!
```

JavaBeans can do more than simply retrieve data items from the model. They can also format specific data items, perform calculations, or generate large blocks of content. Ideally, they're suited to retrieving data items using the built-in JSP tags. If you do much more with them, then your JSP may begin to get cluttered with too much Java code, no matter how much you try to use