2- Java beans

Juan M. Gimeno, Josep M. Ribó

February, 2008

Contents

Tools for model-view separation: tags and beans

2. Java beans. Contents

Java beans. Introduction: The problem

With the elements we have discussed up to now we are able to design simple web applications.

However, when these applications become more complex, more abstract tools are needed (classes, design patterns....)

Goal of beans: Incorporate user-defined classes into web applications
In particular,

- Provide easy ways to define and use classes within web applications
- Work with classes as reusable units which may be incorporated modularly into several web applications

Java beans. Definition

A bean is a Java class provided with the following features:

- It has a constructor with no arguments
- It implements either the interface *Serializable* or *Externalizable*
- It may have both *methods* and *properties*

Bean properties

- A bean may define (however, it is not compulsory) a set of properties
- Properties usually correspond to class attributes (which must be private)
- The bean provides *get* and/or *set* methods for its properties
- A *read/write* property called *propertyName* is accessed by means of the methods:

  `getPropertyName()`
  `setPropertyName(value)`

- A *read-only* property called *propertyName* is accessed by means of the method:

  `getPropertyName()`

  No `setPropertyName(value)` will be defined for a *read-only* property

Example 1: The Book bean

```
package elems;
import java.io.*;
public class Book implements Serializable{
    private String title;
    private String author;
    private String comment;
    private int id;
    private int price;

    public Book(){}

    public void setTitle(String t)
    {
       title=t;
    }
    public void setAuthor(String a)
    {
       author=a;
    }
    public void setComment(String c)
    {
       comment=c;
    }
    public void setId(int i)
    {
       id=i;
    }
```

continues.....

....continues

```
    public void setPrice(int i)
    {
      price=i;
    }
    public String getTitle()
    {
      return title;
    }
    public String getAuthor()
    {
      return author;
    }
    public String getComment()
    {
      return comment;
    }
    public int getId()
    {
      return id;
    }
    public int getPrice()
    {
      return price;
    }
}
```

Example 1: The `Book` bean (2)

Remarks:

- The `Book` bean models a book with all the features that are necessary for the book store
- `TheBook` bean has been defined within the package `elems`. For this reason the complete name of this class is `elems.Book`
- The bean has defined a set of private attributes: `title`, `author`, `comment`, `id`, `price`
- All these attributes have a read/write property associated and accessed by the respective `get/set` methods (e.g., `get/setTitle()`)
- It is **not** necessary that:
  1. Each property corresponds to an attribute
  2. Each attribute has a property associated
  3. Each property has `read/write` methods

Example 2: The `Cart` bean

```
package elems;
import java.io.*;
public class Cart implements Serializable{
    private Book[] contents;
    private int nitems;
    private int total;
    private int current;

    public Cart(){
      contents=new Book[10];
      nitems=0;
      total=0;
    }
    public boolean addBook(Book b)
    {
        if (nitems<10){
          contents[nitems]=b;
          nitems++;
          total=total+b.getPrice();
          return false;
        }
        else return true;
    }
```

continues.....

....continues

```
    public Book getBook(int i)
    {
        if (i<nitems && i>=0) return contents[i];
        else return null;
    }
    public Book getNextBook()
    {
      if (current<nitems){
        current++;
        return contents[current-1];
      }
      else return null;
    }
    public Book getFirstBook()
    {
       current=1;
       if (nitems>0){
         return contents[0];
       }
       else return null;
    }
    public int getNBooks()
    { return nitems;}
    public int getTotal()
    {  return total;  }
}
```

Example 2: The `Cart` bean (2)

Remarks:

- The `Cart` bean models a shopping cart
- `TheCart` bean has been defined within the package `elems`. For this reason the complete name of this class is `elems.Cart`
- The bean uses the bean `Book` defined within the same package
- The bean has defined a set of private attributes: `contents`, `nitems`, `total`, `current`. However, there are not properties defined fora each one of them
- On the other hand, some properties have been defined which do not correspond to attributes:

  `getNextBook()`, `getFirstBook()`
- Finally, other methods have been defined for the bean `Cart`:

  `addBook(Book)`, `getBook(int)`
- The properties of the bean `Cart` are *read-only*:

  `nBooks`, `total`, `firstBook`, `nextBook`

Using beans in a web application

- The beans may be used as usual classes in a web application (in the Java code part:

```
<%
 Book b=new Book();

 b.setTitle(request.getParameter(title));
  ..............
%>
```

- The package of the bean should be imported

```
<%@ page import="elems.*" %>
```

- **The bean class should be located in the directory `WEB-INF/classes` (or `WEB-INF/lib`, if it is packaged in a jar file)**

  The whole package path must be included in the `classes` directory!!!!

- We will see afterwards that JSP defines some tags to work with beans in a more abstract way (`useBean`, `set/getProperty`....)

Example: The bookstore

Example location: beans+tags/examples/ex2.1

This example simulates a very simple virtual bookstore called *Alexandria*

The customer may:

- See the book catalogue available at the bookstore
- Add a book to the shopping cart
- See the contents of the shopping cart
- Reset the shopping cart

It is also possible to add books to the catalogue

Payment procedures have not been implemented

Example: The Bookstore. Files involved

- `showcatalog.jsp`

  It shows the catalog of books to sell. It gets this catalog from a database `Books` which has just one table with the fields: `id, title, author, price, comment`

- `showcontents.jsp`

  It shows the contents of the shopping cart

- `additem.jsp`

  It allows the addition of an item (a book) to the shopping cart. It also shows the contents of the shopping cart after the addition of the new item (by including *showcontents.jsp*)

- `resetCart.jsp`

  It empties the cart, discarding all the previously included items.

- `product.jsp`

  It shows all the features of a specific book (title, author, price and comments)

- `errorHandler.jsp`

  It is the error handling page. Whenever an error occurs, a forward to this page is launched and the cause of the error displayed.

  Two errors have been considered:

  – *Cart full:* The client tries to add a new book into a full shopping cart
  – *Expired session:* The session is no longer valid

Example: The Bookstore. Beans involved

Two java beans have been considered:

- `elems.Book`: Models a book (with author, title, id, price...)
- `elems.Cart`: Models a shopping cart

Notice that both beans belong to the package `elems`

They will be located at `WEB-INF/classes/elems`

The implementation of both beans has been shown in previous slides

Example: The bookstore

File: showcatalog.jsp

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page session="true" %>
<%@ page errorPage="errorHandler.jsp" %>
<%@ include file="header.html"%>

<h2> Book catalog </h2><p><p>
<%
  String DRIVER= "org.postgresql.Driver";
  String URL= "jdbc:postgresql://localhost:5432/Books";
  Connection con = null;
  Statement st = null;
  ResultSet rs = null;
  try{
   st=(Statement)session.getAttribute("dbstatement");
   if (st==null){
    Class c=Class.forName(DRIVER);
    con=DriverManager.getConnection(URL,"josepma","");
    st= con.createStatement();
    session.setAttribute("dbstatement",st);
   }
   String sql=""
     +" select bookid,title,author,price "
     +" from book";
   rs=st.executeQuery(sql);
%>
```

continues.....

.....continues

```
<table border=1 cellpadding=3>
<tr> <th>Title</th> <th>Author</th> <th>Price</th> </tr>
<%
    while (rs.next()) {
      int id=rs.getInt("bookid");
      String title=rs.getString("title");
      String author=rs.getString("author");
      double price=rs.getDouble("price");
%>
<tr>
  <td align="right">
   <a href=<%="\"product.jsp?id="+id+"\">"+ title %></td>
  <td align="right"><%= author %></td>
  <td align="right"><%= price %> </td>
</tr>
<%
 }
}
catch (ClassNotFoundException e){
    throw new Exception("JDBC driver not found");}
catch (SQLException e){
    throw new Exception("Unexpected database error");}
finally{
  if (rs != null) rs.close();}
%>
</table>
```

Comments to `showcatalog.jsp`

- In order to avoid establishing a connection every time the database is accessed, we use always the same connection. We keep the connection in a session attribute called `dbstatement`

  However, if this is the first access to the database, a new connection must be established:

```
Statement st=(Statement)
     session.getAttribute("dbstatement");
if (st==null){
 Connection con=null;
 Class c=Class.forName(DRIVER);
 con=DriverManager.getConnection(URL,"josepma","");
 st= con.createStatement();
 session.setAttribute("dbstatement",st);
}
```

- Notice that a common header is included (`header.html`)

- Notice that session suport is requested (page session="true")

- The directive

  ```
  <%@ page errorPage="errorHandler.jsp" %>
  ```

  indicates that any exception raised in this page will be forwarded to the page `errorHandler.jsp`

  In this case an exception may be thrown:

  – If the JDBC driver is not found and
  – If some unexpected database error occurs

Example: The bookstore

File: product.jsp

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page import="elems.*" %>
<%@ page errorPage="errorHandler.jsp" %>
<%@ page session="true" %>
<%@ include file="header.html"%>
<%
 ResultSet rs = null; Statement st = null;
 try{
  Book b=new Book();
  st=(Statement)session.getAttribute("dbstatement");
  if (st==null) {
    throw new Exception("Sorry, the session has expired");
  }
  String bookId=request.getParameter("id");
  String sql=""
     +" select title,author,price,comments "
     +" from book where bookid="+bookId;
  rs=st.executeQuery(sql);
  while (rs.next()) {
      b.setId(Integer.parseInt(bookId));
      b.setTitle(rs.getString(1));
      b.setPrice(rs.getDouble(3));
      b.setAuthor(rs.getString(2));
      b.setComment(rs.getString(4));
 %>
```

continues.....

.....continues

```
   <b>Title:</b> <%=b.getTitle() %><p>
   <b>Author:</b> <%= b.getAuthor()%><p>
   <b>Price:</b> <%= b.getPrice()%><p>
   <b>Comments:</b> <%= b.getComment()%><p>
<%
 }
 session.setAttribute("book",b);
}
catch(SQLException e){throw new Exception
             ("Unexpected database error");
}
finally{
 if (rs != null) rs.close();
}
%>
<a href="additem.jsp">
Add this item to your shopping cart </a>
```

Example: The bookstore

Comments to the file `product.jsp`

- Notice that the package of the classes `Book` and `Cart` (i.e., `elems`) is included

- If the database connection is not associated to the session attribute `dbstatement`, this means that the session has expired

  If this is the case, an exception will be generated and forwarded to `errorHandler.jsp`:

```
Statement st=(Statement)
     session.getAttribute("dbstatement");
if (st==null) {
  throw new Exception
             ("Sorry, the session has expired");
}
```

Example: The bookstore

File: showcontents.jsp

```
<%@ page session="true" %>
<%@ include file="header.html"%>
<%@ page import="elems.*" %>
<%
 Cart shopcart;
 if (session.getAttribute("shoppingCart")!=null){
    shopcart=(Cart) session.getAttribute("shoppingCart");
 }
 else{ shopcart=new Cart();}
%>
 <h2> Cart contents</h2> <p><p>
<%
 int i;
 for (i=0;i<shopcart.getNBooks();i++){
%>
   <%= shopcart.getBook(i).getTitle()%>.......
   <%= shopcart.getBook(i).getPrice()%><p>
<%
 }
%>
<p>
<b>total=<%=shopcart.getTotal()%></b>
```

Example: The bookstore

Comments to the file `showcontents.jsp`

- It is important to consider a session attribute `shoppingCart` in order to keep the contents of the shopping cart of a specific customer throughout the whole session

  If that session attribute does not exist, this means that the customer has not bought any book in this session. In that case, an empty cart will be shown

- Notice the use of the bean `Cart`

Example: The bookstore

File: additem.jsp

```
<%@ page session="true" %>
<%@ page import="elems.*" %>
<%@ page errorPage="errorHandler.jsp" %>
<%
  Book b=(Book) session.getAttribute("book");
  Cart shopcart;
  boolean cartfull;
  if (session.getAttribute("shoppingCart")==null){
    shopcart=new Cart();
  }
  else{
    shopcart=(Cart) session.getAttribute("shoppingCart");
  }
  cartfull=shopcart.addBook(b);
  session.setAttribute("shoppingCart",shopcart);
  if (cartfull){
    throw new Exception
                ("Sorry, your shopping cart is full");
  }
%>
<jsp:include page="showcontents.jsp" flush="true" />
```

Comments to the file `additem.jsp`

- If this is the first book bought by the customer, create a new Cart and store it as a session attribute:

```
if (session.getAttribute("shoppingCart")==null){
  shopcart=new Cart();
}
else{
  shopcart=(Cart)
    session.getAttribute("shoppingCart");
}
```

- If the shopping cart gets full, an exception is forwarded to `errorHandler.jsp`

- If the book insertion has been OK, the cart content is shown by means of a `jsp:include`

Example: The bookstore

File: resetCart.jsp

```
<%@ page session="true" %>
<%@ include file="header.html"%>
<%@ page import="elems.*" %>

<%
   session.setAttribute("shoppingCart",new Cart());

%>

 Cart empty <p><p>
```

2.6 Standard tags for using beans

In addition to declare and use beans within the Java code of a JSP page:

```
<%

  Book b=new Book();

  b.setTitle(request.getTitle());

%>
```

It is also possible to declare and use them by means of predefined JSP tags.

- `<jsp:useBean ...../>`
- `<jsp:setProperty ...../>`
- `<jsp:getProperty ...../>`

These tags correspond to scripts and provide a higher level and more abstract way to deal with beans

If possible, it is preferable to use these tags than to write direct Java code

The `useBean` tag

The `useBean` tag declares that a JSP page wants to use a specific bean class.

Two forms may be used:

```
<jsp:useBean id="beanName"
  class="fully_qualified_classname" scope="scope"/>


<jsp:useBean id="currentBook"
   class="elems.Book" scope="session" />
```

and

```
<jsp:useBean id="beanName"
  class="classname" scope="scope">
  <jsp:setProperty .../>
</jsp:useBean>
```

The `useBean` tag (2)

- **id:** The identifier of the bean instance which is to be used in this JSP page

- **scope:** The scope in which the bean instance called `id` is stored and can be accessed.

  This scope may be `application`, `session`, `request`, or `page`

- **class:** `classname` is the full qualified name of a class, such that `id` is a bean instance of that class

  - The class must belong to a package
  - The name of the class is its full name
  - If the JSP container can find an object with the identifier `id` in the scope `scope`, which is an instance of the class `classname`, this is the object that will be used.
  - If that object does not exist, a new object instance of the class `classname` with identifier `id` is created
  - Instead of `class`, it is possible to use `type`

- **type:** The name of an interface implemented by a bean instance called `id` (`type="typename"`).
  - If the JSP container can find an object with the identifier `id` in the scope `scope`, which implements the interface `typename`, this is the object that will be used in posterior calls to `id` in this scope.
  - If that object does not exist, the JSP container throws an `InstantiationException`

The `setProperty` tag

The `setProperty` tag assigns a value to a bean property

```
<jsp:setProperty name="beanName" property="propName" ........>
```

This assigns a certain value (see below) to the property `propName`

Values:

The value that is assigned to a property may be given in the following ways:

- *String constant*
- *Request parameter*
- *Java expression*

The `setProperty` tag (2)

Value assigned to the property: String constant

The property gets a string constant as its new value

```
<jsp:setProperty name="beanName"
  property="propName" value="string constant"/>

<jsp:setProperty name="currentbook"
  property="author" value="Shakespeare"/>
```

This pattern may be used:

- For properties of class `String`
- For properties of primitive type (bool, int, float, double...)

  ```
  <jsp:setProperty name="currentBook" property="price"
   value="22" />
  ```

  The JSP container generates the following code (in the translated servlet):

  ```
  currentbook.setPrice(Integer.valueof("22").intValue());
  ```

- For properties of classes that implements the `PropertyEditor` interface and, thus, they have the method `setAsText` defined

The `setProperty` tag (3)

Value assigned to the property: Request parameter

The property gets a parameter of the HTTP request as its new value. This parameter will be a string, hence the previous considerations apply

```
<jsp:setProperty name="beanName"
  property="propName" param="paramName"/>

<jsp:setProperty name="currentbook"
  property="author" param="author"/>
```

It means: The property `author` of the bean object called `beanName` will get the value of the HTTP request parameter called `author`

Two particular cases:

- The property name is the same as the parameter name (as in the previous ex.)

```
<jsp:setProperty name="currentbook"
  property="author"/>
```

- All bean properties which have the same name as request parameters are to be set (with the values of the request parameters):

```
<jsp:setProperty name="currentbook"
  property="*"/>
```

(See the example below)

SetProperty: Conversion of `String` to the specific property type

- **The request parameters sent by the client** and
- **The String literals**....

.... that are used **to set bean properties** are always of class `String`

They are converted to the actual type of the bean property by means of some operation which depends on the specific type of the bean property

The following table shows the operation which is used in each case:

Bean property type	**Operation used to convert a `String` into that type**
Bean Property that implements `PropertyEditor`	setAsText(String)
`boolean`, `Boolean`	Byte.valueOf(String)
`char`, `Character`	`String.charAt(0)`
`double`, `Double`	`Double.valueOf(String)`
`integer`, `Integer`	`Integer.valueOf(String)`
`float`, `Float`	`Float.valueOf(String)`
`long`, `Long`	`Long.valueOf(String)`
`short`, `Short`	`Short.valueOf(String)`

The `setProperty` tag (4)

Value assigned to the property: Java expression

A Java expression may be assigned to a property in the following way:

```
<jsp:setProperty name="currentBook" property="price"
value="<%= java-expression  %>" />
```

The getProperty tag

This element converts the value of the property `propName` of the bean `beanName` into a String and inserts the value into the response stream:

```
<jsp:getProperty name="beanName" property="propName"/>

<jsp:getProperty name="currentbook" property="author"/>
```

This is similar to:

```
<%
  out.write(currentbook.getAuthor());
%>
```

The name must correspond to a bean instance identifier

Example 1: A very simple example

Example location: beans+tags/examples/ex2.2

```
<%@ page import="java.text.*" %>

<html>
<h1>Exemple molt simple de beans</h1>

<p><p><p>

<jsp:useBean id="popo" class="elems.Prova"
             scope="session" />

El nombre d'accessos es:<br>

<jsp:getProperty name="popo" property="counter" />

<jsp:setProperty name="popo" property="counter" value=
       "<%=Integer.toString(
            Integer.parseInt(popo.getCounter())+1)%>" />
</html>
```

Example 2: Update the bookstore database

Example location:
It is included in the bookstore example:
beans+tags/examples/ex2.1/bdupdate.jsp

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page session="true" %>
<%@ include file="header.html"%>
<%@ page import="elems.*" %>
<%@ page errorPage="errorHandler.jsp" %>

<h2> Update book catalog </h2>
<p><p>
<jsp:useBean id="book" scope="session"
                        class="elems.Book" />
    <jsp:setProperty name="book" property="*" />
<%
  String DRIVER= "org.postgresql.Driver";
  String URL= "jdbc:postgresql://localhost:5432/Books";
  Connection con = null;
  Statement st = null;
```

continues.....

.....continues

```
  try{
    Class c=Class.forName(DRIVER);
    con=DriverManager.getConnection(URL,"josepma","");
    String sql=""
     +"insert into book (title, author, price, comments)"
     +" values ('"
     +book.getTitle()+"','"
     +book.getAuthor()+"',"
     +book.getPrice()+",'"
     +book.getComment()+"')";
    st= con.createStatement();
    int j=st.executeUpdate(sql);
  }
  catch (ClassNotFoundException e){
     throw new Exception("JDBC driver not found");
  }
  catch (SQLException e) {
     throw new Exception("Unexpected database error");
  }
  finally{
    if (st != null) st.close();
    if (con != null) con.close();
  }
%>
```

continues.....

.....continues

```
<p>
Insertion done:
<p><p>
<jsp:getProperty name="book" property="title" />
<p>
<jsp:getProperty name="book" property="author" />
<p>
<jsp:getProperty name="book" property="price" />
```