11.13 Comparing JSP Expressions, Scriptlets, and Declarations

This section contains several similar examples, each of which generates random integers between 1 and 10. They illustrate the difference in how the three JSP scripting elements are typically used. All the pages use the `randomInt` method defined in Listing 11.13.

Listing 11.13 RanUtilities.java

```
package coreservlets; // Always use packages!!

/** Simple utility to generate random integers. */

public class RanUtilities {

  /** A random int from 1 to range (inclusive). */

  public static int randomInt(int range) {
    return(1 + ((int)(Math.random() * range)));
  }

  /** Test routine. Invoke from the command line with
   *  the desired range. Will print 100 values.
   *  Verify that you see values from 1 to range (inclusive)
   *  and no values outside that interval.
   */

  public static void main(String[] args) {
    int range = 10;
    try {
      range = Integer.parseInt(args[0]);
    } catch(Exception e) { // Array index or number format
      // Do nothing: range already has default value.
    }
    for(int i=0; i<100; i++) {
      System.out.println(randomInt(range));
    }
  }
}
```

Example 1: JSP Expressions

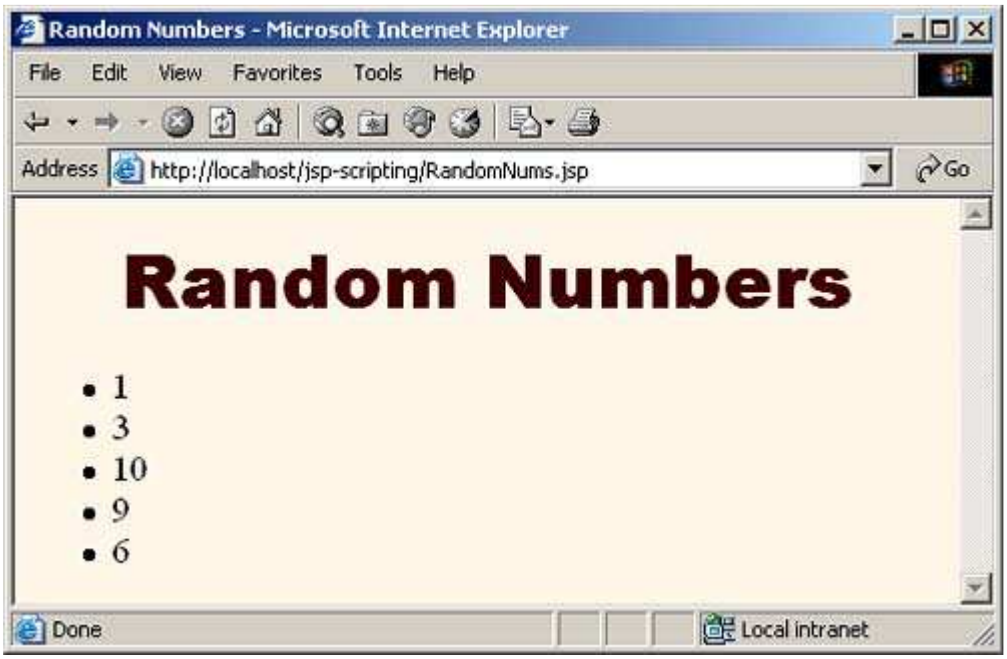
In the first example, the goal is to output a bulleted list of five random integers from 1 to 10. Since the structure of this page is fixed and we use a separate helper class for the `randomInt` method, JSP expressions are all that is needed. Listing 11.14 shows the code; Figure 11-11 shows a typical result.

Listing 11.14 RandomNums.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random Numbers</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
```

```
<BODY>
<H1>Random Numbers</H1>
<UL>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
</UL>
</BODY></HTML>
```

Figure 11-11. Result of `RandomNums.jsp`. Different values are displayed whenever the page is reloaded.



Example 2: JSP Scriptlets

In the second example, the goal is to generate a list of between 1 and 10 entries (selected at random), each of which is a number between 1 and 10. Because the number of entries in the list is dynamic, a JSP scriptlet is needed. But, should there be a single scriptlet containing a loop that outputs the numbers, or should we use the dangling-brace approach described in Section 11.9 (Using Scriptlets to Make Parts of the JSP Page Conditional)? The choice is not clear here: the first approach yields a more concise result, but the second approach exposes the `<LI>` element to the Web developer, who might want to modify the type of bullet or insert additional formatting elements. So, we present both approaches. Listing 11.15 shows the first approach (a single loop that uses the predefined `out` variable); Listing 11.16 shows the second approach (capturing the "static" HTML into the loop). Figures 11-12 and 11-13 show some typical results.

Listing 11.15 RandomList1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random List (Version 1)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
```

```
<H1>Random List (Version 1)</H1>
<UL>
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
  out.println("<LI>" + coreservlets.RanUtilities.randomInt(10));
}
%>
</UL>
</BODY></HTML>
```

Listing 11.16 RandomList2.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random List (Version 2)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 2)</H1>
<UL>
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
%>
<LI><%= coreservlets.RanUtilities.randomInt(10) %>
<% } %>
</UL>
</BODY></HTML>
```

Figure 11-12. Result of `RandomList1.jsp`. Different values (and a different number of list items) are displayed whenever the page is reloaded.

Figure 11-13. Result of `RandomList2.jsp`. Different values (and a different number of list items) are displayed whenever the page is reloaded.

Example 3: JSP Declarations

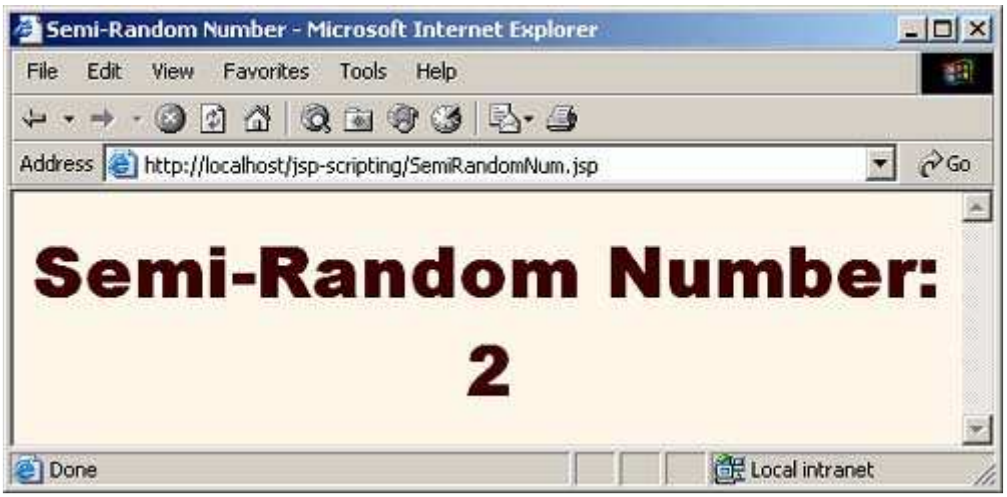
In this third example, the requirement is to generate a random number on the first request, then show the *same* number to all users until the server is restarted. Instance variables (fields) are the natural way to accomplish this persistence. The reason is that instance variables are initialized only when the object is built and servlets are built once and remain in memory between requests: a new instance is not allocated for each request. JSP expressions and scriptlets deal only with code inside the `_jspService` method, so they are not appropriate here. A JSP declaration is needed instead. Listing 11.17 shows the code; Figure 11-14 shows a typical result.

Listing 11.17 SemiRandomNumber.jsp (continued)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Semi-Random Number</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<%!
private int randomNum = coreservlets.RanUtilities.randomInt(10);
%>
<H1>Semi-Random Number:<BR><%= randomNum %></H1>
</BODY>
</HTML>
```

Figure 11-14. Result of `SemiRandomNumber.jsp`. Until the server is restarted, all clients see the same result.



[Team LiB]

PREVIOUS NEXT