Servlets, JSP, Struts and MVC
(Part I)

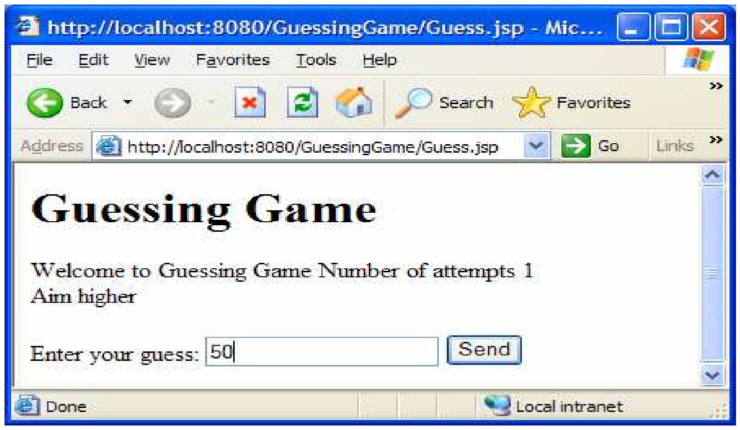
Venkat Subramaniam
venkats@agiledeveloper.com
http://www.agiledeveloper.com/download.aspx

Abstract

Servlets and JSPs bring significant advantage to web development, yet come with significant limitations. In this article, we discuss what's fundamentally wrong with servlets and JSPs, discuss the Model 1 and the MVC based Model 2 architecture, and introduce the benefits of using a framework like Struts.

Let's start with Tomcat

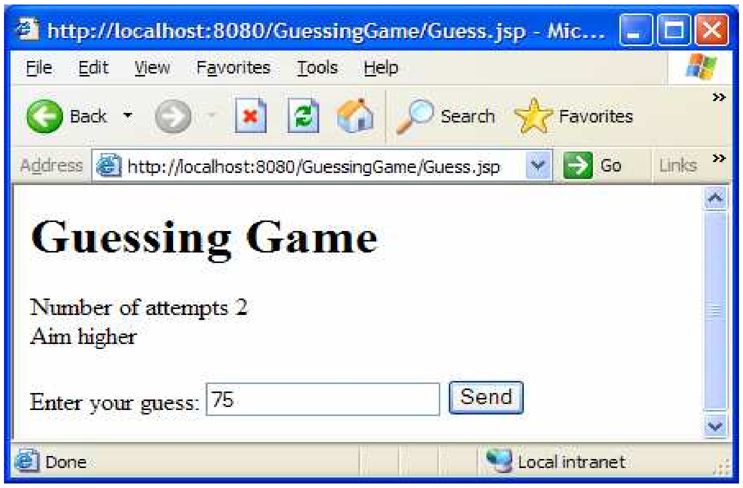
Alright, I have Tomcat[1] (4.1.24) installed on my system. Let's write a JSP page for a guessing game. I create a file named Guess.jsp in the Tomcat's webapps/GuessingGame directory. We will see it working before looking at the code. I start the Tomcat web server and visit the following URL: http://localhost:8080/GuessingGame/Guess.jsp. Here is the series of interaction with the application.


http://localhost:8080/GuessingGame/Guess.jsp - Mic...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/GuessingGame/Guess.jsp Go Links
Send
Done
Local intranet

Guessing Game

Welcome to Guessing Game Number of attempts 1
Aim higher

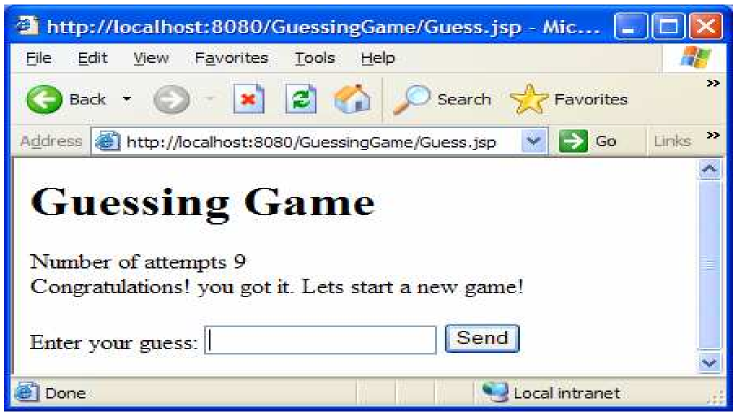
Enter your guess: 50


http://localhost:8080/GuessingGame/Guess.jsp - Mic...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/GuessingGame/Guess.jsp Go Links
Send
Done
Local intranet

Guessing Game

Number of attempts 2
Aim higher

Enter your guess: 75


http://localhost:8080/GuessingGame/Guess.jsp - Mic...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8080/GuessingGame/Guess.jsp
Go
Links
Send
Done
Local intranet

Guessing Game

Number of attempts 9
Congratulations! you got it. Lets start a new game!

Enter your guess:

The guessing game allows me to guess a number. I started with 50 and it asked me to aim higher. Then I tried 75, it asked me to aim lower. After nine attempts, I finally guessed it to be 62 (now you know why I do not buy lotto).

Now, let's look at the JSP code for this program.

```
<!-- Guess.jsp -->
<H1>Guessing Game</H1>
<%
int attempts = 1;
int target = 0;
int guess = -1;

if (session.isNew())
{
      target = (int)(Math.random() * 100);
      out.println("Welcome to Guessing Game");
      session.setAttribute("target", "" + target);
      session.setAttribute("attempts", "1");
}
else
{
      target = Integer.parseInt(
            session.getAttribute("target").toString());
      attempts =
            Integer.parseInt(
                  session.getAttribute("attempts").toString());
      attempts++;
      session.setAttribute("attempts", "" + attempts);
      guess = Integer.parseInt(request.getParameter("guess"));
}
%>
Number of attempts <%= attempts %>
<BR>
<%
      if (guess == target)
      {
            out.println(
          "Congratulations! you got it. Lets start a new game!");
```

```
           target = (int)(Math.random() * 100);
           session.setAttribute("target", "" + target);
           session.setAttribute("attempts", "1");
      }
      else
      {
           if (guess < target)
                out.println("Aim higher");
           else
                out.println("Aim lower");
      }
%>
<BR>
<FORM action="http:Guess.jsp" method="POST" >
Enter your guess:
<INPUT id="guessInput" type="text" name="guess"/>
<INPUT type="submit" value="Send"/>
</FORM>
<SCRIPT>
      document.all.guessInput.focus();
</SCRIPT>
```

If it is a new session (new game) we create a target (that the user is going to guess) and store it in the session along with an initial value for number of attempts. If this is not a new session (i.e., it is a continuing game), then we fetch the target and number of attempts from the session. We check to see if the user's guess is equal to the target. If so, we congratulate the user and reset the variables to start a new game. Otherwise, we instruct the user to guess again.

Servlets to JSP – From bad to worse

Servlets are Java classes that implement the javax.servlet.Servlet interface. They are compiled and deployed in the web server. The problem with servlets is that you embed HTML in Java code. If you want to modify the cosmetic look of the page or you want to modify the structure of the page, you have to change code. Generally speaking, this is left to the better hands (and brains) of a web page designer and not to a Java developer. What's nice about JSP is you are simply writing an HTML page, but then you can embed Java code into it. Looking at the above example, no web page designer in the right mind will volunteer to modify it. There is too much Java code in it. *This is a perfect example of a bad JSP page!*

Model 1 Architecture

Servlets are great for Java code. JSPs are great for HTML. Placing HTML into servlets or placing Java code in JSP leads to a system that is very hard to maintain. This brings us to the so called Model 1 architecture shown below:

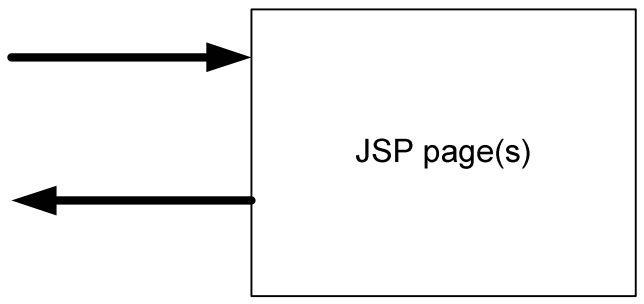


Figure 1. Model I Architecture

In Model 1 architecture, a series of JSP pages (or servlets) do all the work. Each page takes upon the complete task of fulfilling a request. There is no separation of concern. This leads to code that is very hard to maintain. It takes some effort to keep the different aspects of the code separated from each other.

MVC and Model II Architecture

Most of us are familiar with or have heard of Model-View-Controller architecture (or pattern). This fundamentally talks about separation of concern. You want your system to be layered. The Model deals with information and rules that are close to the information. Control deals with the behavior and business logic around the use of the model/information. View deals with the presentation logic. By separating these three, you get a system that is more resilient to change. It improves extensibility. In the Unified Software Development Process[2], the analysis models use three stereo types: Boundary Classes, Control Classes, Entity Classes. The Entity classes represent the model, the boundary classes represent the presentation layer that interacts with the actor. When it comes to web development (with Java), JSP is idea to hold the presentation logic (View). Servlets are ideal to hold the business logic (Control). Simple Java classes could provide access to Information (Model) and some control as well. This leads to the so called Model II Architecture as shown below.

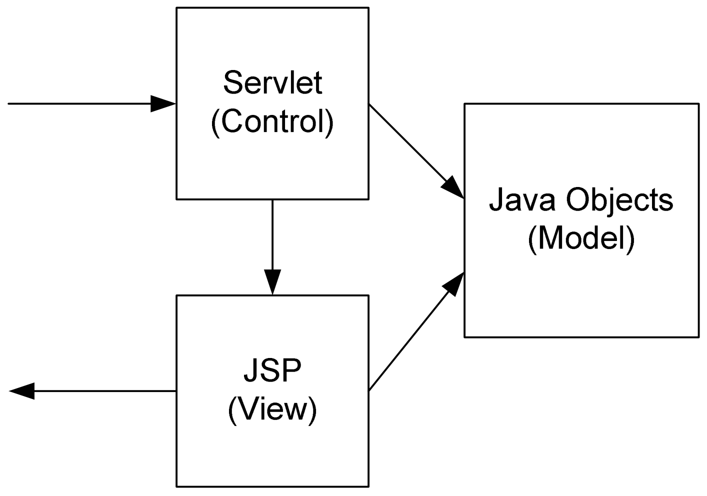


Figure 2. Model II Architecture

In this architecture, the request from the browser is received by a servlet. The servlet (which houses Java code) communicates with simple Java objects to access information and takes care of processing. Once the request has been processed, the flow of control is transferred to an appropriate JSP page. The JSP page, which will contain HTML for most part and some limited JSP tags will then display the information by accessing the simple Java objects and some session variables, if need be.

Guessing Game with MVC

Let's apply the above architecture to our simple Guessing game application. The servlets and JSP involved and the interaction between them is shown below:

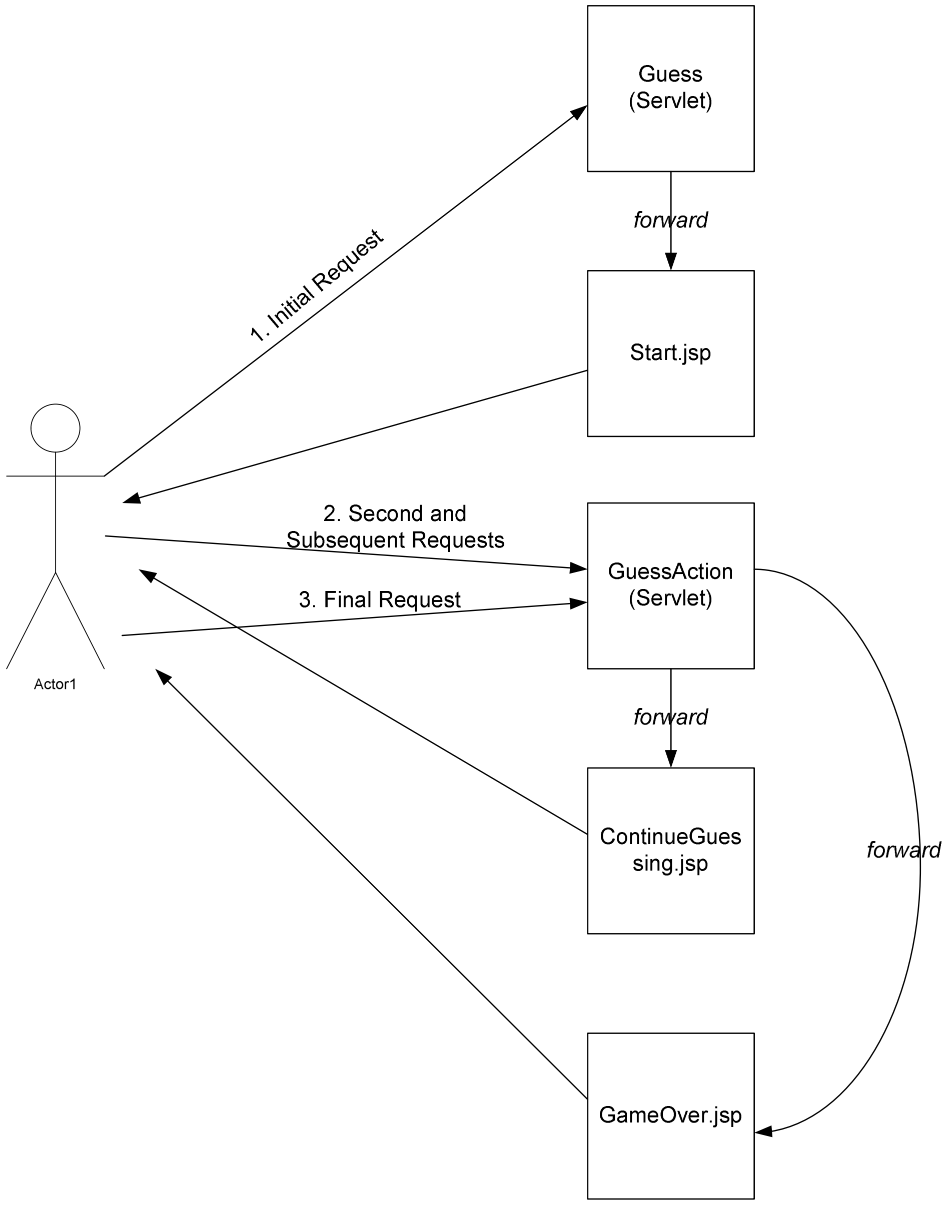
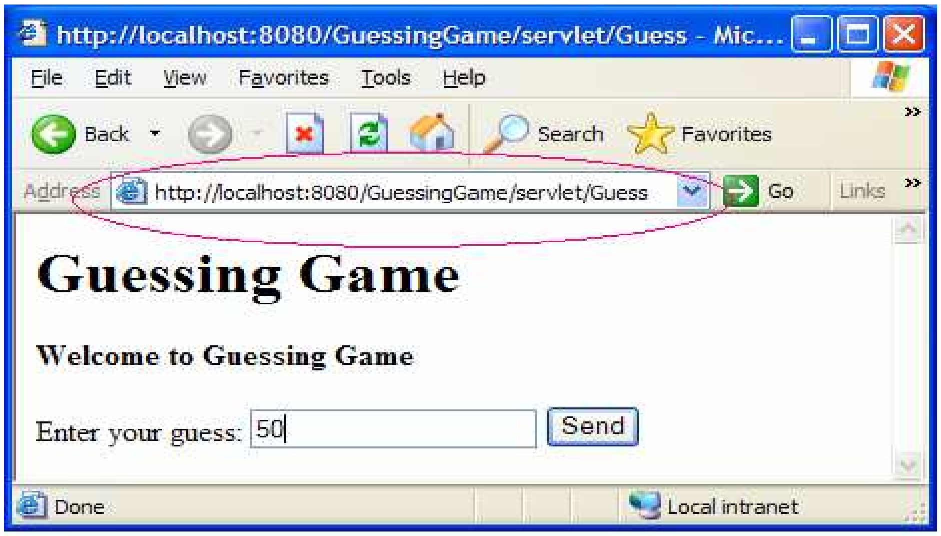


Figure 3. Guessing Game with MVC applied

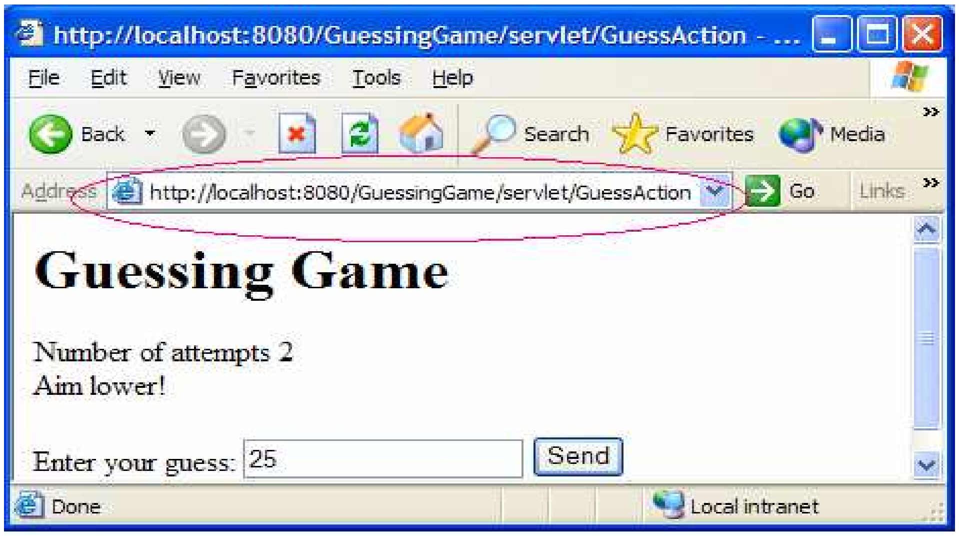
Before we start looking at the code, let us take a look at this application under execution:


http://localhost:8080/GuessingGame/servlet/Guess - Mic...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8080/GuessingGame/servlet/Guess
Go
Links
Done
Local intranet

Guessing Game

Welcome to Guessing Game

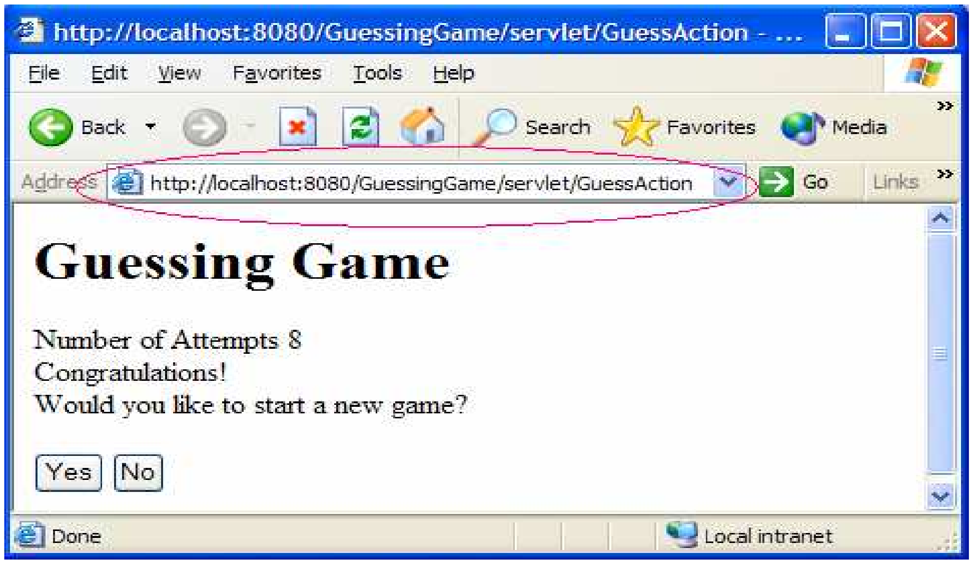
Enter your guess: 50 Send


http://localhost:8080/GuessingGame/servlet/GuessAction - ...
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:8080/GuessingGame/servlet/GuessAction
Go
Links
Done
Local intranet

Guessing Game

Number of attempts 2
Aim lower!

Enter your guess: 25 Send


http://localhost:8080/GuessingGame/servlet/GuessAction - ...
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:8080/GuessingGame/servlet/GuessAction
Go
Links
Done
Local intranet

Guessing Game

Number of Attempts 8
Congratulations!
Would you like to start a new game?

Yes No

This has got to be a better implementation because I was able to guess the number in fewer tries! I would like to point to some thing important to note. From in the address bar, *you can see that the visits are going to one of two servlets (Guess and GuessAction) and not to a JSP page.* Ideally, **a user should never have to visit a JSP page directly**. The request goes to the servlet directly. The servlet takes care of the actual processing of the request and then transfers the control to an appropriate JSP page. The JSP page then will display the response to the user. The subsequent request (after the user fills in the form and clicks the submit button) is directed to a servlet again for processing. **A user of the system should not be aware of the existence of any JSP pages at all**. Let's take a look at the actual code now.

We will first start with the "Guess" servlet.

```
// Guess.java servlet
import javax.servlet.*;
import javax.servlet.http.*;
import Guessing.Info;

public class Guess extends HttpServlet
{
      public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, java.io.IOException
      {
            HttpSession session = request.getSession();
            int target = (int)(Math.random() * 100);

            session.setAttribute("target", "" + target);
            session.setAttribute("info", new Info());

            request.getRequestDispatcher("/start.jsp").
                  forward(request, response);
      }
      public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
            throws ServletException, java.io.IOException
      {
            doGet(request, response);
      }
}
```

The servlet first initializes the target, sets session variables and transfers control to the start.jsp page. The class Guessing.Info is a simple bean that holds two properties attempts and message as shown below:

```
// Info.java bean
package Guessing;

public class Info
{
      private int attempts = 1;
```

```
        private String message = "";

        public int getAttempts() { return attempts; }
        public void setAttempts(int value) { attempts = value; }

        public String getMessage() { return message; }
        public void setMessage(String value) { message = value; }
}
```

The start.jsp (shown below) merely includes GuessRequest.htm.

```
<!-- start.jsp -->
<H1>Guessing Game</H1>

<B>Welcome to Guessing Game</B>

<%@ include file="GuessRequest.htm" %>
```

The GuessRequest.htm contains the form for the user to fill in as shown below:

```
<!-- GuessRequest.htm -->
<FORM action="http:GuessAction" method="POST">
Enter your guess:
<INPUT id="guessInput" type="text" name="guess"/>
<INPUT type="submit" value="Send"/>
</FORM>
<SCRIPT>document.all.guessInput.focus();</SCRIPT>
```

The reason for including GuessRequest.htm in the start.jsp page will become obvious soon as you find that the same form is needed in another jsp page as well.

Note that the form (in GuessRequest.htm) directs the user to the servlet GuessAction. The GuessAction servlet is shown below:

```
//GuessAction.java servlet
import javax.servlet.*;
import javax.servlet.http.*;
import Guessing.Info;

public class GuessAction extends HttpServlet
{
        public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                throws ServletException, java.io.IOException
        {
                int attempts = 1;
                int guess = -1;
                int target = 0;
                String forwardPage = "/ContinueGuessing.jsp";
                String message = "Aim higher";
```

```
            HttpSession session = request.getSession();
            target =
                  Integer.parseInt(
                        session.getAttribute("target").toString());

            Info info = (Info) session.getAttribute("info");
            attempts = info.getAttempts();
            attempts++;
            info.setAttempts(attempts);
            guess = Integer.parseInt(
                        request.getParameter("guess"));

            if (guess == target)
            {
                  forwardPage = "/GameOver.jsp";
                  message = "Congratulations!";
            }
            else
            {
                  if (guess < target)
                        message = "Aim higher!";
                  else
                        message = "Aim lower!";
            }

            info.setMessage(message);
            request.getRequestDispatcher(forwardPage).
                  forward(request, response);
      }

      public void doPost(HttpServletRequest request,
                        HttpServletResponse response)
            throws ServletException, java.io.IOException
      {
            doGet(request, response);
      }
}
```

The GuessAction servlet checks if the user's guess is correct. If the guess is lower (or higher) than the target, it sets a message (in the info bean which resides in the session) to "Aim higher" (or "Aim lower"). Then the control is transferred to the ContinueGuessing.jsp. If the user's guess is correct, then the control is transferred to the GameOver.jsp.

The two jsp pages are shown below:

```
<!-- ContinueGuessing.jsp -->
<H1>Guessing Game</H1>
<jsp:useBean id="info" scope="session" class="Guessing.Info" />
Number of attempts
<jsp:getProperty name="info" property="attempts"/><BR>
<jsp:getProperty name="info" property="message"/>
```

```
<%@ include file="GuessRequest.htm" %>

<!-- GameOver.jsp -->
<H1>Guessing Game</H1>
<jsp:useBean id="info" scope="session" class="Guessing.Info" />
Number of Attempts
<jsp:getProperty name="info" property="attempts"/><BR>
<jsp:getProperty name="info" property="message"/><BR>
Would you like to start a new game?

<FORM action="Guess" method="POST">
<INPUT type="submit" value="Yes"/>
<INPUT type="submit" value="No" onclick="window.close()"/>
</FORM>
```

From the JSP pages we can see that there is no java code in them. It uses regular HTML and the JSP tag libraries. Any web designer (by which I mean one who is not a Java programmer) will be able to maintain it. They can modify it, change the fonts, add pictures and change the esthetics of the page to their hearts content. Notice also that the servlets do not have any HTML in them. Java programmers (by which I mean one who is not interested or capable in the esthetics of the presentation) can modify the code, access any database, create as much or as little Java code to their hearts content as well. This has given a good separation of concern.

What’s the catch

If you are comfortable with the code organization shown above and the goal that is realized from it, you pretty much know the fundamentals behind a framework like Struts. While the above separation of concern has immense value, we could agree that it takes some effort and a lot of discipline to keep it that way. A framework will help us in several ways. It could guide us towards better organization of the code. It can remove the tedium of developing the code. Several steps could be automated for us so we can focus on delivering the core logic. In the next article, we will show how this can be implemented using Struts.

Conclusion

In this article we presented an example of a simple application which suffers from the mixing of HTML and Java code (presentation and logic). We then showed how we managed to achieve the separation of concerns and work towards an implementation that follows the MVC architecture. In the next issue we will see how this can be implemented more elegantly using Struts.

References

1. http://jakarta.apache.org/tomcat/index.html.
2. The Unified Software Development Process, Jacobson, Booch, Rumbaugh, Addison-Wesley.