Model-View-Controller Exercise

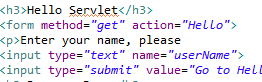
# Internet Software Development

In this exercise you will recreate a simple Hello Servlet and a **Model-View-Controller** application. MVC is the recommended best practice for Web applications in which there is some business logic that determines what page the user should see in a particular situation.

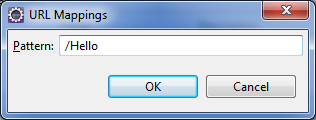
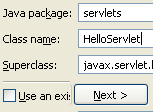
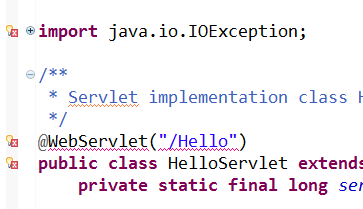
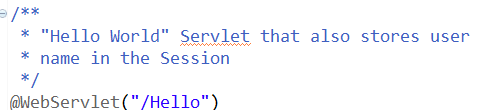
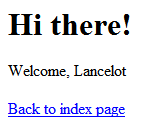
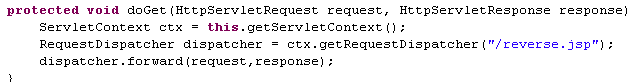
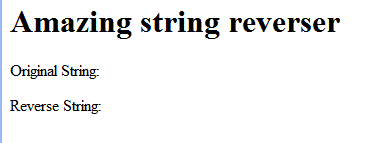
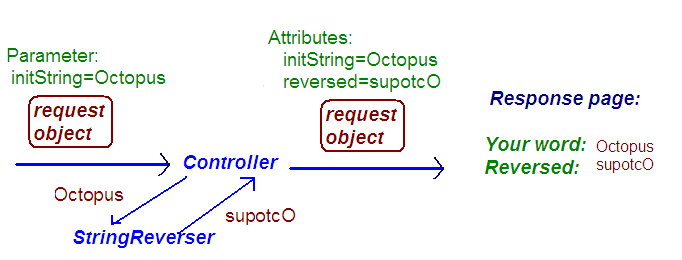
I have noticed that Eclipse frequently gets “stuck” and won’t update the project with the change in your JSP files. When this happens, try **Project/Clean**. If that fails, you might have to quit and restart Eclipse. This seems to be a known issue among Eclipse developers

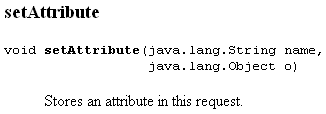
1. Fire up Eclipse and create a Dynamic Web project named **MVC Practice**. If you have installed Tomcat 7 , you may use Web module version 3.0

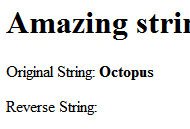
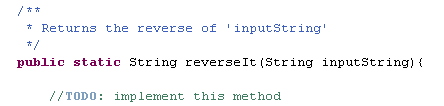
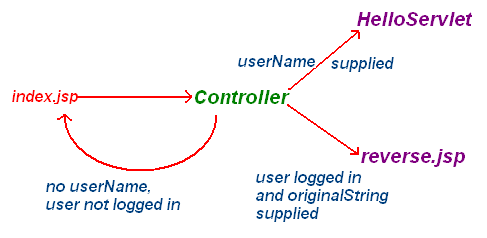
This example will use the Model-View-Controller design pattern: the application reverses an input string, but in order to use this functionality the user must first “log in” by entering their name into a form. (A true login would compare the user name and password to values stored in a database.) Once the user has “logged in,” their user name is stored in the session and they are allowed to use the string-reversing service.

1. Make an **index.jsp** page with two forms, similar to the one shown here. The **action** attribute of the top form should be “Hello”, and the action attribute of the bottom form should be “Controller”. Use the “get” method for both forms – this will allow us to see the request parameters. We normally use POST method for form submits, but you may use the “get” method whenever the operation requested is ***idempotent***, meaning that result of performing the operation twice is the same as the result of performing the operation once.

In situations where a form operation is not idempotent, another standard practice is to have the HTTP Get method handler return the input form (an operation which is idempotent) and the HTTP Post method handler actually submit the data.

1. Create a new Servlet named **HelloServlet** in package **servlets**. When you have entered the package name and the class name, press the button and set the URL pattern to “**/Hello**.” (You’ll use the Edit button to do this.)
2. Yikes! When I created this Servlet on Eclipse, I got a slew of error messages. Checking the errors, it appears that Eclipse cannot resolve many of our standard servlet classes. To fix this problem, use  to add the runtime library tor the Tomcat Web server.
3. A good first step for any class is the class comment, which describes the purpose of the entire class. Based on the JSP page, we can describe the Hello servlet’s function like this. If we later on change the functionality of this servlet, we should go back and update this comment.
4. Here is the starter code for the **doGet**() method of HelloServlet. Test it out by running the app, entering a name in the user name box, and clicking the “Go to Hello Servlet” button. In theory, you should see the cheery “Hello World!” message.
5. In the Project Explorer pane, find . Open this node and find the servlet mapping for HelloServlet. The notation here is a bit tricky: the deployment descriptor lists the URL as /Hello, with a front slash; this is the URL you will use when forwarding a request. But in a form, you should use , without the front slash. Go figure.
6. Now edit the **doGet** code in HelloServlet to print out the user name entered in the input form. You will use the **request.getParameter** method. Of course you must give the input element a “name” attribute in the form in **index.jsp**. Also add a link back to the index page, by printing out the <a href=”index.jsp”> anchor element. To print out quotes inside of quotes, you escape them with backslashes.
7. Once this is working, create a new servlet named **servlets.ControllerServlet** and give it the URL pattern **/Controller**. This servlet will not emit HTML; instead it will forward the request to a JSP page. The starter code is at left This is how a controller forwards a request to a selected resource (in this case, reverse.jsp).
8. Compose the **reverse.jsp** page as shown at right. (We don’t have the actual String content yet). Test that running the app and clicking the “See Reversed String” button sends you to this page.
9. The picture on the left shows our plan for filling in the “Original String” and “Reverse String” blanks. Instead of doing the heavy lifting itself, our Controller will use a ***helper object*** that actually reverses the input String. Our helper object will be named, appropriately enough**, helpers.StringReverser.**
10. In order to pass data objects to the JSP page for display, we will attach them to the request object as **attributes**. A request attribute is very much like a parameter: it is data attached to the request, using the name-value pattern. Unlike request parameters, attributes are not part of the HTTP protocol specification, but are extras added to the request object in Java. Another difference is that parameters are Strings, but request attributes may be of any Object type.

Look up the **HttpServletRequest** API and find the **setAttribute** method. (You will find this method in the methods inherited from the ServletRequest class.) This method allows us to attach objects to the request object, which can then be retrieved by the resource to which we forward the request. We’ll use this technique extensively in MVC –patterned applications.

1. Add some code to the doGet method in **ControllerServlet,** before forwarding the request to reverse.jsp:
   1. Define a String variable to hold the initial String submitted by the user. Please choose a representative name! You can define this variable and give it a value in a single assignment statement; the value will be obtained by a call to request.getParameter(), where the argument is the name attribute of the HTML input element in which the user enters the original string.
   2. Use the **request.setAttribute** method, loading the original input string as an attribute named “**originalString**.” Do this before you forward the request.
2. In reverse.jsp, add a Java Expression Scriptlet to extract the attribute named “originalString,” as shown here. Run the application and enter a String into the “Reverso Page” form. When you click the Submit button, you should now see the original String.
3. Now we will introduce **Expression Language**, which is an easier way to extract request attribute (and allows us to get rid of some of the Java code in the JSP page.) Test this by entering a string into the **Reverso** input form in index.jsp – you should be able to see the original String.
4. Now we need to actually reverse the String. Instead of doing this in the servlet, create a new class **helper.StringReverser**. Give the class a single method, **reverseIt**, as shown here. Since the StringReverser class has no data members, we can make this a static method. That way, it is not necessary to instantiate a StringReverser object to use the reverseIt method. Also, since the class has no data members, it does not need a constructor.
5. Add the JUnit 4 Library to your project. Then generate JUnit test class **TestStringReverser**. (I used the New/Other wizard and choose the base class with a filter text of “Test Case.” ) Write a single test method that tests that StringReverser should return “olleH” as the reverse of “Hello”, “supotcO” as the reverse of “Octopus”, “654321” as the reverse of “123456”, and “” as the reverse of “”. Remember to call the **reverseIt** method in a static way (without instantiating an object), as shown here.
6. To implement **reverseIt**, look up the StringBuilder class. This class has all the tools you need – you just have to read the API! Please remember to remove the //TODO comment shown above when you have finished implementing the reverseIt method.
7. Once you have tested StringReverser into existence, go back to **ControllerServlet**. In the **doGet**() method, use **StringReverser** to reverse the original string. You can do this in a single Java statement, which defines a new String variable named **reversedString**, and gives this variable a value via a call to the StringReverser.reverseIt method. For the input to reverseIt, use the variable you’ve already defined that holds the original String submitted by the user.
8. Load the reversed String into the request as an attribute named **reversed**. Edit reverse.jsp to display the **reversed** attribute in the same way you display the **originalString** attribute.
9. Finally, we need to change the app so that the user is forced to enter their name before they can go to the Amazing String Reverser. The forwarding logic is shown at right, with the three cases. Here’s how to do this:
   1. Edit index.jsp so that both forms submit to “/Controller”.
   2. In ControllerServlet.doGet(), compose an if statement to check for the existence of a “username” request parameter. (Hint: check to see if request.getParameter(userName) is null. If you find it, forward to “/Hello” using the RequestDispatcher method. Test this out by running the app and typing in a name.
   3. Here is the code logic for the controller **doGet** method. (This code assumes that you used “inputString” as the parameter name for the string to be reversed; please adapt this for your own parameter name.) For debugging, we print out the stored user Name and the current user input parameter. Unfortunately unit testing of servlets is harder than testing plain Java objects, so for the present we must debug by means of console printouts.