Waukesha County Technical College
152-198 Distributed Java

Class 6 Plan and Assignments

Discussion Activities:

- **Due Today:**
  1. Review your JSTL Experiments Assignment.
  2. Continued study of JSTL and EL started in Class 5.

- **Q&A**

- **Continue discussion of JSTL and EL**
  - ✓ See lecture notes in Class 5 Plan.

  - Using "<%=" as a shortcut for out.println() in a Java Scriptlet vs using an EL expression:

    You can use this technique anywhere in a JSP, even inside HTML, if the expression evaluates to a string. Here's an example:

    Before (using plain Java)

```
<%
   out.println(msg)
%>
```

    After using Java Scriptlet Shortcut

```
<%= msg %>
or…
<p>The message is: <%=msg%></p>
```

    After using EL

```
${msg}
or…
<p>The message is: ${msg}</p>
```

  - ✓ You cannot mix EL or JSTL with plain Java or anywhere between <% and %>. This is why JSTL is necessary for things like branching and looping logic.

  - ✓ The easiest way to find the proper syntax for JSTL is to look it up by searching Google for something like "JSTL for loop" or "JSTL if logic".

- ✓ Don't forget the "taglib" declarations at the top of JSP pages if you use JSTL. Not necessary for EL alone.

- **Using HTML Forms on the client and IF logic in the Controller to make decisions about control flow**

  A web form should have a name attribute, an id attribute (must be unique) a method attribute set to "POST" and an action attribute set to the url-pattern of the controller you are sending your input to. So, for example, if I'm sending form input to "MainController", my form tag may look like this:

```
<form id="calc" name="calc" method="POST" action="CalcConroller">
    <input type="text" id="firstname" name="firstname" value=""/>
    <input type="submit" id="submit" name="submit" value="Submit Me"/>
</form>
```

  Notice the action value above has no QueryString parameters. You can add them by appending the value with ?key1=value1&key2=value2, etc.

  Also notice the name attribute of the input box. This is the parameter key; it's value is the parameter value. This is not a "QueryString parameter" but it is a parameter. As far as the controller is concerned, they are the same – both are parameters. A good use of a QueryString parameter in an action value of a form tage is to inform the controller which IF statement to execute. So modifying the above sample to use a QueryString parameter looks like this:

```
<form id="rectangleForm" name=" rectangleForm" method="POST"
action="CalcConroller?calcType=rectangle">
    <input type="text" id="length" name="length" value=""/>
    <input type="text" id="width" name="width" value=""/>
    <input type="submit" id="submit" name="submit" value="Get Area"/>
</form>
```

  For example, if I append "?calcType=rectangle" this tells the controller to execute the rectangle calculation vs, say, a circle calculation.

  Your IF logic may look like this in the controller:

```
String calcType = request.getParameter("calcType");

if(calcType.equals("rectangle") {
     // do rectangle work
} else if(calcType.equals("circle") {
    // do circle work, etc.
}
```

  Here's another controller tip: remember, the job of a controller is to control data flow between Model and View, not to process the data. All request parameters are retrieved by the Controller as Strings. Do not convert these into integers, doubles, etc. Rather, pass them as Strings to your model and do the data conversion there.

  When coding a controller, never use any properties that aren't global to all users

accessing the web site. Servlet properties are shared and you must be careful that multiple request threads don't access shared properties at the same time! Best practice is to only use public static final properties to avoid magic numbers in your code.

Lab Activities:

- Create a Maven Project named "CalculatorLabs". Use different pages for 3 different labs, named lab1.jsp, lab2.jsp and lab3.jsp. Provide separate controllers and model objects as necessary:

1. **Calculator Lab #1** –Using what you've learned, create an MVC-style web application to calculate the area of a rectangle. You should have a separate input form for the length and width values, a controller for data flow, and a model object for performing the calculation. The calculation result should go to a separate result page. Choose the names of your controller and model object carefully. Remember, follow all best practices learned in Advanced Java, including proper naming and removal of magic numbers. Try to enhance your input form to use an image representing a rectangle to help the user understand the type of input needed for this calculator. You may draw your own or better yet borrow an image off the internet. Note: borrowing an image for educational use only is legal. Finally, be creative. How can you improve the view pages to make this input and output more user friendly.

2. **Calculator Lab #2** – Create a version of Lab #1 that uses the input form for both input and as a place to output the results. There is no separate result page in this lab. How will you do this? Remember, the output comes from the controller and you won't have any output the first time you go to your form. The solution is to check your request attributes for null and assign blanks or default values if they are null. We'll discuss why you might or might now want to send output back to the same page as input.

3. **Calculator Lab #3** – Use the rectangle calculator from previous labs and then Add calculators for calculating the area of a circle, and another for calculating the hypotenuse side of a triangle when the other 2 sides are known. This is a classic Math problem using the Pythagorean Theorem which states that $a^2 + b^2 = c^2$. You may use this equation to calculate the length of the hypotenuse by following the tips in this video: https://youtu.be/uthjpYKD7Ng . You can borrow some geographic images (circle, rectangle, triangle) from the web. Put all three input forms on one page.

   What will you have to do to support separate forms on one page? Well for one thing you will need to give each form a unique name and id. Very important! Also, send all of the results back to the same input page, so that a user can conveniently see the answers near the input.

Finally, consider the logic in your controller. How will you handle input from different calculators?

What you will need to do is use IF logic to test where the input came from. And you will need a way to tell the controller which calculator input form you are using. This can be done in various ways: using a QueryString parameter for the form's action value, or by using a hidden form field. Use a special flag, e.g., "calcType = Rectangle" in the hidden form field or QueryString parameter.

In the IF logic you will test for that value, which will indicate the type of calculation to perform. Remember, form values and QueryString values are all treated as "parameters" that you can retrieve in the controller. What you send back is one or more "attributes" – this is where your answers and other data are stored and provided to the view page.

Another option, of course, would be to use separate controllers for each calculator. Then you wouldn't need the IF logic. But consider how this could easily grow into hundreds of controllers over time if you keep adding new calculators. We'll discuss the pros and cons of this strategy.

Textbook Chapters (and other resources) covered:

- Java EE v1.7 tutorial: http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
- Java SE API (v1.8): http://docs.oracle.com/javase/8/docs/api/
- Java EE API (v1.7): http://docs.oracle.com/javaee/7/api/
- Printed Servlet Tutorial: "java-servlet-tutoria.pdf" provided on Blackboard. Note: this uses Eclipse as an IDE. We will not be using eclipse. So the techniques for doing stuff in the tutorial may vary with those of Netbeans.
- Online tutorials for client-side: http://w2schools.com
- Netbeans web development tutorials: https://netbeans.org/kb/trails/java-ee.html
- Netbeans Git User Guide: http://netbeans.org/kb/docs/ide/git.html
  (don't use SSH – we'll be using the modern HTTPS approach)

Preparation Work for Next Class

1. Complete any unfinished lab work. Use EL expressions where possible.