LS/EECS Building Ecommerce Applications

Lab 4: StudentCalc R4.0

(have this ready for Test1)

Objectives

-understand model view control programming model
-learn to program **Listeners**, **JavaScript** and **Ajax**

Before You Start

- Export your `studentCalc3` project (make sure you include the source files in the export) to a war file **`studentCalc3.war`** on the desktop.
- Import the war file just created to a new project named ***studentCalc4***.
- Review the Lecture notes and the Samples on Listeners and JavaScript

Requirement 1. Add real time analytics to your application

Use case: your application should monitor the "Principal" parameter across all requests and all sessions; calculate the maximum principal across all principal values and keep this value in memory; when a user queries your application with this url ***http://host:port/studentCalc4/admin***, respond with a the maximum principal.

How to do it:

-You can do this without modifying (or even recompiling) your exiting *controller*, *model*, or *view*. Simply add a *listener* that subscribes to changes in the *Session* Scope (or any scope where you store the form parameters). Whenever an attribute is *added* or *changed*, the listener gets notified and it can recompute the maximum principal and keep it somewhere.

- In order to add *listeners* to your site, follow these steps:
    1. Create a new package (under src) and name it `listener`.
    2. Right click the package and click "New" to add a new listener.
    3. Name your listener class `MaxPrincipal` and register it with the *Session scope*.
        - In the wizard, you are asked to specify the *events* the listener is "listening" to: choose *add* or *change* attribute in the session scope
    4. Supply the needed logic to keep track of the largest principle entry.

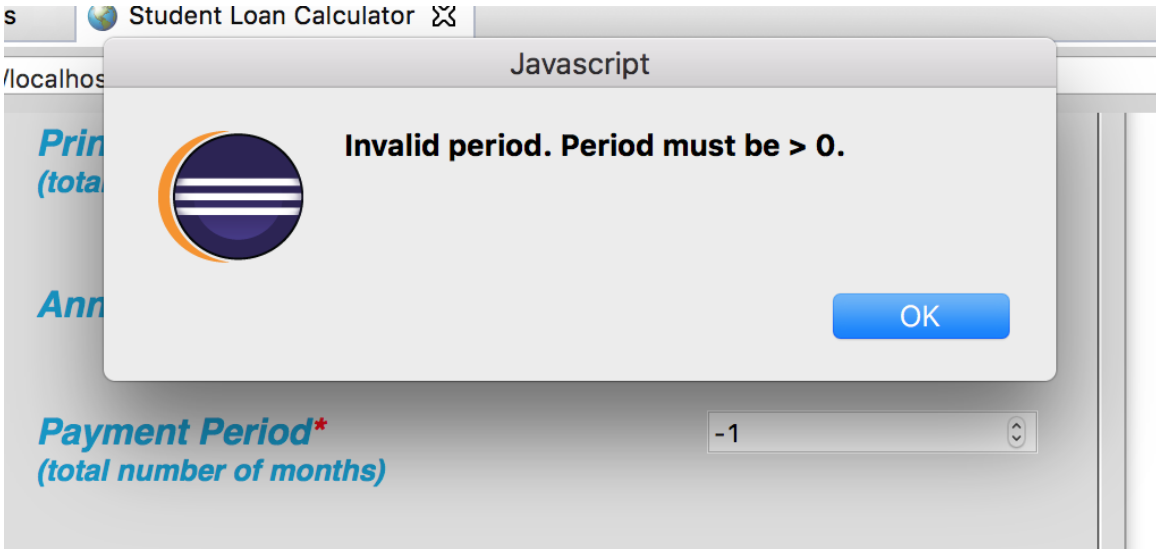
- Note that the Listener receives notifications when any attribute at the session scope changes or is added; you need to select and act only the one related to the "principal" attribute.
- To keep track of the maximum "principal" you probably need to store it in an attribute, using setAttribute

5. Add servlet `Analytics.java` that is mapped to "admin" URL
    - It receives the GET method, *http://host:port/studentCalc4/admin*
    - It gets the attribute you probably saved in the previous step (that holds the maximum principal)
    - Forwards the maximum to the view (see next step).
6. Add a view named `MaxPrincipal.jspx` to display the computed analytics.

Requirement 2. Handle entry errors at client side

Use case: Handle any input errors at the client side. To avoid loading the server with unnecessary computations, if the user enters an input which is not valid you have to detect it in the browser and report it back to the user. For example, if the user enters a negative principal, you should inform the user that the principal should be greater than 0. For improved usability, show an * error in red next to the input that contains errors.

You should still check the errors on the server side (for the case when the client disabled JavaScript).



How to do it:

-Performing certain MVC tasks on the client makes your webapp more responsive and lightens the load on the server. In our webapp, we will do validation on the client using JavaScript. Note that the server will continue to validate (through the model's exception mechanism) in case the request did not come from the form, but the common case will be fast.

- In order to link your html to your JavaScript, add the following to the `head` section of your UI.jspx:

```
<head>
…
   <script type="text/javascript" src="res/mc.js">;</script>
….
</head>
```

- In order to invoke a JavaScript function when the form is submitted, we add the following attribute to the `form` tag in UI:

```
<form… onsubmit="return validate();"….>
```

  When the submit button is pressed, the browser will invoke the `validate` function in your JavaScript file. If the function returned `true`, the browser will form the query string and submit the form as usual. Otherwise, the submission is cancelled, i.e. as if the button was not clicked.

- Right-click the `res` sub-folder of WebContent and select a new JavaScript file and name it *mc.js*.
- Create the `validate` function along these lines (note that you can retrieve an element by id, with *document.getElementById)*:

```
function validate(){
  var ok = true;
  var p = document.getElementById("principal").value;
  if (isNaN(p) || p <= 0){
     alert("Principal invalid!");
     ok = false;
   }
  if (!ok) return false;
  p = document.getElementById("interest").value;
  if (isNaN(p) || p <= 0 || p >= 100){
     alert("Interest invalid. Must be in [0,100].");
     ok = false;
  }
  return ok;
}
```

- In addition to popping an alert, you can embed an error message next to the erroneous input. To that end, in the form, add an html paragraph with

an ID after those inputs and then populate them from JavaScript (using `innerHTML` when an error is detected).

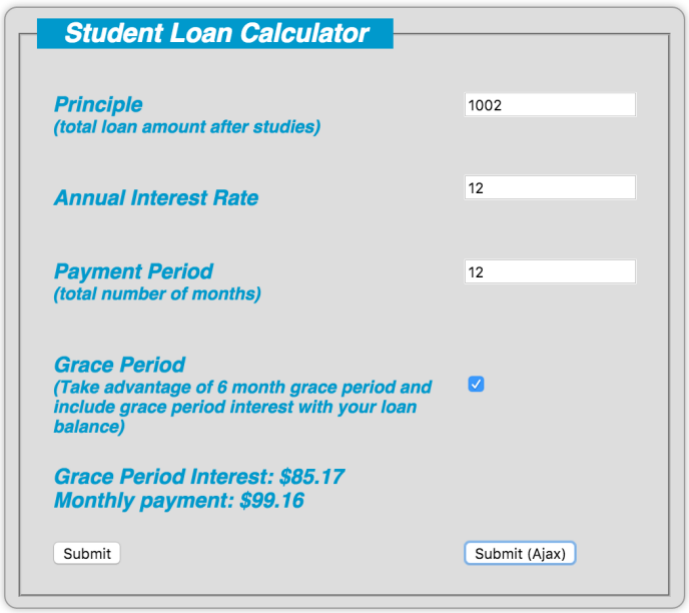
-Explore using other events such as `onload` in the body tag or `onkeypress`, `onselect`, and `onfocus` in an input tag. Note that these intrinsic events, along with many others, can be attached to any html element.

-explore using other Web JavaScript APIs, such as document.querySelector(), document.createElement()

Requirement 3. Add asynchronous calls to your application.

Below you see the caption of how your UI should look like:

1. Add a button that invokes doSimpleAjax() function.
2. Add doSimpleAjax function to your Java Script file
3. In doSimpleAjax function, build the data string from the form inputs
4. Change doGet() method of the Start.java servlet to handle the request and write back the http response.
5. Note below that the Grace Period Interest and Monthly Payments are displayed within the same page when Submit (Ajax) is clicked
6. When Submit button is clicked, the application behaves as in the previous lab, displays a new page.



How to implement Requirement 3

What is asynchronous communication? (for reading, check also the lecture and the samples that come with the lecture)

So far, we used two methods to invoke a servlet: (s) an url in the browser to invoke GET method and pass some data in the query, (b) a form and a submit button to invoke POST with some data. Both of them will get back an html page in the http response and the browser will render the html. Those 2 operations are quite visible and can be inefficient and inelegant in many scenarios. There is another way, a quicker way, to communicate with the server in an asynchronous manner. The asynchronous communication can be implemented with two JavaScript APIs: XMLHttpRequest (Or Ajax) and Fetch.

What is Ajax:

-Asynchronous JavaScript and XML

-a mechanism by which the Java Script Engine (Java Script is standardized by ECMA) can communicate with a server directly, thus bypassing the Layout Engine of the browser.

Is it different from form submission?

Answer: yes in 4 ways:

1. the http request needs to be built manually, particularly form data
2. the http request is not blocking (browser not waiting for response)
3. the http response needs not be html
4. the http response does not automatically trigger page re-rendering

How to do it at Client End?

The client uses **XMLHttpRequest()** to communicate with the server. In the simplest case, define a JS function in your Java Script file that uses the above class:

```
function doSimpleAjax(address){

      var request = new XMLHttpRequest();
      var data='';
      /* add your code here to grab all parameters from form*/
```

```
      request.open("GET", (address + "?" + data), true);
      request.onreadystatechange = function() {
        handler(request);
      };
      request.send();

 }
```

Pass to it the URL of the server (with http | absolute | relative), the data to be sent (as a properly encoded query string), and the function to be called when the response is ready. Typical function:

```
function handler(request){
  if ((request.readyState == 4) && (request.status == 200)){
     var target = document.getElementById("ajaxTarget");
     target.innerHTML = request.responseText;
   }
}
```

More things about **XMLHttpRequest() object:**
Has three properties:
-onreadystatechange: stores the function to be invoked when the server sends the http response
-readyState: has values from 0 to 4 with 4 meaning that the response is received and ready to be processed
-status: when status=200, there was no error on the server side.
-you can invoke the servlet synchronously or asynchronously by using the 3rd parameter of the open() method(*true* means asynchronously).

When is client side Ajax function invoked?

Like any Java Script function, it is invoked when an event is triggered. For example, if a button is clicked, you might want to invoke an Ajax function instead of submitting the whole form and force a rendering. For example:

```
<button name="ajax" value="true"
 onclick="doSimpleAjax('/studentCalc4/Startup/Ajax/');return false;">
 Submit (Ajax)
</button>
```

How is it done at the server end?

First, in the doGet (or doPost) method, you need to distinguish that the request comes from Ajax. You can use the URL path for that, as in the example above. Then, treat it like any other request: the controller handles it; consults the model; updates one of the three persistence scopes. If the response object is returned from servlet, then make sure you use flush() method of the writer.

To get ready for the Test 1, export your project as a war file, with the name YourStudentLastName_YourStudentNumber_Test1.war. Example: Lee_2109800_Test1.war. You will use this project for Test 1. Make sure you have the jstl.jar in the lib folder and your application is working.