Course : Diploma in Multimedia & Infocomm Technology (EGDF15)

Module : Java Enterprise Development (EG3752)

|  |  |
| --- | --- |
| Laboratory : | Lab 2 – Validation, Modularisation and Servlet Chaining |
| Objectives : | This lab will go in depth on building a Java Web Application that takes data validation and modularisation into consideration.  At the end of this session, you should know how to:   * Choose an appropriate location in the code to implement data validation and constraints. * Implement data validations and constraints through HTML, JavaScript and Java. * Incorporate an open source library into your project. * Validate email addresses using the Commons library. * Use Servlet Chaining technique to move from one servlet to another. |
| Software Used : | Java Standard Development Toolkit (JDK™) 8.0  NetBeans IDE 8.0 with GlassFish Server 4.0 bundle |

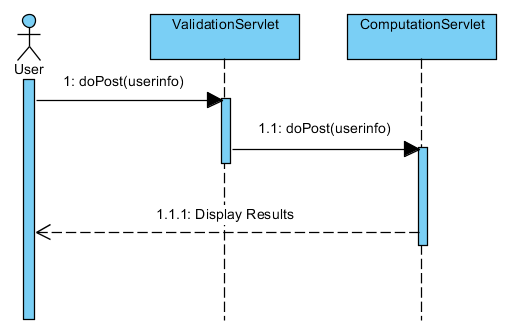
**Enhancing the Registration Page**

You may have noticed that the simple registration that we have developed in Lab 1 has several problems. For instance, the application is not ready to handle erroneous user input.

In this lab, we will be extending from the previous lab to handle common erroneous user input. Additional requirement specifications are listed in the table below:

|  |  |
| --- | --- |
| **Requirement Reference** | **Requirement Specifications** |
| 2.1 Validation and Constraints | The system shall ensure that the users provide all required information correctly.  In particular, the system will need to make special validation for the following fields:   * **Admin Number** must carry a six-digit string followed by an alphabet. * **Email Address** may be validated in accordance to the implementation by Sandeep V. Tamhankar, implemented in the Apache Commons library. * **Height** must be keyed in as a whole number. * **Weight** must be keyed in to the nearest one decimal place. |

Below is a sequence diagram showing how different components of the web application will be interacting with each other:



Validations and Constraints

Data needs to be validated or constrained in order for applications to run smoothly. User can input incorrect data (eg. keying in a word in a field asking for the height of the person just because the field says so visually.) Validations and constraints can help to ensure input data are correct and meaningful to the system, so that it can process without glitches.

What to constrain and validate depends on the nature of the data we expect the user to fill in and how the input fields were implemented in the GUI. Refer to the solution you have put together in Lab 1 and answer the following questions:

* Which of the following input field found on the GUI should be validated/constrained?
* Why do you need to validate the data?

Take a moment to think through, experiment with your Lab 1 solution and discuss with your friend to complete this exercise. Use the following table to guide your thinking process. Two examples are also given for your reference.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Field** | **Control Used** | **Validate/ Constrain?** | **Why?** |
| Name | Textbox | Yes | To ensure user does not leave blank |
| Admin No | Textbox | Yes | To ensure that user gives a valid admin number |
| Email Address | Textbox |  |  |
| Gender | Radio Buttons |  |  |
| Specialisation | Dropdown box |  |  |
| Height (in cm) | Textbox |  |  |
| Weight (in kg) | Textbox |  |  |

In EG3752, we will be learning three places where validation and constrains can be done, namely, HTML (within HTML or JSPs), JavaScript and Servlets.

|  |
| --- |
| **An Important Note about Validations and Constraints**  Choosing the type of validations and constraints required for a field is a design process. For instance, it may actually be alright in one application for the users to leave the name field blank, such as in an anonymous survey, while it may be absolutely necessary for them to fill up their name in other instances, such as registering an account with a delivery system.  There is NO “one correct answer” to do validate and constrain data. It depends largely on the application you intend to build. |

Validating and Constraining Data using HTML

1.1 Open the **index.html** of the **SimpleRegistration** project.

1.2 Add the **required** attribute to all “textbox”-typed input fields. An example using the Admin number field is shown below:

<input type=**"text"** name=**"adminno"** required/>

* 1. For the radio buttons, you can do this in one of the two ways:

<input type=**"radio"** name=**"gender"** value=**"Male"** required/> **Male** <br/>

<input type=**"radio"** name=**"gender"** value=**"Female"**/> **Female** <br/>

<input type=**"radio"** name=**"gender"** value=**"Male"** checked/> **Male** <br/>

<input type=**"radio"** name=**"gender"** value=**"Female"**/> **Female** <br/>

Experiment with the two different methods and find out what the difference between the two methods are.

1.4 For fields requiring a number, you can choose to change the type attribute to **number** as such:

<input type=**"number"** name=**"height"** required/>

1.5 To allow fields to accept decimal places, add a step attribute to a number-typed input field as such:

*Example: To accept up till 1 decimal places*

<input type=**"number"** name=**"weight"** step=**"0.1"** required/>

1.6 Test your programme to ensure that all fields have been validated as it is.

Validating Data using JavaScript

While HTML already provides a robust set of functionalities to constrain the sort of data that can be input into the field, more complicated data validation which requires more coding can be made using JavaScript. For the purpose of this lab, we will be validating the Admin Number field using JavaScript

* 1. Add a JavaScript file to the project:

a. Add a folder named **js** under the **Web Pages** node under the **Project Explorer**.

b. Add a JavaScript file named **validation.js** under the **js** folder

Your project structure will look something like this:

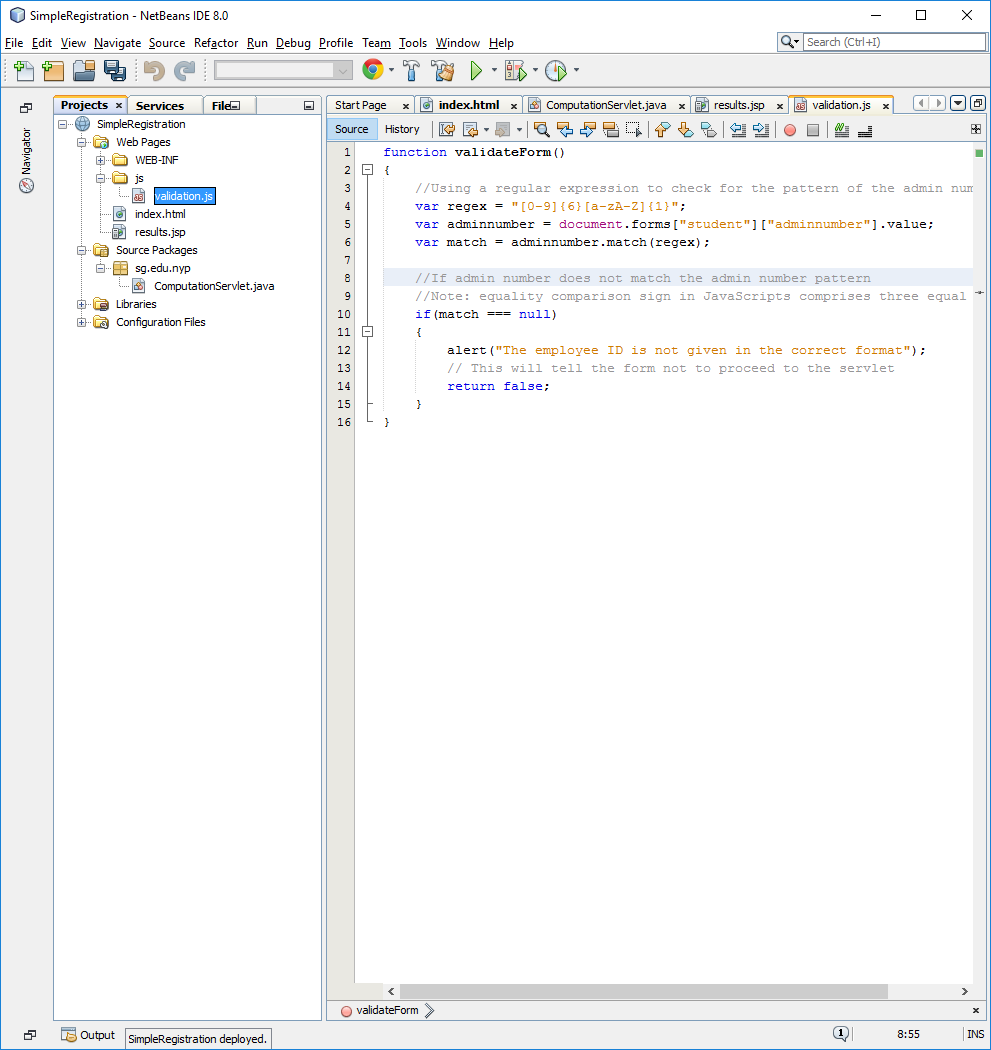


Figure 2.1: Structure of the project after adding the JavaScript file

2.2 Create an empty **validateForm** function in the JavaScript file as such:

***function*** validateForm**()**  
**{**  
**}**

2.3 To let the index.html use the created JavaScript file, typed in the following line under within the **head** tags of the **index.html**:

<script src=**"js/validation.js"** type=**"text/javascript"**></script>

2.4 To make the form use the **validateForm** functionbefore proceeding to the servlet, do the following:

a. Add **student** as the **name** attribute to the form. The name of the form will be needed in the JavaScript in order to obtain the Admin Number variable from the form.

b. Add an **onsubmit** attribute to the form. The value of the attribute will be “**return validateForm()**”.

The code for the form element in the index.html will look like this:

<form name=**"student"** action=**"compute"** method=**"get"** onsubmit=**"return validateForm()"**>

2.5 Copy **Code Snippet Lab 2 P 2.5** and paste it in the **validateForm** function in **validate.js.** Read the comments to understand the flow of logic of the JavaScript before proceeding.

To match the admin number, a technique known an regular expression matching is used. This technique looks for a pattern within the given string and can tell the application whether the input data is written in the pattern it is to match or not.

2.6 Test your application once again to ensure that the regular expression validation is working as intended.

Top of Form

Validating Data using Servlets and Modularising Functionalities

For even more complicated validation, servlets can also be used for validation. For the purpose of this lab, the email address will be validated using the Servlet. In addition, an open source library will be used to validate the email address.

Instead of coding everything into the **ComputationServlet**, a separate **ValidationServlet** will be created for this purpose. This allows two different programmers to programme the application concurrently (possibly one on working with the **ValidationServlet** and the other on the **ComputationServlet**) more effectively. This process of separating different functions into different files / modules is known as **modularisation**.

3.1 Create a **ValidationServlet**. Change the form **action** attribute on your **index.html** to **validate**. Remember to override the **doGet** method in your **ValidationServlet**. Refer to Lab 1 if you are unable to remember how to create a servlet and link it to the **index.html**.

*Using an Open Source Library for Data Validation*

One of the advantages of using Java is the number of open source libraries that is available for usage. This means that developers who need to use the same function do not need to think about how to code a function; there is no need to reinvent the wheel.

In this exercise, we learn to use a function from the **Commons.Validator** library made available by Apache to validate email addresses.

3.2 Download the latest **Commons.Validator** library from <http://commons.apache.org/proper/commons-validator/download_validator.cgi> .

You may realise that there are four options which are given to you. Read the following carefully and download one the library most applicable to your case:

**Binaries**: Compiled Java codes, ready to be imported and used in your project. They are usually compiled as a **.jar** file.

**Source**: Source codes of the library you are download. Unless you are intending to make changes to the functions of the library, a developer who intends to use the functions as it is do not need to work with the source codes.

**.zip files**: These files are used for Windows environment.

**.tar.gz files**: These filesare used for Linux environment.

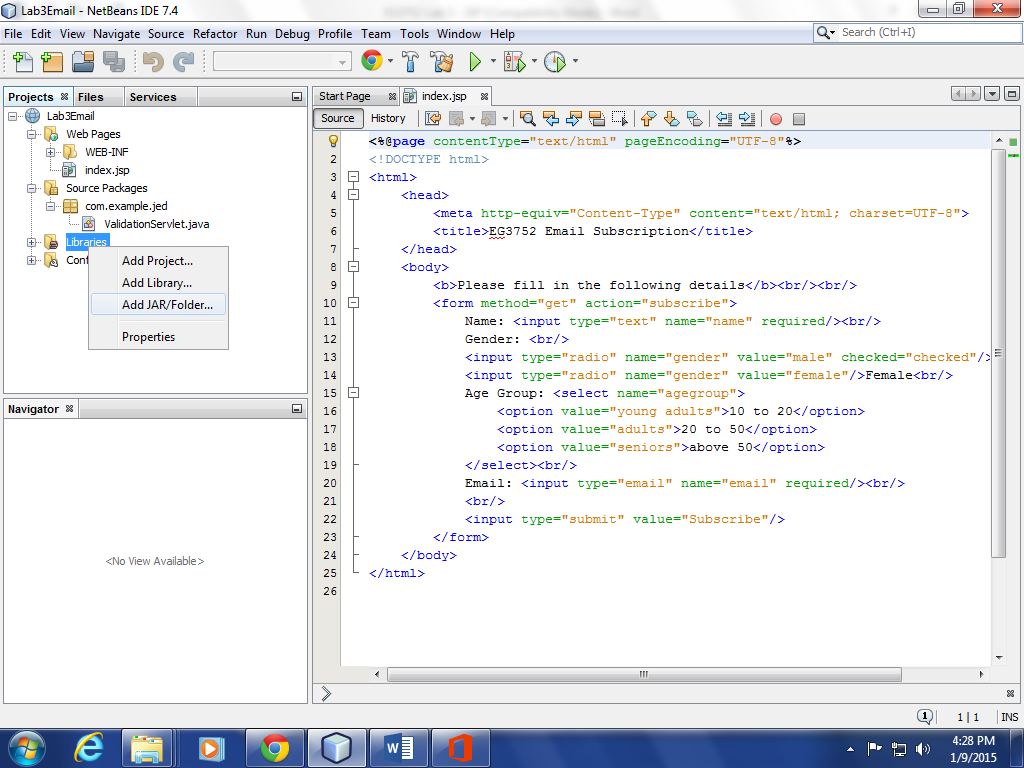
Your choice is: \_\_\_\_\_\_\_\_\_\_

1. Binary .tar.gz file b. Binary .zip file
2. Source .tar.gz file d. Source .zip file

3.3 Browse to your project folder using the **Windows Explorer**.

3.4 Create a **lib** folder in your project.

3.5 Extract the library in the **lib** folder.

3.6 Go back to the NetBeans IDE and import the library into your project. To do that, follow these steps:

1. Under the **Projects** tab, right click on the **Libraries** node found under your project node (See Figure 5).
2. Choose **Add JAR/Folder**.
3. Browse to the location where your library has been extracted and select the library (eg. commons-validator-1.5.1.jar).
4. Ensure that **Relative Path** is selected from the **Reference as** options.

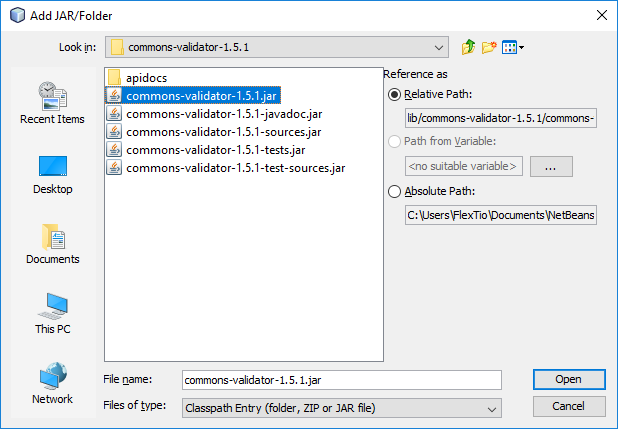


Figure 3.6: Configurations of the Add JAR/Folder dialog box

1. Click **Open**.

It is a common standard to place all external libraries required in a **lib** folder within the project.By using relative path, projects can also be moved to different PCs without the need to change environment variables or configurations required by the project.

3.7 To make the NetBeans IDE show the documentation of the functions the library is using, you can add the documentation that comes with downloaded package to the library in this manner:

1. Expand the **Libraries** node under your project if it has not been done.
2. Right click on the library that you have just imported into your project.
3. Select **Edit…** .
4. Click the **Browse** button found to the right of the **Javadoc** text input.
5. Browse to the folder where your library is extracted and select the Java documentation jar file (eg. commons-validator-1.5.1-javadoc.jar).
6. Ensure that **Relative Path** is selected fomr the **Reference as** options.
7. Click **Open**.
8. Click **OK**.

3.8 To validate the email address, the **EmailValidator** class found under the **org.apache.commons.validator.routines** should be used. The following set of sample codes gives an example of how the EmailValidator class can be used:

//Validate the email address using Apache Commons library

//Step 1: Declare a EmailValidator object

EmailValidator emailValidator;

//Step 2: Initialise the EmailValidator Object by retrieve

//an instance of the object

emailValidator = EmailValidator.getInstance();

//Step 3: Use the function isValid to check for the //correctness of the email address. The variable ‘email’

//is the email address that is retrieved from the request

if(emailValidator.isValid(email))

{

//Email is correct. Go to ComputationServlet

}

else

{

//Email is not correct. Go back to the index page

}

More details on the usage of the EmailValidator can be found via <http://commons.apache.org/proper/commons-validator/apidocs/index.html> .

|  |
| --- |
| **Additional notes about the Apache Commons Library, EmailValidator version 1.5.1 class**  Based on a script by [Sandeep V. Tamhankar](mailto:stamhankar@hotmail.com) http://javascript.internet.com  This implementation is not guaranteed to catch all possible errors in an email address.  *Source: http://commons.apache.org/proper/commons-validator/apidocs/index.html* |

Forwarding, Redirecting and Servlet Chaining

In the previous lab, the flow moves from a servlet (ComputationServlet) into a JSP (results.jsp). We can also control the flow such that it moves from a servlet to a servlet. This technique is known as **Servlet Chaining**.

4.1 To chain the ValidationServlet and ComputationServlet together, the **forward()** method of the **RequestDispatcher** object can be used. Implement the following code snippet to make a forward() to the next resource:

RequestDispatcher rd = request.getRequestDispatcher("/compute");

rd.forward(request, response);

Note that the **forward()** method is also known as the **server-side redirect**. More details on the actual usage of forward() will be covered later.

4.2 To go back to the index page, the **sendRedirect()** method available of the **HttpServletResponse** object can be used. Implement the following code snipped to make a redirect back to the index page.

response.sendRedirect(this.getServletContext().getContextPath());

Note that the **sendRedirect()** method is also known as the **client-side redirect**. More details on the actual usage of sendRedirect() will be covered later.