EECS 4413. LAB 01: HTML+CSS+JS, Java

A. IMPORTANT REMINDERS

- Lab1 is due on **Thursday (May 25) at 9pm**. No late submission will be accepted.
- For this lab, you are welcome to attend the lab sessions on this and next Wednesday (May 17,24). TAs or instructor will be available t. help you. The location is LAS1002. Attendance is optional.
- Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too.
- You can submit your lab work any time before the specified deadline.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

- Download this lab description and the associated files and read it completely. Unzip the compressed file. If uncompressing is successful, you should get a folder **4413Lab01**, which contains html files, CSS files, JavaScript files, and an image folder with several images. Also contained is java folder "java" with several Java files,

- Download the JS and Java program from eClass (week 2), review the code and play with it. This is the program demonstrated in class. You should fully understand the code of the program before you start the lab.

- You should have a good understanding of
  - Html basic tags, `div, span`, form control (`textbox, dropdown list, submit button` etc)
  - Basic css
  - Events (such as `onclick, ondblclick, onchange`)
  - `document.getElementById().innerHTML`
  - `document.getElementById().src`
  - `document.getElementById().ischecked`
  - `document.getElementById().style.xxx` (`xxx` corresponds to all CSS styles, see css slides)

C. GOALS/OUTCOMES FOR LAB

- To learn/recap the basic html tags, html form controls
- To learn/recap the basic CSS styles
- To learn how to change the *behaviour* of an HTML document using JavaScript, how to use JavaScript to validate form inputs (before sending to the server)
- Experience dynamic web page with a remote server (running php file), including database access
- Experience Java multi-thread programming and Object serialization.

D. TASKS

Part1: simple JavaScript to change image source and displaying text.

Part2: html forms to connect to a remote server. Using JavaScript to do form validation.

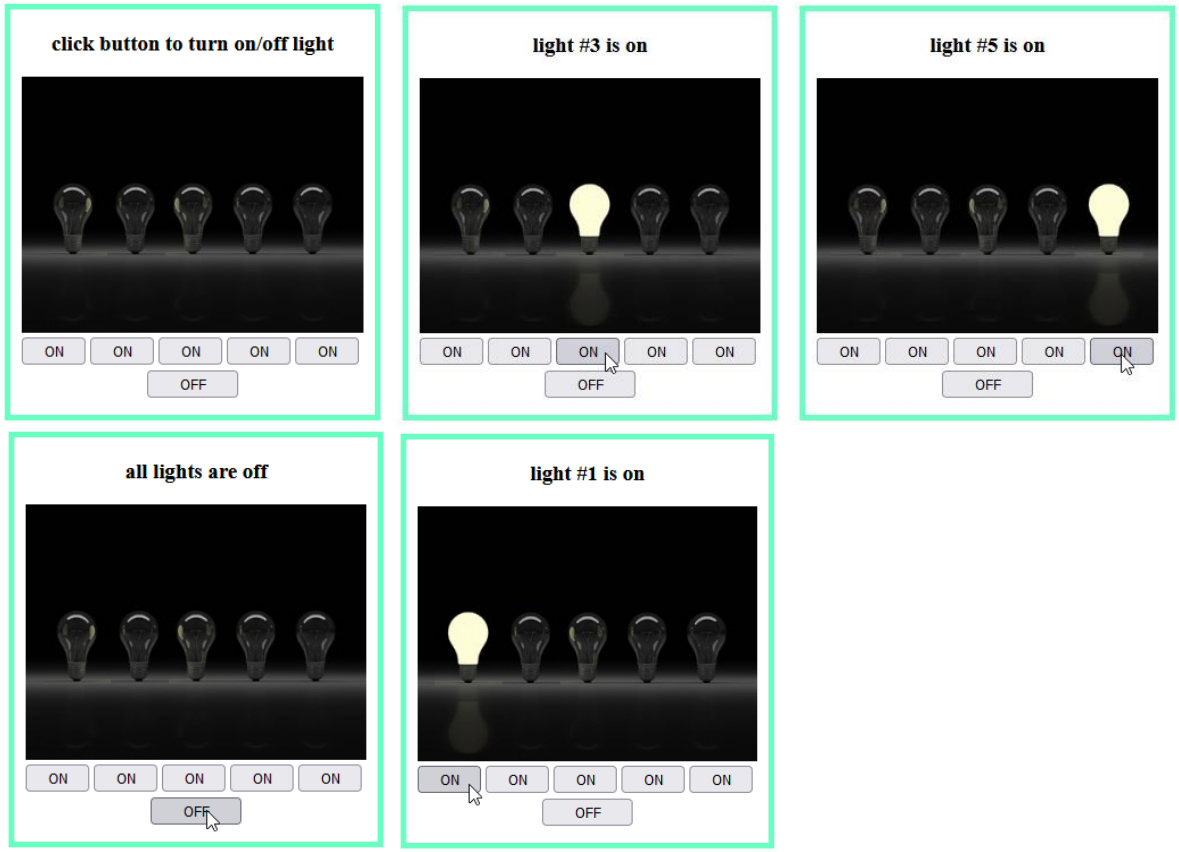
Part3: Simple Java Threads and Object serialization.

E. SUBMISSIONS

- eClass submission. More information can be found at the end of this document.

Part I. Simple JS to change presentation.

In this exercise you practice using simple JavaScript to change the display of message and image source.
This task involves nine files: **light.html**, **light.css**, **light.js**, and six images: **light_0-5.jpg** in **images** folder. The image **light_0.jpg** shows 5 light bulbs all turned off. The images **light_1.jpg** through **light_5.jpg** corresponds to a light being turned on (e.g. **light_2.jpg** is an image of the 2nd light turned on).
Your task it to modify **light.html** to include 5 buttons as shown below. Each time a button is selected, a JS function should be called, which changes the displayed image to be the corresponding one that shows the light above the button is "turned on". The displayed message is also changed to "*light #X is on*".



- Modify your HTML code to allow your JavaScript code to change the image and message.
- Add in the five buttons using the `<button> </button>` tags.
- There is a CSS defined for these buttons, but you'll need to modify the width of the five buttons so to fit in one row and below each of the five bulbs. Make it as close as possible. The images width is 300 pixels.
    - also in CSS, add a rule to align the text to the center of the image.
    - also in CSS, give the whole page a solid border of thickness 5px, and color of r 106, g 255 and b 194
- Following the steps such that when a button is clicked, the corresponding light is on, and corresponding message is displayed.
    - Add an onclick event handler for the left most button so that when this button is clicked, a JS function called `lightOn()` is invoked and executed (you can give other function name if you like, as long as the function with such name is defined in JavaScript).
    - Now implement the function. Open **light.js** where you are already provided with the skeleton of function `lightOn(){…}` .The body of the function is what we write in the pair of curly braces. We want to change the displayed image, by updating the content of the `<img>` element, who has an id "img". Specifically, you want to set the `src` attribute of the DOM object to a proper image.

- Once this left-most button works as expected, consider how to add event to other buttons, including the off button.
  - One approach is to define a separate functions for each button and then implement the functions by setting the image src with proper images and updating the displayed message with proper texts. E.g., button-1 is attached with function lightOn_1(), button-2 is attached with function lightOn_2() etc. A better approach, **which you should do here**, is to define a single JS function, attach it to all the buttons, and pass a parameter. That is, clicking any button will call the same function, but with different input (as parameter to the function). Then the function changes the images according to the input (using bunch of if else statement, or switch case statements).

Part II. HTML forms, form validation using JS, Client Server communication, database.

In this exercise you are going to implement a simple HTML form using various form controls mentioned in class, including textbox, radio button, checkbox, dropdown list, text areas, buttons. You also practice using external CSS to define the styles, and, using external JS to validate forms as well as change presentations.

Basic form controls

- Open the file `FormsInput.html` with an editor such as VS Code, and complete it following the comment instructions in it. Note that Program, Year and Hobbies have default selections.

An important tag for form element is <input type="??" > where ?? can be "text", "radio", "checkbox", "password", "reset", "submit", … Other tags used for form element include <textarea> <selection> <options>.

After adding the components, open the file using your browser. Your form should look like

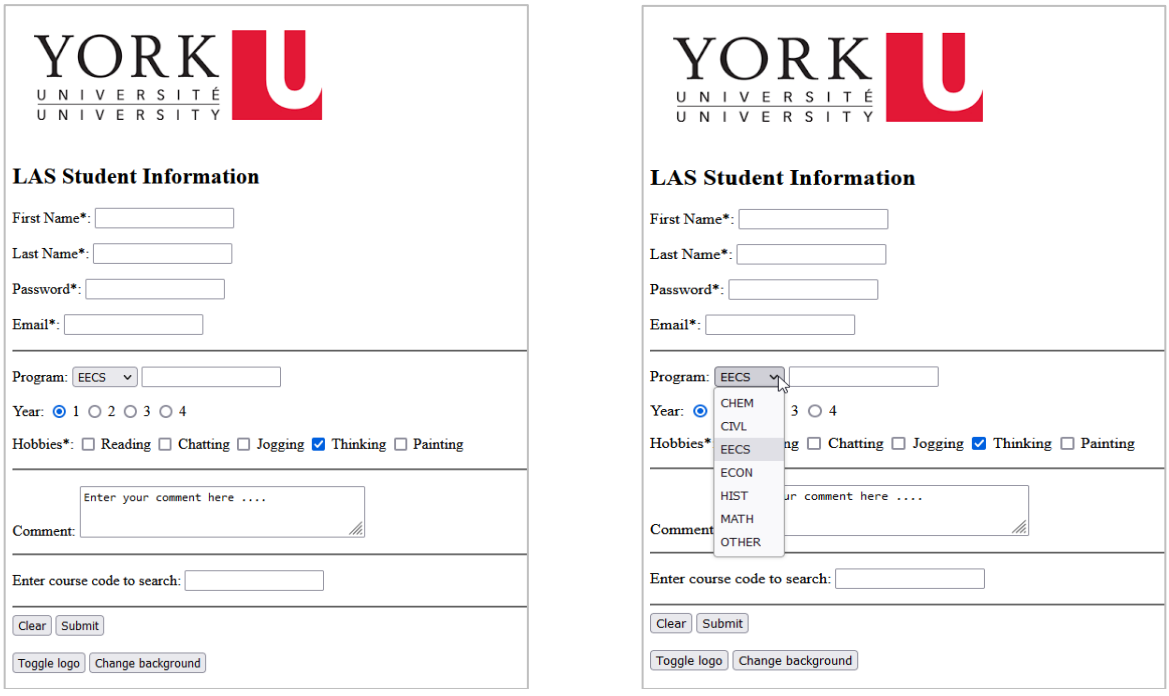


Figure 1 Form with no CSS styles

- Now, link the file to the provided CSS file by removing the comment in head part of the html and complete the link. Study how the CSS is defined. Save the html file and refresh your browser. You should get the following (depending on your browser/version, the rendering may be slightly different). Here we use Firefox, which is the recommended browser for this course.

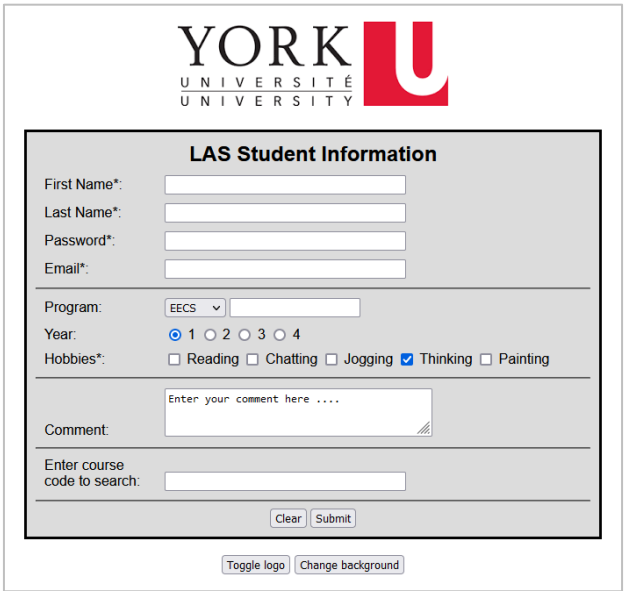


Figure 2 Form with provided CSS styles

- Next, add a rule to the css file, so that the textbox for the Program field (beside the dropdown list) is hidden initially (set *display* style of the element to "*none*" or, *visibility* to "*hidden*"). This textbox will be visible when the "OTHERE" in the program dropdown list is selected (elaborated below).
- Next, add an onclick event to the Toggle logo button, so that when it is clicked, a JS function is called, which change the logo file from "York.png" to "LAS.png" or vice visa. These two images are in the Images folder. Next, implement this function in the JS code so that it switches between the two images "York.png" "LAS.png".
- Next, add an onclick event to the "Change background" button, so that when clicked, it will change to a lightblue background. Next, implement this function in the JS code so that it works correctly.

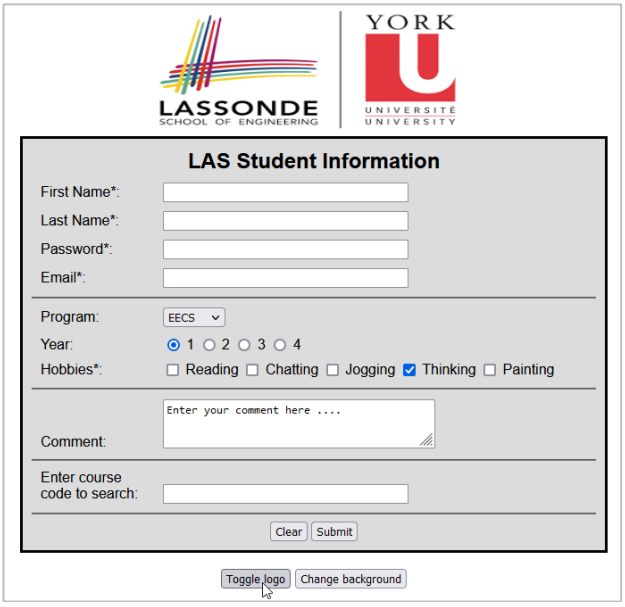


Figure 3 Change logo image

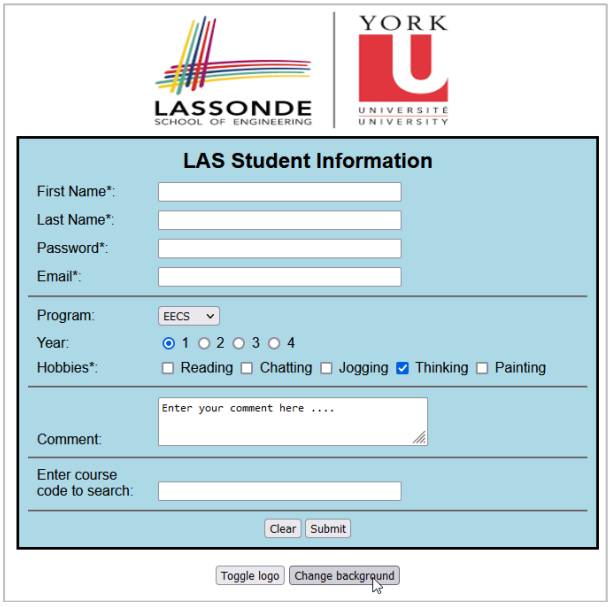


Figure 4 Change background color

- Next, implement the functionality that when "OTHER" in the Program dropdown list is selected, a textbox beside it is displayed, allowing user to enter the other program. In this case the mouse cursor should be set inside the textbox. Then if other program is selected, the textbox should disappear. Hint: add an *onchange* event to the dropdown list and pass the selected value to the function. Implement that function in JS, setting the *display* to "inline" or "none" according to the selected value (or *visibility* to "visible" or "hidden", depending on your css style for this). For putting the mouse cursor in the textbox, consider focus() on the element.

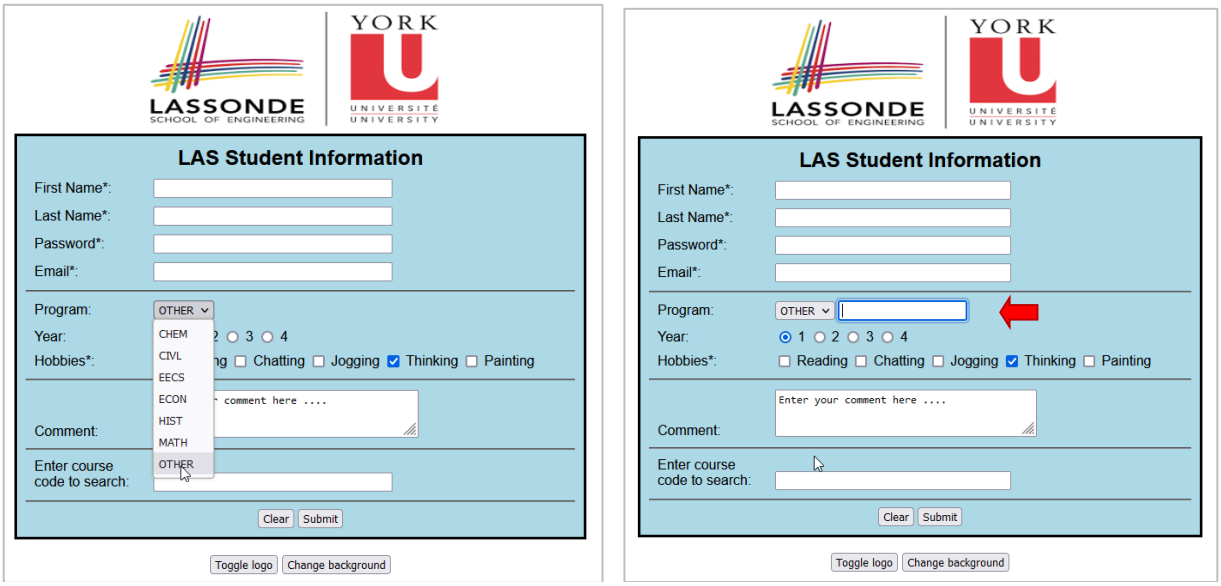


Figure 5 Selecting OTHER make a textbox displayed

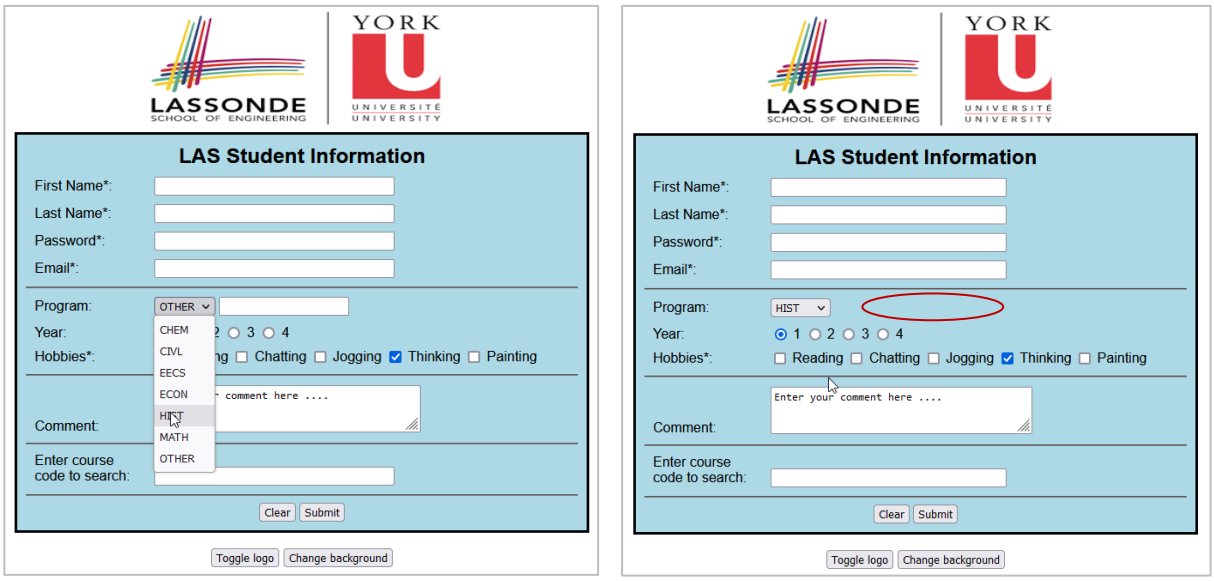


Figure 6 Selecting a program makes the textbox disappear

- Now enter some data, and test if the Clear button clears all the entered data, resuming the default selections.
- Next, enter some data again and click 'Submit' button. If implemented correctly, you should get the feedback from a file on the EECS server. An example is shown in Figure 7. The URL of this file is specified in the <form> part of your html. This gives you some idea of client-server model – the form page is a client and your browser sends a http request to the server file. The server file receives the data and then based on the data, displays something for you. Also, if a course code is entered, the server side will search the course code in the database and display the course details. In this example, the entered course code EECS4413 generated some data from the database on the server side.
  - Note that in this example, some data fields are left blank deliberately. The email is not a valid email address. Currently the form is still sent to the server. Preferably, we need to give some messages if the required data (annotated with *, all except Comment and Search course code) is missing. We can let the server validate and give feedback, but this will generate extra burden on the network and server, and will be slower. A better approach, as mentioned in class, is to use client-side JavaScript code to do validations (before the data is sent to the server). We will do this next.

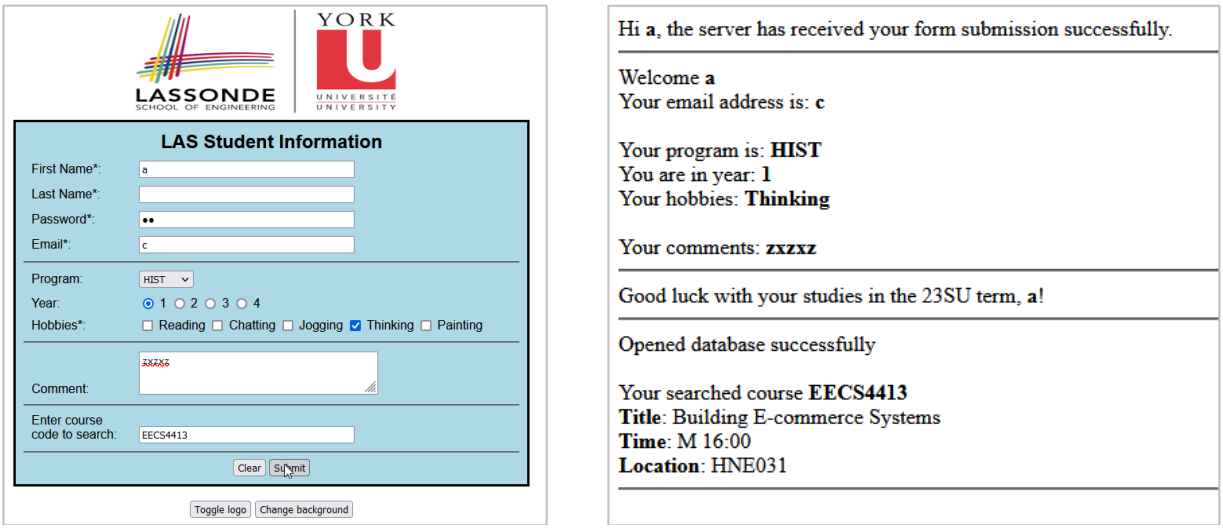


Figure 7 some input data and server feedback

Form Validation using client-side JS.

First, on the form, attach a JS validation function (where and how?), so that when the user trying to submit the form, the validation function is invoked. Then in JS, implement this function as follows.

- First, check if the First Name data is missing, if it is, generate a pop-up a window saying "First name should be filled \out", and, after the user clicks OK, the cursor should be in the First Name text box.

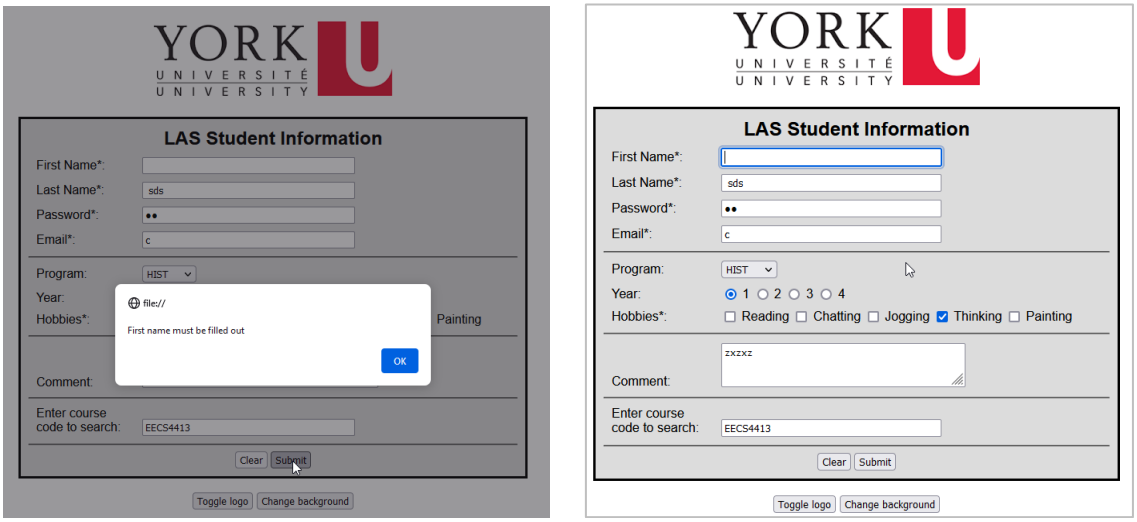


Figure 8 Validate First name field

- Next, check if the Last Name data is missing, if it is, generate a pop-up window with message "Last name should be filled out", and after the users click OK, the cursor should be in the Last Name text box.
- Next, check if the Password field is missing, if it is, generate a pop-up window with message "Password should be filled out", and after the users click OK, the cursor should be in the Password text box.
- Next, check if the Email field is missing, if it is, generate a pop-up window with message "Email should be filled out", and after the users click OK, the cursor should be in the Email text box.
- If email filed is entered, further check if the email is a valid email format, such as abc@domain.com or abc@domain.us That is, start with some characters for personal part, followed by @, and then domain info, then ends with 2 or 3 characters (e.g. com, ca, edu). More specifically, personal part contains
  - Uppercase (A-Z)and the lowercase (a-z) letter.
  - All the (0-9) Numeric characters.

- Special characters like:! - / = ? # $ % & ' * + ^ _ ` { | } ~,
- Period, dot, or full stop (.)

The domain part contains

- letters
- digits
- hyphens
- dots

Hint, this is a bit complicated part, a common approach is to use regular expression to match. If you are not very familiar with regular expressions, you may want to search on the web for some solutions (not all work correctly).

If email data is not valid, generate a pop-up window with message “Email format invalid”, after the users click OK, the cursor should be in the Email textbox. Some examples of invalid emails are shown below.

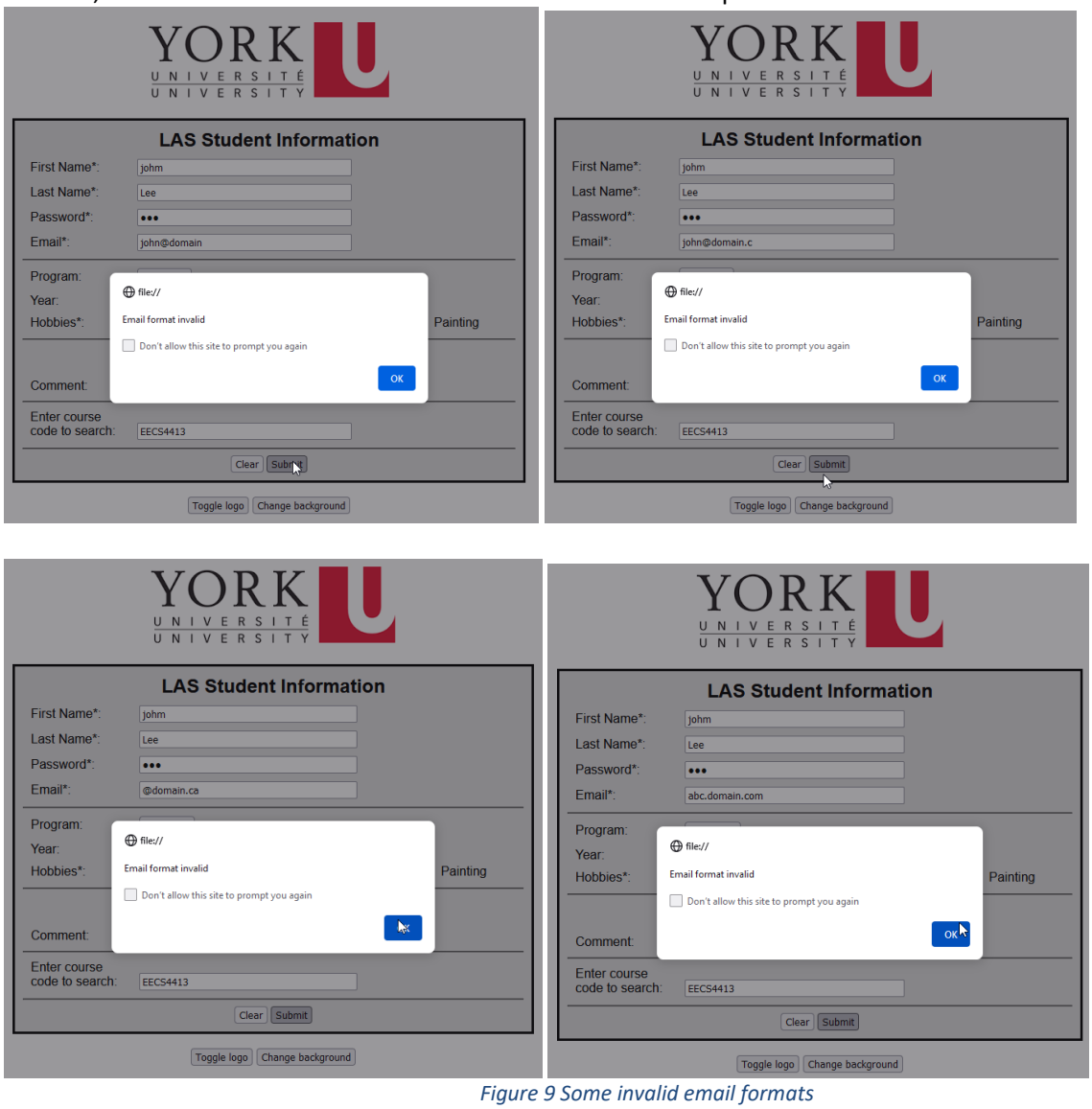


Figure 9 Some invalid email formats

- Next, check if at least one hobby is selected. If not, generate a pop-up window with message “at least one hobby should be selected”, as shown below.

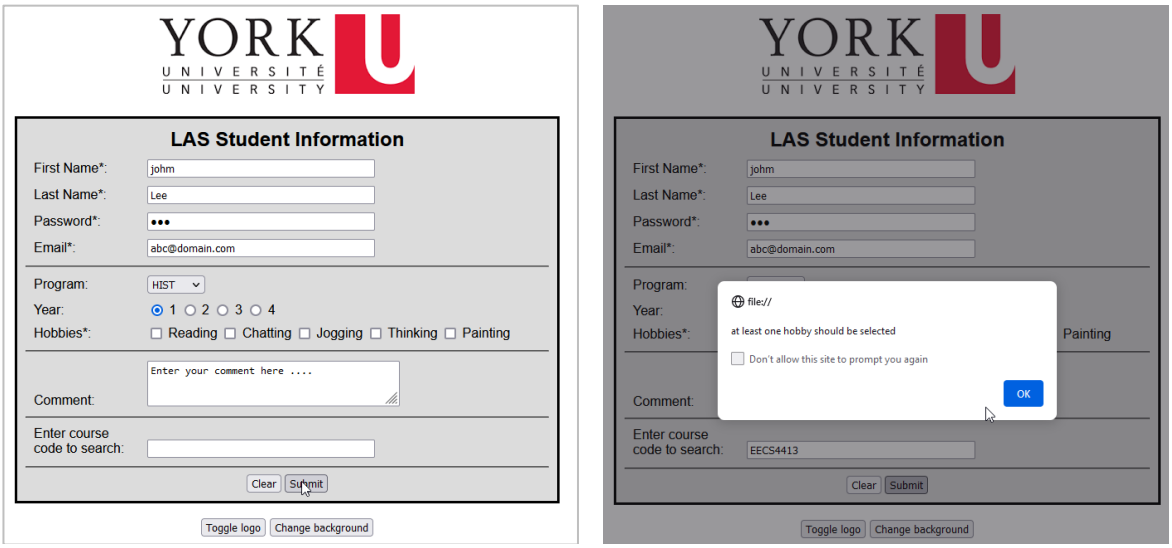


Figure 10 Check if at least one hobby is selected

- Next, check if the search course code is entered. This is an optional field, so if no code is entered, it is okay, but if a course code is entered, check if the format is EECSxxxx where xxxx are digits. If not, generate a pop-up window as shown below. Examples of invalid course code are EECS412, eecs4413, 4413, COSC3213.

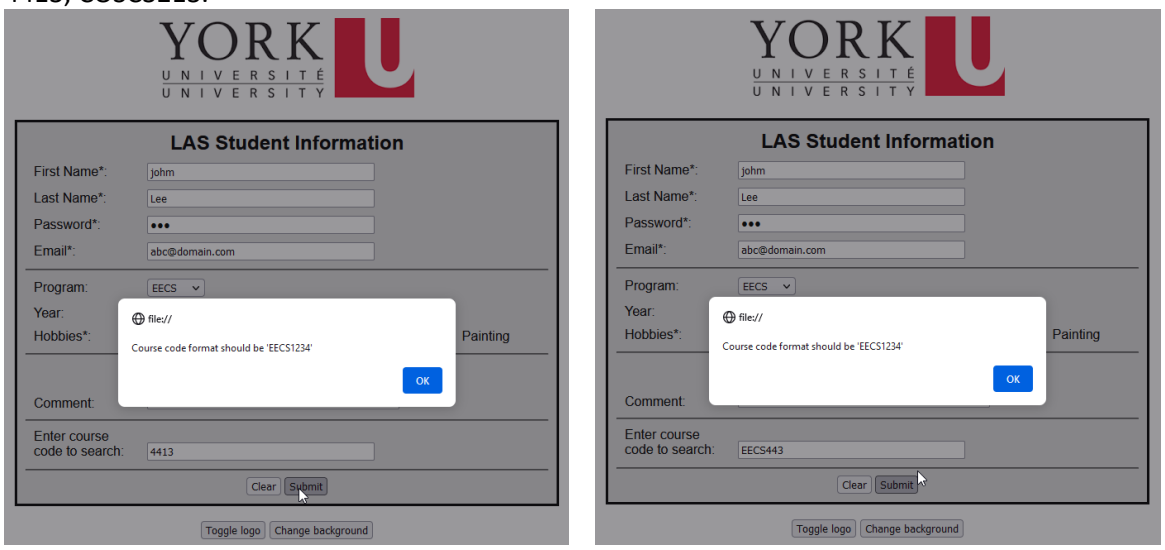
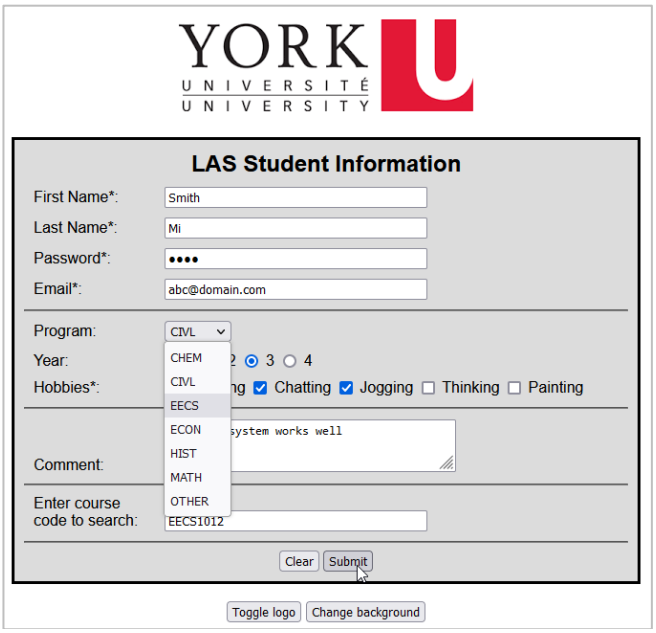


Figure 11 Checking course code format

- If validation process passes (i.e., validation function does not return a false), the form data will be sent to the server. Some examples are shown below. Note that, 10 courses are in the database, they are: EECS1012, EECS1022, EECS2011, EECS2030, EECS2031, EECS3101, EECS3310, EECS3421, EECS4101 and EECS4413.



Hi **Smith**, the server has received your form submission successfully.

Welcome **Smith Mi**
Your email address is: **abc@domain.com**

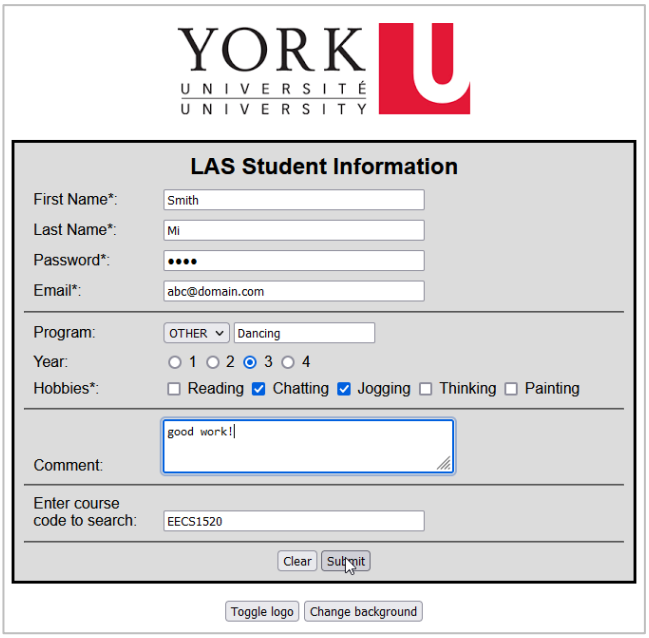
Your program is: **CIVL**
You are in year: **3**
Your hobbies: **Chatting Jogging**

Your comments: **hope the system works well**

Good luck with your studies in the 23SU term, **Smith**!

Opened database successfully

Your searched course **EECS1012**
Title: Introduction to Computing: a net-centric approach
Time: T 14:30
Location: CLE E



Hi **Smith**, the server has received your form submission successfully.

Welcome **Smith Mi**
Your email address is: **abc@domain.com**

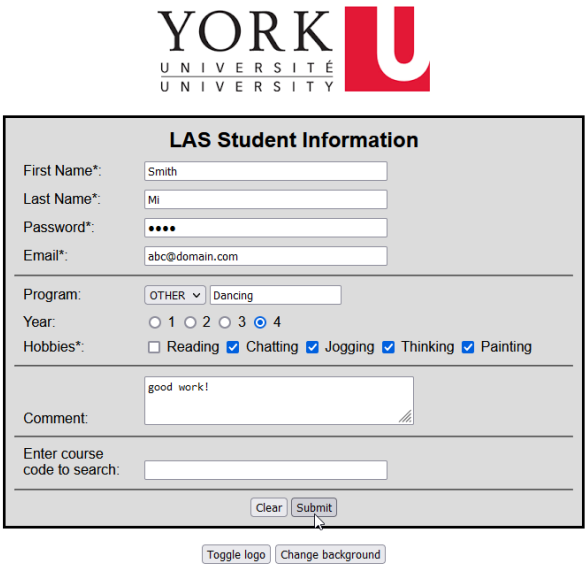
Your program is: **Dancing**
You are in year: **3**
Your hobbies: **Chatting Jogging**

Your comments: **good work!**

Good luck with your studies in the 23SU term, **Smith**!

Opened database successfully

Sorry, your searched course **EECS1520** does not exist in database now.



Hi **Smith**, the server has received your form submission successfully.

Welcome **Smith Mi**
Your email address is: **abc@domain.com**

Your program is: **Dancing**
You are in year: **4**
Your hobbies: **Chatting Jogging Thinking Painting**

Your comments: **good work!**

Good luck with your studies in the 23SU term, **Smith**!

No course to search

Part III: Simple Java Threads and Object serialization.

In this part, you get exposure to simple multi-thread programming in Java, and the concept of Object serialization.

Given an array of integers, we want to calculate the sum and maximum value in the array, and then write the result into a disk file for future retrieval by another program.

- In the first step, we could use the traditional way where one (main) process scans the array from beginning to the end. But when the array size is very large, e.g., 100000 elements, it could take a lot of time. In such cases, it would be beneficial to divide the array into multiple parts and then find the sum and max of each part simultaneously, i.e., operate on different parts of the array in parallel. This can be done by using **multi-threading** where several threads operate on the array in parallel. Specifically, we create two classes, one for calculating sum in a given range of array, and one for calculating the maximum value in a given range of array. These classes receive a portion of the array and calculate the sum and max in the portion of the array, respectively. To make the objects/instances of the classes run as threads, the two classes can either extend *Thread* class, or implement *Runnable* interface, and then override/implement the *run()* method. Call these two classes *SumThread* and *MaxThread*. The main thread creates two instances of the *SumThread* class, giving them half of the array each. It also creates two instances of the *MaxThread* class and gives them half of the array each. It then fires the four threads so they all work in parallel. That is, sum is calculated on two portions of the array simultaneously, the max is calculated on two portions of array simultaneously. Also, sum and max calculations are conducted in parallel.
  The main thread waits for all the four threads to finish, and then retrieves the results from the threads.

- In the second step, we want to persist the results, i.e., when the main program finishes, the data still exists somewhere. One approach is to write the result into a disk file, so that other programs can retrieve the data from the disk file later. Instead of writing each result into a disk file separately, and later reading out piece by piece, an easier approach is to write the results as an whole object and later retrieve the data as a whole object. To do this, we create a *SumMaxResult* class which encapsulates the max and sum values, as well as the time info. The main program stores the results into an instance/object of the *SumMaxResult* class, and then writes the object to a disk file, using *ObjectOutputStream*. To do this, as mentioned in class, the *SumMaxResult* class should be serializable (how?).
- Later, another program *Depersist* can read the disk file and retrieve the whole object, using *ObjectInputStream*.

You are given starter code for the relevant classes. Complete the classes. You can develop in plain editor and compile/run in command line, or, develop in Eclipse IDE.

In either case, If implemented correctly, run the tester class *SumMultithreadedTest* will generate output similar to the following:

```
sum thread B finish
max thread X finish
max thread Y finish
sum thread A finish
sum:66 max:15
Tue May 16 12:09:35 EDT 2023
Writing success
```

Different runs of the tester will generate different sequence od the first four lines, and different time. That is, the blue text above will change for each run.

Then, run the *Depersist* program, should generate

```
[7, 3, 5, 6, 8, 15, 9, 3, 10]
sum: 66
max: 15
Tue May 16 12:09:35 EDT 2023
```

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.

Submissions.

You should already have a **4413Lab01** folder that contains the following files and folder.

light.html light.css light.js FormsInput.html FormsInput.css form.js
images folder
java folder

If you develop the Java part using plain editor and command line, put the java files inside the java folder.
If you develop using Eclipse, export project as Archive File (zip file), and put the zip file in the java folder (please remove the provided Java starter files there).

Finally, compress the *4413Lab01* folder (.zip **or** .tar **or** .gz**), and then submit the (single) compressed file on eClass.**

Late submissions or submissions by email will NOT be accepted. Plan ahead and submit early.