

CS303A: Object-Oriented Programming & Data Structures
Northeastern Illinois University
Project: Forms & Strings

Goal:

The goal of this research lab is to modify the given HTML and Java files to practice and investigate how to validate information in String formats submitted through an HTML web form and through a JavaFX application and to compare and contrast the difference between web application and server-side applications. As this is a "research" lab, you will need to analyze the code that has been provided for you in order to be able to modify it using the materials presented in class.

The Problem:

You have been provided with an HTML file named `simple_form.html`, and `SweepstakesJavaFX.java`, which contains the code needed to display a sweepstakes entry form. You will need to add the necessary text validators to the HTML code to create a functioning form with data validation. Then, you will need to transfer this text validation to the JavaFX form.

Instructions:

- **Your group should submit ONE project write-up.**
- The project write-up should be typed! Type each question (and the question number) followed by your answer. Convert your project write-up to a .pdf.
- You should use complete sentences and proper grammar in your write-up. Use spell-check! This counts as part of your grade.
- Do **not** copy/paste directly from your sources for your answers (this is called plagiarism). Instead, you should re-word the information in your own words.
- Submit the pdf, html and java files (`simple_form.html` and `SweepstakesJavaFX.java` files) to D2L by the specified due date.

Part A: Getting Started

Download the project files. You need to finish the HTML form so that it includes data validation for each field.

Question #A.1

Double-click to open `simple_form.html` with the browser of your choice.

Now, open this file in a text editor program (such as Notepad, TextEdit, or Notepad++) and examine the code. HTML tags are enclosed by angle brackets, `<` and `>`. The first word inside the brackets indicates the type of tag. Examine the code and list 5 different types of tags (other than “html” and “input”).

Question #A.2

HTML tags are often followed by attributes– the number of attributes can vary, depending on the type of tag. Attributes typically have the format `attributename="attributevalue"` or just `attributename`. For each input tag, examine the type attribute– list 3 different types that you find.

Question #A.3

Return to your web browser. Test the code by filling out the form with obviously invalid input data. Submit your form – do you receive an error message? If so, what does it say?

Question #A.4 + coding

This form has no validation yet and can be submitted without typing in any values! You can add the required attribute to an input tag, which means that the user must enter something into that field before the form can be submitted. Go back to your text editor and add this attribute to the end of every input tag – just before the `>`. Save your code, then refresh the page in your web browser. Test your code by leaving every field blank and trying to submit. Did you receive an error message? If so, what error message(s) did you receive? Do you receive one error message, or multiple error messages?

Section B – Validating Input

Question #B.1

Each field is now required, but you can still enter wrong information – you will need to add data validation to each input field, so that we can be more confident that the submitted information will be correct. The validation rules for each field are listed below.

If you add the `pattern=""` attribute to your input tags, you can validate using regular expressions, which you learned about in class! You will put your regular expressions between the quotation marks, just like you do in Java. (**Note:** In HTML there are other attributes we can use to validate fields, such as selecting `type="tel"` for a phone number, `type="date"`, or `max=""` and `min=""` attributes for numbers/dates. For this research lab, you may only use the `required` and `pattern` attributes, so convert any number fields to text and use regex!)

Validation requirements

Every field is required

First Name and Last Name: at least 2 characters, and only contains uppercase and lowercase letters

Phone Number: Format is ###-###-####, where #'s can only be digits

Email Address: Format is xxx@xxx.yyy, where xxx and yyy stand in for any number of characters (at least one). yyy's can be any letter or digit. xxx's can be any letter, digit, or special characters: . _ or -

Lucky Number: Can be any number between 1 and 100.

Date of Birth: Format is mm/dd/yyyy or m/d/yyyy, where m, d, and y's can only be digits. The first two values for yy must be either 19 or 20.

You can choose to format the date as: dd/mm/yyyy or d/m/yyyy

Question #B.2:

For each of the six form fields, draft a regular expression that you think will work for each field – try to write them out by hand first! The six fields are:

First Name, Last Name, Phone Number, Email Address, Lucky Number, Date of Birth

Question #B.3: coding only

One at a time, test each regular expression – it is very important to test one regex at a time, because it is much easier to troubleshoot! For each regex that you test, follow these steps:

- Add one pattern attribute to an input tag and use your regular expression.
- Save your work and refresh the page.
- Test your regex by trying to input both valid and invalid data.

Part C: Creating the Java Validation

Great work! You were able to create a functioning HTML form with validation! Now, you need to create this same form in Java. You have been given the JavaFX code for the layout; you will just need to write the code that does the validation.

Question #C.1 – reading only

As you saw in the previous research lab, JavaFX is a set of packages which creates event-driven applications. Applications are laid out differently than the types of programs we have been writing so far. Look at the `main` method – there is only one line, which calls the `Application.launch` method. This method then calls the `start` method, which is directly below `main`. Read through the beginning of this method (lines 28-65), to get a general sense of what is happening in this section of code.

Unlike in HTML, Java does not have attributes to accomplish data validation. Instead, you need to create validation methods, utilizing the String methods.

Question #C.2

Since there can be multiple errors on a form, you will need to accumulate your error messages. In other words, if the form has more than one error, it should show ALL of the error messages. In this program, what type of object is error? Look up this class in the JavaFX 8 documentation. What method can retrieve the current value stored in error? What method allows you to update the value in error?

Question #C.3 - coding only

At the top of the `handle` method, you will see six Strings which correspond to the six form fields. Each validation method takes a String parameter and does not return anything. We will create the methods one by one.

First, create the `checkFirst` method, which validates the first name. If the String parameter is empty, the method should update the text of the error label to read "First name required". If the first name is invalid, the method should update error to read "First name invalid".

Inside the `handle` method, call this method on `fName`. Compile and run your code. Test using both valid and invalid data and fix any issues.

Question #C.4 - coding only

Next create the `checkLast` method, which validates the last name. If empty, the method should update the text of the error label with "Last name required". If the last name is invalid, the method should update error with "Last name invalid".

In the `handle` method, call this method on `lName`. Compile and run your code. Test and fix any issues. If you enter an incorrect first name and last name, does the form display both error messages?

Question #C.5 - coding only

Create the `checkPhone` method, which validates the phone number. If the String parameter is empty, the method should add "Phone number required" to the error label. If the phone number is invalid, add "Phone number format: ###-###-#### " to the error label. Inside the `handle` method, call this method on `inPhone`. Compile and run your code. Test and fix any errors.

Question #C.6 - coding only

Use the same process to create the method `checkEmail`, which validates the email address. If the String parameter is empty, the method should add "Email address required" to the error label. If the email address is invalid, add "Email address invalid" to the error label. Inside the `handle` method, call this method on `inEmail`. Compile and run your code. Test and fix any errors.

Question #C.7 - coding only

Use the same process to create the method `checkLuckyNum`, which validates the lucky number. If the String parameter is empty, the method should add "Lucky number required" to the error label. If the lucky number is invalid, the method should add "Lucky number must be between 1 and 100" to the error label. Inside the `handle` method, call this method on `inLuckyNum`. Compile and run your code. Test and fix any errors.

Question #C.8 - coding only

Finally, create the method `checkDob`, which validates the date of birth. If the String parameter is empty, the method should add "Date of Birth required " to the error label. If the date of birth is invalid, the method should add "Date of Birth format: ##/##/#### or ##/##/#### " to the error label. Inside the `handle` method, call this method on `inLuckyNum`. Compile and run your code. Test and fix any errors.

Part D: Final Summary

Whenever you complete a project, it is important to assess what you think went well and what you need to improve on.

Question #D.1

Which methods/parts were you able to create on your own (i.e. as a group, by researching the Java docs, etc.) and which methods/parts required you to do some research?

Question #D.2

What was a main lecture topic from class that was reinforced by this project?

Question #D.3

Based on your experience with this project, how would you approach future projects similar in nature to this one, assuming you were not provided with instructions?