Java Exercise – Advanced forms

Part 1 – The form

1. In this step we are going to build a job application entry form where you can enter the following details:
    a. Your Name
    b. Address1 + Address2
    c. Zipcode
    d. City
    e. Country (as a dropdown)
    f. Gender (male/female/don't want to say) as radio buttons
    g. Age
    h. Checkboxes for the following options:
        i. Have driver license
        ii. Can work night-shift
        iii. Can work weekends
        iv. Prefers to work full time only
    i. Personal description (multi line text form)
    j. Résumé/CV (multi-line text form)

2. Create a new Spring Boot project and create start page that contains a link to a "Submit job application".
3. Create a form that implements the form above.
    a. There are instructions how to bind the form to a form object over at http://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html#creating-a-form
4. Implement a **class** for your **model object** that will contain the data submitted and that can be used to send to the database/repository. Call it **JobApplication**.
5. Here is a resource where you can read about model binding and form submission. You'll want to base your solution on this guide, so have a quick read-through. You might also want to review Thymeleaf's standard expression syntax.
    a. https://spring.io/guides/gs/handling-form-submission/
    b. http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#standard-expression-syntax
6. When pressing submit the data is stored in a `static` in-memory object (the model, as a way to simulate a database). If you are unsure about the use of `static`, then read:
    a. http://stackoverflow.com/questions/413898/what-does-the-static-keyword-do-in-a-class
    b. http://www.javatpoint.com/static-keyword-in-java
7. When a form is submitted, you are redirected to a separate "Thanks for your application" page using a **302 redirect**. If you are unsure about what 301/302 means, then visit:

   a. http://301redirect.se/
   b. https://en.wikipedia.org/wiki/HTTP_301
   c. https://en.wikipedia.org/wiki/HTTP_302

Part 2 – Manual Validation

1. The form we created in the previous part accepts incomplete, invalid or empty forms. To address this and to remove junk-applications then we want to introduce input validation. One common issue is that **spam robots** crawl the web and post junk data to every form they find. Read more about that problem here:
   a. http://webmasters.stackexchange.com/questions/3588/how-do-spambots-work
2. So we need to add validation to our system and there are different approaches to input validation.
   See http://stackoverflow.com/questions/12146298/spring-mvc-how-to-perform-validation
3. In this step we will be using the Validator class approach, so read this article: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/validation.html
4. Implement an **ApplicationValidator** class that implements the Validator base class
5. In this class add the necessary validation rules for the forms above, including:
   a. All fields are required (should not be empty)
   b. All fields have a suitable max length
   c. The age and Zip code must contain realistic values (not 99999) and must be an integer
   d. The multi-line boxes both must have at least 100 characters of text in them.
6. If the form is not valid, then
   a. Display all the errors from the error object next to each form input element. See the Thymeleaf example at http://spring.io/guides/gs/validating-form-input/ for how one can show content based on whether there's an error.
   b. Redirect the user back to the input form page and **make sure all the fields remember the data provided by the user**.

Part 3 – Model Validation

1. Another approach is to use model validation annotations
2. Stop using the validator class from the previous step and annotate the **JobApplication** with the necessary annotations to achieve the rules from the previous step. See this tutorial for guidance https://spring.io/guides/gs/validating-form-input/
3. Also try to display the display the error messages next to each form.

Part 4 – Updating the data

1. Create a new link on the start page that links to a new page that can display the job application submitted earlier and that allows you to update your application. The same input validation rules should be applied here too

2. When a successful form is submitted, redirect the user to a page that contains a "Thanks for your update" message.
3. What happens if you enter letters A-Z in the age form?

   Hint, to simplify this exercise populate your model object with some dummy data, so you don't have to first submit a new application each time you want to try the update feature.

If you have time

- Read about what a **wysiwyg** term means:
  - https://en.wikipedia.org/wiki/WYSIWYG
  - https://en.wikipedia.org/wiki/HTML_editor
- **CKEdit** is one of the most popular editors for web forms and feel free to try to incorporate it in the form above: http://ckeditor.com/
- Using a **captcha** is a way to further protect forms from spam-robots and junk-data, read more about what a captcha is:
  - https://en.wikipedia.org/wiki/CAPTCHA
  - https://developers.google.com/recaptcha/
  - https://akismet.com/