Defining Controllers and Views

Defining Controllers

- A Controller in Spring MVC is a Java class that handles incoming requests
- The class is annotated with @Controller
- Methods are annotated with @GetMapping or @PostMapping with a URL to map specific HTTP requests to specific methods
- The Controller method selects the name of the view that should generate the response by returning the view name as a String

Defining Views

- The View is responsible for rendering the response to the client
- Thymeleaf is a typical view engine used in Spring Boot applications
- The Thymeleaf template is an html file located in the src/main/resources/templates directory in a Spring Boot project
- If there is no data in the Model to display in the View, the template could be just an ordinary html file, and the response will be identical to the html in the template

Defining Views - display data

- Thymeleaf uses the th:text attribute in the start tag of an html element for displaying data from the Model inside a tag
- The value of the th:text attribute is a Spring Expression Language (SpEL) expresstion
- The SpEL expression could look like this to display the username of a user object: ${user.username}

Demo 1 - Using th:text

- Controller method runs and selects a view by returning the name
- Controller adds attributes to a Model which can be displayed in the view

Defining Views - iteration

- The th:each attribute allows you to iterate over a collection and generate html output for each element in the collection
- The element that includes the th:each in the start tag will be repeated
- All nested elements within the element with the th:each will also be repeated
- The syntax: th:each=”item : ${collection}”

Demo 2 - Using th:each

- Using th:each to iterate over a list
- Using the status variable with th:each

Exercise 1 - Create a MVC application

- Create a Spring MVC web application on https://start.spring.io by selecting the dependencies Web and Thymeleaf (and Maven under project)
- Create a Controller class with a @GetMapping method that returns the name of a view
- Create the view with the correct name

Exercise 2 - Use the Model

- Create a new @GetMapping method with a Model as input argument
- Create or copy a Dog class to the project, add a few instance variables into the Dog class, like name and age
- Add a dog object as an attribute in the model and display the name and age of the dog in a template
- When using the th:text you can make the expression safe from nullpointer exceptions by adding a question mark after the name of the attribute. This makes Thymeleaf only call the getter in the object if the object really exists. For a dog attribute it would look like this: th:text="${dog?.name}

Exercise 3 - Use th:each

- Create a new @GetMapping method with a Model as input argument
- Create a list of type Dog and add some dog objects with different names and ages to the list, then add the list as an attribute in the Model
- Create a template and use it from the Controller method, use th:each to display all dogs with name and age in some html element. Perhaps use an unordered list <ul> with dogs in list items <li>

Defining Views - conditional statement

- Thymeleaf provides a conditional statement using the th:if attribute
- The syntax allows you to conditionally include or exclude content based on an expression
- If the expression in the th:if attribute evaluates to true then the html element will be included in the output
- If the expression evaluates to false then the element will be excluded from the output

Demo 3 - Using th:if

- Using th:if to include or exclude elements in the output

Defining Views - other attributes

- Thymeleaf also provides other attributes that look like html attributes to make it possible to use SpEl expressions as the value of the attribute

- Examples includes th:src to create a src attribute or th:href to create a href attribute and be able to use SpEL expressions as values

Demo 4 - Using th:href

- Using th:href in an anchor tag

Exercise 4 - Use th:if

- Change the template with the th:each displaying dogs so that it only display dogs with an age greater than 2

Exercise 5 - Use @RequestParam

- Create a new @GetMapping method that also take a @RequestParam input argument of type String that should be used as the name of a dog
- In the method create a HashMap with dog objects as values and the name of each dog as the key
- Use the @RequestParam input argument to get the dog out of the map with the name of the @RequestParam input argument
- Add the dog object to the model (it could be null if no dog is found by the name sent to the method). Use the same name for the attribute as in Exercise 2 and you can use the same template displaying the name and age of a dog
- Try this Controller method by calling the @GetMapping method with a request param in the URL

Exercise 6 - Use th:href

- Create a copy of the method in Exercise 3 with the list of dogs but return a different View name and create the template with this name
- Use th:each to create an anchor tag for each dog, with a th:text to display the name of each dog inside the a tag.
- Use the th:href attribute in the anchor start tag, and create a URL that matches Exercise 5 with the name of the dog as request param so that a click on a link will call the exercise 5 method with a dog name
- Use single quotes around the hardcoded part of the URL and a plus sign to add the dog.name to the URL