**Cheat Sheet**

This cheat sheet contains important takeaways from section two.

**Model View Controller**

* **Model**: Stores data needed by the view in the form of key-value pairs.
* **View**: Visual elements of a webpage.
* **Controller**: Serves web requests by managing the model and presenting the view.

|  |  |
| --- | --- |
| **Annotation** | **Purpose** |
| @Controller | Instruments the target to serve web requests. |
| @GetMapping("path") | Maps a GET request to a handler method |
| @RequestParam | Parameter to be received from a GET request |
| @PostMapping("path") | Maps a POST request to a handler method |

**Handler Method for GET Requests**

@GetMapping(value="path")

public String getMethodName(Model model,

@RequestParam(required = false) String param) {

return "view";

}

**Handler Method for POST Requests**

@PostMapping(value="path")

public String postMethodName(POJO payload) {

//submission logic here...

return "redirect:/path";

}

**Thymeleaf Expressions**

* Variable Expression ${...}: executes on a model attribute in some way.
* Selection Expression \*{field}: selects a field from a previously bound object.
* Link Expression @{/path}: URL endpoint or path to a resource.

**Variable Expressions can be used to:**

1. return a model attribute or a value that derives from it.

${object}

${object.field}

1. execute a condition.

${modelAttribute == 'plain string'}

${modelAttribute == 2}

${modelAttribute > 2}

${modelAttribute < 3}

${condition} ? 'plain string 1' : 'plain string 2'

${condition} ? 1 : 2

1. grab a collection from the model. th:each loops through every entry in the collection.

th:each= "entry: ${collection}"

1. execute utility methods on a model attribute. The utility method returns a value.

"${#class.method(target, other params)}"

You can find thymeleaf utility classes [**here**](https://github.com/thymeleaf/thymeleaf/tree/3.1-master/lib/thymeleaf/src/main/java/org/thymeleaf/expression).

**Thymeleaf Attributes**

* th:text: displays the result of an expression as simple text.

**Conditionals**

* th:if: renders an element if a condition is true.
* th:unless: renders an element if a condition is false.

<element th:if="${condition}">

<element th:unless="${condition}">

* th:switch: renders the matching case.

<element th:switch="${modelAtt}">

<sub-element th:case="'plain string'">

<sub-element th:case="2">

<sub-th:case="\*">

<element/>

**Loop**

* th:each: loop that generates as many HTML elements as there are entries in a collection.

<element th:each="entry : ${collection}">

<sub-element th:text="${entry}" th:value="${entry} th:style="${entry} ....">

<sub-element th:text="${entry}" th:value="${entry} th:style="${entry} ....">

<sub-element th:text="${entry}" th:value="${entry} th:style="${entry} ....">

</element>

**Other**

* th:object: binds an HTML element to an object.
* th:field: binds a form element to a field in the form-backing object.
* th:action: specifies an endpoint – using a link expression – where form data will be sent.

<form method="post" th:object="${object}" th:action="@{/path}">

<input th:field="\*{field}">

</form>

* th:href: uses a link expression to specify a URL endpoint or a path to an external resource.

<a th:href="@{/path(parameter = ${value})}"> <- URL

<link th:href="@{/path-starts-from-static}"> <- Path to external document

* th:style: applies a style to an HTML element based on a thymeleaf expression.
* th:value: the value of an element (usually <option> elements) is obtained from an expression.

**Arithmetic Operators**

You can perform arithmetic using the binary operators: +, -, \*, /, %.

**String Concatenation**

You can concatenate strings using the + operator.

//hello //spring

th:text="${modelAttribute} + ${modelAttribute}"

>> Result: hello spring

If the value is plain text, you **must** wrap it in single quotes.

th:text="'plain text' + ${modelAttribute}"

**Comparisons**

You can perform comparisons using: >, <, >=, <=, ==, !=.

th:if="${modelAtt > 5}"

th:if="${modelAtt == 'hey'}"

**Combining Comparisons**

The binary operators: and, or combine comparisons. They are equivalent to the Java operators && and ||.

th:if="${modelAtt > 5 and modelAtt2 == 'hello'}"

th:if="${modelAtt > 5 or modelAtt2 == 'hello'}"

**Breakpoints**

A screenshot of a computer

Description automatically generated

1. **Continue**: continues to the next breakpoint.
2. **Step over**: steps over a line.
3. **Step into**: steps into a function/constructor.
4. **Step out**: steps out of a function/constructor.
5. **Restart**: restarts the runtime.
6. **Stop**: stops the runtime

The IntelliJ buttons are very similar.

**Subscribe to our**[**YouTube Channel**](https://www.youtube.com/@RayanSlim087?sub_confirmation=1)**and Discover More Valuable Content!**