Topic 7: Intro to Spring

Part c: Building a REST Controller in Spring

Server Side Response

REST Controller

Server Application

- The controller on the server application is responsible for “controlling” the application logic and acts as the coordinator between the View (User Interface) and the Model (Data)
  - The Controller receives requests from the client (via the View), then processes the client data with the help of the Model and passes the results back to the View.
- A **REST Controller** is typically characterized by responding to client requests with web objects such as JSON format

REST Controller Annotations

- Annotations are used to indicate the purpose of each class, method, or parameter in the application
  - For example the `@SpringBootApplication` annotation indicates the main class for the application and also tells the framework to set up auto-configurations, scan for components, and read the extra configuration files.
- The `@RestController` annotation indicates that the class being implemented will act as a controller to listen for requests from clients.
  - Implicitly this also means that any requests handled by this controller will immediately be passed back to the client in the response body (illuminating the need to express this via a `@ResponseBody` annotation)

REST Controller Annotations (2)

`@RequestMapping`

- The `@RequestMapping` annotation indicates a path that the controller will respond to
  - It typically contains a `value` (path of URL) and a `method`
  - i.e. `@RequestMapping(value="/person", method=GET)`
  - If looking for a particular placeholder (identifier) in the path of the URL, we can use `value="/person/{pid}"`
  - i.e. `@RequestMapping(value="/person/2", method=GET)`
- Alternatively, we can use the following annotations to set the method
  - `@GetMapping("...")`
  - `@PostMapping("...")`
  - `@DeleteMapping("...")`

REST Controller Annotations (3)

`@RequestParam, @PathVariable`

- The `@RequestParam` annotation collects the extra information of a request that is passed in from a query string. For example
  - `@RequestParam(value="name") String name` indicates that we are looking for a (required) field with value `name` in the form of a String that will be put into a system object called `name`
  - Optional parameters can be default values by adding the annotation member `defaultValue`
- The `@PathVariable` annotation obtains the placeholder from the URL (URI)
  - The placeholder is typically called a URI template

REST Controller Annotations (4)

`@RequestBody, @PathVariable`

- The `@RequestBody` annotation specifies the body of the request from the client.
  - It must correspond to the JSON sent from the client-side
  - A POJO will be created on the server side
  - i.e.

```
@PostMapping("/request")
public ResponseObject postController(
  @RequestBody RequestObject requestObj) {
    ...
}
```


REST Controller Annotations (5)

`@PostConstruct`

- The `@PostConstruct` annotation is used on a method that needs to be executed after dependency injection is done to perform any initialization.
  - This is code that is executed after initialization of the class but before it is put into service.
  - is useful for any initializations that must be made to a class.

Client Side Requests

Client Side application

- The `java.net` package contains the URL class that allows us to create a URL object given a URL string.

```
String urlStr = "http://www.google.ca";
URL url = new URL(urlStr);
```

- The `java.net` package also contains classes for `HttpURLConnection` and `HttpsURLConnection`.
  - These classes allows the client program to make requests to a particular server either through the HTTP (default port 80) or HTTPS (default port 443) protocols:

```
HttpURLConnection connection =
        (HttpURLConnection) url.openConnection();
```

- The URL `openConnection()` method returns a **URLConnection** object that holds the attributes of the connection with the external resource.

Client Side application (2)

- The communication between client and server is in the form of Input and Output Streams
- Recall that the TCP layer breaks the messages into packages for transport
- The connection's response from the server is an input stream and can be obtained by the `getInputStream()` method
- The connection's request to the server can be specified to an output stream and can be obtained by the `getOutputStream()` method