1. Goal

Our goal is simple. We need to return the right status code - back to the client.

Note this this is something we already touched on, back in Lesson 1 of Module 2.

2. Lesson Notes

Before the release of Spring 5 - this was entirely drive with the help of the *@ResponseStatus* annotation.

Keep in mind the annotation determines the status code of the response back to the client - and can be applied both at the method level, but also on full, custom exception implementations.

2.1. Why Do We Need Anything Else?

The answer here is nuanced and depends on the type of project you're working on.

Simply put, in complex projects, creating individual exceptions for each individual and specific case can get quite complex. Instead, it's easier to be able to programmatically decide the type of response and status code on a case-by-case basis, without having to roll out new exceptions for each of these cases.

In some projects, this can be significant and can greatly simplify the codebase.

Plus, the framework already provides child classes for more specific usecases, which also save us from having the implement these ourselves.

2.2. The *ResponseStatusException*

Let's have a look at this exception by using it in a practical scenario here.

In the *create* implementation of the PrivilegeController, we'll add a simple check:

```
if (StringUtils.isBlank(resource.getName())) {

 throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "name is mandatory");
```

}

Now, we can consume this operation from the client side and see the resulting error:

{

"name": "",

"description": "New Privilege"

}

Note that the name of the operation is: *M13-L2 - Create Privilege (error)*

Of course, the response we're expecting is the one we're going to receive. The simple version is:

400 Bad Request: name is mandatory

And, we can also see exactly how this gets processed internally, by defining a breakpoint in the *ResponseStatusExceptionResolver*.

2.3. *ResponseStatusException* Subclasses

Now, let's end the lesson by having a quick look at the *ResponseStatusException* subclasses.

For example - the *ServerWebInputException* - which will result in a 400 Bad Request by default.

This is also semantically richer, as we get some extra meta information we can fill in when throwing the exception.

So, we can simply use it just like we used the parent, this time in the *update* operation of the *PrivilegeController*:

```
if (StringUtils.isBlank(resource.getName())) {
  throw new ServerWebInputException("name is mandatory for update");
}
```

Now, we can trigger an update of a Privilege resource, with a blank name (to trigger it):

{

"id": 1,

"name": "",

"description": null

}

Note that the name of the operation is: *M13-L2 - Update Privilege (error)*

And we can, of course, see the response is what we were expecting.

And we're done. These set of exceptions don't have a huge impact on the way we were handling response statuses before, but they do provide a nice, flexible new option we can leverage beyond the standard @ResponseStatus annotation.

3. Resources

- Spring ResponseStatusException