1. Goal

Understand the error semantics in the API and learn how to implement these semantics with Spring.

2. Lesson Notes

Example 3 - valid JSON, valid DTO, invalid Entity

POST http://localhost:8082/api/privileges

```
{ "name" : "PRIV13", "description" : null }
```

Example 4 - valid JSON, valid DTO - the request URI doesn't match the DTO

PUT http://localhost:8082/api/privileges/100

```
{ "id" : 1, "name" : "ROLE_PRIVILEGE_READ2", "description" : "ROLE_PRIVILEGE_READ" }
```

Handling Multiple Exceptions

One quick but important note here, that the video lesson doesn't call out explicitly is - whenever a single method is handling multiple types of exceptions, we need to make the exception argument more generic.

For example, before we had:

```
@ExceptionHandler(value = { DataIntegrityViolationException.class })
public ResponseEntity<Object> handleBadRequest(DataIntegrityViolationException ex, WebRequest request) {
```

But when we add the *MyBadRequestException* into the *@ExceptionHandler*, we need to change the argument of the method as well.

We're going to simply change that to RuntimeException - because it's the common parent for both of these:

```
@ExceptionHandler(value = { DataIntegrityViolationException.class, MyBadRequestException.class })
public ResponseEntity<Object> handleBadRequest(RuntimeException ex, WebRequest request) {
```

3. Resources

- Error Handling for REST with Spring