1. Goal

The goal of this lesson is to give you a quick-start to start using the new validation functionality in the Bean Validation 2 spec.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is : m12-lesson1-start

If you want to skip and see the complete implementation, feel free to jump ahead and import: m12-lesson1

The Bean Validation 2.0 spec was introduced with JSR 380 and has Hibernate Validator as its reference implementation.

If you're using Spring Boot in your project, you should already have the necessary dependencies on your classpath. Otherwise, you can simply include the *hibernate-validator* dependency manually.

2.1. Container Elements Validations

We're going to be working in the User entity - let's add a simple field, holding email addresses:

```
@Column

@ElementCollection

private Set<String> alternativeEmailAddresses;
```

And now let's use the new style of annotation:

```
private Set<@Email String> alternativeEmailAddresses;
```

Now, we can use PostMan and try to create a new User Resource - with an invalid email address:

// M12-L1 - 1 - Container Elements Validations 1

Naturally, we get back the 400 Bad Request, with a clear and helpful error message:

alternativeEmailAddresses - must be a well-formed email address

Next, if we fix the email address problem and send the request again:

// M12-L1 - 2 - Container Elements Validations 2 request

We get back the *201 Created* - as expected.

2.2. Date Validations

Here are the core new annotations:

- *@PastOrPresent*
- *@FutureOrPresent*
- *@Past*
- *@Future*

Let's start by adding a *dateOfBirth* attribute in our User entity:

@Temporal(value = TemporalType.DATE)

@Column

private Date dateOfBirth;

Naturally, the date of birth cannot be in the future. Let's define the new, simple validation:

@PastOrPresent

When we try to create a user - with an invalid date of birth (in the future):

// M12-L1 - 3 - Date Validation 1

We simply get a helpful, to-the-point error message:

dateOfBirth - must be a date in the past or in the present

When we fix the problem:

// M12-L1 - 4 - Date Validation 2

We get back the expected *201 Created*.

2.3. Other Validations

Beyond just the date-specific new functionality, we also get a number of other interesting and useful validation annotations:

- *@NotEmpty*
- *@NotBlank*
- *@Positive*
- *@PositiveOrZero*
- *@Negative*
- *@NegativeOrZero*

Let's start by removing the *dateOfBirth* field we introduced, and replacing it with a simple *age*:

@Column

private Long age;

Now, we can use this simple numerical validation:

@Positive

And, we can easily see this work, by trying to create a user with an invalid (negative) age:

// M12-L1 - 5 - Other Validations 1

We get the *400 Bad Request* back, along with a simple but helpful error message:

age - must be greater than 0

Once we fix the problem and try the operation again:

// M12-L1 - 6 - Other Validations 2

We get back the *201 Created*.

2.4. Optional Validation

Finally, Bean Validation 2.0 supports validation on Java 8 Optional types, including:

- *java.util.Optional*
- *java.util.OptionalInt*
- *java.util.OptionalDouble*
- *java.util.OptionalLong*

We're going to use *java.util.Optional* here.

And, since we don't want to have an *Optional* in our entity, we're going to use a DTO here.

In our UserDTO, we have an age, define as an Optional<Long> - and we're now using the *@Positive* annotation:

Optional<@Positive Long> age

In the UserController, we have the new *create* operation prepared and ready to go.

And, finally, we can try out the incorrect operation in PostMan:

// M12-L1 - 7 - Optional Validation 1

And we get the expected, helpful error message:

age - must be greater than 0

And, with a correct operation:

// M12-L1 - 8 - Optional Validation 2

We naturally get back the *201 Created*.

3. Resources

- Validating Container Elements with Bean Validation 2.0