Form Validation Annotations

Configuring a web application to support validation:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

1. Using Validation Annotations

Some simple validation examples:

```
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

public class PersonForm {

    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue(message = "Working must be true")
    private boolean working;

    @Size(min = 10, max = 200, message
      = "About Me must be between 10 and 200 characters")
    private String aboutMe;

    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    @Email(message = "Email should be valid")
    private String email;

    // standard setters and getters
}
```

All of the annotations we've used in the example are standard JSR annotations, which are part of the *jakarta.validation.constraints* package:

- `@NotNull` validates that the annotated property value isn't *null*.
- `@AssertTrue` validates that the annotated property value is *true.*
- `@Size` validates that the annotated property value has a size between the attributes *min* and *max.* We can apply it to *String*, *Collection*, *Map*, and array properties.
- `@Min` validates that the annotated property has a value no smaller than the *value* attribute.
- `@Max` validates that the annotated property has a value no larger than the *value* attribute.
- `@Email` validates that the annotated property is a valid email address.

Some annotations accept additional attributes, but the *message* attribute is common to all of them. This is the message that will usually be rendered when the value of the respective property fails validation.

Here are some additional annotations we can find in the JSR:

- `@NotEmpty` validates that the property isn't null or empty. We can apply it to *String*, *Collection*, *Map* or *Array* values.
- `@NotBlank` can be applied only to text values, and validates that the property isn't null or whitespace.
- `@Positive` and `@PositiveOrZero` apply to numeric values, and validate that they're strictly positive, or positive including 0.
- `@Negative` and `@NegativeOrZero` apply to numeric values, and validate that they're strictly negative, or negative including 0.
- `@Past` and `@PastOrPresent` validate that a date value is in the past, or the past including the present. We can apply it to date types, including those added in Java 8.
- `@Future` *and* `@FutureOrPresent` validate that a date value is in the future, or in the future including the present.

We can also apply the validation annotations to elements of a collection:

```
List<@NotBlank String> preferences;
```

In this case, any value added to the preferences list will be validated.

2. In the Web Controller

```
// Some code
@PostMapping("/")
public String checkPersonInfo(@Valid PersonForm personForm,
BindingResult bindingResult) {

    if (bindingResult.hasErrors()) {
        return "form";
    }

    return "redirect:/results";
}
```

The `checkPersonInfo` method accepts two arguments:

- A `personForm` object marked with `@Valid` to gather the attributes filled out in the form.
- A `bindingResult` object so that you can test for and retrieve validation errors.

You can retrieve all the attributes from the form, which is bound to the `PersonForm` object. In the code, you test for errors. If you encounter an error, you can send the user back to the original `form` template. In that situation, all the error attributes are displayed.

If all of the person's attribute are valid, it redirects the browser to the final `results` template.

3. HTML Front End

Now build the "main" page (from `src/main/resources/templates/form.html`) shows:

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
    <body>
        <form action="#" th:action="@{/}" th:object="${personForm}"
method="post">
            <table>
                <tr>
                    <td>Name:</td>
                    <td><input type="text" th:field="*{name}" /></td>
                    <td th:if="${#fields.hasErrors('name')}"
th:errors="*{name}">Name Error</td>
                </tr>
                <tr>
                    <td>Age:</td>
                    <td><input type="text" th:field="*{age}" /></td>
                    <td th:if="${#fields.hasErrors('age')}"
th:errors="*{age}">Age Error</td>
                </tr>
                <tr>
                    <td><button type="submit">Submit</button></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

In general, if the user enters a name or age that violates the `@Valid` constraints, it bounces back to this page with the error message displayed. If a valid name and age is entered, the user is routed to the next web page.