Input Validation

Annotations for Validation

S. No.	Annotation	Description
1	`@NotNull`	Ensures that the annotated field is not null.
2	`@NotEmpty`	Ensures that the annotated field is not null and its size/length is greater than zero. (For collections, arrays, and strings)
3	`@NotBlank`	Ensures that the annotated string is not null and its trimmed length is greater than zero.
4	`@Size`	Validates that the annotated element's size falls within the specified range.
5	`@Max`	Ensures that the annotated element is a number with a value no greater than the specified maximum.
6	`@Email`	Validates that the annotated string is a valid email address.
7	`@Pattern`	Validates that the annotated string matches the specified regular expression.
8	`@Positive`	Ensures that the annotated element is a positive number (greater than zero).
9	`@PositiveOrZero`	Ensures that the annotated element is a positive number or zero.
10	`@Negative`	Ensures that the annotated element is a negative number (less than zero).
11	`@NegativeOrZero`	Ensures that the annotated element is a negative number or zero.
12	`@Past`	Ensures that the annotated date or calendar value is in the past.
13	`@PastOrPresent`	Ensures that the annotated date or calendar value is in the past or present.

14	`@Future`	Ensures that the annotated date or calendar value is in the future.
15	`@FutureOrPresent`	Ensures that the annotated date or calendar value is in the future or present.
16	`@Digits`	Ensures that the annotated number has up to a specified number of integer and fraction digits.
17	`@DecimalMin`	Ensures that the annotated element is a number with a value no less than the specified minimum, allowing for decimal points.
18	`@DecimalMax`	Ensures that the annotated element is a number with a value no greater than the specified maximum, allowing for decimal points.
19	`@AssertTrue`	Ensures that the annotated boolean field is true.
20	`@AssertFalse`	Ensures that the annotated boolean field is false.
21	`@Valid`	Validates the associated object recursively (applies bean validation to nested objects).

Handling Validation Exceptions

- `MethodArgumentNotValidException` is thrown by the `@Valid` validation
- You can get a list of all the errors from the bindingResult of this exception
- You can use this list to return some useful error messages as the API response

Inorder to make use of the above mentioned pre-defined validators we should be adding a new dependency that allows us to work on all of them. Along with that in the end we will also look at how can I make a **custom** validator.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

At the controller level where information is accepted as part of request body from user we must provide a Validator so that the object recognises when to run the validation.

```
@PostMapping
public ResponseEntity<EmployeeDTO> createNewEmployee(
        @RequestBody @Valid EmployeeDTO inputEmployee
    ) {
    EmployeeDTO savedEmployee = employeeService.createNewEmployee(i
nputEmployee);
  return new ResponseEntity<>(savedEmployee, HttpStatus.CREATED);
}
```

At the DTO level which we send back as POJO to the user back in response and also use to accept info from them is where we apply these pre-defined (or) custom validators.

String validators

```
public class EmployeeDTO {
    private long id;

        // checks for presence of field
    @NotNull(message = "Required field in employee: name")
    private String name;
}
```

```
public class EmployeeDTO {
    private long id;

        // checks for not null and length > 0
```

```
    @NotEmpty(message = "Name of the employee cannot be empty")
    private String name;
}
```

```
public class EmployeeDTO {
    private long id;

    // checks for not null and trimmed length > 0
    @NotBlank(message = "Name of the employee cannot be blank")
    @Size(min = 3, max = 10, message = "Number of characters in name sho
uld be in the range: [3, 10]")
    private String name;
}
```

Regex Validator

```
public class EmployeeDTO {
    @Pattern(regexp = "^(ADMIN|USER)$", message = "Role of Employee ca
n be USER or ADMIN")
    @NotBlank(message = "Role of the employee cannot be blank")
    private String role; // ADMIN, USER
}
```

Email Validators

```
public class EmployeeDTO {
    @NotBlank(message = "Email of the employee cannot be blank")
    @Email(message = "Email should be in valid format")
    private String email;
}
```

Number Validators

```
public class EmployeeDTO {
    @NotNull(message = "Salary of the employee cannot be blank")
    @Positive(message = "Salary of the employee should be positive")
```

```
    @Digits(integer = 6, fraction = 2, message = "Salary of the employee ca
n be in the form X,XX,XXX.YY")
    @DecimalMax(value = "100000.99")
    @DecimalMin(value = "100.50")
    private Double salary;
}
```

Date Validator

```
public class EmployeeDTO {
    @PastOrPresent(message = "Date of joining of the employee cannot be i
n the future")
    private LocalDate dateOfJoining;
}
```

Boolean Validator

```
public class EmployeeDTO {
    @AssertTrue(message = "Employee should be active")
    @JsonProperty("isActive")
    private boolean active;
}
```

Custom Validator Defined by Developer

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Constraint(
        validatedBy = {EmployeeRoleValidator.class}
)
public @interface EmployeeRoleValidation {
    String message() default "Role of Employee can be USER or ADMIN";

    Class<?>[] groups() default {};
```

```
    Class<? extends Payload>[] payload() default {};
}
```

```
public class EmployeeRoleValidator implements ConstraintValidator<Emplo
yeeRoleValidation, String> {

    @Override
    public boolean isValid(String inputRole, ConstraintValidatorContext const
raintValidatorContext) {
        List<String> roles = List.of("USER", "ADMIN");
        return roles.contains(inputRole);
    }

}
```

Once we define that, we have to make use of the validator at the DTO level, which would result in something like below

```
public class EmployeeDTO {
    @NotBlank(message = "Role of the employee cannot be blank")
    @EmployeeRoleValidation
    private String role; // ADMIN, USER
}
```

Full Example Snippet

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class EmployeeDTO {
    private long id;

    // @NotNull(message = "Required field in employee: name") → checks f
```

```
or presence of field
    // @NotEmpty(message = "Name of the employee cannot be empty") → checks for not null and length > 0
    @NotBlank(message = "Name of the employee cannot be blank") // checks for not null and trimmed length > 0
    @Size(min = 3, max = 10, message = "Number of characters in name should be in the range: [3, 10]")
    private String name;

    @NotBlank(message = "Email of the employee cannot be blank")
    @Email(message = "Email should be in valid format")
    private String email;

    @NotNull(message = "Age of the employee cannot be blank")
    @Max(value = 80, message = "Age of Employee cannot be greater than 80")
    @Min(value = 18, message = "Age of Employee cannot be less than 18")
    private Integer age;

    // @Pattern(regexp = "^(ADMIN|USER)$", message = "Role of Employee can be USER or ADMIN")
    @NotBlank(message = "Role of the employee cannot be blank")
    @EmployeeRoleValidation
    private String role; // ADMIN, USER

    @NotNull(message = "Salary of the employee cannot be blank")
    @Positive(message = "Salary of the employee should be positive")
    @Digits(integer = 6, fraction = 2, message = "Salary of the employee can be in the form X,XX,XXX.YY")
    @DecimalMax(value = "100000.99")
    @DecimalMin(value = "100.50")
    private Double salary;

    @PastOrPresent(message = "Date of joining of the employee cannot be in the future")
    private LocalDate dateOfJoining;

    @AssertTrue(message = "Employee should be active")
```

```
    @JsonProperty("isActive")
    private boolean active;
}
```