Hibernate Validator with Example

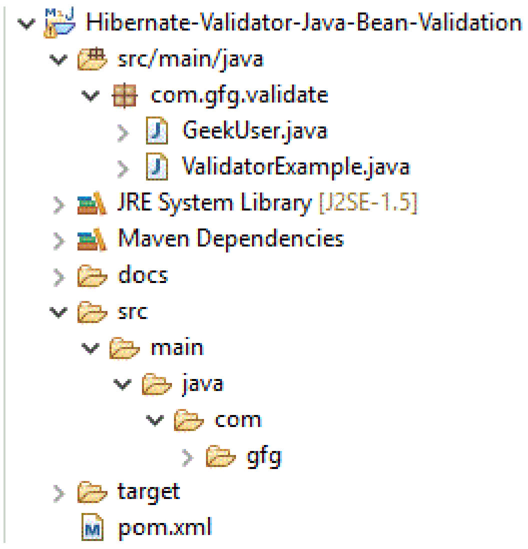
Few Annotations :

- @NotNull -> The field should not be null
- @NotEmpty -> The field should not be empty
- @Min(value = 2)-> The field value should have the min value as 2
- @Max(value = 5) -> The field maximum allowed value is 5
- @Email -> validates for a proper email

In this article let us see how to validate the data by using Hibernate validator. For that let us take a sample Maven Project containing a bean (model) class of a geekuser containing different attributes. We will see Hibernate Validator first before moving to the project.

Example Project

Project Structure:



```
Hibernate-Validator-Java-Bean-Validation
  src/main/java
    com.gfg.validate
      GeekUser.java
      ValidatorExample.java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  docs
  src
    main
      java
        com
          gfg
  target
  pom.xml
```

As it is the maven project, all the dependencies are in

pom.xml

- XML

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.gfg.validate</groupId>

    <artifactId>Hibernate-Validator-Java-Bean-Validation-
Sample</artifactId>

    <version>1.0.0</version>

    <name>Hibernate Validator</name>

    <description>Basic Example for Java Bean Validation using
Hibernate</description>



    <dependencies>

        <!-- Java bean validation API - Spec -->

        <dependency>

            <groupId>javax.validation</groupId>

            <artifactId>validation-api</artifactId>

            <version>2.0.1.Final</version>

        </dependency>



        <!-- Hibernate validator - Bean validation API Implementation --
>

        <dependency>

            <groupId>org.hibernate</groupId>

            <artifactId>hibernate-validator</artifactId>

            <version>6.0.11.Final</version>
        </dependency>



        <!-- Verify validation annotations usage at compile time -->

        <dependency>

            <groupId>org.hibernate</groupId>

            <artifactId>hibernate-validator-annotation-
processor</artifactId>

            <version>6.0.11.Final</version>

        </dependency>



        <!-- Unified Expression Language - Spec -->

        <dependency>

            <groupId>javax.el</groupId>

            <artifactId>javax.el-api</artifactId>

            <version>3.0.1-b06</version>

        </dependency>



        <!-- Unified Expression Language - Implementation -->

        <dependency>

            <groupId>org.glassfish.web</groupId>

            <artifactId>javax.el</artifactId>

            <version>2.2.6</version>
        </dependency>

    </dependencies>



</project>
```

Below two are mandatory dependencies that are required to use the annotations that got used in the code.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.0.11.Final</version>
</dependency>

<!-- Verify validation annotations usage at compile time -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator-annotation-processor</artifactId>
  <version>6.0.11.Final</version>
</dependency>
```

While defining the bean class itself, we have to annotate according to our requirements

GeekUser.java

- Java

```
import java.util.Date;



import javax.validation.constraints.Digits;

import javax.validation.constraints.Email;
import javax.validation.constraints.Future;

import javax.validation.constraints.FutureOrPresent;

import javax.validation.constraints.Max;

import javax.validation.constraints.Min;

import javax.validation.constraints.NegativeOrZero;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotEmpty;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Null;

import javax.validation.constraints.PastOrPresent;

import javax.validation.constraints.Pattern;

import javax.validation.constraints.Pattern.Flag;

import javax.validation.constraints.Positive;

import javax.validation.constraints.Size;



import org.hibernate.validator.constraints.CreditCardNumber;

import org.hibernate.validator.constraints.Range;

import org.hibernate.validator.constraints.URL;



public class GeekUser {



    @NotNull(message = "Invalid Id. Please enter your Id")
    private Long geekUserId;



    @Size(max = 20, min = 3, message = "Invalid Name, Size should be 
between 3 to 20")

    @NotEmpty(message = "Please enter your name")

    private String geekUserName;



    @Email(message = "Invalid EmailId.Please enter proper EmailId")

    @NotEmpty(message = "Please enter your EmailId")

    private String geekUserEmailId;



    @Digits(integer = 3, fraction = 3, message = "Invalid age, Maximum 
valid number for age is 3 digits")

    private int geekAge;



    @Max(value = 5, message = "Invalid currentTimeOfWritingArticles, 
Maximum allowed is 5")

    private String currentTimeOfWritingArticles;



    @Min(value = 3, message = "Invalid allowedArticles, Minimum should 
be 3")

    private String allowedForArticleReviewing;
    @NotBlank(message = "Invalid Proficiency 3, Proficiency 3 Should not
be blank")

    private String proficiency3;



    @Null(message = "Invalid Proficiency 4, Proficiency 4 should be
null")

    private String proficiency4;



    @Pattern(regexp = "YN", flags = {

            Flag.CASE_INSENSITIVE }, message = "Invalid Proficiency 5,
Enter text not matches with the standards")

    private String proficiency5;



    @Positive(message = "Invalid Rating, Value should be positive")

    private int rating;



    @NegativeOrZero(message = "Invalid value for blocklisted, Input
Number should be negative or Zero")

    private int blocklisted;



    @Future(message = "Invalid date, It should be provided as future
date")

    private Date futureDate;
    @FutureOrPresent(message = "Invalid date, It should be as future or
present date")

    private Date futureOrPresent;



    @PastOrPresent(message = "Invalid date, It should be as Past or
present date")

    private Date pastOrPresent;



    @Range(min = 1, max = 3, message = "Invalid Range is given, Range
should be within 1 to 3")

    private int rangeExample;



    @URL(message = "Invalid Url, Please provide a valid URL")

    private String urlExample;



    @CreditCardNumber(message = "Invalid Creditcard, It should not
contain invalid character")

    private String creditCardExample;



    /**

     * @param geekUserId - It should not be null

     * @param geekUserName - It should not be empty and its size should
be between 3 to 20

     * @param geekUserEmailId - It should not be empty and should be a
proper emailId
```

```
     * @param geekAge - Age value should be in 3 digit

     * @param currentTimeOfWritingArticles - Maximum
currentTimeOfWritingArticles  is 5

     * @param proficiency2 - Minimum Length of proficiency2  is 3

     * @param proficiency3 - Proficiency 3 Should not be blank

     * @param proficiency4 - Proficiency 4 should be null

     * @param proficiency5 - Invalid Proficiency 5, Enter text not
matches with the standards

     * @param rating - Invalid Rating, Value should be positive

     * @param blocklisted - Invalid value for blocklisted, Input Number
should be negative or Zero

     * @param futureDate - Invalid date, It should be provided as future
date

     * @param futureOrPresent - Invalid date, It should be as future or
present date

     * @param pastOrPresent - Invalid date, It should be as Past or
present date

     * @param rangeExample - Invalid Range is given, Range should be
within 1 to 3

     * @param urlExample - Invalid Url, Please provide a valid URL

     * @param creditCardExample - Invalid Creditcard, It should not
contain invalid character

     */

    public GeekUser(Long geekUserId, String geekUserName, String
geekUserEmailId, int geekAge,

            String currentTimeOfWritingArticles, String proficiency2,

            String proficiency3, String proficiency4,

            String proficiency5, int rating, int blocklisted,
            Date futureDate, Date futureOrPresent,Date pastOrPresent,

            int rangeExample, String urlExample, String
creditCardExample) {

        super();

        this.geekUserId = geekUserId;

        this.geekUserName = geekUserName;

        this.geekUserEmailId = geekUserEmailId;

        this.geekAge = geekAge;

        this.currentTimeOfWritingArticles =
currentTimeOfWritingArticles;

        this.allowedForArticleReviewing = proficiency2;

        this.proficiency3 = proficiency3;

        this.proficiency4 = proficiency4;

        this.proficiency5 = proficiency5;

        this.rating = rating;

        this.blocklisted = blocklisted;

        this.futureDate = futureDate;

        this.futureOrPresent = futureOrPresent;

        this.pastOrPresent = pastOrPresent;

        this.rangeExample = rangeExample;

        this.urlExample = urlExample;

        this.creditCardExample = creditCardExample;

    }
    // Setter And Getter



}
```

```
@NotNull(message = "Invalid Id. Please enter your Id")
private Long geekUserId;

@Size(max = 20, min = 3, message = "Invalid Name, Size should be
between 3 to 20")

@NotEmpty(message = "Please enter your name")
private String geekUserName;

@Email(message = "Invalid EmailId.Please enter proper EmailId")

@NotEmpty(message = "Please enter your EmailId")
private String geekUserEmailId;

@Digits(integer = 3, fraction = 3, message = "Invalid age, Maximum
valid number for age is 3 digits")
private int geekAge;

@Max(value = 5, message = "Invalid currentTimeOfWritingArticles,
Maximum allowed is 5")
private String currentTimeOfWritingArticles;

@Min(value = 3, message = "Invalid allowedArticles, Minimum should
be 3")
private String allowedForArticleReviewing;
```

```
@NotBlank(message = "Invalid Proficiency 3, Proficiency 3 Should not
be blank")

private String proficiency3;


@Null(message = "Invalid Proficiency 4, Proficiency 4 should be
null")

private String proficiency4;


@Pattern(regexp = "YN", flags = {

  Flag.CASE_INSENSITIVE }, message = "Invalid Proficiency 5, Enter
text not matches with the standards")

  private String proficiency5;


@Positive(message = "Invalid Rating, Value should be positive")

private int rating;


@NegativeOrZero(message = "Invalid value for blocklisted, Input
Number should be negative or Zero")

private int blocklisted;


@Future(message = "Invalid date, It should be provided as future
date")

private Date futureDate;


@FutureOrPresent(message = "Invalid date, It should be as future or
present date")

private Date futureOrPresent;


@PastOrPresent(message = "Invalid date, It should be as Past or
present date")

private Date pastOrPresent;


@Range(min = 1, max = 3, message = "Invalid Range is given, Range
should be within 1 to 3")

private int rangeExample;
```

```
@URL(message = "Invalid Url, Please provide a valid URL")
private String urlExample;


@CreditCardNumber(message = "Invalid Creditcard, It should not
contain invalid character")
private String creditCardExample;
```

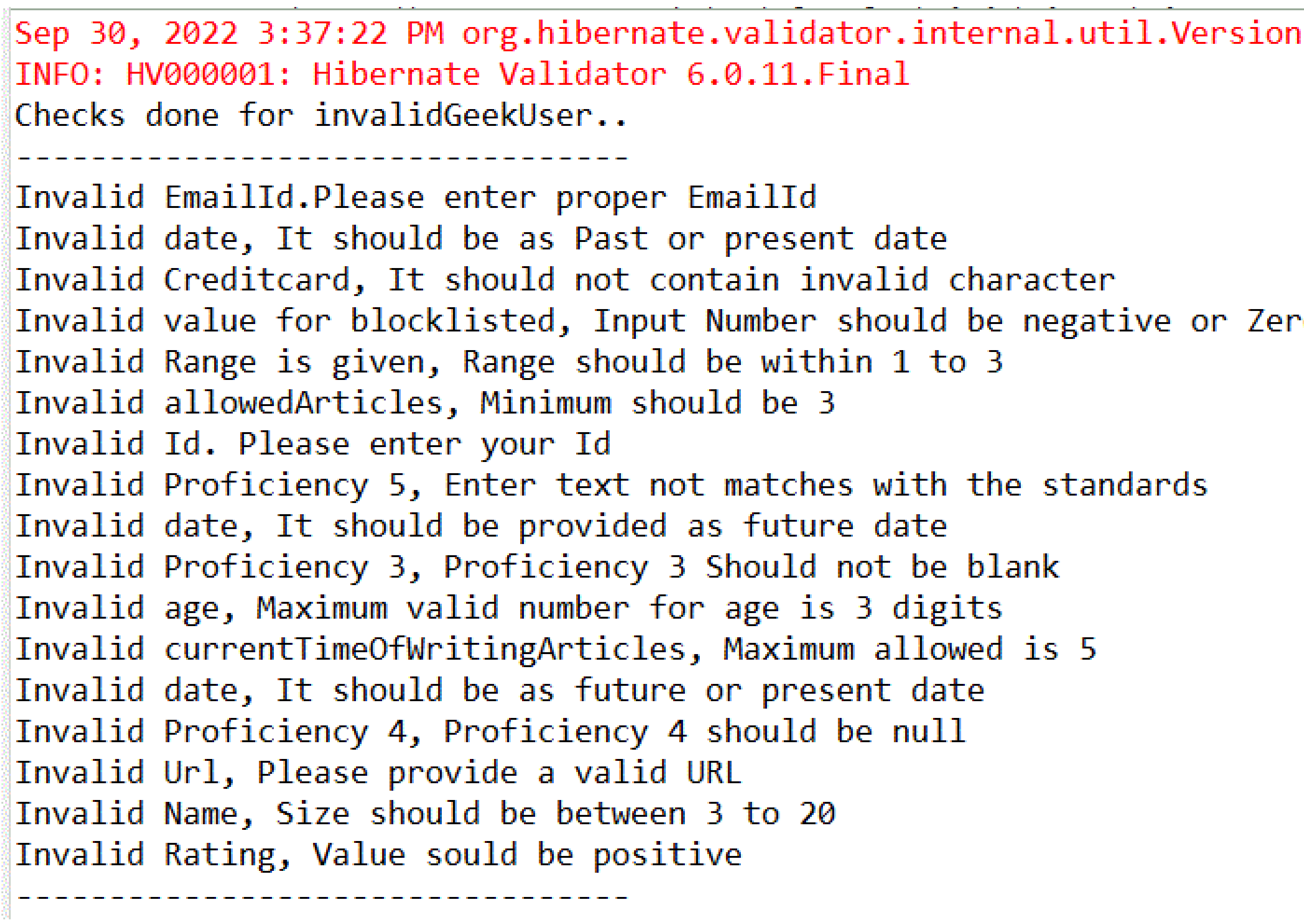
While giving the annotations, provide the message correctly which will be given as an error message if there is a violation happens with the given annotation. Now let us see the testing file that contains both the valid and invalid user. Need to write the code to print the error messages in case of errors. A lot of attributes need to be validated, and as a group, they are tested.

ValidatorExample.java

- Java

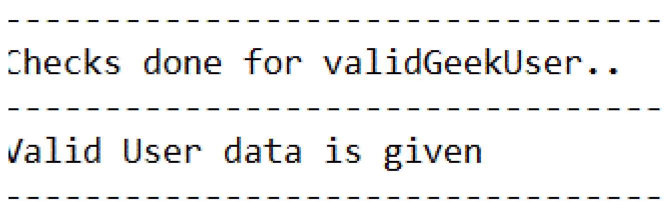
```
import java.util.Calendar;

import java.util.Date;

import java.util.Set;



import javax.validation.ConstraintViolation;

import javax.validation.Validation;

import javax.validation.Validator;

import javax.validation.ValidatorFactory;



public class ValidatorExample {



    public static void main(String[] args) {
        ValidatorFactory validatorFactory =
Validation.buildDefaultValidatorFactory();



        Validator validator = validatorFactory.getValidator();

        System.out.println("Checks done for invalidGeekUser..");

        System.out.println("-----------------------------------");

        GeekUser invalidGeekUser = new GeekUser(null, "a", "test123",

                12456, "Javatechnology", "db", "", "1234", "y", -2, 1,

                new Date(),

                getPastOrFutureDate(-2), getPastOrFutureDate(2),

                5, "sample1.com", "123@");



        Set<ConstraintViolation<GeekUser>>
constraintViolationsInvalidUser = validator.validate(invalidGeekUser);

        if (constraintViolationsInvalidUser.size() > 0) {

            for (ConstraintViolation<GeekUser> constraintViolation :
constraintViolationsInvalidUser) {

                System.out.println(constraintViolation.getMessage());

            }

        } else

            System.out.println("Valid User data is given");

        System.out.println("-----------------------------------");

        System.out.println("-----------------------------------");
        // Now check for valid geek user

        System.out.println("Checks done for validGeekUser..");

        System.out.println("--------------------------------");

        GeekUser geekUser = new GeekUser(1L, "geekauthor",
"geeka@gmail.com",

                16, "4", "3", "ML", null, "YN", 2, 0,

                getPastOrFutureDate(2),getPastOrFutureDate(1),

                getPastOrFutureDate(-2),

                2, "https://www.geeksforgeeks.org/",
"6011111111111117");



        Set<ConstraintViolation<GeekUser>> constraintViolationsvalidUser
= validator.validate(geekUser);

        if (constraintViolationsvalidUser.size() > 0) {

            for (ConstraintViolation<GeekUser> constraintViolation :
constraintViolationsvalidUser) {

                System.out.println(constraintViolation.getMessage());

            }

        } else

            System.out.println("Valid User data is given");

        System.out.println("--------------------------------");



    }
    public static Date getPastOrFutureDate(int days) {

        Calendar cal = Calendar.getInstance();

        cal.setTime(new Date());

        cal.add(Calendar.DATE, days);

        return cal.getTime();

    }



}
```

On execution of the above code, we see the below output in the console



```
Sep 30, 2022 3:37:22 PM org.hibernate.validator.internal.util.Version
INFO: HV000001: Hibernate Validator 6.0.11.Final
Checks done for invalidGeekUser..
---------------------------------
Invalid EmailId.Please enter proper EmailId
Invalid date, It should be as Past or present date
Invalid Creditcard, It should not contain invalid character
Invalid value for blocklisted, Input Number should be negative or Zer
Invalid Range is given, Range should be within 1 to 3
Invalid allowedArticles, Minimum should be 3
Invalid Id. Please enter your Id
Invalid Proficiency 5, Enter text not matches with the standards
Invalid date, It should be provided as future date
Invalid Proficiency 3, Proficiency 3 Should not be blank
Invalid age, Maximum valid number for age is 3 digits
Invalid currentTimeOfWritingArticles, Maximum allowed is 5
Invalid date, It should be as future or present date
Invalid Proficiency 4, Proficiency 4 should be null
Invalid Url, Please provide a valid URL
Invalid Name, Size should be between 3 to 20
Invalid Rating, Value sould be positive
---------------------------------
```

So error messages help us to identify what is wrong and we can correct it appropriately. At the same time, if a few inputs are correct and a few are wrong, then the relevant error messages are seen. If all are correct,



```
----------------------------------
Checks done for validGeekUser..
----------------------------------
Valid User data is given
----------------------------------
```

So, by using these validation mechanisms, nicely we can validate data even for URLs, and Creditcard also. Annotations are user-friendly and according to our user requirements, they need to be applied and get the appropriate flow. This is an efficient way of validating the form data as well. Hibernate provides the facility effectively.