Bộ môn: Kỹ Thuật Phần Mềm

Lập trình WWW *Java*

.

Spring MVC – Form Validation

Applying Built-In Validation Rules

The Need of Validation

Check of user input form for

- Require fields
- Valid numbers in range
- Valid format
- Custom business rule

Java's Standard Bean Validation API

- Defines **a meta model** and **API** for entity valadation
- Available for **server-side** applications or **client-side** applications
- www.beanvalidation.org

Spring and Validation

- Spring **version four** and **higher** supports the Bean validation API
- Preferred **method** for **validation** when **building Spring apps**
- Simply **add** the **validation JARs** to **our project**

Bean Validation Features

Validation Features
Required
Validate length
Validate numbers
Validate with regular expressions
Custom validation

Validation Annotations

Annotation	Description
@NotNull	The value can't be null.
@Min	Number must be equal or greater than the specified value.
@Max	Number must be equal or less than the specified value.
@Size	Size must be equal to the specified value.
@Pattern	Sequence follows the specified regular expression.
@Future/Past	Date must be in future or past of given date
Others...	

Our Road Map

1. Set up our development environment
2. Write code to check for a required field
3. Validate a number range with min and max
4. Apply regular expressions
5. Custom validation

Set up our development environment

Hibernate Validator

- If you're using **Jakarta EE 9**, then use **Hibernate Validator 7.x**
- If you're using **Spring 5**, then use **Hibernate Validator 6.x**
- **Hibernate Validator 6.2** has **the same features** as **Hibernate Validator 7**

Development Process

- Download Validator JAR files from Hibernate Validator website
- Add JAR files to process

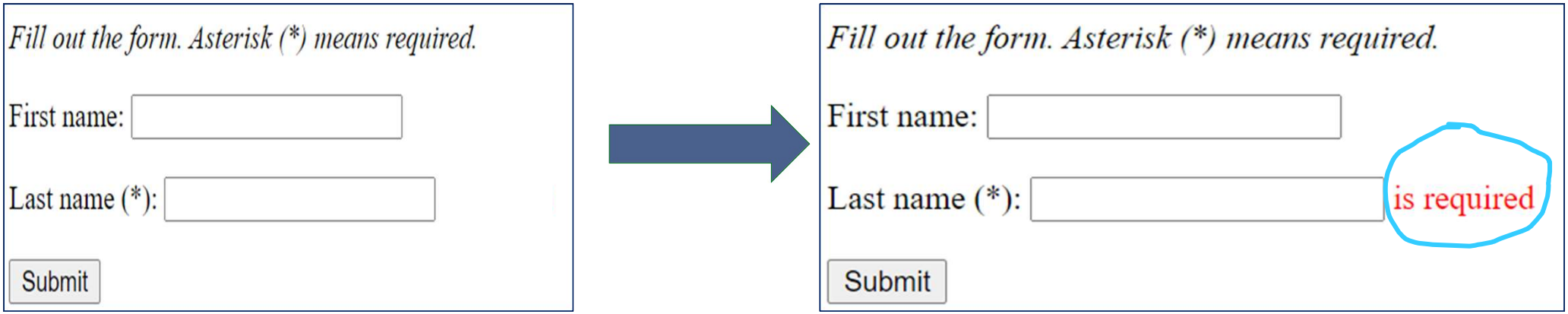
OR

- Using of Marven to add dependencies

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.2.0.Final</version>
</dependency>
```

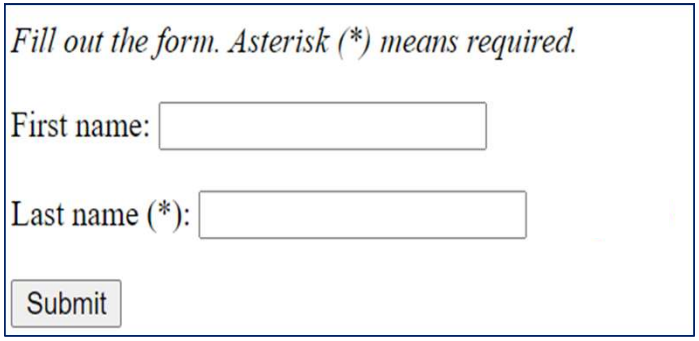
Form Validation – Required Fields

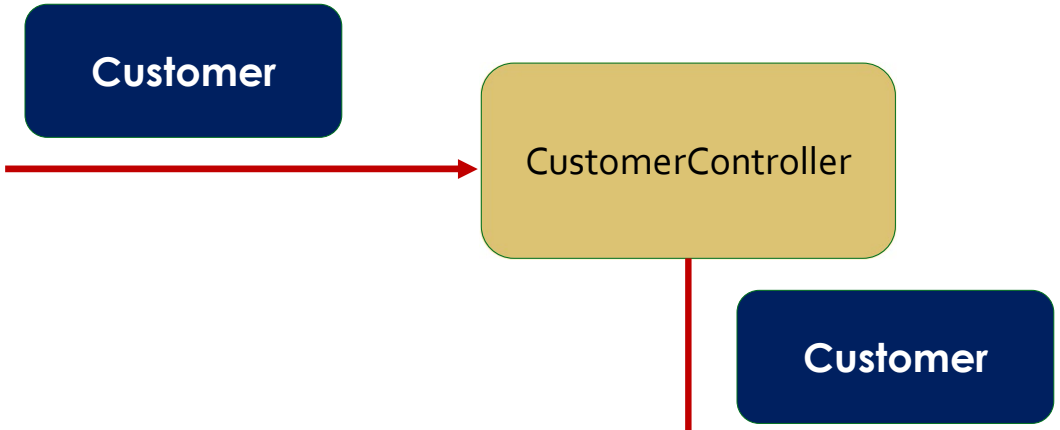
Required Field



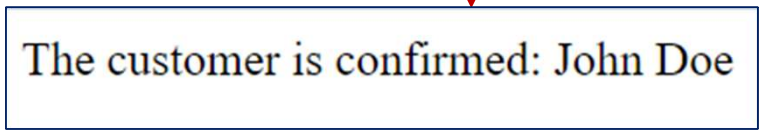
Pulling It All Together

customer-form.jsp





customer-confirmaton.jsp



Development Process

- Add **validation rule** to the **customer class**
- Display **error message** on the **HTML form**
- Perform **validation** in the **controller class**
- Update **confirmation page**

Step 1: Add validation rule to the customer class


```
package com.se.springmvc.validation;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
public class Customer {
    @NotNull(message="is required")
    @Size(min=1, message="is required")
    private String lastName;
```

Step 2: Display error message on the HTML form

File: customer-form.jsp

```
First name: <form:input path="firstName" />
<br><br>
Last name (*): <form:input path="lastName" />
<form:errors path="lastName" cssClass="error" />
<br><br>
<input type="submit" value="Submit" />
```

Strep 3: Perform validation in the controller class

@Valid: perform validation rules on this customer object

File: CustomerController.java

```
@RequestMapping("processForm")
public String processForm(
    @Valid @ModelAttribute("customer") Customer theCustomer,
    BindingResult theBindingResult) {
    if (theBindingResult.hasErrors()) {
        return "customer-form";
    else {
        return "customer-confirmation"; }
}
```

results of validation test placed in BindingResult

Step 4: Update confirmation page

File: customer-confirmation.jsp

```
<html>
<head>
    <title>Customer Confirmation</title>
</head>
<body>
The customer is confirmed: ${customer.firstName} ${customer.lastName}
</body>
</html>
```

Form Validation – @InitBinder

White Space

Our previous example had a **problem** with **white space**

- Last name with all white space passes…!!!
- it should failed
- We need to trim the white space from the input fields
  → use **@InitBinder** annotation

@InitBinder

- **@InitBinder** annotation **works** as **a pre-processor**
- It will **pre-process** each **web request** to our **controller**
- We use it to **trim leading** and **trailing white space**
- If the String only has white spaces→ trim it to **null**
- **Resolve** our **validation problem**

Register Custom Editor in Controller

Defined in Spring API. remove the leading and trailing white space

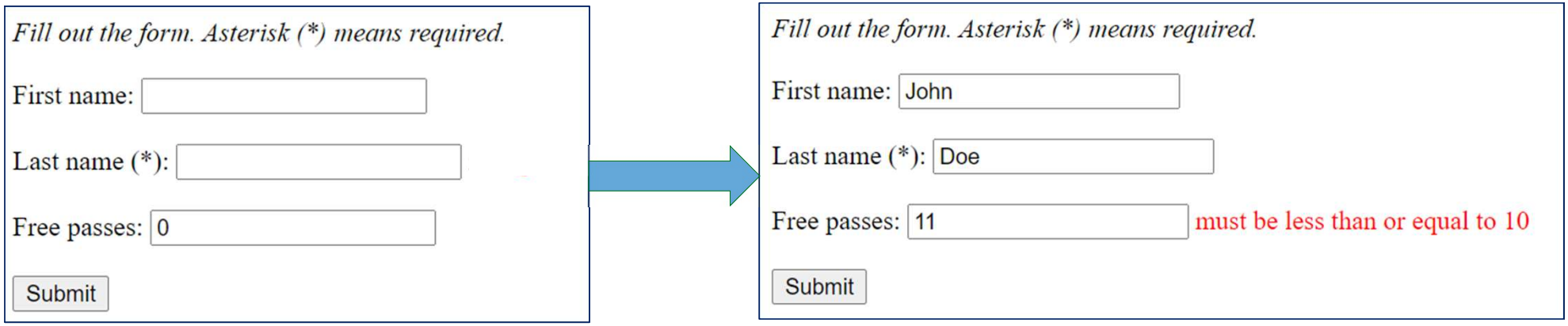
File: CustomerController.java

```
@InitBinder
public void initBinder(WebDataBinder dataBinder) {
    StringTrimmerEditor stringTrimmerEditor =
            new StringTrimmerEditor(true);
    dataBinder.registerCustomEditor(String.class, stringTrimmerEditor);}
```

trim the string down to null if it's entirely all white space

Form Validation – @Min and @Max

Validate a Number Ranger



Development Process

1. Add **the** validation rule to **Customer** class
2. Display the **error message** on the **HTML form**
3. Perform **validation** in the **Controller** class
4. Update our **confirmation page**

Development Process

File: Customer.java

```
@Min(value=0, message="must be greater than or equal to zero")
@Max(value=10, message="must be less than or equal to 10")
private int freePasses;
```

1

File: customer-form.jsp

```
<br><br>
Free passes: <form:input path="freePasses" />
<form:errors path="freePasses" cssClass="error" />
```

2

File: CustomerController.java

```
@RequestMapping("/processForm")
public String processForm(
    @Valid @ModelAttribute("customer") Customer theCustomer
    BindingResult theBindingResult) {
    if (theBindingResult.hasErrors()) {
       return "customer-form";    }
    else {
       return "customer-confirmation"; }
}
```

3

File: customer-confirmation.jsp

```
<html>
<head>
    <title>Customer Confirmation</title>
</head>
<body>
The customer is confirmed: ${customer.firstName}
${customer.lastName}
<br><br>
Free passes: ${customer.freePasses
</body>
</html>
```

4

Make an integer field required

What happen if the user doesn't provide a value for [free passe] field?

Free passes: Failed to convert property value of type java.lang.String to required type int for property freePasses; nested exception is java.lang.NumberFormatException: For input string: ""

Add @NotNull for [free passes] field then run again→ nothing change

```
@NotNull(message="is required")
@Min(value=0, message="must be greater than or equal to zero")
@Max(value=10, message="must be less than or equal to 10")
private int freePasses;
```

Solution: use the Integer type for [free passes] field and also change the type for getter and setter (to Integer)

Handle string input for integer fields


```
@NotNull(message="is required")
@Min(value=0, message="must be greater than or equal to zero")
@Max(value=10, message="must be less than or equal to 10")
private Integer freePasses;
```

What happen if the user entering some text for [free passe] field?

Free passes: ddasdsadsa Failed to convert property value of type java.lang.String to required type java.lang.Integer for property freePasses; nested exception is java.lang.NumberFormatException: For input string: "ddasdsadsa"

Solution: create a custom error message

Development Process

1. Create a custom error message

   - Source packages/resources/message.properties

   Or

   - src/resources/message.properties (Eclipse)

2. Load custom messages resources in Spring config file

   - Web Pages/WEB-INF/spring-mvc-context.xml

   Or

   - WebContent/WEB-INF/spring-mvc-context.xml

Development Process

Step 1:

File: messages.properties

```
typeMismatch.customer.freePasses=Invalid number
```

Error Type

Spring model Attribute name

Field name

Custom message

Step 2:

File: spring-mvc-context.xml

```
<bean id="messageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
<property name="basenames" value="resources/messages" />
```

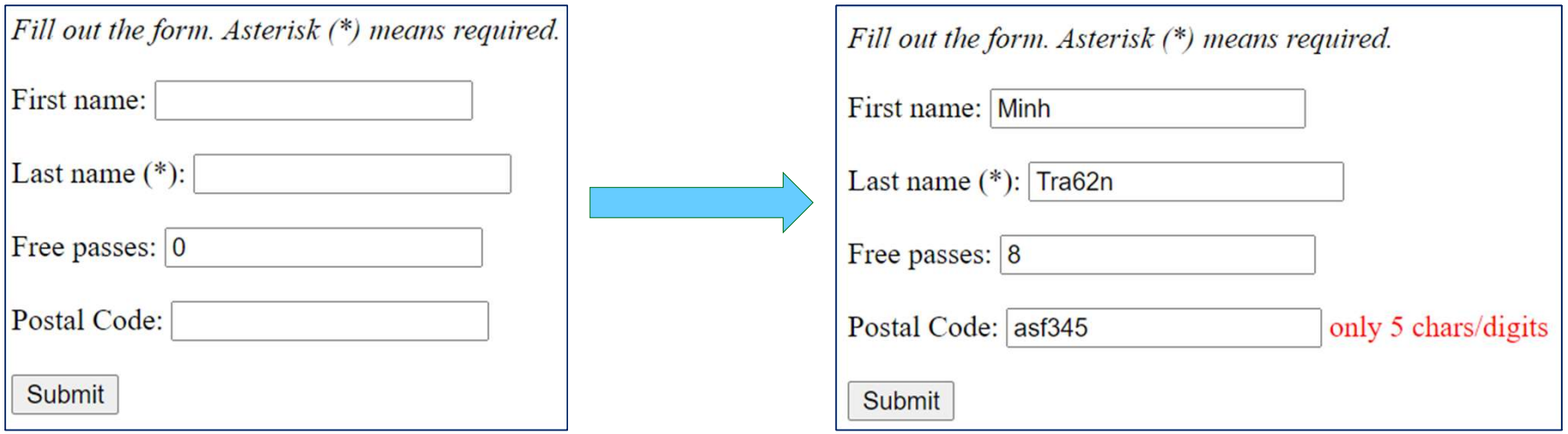
Form Validation – Regular Expressions

Regular Expressions

- A **sequence** of **characters** that **define** a **search pattern**
- This pattern is used to **find** or **match** strings
- You can also **use it** for **validation**

Apply Regular Expression to Validate a Postal Code

1. Add a **validation rule** to the **customer class**
2. Display the **error messages** on the **HTML form**
3. Update **confirmation page**



Development Process

File: Customer.java

```
@Pattern(regexp="^[a-zA-Z0-9]{5}", message="only 5 chars/digits")
private String postalCode;
```

1

File: customer-form.jsp

```
Postal Code: <form:input path="postalCode" />
<form:errors path="postalCode" cssClass="error" />
```

2

File: CustomerController.java

```
@RequestMapping("/processForm")
public String processForm(
    @Valid @ModelAttribute("customer") Customer theCustomer,
    BindingResult theBindingResult) {
    if (theBindingResult.hasErrors()) {
       return "customer-form";    }
    else {
       return "customer-confirmation"; }
}
```

File: customer-confirmation.jsp

```
<html>
<head>
    <title>Customer Confirmation</title>
</head>
<body>
The customer is confirmed: ${customer.firstName}
${customer.lastName}
<br><br>
Free passes: ${customer.freePasses}
<br><br>
Postal Code: ${customer.postalCode}
</body>
</html>
```

3

Form Validation – Custom Validation

Custom Validation

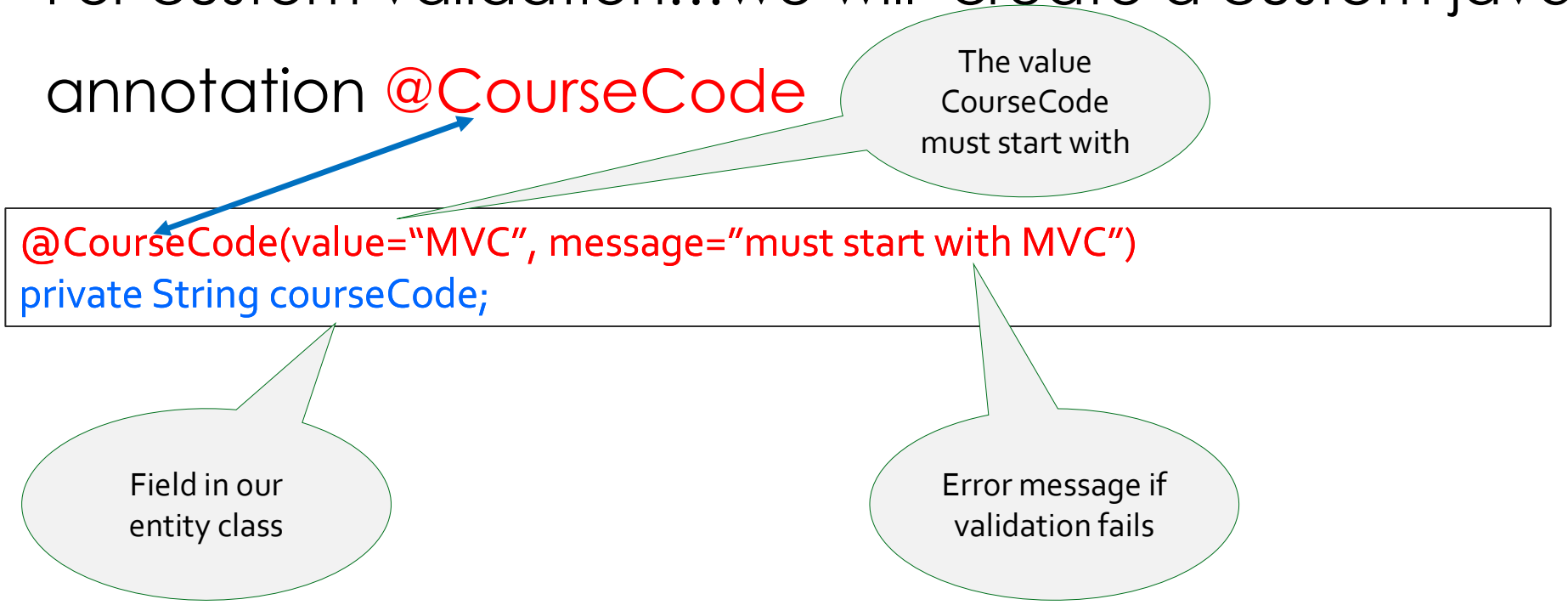
- Perform **custom validation** base on **your business rule**

  Example: Course Code **must start with "MVC"**

- Spring MVC will call our custom validation
- Custom validation returns Boolean value for pass/fail (true/false)

Create a Custom Java Annotation

» For custom validation…we will create a custom java annotation @CourseCode



Development Process

1. Create **custom validation rule**
2. Add **validation rule** to your class **(Customer)**
3. Display **error message** on HTML form
4. Update **confirmation page**

Step 1: Create custom validation rule

a) Create **@CourseCode annotation**

b) Create **@CourseCodeConstraintValidator** (the Helper class, contains business logic validation)

Step 1a: Create @CourseCode annotation

Apply annotation to a method or field

Helper class

File: CourseCode.java

```
@Constraint(validatedBy = CourseCodeConstraintValidator.class)
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface CourseCode {
```

special type interface use to define a custom annotation

Process it at runtime

```
@CourseCode(value="MVC", message="must start with MVC")
private String courseCode;
```

Step 1a: Create @CourseCode annotation

File: CourseCode.java

```
public @interface CourseCode {
// define default course code
    public String value() default "MVC";
    // define default error message
    public String message() default "must start with MVC";}
```

keep this annotation customizable

```
@CourseCode(value="MVC", message="must start with MVC")
private String courseCode;
```

Step 1b: Create @CourseCodeConstraintValidator

File: CourseCode.java

Our annotation

Type of data to validate

```
public class CourseCodeConstraintValidator implements
        ConstraintValidator<CourseCode, String> {
private String coursePrefix;
@Override
public void initialize(CourseCode theCourseCode) {
    coursePrefix = theCourseCode.value();}
@Override
public boolean isValid(String theCode,
         ConstraintValidatorContext theConstraintValidatorContext) {
    boolean result;
    if (theCode != null) {
        result = theCode.startsWith(coursePrefix);    }
    else { result = true; }
    return result;}
}
```

HTML form data entered by user

Helper class for additional error message

Spring MVC will call isValid

```
@CourseCode(value="MVC", message="must start with MVC")
private String courseCode;
```

Step 2: Add validation rule to your class (Customer)

File: Customer.java

```
public class Customer {
    @CourseCode(value="TOPS", message="must start with TOPS") //custom annotation
    private String courseCode; //courseCode courseCode
        //default of value is "MVC" and message is "must start with MVC"
    public String getCourseCode() {
        return courseCode;}
    public void setCourseCode(String courseCode) {
        this.courseCode = courseCode;}
```

Step 3: Display error message on HTML form

File:customer-form.jsp

```
Course Code: <form:input path="courseCode" />
<form:errors path="courseCode" cssClass="error" />
<br><br>
```

Step 4: Update confirmation page

File:customer-confirmation.jsp

```
<br><br>
Course code: ${customer.courseCode}
```

Fill out the form. Asterisk (*) means required.

First name: Pham

Last name (*): Quang

Free passes: 10

Postal Code: 12345

Course Code: Asdbcvbv must start with TOPS

Submit

```
public class Customer {
    @CourseCode(value="TOPS", message="must start with TOPS")
    private String courseCode; //courseCode courseCode
```

Questions