What you will learn in Topic 2

Azzeddine
RIGAT

Topic 2: Introduction to Spring MVC

- Spring MVC - Building Spring Web Apps
- Spring MVC - Creating Controllers and Views
- Spring MVC - Request Params and Request Mappings
- Spring MVC - Form Tags and Data Binding
- Spring MVC Form Validation - Applying Built-In Validation Rules
- Spring MVC Form Validation - Validating Number Ranges and Regular Expressions
- Spring MVC Form Validation - Creating Custom Validation Rules

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

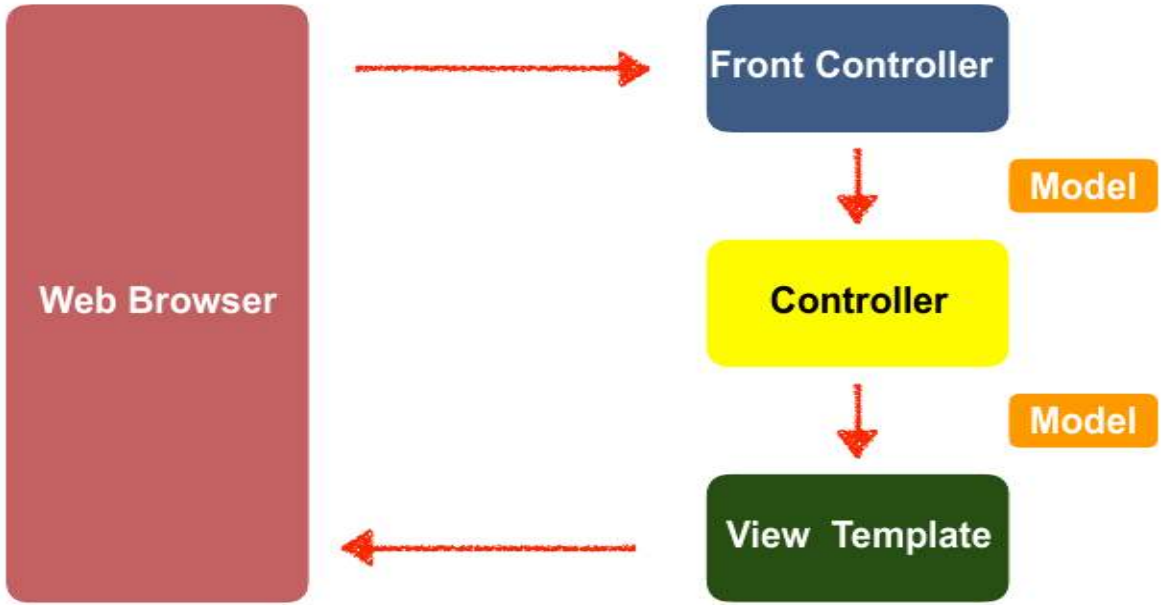
What is Spring MVC? ?

Spring MVC in a Nutshell

- Framework for building web applications in Java
- Based on Model-View-Controller design pattern
- Leverages features of the Core Spring Framework (IoC, DI)

Azzeddine RIGAT

Model-View-Controller (MVC)



Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Benefits

- The Spring way of building web app UIs in Java
- Leverage a set of reusable UI components
- Help manage application state for web requests
- Process form data: validation, conversion etc
- Flexible configuration for the view layer

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Documentation

https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc

Topic 2, Spring MVC - Building Spring Web Apps

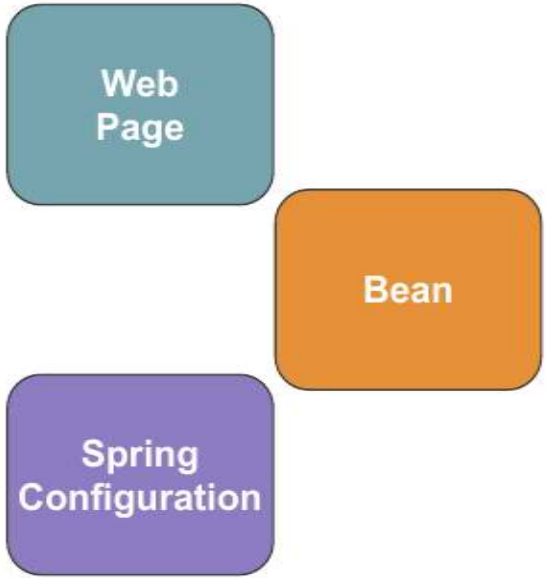
Azzeddine
RIGAT

Spring MVC Behind the Scenes

Azzeddine RIGAT

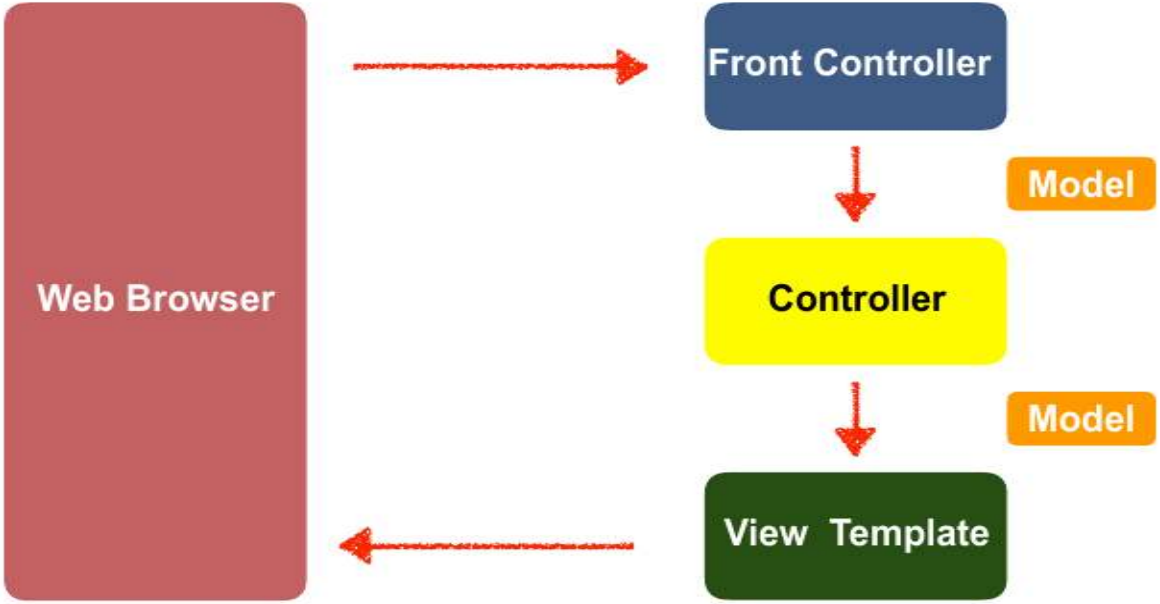
Components of a Spring MVC Application

- A set of web pages to layout UI components
- A collection of Spring beans (controllers, services, etc…)
- Spring configuration (XML, Annotations or Java)



Azzeddine RIGAT

How Spring MVC Works Behind the Scenes



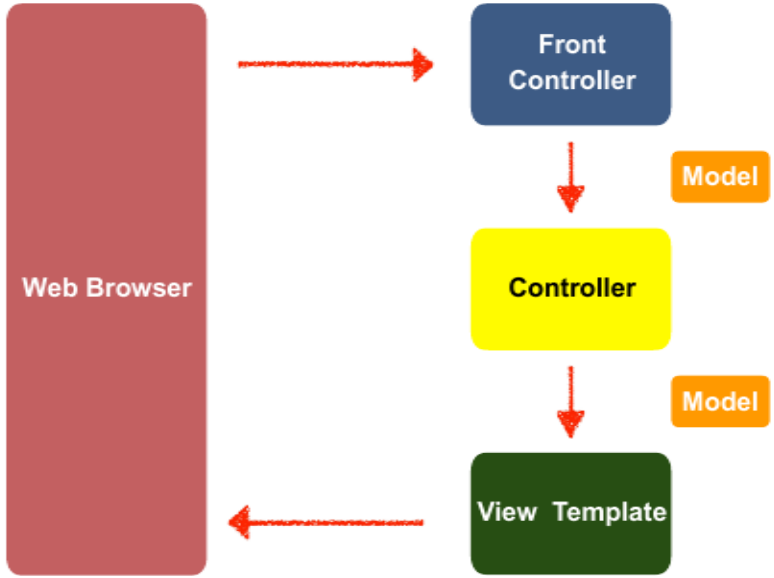
Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Front Controller

Front controller known as DispatcherServlet

- Part of the Spring Framework
- Already developed by Spring Dev Team



You will create

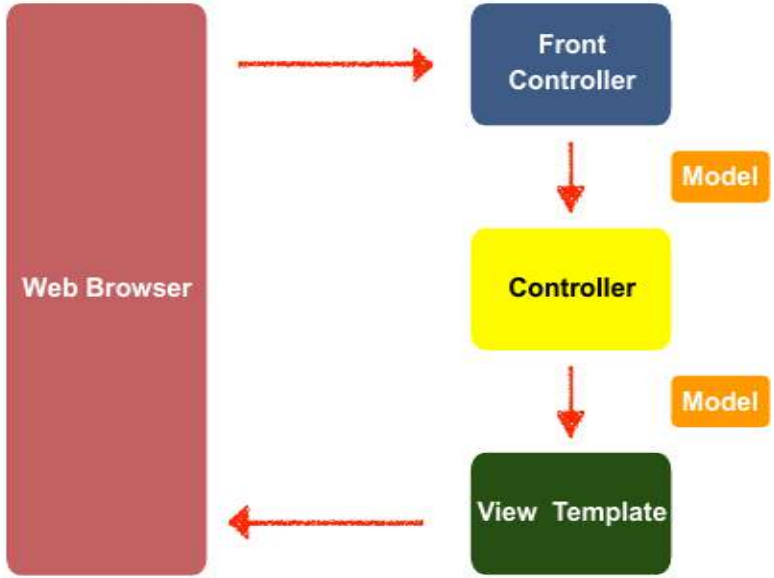
- **M**odel objects (orange)
- **V**iew templates (dark green)
- **C**ontroller classes (yellow)

Azzeddine RIGAT

Controller

Code created by developer

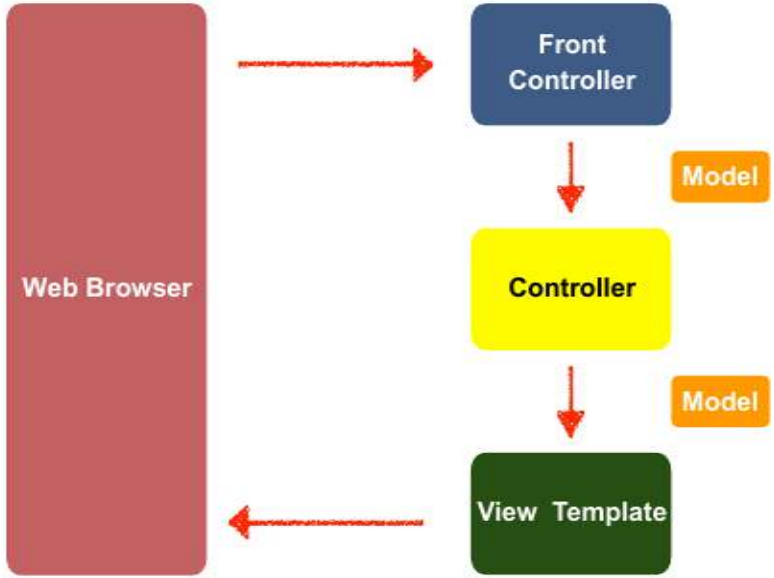
- Contains your business logic
- Handle the request
- Store/retrieve data (db, web service…)
- Place data in model
- Send to appropriate view template



Azzeddine RIGAT

Model

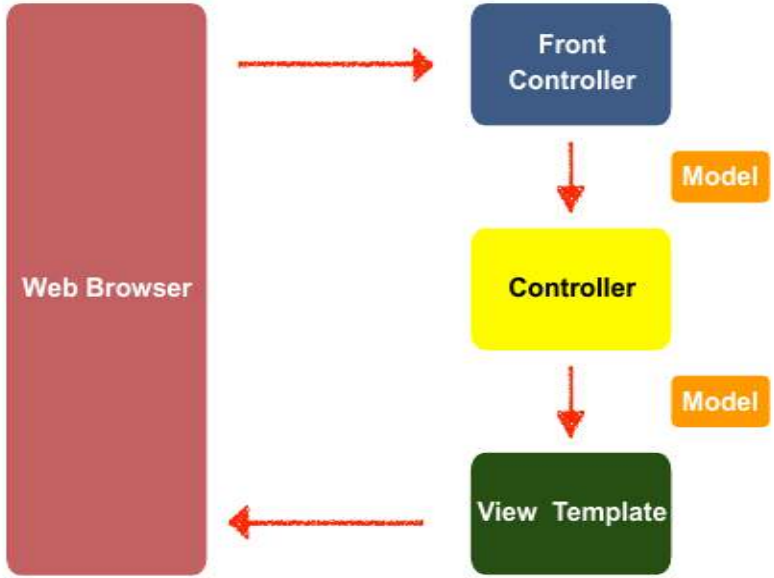
- Model: contains your data
- Store/retrieve data via backend systems
  - database, web service, etc…
  - Use a Spring bean if you like
- Place your data in the model
  - Data can be any Java object/collection



Azzeddine RIGAT

View Template

- Spring MVC is flexible
  - Supports many view templates
- Most common is **JSP** + **JSTL**
- Developer creates a page
  - Displays data



Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

View Template (more)

- Other view templates supported
  - Thymeleaf, Groovy
  - Velocity, Freemarker, etc…
- For details, see:

https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-view

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Dev Environment Check Point

At this point of the course you should have installed:

- Apache Tomcat 8.03
- Eclipse (Java EE version)
- Connected Eclipse to Tomcat

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Dev Environment Check Point

Additional Things To Do:

- Download the **Jars** necessary for annotations, and validation from Github

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Configuration

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Configuration Process - Part 1

Add configurations to file: **WEB-INF/web.xml**

1. Configure Spring MVC Dispatcher Servlet

2. Set up URL mappings to Spring MVC Dispatcher Servlet

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Spring MVC Configuration Process - Part 2

Add configurations to file: **WEB-INF/spring-mvc-servlet.xml**

3. Add support for Spring component scanning

4. Add support for conversion, formatting and validation

5. Configure Spring MVC View Resolver

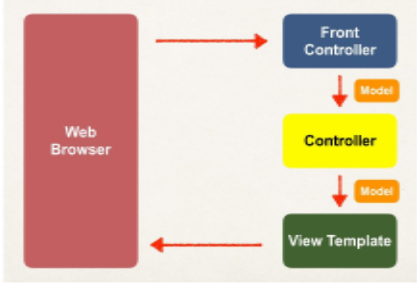
Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine RIGAT

Step 1: Configure Spring DispatcherServlet

File: web.xml

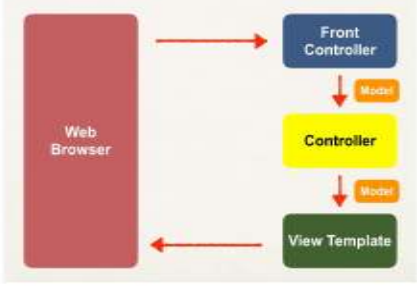
```
<web-app>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc-servlet.xml</param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
    </servlet>

</web-app>
```



Step 2: Set up URL mappings to Spring MVC Dispatcher Servlet

File: web.xml

```
<web-app>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        ...
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```



Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine RIGAT

Step 3: Add support for Spring component scanning

File: spring-mvc-servlet.xml

```
<beans>

    <!-- Step 3: Add support for component scanning -->
    <context:component-scan base-package="edu.javaweb.spring.mvc" />

</beans>
```

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine RIGAT

Step 4: Add support for conversion, formatting and validation

File: spring-mvc-servlet.xml

```
<beans>

    <!-- Step 3: Add support for component scanning -->
    <context:component-scan base-package="edu.javaweb.spring.mvc" />

    <!-- Step 4: Add support for conversion, formatting and validation support -->
    <mvc:annotation-driven/>

</beans>
```

Azzeddine RIGAT

Step 5: Configure Spring MVC View Resolver

File: spring-mvc-servlet.xml

```
<beans>

    <!-- Step 3: Add support for component scanning -->
    <context:component-scan base-package="edu.javaweb.spring.mvc" />

    <!-- Step 4: Add support for conversion, formatting and validation support -->
    <mvc:annotation-driven/>

    <!-- Step 5: Define Spring MVC view resolver -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

Azzeddine RIGAT

View Resolver Configs - Explained

When your app provides a “view” name, Spring MVC will

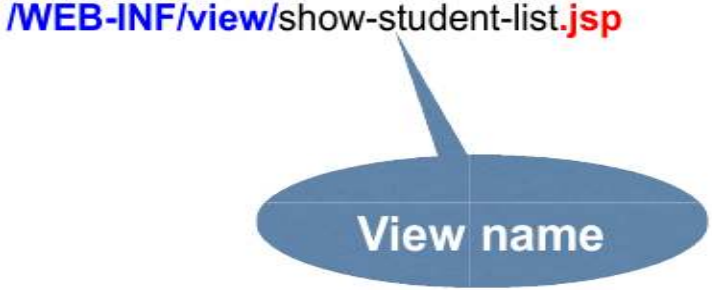
- prepend the prefix
- append the suffix

```
<bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/view/" />
      <property name="suffix" value=".jsp" />
</bean>
```

Azzeddine RIGAT

View Resolver Configs - Explained

```
<bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/view/" />
      <property name="suffix" value=".jsp" />
</bean>
```



Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine RIGAT

Summary of Spring Config:

- Two steps in **WEB-INF/web.xml**
- Three steps in our xml config

Check **spring-mvc- demo1-config-files**

Topic 2, Spring MVC - Building Spring Web Apps

Azzeddine
RIGAT

Assignment 8

Configure the Spring Dispatcher Servlet using all Java Code (no xml).html

Topic 2, Spring MVC - Creating Controllers and Views

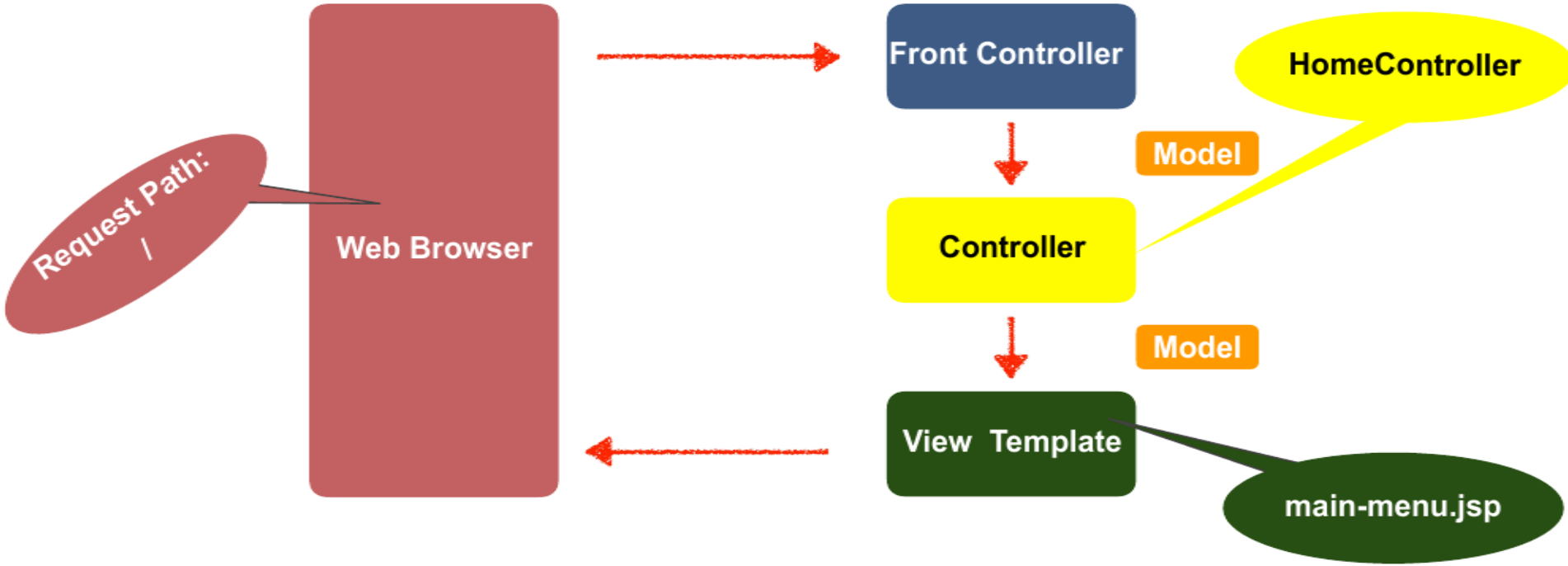
Azzeddine
RIGAT

Creating Controllers and Views

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Our First Spring MVC Example



Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Development Process

1.Create Controller class

2.Define Controller method

3.Add Request Mapping to Controller method

4.Return View Name

5.Develop View Page

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Step 1: Create Controller class

Annotate class with **@Controller**

- **@Controller** inherits from @Component … supports scanning

```
@Controller
public class HomeController {

                                        }
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Step 2: Define Controller method

```
@Controller
public class HomeController {

    public String showMyPage() {
    ...
                                    }

}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Step 3: Add Request Mapping to Controller method

```
@Controller
public class HomeController {
        @Request  Mapping("/")
        public String showMyPage() {
                ...
    }

}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Step 4: Return View Name

```
@Controller
public class HomeController {

        @RequestMapping("/")
        public String showMyPage() {
                        return "main-menu";
            }

}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine RIGAT

Finding the View Page

```
<bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/view/" />
      <property name="suffix" value=".jsp" />
</bean>
```

/WEB-INF/view/main-menu.jsp

View name

```
@Controller
public class HomeController {

      @RequestMapping("/")
      public String showMyPage() {
            return "main-menu";
      }
}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Step 5: Develop View Page

File: /WEB-INF/view/main-menu.jsp

```
<html>
   <body>

   <h2>Spring MVC Demo - Home Page</h2>

   </body>
</html>
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Recap - Development Process

1.Create Controller class

2.Define Controller method

3.Add Request Mapping to Controller method

4.Return View Name

5.Develop View Page

Check **spring-mvc-demo2-create-home-controller-and-view**

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Assignment 9

Use CSS, JavaScript and Images in Spring MVC Web App.html

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Reading Form Data with Spring MVC

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

High Level View

helloworld-form.jsp

helloworld.jsp

Hello World of Spring!

Student name: John Doe

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Application Flow

Request Mapping
/showForm

HelloWorld Controller

helloworld-form.jsp

What's your name? Submit Query

helloworld-form.jsp

What's your name? Submit Query

Request Mapping
/processForm

HelloWorld Controller

helloworld.jsp

Hello World of Spring!

Student name: John Doe

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Controller Class

```
@Controller
public class HelloWorldController {
    // need a controller method to show the initial HTML form
    @RequestMapping("/showForm")
    public String showForm() {
        return "helloworld-form";
    }
    // need a controller method to process the HTML form
    @RequestMapping("/processForm")
    public String processForm() {
        return "helloworld";
    }
}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Development Process

1. **Create Controller class**

2. **Show HTML form**
   a. Create controller method to show HTML Form
   b. Create View Page for HTML form

3. **Process HTML Form**
   a. Create controller method to process HTML Form
   b. Develop View Page for Confirmation

Check **spring-mvc- demo3-reading-html-form-data**

Topic 2, Spring MVC - Creating Controllers and Views

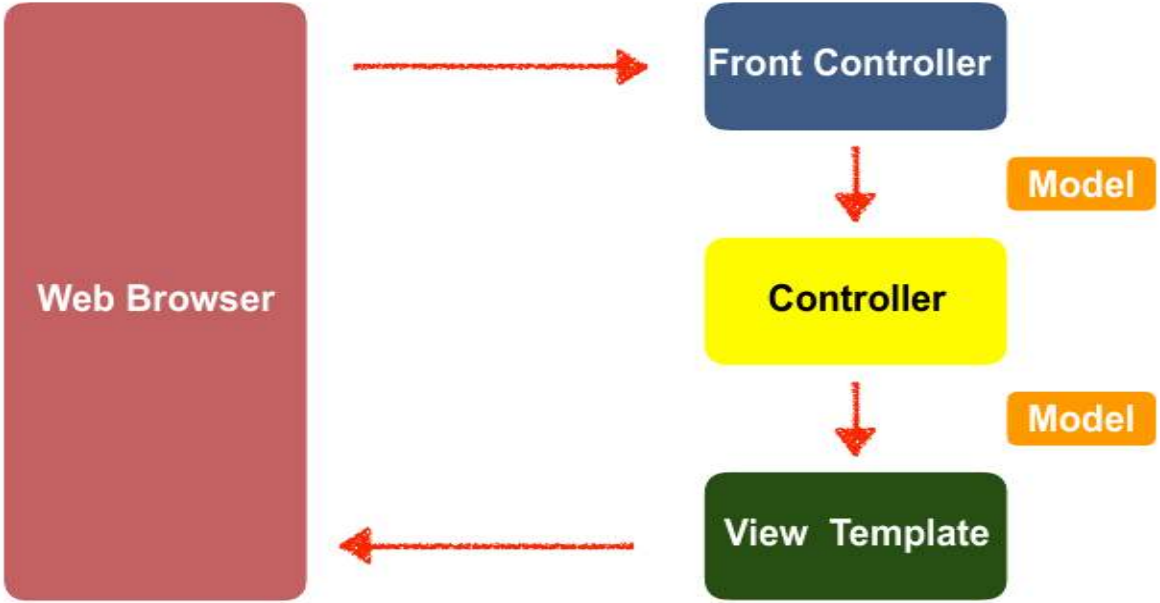
Azzeddine
RIGAT

Adding Data to Spring Model

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Focus on the Model

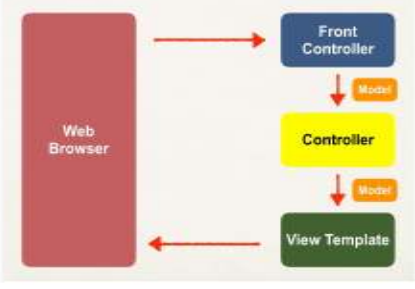


Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine RIGAT

Spring Model

- The **Model** is a container for your application data
- In your Controller
  - You can put anything in the **model**
  - strings, objects, info from database, etc…
- Your View page (JSP) can access data from the **model**



Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Code Example

- We want to create a new method to process form data
- Read the form data: student's name
- Convert the name to uppercase
- Add the uppercase version to the model

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine RIGAT

Passing Model to your Controller

```
@RequestMapping("/processFormVersionTwo")
public String letsShoutDude(HttpServletRequest request, Model model) {

    // read the request parameter from the HTML form
    String theName = request.getParameter("studentName");
    // convert the data to all caps
    theName = theName.toUpperCase();
    // create the message
    String result = "Yo! " + theName;
    // add message to the model
    model.addAttribute("message", result);

    return "helloworld";
}
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

View Template - JSP

```
<html>
   <body>

      Hello World of Spring!
      ...

      The message: ${message}

      </body>
</html>
```

Check **spring-mvc- demo4-adding-data-to-the-spring-model**

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Adding more data to your Model

```
// get the data
//
String result = …
List<Student> theStudentList = …
ShoppingCart theShoppingCart = …

// add data to the model
//
model.addAttribute("message", result);
model.addAttribute("students", theStudentList);
model.addAttribute("shoppingCart", theShoppingCart);
```

Topic 2, Spring MVC - Creating Controllers and Views

Azzeddine
RIGAT

Assignment 10

Add a shoppingcart class, with at least two attributes: item name and its price
Add a form text that has at least three text inputs then submit them using the the HttpServerletRequest
Finally, display your shopping items, and the sum of the items.

Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

Reading HTML Form Data with **@RequestParam** Annotation

Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

Instead of using HttpServletRequest

```
@RequestMapping("/processFormVersionTwo")
public String letsShoutDude(HttpServletRequest request, Model model) {

    // read the request parameter from the HTML form
    String theName = request.getParameter("studentName");

    ...
}
```

Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

Bind variable using @RequestParam Annotation

```
@RequestMapping("/processFormVersionTwo")
    public String letsShoutDude(
        @RequestParam("studentName") String theName,  Model model) {

    // now we can use the variable: theName

}
```

Behind the scenes:

- **Spring will read param from request: studentName**
- **Bind it to the variable: theName**

Check **spring-mvc- demo5-binding-request-params**

Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

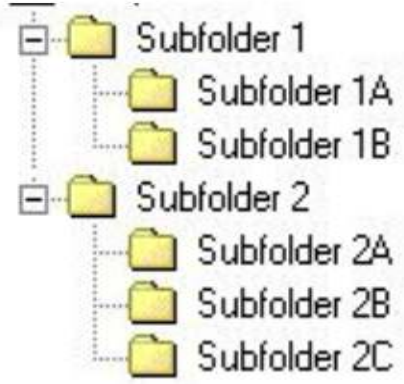
Add **@RequestMapping** to Controller

Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

Adding Request Mappings to Controller

- Serves as parent mapping for controller
- All request mappings on methods in the controller are relative
- Similar to folder directory structures



Topic 2, Spring MVC - Request Params and Request Mappings

Azzeddine
RIGAT

Controller Request Mapping

```
@RequestMapping("/funny")
public class FunnyController {

    @RequestMapping("/showForm")
    public String showForm() {
    ...
    }

    @RequestMapping("/processForm")
    public String process(HttpServletRequest request, Model model) {
    ...
    }
}
```

Controller Mapping

/funny/showForm

/funny/processForm

Check **spring-mvc- demo6-controller-level-mappings**

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tags

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Review HTML Forms

- HTML Forms are used to get input from the user

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tags

- Spring MVC Form Tags are the building block for a web page

- Form Tags are configurable and reusable for a web page

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Data Binding

- Spring MVC Form Tags can make use of data binding
- Automatically setting / retrieving data from a Java object / bean

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tags

- Form tags will generate HTML for you :-)

Form Tag	Description
form:form	Main form container
form:input	Text field
form:textarea	Multi-line text filed
form:checkbox	Check box
form:radiobutton	Radio buttons
Form:select	Drop down list
more….	

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Web Page Structure

- JSP page with special Spring MVC Form tags

```
<html>

   … regular html …

   … Spring MVC form tags …

   … more html …

</html>
```

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

How To Reference Spring MVC Form Tags

- Specify the Spring namespace at beginning of JSP file

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

Topic 2, Spring MVC - Form Tags and Data Binding

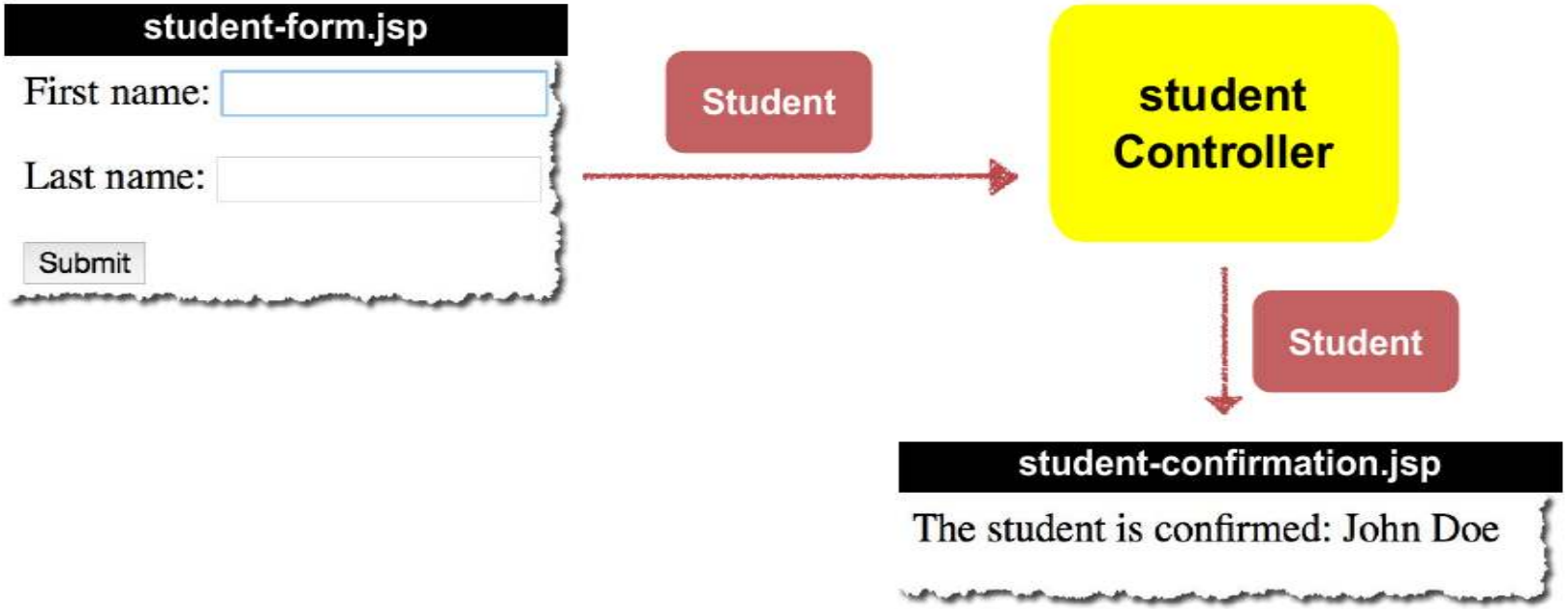
Azzeddine
RIGAT

Spring MVC Form Tag - Text Field

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Big Picture



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Showing Form

In your Spring Controller

- Before you show the form, you must add a **model attribute**
- This is a bean that will hold form data for the **data binding**

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Show Form - Add Model Attribute

Code snippet from Controller

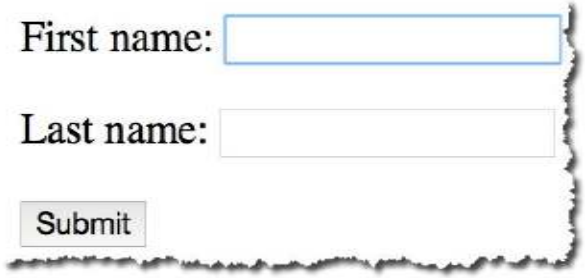
```
@RequestMapping("/showForm")
public String showForm(Model theModel) {
    theModel.addAttribute("student", new Student());
    return "student-form";
}
```

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Setting up HTML Form - Data Binding

```
<form:form action="processForm" modelAttribute="student">

   First name: <form:input path="firstName" />

   <br><br>

   Last name: <form:input path="lastName" />

   <br><br>

   <input type="submit" value="Submit" />

</form:form>
```



Azzeddine RIGAT

When Form is Loaded … fields are populated

```
<form:form action="processForm" modelAttribute="student">

   First name: <form:input path="firstName" />

   <br><br>

   Last name: <form:input path="lastName" />

   <br><br>

   <input type="submit" value="Submit" />

</form:form>
```

When form is **loaded**, Spring MVC will call:

student.getFirstName()

student.getLastName

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

When Form is submitted … calls setter methods

```
<form:form action="processForm" modelAttribute="student">

   First name: <form:input path="firstName" />

   <br><br>

   Last name: <form:input path="lastName" />

   <br><br>

   <input type="submit" value="Submit" />

</form:form>
```

When form is **submitted**, Spring MVC will call:

student.setFirstName(…)

student.setLastName(…)

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Handling Form Submission in the Controller

Code snippet from Controller

```
@RequestMapping("/processForm")
public String processForm(@ModelAttribute("student") Student theStudent) {

    // log the input data
    System.out.println("theStudent: " + theStudent.getFirstName()
    + " " + theStudent.getLastName());

    return "student-confirmation";
}
```

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Confirmation page

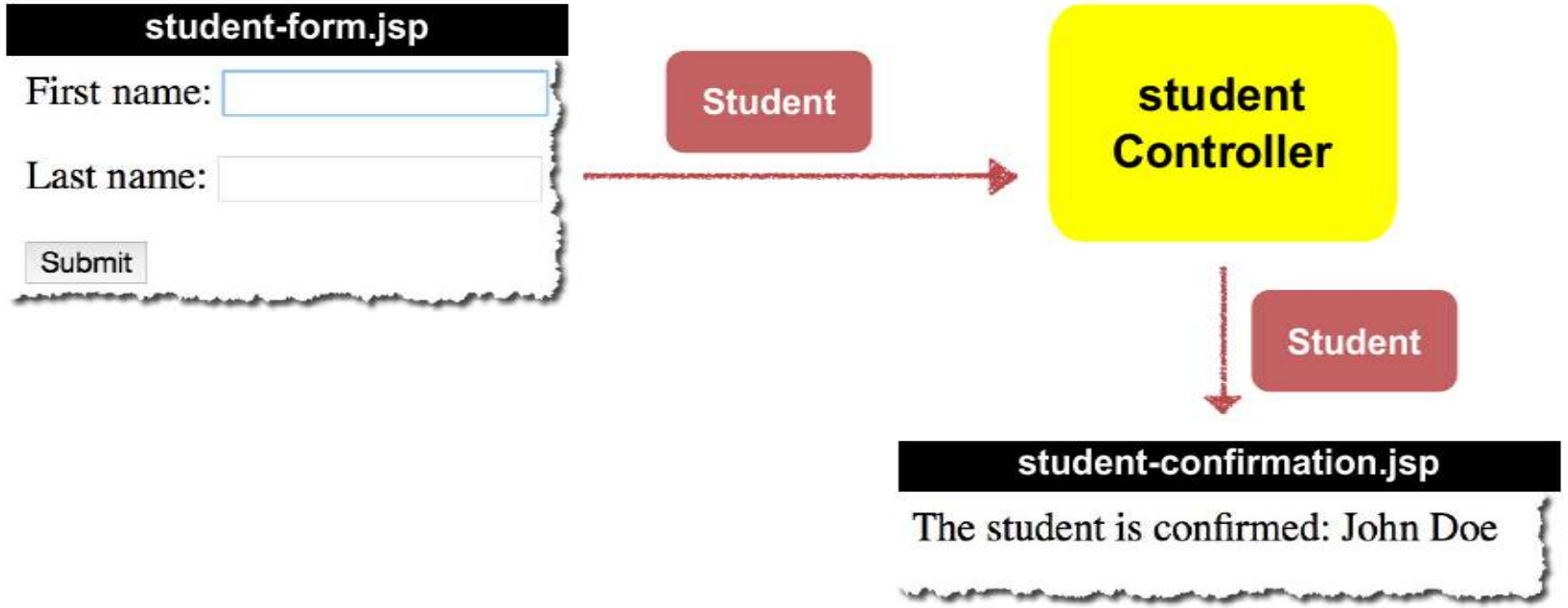
```
<html>
   <body>

   The student is confirmed: ${student.firstName} ${student.lastName}

   </body>
</html>
```

The student is confirmed: John Doe

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Pulling It All Together



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Development Process

1.Create Student class

2.Create Student controller class

3.Create HTML form

4.Create form processing code

5.Create confirmation page

Check **spring-mvc- demo7-form-tags-text-fields**

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tag - Drop Down List

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Review of HTML <select> Tag

First name:

Last name:

Country: Brazil

Submit

```
<select name="country">
        <option>Brazil</option>
        <option>France</option>
        <option>Germany</option>
        <option>India</option>
        ...
</select>
```

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Tag

Drop-Down List is represented by the tag

<form:select>

Azzeddine RIGAT

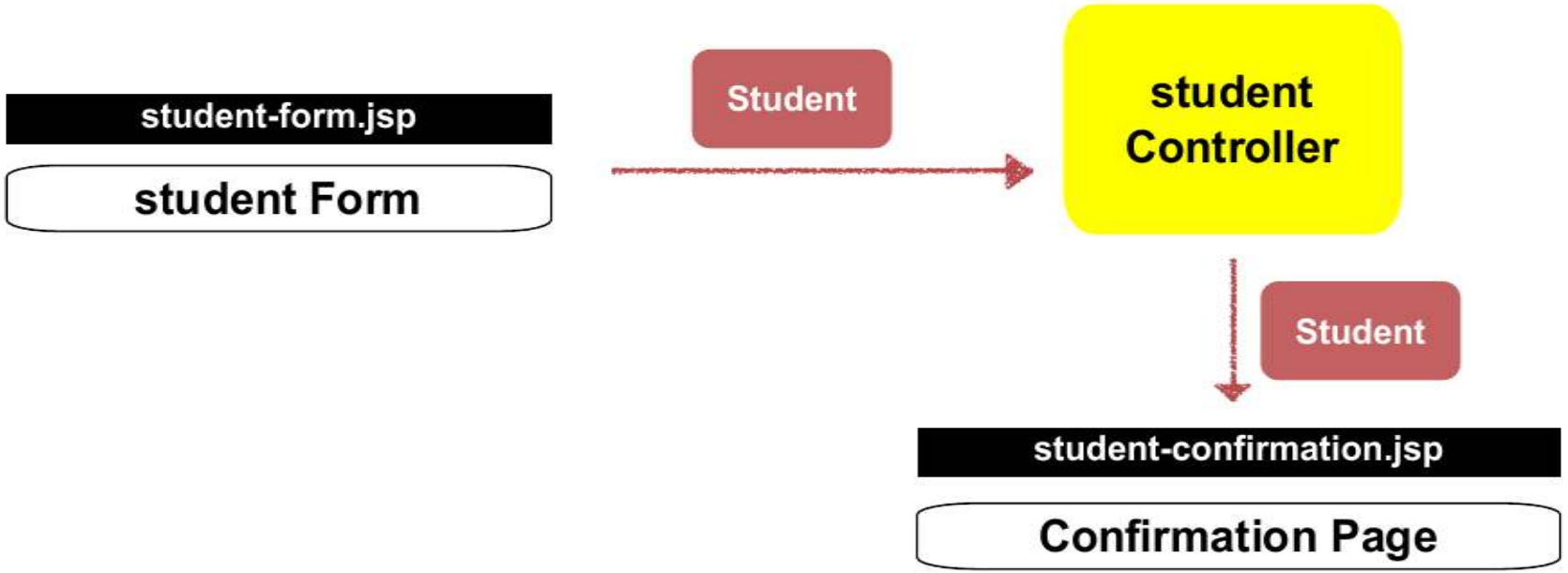
Code Snippet

File: /WEB-INF/view/....jsp

```
<form:select path="country">

   <form:option value="Brazil" label="Brazil" />
   <form:option value="France" label="France" />
   <form:option value="Germany" label="Germany" />
   <form:option value="India" label="India" />

</form:select>
```

Azzeddine RIGAT

Pulling It All Together



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Development Process

1.Update HTML form

2.Update Student class - add getter/setter for new property

3.Update confirmation page

Check **spring-mvc- demo8-form-tags-drop-down-lists**

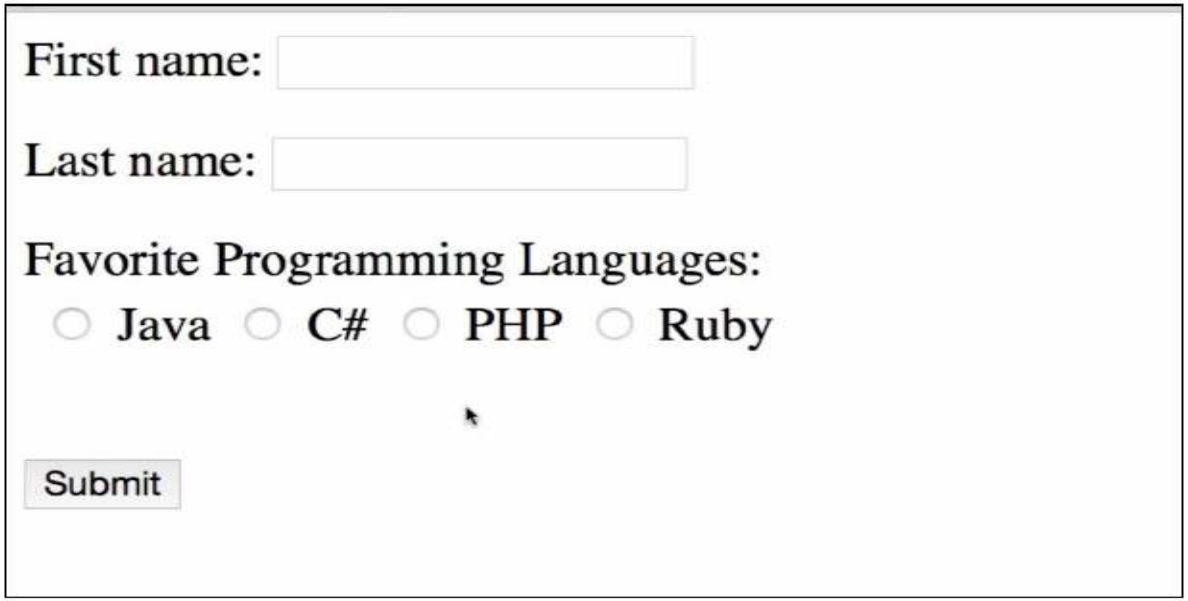
Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tag - Radio Buttons

Azzeddine
RIGAT

Radio Button Demo



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Tag

A Radio Button is represented by the tag

<form:radiobutton>

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

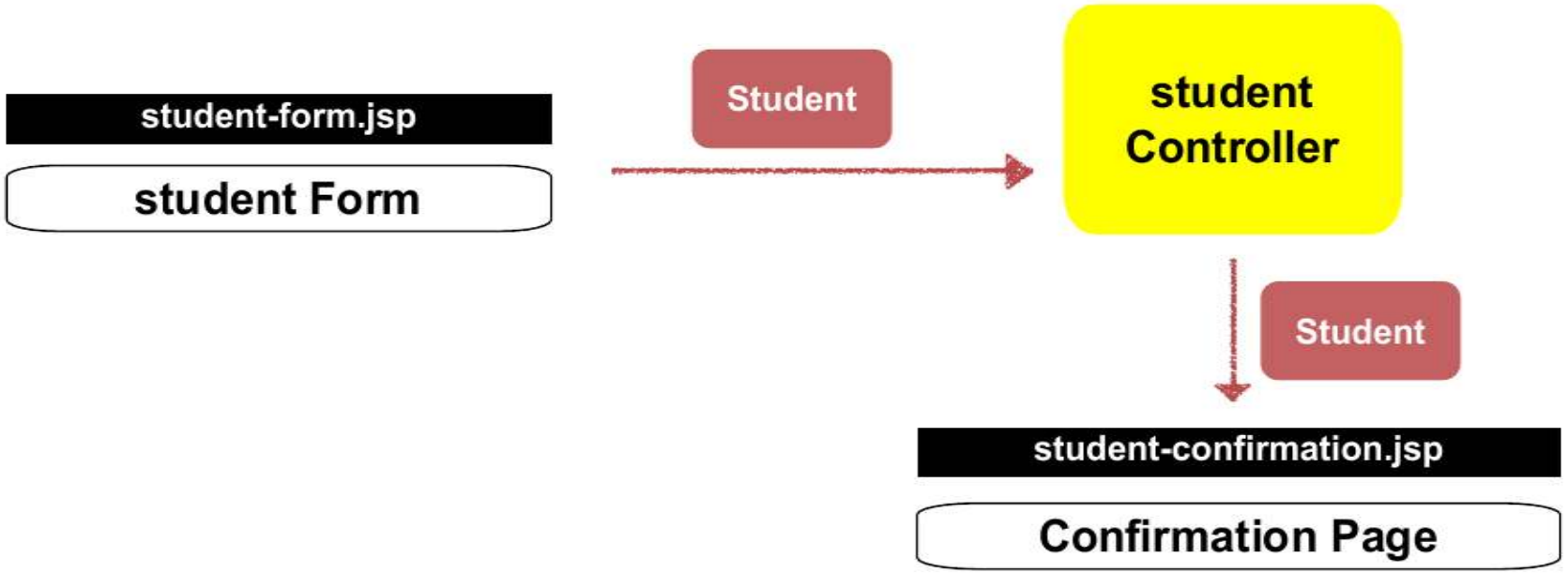
Code Example

File: /WEB-INF/view/....jsp

```
Java <form:radiobutton path="favoriteLanguage" value="Java" />
C# <form:radiobutton path="favoriteLanguage" value="C#" />
PHP <form:radiobutton path="favoriteLanguage" value="PHP" />
Ruby <form:radiobutton path="favoriteLanguage" value="Ruby" />
```

Azzeddine RIGAT

Pulling It All Together



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Development Process

1.Update HTML form

2.Update Student class - add getter/setter for new property

3.Update confirmation page

Check **spring-mvc- demo9-form-tags-radio-buttons**

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Assignment 11

Populate radiobuttons with items from Java class

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Form Tag - Check Box

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Check Box - Pick Your Favorite OS

Operating Systems: Linux ☐ Mac OS ☐ MS Windows ☐

Submit

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Spring MVC Tag

A Check Box is represented by the tag

<form:checkbox>

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

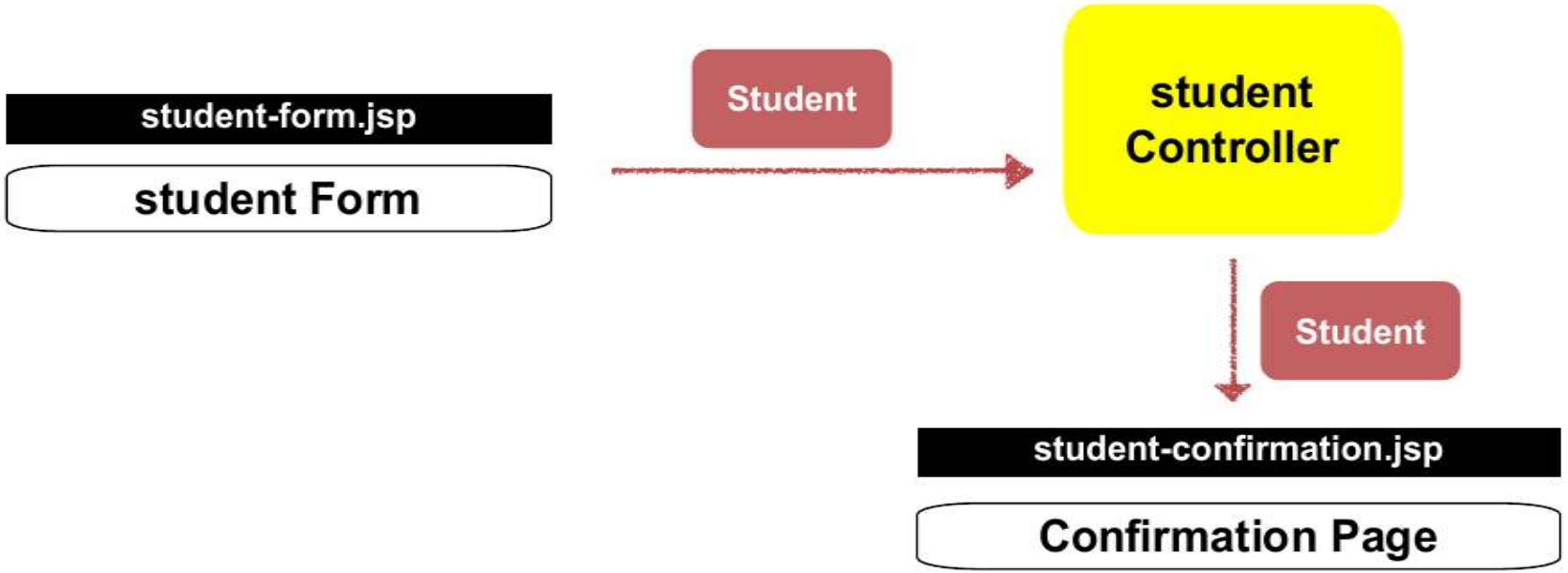
Code Example

File: /WEB-INF/view/....jsp

```
Linux <form:checkbox path="operatingSystems" value="Linux" />
Mac OS <form:checkbox path="operatingSystems" value="Mac OS" />
MS Windows <form:checkbox path="operatingSystems" value="MS Windows" />
```

Azzeddine RIGAT

Pulling It All Together



Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Development Process

1.Update HTML form

2.Update Student class - add getter/setter for new property

3.Update confirmation page

Topic 2, Spring MVC - Form Tags and Data Binding

Azzeddine
RIGAT

Assignment 12

Add checkbox to **spring-mvc-demo9-form-tags-radio-buttons**

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Spring MVC Form Validation

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

The Need for Validation

- Check the user input form for
- Required fields
- Valid numbers in a range
- Valid format (postal code)
- Custom business rule

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Java's Standard Bean Validation API

- Java has a standard Bean Validation API
- Defines a metadata model and API for entity validation
- Not tied to either the web tier or the persistence tier
- Available for server-side apps and also client-side JavaFX/Swing apps

http://www.beanvalidation.org

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Spring and Validation

- Spring version 4 and higher supports Bean Validation API
- Preferred method for validation when building Spring apps
- Simply add Validation JARs to our project

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Bean Validation Features

Validation Feature
required
validate length
validate numbers
validate with regular expressions
custom validation

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Validation Annotations

	Annotation Description
@NotNull	Checks that the annotated value is not null
@Min	Must be a number >= value
@Max	Must be a number <= value
@Size	Size must match the given size
@Pattern	Must match a regular expression pattern
@Future / @Past	Date must be in future or past of given date
others …	

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Our Road Map

1. set up our development environment
2. required field
3. validate number range: min, max
4. validate using regular expression (regexp)
5. Custom validation

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Setting Up Development Environment

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Java 's Standard Bean Validation API

- Java's standard Bean Validation API (JSR-303/309)
- Only a specification … vendor independent … portable
- BUT we still need an implementation …

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

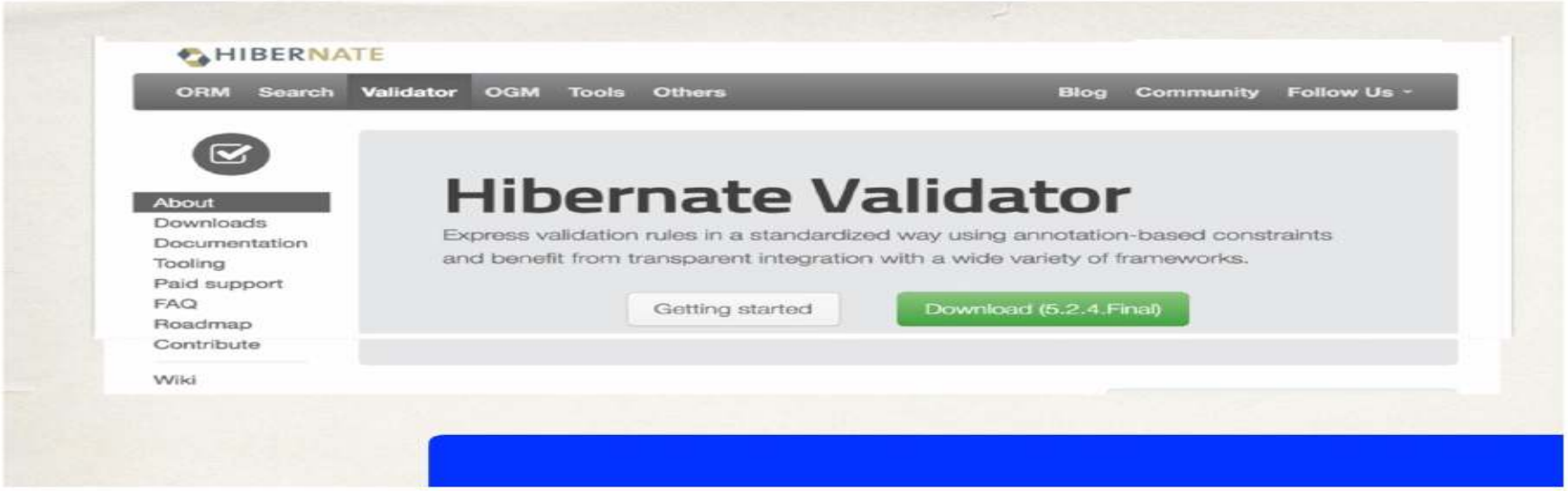
The Hibernate Team to the rescue!

- Hibernate started as an ORM project
- But in recent years, they have expanded into other areas
- They have a fully compliant JSR-303/309 implementation
- Not tied to ORM or database work … separate project

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

The Hibernate Team to the rescue!



http://www.hibernate.org

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Development Process

1. Download Validation JAR files from Hibernate website, I already uploaded them to the github account.

2. Add JAR files to project

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Spring MVC Form Validation Required Fields

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

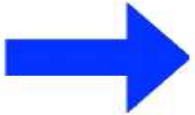
Required Fields

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

First name:

Last name (*):

Submit



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

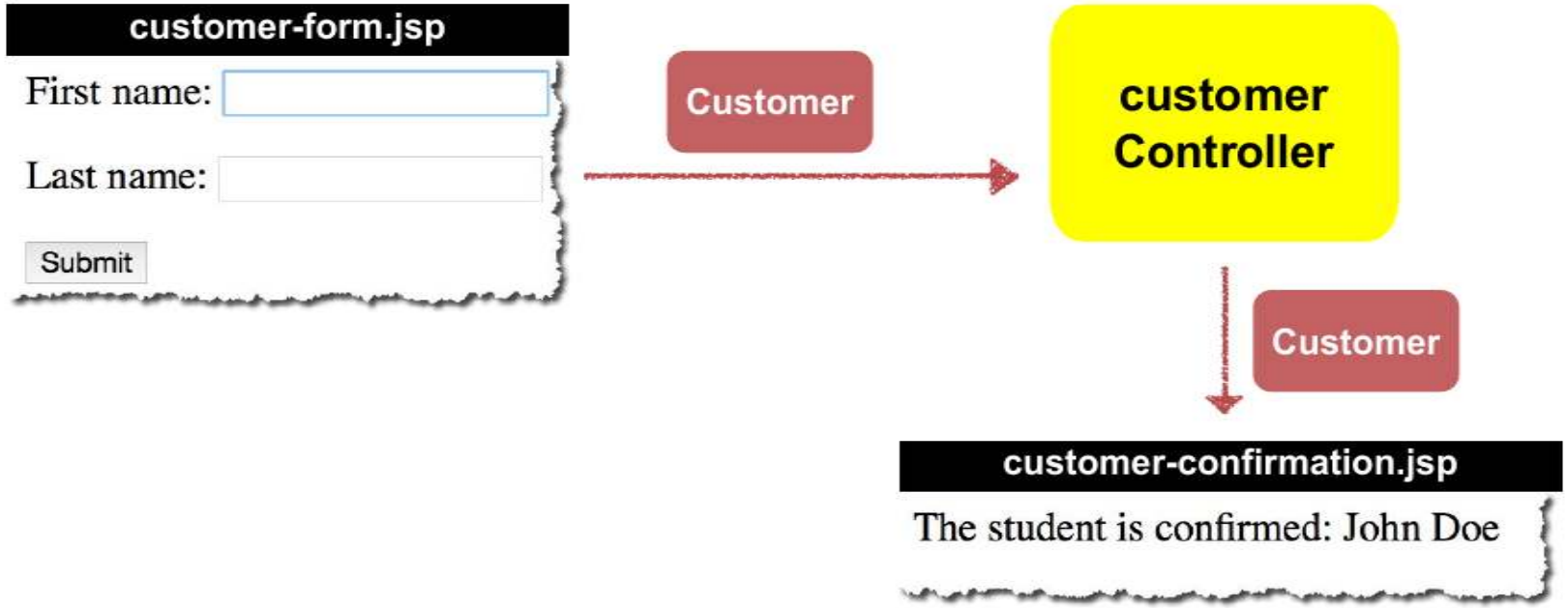
First name:

Last name (*): is required

Submit

Azzeddine RIGAT

Pulling It All Together



Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Development Process

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

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Step 1: Add validation rule to Customer class

```
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class Customer {

   private String firstName;

   @NotNull(message="is required")
   @Size(min=1, message="is required")
   private String lastName;

   // getter/setter methods
}
```

Azzeddine RIGAT

Step 2: Display error message on HTML form

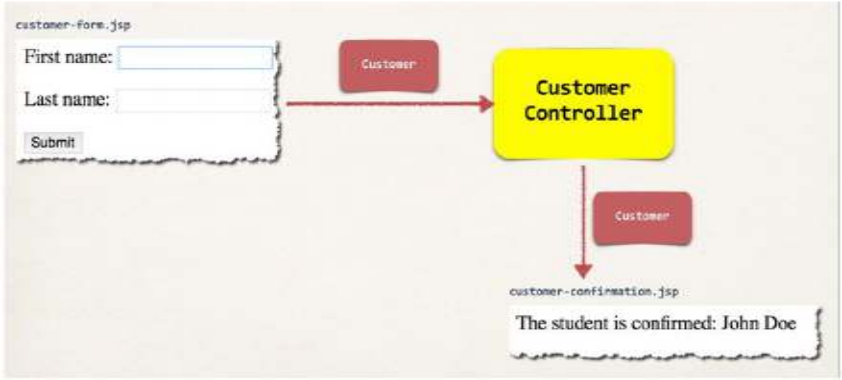
customer-form.jsp

```
<form:form action="processForm" modelAttribute="customer">

    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" />

</form:form>
```

Azzeddine RIGAT

Step 3: Perform validation in Controller class

```
@RequestMapping("/processForm")
public String processForm(@Valid @ModelAttribute("customer") Customer theCustomer,
BindingResult theBindingResult) {

    if (theBindingResult.hasErrors()) {
        return "customer-form";
    }
    else {
        return "customer-confirmation";
    }
}
```



Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Step 4: Update confirmation page

```
<html>

   <body>

      The customer is confirmed: ${customer.firstName} ${customer.lastName}

   </body>

</html>
```

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Assignment 13

Make an implementation of the Built-in validation rules using the sides code

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Spring MVC Validation @InitBinder

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

White Space

- After your implementation you will figure out that our previous example had a problem with white space
  - Last name field with all whitespace **passed** … YIKES!
  - Should have **failed**!
- We need to trim whitespace from input fields

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

@InitBinder

- **@InitBinder** annotation works as a pre-processor
- It will pre-process each web request to our controller
- Method annotated with **@InitBinder** is executed

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

@InitBinder

- We will use this to trim Strings
  - Remove leading and trailing whitespace
- If String only has white spaces … trim it to **null**
- Will resolve our validation problem … whew :-)

Topic 2, Spring MVC Form Validation - Applying Built-In Validation Rules

Azzeddine RIGAT

Register Custom Editor in Controller

CustomerController.java

```
...

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

...
```

Check **spring-mvc- demo10-form-initBinder**

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

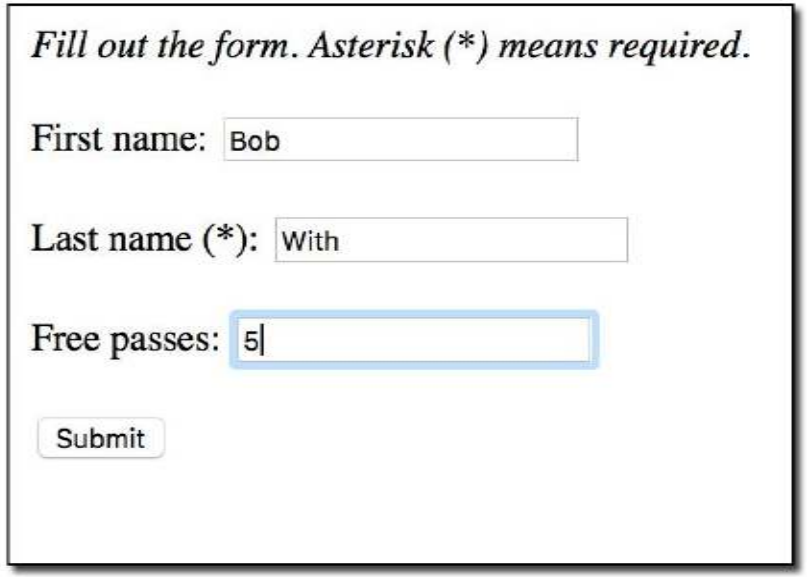
Spring MVC Validation Number Range: @Min and @Max

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Validate a Number Range

- Add a new input field on our form for: Free Passes
- User can only enter a range: 0 to 10



Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Development Process

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

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Step 1: Add validation rule to Customer class

```
import javax.validation.constraints.Min;
import javax.validation.constraints.Max;
public class Customer {

   @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;

   // getter/setter methods

}
```

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Assignment 14

Make an implementation of the number ranges validation rules using the sides code

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Spring MVC Validation Regular Expressions

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Regular Expressions

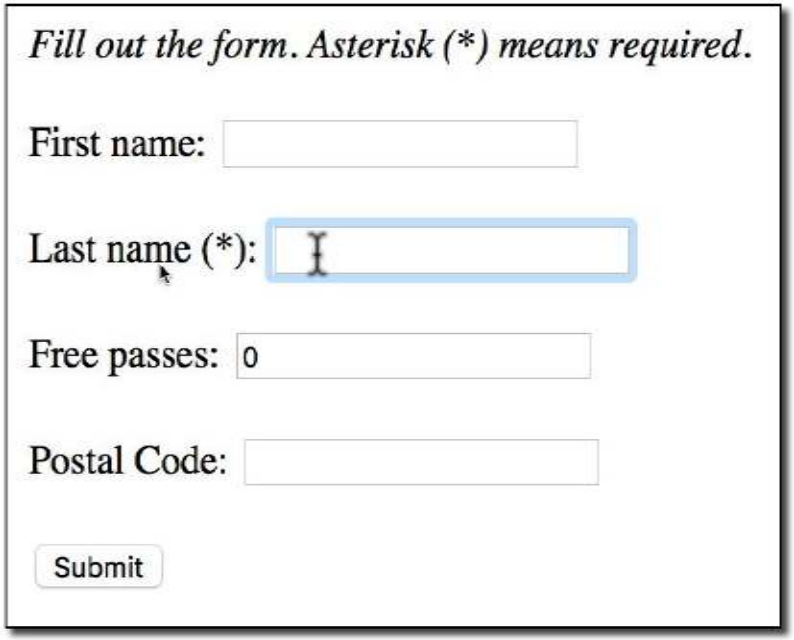
- A sequence of characters that define a search pattern
  - This pattern is used to find or match strings
- Regular Expressions is like its own language
  - I will assume you already know about regular expressions
- If not, then plenty of free tutorials available
  - **https://docs.oracle.com/javase/tutorial/essential/regex/**

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Validate a Postal Code

- Add a new input field on our form for: **Postal Code**
- User can only enter 5 chars / digits
- Apply **Regular Expression**



Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Development Process

1. Add validation rule to Customer class
2. Display error messages on HTML form
3. Update confirmation page

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Step 1: Add validation rule to Customer class

```
import javax.validation.constraints.Pattern;
public class Customer {

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

   // getter/setter methods

}
```

Check **spring-mvc- demo11-regexp**

Topic 2, Spring MVC Form Validation - Validating Number Ranges and Regular Expressions

Azzeddine RIGAT

Assignment 15

Use @pattern to reproduce the same result of **spring-mvc- demo10-initBinder**

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Spring MVC Validation: Custom Validation

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Custom Validation Demo

First name:

Last name:

Course Code:

Submit

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Custom Validation

- Perform custom validation based on your business rules
  - Our example:
    - Course Code must start with “**EDU**”
- Spring MVC calls our custom validation
- Custom validation returns boolean value for pass/fail (true/false)

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Create a custom Java Annotation … from scratch

- So far, we've used predefined validation rules: @Min, @Max, …
- For custom validation … we will create a **Custom Java Annotation**
  - **@CourseCode**

Java class

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

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Development Process

1. Create custom validation rule
2. Add validation rule to Customer class
3. Display error messages on HTML form
4. Update confirmation page

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Development Process - Drill Down

1. Create custom validation rule
    a. Create **@CourseCode** annotation
    b. Create **CourseCodeConstraintValidator**

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Step 1a: Create @CourseCode annotation

Usage example:

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

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Step 1a: Create @CourseCode annotation

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

    ....
}
```

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Step 1a: Create @CourseCode annotation

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

	// define default course code
	public String value() default "EDU";
	// define default error message
	public String message() default "must start with EDU";
	// define default groups
	public Class<?>[] groups() default {};
	// define default payloads
	public Class<? extends Payload>[] payload() default {};
}
```

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Step 1b: Create CourseCodeConstraintValidator

```
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

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;
   }
}
```

Topic 2, Spring MVC Form Validation - Creating Custom Validation Rules

Azzeddine RIGAT

Assignment 16

Make an implementation of custom validation rules using the sides code

Topic 2, Spring-MVC

Azzeddine
RIGAT

Done with the one of the most useful **Spring modules**, Now we are ready to work with **Hibernate module**.