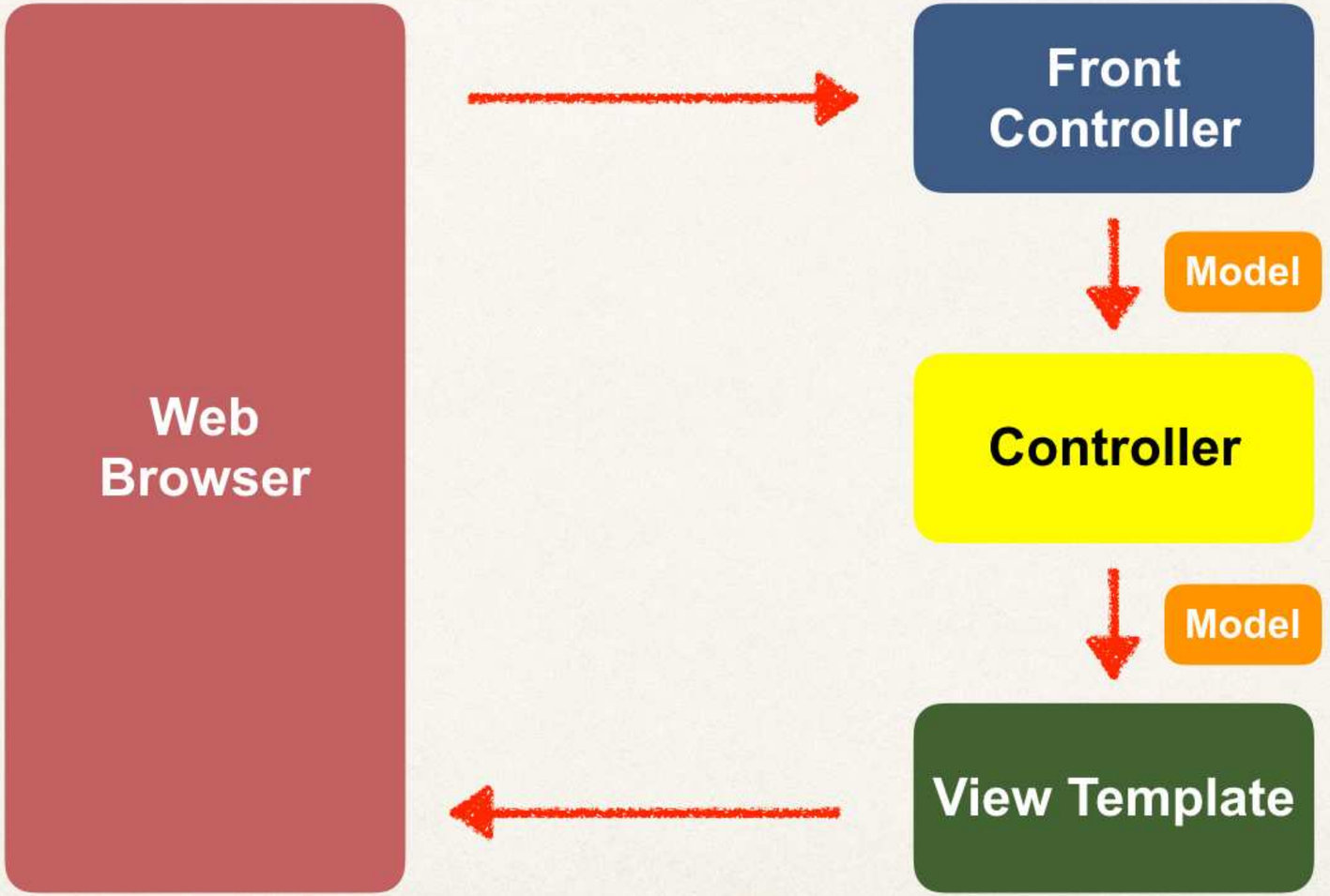
luv2code

Spring MVC
Behind the Scenes

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)

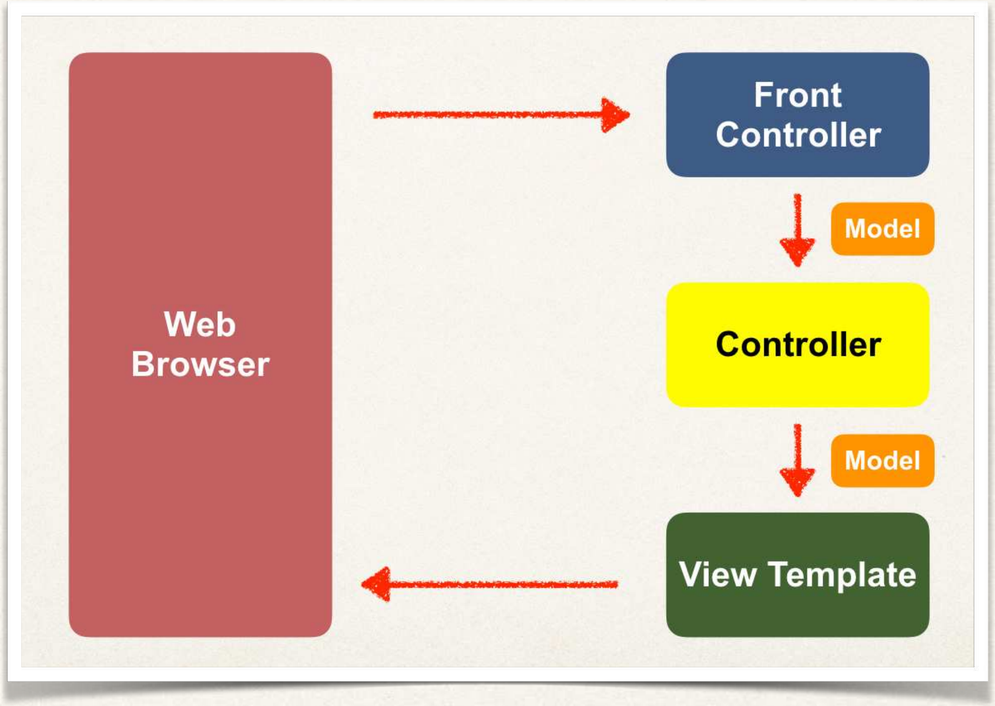
How Spring MVC Works Behind the Scenes



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)

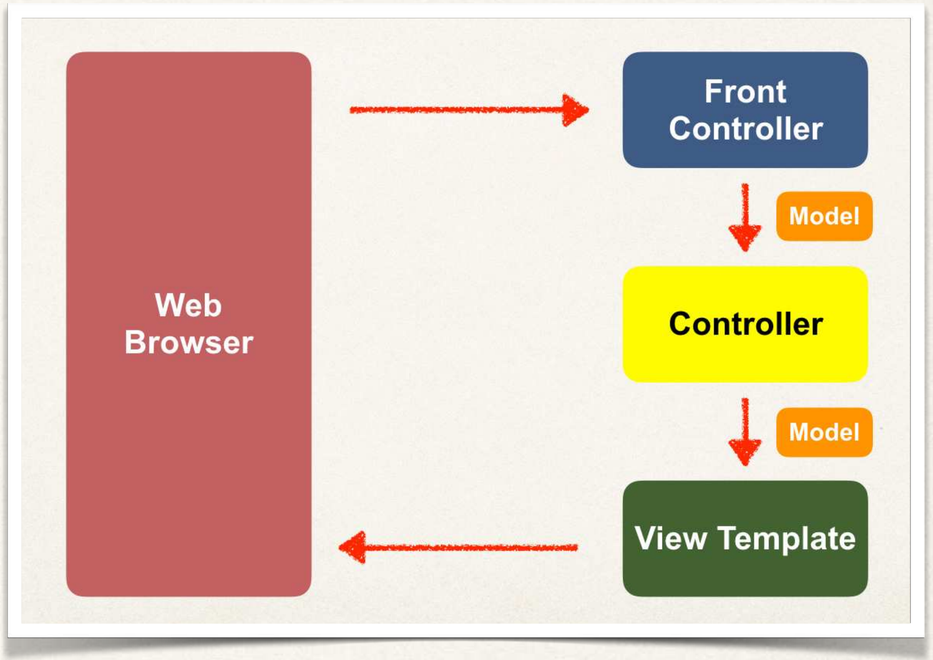


Controller

- 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

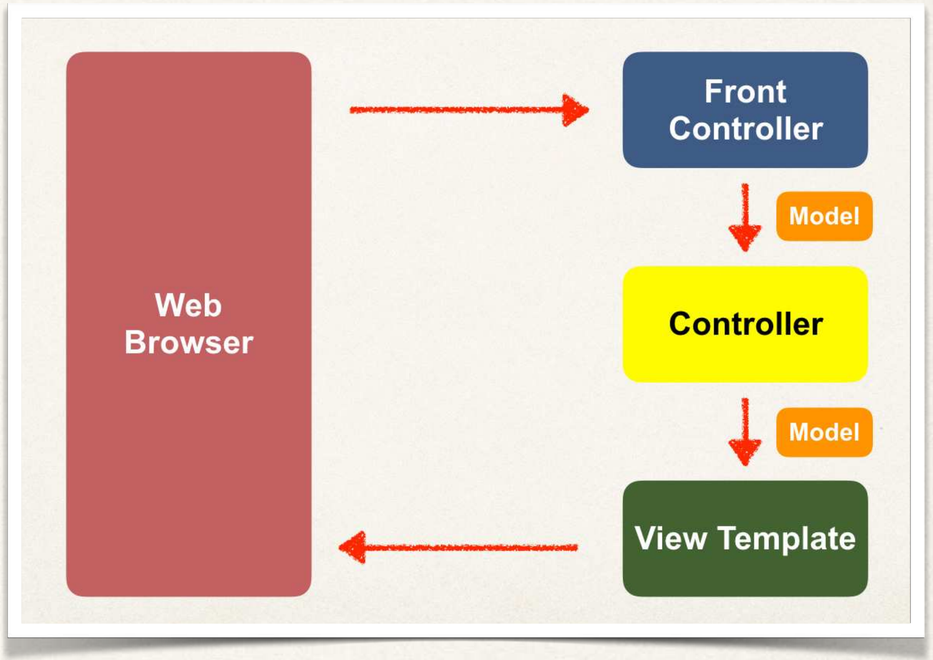


Model

- Model: contains your data

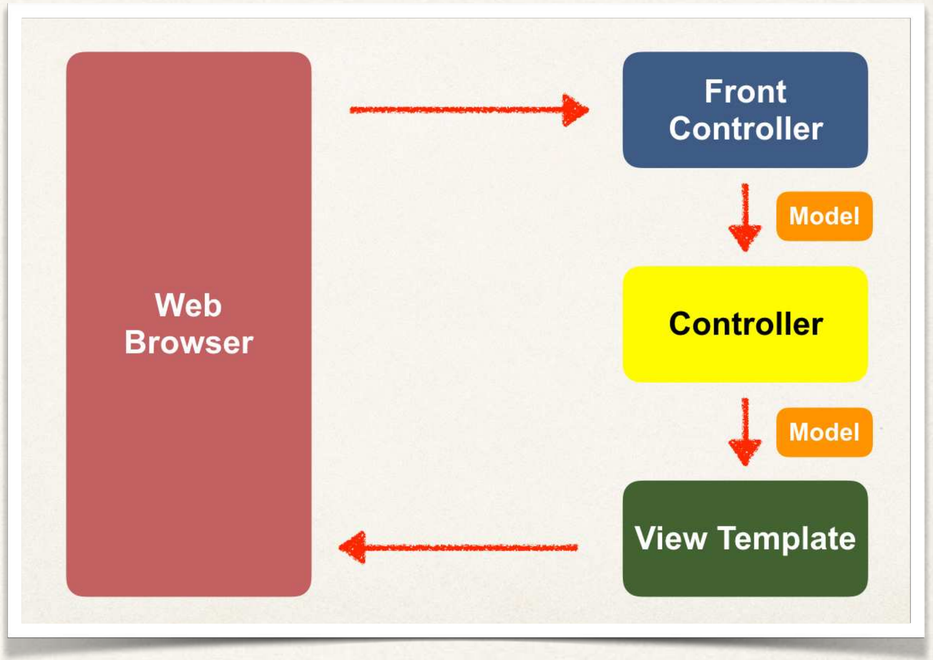
- Store/retrieve data via backend systems
  - database, web service, etc…
  - Use a Spring bean if you like

- Place your data in the model
  - Data can be any Java object/collection



View Template

- Spring MVC is flexible
  - Supports many view templates
- Recommended: Thymeleaf
- Developer creates a page
  - Displays data



View Template (more)

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

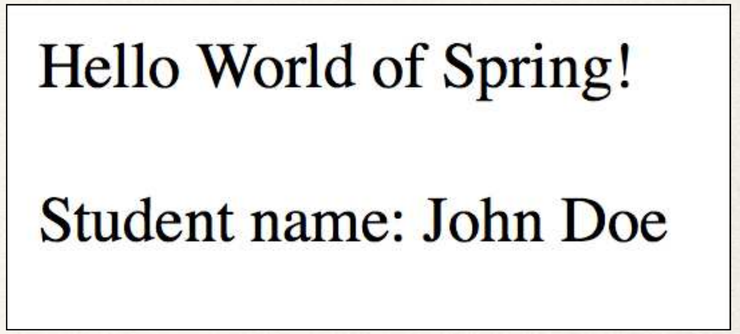
www.luv2code.com/spring-mvc-views

Reading Form Data with Spring MVC

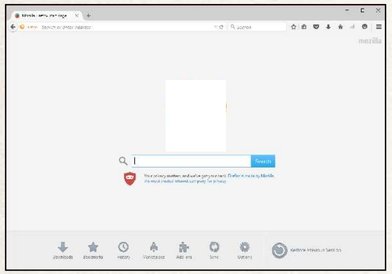
High Level View

helloworld-form.html

helloworld.html



Application Flow



Request Mapping

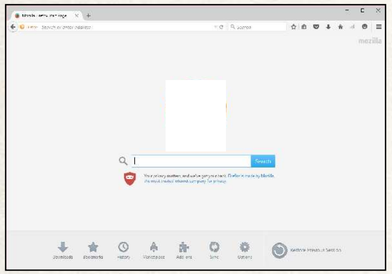
/showForm

HelloWorld Controller

helloworld-form.html

What's your name? Submit Query

Application Flow



Request Mapping

/showForm

HelloWorld Controller

helloworld-form.html

What's your name? Submit Query

helloworld-form.html

What's your name? Submit Query

Request Mapping

/processForm

HelloWorld Controller

helloworld.html

Hello World of Spring!

Student name: John Doe

Controller Class

```
@Controller
public class HelloWorldController {

   // need a controller method to show the initial HTML form

   @RequestMapping("/showForm")
   public String showForm() {
      return "helloworld-form";
   }
```

Any method name

src/main/resources/templates/helloworld-form.html

```
   // need a controller method to process the HTML form

   @RequestMapping("/processForm")
   public String processForm() {
      return "helloworld";
   }
}
```

Any method name

Development Process

Step-By-Step

1. **Create Controller class**

2. **Show HTML form**

    a. Create controller method to show HTML Form

    b. Create View Page for HTML form

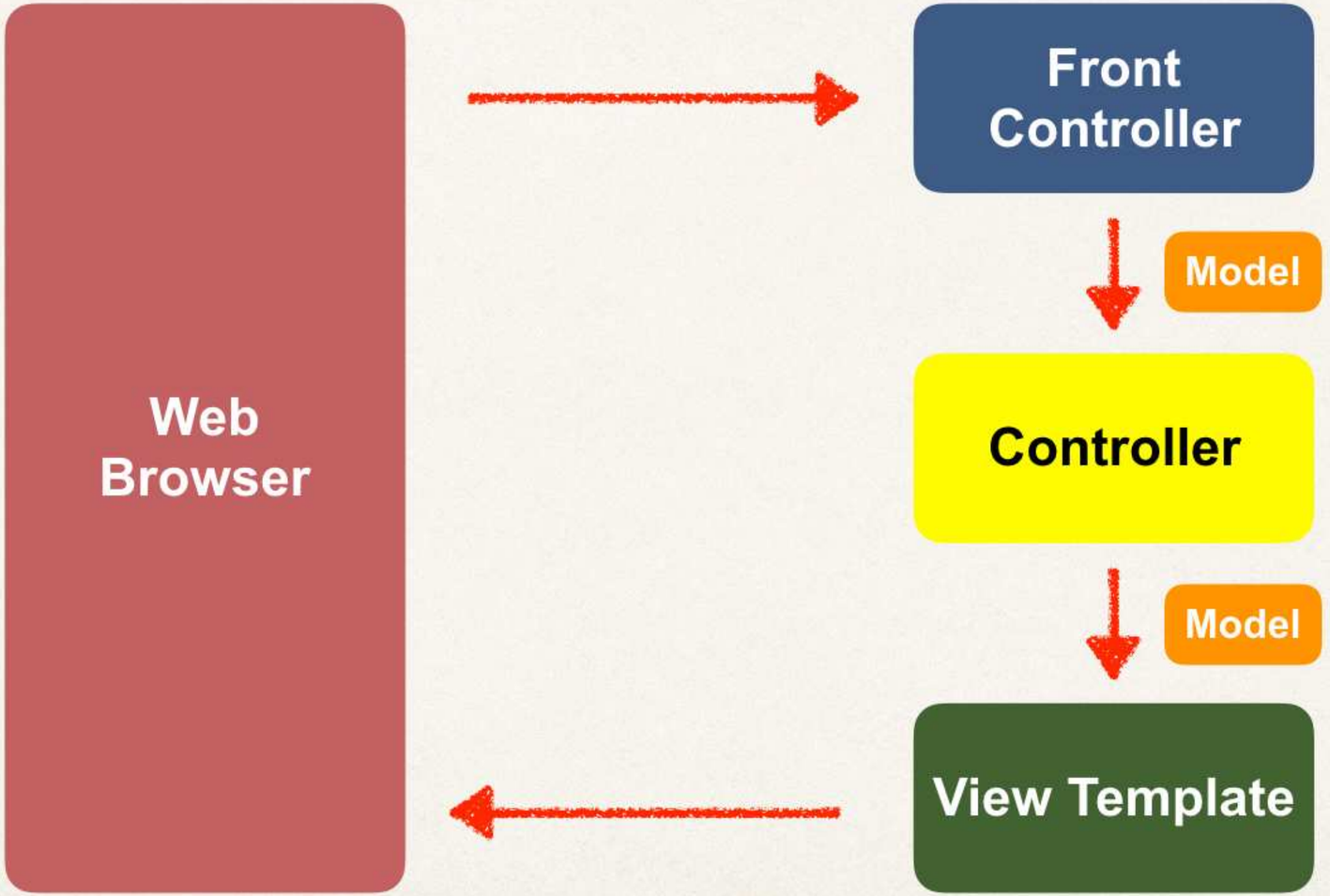
3. **Process HTML Form**

    a. Create controller method to process HTML Form

    b. Develop View Page for Confirmation

Adding Data to Spring Model

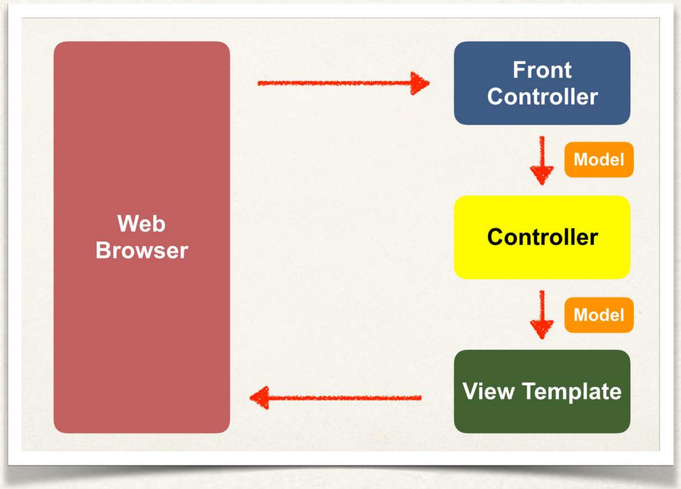
Focus on the Model



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 can access data from the **model**

Code Example

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

Passing Model to your Controller

Container for your data

```
@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";
}
```

Holds HTML form data

You can give any attribute name

Name

Value

View Template - Thymeleaf

```
<html><body>

Hello World of Spring!

...

The message: <span th:text="${message}" />

</body></html>
```

Attribute name

```
// add message to the model
model.addAttribute("message", result);
```

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);
```

Reading HTML Form Data with @RequestParam Annotation

Code Example

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

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");

    ...
}
```

Bind variable using @RequestParam Annotation

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

    // now we can use the variable: theName

}
```

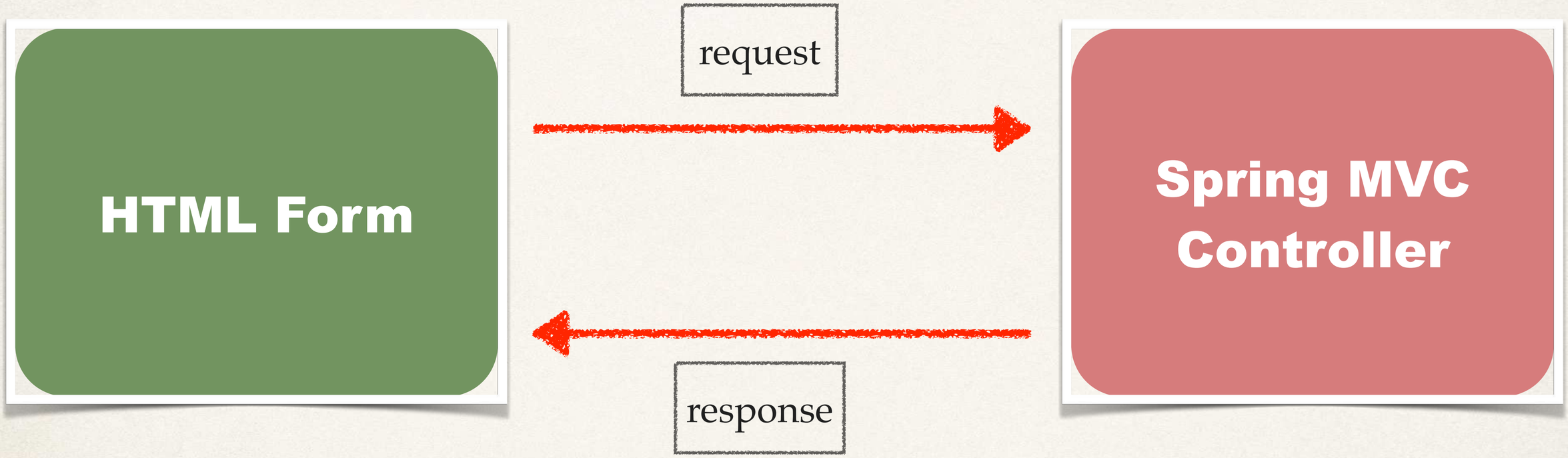
Behind the scenes:

Spring will read param from request: studentName

Bind it to the variable: theName

@GetMapping and @PostMapping

HTTP Request / Response



Most Commonly Used HTTP Methods

Method	Description
GET	Requests data from given resource
POST	Submits data to given resource
others	…

Sending Data with GET method

```
<form th:action="@{/processForm}" method="GET" ...>
  ...
</form>
```

- Form data is added to end of URL as name/value pairs
  - **theUrl?field1=value1&field2=value2...**

Handling Form Submission

```
@RequestMapping("/processForm")
public String processForm(...) {
   ...
}
```

- This mapping handles **ALL HTTP** methods
  - GET, POST, etc …

Constrain the Request Mapping - GET

```
@RequestMapping(path="/processForm", method=RequestMethod.GET)
public String processForm(...) {
   ...
}
```

- This mapping **ONLY** handles **GET** method
- Any other HTTP REQUEST method will get rejected

Annotation Short-Cut

```
@GetMapping("/processForm")
public String processForm(...) {
    ...
}
```

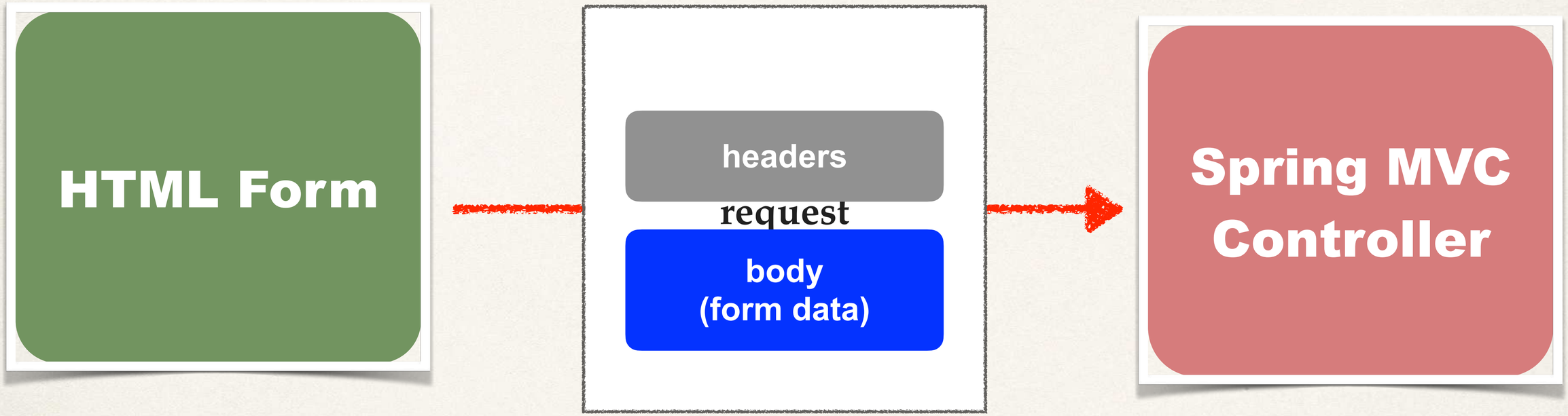
- @GetMapping: this mapping **ONLY** handles **GET** method
- Any other HTTP REQUEST method will get rejected

Sending Data with POST method

```
<form th:action="@{/processForm}" method="POST" ...>
   ...
</form>
```

- Form data is passed in the body of HTTP request message

Sending Data with POST method



Constrain the Request Mapping - POST

```
@RequestMapping(path="/processForm", method=RequestMethod.POST)
public String processForm(...) {
   ...
}
```

- This mapping **ONLY** handles **POST** method
- Any other HTTP REQUEST method will get rejected

Annotation Short-Cut

```
@PostMapping("/processForm")
public String processForm(...) {
    ...
}
```

- **@PostMapping:** This mapping **ONLY** handles **POST** method
- Any other HTTP REQUEST method will get rejected

Well …. which one???

GET

- Good for debugging
- Bookmark or email URL
- Limitations on data length

POST

- Can't bookmark or email URL
- No limitations on data length
- Can also send binary data

Spring MVC Form Tag - Text Field

Review HTML Forms

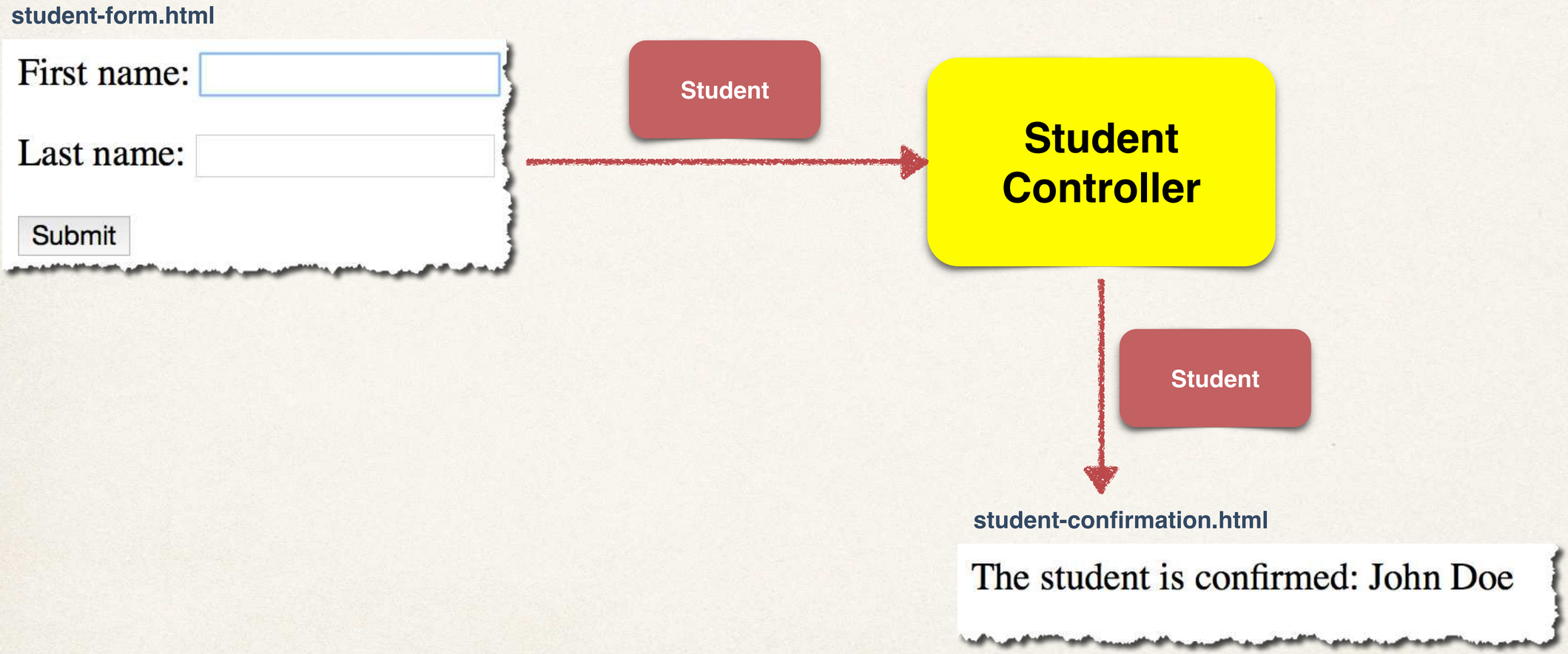
- HTML Forms are used to get input from the user

Data Binding

- Spring MVC forms can make use of data binding

- Automatically setting / retrieving data from a Java object / bean

Big Picture



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***

Show Form - Add Model Attribute

Code snippet from Controller

```
@GetMapping("/showStudentForm")
public String showForm(Model theModel) {

    theModel.addAttribute("student", new Student());

    return "student-form";
}
```

Setting up HTML Form - Data Binding

```
<form th:action="@{/processStudentForm}" th:object="${student}" method="POST">

   First name: <input type="text"  th:field="*{firstName}" />

   <br><br>

   Last name: <input type="text"  th:field="*{lastName}" />

   <br><br>

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

</form>
```

Setting up HTML Form - Data Binding

Name of model attribute

```
<form th:action="@{/processStudentForm}" th:object="${student}" method="POST">

   First name: <input type="text"  th:field="*{firstName}" />

   <br><br>

   Last name: <input type="text"  th:field="*{lastN

   <br><br>

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

</form>
```

```
@GetMapping("/showStudentForm")
public String showForm(Model theModel) {

   theModel.addAttribute("student", new Student());

   return "student-form";
}
```

Setting up HTML Form - Data Binding

```
<form th:action="@{/processStudentForm}" th:object="${student}" method="POST">

    First name: <input type="text"  th:field="*{firstName}" />

    <br><br>

    Last name: <input type="text"  th:field="*{lastName}" />

    <br><br>

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

</form>
```

*{ … } is shortcut syntax for: ${student.firstName}

*{ … } is shortcut syntax for: ${student.lastName}

When Form is Loaded … fields are populated

```
<form th:action="@{/processStudentForm}" th:object="${student}" method="POST">

    First name: <input type="text"  th:field="*{firstName}" />

    <br><br>

    Last name: <input type="text"  th:field="*{lastName}" />

    <br><br>

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

</form>
```

When form is **loaded**,
Spring MVC will read student
from the model,
then call:

student.getFirstName()

…

student.getLastName()

When Form is submitted … calls setter methods

```
<form th:action="@{/processStudentForm}" th:object="${student}" method="POST">

    First name: <input type="text"  th:field="*{firstName}" />

    <br><br>

    Last name: <input type="text"  th:field="*{lastName}" />

    <br><br>

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

</form>
```

When form is **submitted**,
Spring MVC will
create a **new** Student instance
and add to the model,
then call:

student.setFirstName(…)
…
student.setLastName(…)

Handling Form Submission in the Controller

Code snippet from Controller

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

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

    return "student-confirmation";
}
```

Confirmation page

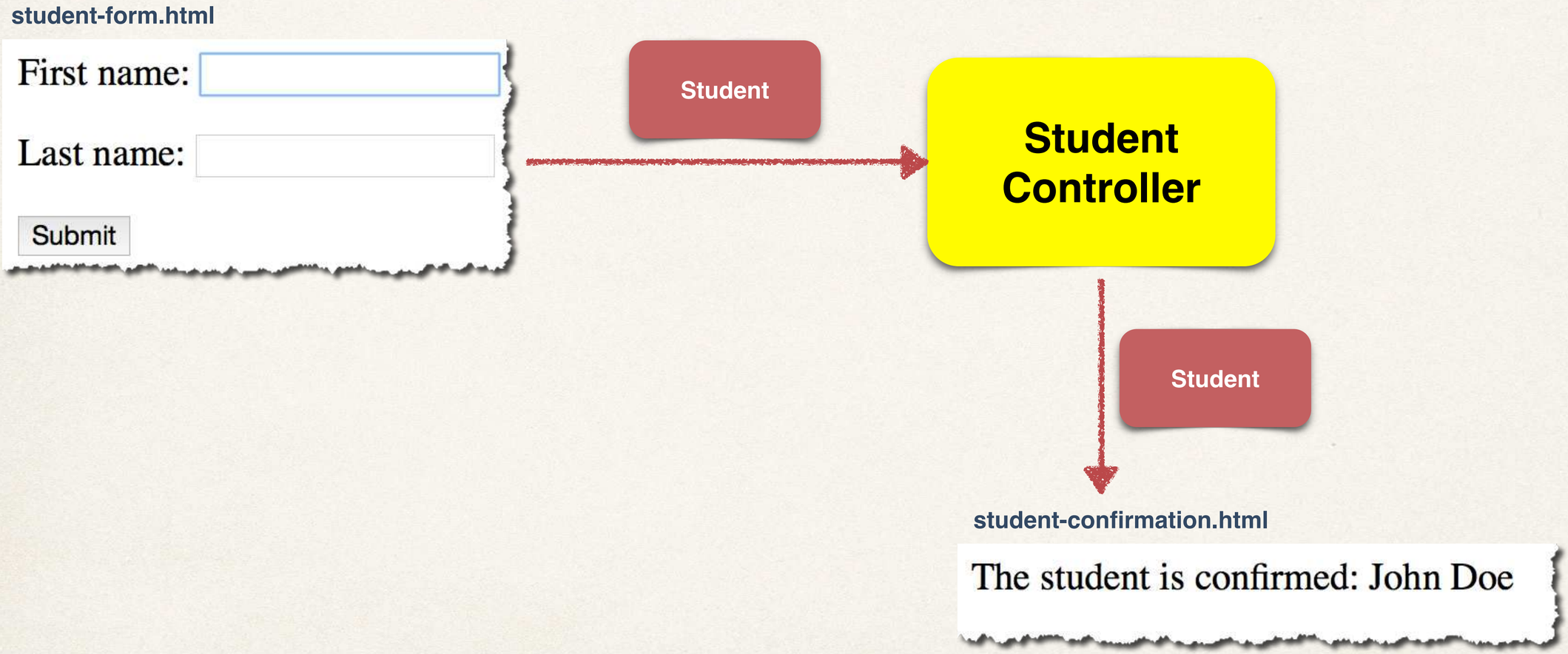
```
<html>

<body>

   The student is confirmed: <span th:text="${student.firstName} + ' ' + ${student.lastName}" />

</body>

</html>
```

Calls student.getFirstName()

Calls student.getLastName()

The student is confirmed: John Doe

Pulling It All Together



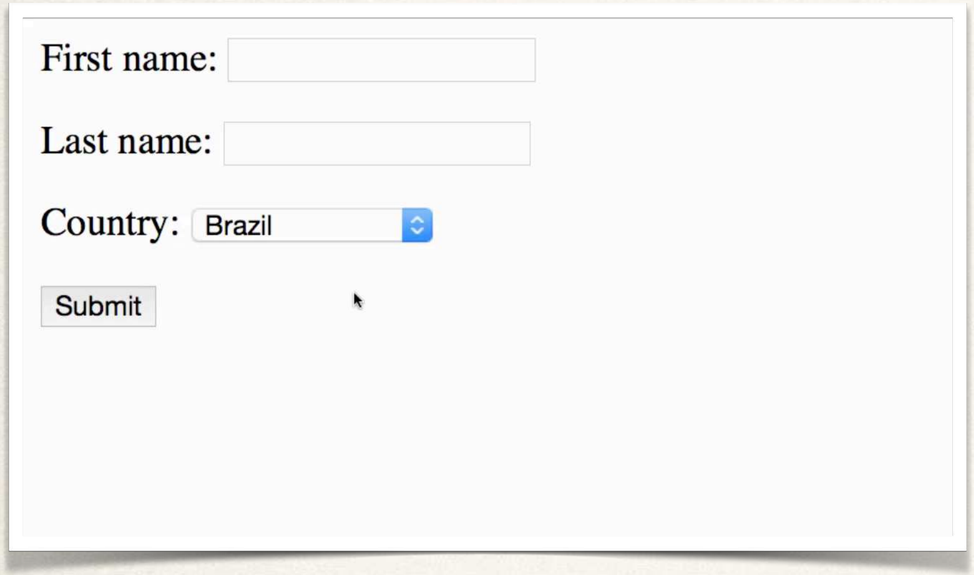
Development Process

Step-By-Step

1. Create Student class
2. Create Student controller class
3. Create HTML form
4. Create form processing code
5. Create confirmation page

Spring MVC Form - Drop Down List

Review of HTML <select> Tag




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

BR
FR
DE
IN

Value sent during form submission

Displayed to user

Thymeleaf and <select> tag

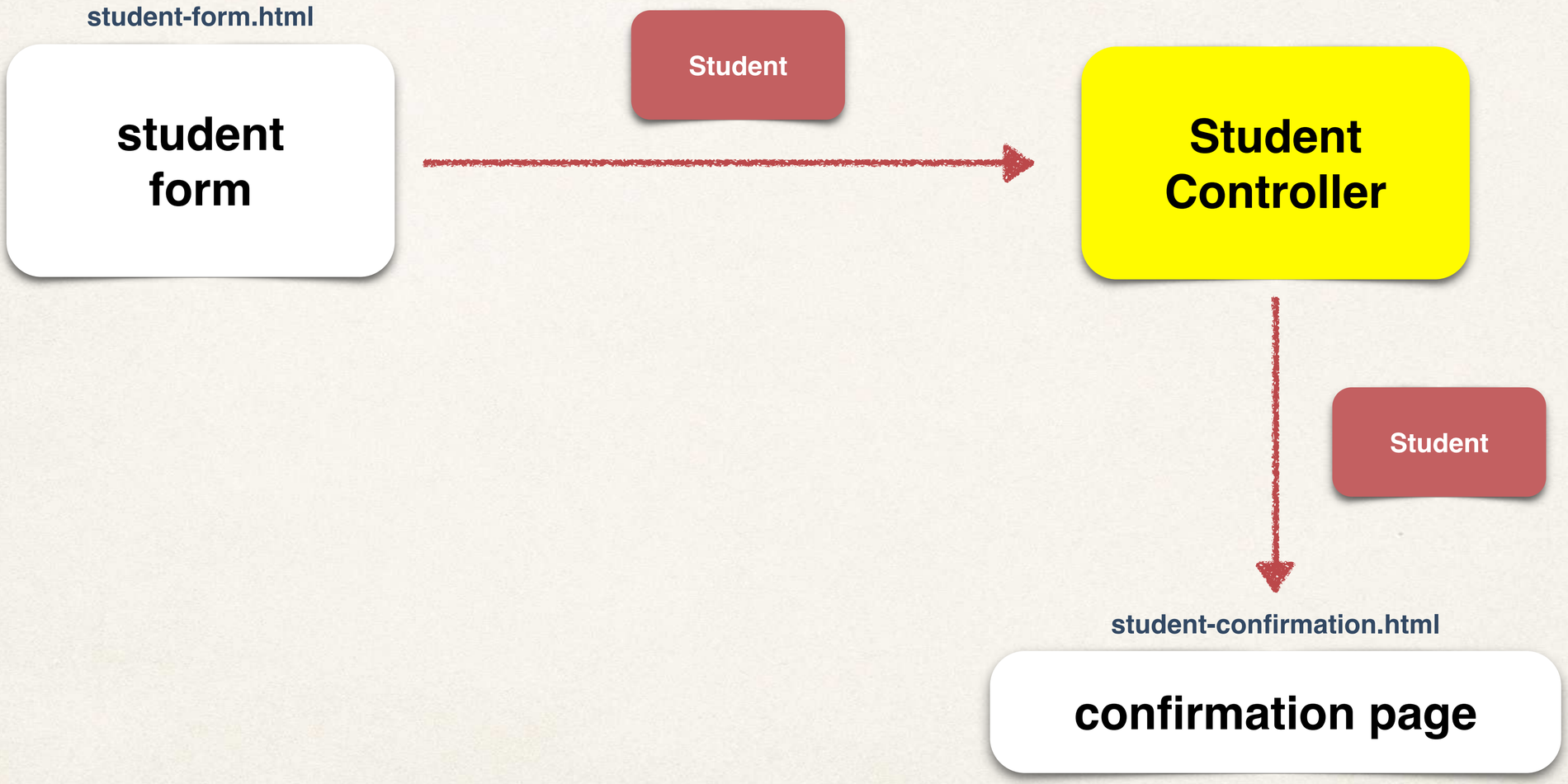
```
<select th:field="*{country}">

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

</select>
```

Name of property on our Student class

Value sent during form submission

Displayed to user

Pulling It All Together



Development Process

Step-By-Step

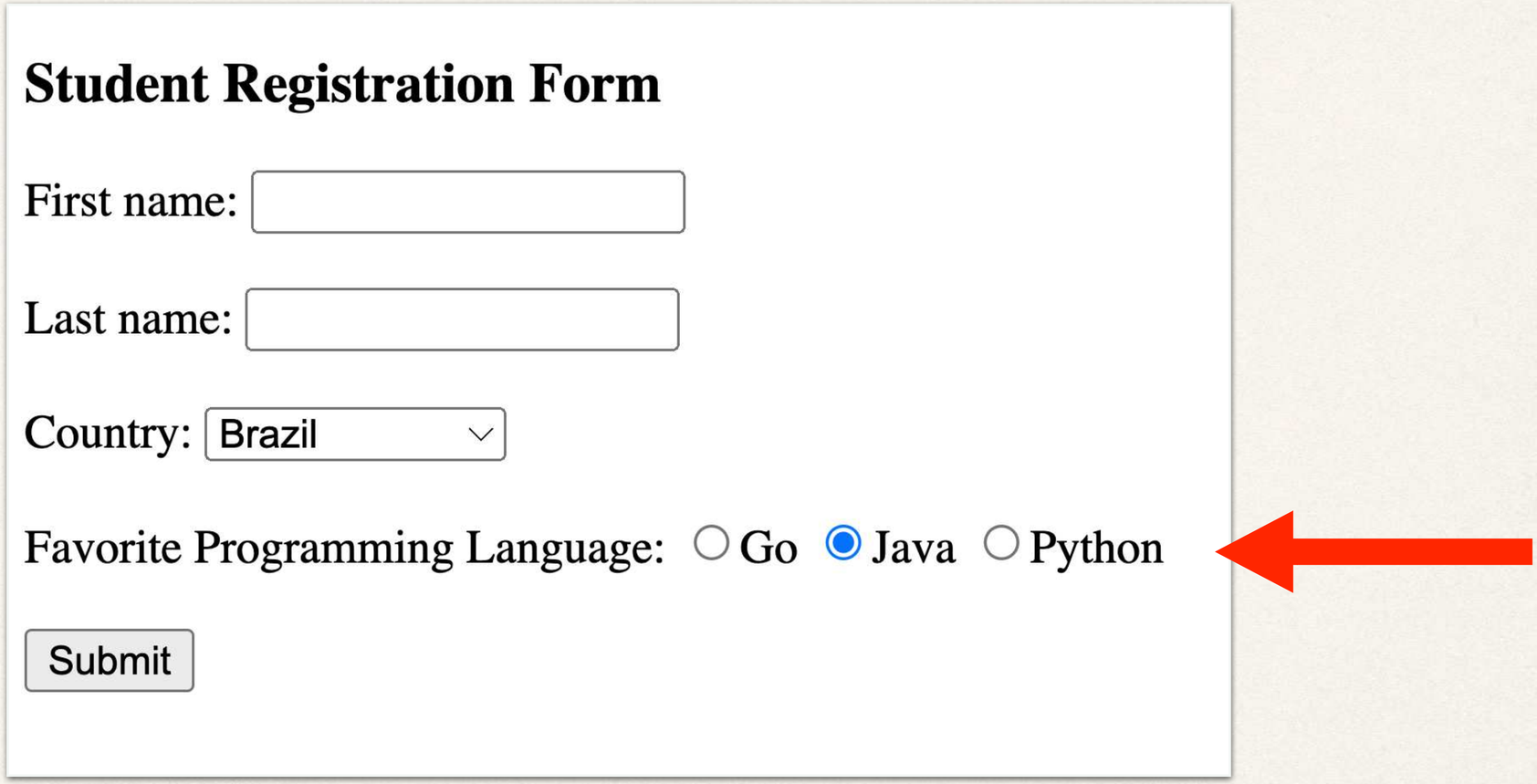
1. Update HTML form

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

3. Update confirmation page

Spring MVC Form - Radio Buttons

Radio Buttons



Code Example

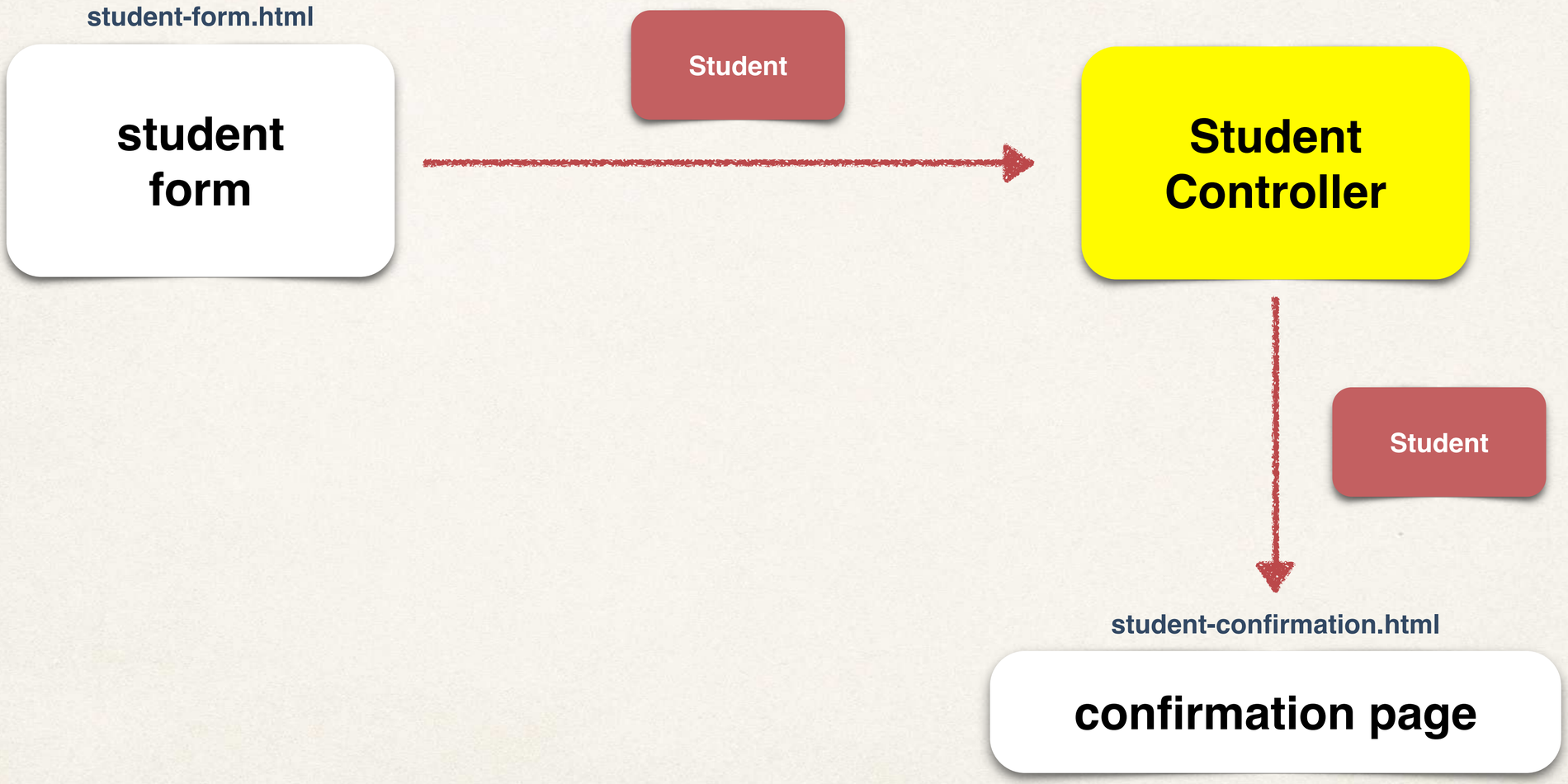

```
Favorite Programming Language:

<input type="radio" th:field="*{favoriteLanguage}" th:value="Go">Go</input>
<input type="radio" th:field="*{favoriteLanguage}" th:value="Java">Java</input>
<input type="radio" th:field="*{favoriteLanguage}" th:value="Python">Python</input>
```

Binding to property on Student object

Value sent during form submission

Displayed to user

Pulling It All Together



Development Process

Step-By-Step

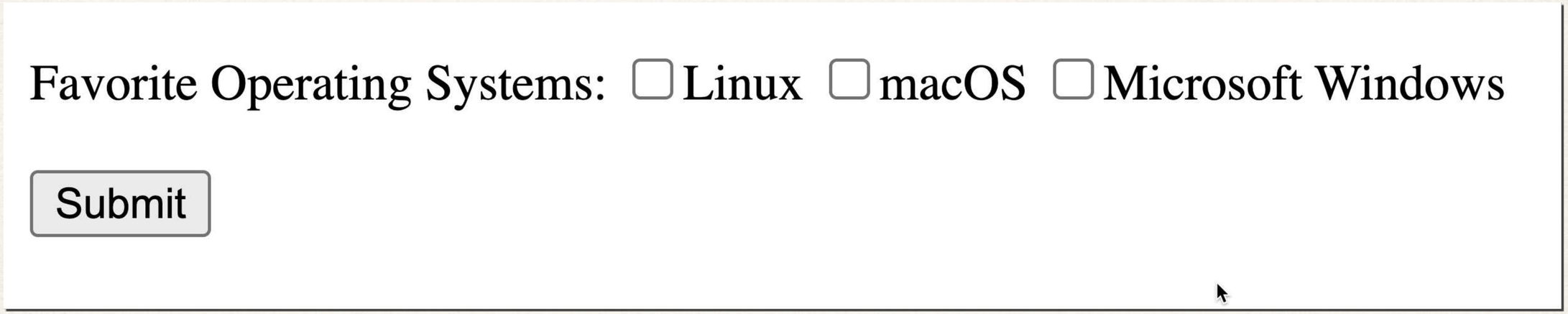
1. Update HTML form

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

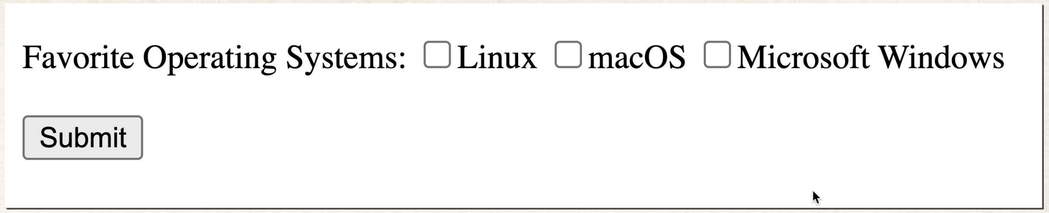
3. Update confirmation page

Spring MVC Forms - Check Box

Check Box - Pick Your Favorite Operating Systems



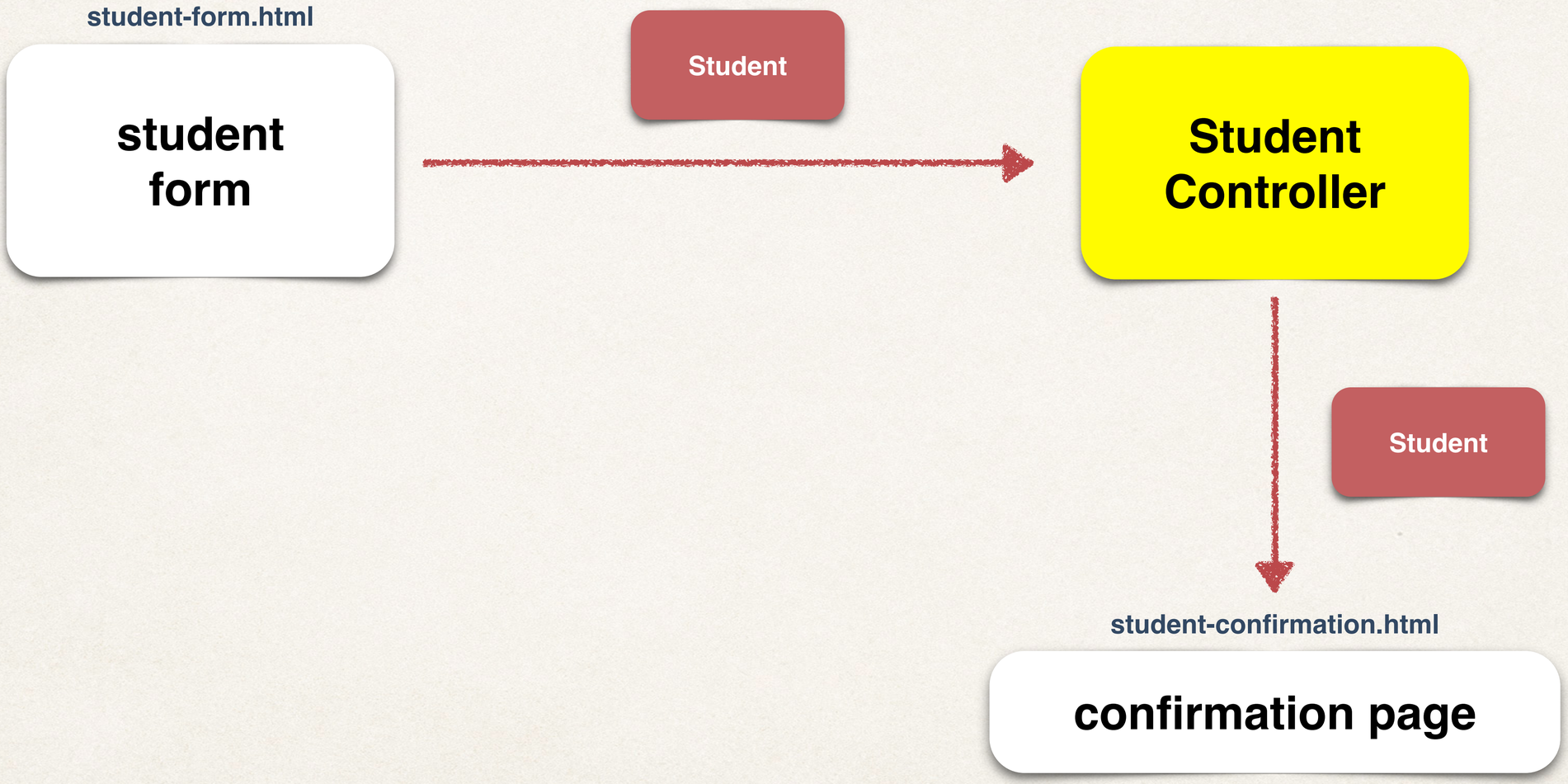
Code Example




```
<input type="checkbox" th:field="*{favoriteSystems}" th:value="Linux">Linux</input>
<input type="checkbox" th:field="*{favoriteSystems}" th:value="macOS">macOS</input>
<input type="checkbox" th:field="*{favoriteSystems}"
                       th:value="'Microsoft Windows'">Microsoft Windows</input>
```

If a value has spaces, place it in single-quotes

Pulling It All Together



Development Process

Step-By-Step

1. Update HTML form

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

3. Update confirmation page