Module 3-6

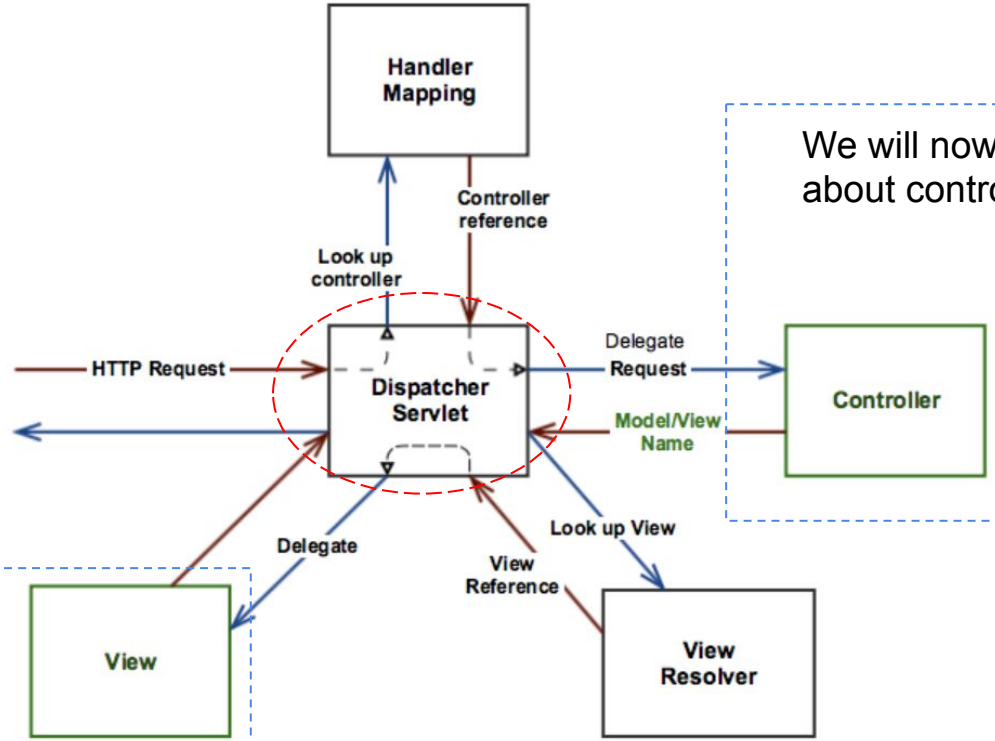
Forms, Controllers, and HTTP get

Making a HTTP Request

- An interaction with a resource on a network through HTTP is composed of a request and a response.
  - Both requests and responses can data in a header and/or body (payload).
  - Responses back to the sender will also include a status code.
- The goal for this module is to create applications capable of handling a “Get Request”

Review Spring Architecture (with more detail)

At the heart of Spring framework, we find the **Dispatcher Servlet**, which is responsible for coordinating all the communication between the controller and the views.



We will now start to learn about controllers!

We know a thing or two about views already! (i.e. JSP pages)

Review Spring Architecture (with more detail)

- In order for controllers to properly handle a request off to a specific JSP, Spring needs to know where to find the JSP pages within the project.
  - This is defined within the View Resolver bean in the **springmvc-servlet.xml** file:

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">
   <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
   <property name="prefix" value="/WEB-INF/jsp/"/>
   <property name="suffix" value=".jsp"/>
</bean>
```

- In this specific example, all the JSP files should be located in a folder called WEB-INF/jsp.
- **Admin Note**: You can assume that for any assignment in Tech Elevator, this step has been done for you.

Controllers

- In Spring, controllers are responsible for redirecting a client request to the appropriate view.
  - With our technology stack, the view will normally be implemented with **JSP**.
  - Controllers will be implemented as **Java** classes.

Controllers Class Syntax

- Suppose we have an application called (https://myTeSite.com)
  - Within this application we would like the site show a specific JSP page when we go to the following URL: https://myTeSite.com**/showSchedule**
- Within a controller class we want to use the following syntax:

```
@Controller
public class HomeController {

        @RequestMapping("/displaySchedule")
        public String displayHomePage() {
                return "homePage";
        }
}
```

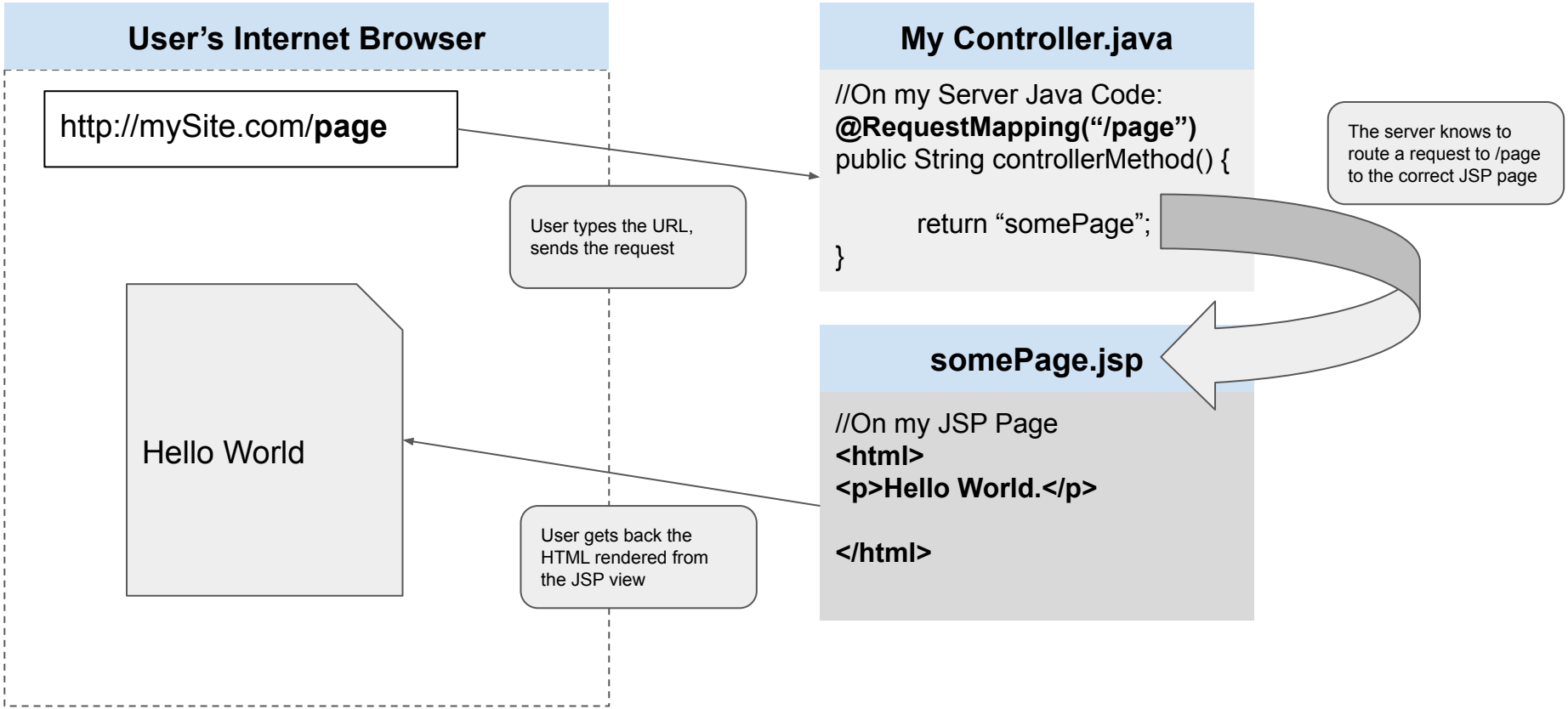
This annotation specifies that the class is a controller

This annotation specifies that the method displaySchedule will handle the request for “/displaySchedule”.

homePage is the name of a JSP file in your project.

Let’s do some coding!

Let’s summarize and Review



Specifying Request Mapping Parameters

- Suppose our request mapping requires that certain inputs be provided in order for it to do its job properly.
- We can accomplish this by the use of the @RequestParam annotation within the method's arguments:

```
@Controller
public class HomeController {

        @RequestMapping("/displaySchedule")
        public String displayHomePage(@RequestParam String inputDate) {
                return "homePage";
        }
}
```

This annotation specifies that the request only be handled if a parameter called inputDate with a valid String value is passed to it.

Specifying Request Mapping Parameters

We can also specify that the parameter be optional.

```
@Controller
public class HomeController {

    @RequestMapping("/displaySchedule")
    public String displayHomePage(@RequestParam (required=false) String inputValue) {

        System.out.println(inputValue);
        return "homePage";
    }
}
```

If no parameters were provided,
this will output **null**.

Specifying Request Mapping Parameters

- When a GET request is sent, the parameter is included in the URL of the request itself.
- In the current example we know that the mapping is expecting a parameter of inputDate, so the URL of the request would be:
  - https://myTeSite.com/showSchedule?inputDate=2020-01-01
- If there are multiple parameters, they are separated with ampersands:
  - https://myTeSite.com/showSchedule?param1=param1Value&param2=param2Value

Passing Data from a Controller to the View

Passing Data from the Controller to the JSP

- We just discussed a scenario where data is passed to the controller, now suppose we wanted to do the reverse - pass data from the controller to the JSP.
- One such way to do this is to use a collaborator object of class **ModelMap**.

Passing Data from the Controller to the JSP

- We just discussed a scenario where data is passed to the controller, now suppose we wanted to do the reverse - pass data from the controller to the JSP.
- One such way to do this is to use a collaborator object of class **ModelMap**.

Passing Data from the Controller to the JSP

Using a ModelMap takes on the following pattern:

```
@Controller
public class HomeController {

    @RequestMapping("/displaySchedule")
    public String displayHomePage(@RequestParam String inputDate, ModelMap model) {

        model.put(“greetingMsg”, “Hello”);
        return "homePage";
    }
}
```

We define a ModelMap reference on the arguments list.

We create a key value pair here. The JSP be able to use a variable called greetingMsg, with a value of “Hello”

Passing Data from the Controller to the JSP

- In the previous example we saw how we added a key called greetingMsg to the ModelMap and gave it a value of “Hello.”
- Within homePage.jsp, the following JSP code can be used to display the value:

```
<c:out value="${greetingMsg}" />
```

- In the previous example we dealt with a simple String but if you had an object, you can use dot notation:

```
<c:out value="${object.property}" />
```

Passing Data from the Controller to the JSP

Let's look at the situation for when a list of objects is passed to the JSP (follow the colors).

```
@Controller
//….
 // You have a widgetsDAO declared that interfaces with your data source
public class HomeController {

        @RequestMapping("/displaySchedule")
        public String displayHomePage(@RequestParam String inputDate, ModelMap model) {
                List<Widgets> widgets = widgetsDAO.getWidgets();
                model.put("widgetsList", widgets)
                return "homePage";
        }
}
```

Java

```
public class Widget {
        private long id;
        private String name;
        private String description;
}
```

Java

```
<c:forEach var="thisWidget" items="${widgetsList}">
        <c: out value= "${thisWidget.name}">
        <c: out value= "${thisWidget.description}">
</c:forEach>
```

JSP

1. We have a list of objects of class Widgets.

2. We place these widgets in the modelMap and give it a name of "widgetList"

3. On our JSP, it has received this java object called widgetList.

4. On our JSP, we can write a foreach loop against widgetList

Passing Data from a View to a Controller (A Preview)

Passing Data from a View to a Controller

- So far, we have used a browser and passed parameters manually via a URL:

  https://myTeSite.com/showSchedule?param1=abc&param2=123

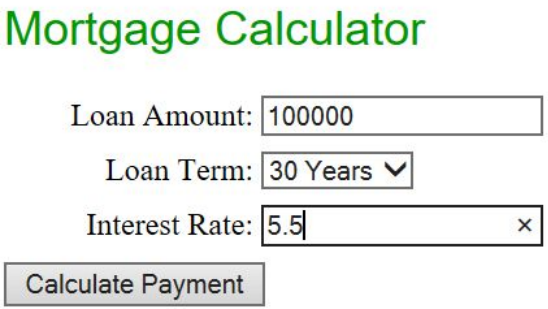
- This is not very convenient, we can instead use a form to send data to the server.
- Forms **should be sent using a POST** and not a GET, but technically speaking, you can do it with a GET.
  - … more on this when we talk about POST requests. For now, use a GET

Passing Data from a View to a Controller

Let’s consider the following JSP code:

```
<c:url var="formAction" value="/mortgageCalculatorResult" />
<form method="GET" action="${formAction}">
    <div class="formInputGroup">
        <label for="loanAmount">Loan Amount:</label>
        <input type="text" name="loanAmount" id="loanAmount" />
    </div>
    <div class="formInputGroup">
        <label for="loanTerm">Loan Term:</label>
        <select name="loanTerm" id="loanTerm">
            <option value="10">10 Years</option>
            <option value="15">15 Years</option>
            <option value="30">30 Years</option>
        </select>
    </div>
    <div class="formInputGroup">
        <label for="rate">Interest Rate:</label>
        <input type="text" name="rate" id="rate" />
    </div>
    <input class="formSubmitButton" type="submit" value="Calculate Payment"
/>
</form>
```

The JSP code renders this HTML form:



Passing Data from a View to a Controller

Let’s consider the following JSP code:

```
<c:url var="formAction" value="/mortgageCalculatorResult" />
<form method="GET" action="${formAction}">
    <div class="formInputGroup">
        <label for="loanAmount">Loan Amount:</label>
        <input type="text" name="loanAmount" id="loanAmount" />
    </div>
    <div class="formInputGroup">
        <label for="loanTerm">Loan Term:</label>
        <select name="loanTerm" id="loanTerm">
            <option value="10">10 Years</option>
            <option value="15">15 Years</option>
            <option value="30">30 Years</option>
        </select>
    </div>
    <div class="formInputGroup">
        <label for="rate">Interest Rate:</label>
        <input type="text" name="rate" id="rate" />
    </div>
    <input class="formSubmitButton" type="submit" value="Calculate Payment" />
</form>
```

1. The endpoint the form will “take us” is defined here.
2. The request will have a param called loanAmount with the amount the user entered.
3. The request will have a param called loanTerm with the amount the user entered.
4. Same for the rate.

http://localhost:8080/controllers-part1-lecture-fbn/mortgageCalculatorResult?loanAmount=100000&loanTerm=30&rate=5.5

Let’s do some coding!