



# NS4307

# Network Programming

Java Web Programming II

# Disclaimer

- Some parts of the notes is based on the online course provided in Treehouse ([teamtreehouse.com](https://teamtreehouse.com)).
  - **Course:** Spring
  - **Teacher:** Chris Ramacciotti

# Review

- Please make sure your web application can respond to a user request to the home page and send response body with a plain message. Then we can continue.
- But for the moment it is not that impressive, since this is a website, usually it involves with a user interface.
- This module will not cover how to do front end development, but we are going to use an existing template that uses HTML, CSS and JavaScript.
- It is recommended for you to be familiar with HTML, CSS and JavaScript as a web developer.

# Spring View

- In the past, people have used Java server pages, Java server faces, or struts to transform Java application data into HTML.
- But we will not be using these for this module. We will use the library that we added on previous lecture called Thymeleaf.
- The HTML file with Thymeleaf is the Spring View component.

# Thymeleaf

- Review: Thymeleaf is a modern server-side Java template engine for both web and standalone environments.
- It allows you to write HTML while including place holders for data that will come from Java objects.
- Thymeleaf has its own expression syntax that allows us to leverage its templating engine to do things like substitute data from Java objects, iterate over a Java list to create an HTML table and generate a URL for us.
  - <http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>

# Thymeleaf (cont.)

- By default, Thymeleaf library will expect the templates to be found in `src/main/resources/templates` directory of the project.
- Unless we configure it otherwise it will try to search for HTML files in `src/main/resources/templates` directory. So you need to create the folders required.
- Inside the templates folder, we will put in the view component of the Spring Framework. It consists of HTML files.

# Create HTML file

- Now lets create a HTML file with the following content in the folder that you have created and lets call it, home.html . (You may use different file name).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>I like Network Programming</title>
</head>

<body>
  <h1>ICE2305 Network Programming</h1>
  <p>I love to learn this module</p>
</body>
</html>
```

# Response with HTML file

- To response user request with a HTML file we need to state it in the Controller class.
- In the Controller class:
  - The `@RequestMapping` annotation is still required.
  - Since we are going to response with HTML file, we can omit `@ResponseBody`
  - A method that returns String value.
  - The return value should be the name of the HTML file.
    - Example:
      - For home.html file, the method should -> return “home”;
      - For index.html file, the method should -> return “index”;



# Example Response with HTML file

- The following is an example of the controller class where that wants to response with home.html file which we created in previous slide.

```
package controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class ControllerClass {

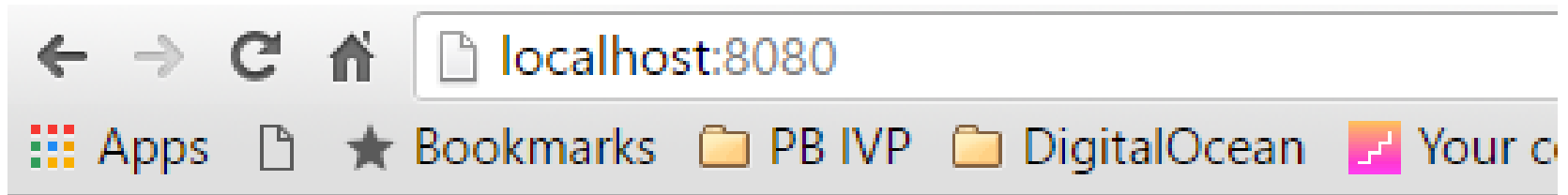
    @RequestMapping(value="/")
    public String home() {
        return "home";
    }
}
```

We only use  
@RequestMapping  
annotation.

The HTML file  
name is home.html  
so we return the  
file name "home"

# Example Response with HTML file

- Congratulation, you have send a response for a request with a HTML file.



## ICE2305 Network Programming

I love to learn this module

# Static File

- Static files are files that its content does not change that your template will use.
  - The content of the static files are not generated by the web application.
  - For example, image, css and javascript files.
- By default, Thymeleaf library will expect the static files to be found in src/main/resources/static directory of the project.
- Unless we configure it otherwise it will try to search for static files in src/main/resources/static directory. So you need to create the folders required.

# Static File (cont.)

- In the HTML file, instead of entering an absolute path for the static files, we'll let Thymeleaf figure out our path by using what's called url expression.

# Example Static File

- Lets add an image file into your current web application.



- You can find the image file called SICT.jpg in LMS under same section as this slide.
- In your implementation, lets create another folder inside the static folder and call it images. Then put the SICT.jpg into the folder.
  - `src/main/resources/static/images/SICT.jpg`

# Thymeleaf URL Expression

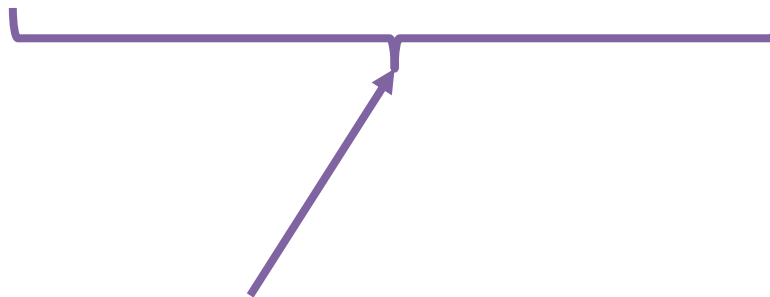
- Using Thymeleaf url expression allows us to leverage its templating engine to generate a URL for us.
- To do this we'll prefix to HTML tags attributes with "th" followed by a colon ":".
  - Example in <img> tag we can add th: to the src attribute:
    - 
  - Example in <a> tag we can add th: to the href attribute:
    - <a **th:**href="@{/contact}" />
- This will notify Thymeleaf to process the content of the attributes.

# Thymeleaf URL Expression (cont.)

- You noticed that in the previous slide, the example of Thymeleaf URL Expression contain @ and { }.
  - ``
- @ sign indicates to the Thymeleaf engine that a URL should be generate here.
- { } sign indicates the content inside the curly braces { } is the URL or URI that needs to be modified.

# Thymeleaf XML Namespace

- If you have added “th:” to some attributes, some IDE might consider this as an error.
  - This is because “th” is not in the XML name space.
  - To identify “th” into the XML name space, you need to identify this into the html tag.
  - Example:
    - `<html lang=“en” xmlns:th=“http://www.thymeleaf.org” > </html>`



A thymeleaf namespace is being declared  
for th:\* attributes.



# Thymeleaf URL Expression (cont.)

- The content of the attributes with Thymeleaf url expression will contain as follows:
  - **Absolute URLs: Starts with http:// or https://**
    - Thymeleaf will not modify if the content contain Absolute URL.
    - Example: `<a th:href="@{http://www.thymeleaf/documentation.html}">`
      - No changes will be made to the content.
      - `<a href="http://www.thymeleaf/documentation.html">`
  - **Context-relative URLs: Starts with /**
    - These are URLs which are supposed to be relative to the web application root once it is installed on the server.
    - Example: `<a th:href="@{/order/list}">`
      - If our app is installed at `http://localhost:8080/myapp`, this URL will output:
      - `<a href="/myapp/order/list">`

# Thymeleaf URL Expression (cont.)

## – Server-relative URLs: Starts with ~/

- These are URLs which are supposed to be relative to the server root.
- Example: `<a th:href="@{~/billing-app/showDetails.htm}">`
  - The current application's context will be ignored, therefore although our application is deployed at `http://localhost:8080/myapp`, this URL will output.
  - `<a href="/billing-app/showDetails.htm">`

## – Protocol-relative URLs: Starts with //

- *Protocol-relative* URLs are in fact absolute URLs which will keep the protocol (HTTP, HTTPS) being used for displaying the current page. They are typically used for including external resources like styles, scripts, etc.
- Example: `<script th:src="@{///scriptserver.example.net/myscript.js}">`
  - No changes will be made to the content.
  - `<script src="//scriptserver.example.net/myscript.js">`

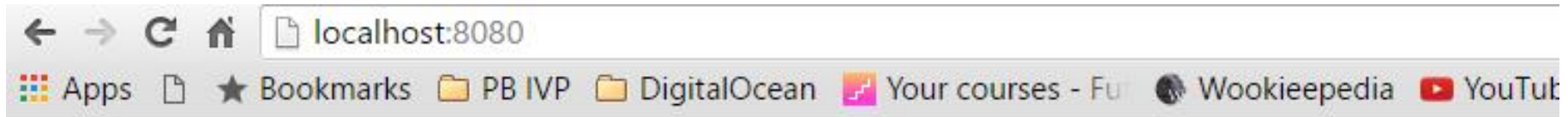
# Example Thymeleaf URL Expression

- Edit the HTML file home.html file that we created earlier and add `<img>` tag to show the image.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <title>I like Network Programming</title>
</head>
<body>
  <h1>ICE2305 Network Programming</h1>
  <p>I love to learn this module</p>
  
</body>
</html>
```

Added these  
to home.html  
file

# Page with Image



## ICE2305 Network Programming

I love to learn this module



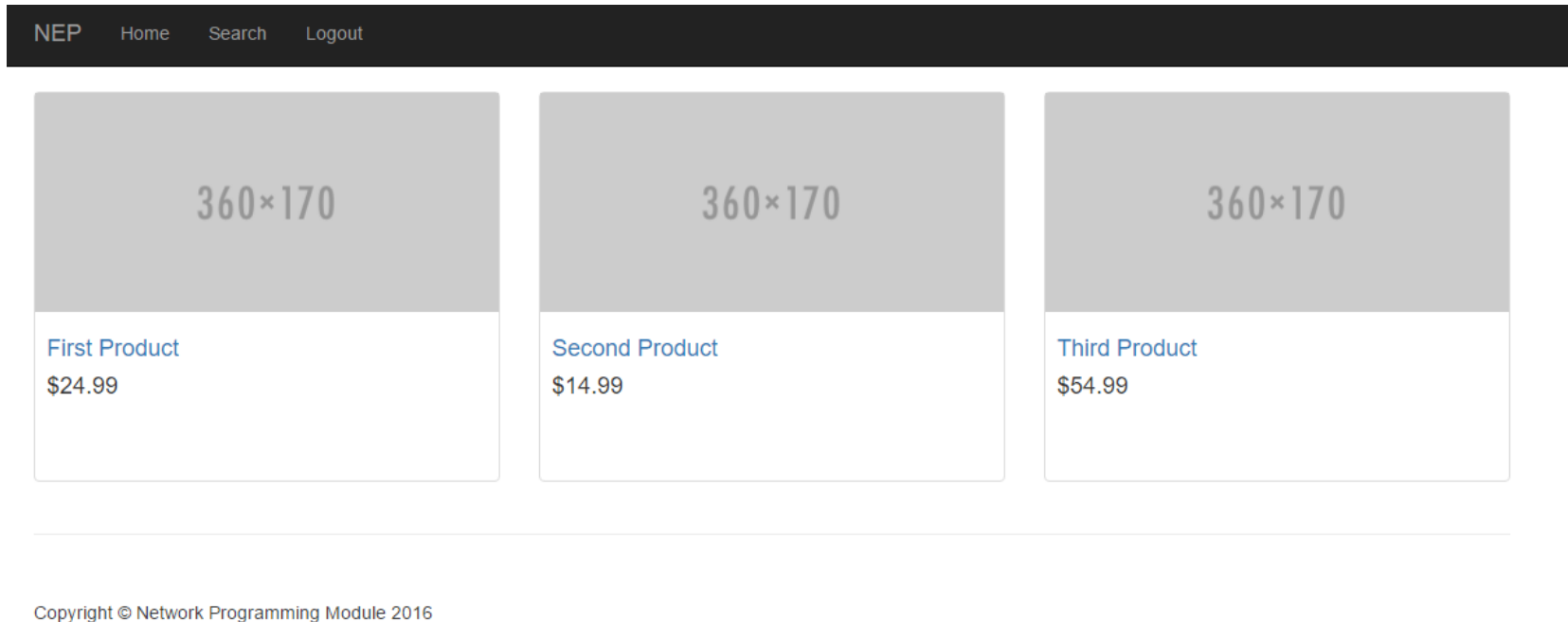
# Using Template

- Now that you get the basic idea on how Spring Framework works. Currently your website is just a static website where it does not have any server generated content.
- From here on we are going to use a template which will make the learning process much more faster..
- I have attached the template in LMS under same section as this slide called template.zip .
- The template is a modified version of Shop Homepage template from Start Bootstrap website.  
(<https://startbootstrap.com/template-overviews/shop-homepage/>).

# Using Template (cont.)

- The template uses bootstrap (<http://getbootstrap.com/>) framework and it uses HTML5 tags.
- Unfortunately Thymeleaf currently does not understand HTML5 tags, this is why we added nekoHTML to make spring boot understands HTML5 tags.
- Lets add the template into your web application.
  - All html file into the templates folder.
  - All other files into the static folder.

# NEP Shop Template

**NOTE:**

The website might look different on your end depending on your window size and resolution.

# Spring Model

- Currently, we managed to serve web application with static web content only. We have yet to implement a web application that have data that are dynamically generated.
- We are going to create Plain Old Java Object (POJO) classes to model the data of the web application.
- Lets create a new package called model in src/main/java directory and create a Java class into the package.



# Plain Old Java Object

- A Plain Old Java Object (POJO) is a Java object whose class is coded for its natural functionality and not for the framework which it will be used in.
- The class is coded with fields, constructors, getters and setters, and methods that are useful to the object.
- In the next slide, is an example of a POJO class. Lets implement it as well into our Spring Web Application, create a class called Product.java in package called model.

# Example Plain Old Java Object

```
public class Product {  
  
    private String name;  
    private double price;  
    private String picFile;  
    private boolean inStock;  
  
    public Product(String name, double price, String picFile, boolean inStock) {  
        super();  
        this.name = name;  
        this.price = price;  
        this.picFile = picFile;  
        this.inStock = inStock;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public double getPrice() {  
        return price;  
    }  
  
    public void setPrice(double price) {  
        this.price = price;  
    }  
  
    public String getPicFile() {  
        return picFile;  
    }  
  
    public void setPicFile(String picFile) {  
        this.picFile = picFile;  
    }  
  
    public boolean isInStock() {  
        return inStock;  
    }  
  
    public void setInStock(boolean inStock) {  
        this.inStock = inStock;  
    }  
}
```

# POJO in Controller

- Making an object in a controller is the same as making an object in any other Java classes.
- For now, lets create the object inside the method that handles `@RequestMapping(value="/")`.

```
Product p = new Product("HTC One X9", 900.00, "htconex9.png", false);
```

- For Thymeleaf to access the object in the controller, you need to pass a `ModelMap` object as the parameter.
  - Then use the `ModelMap` `put` method to insert objects into it.
  - Example: `modelMap.put("product", p);`
  - “product” is the key for the object `p` in the map.

# POJO in Controller (cont.)

- When the Spring framework calls our controller method as a result of a user requesting certain URI, Spring will pass in an instance of a ModelMap into this parameter.
- The data type for the ModelMap key should be String.
- The key mentioned in the ModelMap can be used by Thymeleaf to access the data and generate the data into its template, HTML file.



# Example POJO in Controller

- Lets modify the home method that we created earlier.

Add parameter for ModelMap object called modelMap.  
This allows Thymeleaf to access data added into the ModelMap.

```
@RequestMapping(value="/")  
public String home(ModelMap modelMap) {  
    Product p = new Product("HTC One X9", 900.00, "htconex9.png", false);  
    modelMap.put("product", p);  
    return "index";  
}
```

Insert the newly create  
Product object with a  
key "product".

Create Product object

# Data to Thymeleaf Templates

- The advantage of using Thymeleaf is the ability to combine static HTML with dynamic data.
- It is possible to feed data stored in POJOs, into Thymeleaf Templates.


# Data to Thymeleaf Templates (cont.)

- In the HTML file, to get the data stored in POJOs using Thymeleaf. You need to add the following into the attribute inside certain HTML tags.
  - Add **th:** prefix to the attribute.
  - In the value of the attribute, use dollar sign **\$** symbol and then curly braces **{ }**
  - Inside the curly braces, put in the key of the object data that you want to retrieve.
  - Example: `<h4 th:text="$${product.price}"></h4>`

The key stated in  
ModelMap object.

Instance Variable in the  
object class.

# Example Data to Thymeleaf Templates

- Lets edit the index.html file provided in the template with comment `<!-- Slide #32 Example Data to Thymeleaf Templates -->`
  - Change the src location to the location of the image from the Product object.
    - ``


Using single quote to add static text to be concatenated.

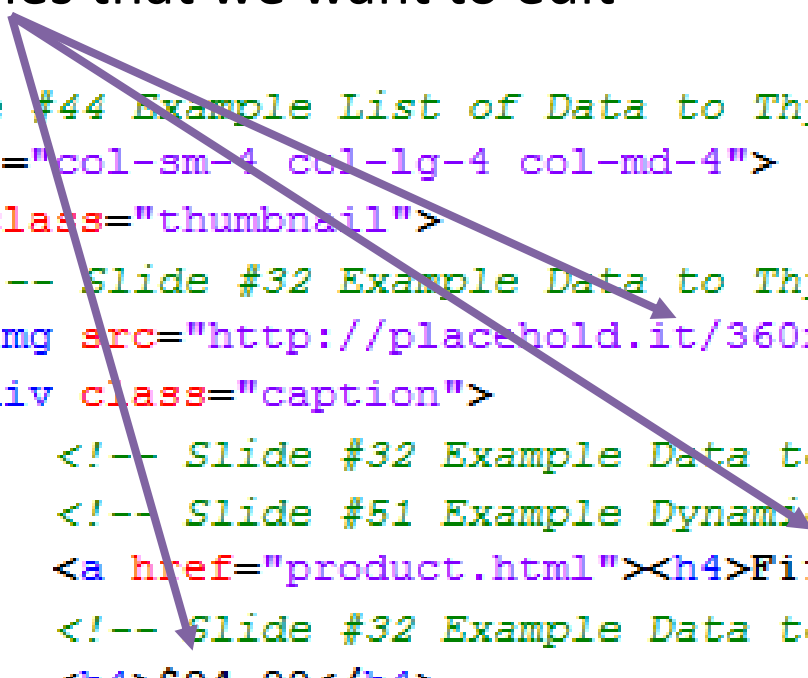
Concatenate with the image file name
  - Remove the text in between `<h4></h4>`. Inside `<h4>` tag add an attribute called text. Finally, get the name from the Product object.
    - `<h4 th:text="${product.name}"></h4>`
  - Get the price from the Product object.
    - `<h4 th:text="${product.price}"></h4>`



# Example Data to Thymeleaf Templates (cont.)

- The lines that we want to edit

```
<!-- Slide #44 Example List of Data to Thymeleaf Templates -->
<div class="col-sm-4 col-lg-4 col-md-4">
  <div class="thumbnail">
    <!-- Slide #32 Example Data to Thymeleaf Templates -->
    
    <div class="caption">
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <a href="product.html"><h4>First Product</h4></a>
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <h4>$24.99</h4>
    </div>
  </div>
</div>
</div>
```



# Example Data to Thymeleaf Templates (cont.)

- After the changes.


```

<!-- Slide #44 Example List of Data to Thymeleaf Templates -->
<div class="col-sm-4 col-lg-4 col-md-4">
  <div class="thumbnail">
    <!-- Slide #32 Example Data to Thymeleaf Templates -->
    
    <div class="caption">
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <!-- Slide #51 Example Dynamic Page (cont.) -->
      <a href="product.html"><h4 th:text="${product.name}"></h4></a>
      <!-- Slide #32 Example Data to Thymeleaf Templates -->
      <h4 th:text="${product.price}"></h4>
    </div>
  </div>
</div>
</div>
  
```

# Example Data to Thymeleaf Templates (cont.)

- When you run and access the web application.
  - The page will display the Product object that you added into the Controller ModelAndView earlier.

NEP Home Search Logout



HTC One X9  
900.0

360×170

Second Product  
\$14.99

360×170

Third Product  
\$54.99