Acceso a Datos

UT5 – ACCESO WEB CON SPRING BOOT

THYMELEAF

1. Configuración de Thymeleaf

Spring Boot nos ofrece una configuración por defecto de Thymeleaf, pero si lo deseamos podemos cambiar dichas propiedades en el archivo de aplication.properties. todas las propiedades que podemos cambiar las podemos consultar en la siguiente página web.

https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

1. Configuración de Thymeleaf

Las propiedades por defecto de Thymeleaf serían:

```
# THYMELEAF (ThymeleafAutoConfiguration)
spring.thymeleaf.cache=true # Whether to enable template caching.
spring.thymeleaf.check-template=true # Whether to check that the template exists before rendering it.
spring.thymeleaf.check-template-location=true # Whether to check that the templates location exists.
spring.thymeleaf.enabled=true # Whether to enable Thymeleaf view resolution for Web frameworks.
spring.thymeleaf.enable-spring-el-compiler=false # Enable the SpringEL compiler in SpringEL expressions.
spring.thymeleaf.encoding=UTF-8 # Template files encoding.
spring.thymeleaf.excluded-view-names= # Comma-separated list of view names (patterns allowed) that should be excluded from resolution.
spring.thymeleaf.mode=HTML # Template mode to be applied to templates. See also Thymeleaf's TemplateMode enum.
spring.thymeleaf.prefix=classpath:/templates/ # Prefix that gets prepended to view names when building a URL.
spring.thymeleaf.reactive.chunked-mode-view-names= # Comma-separated list of view names (patterns allowed) that should be the only ones executed
spring.thymeleaf.reactive.full-mode-view-names= # Comma-separated list of view names (patterns allowed) that should be executed in FULL mode eve
spring.thymeleaf.reactive.max-chunk-size=0B # Maximum size of data buffers used for writing to the response.
spring.thymeleaf.reactive.media-types= # Media types supported by the view technology.
spring.thymeleaf.render-hidden-markers-before-checkboxes=false # Whether hidden form inputs acting as markers for checkboxes should be rendered
spring.thymeleaf.servlet.content-type=text/html # Content-Type value written to HTTP responses.
spring.thymeleaf.servlet.produce-partial-output-while-processing=true # Whether Thymeleaf should start writing partial output as soon as possibl
spring.thymeleaf.suffix=.html # Suffix that gets appended to view names when building a URL.
spring.thymeleaf.template-resolver-order= # Order of the template resolver in the chain.
spring.thymeleaf.view-names= # Comma-separated list of view names (patterns allowed) that can be resolved.
```

1. Configuración de Thymeleaf

Nuestro archivo de **application.properties** podría quedar así:

```
# OSIV (Open Session in View) está habilitado de forma predeterminada en Spring Boot,
# y OSIV es realmente una mala idea desde una perspectiva de rendimiento y escalabilidad .
spring.jpa.open-in-view=false
# URL jdbc de conexión a la base de datos
spring.datasource.url=jdbc:mysql://localhost/mvc
# Usuario y contraseña de la base de datos
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
# Habilitamos los mensajes sql en el log
spring.jpa.show-sql=true
# Le indicamos a JPA/Hibernate si queremos que se encargue de generar el DDL
spring.jpa.hibernate.ddl-auto=create
# Dialecto de Hibernate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
# La caché está habilitada por defecto. Si la deshabilitamos, podemos realizar un hot swapping
spring.thymeleaf.cache=false
# Comprobación de la existencia de la ruta de la plantilla
spring.thymeleaf.check-template-location=true
# Habilitación/Deshabilitación de Thymeleaf
spring.thymeleaf.enabled=true
# Codificación de las plantillas
spring.thymeleaf.encoding=UTF-8
# Modo en el que trabajará Thymeleaf (HTML, CSS, JAVASCRIPT, RAW, TEXT, XML)
spring.thymeleaf.mode=HTML
# Prefijo para las rutas de las plantillas
spring.thymeleaf.prefix=classpath:/templates/
# Sufijo para las rutas de las plantillas
spring.thymeleaf.suffix=.html
```

1. Configuración de Thymeleaf

Más sobre hot swapping:
https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/howto-hotswapping.html

Para más información de OSIV:
https://picodotdev.github.io/blog-bitix/2020/05/el-patron-open-session-in-view-que-es-ventajas-problemas-y-alternativas

2. Tipos de expresiones

Permite trabajar con varios tipos de expresiones:

- **Expresiones variables**: Son quizás las más utilizadas, su forma es: ${…}
- **Expresiones de selección**: Son expresiones que nos permiten reducir la longitud de la expresión si prefijamos un objeto mediante una expresión variable, su forma es: *{…}
- **Expresiones de mensaje**: Que nos permiten, a partir de ficheros properties o ficheros de texto, cargar los mensajes e incluso realizar la internalización de nuestras aplicaciones, su forma es: #{…}
- **Expresiones de enlace**: Nos permiten crear URL que pueden tener parámetros o variables, su forma es: @{…}
- **Expresiones de fragmentos**: Nos van a permitir dividir nuestras plantillas en plantillas más pequeñas e ir cargándolas según las vayamos necesitando, su forma es: ~{…}

2. Tipos de expresiones

Por defecto, cuando se trabaja con Thymeleaf se suele hacer con el lenguaje de expresiones OGNL (Object-Graph Navigation Language), aunque cuando trabajamos conjuntamente con Spring MVC podemos utilizar el SpEL (Spring Expression Language).

3. Texto y variables en Thymeleaf

Lo podemos manejar mediante los atributos `th:text` y `th:utext`. Con ellos, podemos inyectar cualquier tipo de contenido textual.

```
<span th:text="${nombre}">mundo</span>
```

Si el texto a inyectar incluye etiquetas HTML y queremos que estas sean procesadas, th:text no va a producir el resultado esperado. Para ello, deberíamos usar th:utext, que mostrará el texto procesando las etiquetas.

También podemos utilizar th:text y th:utext para inyectar valores numéricos. Se realizará el casting correspondiente para mostrarlo como texto.

3. Texto y variables en Thymeleaf

PLANTILLA BASE

- Actualizamos la plantilla recogiendo el valor del controlador.

Valor *por defecto*.
Solo aparece cuando visualizamos la plantilla estáticamente.

Atributo *thymeleaf* que indica que vamos a setear un texto.

```
</head>
<body>
  <h1>Hola <span th:text="${nombre}">Mundo</span></h1>
  ....
</body>
```

Nombre de la variable que busca en el *contexto*

3. Texto y variables en Thymeleaf

Si queremos inyectar en la plantilla una variable que contiene texto HTML, por ejemplo la variable saludo que contiene el siguiente texto:

```
model.addAttribute("saludo","<strong>Bienvenidos a tod@s al curso de Spring con <em>Thymeleaf</em></strong>");
```

Si utilizamos la etiqueta th:text el resultado sería el siguiente, y no es lo que deseamos:

Hola Pepe

Bienvenidos a tod@s al curso de Spring con Thymeleaf

3. Texto y variables en Thymeleaf

Por lo tanto tendríamos que utilizar th:utext y el resultado sería

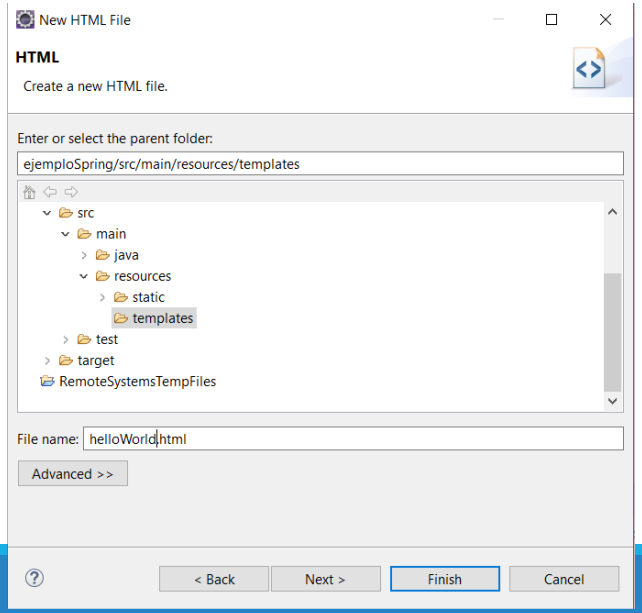
```
<h1>
    Hola <span th:text="${nombre}">mundo!!</span>
</h1>
<h2><span th:utext="${saludo}"></span></h2>
```

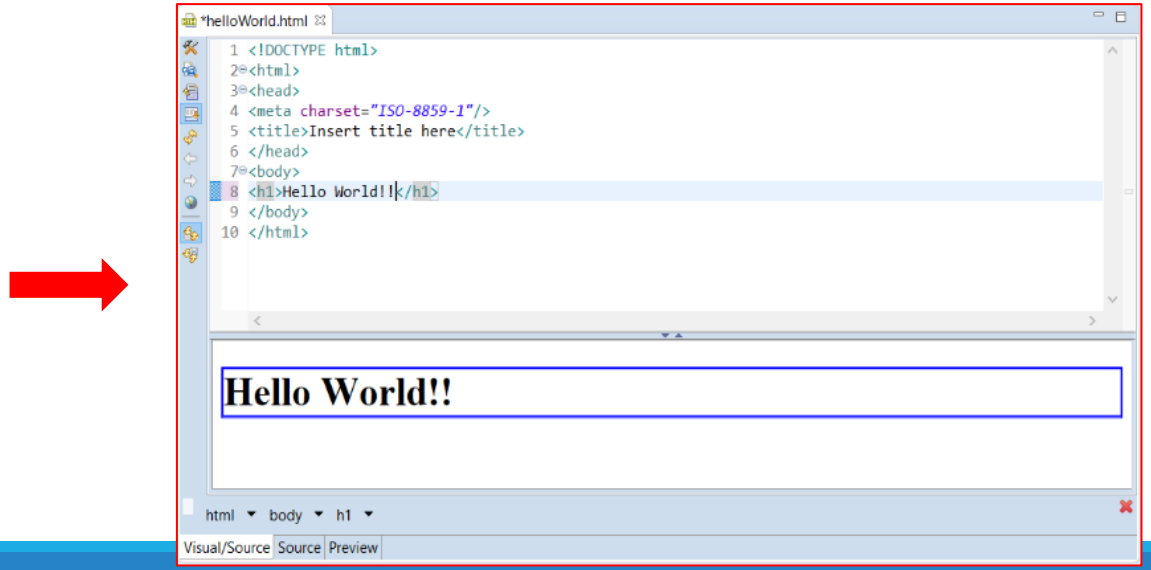
Hola Pepe

Bienvenidos a tod@s al curso de Spring con *Thymeleaf*

4. Ejemplo 1 Spring MVC con Thymeleaf

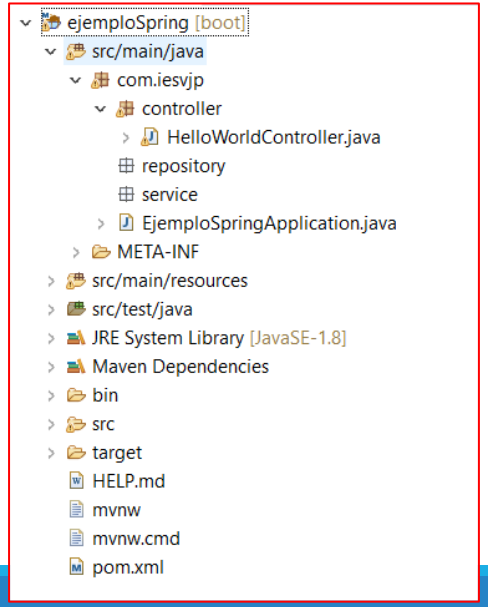
Lo primero que vamos a hacer es crearnos una vista llamada “HelloWorld.html” y la vamos a llamar desde el navegador. Lo primero que tenemos que hacer es crearnos un template para esta página:





4. Ejemplo 1 Spring MVC con Thymeleaf

Ya tenemos creada la vista, y tan solo hace falta llamarla desde un Controlador. Nos vamos a crear una nueva clase HelloworldController.java dentro del paquete **controller**:



4. Ejemplo 1 Spring MVC con Thymeleaf

El controlador HelloWorldController va a atender las peticiones que hagamos con el navegador y retornará las vistas. Podemos hacerlo de 2 formas (ver diapositiva siguiente).

Si vamos a llamar a la misma vista en varios métodos sería recomendable crearse una constante con el nombre de la vista para facilitar el cambio.

4. Ejemplo 1 Spring MVC con Thymeleaf

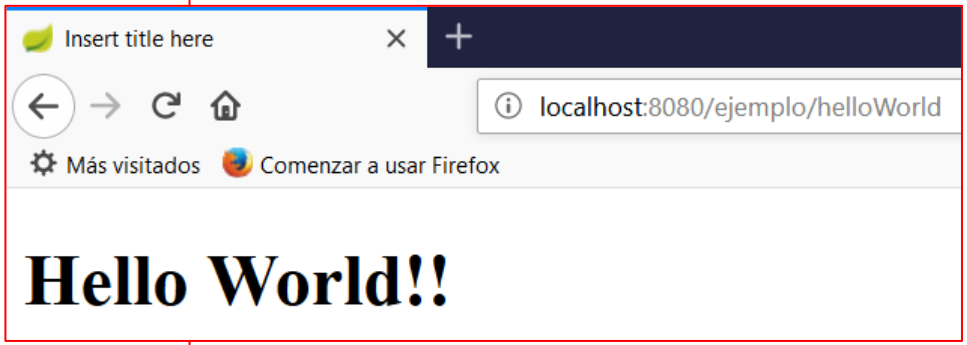
```java
package com.iesvjp.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/ejemplo")
public class HelloWorldController {
    public static final String HELLO_WORLD_VIEW = "helloWorld";

    //primera forma
    @GetMapping("/helloWorld")
    public String helloWorld() {
        return HELLO_WORLD_VIEW;
    }

    //segunda forma
    @GetMapping("/helloWorldMAV")
    public ModelAndView helloWorldMAV() {
        return new ModelAndView(HELLO_WORLD_VIEW);
    }
}
```



5. Construcción de URLs

De forma general, una URL en Thymeleaf será una expresión del tipo @{...}.

Distinguimos dos tipos principales:

- URLs con parámetros (QueryString)
- URLs con variables en el path

5.1. URLs con parámetros

Son del estilo a `www.mitienda.com/productos/detalle?idCategoria=17`

Cuando queremos construir URLs que contengan parámetros, y obtener estos en base a una expresión, la notación será:

```
@{/../path?(p1=v1, p2=v2, p3=v3, ...)}
```

5.1. URLs con parámetros

Por ejemplo:

```
<a th:href="@{/(idCategoria=${categoria.id})}"
th:text="${categoria.nombre}">Categoría</a>
```

Y, en otra forma:

```
<a th:href="@{/?idCategoria=__${categoria.id}__}
th:text="${categoria.nombre}">Categoría</a>
```

En ambos casos daría como resultado:

```
<a href="/?idCategoria=7" >Categoría 7</a>
```

5.2. URLs con variables en el path

Son del estilo a `www.mitienda.com/productos/detalle/17`

Cuando queramos que una URL tenga variables en el path, la notación será:

```
@{/.../path/{var1}/{var2}(var1=v1, var2=v2)}
```

5.2. URLs con variables en el path

Por ejemplo:

```
<a href="#"
     th:href="@{/product/{id}(id=${producto.id})}">
     <div class="col-item">
     …
     </div>
</a>
```

Daría como resultado:

```
<a th:href="@{/product/6}">
     <div class="col-item">
     …
     </div>
</a>
```

6. Recibiendo peticiones GET

Nuestra plantilla html helloWorld2.html sería la siguiente:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p>Hello <span th:text="${name_model}">World</span></p>
</body>
</html>
```

6. Recibiendo peticiones GET

Y la clase HelloWorldController2.java

```java
package com.iesvjp.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/helloWorld")
public class HelloWorldController2 {
    private static final String HELLOWORLD_VIEW = "helloWorld";

    //localhost:9000/helloWorld/request1?nm=Ana
    @GetMapping("/request1")
    public ModelAndView request1(
            @RequestParam(name = "nm", required = false, defaultValue = "World")
            String name) {
        ModelAndView mav = new ModelAndView(HELLOWORLD_VIEW);
        mav.addObject("name_model", name);
        return mav;
    }

    //FORMA 2
    //localhost:9000/helloWorld/request2/Ana
    @GetMapping("/request2/{nm}")
    public ModelAndView request2(@PathVariable("nm") String name) {
        ModelAndView mav = new ModelAndView(HELLOWORLD_VIEW);
        mav.addObject("name_model", name);
        return mav;
    }
}
```

7. Expresiones en Thymeleaf

Thymeleaf utiliza en su dialecto estándar el **lenguaje OGNL (Object-Graph Navigation Language),** un lenguaje de expresiones para Java, que permite trabajar con objetos, tomando y estableciendo propiedades, haciendo llamadas a métodos, manejo de arrays, …

Sin embargo, el dialecto para Spring utiliza **SpEL (Spring Expression Language),** otro lenguaje de expresiones que es común a todos los módulos y tecnologías de Spring, con análogas funcionalidades.

En nuestro caso, dado que utilizaremos Thymeleaf con Spring (con su correspondiente dialecto), tendremos que hacer uso de SpEL, si bien, en la gran mayoría de los casos, las expresiones OGNL y SpEL serán iguales.

8. Expresiones variables

Dada una expresión, como `${today}`, Thymeleaf buscará en el contexto una variable llamada *today*.

Tenemos también múltiples posibilidades, con la notación de punto (`${persona.nombre}`) o de corchete (`${persona['padre']['nombre']}`); así como la manipulación de *maps* (`${personas['Luismi'].edad}`) y arrays (`${arrayPersonas[0].nombre}`).

También podemos realizar llamadas a métodos definidos en los objetos (`${persona.nombreCompleto()}`).

9. Expresiones de selección

Las expresiones de selección o expresiones en selección son expresiones variables que nos permiten ejecutarlas en el marco de un objeto, creando entonces expresiones más abreviadas.

Los siguientes fragmentos de una plantilla nos ilustrarán como ejemplo, y son equivalentes.

```
<div th:object="${session.usuario}">
    <p>Nombre: <span th:text="*{nombre}">Luis
Miguel</span>.</p>
    <p>Apellidos: <span th:text="*
{apellidos}">López</span>.</p>
    <p>Nacionalidad: <span th:text="*
{nacionalidad}">Español</span>.</p>
</div>
```

9. Expresiones de selección

Sin expresiones abreviadas:

```
<div>
    <p>Nombre: <span
th:text="${session.usuario.nombre}">Luis Miguel</span>.</p>
    <p>Apellidos: <span
th:text="${session.usuario.apellidos}">López</span>.</p>
    <p>Nationality: <span
th:text="${session.usuario.nacionalidad}">Español</span>.</p
>
 </div>
```

10. Expresiones de utilidad

Disponemos de múltiples objetos de utilidad, con decenas de métodos, que nos ayudarán en las tareas más comunes:

- `#execInfo`: información sobre la plantilla que estamos procesando.
- `#messages`: tratamiento de expresiones de mensajes conjuntamente con expresiones variables.
- `#uris`: métodos para el tratamiento de URLs/URIs
- `#conversions`: métodos para ejecutar conversiones (si es que las hay configuradas)
- `#dates`: tratamiento de fechas `java.util.Date`.
- `#calendars`: tratamiento de fechas `java.util.Calendar`.
- `#numbers`: formateo de números
- `#strings`: tratamiento de cadenas de caracteres.
- `#objects`: métodos para objetos en general.

10. Expresiones de utilidad

- `#bools`: métodos para la evaluación de booleanos
- `#arrays`: métodos para el tratamiento de arrays.
- `#lists`: métodos para el tratamiento de listas.
- `#sets`: métodos para el tratamiento de *sets*.
- `#maps`: métodos para el tratamiento de *maps*.
- `#aggregates`: metodos para el cálculo de medias aritméticas y sumas en arrays o colecciones.
- `#ids`: métodos para la gestión del atributo id.

10. Expresiones de utilidad

Dada la cantidad de clases y métodos disponibles, es recomendable visitar la documentación del paquete `org.thymeleaf.expressions`, donde están definidos los métodos de cada uno de estos objetos:

https://www.thymeleaf.org/apidocs/thymeleaf/3.0.0.RELEASE/org/thymeleaf/expression/package-summary.html

11. Operaciones aritméticas, condiciones y comparaciones

Operadores aritméticos:

- Suma: `${4 + 0.2}`
- Resta: `${3 - 0.12}`
- Multiplicación: `${4 * 3}`
- Divisón entera: `${4 / 3}`
- Resto: `${5 % 3}`
- Divisón no entera `${4 / 3.0}`
- Potencia `${4^3}`
- Cambio de signo `${-(-2)}`

11. Operaciones aritméticas, condiciones y comparaciones

Operadores textuales:

- Concatenación: `${'Hola ' + 'mundo’}`
- Sustitución de literales: `|Mi nombre es ${nombre}|`

11. Operaciones aritméticas, condiciones y comparaciones

Operadores relacionales:

- Menor: `${3 < 2}` o `${3 lt 2}`
- Menor o igual: `${3 <= 3}` o `${3 le 3}`
- Mayor: `${3 > 2}` o `${3 gt 2}`
- Mayor o igual: `${3 >= 3}` o `${3 ge 3}`
- Igual: `${'hola' == 'Hola'}` `${'hola' eq 'Hola'}`
- Distinto: `${'hola' != 'Hola'}` o `${'hola' ne 'Hola'}`

11. Operaciones aritméticas, condiciones y comparaciones

Operadores booleanos:

- And: `${(3 < 4) and (4 < 5)}`
- Or: `${(3 > 4) or (4 < 5)}`
- Negación: `${ not (3 < 4) }`

11. Operaciones aritméticas, condiciones y comparaciones

Comparaciones:

th:if y th:unless (a no ser qué)

```
<div th:if="${not #lists.isEmpty(lista)}">
<p th:text="|Nombre del producto:
${lista[0].nombre}|">nombre</p>
</div>

<div th:unless="${not #lists.isEmpty(lista)}">
<p>No hay productos disponibles en la lista</p>
</div>
```

11. Operaciones aritméticas, condiciones y comparaciones

Operadores condicionales:

Mediante el uso del operador *condición ? valor si verdadero : valor si falso*

```
<p th:text="${not #lists.isEmpty(lista)} ? |Nombre del producto: ${lista[0].nombre}| : 'No hay productos disponibles en la lista'">nombre</p>
```

11. Operaciones aritméticas, condiciones y comparaciones

Elemento th:block (o th-block):

`th:block` es un contenedor de atributos Thymeleaf. Estos atributos serán procesados, y posteriormente desaparecerá el bloque, pero no su contenido. Es muy útil en varios contextos, como aplicación de condiciones a un grupo de elementos.

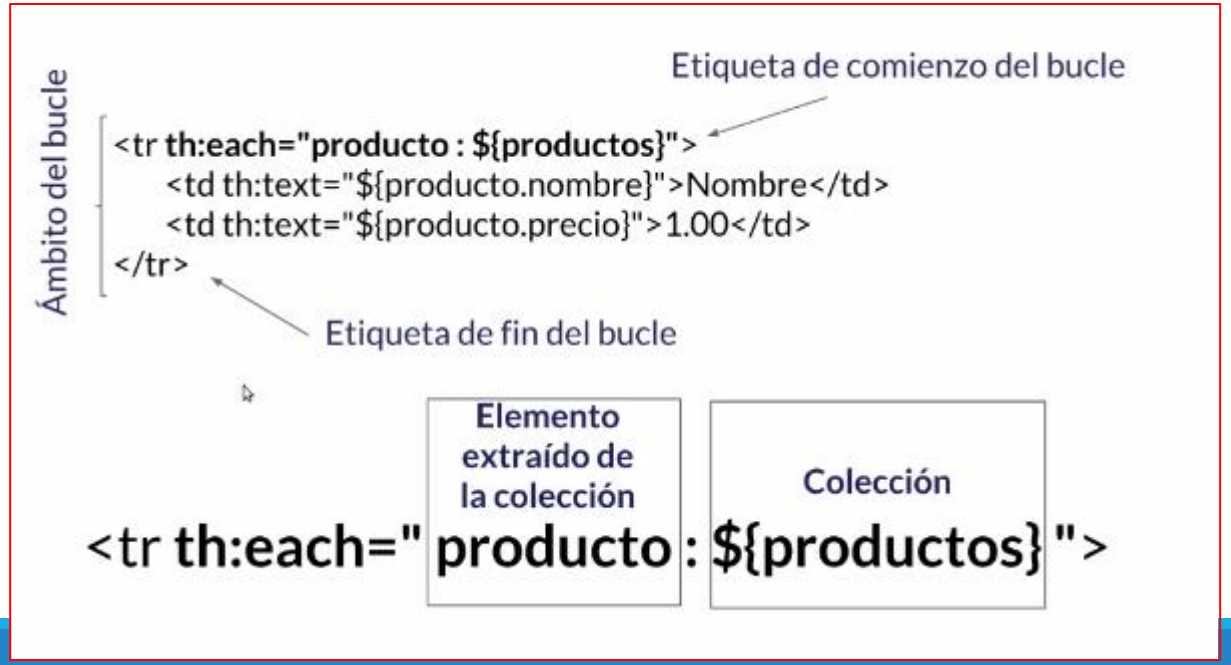
Por ejemplo:

```
<table>
    <th:block th:each="user : ${users}">
        <tr>
            <td th:text="${user.login}">...</td>
            <td th:text="${user.name}">...</td>
        </tr>
        <tr>
            <td colspan="2" th:text="${user.address}">...</td>
        </tr>
    </th:block>
</table>
```

Nos permite crear dos filas por usuario sin usar una etiqueta HTML adicional, como *div*, que no sería correcta en este contexto.

12. Bucles e iteraciones

Thymeleaf nos permite iterar sobre colecciones utilizando el atributo `th:each`.



12. Bucles e iteraciones

Nos permite iterar sobre una extensa variedad de colecciones :

- `java.util.List`.
- `Arrays`
- `java.util.Iterable`.
- `java.util.Collection`.
- `java.util.Enumeration`.
- `java.util.Iterator`.
- `java.util.Map` (devuelve objetos de tipo java.util.Map.Entry)

12. Bucles e iteraciones

Estado de la iteración:

Al usar `th:each`, Thymeleaf nos ofrece un buen mecanismo para identificar el estado de la iteraciones: la variable *status*:

- El índice actual (`index`), con valor inicial 0.
- El índice actual (`count`), con valor inicial 1.
- El número total de elementos (`size`).
- Si la iteración es par o impar (`even`/`odd`).
- Si es la primera iteración (`first`), o la última (`last`).

```
<tr th:each="producto, iterStat : ${productos}">
   ...
</tr>
```

- Si no la declaramos explícitamente, Thymeleaf siempre crea una variable añadiendo el sufijo *Stat* a nuestra variable de iteración.

```
<tr th:each="producto : ${productos}">
   <td th:text="${productoStat.count}">Num</td>
   ....
</tr>
```

Dudas y preguntas