

# Tema 3

## Controladores y Vistas



<b>Ejercicio 3.1.....</b>	<b>2</b>
<b>Ejercicio 3.2.....</b>	<b>6</b>
<b>Ejercicio 3.3.....</b>	<b>8</b>
<b>Ejercicio 3.4.....</b>	<b>10</b>
<b>Ejercicio 3.5.....</b>	<b>12</b>

## Ejercicio 3.1

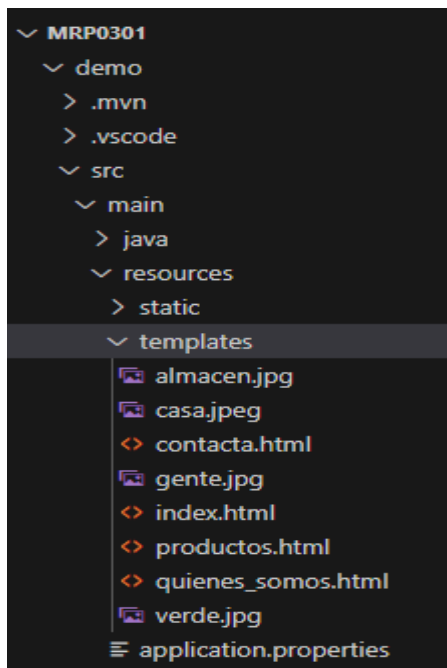
3.1. Toma el proyecto del ejercicio 2.4 del tema anterior y desarrolla una clase controladora que contenga diferentes `@GetMapping` que devuelvan las vistas solicitadas (index, quienes-somos, productos, contacta).

- a) ¿Tienes que cambiar de ubicación las vistas? ¿Por qué?
- b) ¿Tienes que cambiar el código HTML del menú de navegación de las páginas?
- c) ¿Tienen que llamarse igual las rutas del `GetMapping` y las vistas?

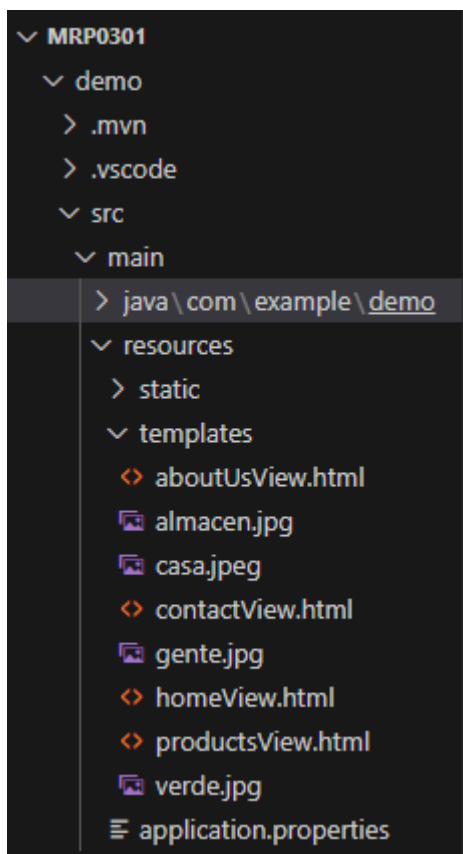
La página index será servida para las URL: /index, /home, o simplemente /. Ya que las rutas y las vistas no tienen por qué llamarse igual, renombra las vistas con el sufijo “view”: `homeView.html`, `aboutUsView.html`, `productsView.html`, `contactView.html`. (Opcional) Elimina el mapping quienes-somos y haz que muestre la vista `aboutUsView.html` mediante un archivo de configuración (implementando `WebMvcConfigurer`).

Respuesta:

Para resolver este ejercicio, y contestando a la pregunta del apartado a), es necesario copiar los archivos html (junto con las imágenes) de la carpeta static del ejercicio 2.4 a la carpeta templates de este ejercicio:



Se renombran los archivos html:



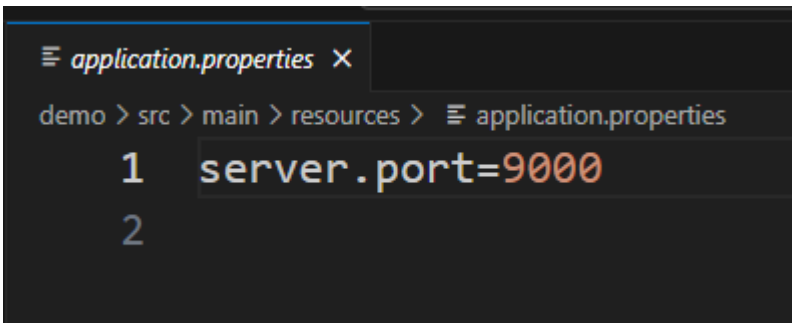
Luego se crea y escribe la clase Controller, con sus @GetMapping:

```

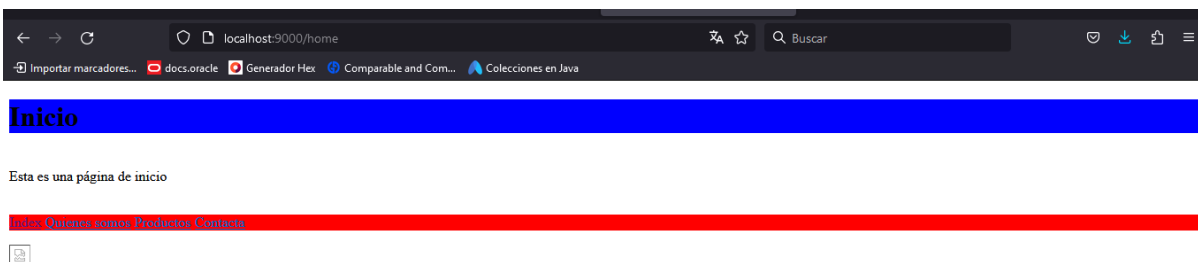
@Controller//Indica que esta clase será un controlador
public class ControladorUno {
    @GetMapping({"/index","/home","/"})
    /*GetMapping contiene la ruta o rutas url que recibirá para devolver index.html
    En este caso, devolverá index si en la url se busca "/index","/home"
    o simplemente"/"/
    public String indexView(){
        return "indexView";/*El archivo html que devolverá sin su extensión html */
    }
    @GetMapping("/about-us")
    public String quienesSomosView(){
        return "aboutUsView";
    }
    @GetMapping("/contact")
    public String contactaView(){
        return "contactView";
    }
    @GetMapping("/products")
    public String productosView(){
        return "productsView";
    }
}

```

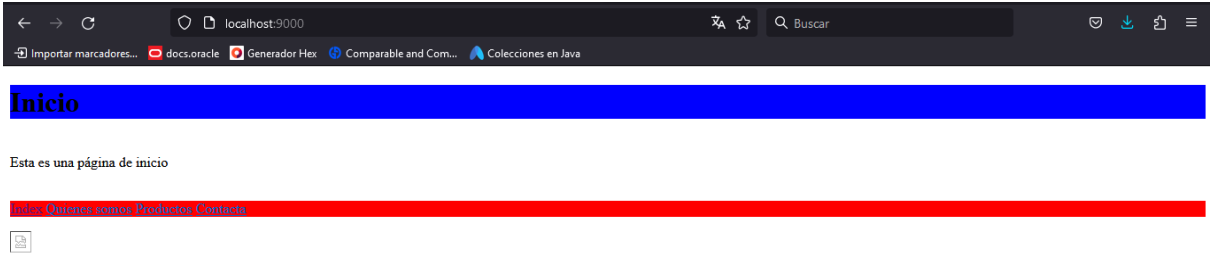
Y por último, se configura el puerto:



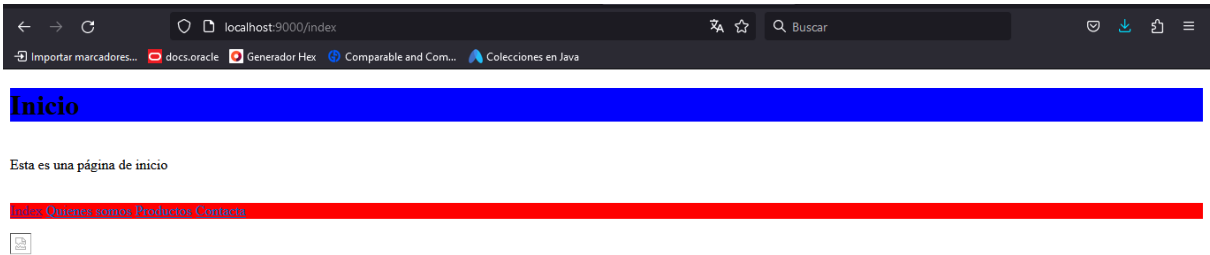
Resultado en home:



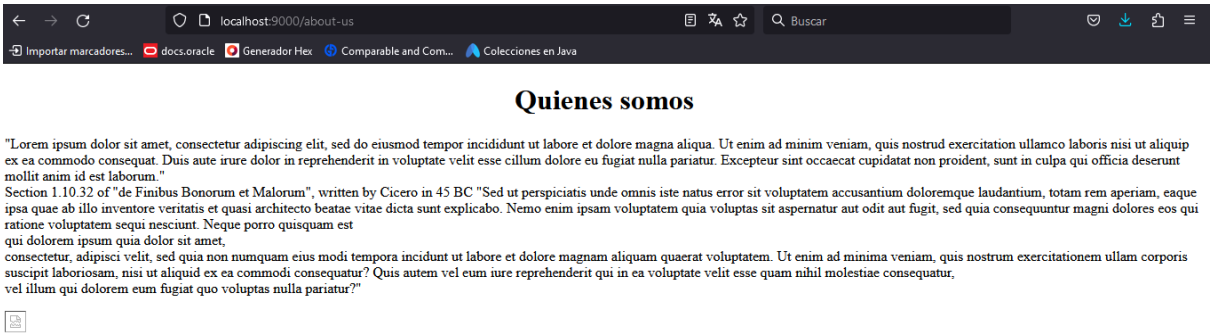
Resultado con /:



Resultado en index:



Resultado en about-us:



Resultado en contact:



Resultado en products:



## Ejercicio 3.2

3.2. Añade al proyecto anterior contenido dinámico pasándole información a las plantillas mediante un model y representándolo con etiquetas Thymeleaf. La página de inicio puede tener el año actual, por ejemplo ©2023 tomado de la fecha del sistema del servidor. La página de productos puede recibir la lista con los nombres de los productos que ofrece (por ahora puede ser un arraylist de String en el controlador, pero más adelante tomará los datos de la base de datos).

Respuesta:

Para añadir contenido dinámico, es necesario modificar los controller que gestionan dichas páginas, añadiéndoles un model que contendrá los valores del susodicho contenido dinámico. Así sería index:

```
@GetMapping({"/index", "/home", "/"})
public String indexView(Model model){
    //Model añade valores que podrán ser insertados en el html
    model.addAttribute(attributeName:"ayo","@"+Integer.toString(LocalDate.now().getYear()));
    /*addAttribute() contiene dos argumentos: el nombre de la variable que
    se usará en el html y el valor de dicha variable*/
    return "index";
}
```

Se modifica index.html para insertar la variable de model:

```
<body>
<h1 style="background-color: blue">Inicio</h1><br>
<div>Esta es una página de inicio</div><br>
<span th:text="{ayo}">*</span>
<p style="background-color: red">
    <a href="index.html">Index</a>
    <a href="quienes_somos.html">Quienes somos</a>
    <a href="productos.html">Productos</a>
    <a href="contacta.html">Contacta</a>
</p>

</body>
```

Y en el navegador quedaría así:

## Inicio

Esta es una página de inicio

©2023

¡Todos Quiéramos ser Productos! Contáctanos



En el caso de productos funcionaría de la siguiente manera:

```
@GetMapping("/products")
public String productoView(Model model){
    //Model añade valores que podrán ser insertados en el html
    ArrayList<String>lista=new ArrayList<>();//Se crea el arrayList
    lista.add("Barril");//Se le añaden valores
    lista.add("Caja");
    lista.add("Contenedor");
    lista.add("Remolque");
    lista.add("Almacen");

    model.addAttribute(attributeName:"elementos", lista);
    /*addAttribute() contiene dos argumentos: el nombre de la variable que
    se usará en el html y el valor de dicha variable
    En este caso, el nombre que usará en el html será elementos, y el valor
    que contendrá elementos será el arrayList lista*/
    return "productos";
}
```

Se modifica el html productos con th:each para que muestre cada valor del arraylist lista que contendrá elementos:

```
<body>
  <h1 style="text-align: center;">Productos</h1>
  <ul>
    <li>Sillas</li>
    <li>Mesas</li>
    <li>Armarios</li>
  </ul><br>
  <p th:each="elemento:${elementos}"><!--Por cada valor que contenga elementos, ese
  valor se llamará elemento y realizará una acción concreta-->
    <span th:text="${elemento}">*</span>
    <!--En este caso, mostrará el valor del elemento-->
  </p>
  
</body>
```

Resultado final:

## Productos

- Sillas
- Mesas
- Armarios

Barril

Caja

Contenedor

Remolque

Almacén



## Ejercicio 3.3

3.3. Haz una copia del proyecto anterior y sobre él: si en la página de inicio, se le pasa el parámetro usuario=XXX mostrará el mensaje de bienvenida con un texto personalizado para ese usuario, pero si no le pasa nada, será un mensaje genérico (Bienvenido XXX a nuestra web vs. Bienvenido a nuestra web). (Opcional) Haz una versión sin Optional, luego ponla entre comentarios y haz una segunda versión con Optional.

Respuesta:

Para poder cumplir con el enunciado del ejercicio, es necesario modificar el `@GetMapping` de `index` con `@RequestParam` para que la url pueda solicitar el dato del cliente:

```
@Controller
public class ControladorDos {
    @GetMapping({"/index", "/home", "/"})
    public String indexView(@RequestParam(name = "cliente", required=false) String usuario, Model
        /*@RequestParam recibe od argumentos, el nombre de la variable que se pedirá en
        la url y required=false que significa que no será obligatorio escribirlo en la url*/
        String usuario guardará el valor que se escriba en la url*/
        model.addAttribute(attributeName:"customer",usuario);
        /*addAttribute() contiene dos argumentos: el nombre de la variable que
        se usará en el html y el valor de dicha variable, en este caso el usuario
        que recibió en la url*/
        model.addAttribute(attributeName:"ayo", "@"+Integer.toString(LocalDate.now().getYear()));
        return "indexView";
    }
}
```

Luego se modifica el html con un `th:if` para que muestre dos mensajes distintos dependiendo del valor de `customer`



```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <div th:if="${customer=='XXX'}">Bienvenido XXX a nuestra web</div>
  <!--Si al introducir la url, customer es igual a XXX
  mostrará el mensaje de arriba-->
  <div th:unless="${customer=='XXX'}">Bienvenido a nuestra web</div>
  <!--Si al introducir la url, customer no es igual a XXX
  mostrará el mensaje de arriba-->
</head>

```

Resultado introduciendo XXX:



Resultado introduciendo otraCosa:



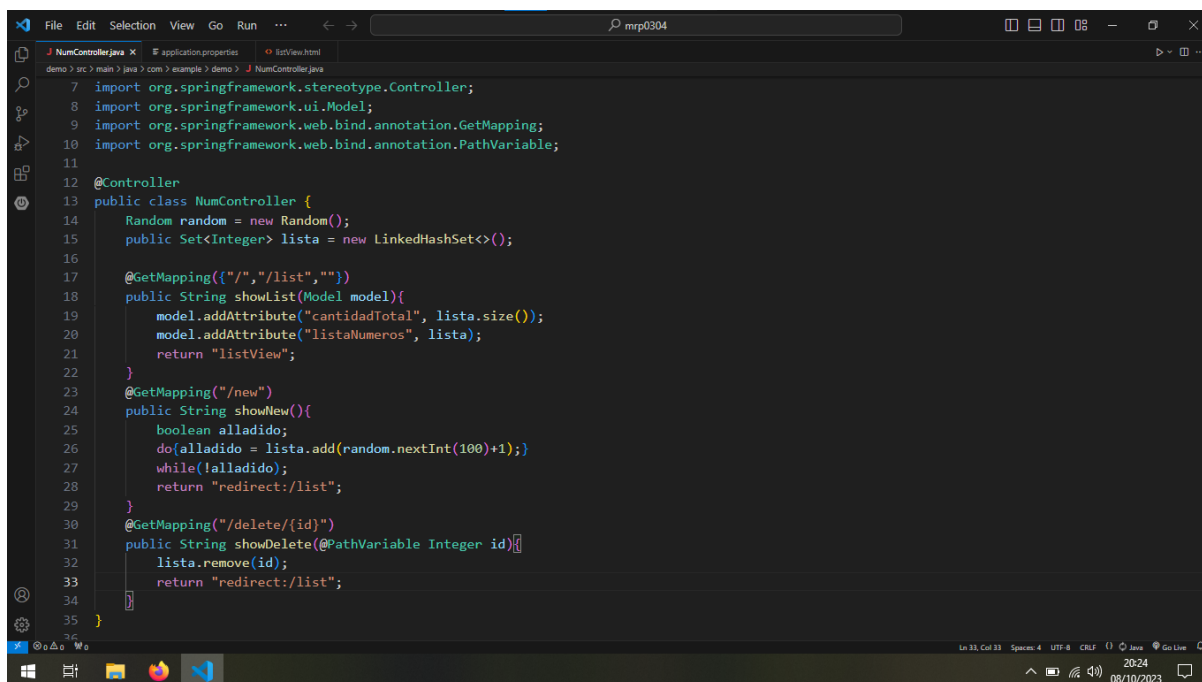
## Ejercicio 3.4

3.4. Implementa el ejemplo de los apuntes que genera números aleatorios en un nuevo proyecto. Simplemente debes crear el controlador y la plantilla con el código mostrado.

Respuesta:

Este ejercicio era tan simple como copiar y pegar.

Controller:



```
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11
12 @Controller
13 public class NumController {
14     Random random = new Random();
15     public Set<Integer> lista = new LinkedHashSet<>();
16
17     @GetMapping("/{}/list","")
18     public String showList(Model model){
19         model.addAttribute("cantidadTotal", lista.size());
20         model.addAttribute("listaNumeros", lista);
21         return "listView";
22     }
23     @GetMapping("/new")
24     public String showNew(){
25         boolean alladido;
26         do{alladido = lista.add(random.nextInt(100)+1);}
27         while(!alladido);
28         return "redirect:/list";
29     }
30     @GetMapping("/delete/{id}")
31     public String showDelete(@PathVariable Integer id){
32         lista.remove(id);
33         return "redirect:/list";
34     }
35 }
```

Html:

```

<!DOCTYPE html>
<html lang="es" xmlns:th="https://www.thymeleaf.org/">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Listado de numeros aleatorios</h1>
  <table border="1px">
    <thead><th>Numero</th><th>Operacion</th></thead>
    <tbody>
      <tr th:each="numero: ${listaNumeros}">
        <td th:text="${numero}">numero</td>
        <td><a th:href="@{/delete/{id}(id=${numero})}">delete</a></td>
      </tr>
    </tbody>
  </table>
  Total numeros:<span th:text="${cantidadTotal}">0</span><br>
  <a href="/new">Nuevo numero</a><br>
</body>
</html>

```

Resultado:

## Listado de numeros aleatorios

Numero	Operacion
--------	-----------

Total numeros:0

[Nuevo numero](#)

Añadiendo un número:

## Listado de numeros aleatorios

Numero	Operacion
90	<a href="#">delete</a>

Total numeros:1

[Nuevo numero](#)

Borrando un número:

# Listado de numeros aleatorios

Numero	Operacion
--------	-----------

Total numeros:0

[Nuevo numero](#)

## Ejercicio 3.5

3.5. Vuelve al proyecto del ejercicio 2.2 para trabajar con el pase de parámetros a los controladores en el path. Vamos a crear un nuevo controlador, con una nueva ruta base llamada /calculos que deberá realizar las siguientes tareas:

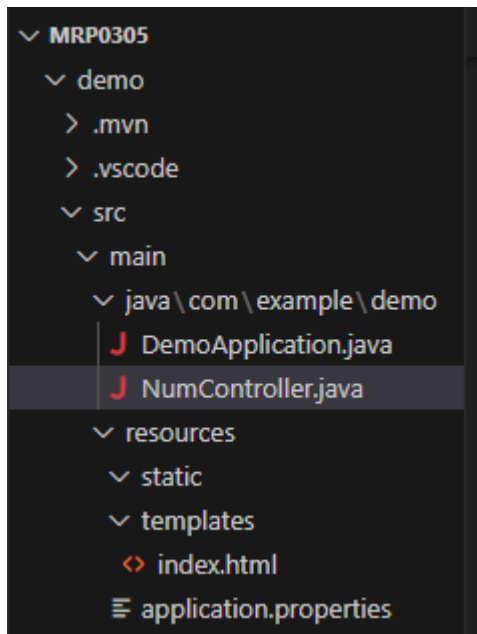
- a) Pasa la página index.html a la carpeta /templates y crea un controlador para ella.
- b) Mediante la URL: /calculos/primo?numero=X devolverá una página diciendo que el número X es primo o no. Por ahora, haz los cálculos en un método del propio controlador, aunque en capítulos siguientes los moveremos a otras capas.
- c) Modifica el apartado anterior, para que, si no introduce ningún número al que calcular si es primo o no, lo redirija a un controlador que trate el error. Este controlador por ahora simplemente mostrará la página de inicio.
- d) Mediante la URL /calculos/hipotenusa/X/Y devolverá una página con el valor de la hipotenusa correspondiente a los catetos X e Y. Si los números son negativos deberá redirigir al controlador que trate el error.
- e) Mediante la URL /calculos/sinRepetidos/X devolverá una página con X números aleatorios comprendidos entre 1 y 100, sin repetidos y ordenados ascendentemente. Obviamente X debe ser un número entero y estar comprendido entre 1 y 100, sino redirigirá a la página de error. Los mostrará en una tabla con una sola columna y cada número en una fila.

Pregunta: ¿Qué colección de Java es la más adecuada para que no contenga repetidos y estén ordenados?

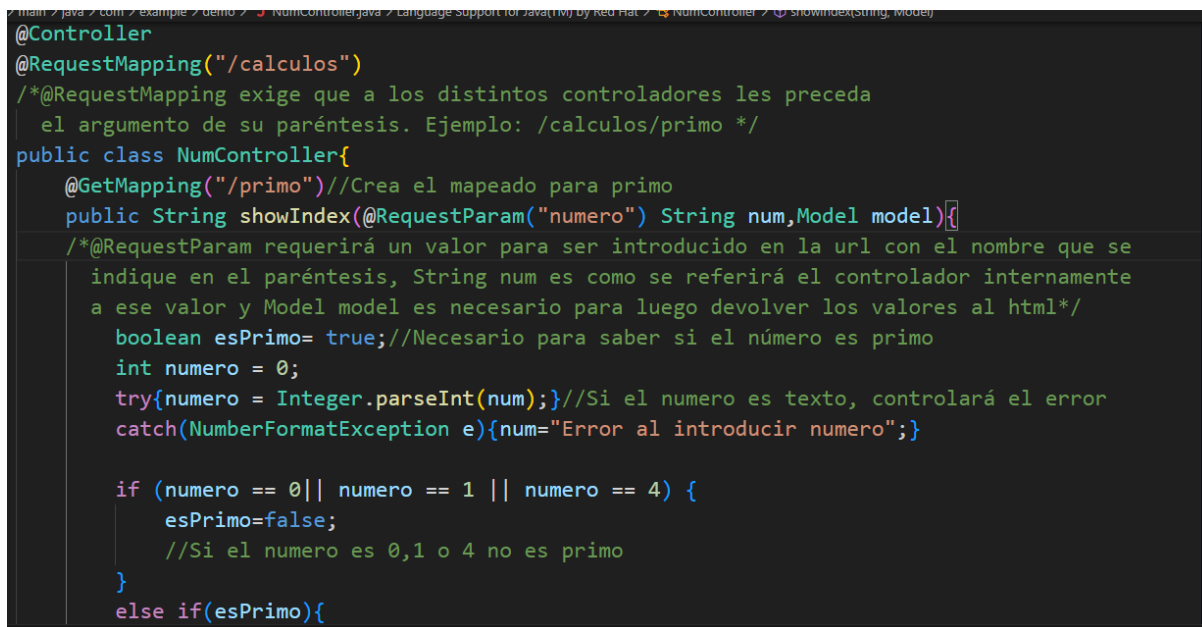
- f) Mediante la URL /calculos/divisores/X devolverá una lista con divisores del número X (cada número en un párrafo). Cada elemento de esa lista será un enlace, de forma que si el usuario clicla en uno de ellos mostrará los divisores del mismo. Ejemplo: para la URL /calculos/divisores/12 mostrará 1,2,3,6, 12 y podremos clicar por ejemplo en el 6 y mostrará 1,2,3,6 y así sucesivamente.
- g) Haz una versión del ejercicio anterior, pero con el parámetro X en la query de la URL, por ejemplo: /calculos/divisores?num=X. También las URL generadas para los divisores serán enlaces con el parámetro en la query.

Respuesta:

a) Pasa la página index.html a la carpeta /templates y crea un controlador para ella. Muevo la página index a templates:



Creo el controlador:



b) Mediante la URL: /calculos/primo?numero=X devolverá una página diciendo que el número X es primo o no. Por ahora, haz los cálculos en un método del propio controlador, aunque en capítulos siguientes los moveremos a otras capas.

El primer controlador maneja dicha situación. Primera parte:

```
@GetMapping("/primo")//Crea el mapeado para primo
public String showIndex(@RequestParam("numero") String num,Model model){
/*@RequestParam requerirá un valor para ser introducido en la url con el nombre que se
indique en el paréntesis, String num es como se referirá el controlador internamente
a ese valor y Model model es necesario para luego devolver los valores al html*/
    boolean esPrimo= true;//Necesario para saber si el número es primo
    int numero = 0;
    try{numero = Integer.parseInt(num);}//Si el numero es texto, controlará el error
    catch(NumberFormatException e){num="Error al introducir numero";}

    if (numero == 0 || numero == 1 || numero == 4) {
        esPrimo=false;//Si el numero es 0,1 o 4 no es primo
    }
    else if(esPrimo){
        for (int i = 2; i < numero / 2; i++) {
            if (numero % i == 0)esPrimo=false;
            // Si es divisible por cualquiera de estos números, no es primo
        }
    }
}
```

En la captura se comienza definiendo el mapeado /primo con @GetMapping, después defino que argumentos usará showIndex(el parámetro que recogerá en la url, la designación que recibirá dicho parámetro y model para insertar valores en el html). esPrimo controlará si el número final es primo o no y numero guardará el número que se introdujo en la url,

try-catch comprueba si el usuario introdujo un número u otra cosa. Si introdujo un valor no numérico convertirá el valor introducido por el usuario en un mensaje de error.

El primer if es la primera de las comprobaciones para saber si el número es primo; si es igual a 0,1 o 4 no es primo.

El else if comprueba si es primo dividiéndolo entre el resto de números hasta llegar al que introdujo el usuario. Si ninguno de estos dos if cambia su valor esPrimo se mantendrá true y por tanto, el número es primo.

Segunda parte:

```

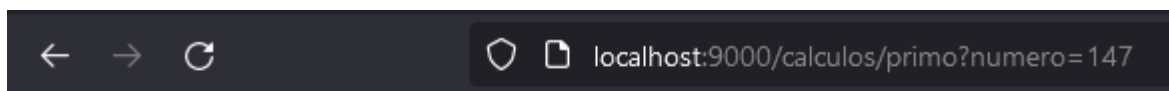
    for (int i = 2; i < numero / 2; i++) {
        if (numero % i == 0) esPrimo=false;
        // Si es divisible por cualquiera de estos números, no es primo
    }
}
if(num=="Error al introducir numero"){
    //Si se introdujo texto en la url, devolverá un error
    model.addAttribute(attributeName:"primo", attributeValue:false);
    model.addAttribute(attributeName:"valor", num);//Texto que contiene el mensaje de error
}
else{
    //Si todo esta en orden, devolverá los valores correctos
    model.addAttribute(attributeName:"primo", esPrimo);
    model.addAttribute(attributeName:"valor", numero);
}
return "index";
}

```

Si num es igual al mensaje de error, en el html insertará que el valor no es primo y en la ubicación del número añadirá el mensaje de error, sino insertará el valor y si es primo o no return index devuelve la página html index

Resultado:

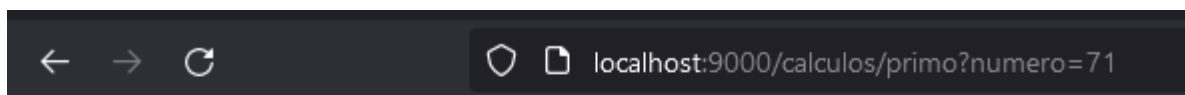
- Introduciendo de número 147



## Hola Mundo

El numero 147 no es primo

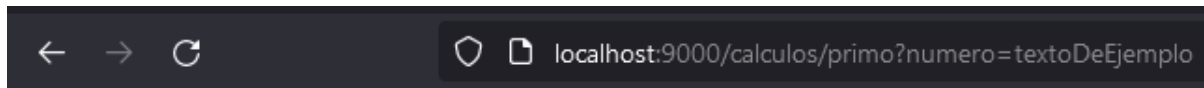
- Introduciendo de número 71



## Hola Mundo

El numero 71 es primo

- Introduciendo de número textoDeEjemplo



## Hola Mundo

El numero [Error al introducir numero] no es primo

c) Modifica el apartado anterior, para que, si no introduce ningún número al que calcular si es primo o no, lo redirija a un controlador que trate el error. Este controlador por ahora simplemente mostrará la página de inicio.

Primero cree un controlador para manejar el error en cuestión:

```
@Controller
public class ErrorController {
    @GetMapping("/errorNum")//url a la que redirigirá si se produce el error
    public String showError(){
        return "index";//Devuelve index como establece el enunciado
    }
}
```

Luego compruebo con un if si el valor es nulo para saber si redirigir al usuario o no al error:

```
if(num==null) return "redirect:/errorNum";/*Si el usuario no introdujo valor en la url
redirige al controlador del error*/
```

d) Mediante la URL /calculos/hipotenusa/X/Y devolverá una página con el valor de la hipotenusa correspondiente a los catetos X e Y. Si los números son negativos deberá redirigir al controlador que trate el error.

Para hacer este ejercicio, cree un controlador que calcula la hipotenusa:



```

@GetMapping("/hipotenusa/{X}/{Y}")//url que recibirá dos números X e Y
public String showHipotenusa(@PathVariable int X,@PathVariable int Y, Model model){
    /*Debido al requerimiento del enunciado de mostrar en la url solo el valor de X
    e Y es obligatorio utilizar @PathVariable
    showHipotenusa recibe tres argumentos, la X que siempre recibirá el mismo nombre que
    en la url si se usa @PathVariable, Y que funciona igual que X y model que
    añadirá los valores al html */
    if(X<0||Y<0) return "redirect:/errorNum";
    /*Si X o Y son números negativos, redirigirá al controlador del error*/
    model.addAttribute(attributeName:"resultado", Math.pow(X,b:2)+Math.pow(Y,b:2));
    /*Fórmula de la hipotenusa: es la suma de los cuadrados de X e Y
    model.addAttribute() los envía al html*/
    return "index";
}

```

@GetMapping recibe la dirección /hipotenusa junto con dos números X e Y. @PathVariable sirve para recibir X e Y de la url y convertirlos en argumentos de la función (razón por la que es necesario usar @PathVariable). El if comprueba si son números negativos, y en caso afirmativo, redirigirá al controlador del error.

Dentro del model.addAttribute() le asigno el nombre resultado a la fórmula de la hipotenusa, que consiste en la suma de los cuadrados de ambos

html modificado:

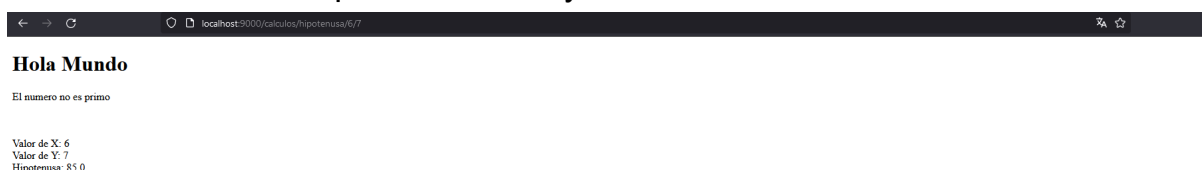
```

<body>
    <h1>Hola Mundo</h1>

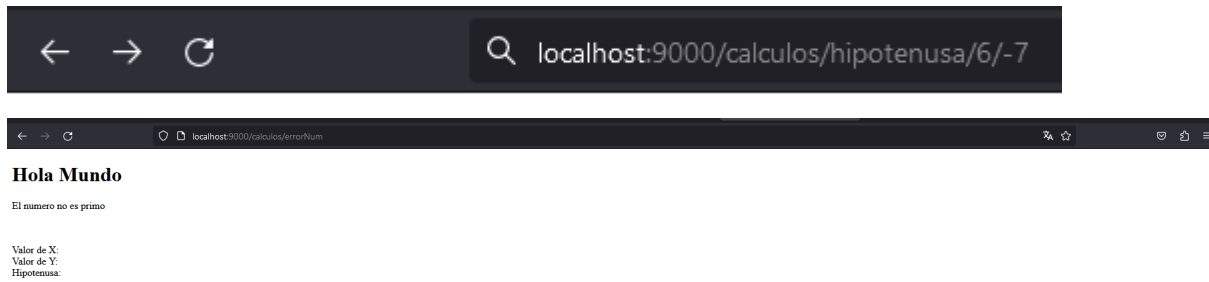
    <p th:if="${(primo)}">El numero <span th:text="${valor}"></span> es primo</p>
    <p th:unless="${(primo)}">El numero <span th:text="${valor}"></span> no es primo</p>
    <br><br>
    <span>Valor de X: <span th:text="${X}"></span></span><br>
    <span>Valor de Y: <span th:text="${Y}"></span></span><br>
    <span>Hipotenusa: <span th:text="${resultado}"></span></span><br>
    <!--Cada uno de estos span muestra el valor asignado en el controlador con
    model.addAttribute-->
</body>
</html>

```

Resultado final con hipotenusa de 6 y 7:



Introduciendo un número negativo:



e) Mediante la URL `/calculos/sinRepetidos/X` devolverá una página con X números aleatorios comprendidos entre 1 y 100, sin repetidos y ordenados ascendentemente. Obviamente X debe ser un número entero y estar comprendido entre 1 y 100, sino redirigirá a la página de error. Los mostrará en una tabla con una sola columna y cada número en una fila.  
Pregunta: ¿Qué colección de Java es la más adecuada para que no contenga repetidos y estén ordenados?

Empecé creando un controlador:

```

@GetMapping("/calculosSinRepetidos/{X}")
public String showCalculosSinRepetidos(@PathVariable Integer X, Model model){
    //@PathVariable recibe una X como valor
    if(X<1||X>100) return "redirect:/errorNum";
    Random r = new Random();
    TreeSet<Integer> resultado = new TreeSet<>();
    int adjudicar=0;
    for(int i = 0; i<X;i++){
        adjudicar=r.nextInt(100)+1;
        while(resultado.contains(adjudicar)){adjudicar=r.nextInt(100)+1;}
        //Evita que haya iteraciones en las que no se introduzcan números
        resultado.add(adjudicar);
    }
    model.addAttribute(attributeName:"lista", resultado);
    return "index";
}

```

Como el enunciado establece que en la url solo puede figurar el valor de X, se usa `@PathVariable`. Si el usuario introduce un número incorrecto redirigirá al controlador del error. En caso contrario, el programa comenzará creando un `TreeSet` (colección más adecuada para ordenar elementos, y si fuese necesario, definir el criterio de orden). A continuación, un bucle `for` iterará todas las veces que figure en la url, añadiendo números al azar entre 0 y 100. El bucle `while` evita que una iteración del bucle `for` pueda terminar con un número repetido en el `TreeSet`. Sin este bucle, podría haber iteraciones en las que no se añadiesen valores nuevos, porque `TreeSet` no admite valores duplicados. Por último, `model.addAttribute()` hace accesible dicho valor en el html

Html:

```

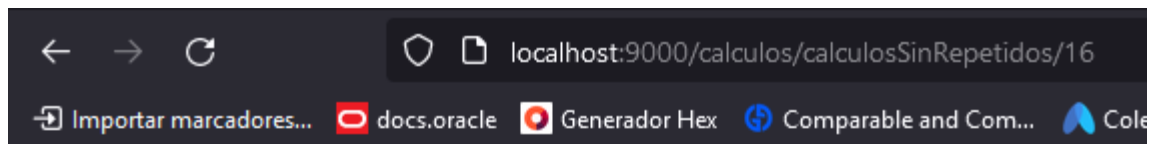
<div th:each="numero:${lista}">
    <span th:text="${numero}"></span><br>
</div>

```

Con un bucle `th:each` por cada elemento `numero` en `lista` mostrará el valor que contiene dicho número

Resultado:

- Con un valor correcto:



# Hola Mundo

El numero no es primo

Valor de X: 16

Valor de Y:

Hipotenusa:

5

7

17

26

27

37

42

48

49

56

73

75

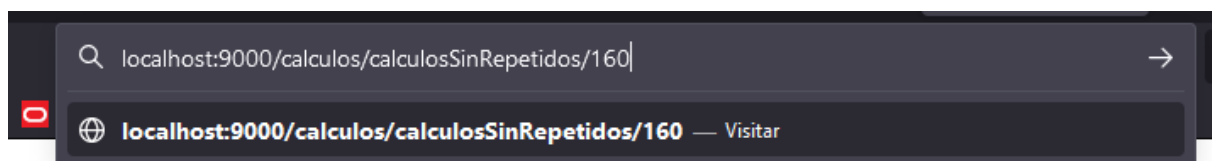
86

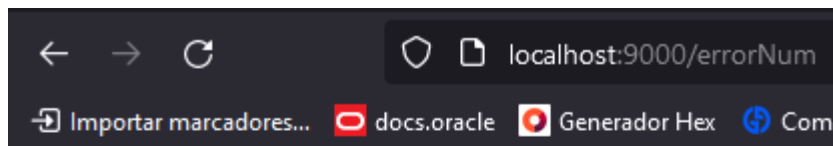
87

96

97

- Con un valor incorrecto:





# Hola Mundo

El numero no es primo

Valor de X:  
Valor de Y:  
Hipotenusa:

f) Mediante la URL /calculos/divisores/X devolverá una lista con divisores del número X (cada número en un párrafo). Cada elemento de esa lista será un enlace, de forma que si el usuario clicla en uno de ellos mostrará los divisores del mismo. Ejemplo: para la URL /cálculos/divisores/12 mostrará 1,2,3,6, 12 y podremos clicar por ejemplo en el 6 y mostrará 1,2,3,6 y así sucesivamente.

En primer lugar, genere un controlador personalizado:

```
@GetMapping("/divisores/{X}")
public String showDivisores(@PathVariable int X, Model model){
    ArrayList<Integer> resultado = new ArrayList<>();

    for(int i = 1; i<X;i++){
        if(X%i==0)resultado.add(i); //Si el número i es divisor exacto de 0, se añadirá
        //al resultado final
    }
    model.addAttribute(attributeName:"divisores",resultado);
    return "index";
}
```

El enunciado establece que en la url solo puede existir el valor de X, por lo que es obligatorio usar @PathVariable. El bucle for comprobará si todos los números de 1 al anterior a X son divisores exactos de X y los almacenará en un arraylist que model.addAttribute() hará disponible en el html

Html:

```
<div th:each="divisor:${divisores}">
    <a th:href="@{/calculos/divisores/{X}(X=${divisor})}" th:text="${divisor}"></a>
</div>
```

th:each crea un bucle que por cada divisor exacto de X mostrará cada uno de esos divisores; a su vez, cada divisor contendrá un enlace con los divisores exactos de dicho divisor

Resultado usando de ejemplo 30:



# Hola Mundo

El numero no es primo

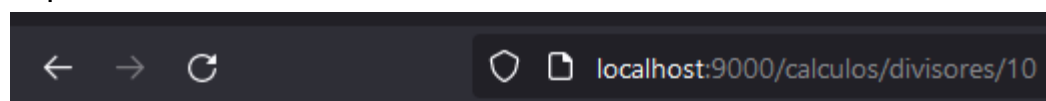
Valor de X: 30

Valor de Y:

Hipotenusa:

1  
2  
3  
5  
6  
10  
15

Al pulsar en 10:



# Hola Mundo

El numero no es primo

Valor de X: 10

Valor de Y:

Hipotenusa:

1  
2  
5

g) Haz una versión del ejercicio anterior, pero con el parámetro X en la query de la URL, por ejemplo: /calculos/divisores?num=X. También las URL generadas para los divisores serán enlaces con el parámetro en la query.

Este ejercicio consistió básicamente en copiar texto del anterior apartado y cambiar nombres de variables y del mapping, además de la ruta del html en cuestión:

Mapping creado:

```
@GetMapping("/divisoresDos")//Mismo mapping con @RequestParam
public String showDivisoresDos(@RequestParam int X,Model model){
    ArrayList<Integer> resultado = new ArrayList<>();

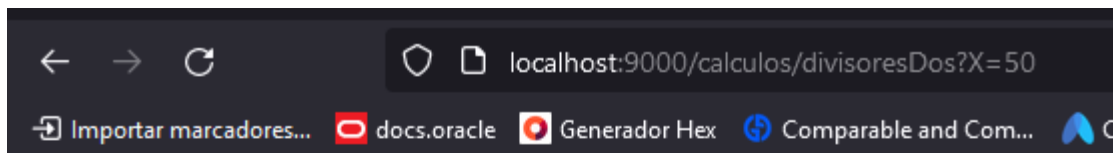
    for(int i = 1; i<X;i++){
        if(X%i==0)resultado.add(i);//Si el número i es divisor exacto de 0, se añadirá
        //al resultado final
    }
    model.addAttribute(attributeName:"divisoresDos",resultado);
    return "index";
}
```

Html:

```
<div th:each="divisorDos:${divisoresDos}">
    <a th:href="@{/calculos/divisoresDos(X=${divisorDos})}"><span th:text="${divisorDos}"></span>
</div>
```

A diferencia de @PathVariable, en la ruta del @RequestMapping no se pone una barra al añadir la variable

Resultado al introducir 50:



## Hola Mundo

El numero no es primo

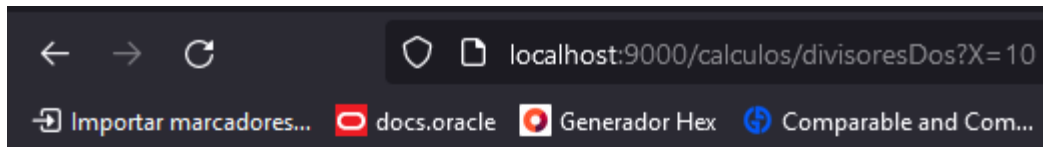
Valor de X:

Valor de Y:

Hipotenusa:

1  
2  
5  
10  
25

Resultado al hacer click en 10:



# Hola Mundo

El numero no es primo

Valor de X:

Valor de Y:

Hipotenusa:

1

2

5