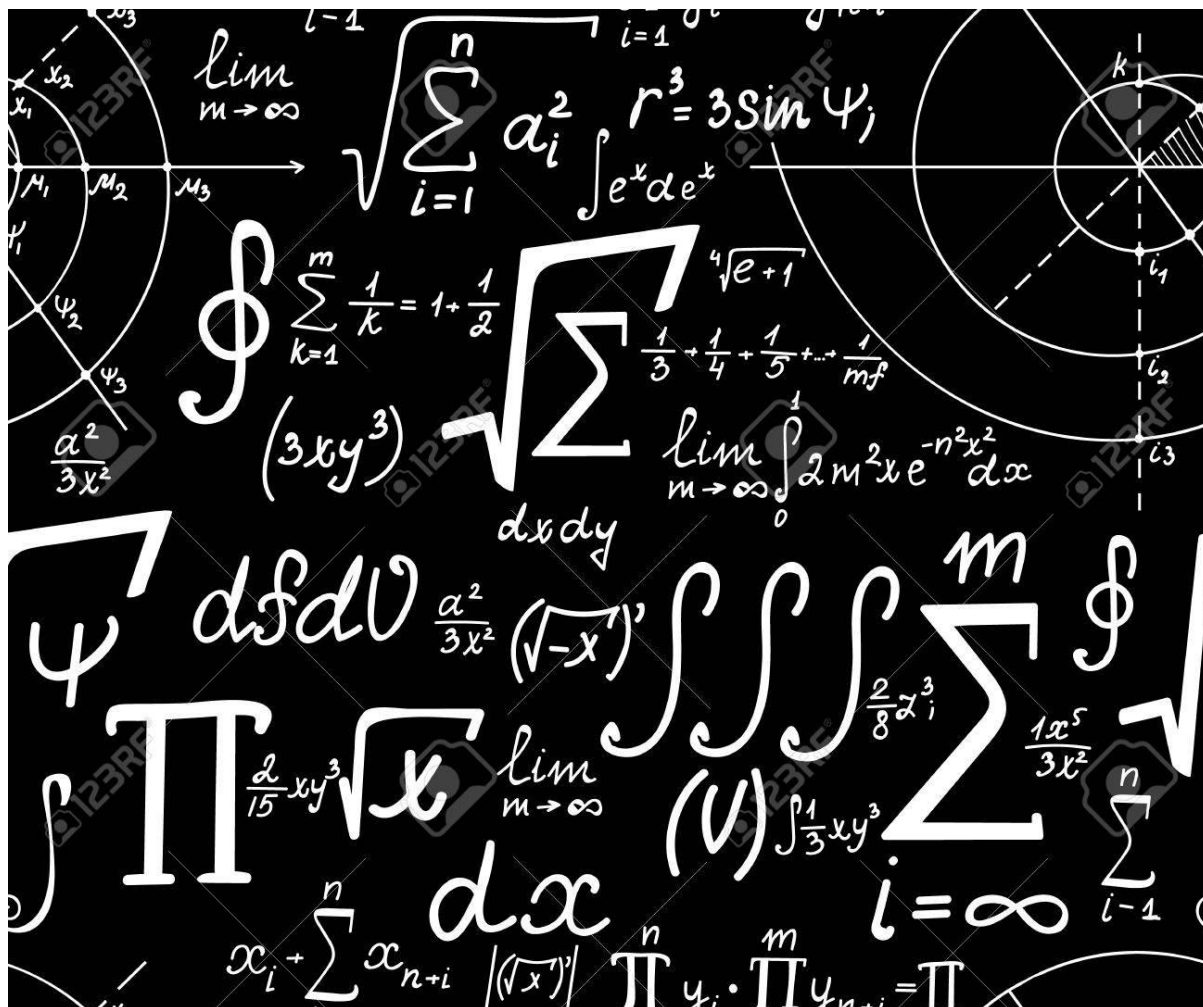


# tema 4

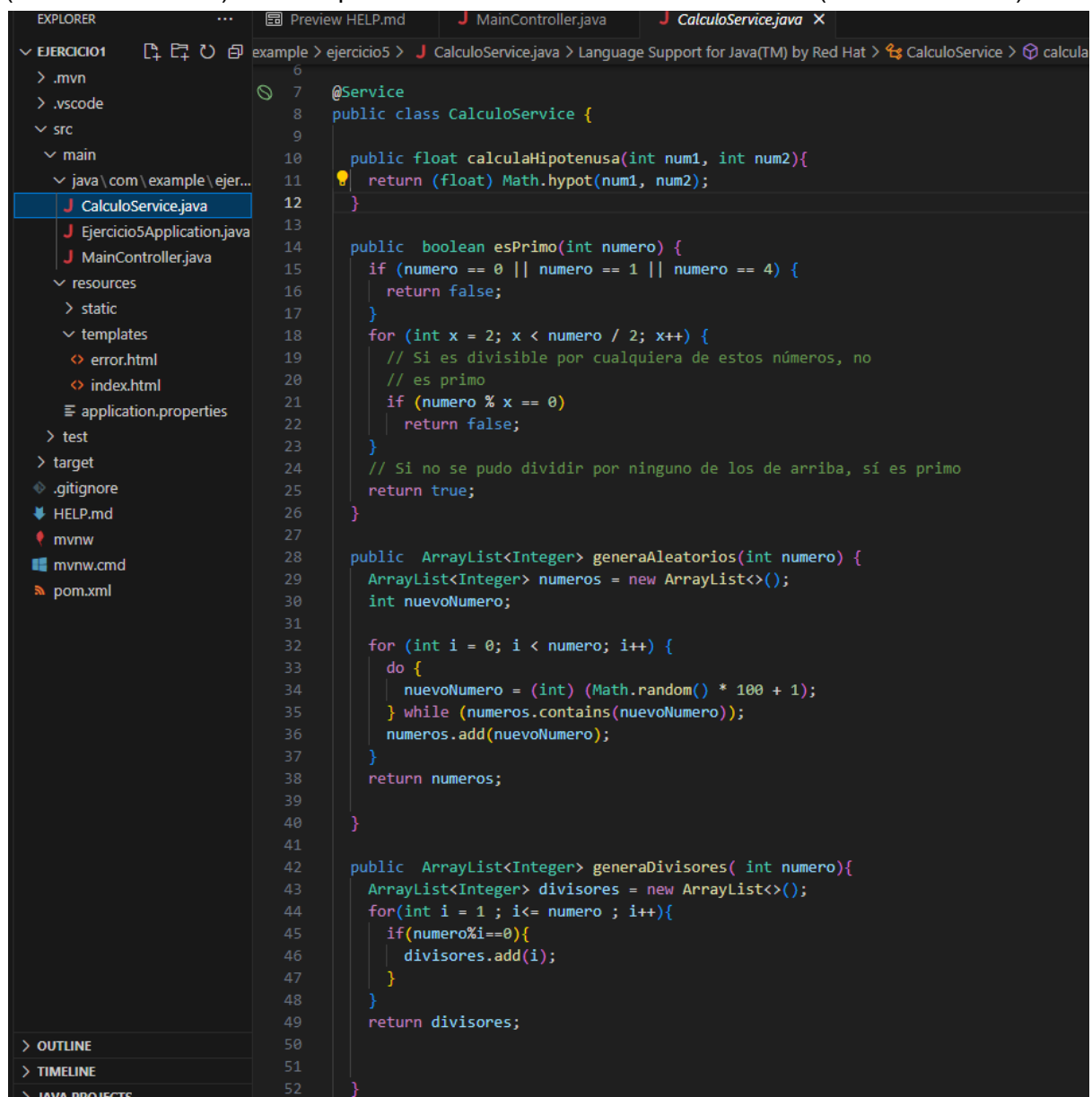


# Índice

<b>tema 4</b>	<b>1</b>
Índice	2
4.1.	3
4.2.	4
4.3.	8
4.4.	9

#### 4.1.

Haz una copia del proyecto del ejercicio 3.5 de temas anteriores y pasa toda la lógica de negocio (todos los cálculos) a una capa de servicio. Puede ser una sola clase (CalculosService).

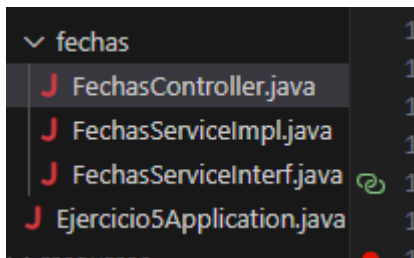


```
6
7 @Service
8 public class CalculoService {
9
10     public float calculaHipotenusa(int num1, int num2){
11         return (float) Math.hypot(num1, num2);
12     }
13
14     public boolean esPrimo(int numero) {
15         if (numero == 0 || numero == 1 || numero == 4) {
16             return false;
17         }
18         for (int x = 2; x < numero / 2; x++) {
19             // Si es divisible por cualquiera de estos números, no
20             // es primo
21             if (numero % x == 0)
22                 return false;
23         }
24         // Si no se pudo dividir por ninguno de los de arriba, sí es primo
25         return true;
26     }
27
28     public ArrayList<Integer> generaAleatorios(int numero) {
29         ArrayList<Integer> numeros = new ArrayList<>();
30         int nuevoNumero;
31
32         for (int i = 0; i < numero; i++) {
33             do {
34                 nuevoNumero = (int) (Math.random() * 100 + 1);
35             } while (numeros.contains(nuevoNumero));
36             numeros.add(nuevoNumero);
37         }
38         return numeros;
39     }
40
41
42     public ArrayList<Integer> generaDivisores( int numero){
43         ArrayList<Integer> divisores = new ArrayList<>();
44         for(int i = 1 ; i<= numero ; i++){
45             if(numero%i==0){
46                 divisores.add(i);
47             }
48         }
49         return divisores;
50     }
51
52 }
```

Creo una clase llamada CalculoService que va a contener toda la lógica de los cálculos de la aplicación e implemento los métodos ya creados anteriormente.

## 4.2.

Haz una copia del proyecto anterior y crea un área nueva para trabajar con fechas (nuevo controlador, nuevas vistas y nuevo servicio) con las siguientes características:



Creo un fichero llamado fecha que contendrá todo lo relativo a ellas.

a) La url base de esta parte será /fechas.

```
@Controller
@RequestMapping("/fechas")
public class FechasController {
```

Para fijar la url base como /fechas uso un RequestMapping

b) Si le pasamos en la parte query una fecha, mostrará los días transcurridos desde el 1 de enero del mismo año. Las fechas se pasan siempre en formato YYYY-MM-DD.

```
@Controller
@RequestMapping("/fechas")
public class FechasController {

    @Autowired
    FechasServiceInterf fechasService;

    @GetMapping("/{fecha}")
    public String showFechas(@PathVariable String fecha, Model model) {
        model.addAttribute(attributeName:"diferencia", fechasService.muestraDiasTranscurridosPrimeroAño(fecha));
        return "fechas";
    }
}
```

Creo un GetMapping que si le introduces una fecha usará la función del servicio muestraDiasTranscurridosPrimerAño() y se la pasa al modelo.

# Fechas

122

Introducimos el link correspondiente y vemos que funciona perfectamente.

c) Si le pasamos en la parte query dos fechas, mostrará los días comprendidos entre ambas fechas.

```
@GetMapping("/{fecha1}/{fecha2}")
public String showDiferenciaFechas(@PathVariable String fecha1, @PathVariable String fecha2, Model model) {
    model.addAttribute(attributeName:"diferencia", fechasService.muestraDiferenciaDias(fecha1, fecha2));
    return "fechas";
}
```

Creo un GetMapping para introducir ambas fechas y llama a la funcion muestraDiferenciasDias que está definida en el servicio.

# Fechas

243

Al introducir las dos fechas en el link muestra la diferencia de días correctamente.

d) Si no se pasan ninguna fecha, mostrará los días transcurridos entre el 1 de enero y hoy.

```
@GetMapping("/{", "/" })
public String showDiferenciaFechaHoy(Model model) {
    model.addAttribute(attributeName:"diferencia", fechasService.comparaFechaHoy());
    return "fechas";
}
```

En este get mapping recojo la ruta “/fechas” y “fechas/”

← → ↻ ⓘ localhost:9000/fechas

# Fechas

288

Funciona correctamente.

e) Si pasamos /bisiesto/fecha1 mostrará si fecha1 pasada en el path es de un año bisiesto.

```
@GetMapping({ "/bisiesto/{fecha}" })
public String showBisiesto(@PathVariable String fecha, Model model) {
    String mensaje = "";
    if (fechasService.obtenBisiestoEnBaseString(fecha)) {
        mensaje = "En le fecha " + fecha + " el año es bisiesto";
    } else {
        mensaje = "En le fecha " + fecha + " el año no es bisiesto";
    }
    model.addAttribute(attributeName:"diferencia", mensaje);
    return "fechas";
}
```

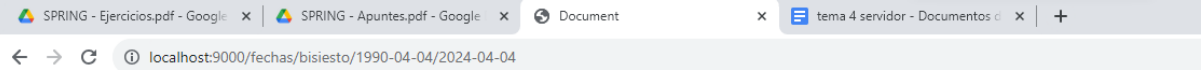
← → ↻ ⓘ localhost:9000/fechas/bisiesto/2020-04-04

# Fechas

**En le fecha 2020-04-04 el año es bisiesto**

f) Si le pasamos /bisiesto/año1/año2 mostrará los años bisiestos comprendidos entre ambos años.

```
@GetMapping({ "/bisiesto/{fecha1}/{fecha2}" })
public String showBisiesto(@PathVariable String fecha1, @PathVariable String fecha2, Model model) {
    String respuesta = "Los años bisiestos entre " + fecha1 + " y " + fecha2 + " son "
        + fechasService.muestraAñosBisiestosEntreDosFechas(fecha1, fecha2);
    model.addAttribute(attributeName:"diferencia", respuesta);
    return "fechas";
}
```

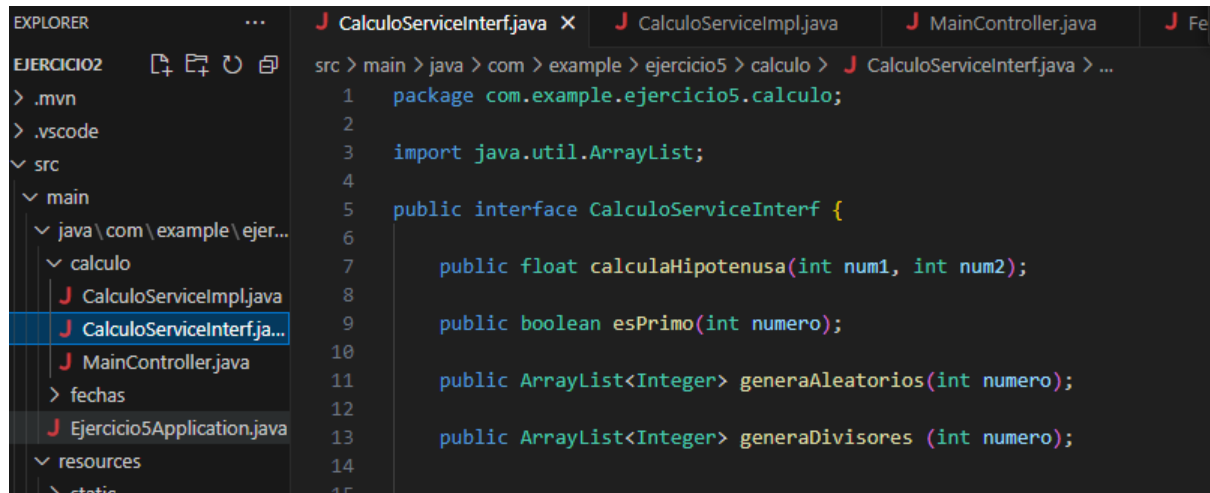


## Fechas

**Los años bisiestos entre 1990-04-04 y 2024-04-04 son 1992 1996 2000 2004 2008 2012 2016 2020**

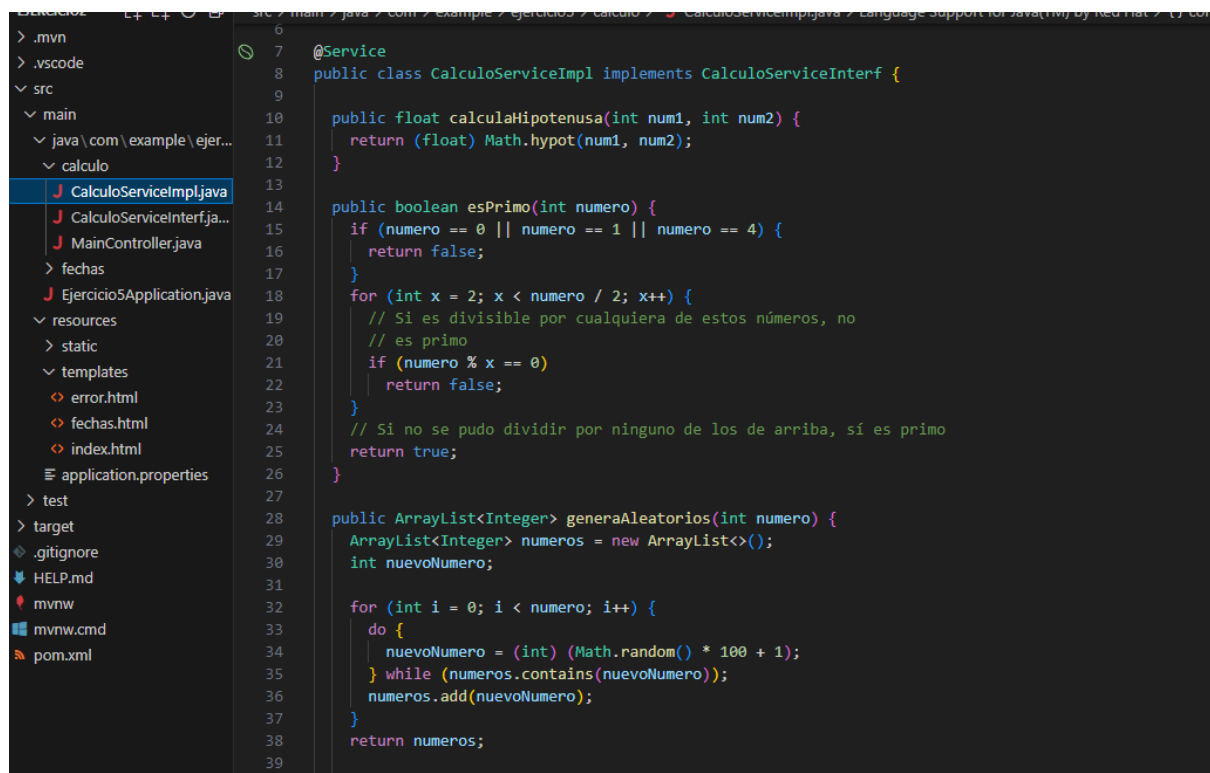
### 4.3.

Haz una copia del proyecto anterior y crea una interfaz `CalculosService` pasando la clase anterior a llamarse `CalculosServiceImpl` y lo mismo con el servicio de cálculo de fechas.



The screenshot shows the VS Code Explorer on the left with the project structure. The file `CalculoServiceInterf.java` is selected. The main editor shows the content of this file:

```
src > main > java > com > example > ejercicio5 > calculo > CalculoServiceInterf.java > ...
1  package com.example.ejercicio5.calculo;
2
3  import java.util.ArrayList;
4
5  public interface CalculoServiceInterf {
6
7      public float calculaHipotenusa(int num1, int num2);
8
9      public boolean esPrimo(int numero);
10
11     public ArrayList<Integer> generaAleatorios(int numero);
12
13     public ArrayList<Integer> generaDivisores (int numero);
14
15 }
```



The screenshot shows the VS Code Explorer on the left with the project structure. The file `CalculoServiceImpl.java` is selected. The main editor shows the content of this file:

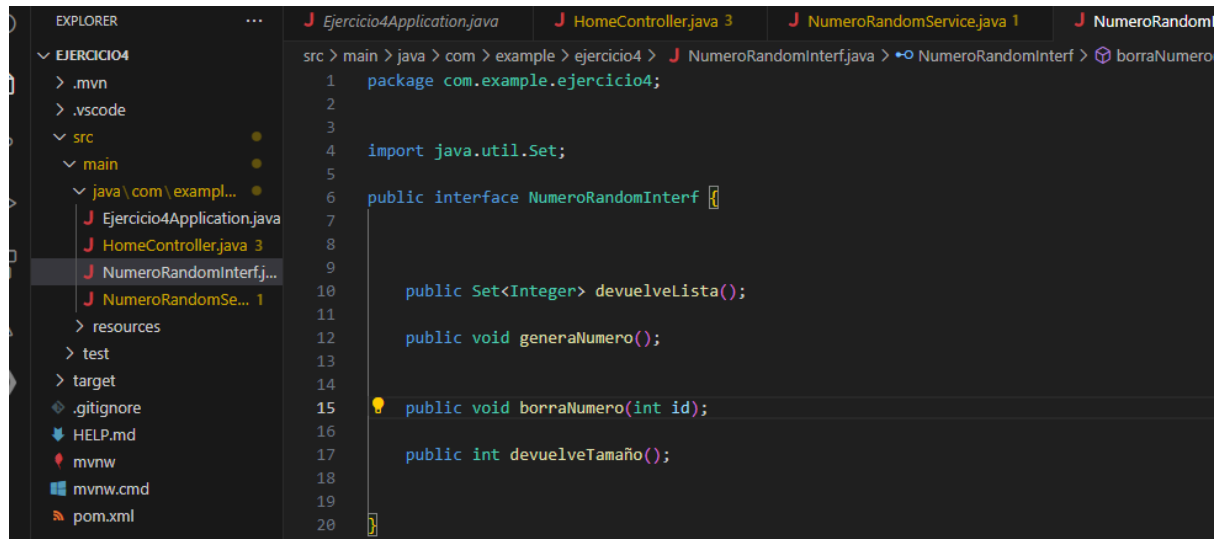
```
src > main > java > com > example > ejercicio5 > calculo > CalculoServiceImpl.java > language support for java(rn) by Red Hat > (1) con
6
7  @Service
8  public class CalculoServiceImpl implements CalculoServiceInterf {
9
10     public float calculaHipotenusa(int num1, int num2) {
11         return (float) Math.hypot(num1, num2);
12     }
13
14     public boolean esPrimo(int numero) {
15         if (numero == 0 || numero == 1 || numero == 4) {
16             return false;
17         }
18         for (int x = 2; x < numero / 2; x++) {
19             // Si es divisible por cualquiera de estos números, no
20             // es primo
21             if (numero % x == 0)
22                 return false;
23         }
24         // Si no se pudo dividir por ninguno de los de arriba, sí es primo
25         return true;
26     }
27
28     public ArrayList<Integer> generaAleatorios(int numero) {
29         ArrayList<Integer> numeros = new ArrayList<>();
30         int nuevoNumero;
31
32         for (int i = 0; i < numero; i++) {
33             do {
34                 nuevoNumero = (int) (Math.random() * 100 + 1);
35             } while (numeros.contains(nuevoNumero));
36             numeros.add(nuevoNumero);
37         }
38         return numeros;
39     }
40 }
```



#### 4.4.

Haz una copia del proyecto 3.4 pasando la lógica de negocio a una capa de servicio que contenga

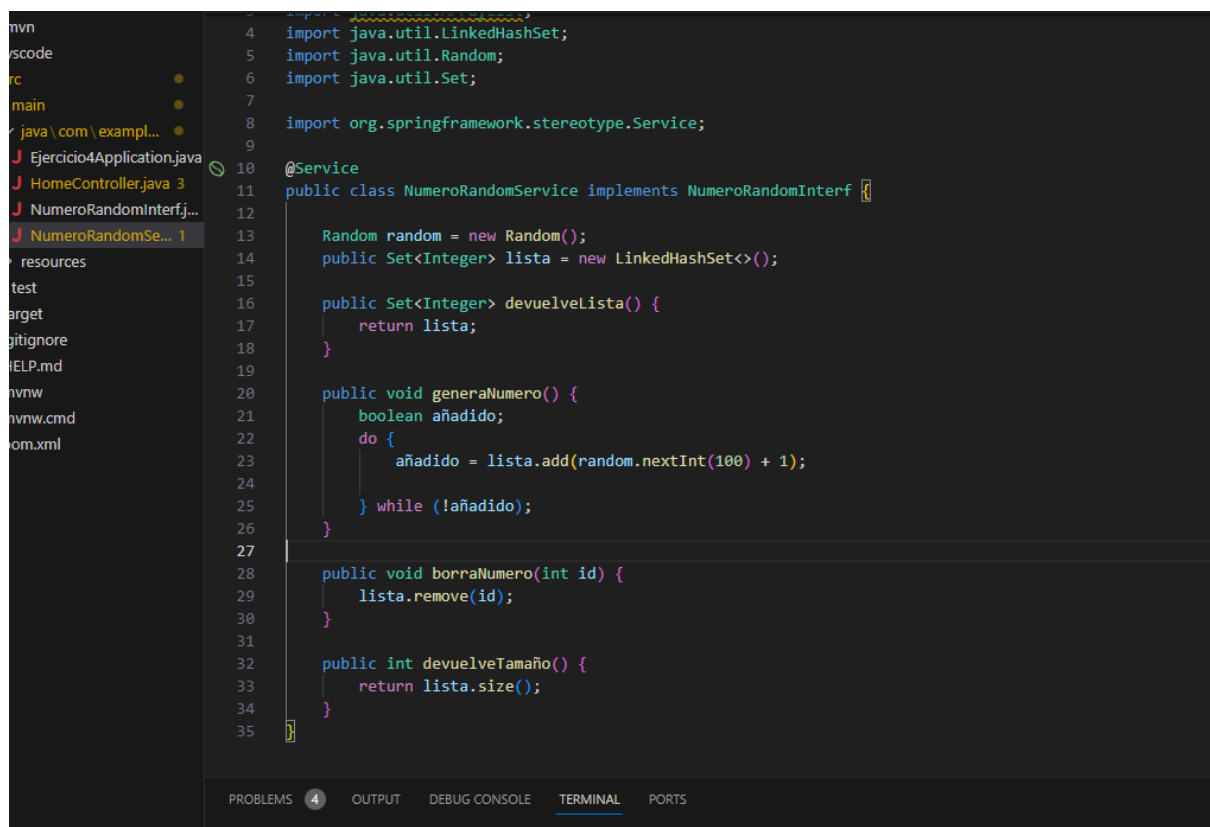
la colección con los números aleatorios y los métodos para añadir nuevos números, eliminar números existentes, devolver los elementos de la colección y devolver la cantidad de elementos de la colección.



The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the project structure: EJERCICIO4 > src > main > java > com > example > ejercicio4. The Editor shows the file NumeroRandomInterf.java with the following code:

```
1 package com.example.ejercicio4;
2
3
4 import java.util.Set;
5
6 public interface NumeroRandomInterf {
7
8
9
10     public Set<Integer> devuelveLista();
11
12     public void generaNumero();
13
14
15     public void borraNumero(int id);
16
17     public int devuelveTamaño();
18
19
20 }
```

Creo una interfaz que va a definir los métodos que implementará la clase NumeroRandomService.



The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the project structure: EJERCICIO4 > src > main > java > com > example > ejercicio4. The Editor shows the file NumeroRandomService.java with the following code:

```
1 import java.util.*;
2
3 import java.util.LinkedHashSet;
4 import java.util.Random;
5 import java.util.Set;
6
7 import org.springframework.stereotype.Service;
8
9
10 @Service
11 public class NumeroRandomService implements NumeroRandomInterf {
12
13     Random random = new Random();
14     public Set<Integer> lista = new LinkedHashSet<>();
15
16     public Set<Integer> devuelveLista() {
17         return lista;
18     }
19
20     public void generaNumero() {
21         boolean añadido;
22         do {
23             añadido = lista.add(random.nextInt(100) + 1);
24         } while (!añadido);
25     }
26
27
28     public void borraNumero(int id) {
29         lista.remove(id);
30     }
31
32     public int devuelveTamaño() {
33         return lista.size();
34     }
35 }
```

Creo la clase NumeroRandomService que implementa la interfaz creada anteriormente y defino cada método introduciendo la lógica de las funciones para así separar las capas.

```
@Controller
public class HomeController {

    @Autowired
    NumeroRandomService numeroRandomService;

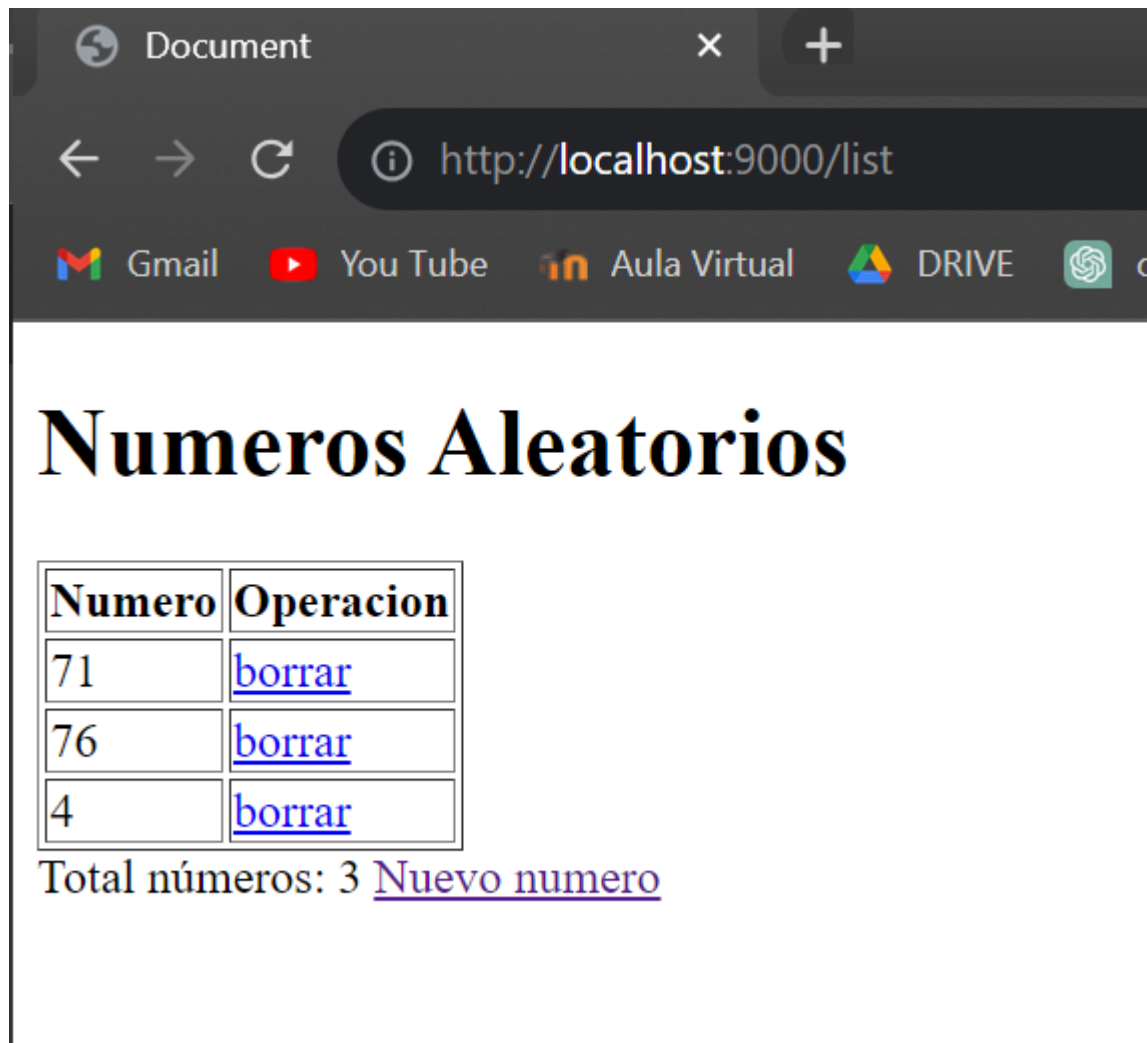
    @GetMapping({ "/", "/list" })
    public String showNumber(Model model) {
        model.addAttribute(attributeName:"cantidadTotal", numeroRandomService.devuelveTamaño());
        model.addAttribute(attributeName:"listaNumeros", numeroRandomService.devuelveLista());
        return "index";
    }

    @GetMapping("/new")
    public String addNumber() {
        numeroRandomService.generaNumero();
        return "redirect:/list";
    }

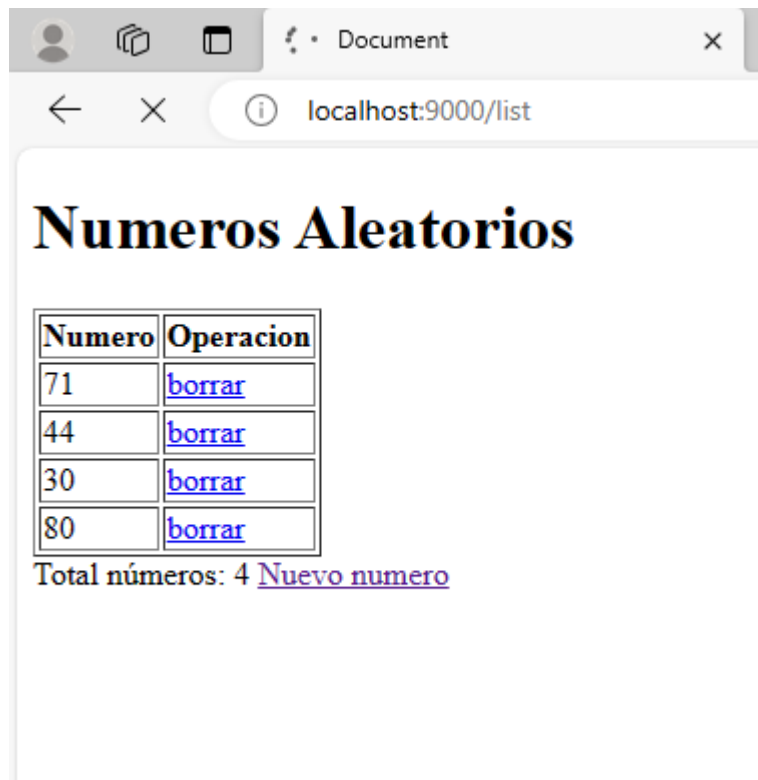
    @GetMapping("/delete/{id}")
    public String deleteNumber(@PathVariable Integer id) {
        numeroRandomService.borraNumero(id);
        return "redirect:/list";
    }
}
```

Creo un `@Autowired` para poder usar el servicio y quito toda la lógica anterior y uso las funciones del servicio.

Una vez que funcione la aplicación, prueba a ejecutarla en distintos navegadores a la vez. Explica en el PDF de soluciones de este tema qué ocurre, por qué y cómo solucionarlo.



Aquí he abierto la app con google chrome y he añadidos algunos números de forma aleatoria.



Al abrirlo en edge vemos como la aplicación sigue conteniendo los números ya creados en vez de empezar de nuevo.

Esto pasa porque usan la misma instancia al usar el patrón “Singleton” que es el que viene por defecto.