

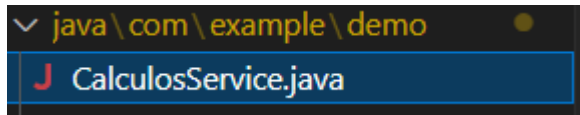
Tema 4

Servicios



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).

Para realizar este ejercicio, hice la copia del enunciado y cree una nueva clase llamada CalculosService:



Escribí la anotación `@Service` para indicar que será un servicio

```
@Service
public class CalculosService{
```

Moví la lógica de los métodos que antes estaba en el controlador:

```
public boolean esPrimo(String num){
    boolean esPrimo= true;//Necesario para saber si el número es primo
    Integer numero = 0;
    if(num==null) return false; /*Si el usuario no introdujo valor en la url
    redirige al controlador del error*/
    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
        }
    }
    return esPrimo;
}

public Double obtHipotenusa(Integer X, Integer Y){
    if(X<0 || Y<0) return (double) -1;
    return Math.pow(X,2)+Math.pow(Y,2);
}
```

```

public TreeSet<Integer> obtListaSinRepetidos(Integer X){
    if(X<1||X>100) return null;
    Random r = new Random();
    TreeSet<Integer> resultado = new TreeSet<>();
    int adjudicar=0;
    for(int i = 0; i<X;i++){
        adjudicar=r.nextInt(bound:100)+1;
        while(resultado.contains(adjudicar)){adjudicar=r.nextInt(bound:100)+1;}
        //Evita que haya iteraciones en las que no se introduzcan números
        resultado.add(adjudicar);
    }
    return resultado;
}
public ArrayList<Integer> verDivisores(int X){
    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
    }
    return resultado;
}

```

Llamé al servicio en el controlador:

```

public class NumController{
    @Autowired
    CalculosService calculosService;
}

```

E introduje el método correspondiente del servicio en cada addAttribute():

```

model.addAttribute(attributeName:"primo", calculosService.esPrimo(num));
model.addAttribute(attributeName:"valor", num);
return "index";

```

```

model.addAttribute(attributeName:"resultado", calculosService.obtHipotenusa(X, Y));
/*Fórmula de la hipotenusa: es la suma de los cuadrados de X e Y
| model.addAttribute() los envía al html*/
return "index";

```

```

model.addAttribute(attributeName:"lista", calculosService.obtListaSinRepetidos(X));
return "index";

```

```

model.addAttribute(attributeName:"divisores", calculosService.verDivisores(X));
return "index";

```

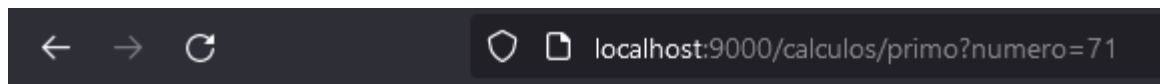
```

model.addAttribute(attributeName:"divisoresDos",calculosService.verDivisores(X));
return "index";

```

Resultado:

- Método que averigua si un número es primo:

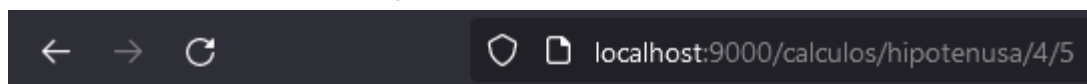


Hola Mundo

El numero 71 es primo

Valor de X:
Valor de Y:
Hipotenusa:

- Método que averigua la hipotenusa:



Hola Mundo

El numero no es primo

Valor de X: 4
Valor de Y: 5
Hipotenusa: 41.0

- Método que devuelve una lista de números ordenados sin repetidos:

Hola Mundo

El numero no es primo

Valor de X: 20

Valor de Y:

Hipotenusa:

3
8
15
17
19
20
43
47
48
53
56
60
73
76
80
81
86
87
88
94

- Método que devuelve divisores exactos de un número:



Hola Mundo

El numero no es primo

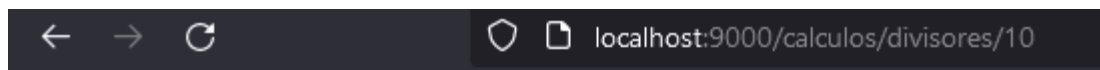
Valor de X: 30

Valor de Y:

Hipotenusa:

[1](#)
[2](#)
[3](#)
[5](#)
[6](#)
[10](#)
[15](#)

- Al pulsar en 10:



Hola Mundo

El numero no es primo

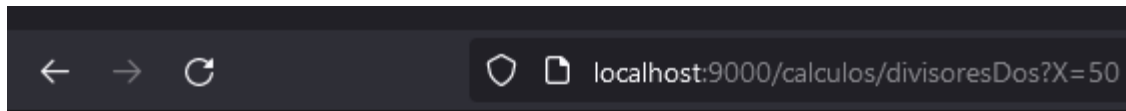
Valor de X: 10

Valor de Y:

Hipotenusa:

[1](#)
[2](#)
[5](#)

- El mismo método de antes con @RequestParam:



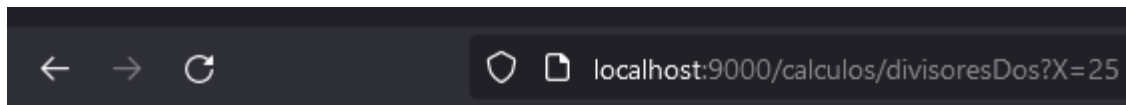
Hola Mundo

El numero no es primo

Valor de X:
Valor de Y:
Hipotenusa:

1
2
5
10
25

- Al pulsar en 25:



Hola Mundo

El numero no es primo

Valor de X:
Valor de Y:
Hipotenusa:

1
5

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:

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

- 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.
 - c) Si le pasamos en la parte query dos fechas, mostrará los días comprendidos entre ambas fechas.
 - d) Si no se pasan ninguna fecha, mostrará los días transcurridos entre el 1 de enero y hoy.
 - e) Si pasamos /bisiesto/fecha1 mostrará si fecha1 pasada en el path es de un año bisiesto.
 - f) Si le pasamos /bisiesto/año1/año2 mostrará los años bisiestos comprendidos entre ambos años.
- Puedes crear en el index unos enlaces a las URL creadas con distintas fechas como parámetro y probar de forma sencilla los casos anteriores.

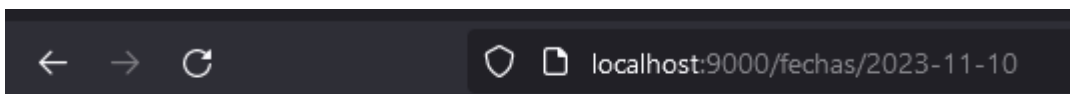
(NOTA: Pregunte al profesor si podía empezar desde cero en vez de copiar todos los archivos y obtuve una respuesta afirmativa)

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

Para establecer esta base se fijará un @RequestMapping("/fechas")

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

Resultado:

A screenshot of a web browser's address bar. It shows navigation icons (back, forward, refresh) on the left and a lock icon followed by the URL 'localhost:9000/fechas/2023-11-10' on the right.

Días desde el 1 de enero hasta 2023-11-10: 313

- 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.

Para realizar esta función (y las posteriores) hice un servicio:

```
@Service
public class FechaService {
```

Y realice una función:


```
public int desdeUno(LocalDate ld){
    return ld.getDayOfYear()-1;
}
```

Esta función recibe una fecha y devuelve la resta entre el día de la fecha (*getDayOfYear()* devuelve un día entre 0 y 365 o 366 si es bisiesto) y 1, que sería el resultado de ejecutar *getDayOfYear()* al 1 de enero.

En el controlador cree una instancia del servicio:

```
@Autowired
FechaService fechaService;
```

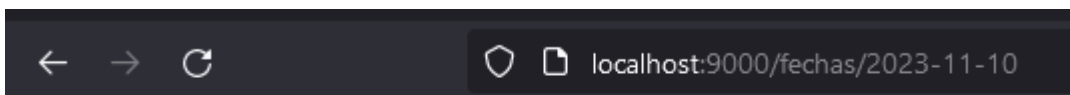
Y asigne la función a un `@GetMapping`:

```
@GetMapping("/{f1}")
public String show2B(@PathVariable LocalDate f1, Model model){
    model.addAttribute(attributeName:"f1Parametro", f1);
    model.addAttribute(attributeName:"f1Resultado", fechaService.desdeUno(f1));
    return "index";
}
```

Html:

```
<span th:if="{f1Resultado!=null}">Días desde el 1 de enero hasta <span th:text="{f1Parametro}"></span>: <span th:text="{f1Resultado}"></span></span>
```

Resultado:



Días desde el 1 de enero hasta 2023-11-10: 313

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

Realice una función:

```
public int entreDosFechas(LocalDate ld1, LocalDate ld2){
    return ld1.getDayOfYear()-ld2.getDayOfYear();
}
```

Esta función resta el día del año de la primera fecha entre la segunda

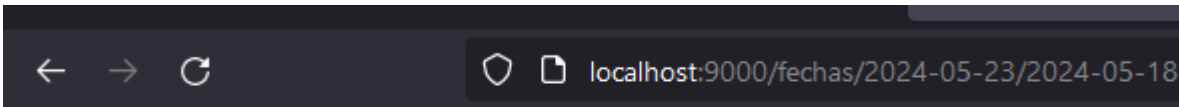
Y usando la instancia al servicio anterior, lo asigne a otro `@GetMapping`:

```
@GetMapping("/{f2}/{f3}")
public String show2C(@PathVariable LocalDate f2, @PathVariable LocalDate f3, Model model){
    model.addAttribute(attributeName:"f2Parametro", f2);
    model.addAttribute(attributeName:"f3Parametro", f3);
    model.addAttribute(attributeName:"f2Resultado", fechaService.entreDosFechas(f2, f3));
    return "index";
}
```

Html:

```
<span th:if="${f2Resultado!=null}">Días entre <span th:text="${f2Parametro}"></span> y <span th:text="${f3Parametro}"></span>: <span th:text="${f2Resultado}"></span></span>
```

Resultado:



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

Para este apartado, cree otro @GetMapping y reutilice la función del servicio que diseñe en el primer apartado:

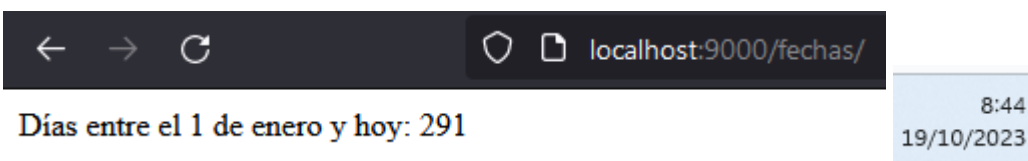
```
@GetMapping("/")
public String show2D(Model model){
    model.addAttribute(attributeName:"f3Resultado", fechaService.desdeUno(LocalDate.now()));
    return "index";
}
```

La función recibirá como argumento el día de hoy

Html:

```
<span th:if="${f3Resultado!=null}">Días entre el 1 de enero y hoy: <span th:text="${f3Resultado}"></span></span></span>
```

Resultado:



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

De nuevo, diseñe otro método en el servicio:

```
public String esBisiesto(LocalDate ld1){
    if(LocalDate.of(ld1.getYear(),month:12,dayOfMonth:31).getDayOfYear()==366) return "si";
    else return "no";
}
```

Este método recibe una fecha y creará otra fecha que será el 31/12/ del año de la fecha que se introdujo. Después compruebo si con getDayOfYear() ese número equivale a 366 (los años bisiestos tienen 366 días) y devuelvo un string que pone si o no

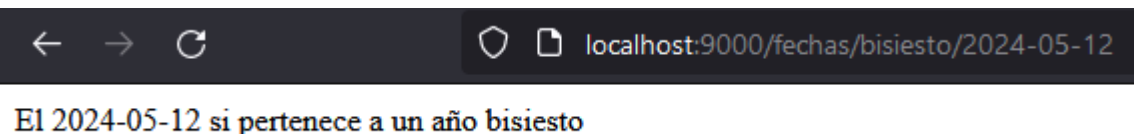
Le asignó un @GetMapping:

```
@GetMapping("/bisiestro/{f4}")
public String show2E(@PathVariable LocalDate f4, Model model){
    model.addAttribute(attributeName:"f4Parametro", f4);
    model.addAttribute(attributeName:"f4Resultado", fechaService.esBisiesto(f4));
    return "index";
}
```

Html:

```
<span th:if="{f4Resultado!=null}">El <span th:text="{f4Parametro}"></span> <span th:text="{f4Resultado}"></span> pertenece a un año bisiesto</span>
```

Resultado:



El 2024-05-12 si pertenece a un año bisiesto

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

Escribo la última función del ejercicio en el servicio:

```
public ArrayList<Integer> esBisiestoEntreDos(Integer alloUno, Integer alloDos){
    ArrayList<Integer> resultado = new ArrayList<>();
    while(alloUno<alloDos){
        if(LocalDate.of(alloUno,month:12,dayOfMonth:31).getDayOfYear()==366) resultado.add(alloUno);
        alloUno++;
    }
    System.out.println(resultado);
    return resultado;
}
```

Esta función recibe dos números. El bucle utiliza la lógica de la anterior función y si la condición se cumple, añadirá dichos años al arraylist que devolverá posteriormente

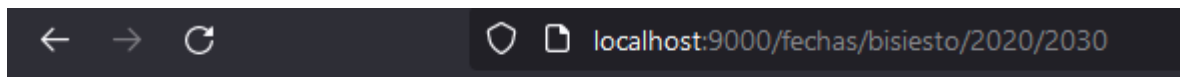
@GetMapping de la función:

```
@GetMapping("/bisiestro/{alloUno}/{alloDos}")
public String show2F(@PathVariable Integer alloUno,@PathVariable Integer alloDos,Model model){
    model.addAttribute(attributeName:"alloUnoParametro",alloUno);
    model.addAttribute(attributeName:"alloDosParametro", alloDos);
    model.addAttribute(attributeName:"alloResultado", fechaService.esBisiestoEntreDos(alloUno, alloDos));
    return "index";
}
```

En el html lo muestro con un th:each:

```
<span th:if="{alloUnoParametro!=null && alloDosParametro!=null}">Entre <span th:text="{alloUnoParametro}"></span> y <span th:text="{alloDosParametro}"></span> tienen como años bisiestos:</span>
<span th:each="factor:{alloResultado}">
    <span th:text="{factor}"></span>Texto de ejemplo</span>
</span>
```

Resultado:



Entre 2020 y 2030 tienen como años bisiestos: 2020 2024 2028

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.

Interfaz de Calculos Service:

```
public interface CalculosService{
    public boolean obtPrimo(String num,Model model);
    public Double obtHipotenusa(Integer X, Integer Y);
    public TreeSet<Integer> obtCalculosSinRepetidos(Integer X);
    public ArrayList<Integer> obtDivisores(int X);
}
```

Clase CalculosServiceImpl:

```
import org.springframework.stereotype.Service;
import org.springframework.ui.Model;
@Service
public class CalculosServiceImpl implements CalculosService{

    public boolean obtPrimo(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
        Integer numero = 0;
        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
            }
        }
        return esPrimo;
    }
}
```

Controlador NumController:

```
@Controller
@RequestMapping("/calculos")
/*@RequestMapping exige que a los distintos controladores les preceda
el argumento de su paréntesis. Ejemplo: /calculos/primo */
public class NumController{
    @Autowired
    CalculosService calculosService;
    @GetMapping("/primo")//Crea el mapeado para primo
    public String showPrimo(@RequestParam(name="numero",required = false) 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
        Integer numero = 0;
        if(num==null) return "redirect:/errorNum";//Si el usuario no introdujo valor en la url
        redirige al controlador del error*/
        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
        }
    }
}
```

Interfaz de FechasService:

```
public interface FechasService {
    public int desdeUno(LocalDate ld);
    public int entreDosFechas(LocalDate ld1, LocalDate ld2);
    public String esBisiesto(LocalDate ld1);
    public ArrayList<Integer> esBisiestoEntreDos(Integer alloUno, Integer alloDos);
}
```

Clase FechasServiceImpl:

```
@Service
public class FechaServiceImpl implements FechasService{
    public int desdeUno(LocalDate ld){
        return ld.getDayOfYear()-(LocalDate.of(LocalDate.now().getYear(),month:1,dayOfMonth:1)).getDayOfYea
    }
    public int entreDosFechas(LocalDate ld1, LocalDate ld2){
        return ld1.getDayOfYear()-ld2.getDayOfYear();
    }
    public String esBisiesto(LocalDate ld1){
        if(LocalDate.of(ld1.getYear(),month:12,dayOfMonth:31).getDayOfYear()==366) return "si";
        else return "no";
    }
    public ArrayList<Integer> esBisiestoEntreDos(Integer alloUno, Integer alloDos){
        ArrayList<Integer> resultado = new ArrayList<>();
        while(alloUno<alloDos){
            if(LocalDate.of(alloUno,month:12,dayOfMonth:31).getDayOfYear()==366) resultado.add(alloUno);
            alloUno++;
        }
        System.out.println(resultado);
        return resultado;
    }
}
```

Clase FechasController:

```
@Controller
@RequestMapping("/fechas")
public class FechasController {
    @Autowired
    FechasService fechaService;

    @GetMapping("/{f1}")
    public String show2B(@PathVariable LocalDate f1, Model model){
        model.addAttribute(attributeName:"f1Parametro", f1);
        model.addAttribute(attributeName:"f1Resultado", fechaService.desdeUno(f1));
        return "index";
    }
    @GetMapping("/{f2}/{f3}")
    public String show2C(@PathVariable LocalDate f2, @PathVariable LocalDate f3, Model model){
        model.addAttribute(attributeName:"f2Parametro", f2);
        model.addAttribute(attributeName:"f3Parametro", f3);
        model.addAttribute(attributeName:"f2Resultado", fechaService.entreDosFechas(f2, f3));
        return "index";
    }
    @GetMapping("/")
    public String show2D(Model model){
        model.addAttribute(attributeName:"f3Resultado", fechaService.desdeUno(LocalDate.now()));
        return "index";
    }
}
```

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. 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. Para responder a estas últimas cuestiones vuelve al tema 1 a la sección de Scopes.

Clase del servicio que contiene la lógica:

```
@Service
public class NumServicio {
    Random random = new Random();
    public Set<Integer> lista = new LinkedHashSet<>();

    public Set<Integer> obtLista(){
        return lista;
    }

    public void nuevoNumero(){
        boolean alladido;
        do{alladido = lista.add(random.nextInt(bound:100)+1);}
        while(!alladido);
    }

    public void borrarNumero(Integer numero){
        lista.remove(numero);
    }
}
```

Controlador que utiliza el servicio:

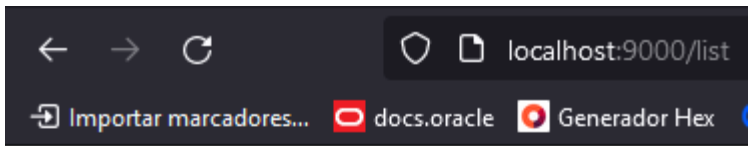
```
@Controller
public class NumController {
    @Autowired
    NumServicio numServicio;

    @GetMapping("/{}/list","")
    public String showList(Model model){
        model.addAttribute(attributeName:"cantidadTotal", numServicio.obtLista().size());
        model.addAttribute(attributeName:"listaNumeros", numServicio.obtLista());
        return "listView";
    }

    @GetMapping("/new")
    public String showNew(){
        numServicio.nuevoNumero();
        return "redirect:/list";
    }

    @GetMapping("/delete/{id}")
    public String showDelete(@PathVariable Integer id){
        numServicio.borrarNumero(id);
        return "redirect:/list";
    }
}
```

Resultado en firefox tras añadir unos cuantos números:



Listado de numeros aleato

Numero	Operacion
14	delete
95	delete
77	delete
32	delete
89	delete
62	delete

Total numeros:6

[Nuevo numero](#)

Resultado al cargar el mismo host (en la misma instancia) en Edge:

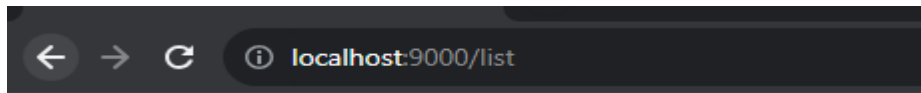


Numero	Operacion
14	delete
95	delete
77	delete
32	delete
89	delete
62	delete

Total numeros:6

[Nuevo numero](#)

Resultado al cargar el mismo host (en la misma instancia) en Chrome:



Listado de numeros aleatorios

Numero	Operacion
14	delete
95	delete
77	delete
32	delete
89	delete
62	delete

Total numeros:6

[Nuevo numero](#)

En todos los casos se puede observar que siempre aparecen los mismos valores. Esto es debido a que Spring establece por defecto el patrón Singleton en todos sus `@Component`, el cual realiza una única instancia para todas las conexiones. Para arreglar dicho problema es necesario cambiar el scope a prototype (genera una nueva instancia por cada sesión http) en el service y el controller. Servicio modificado:

```
@Service
@Scope("prototype")
public class NumServicio {
    Random random = new Random();
    public Set<Integer> lista = new LinkedHashSet<>();

    public Set<Integer> obtLista(){
        return lista;
    }

    public void nuevoNumero(){
        boolean alladido;
        do{alladido = lista.add(random.nextInt(100)+1);}
        while(!alladido);
    }

    public void borrarNumero(Integer numero){
        lista.remove(numero);
    }
}
```

Controlador modificado:

```
@Controller
@SessionScope
public class NumController {
    @Autowired
    NumServicio numServicio;

    @GetMapping("/{"/,"/list",""})
    public String showList(Model model){
        model.addAttribute(attributeName:"cantidadTotal", numServicio.obtLista().size());
        model.addAttribute(attributeName:"listaNumeros", numServicio.obtLista());
        return "listView";
    }
    @GetMapping("/new")
    public String showNew(){
        numServicio.nuevoNumero();
        return "redirect:/list";
    }
}
```

Resultado en firefox, chrome y edge:

