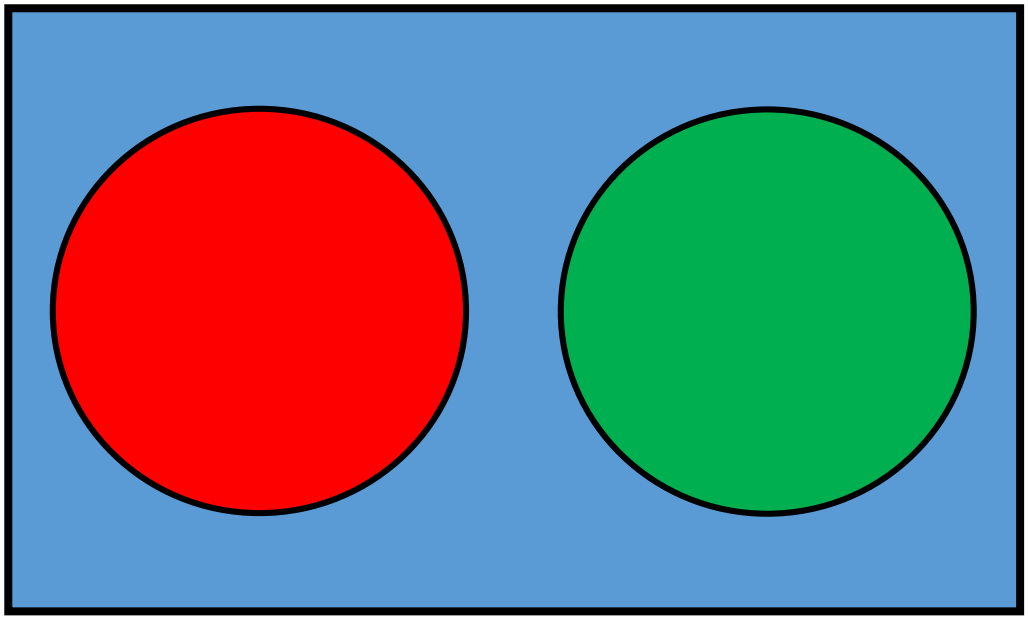
# SISTEMAS DISTRIBUIDOS

# CURSO 2019-20

## PRACTICA 1 NO GUIADA

## SOCKETS Y RMI



IVÁN MAÑÚS MURCIA - 48729799K
GRUPO MARTES 17:00-19:00
UNIVERSIDAD DE ALICANTE
GRADO EN INGENIERÍA INFORMÁTICA

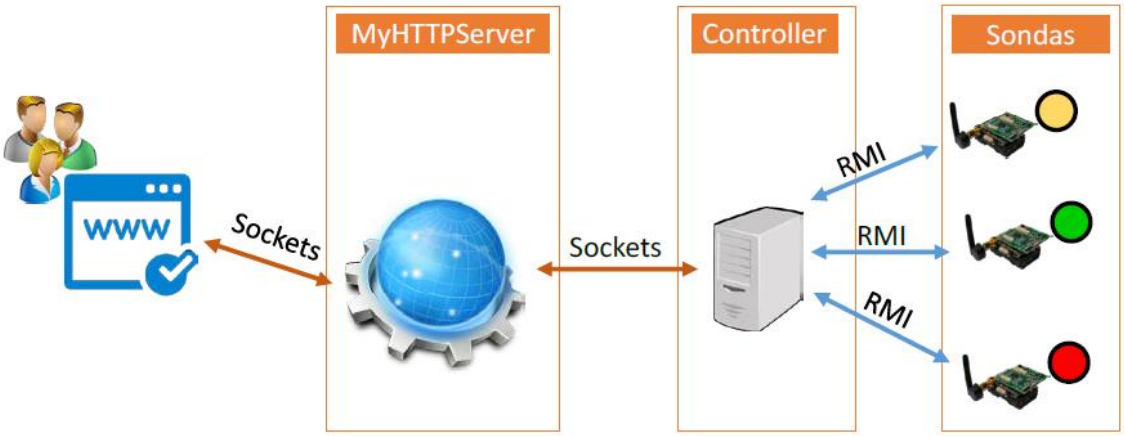
## Contenido

# Documentación de la práctica

## 1. Descripción de la práctica

- En esta práctica vamos a crear un sistema de comunicación distribuido para el control de un parking.

- El despliegue del escenario va a ser el siguiente:




- El usuario accederá a la información del parking a través de un navegador web, que accederá al servidor web llamado, MyHTTPServer el cual se comunicará con un controlador mediante sockets y éste a su vez, con los sensores del parking mediante RMI.

## 2. Estructura del proyecto

- Este proyecto se va a realizar en el lenguaje de programación Java.

- La implementación se va a realizar en 3 PCs distintos y a continuación voy a explicar componente a componente.

### 2.1 MyHTTPServer

- content
    - error404.html – Página estática que muestra un error 404.
    - error409.html – Página estática que muestra un error 409.
    - index.html – Página estática de inicio, por defecto.
    - pepe.html – Página estática para comprobaciones.

- MyHTTPServer.java
    - Archivo para iniciar el servidor, mediante argumentos se especifica el puerto en el que escucha peticiones del navegador del cliente, las conexiones máximas concurrentes que pueden haber, la IP del controlador conjunto a su puerto para enviar.

```
ServerSocket skServidor = new ServerSocket(puerto);
while( Thread.activeCount() < conexiones_max ) {
    Socket skCliente = skServidor.accept(); // Crea objeto
    System.out.println("SIRVIENDO CLIENTE");
     Thread t = new ServerThread(skCliente, ip_controller,puerto_controller);
     t.start();   // Crea un hilo(arriba) y lo inicia
}
```

- ServerThread.java

  - Es el hilo del servidor, el que responde a todo, se encarga de leer ficheros estáticos, y gestionar las URLs para comunicarle al controlador los datos necesarios.

## 2.2 ControllerSD

- Controller.sh – Un archivo sh para ejecutar todo más sencillo
- Registrar.policy – Archivo POLICY para hacer la comunicación con el RMIServer
- ControllerSD.java

  - Se encarga de leer la información del socket procedente de MyHTTPServer y responder, también se encarga de comunicarse con los sensores mediante RMI, para ello analiza los datos que le llegan procedente de la petición HTTP y genera una llamada al registrador para que le devuelva los datos que se le piden.

## 2.3 RMIServer

- CrearRMI.sh – Archivo sh para compilar y crear el archivo .jar que se le mandará al controlador para que se pueda comunicar.
- rmiregister.sh – Archivo sh para ejecutar el rmiregister de Java
- registrador.sh – Archivo sh para crear Sensores y registrarlas mediante la ejecución de Registro
- reiniciar.sh – Archivo sh para reiniciar todo el proceso.
- Registrar.policy – Archivo POLICY para hacer la comunicación posible con el Controlador.
- InterfazRemoto.java – Interfaz que tiene todos los métodos de las sondas declarados.
- ObjetoRemoto.java – Este es el archivo que contiene todos los métodos de los sensores.

  Se encarga de leer TXT, crear TXT y devolver y escribir datos en las sondas.

- Registro.java – Este archivo registra sondas para que puedan ser llamadas desde el controlador.

## 3. Guía de despliegue

- A continuación, se va a proceder a desplegar el escenario de la práctica.

- Primero de todo ejecutamos las instrucciones necesarias para compilar y ejecutar todo lo necesario para iniciar el servidor HTTP.

```
C:\Users\IVAN\Documents\MyHTTPServer>javac ServerThread.java

C:\Users\IVAN\Documents\MyHTTPServer>javac MyHTTPServer.java

C:\Users\IVAN\Documents\MyHTTPServer>java MyHTTPServer
Formato obligado
$./Servidor puerto_servidor conexiones_max ip_controller puerto_controller

C:\Users\IVAN\Documents\MyHTTPServer>java MyHTTPServer 9799 100 192.168.56.101 4872
SERVIDOR WEB ABIERTO POR EL PUERTO 9799
```

- Así si iniciamos un servidor web y llamamos al servidor HTTP nos saldrá:


- Si le pedimos algún recurso estático sin el indicador “controladorSD”. Aparecerá esa respuesta.


En cambio, si pedimos un recurso que no existe:


Si intentamos comunicarnos con el controlador sin estar conectado:

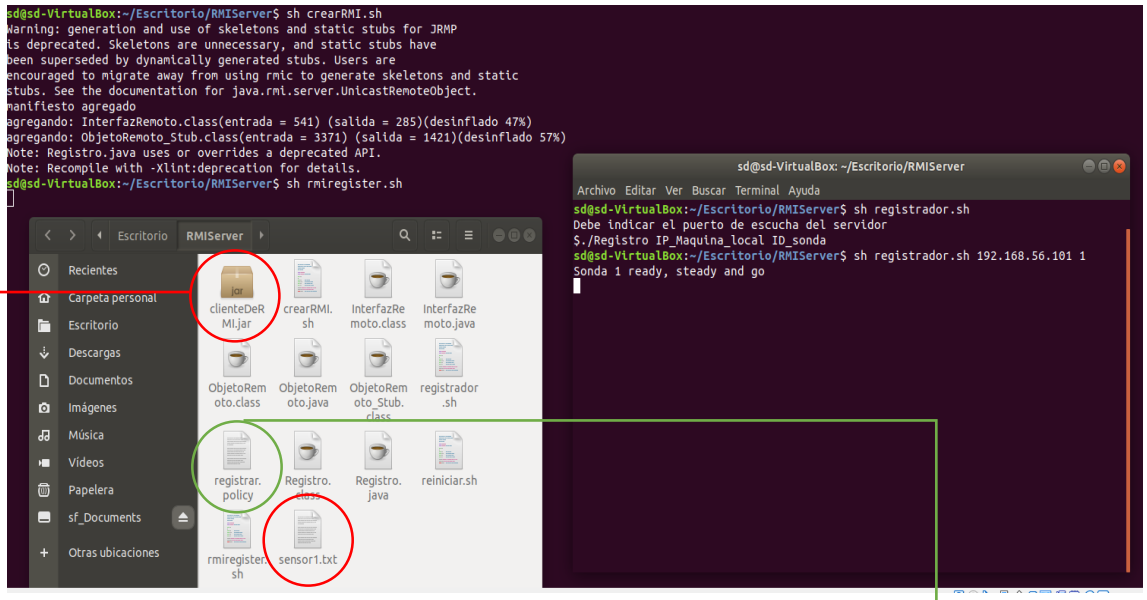

Encendamos ahora el controlador:

Si tratamos de compilarlo nos aparecerá el siguiente error:

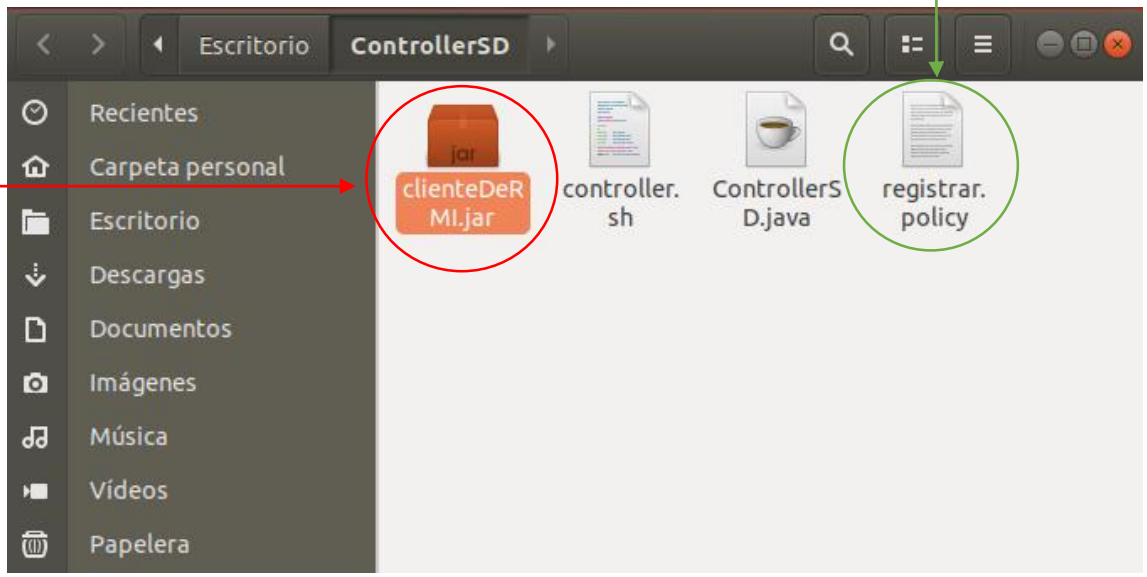
```
sd@sd-VirtualBox:~/Escritorio/ControllerSD$ sh controller.sh
ControllerSD.java:56: error: cannot find symbol
              InterfazRemoto objetoRemoto = null;
              ^
  symbol:   class InterfazRemoto
  location: class ControllerSD
ControllerSD.java:78: error: cannot find symbol
                                        if(obj instanceof InterfazRemoto) {
                                                          ^
  symbol:   class InterfazRemoto
  location: class ControllerSD
ControllerSD.java:79: error: cannot find symbol
                                                final InterfazRemoto server = (InterfazRemoto)obj;
                                                      ^
  symbol:   class InterfazRemoto
  location: class ControllerSD
ControllerSD.java:79: error: cannot find symbol
                                                final InterfazRemoto server = (InterfazRemoto)obj;
                                                                              ^
  symbol:   class InterfazRemoto
  location: class ControllerSD
ControllerSD.java:119: error: cannot find symbol
                           objetoRemoto = (InterfazRemoto) Naming.lookup(server);
                                           ^
  symbol:   class InterfazRemoto
  location: class ControllerSD
5 errors
Error: no se ha encontrado o cargado la clase principal ControllerSD
sd@sd-VirtualBox:~/Escritorio/ControllerSD$
```

Esto pasa porque no hay un archivo para comunicarse con el registrador RMI para que le muestre información.

Encendamos el registrador con 1 sonda de ejemplo:




Ahora en el controlador:

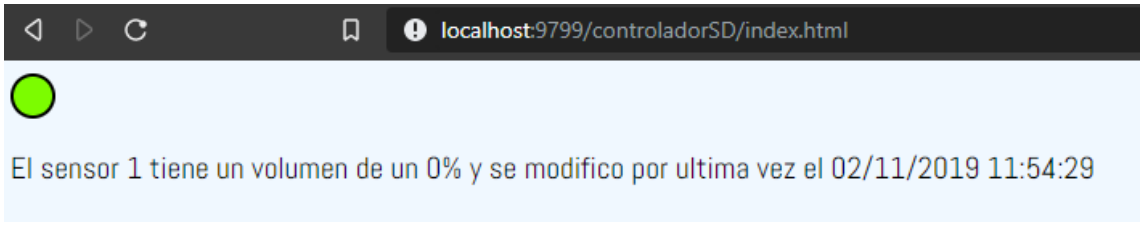



A continuación, vamos a ejecutar el controlador:

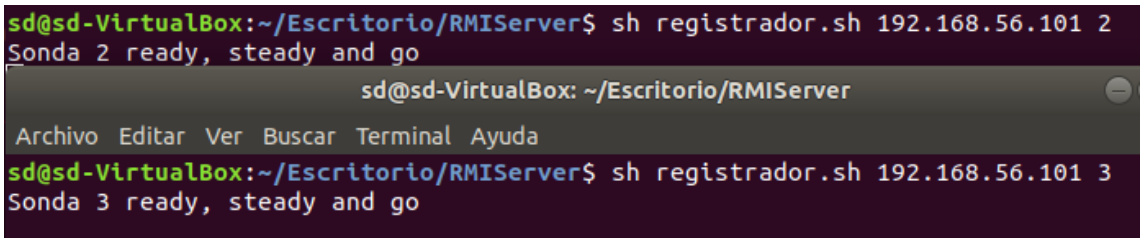
```
sd@sd-VirtualBox:~/Escritorio/ControllerSD$ sh controller.sh
Debe indicar el puerto de escucha del servidor
$./Servidor puerto_servidor IP_RMIServer puerto_RMIServer
sd@sd-VirtualBox:~/Escritorio/ControllerSD$ sh controller.sh 4872 192.168.56.101 1099
Escucho el puerto 4872
```

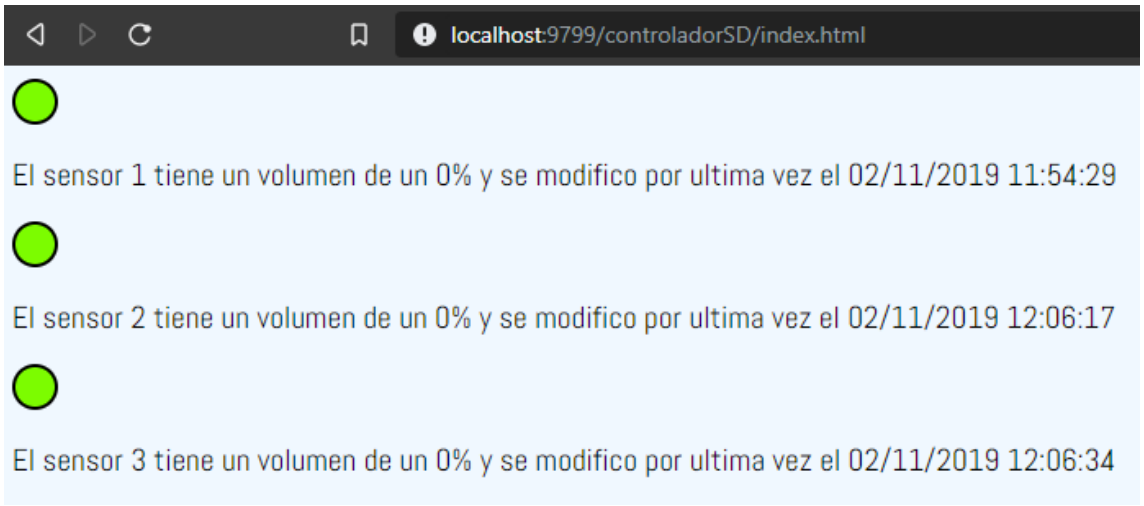
Ahora, como podemos ver, sí que compila.

Si volvemos a solicitar el index de las sondas:

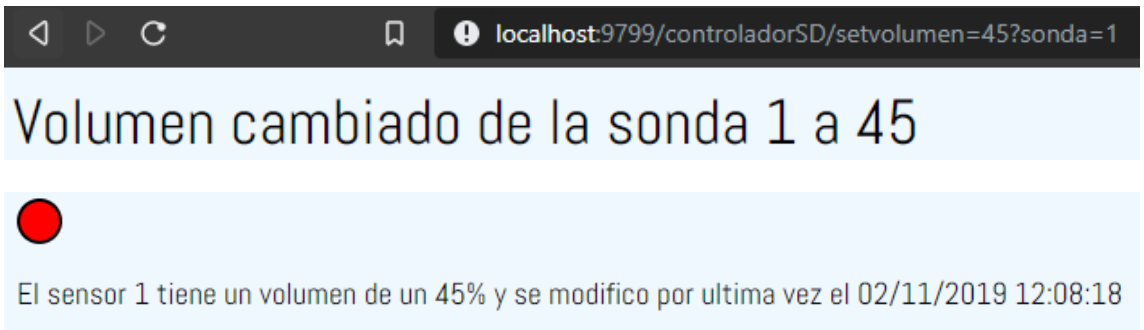



Ahora, sí que muestra las sondas…vamos a crear 2 más.




Ahora, si modificamos el volumen del sensor 1, vamos a ocupar ese sitio en el parking.



## CASO ESPECIAL

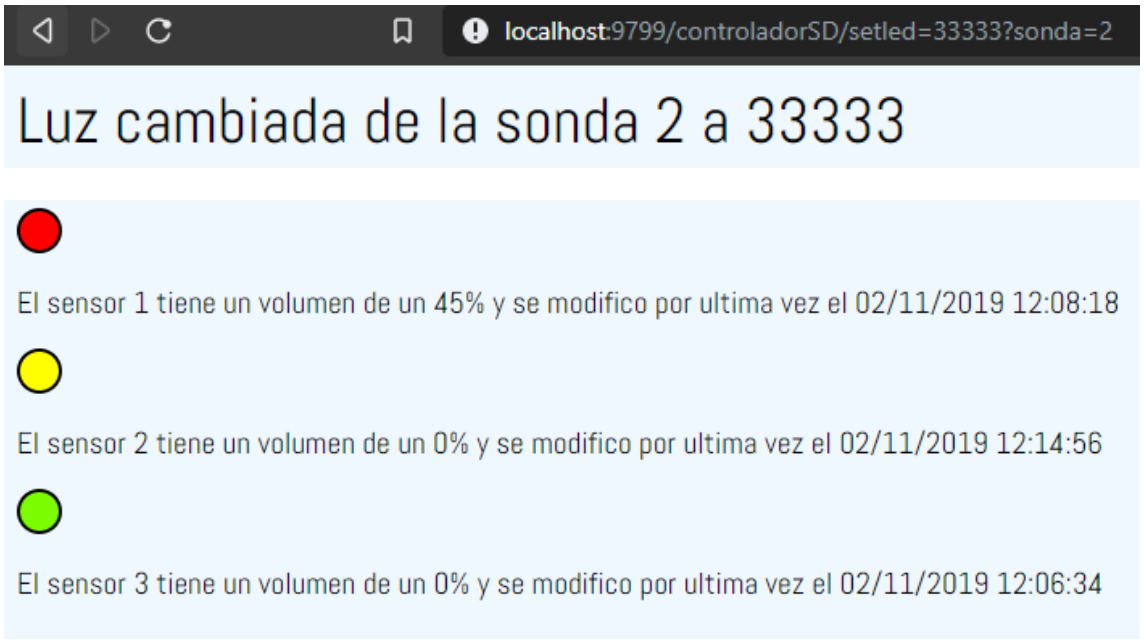
- Si no cambias el color del led manualmente, mediante setled, no podrás ver el caso especial, y es que, en mi práctica he dividido el rango de colores en 3:

- 0-21845 – Libre (Verde).

- 21864-43690 – Bloqueado (Amarillo).

-43691-65535 – Ocupado (Rojo).

-Por lo tanto, si queremos bloquear una plaza, tan solo habrá que modificar la luz a un número en el rango elegido.




## Bibliografía

Todo lo he sacado de las prácticas guiadas ofrecidas sin contar con StackOverflow para dudas del lenguaje Java en sí.