

## Práctica 2: IMPLEMENTACIÓN DE UN SERVIDOR WEB

### 1. Objetivos de la práctica

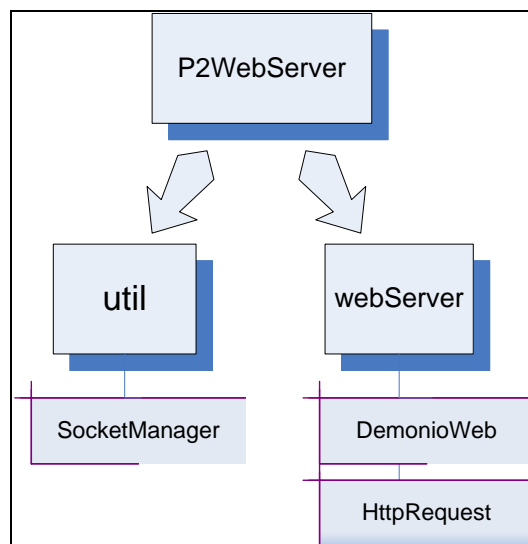
- Realizar una primera aplicación real con sockets, utilizando para ello la clase SocketManager implementada en la práctica 1.
- Implementar un servidor capaz de atender a varios clientes simultáneamente.
- Implementar un servidor Web.

### 2. Guía de la práctica

En esta práctica vamos a implementar un servidor web. Como cliente vamos a utilizar un navegador web cualquiera.

Nuestro servidor web va a ser capaz de atender múltiples peticiones simultáneamente. Por lo tanto, tendremos que entender y utilizar threads en nuestra implementación.

En esta práctica nuestro programa tendrá tres clases: nuestra clase SocketManager y dos clases más para implementar el servidor web a las que llamaremos WebServer y HttpRequest. La estructura en paquetes, habrá de seguir el esquema siguiente:



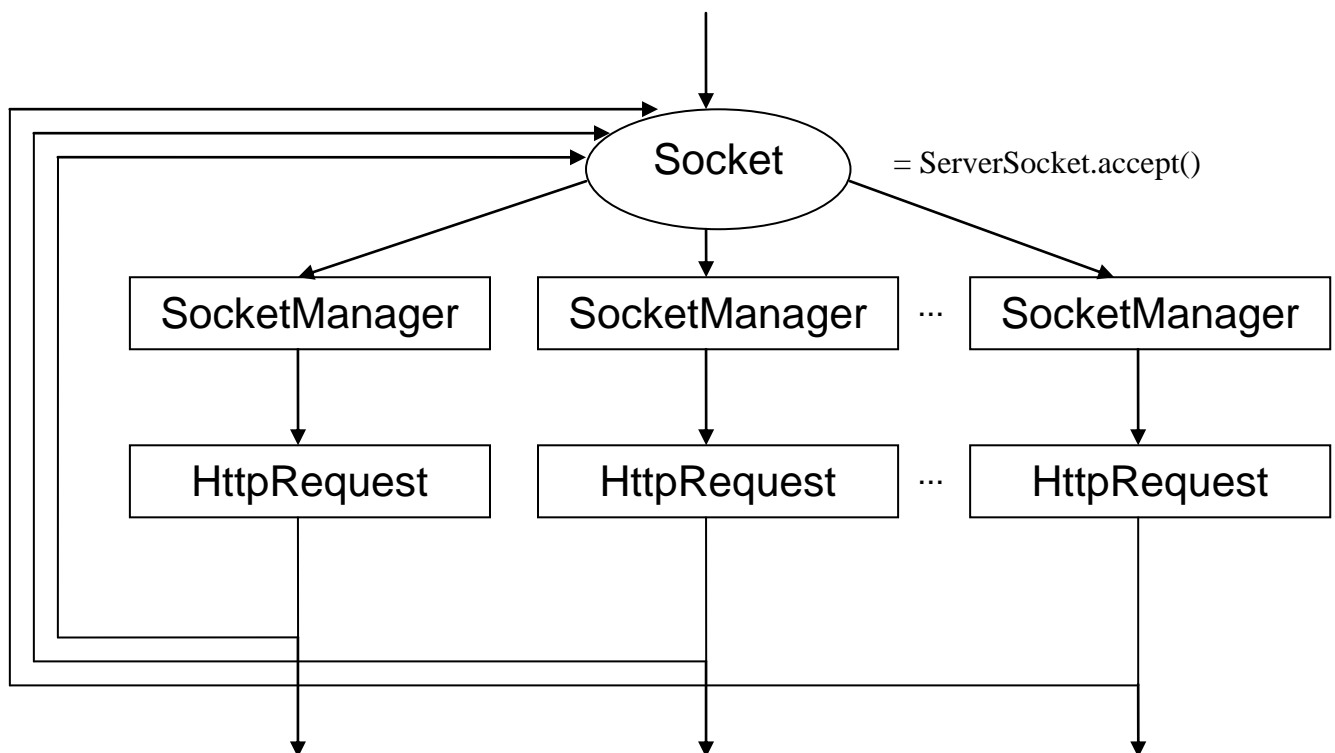
La clase WebServer se encargará de escuchar peticiones por un puerto y crear threads de la clase HttpRequest por cada petición cliente para que las respuestas puedan atenderse simultáneamente. La clase HttpRequest será la que se encargue de enviar al cliente la respuesta a su petición HTTP.

#### A. La clase WebServer

Hay algunos puntos a comentar sobre esta clase. Para empezar decir que será la clase que iniciará la ejecución del servidor web. Por lo tanto, tendrá el método main(). Al arrancar nuestro servidor, debemos indicar el puerto en el que va a estar

esperando peticiones (*java WebServer 1800*, si ejecutamos desde línea de comandos, por ejemplo) por lo que tendremos que recibir ese parámetro en nuestro programa `main()`.

Ya hemos indicado antes que el servidor web deberá atender peticiones simultáneamente por lo que cuando recibamos una petición de un cliente, el método `accept()` nos devuelve un `Socket` (recuerda que para algo hemos implementado la clase `SocketManager`) y deberemos crear un nuevo thread de ejecución sobre un objeto `HttpRequest` para responder al cliente. Lógicamente deberemos haber creado antes el objeto `HttpRequest`, al que deberemos pasarle el `SocketManager` para poder realizar la comunicación con el cliente.




## B. La clase `HttpRequest`

Ya hemos dicho anteriormente que esta clase implementará el proceso a realizar para responder convenientemente al cliente y también hemos dicho que los threads que crearemos para simultanear las respuestas HTTP serán threads creados sobre esta clase. Por eso, nuestra clase `HttpRequest` deberá implementar el interfaz `Runnable` y deberá tener el método que ejecuta un thread cuando se pone en marcha.

### ***Atributos de la clase `HttpRequest`***

¿Qué único atributo necesitaremos para esta clase conociendo que su cometido va a ser comunicarse con el cliente? Ahora es cuando comenzaremos a ver la valía de nuestro objeto “`SocketManager`”.

	PROGRAMACIÓN CON SOCKETS	Práctica 2
---	--------------------------	------------

### **Constructor de la clase *HttpRequest***

¿Y el constructor de la clase? Con la información dada hasta ahora no deberías tener problema para saber qué debe recibir como parámetro.

### ***El proceso para responder al cliente***

La clase *HttpRequest* tiene como objetivo responder al cliente adecuadamente. Por tanto tendremos que crear un método en el que implementar el proceso para dar una respuesta a la petición del cliente. Y este método deberemos invocarlo desde el método que ejecuta un thread cuando se pone en marcha.


Recuerda que estamos implementando un servidor web así que lo primero que tendremos que hacer será leer la petición HTTP que nos ha enviado el cliente web. En la petición deberemos buscar el nombre del fichero que pide el cliente. De momento podemos ir abriendo dicho fichero para, más tarde, enviar su información al cliente. Antes de construir el mensaje de respuesta podemos leer la cabecera HTTP leyendo de nuevo del socket (puedes escribirlas por pantalla para ver lo que estamos recibiendo). Quizás tengas que realizar un proceso iterativo para leer todas las líneas de la cabecera.

Por último nos queda construir el mensaje de respuesta, en el que tendremos que indicar la siguiente información:

- La línea de estado. La petición recibida del cliente puede ser atendida o no. Si el fichero de la petición no existe en el servidor indicaremos al cliente sobre dicha situación. Si se da esta situación no enviaremos ningún fichero sino que enviaremos un código HTML (etiquetas HTML) indicando que dicho fichero no existe.  
Si existe el fichero enviaremos como línea de estado "HTTP/1.0 200 OK" y si no existe enviaremos "HTTP/1.0 404 Not Found". Recuerda enviar una marca de fin de línea "\r\n" tras la línea de estado.
- El contenido del tipo (content-type). Indicaremos el tipo del fichero pedido. Para obtener el tipo y enviarlo al cliente nos fijaremos simplemente en la extensión del fichero de la petición web. Si el fichero existe enviaremos "Content-type: <tipo\_del\_contenido>" Si el fichero no existiese el tipo del contenido sería "Content-type: text/html". Recuerda enviar una marca de fin de línea "\r\n" tras esta información.

Indico a continuación algunos de los tipos posibles:

- Si el fichero tiene como extensión "htm" o "html" el content-type será "text/html"
- Si el fichero tiene como extensión ".ram" o ".ra" el content-type será "audio/x-pn-realaudio"
- Si la extensión no es conocida el content-type será "application/octet-stream"
- Antes de enviar la información pedida por el cliente debemos dejar una línea en blanco ("\r\n") para indicar el final de las líneas de cabecera.
- Por último enviaremos la información que el cliente ha pedido en su petición HTTP. Ya hemos dicho antes que si el fichero pedido no existe enviaremos unas etiquetas HTML indicando el problema. Si existe, tendremos que leer del fichero que ya teníamos abierto y enviarlo al cliente.

	PROGRAMACIÓN CON SOCKETS	Práctica 2
---	--------------------------	------------

En este último caso debemos escribir en el socket cantidades grandes de información que debemos leer de un fichero. Por tanto tendremos que crear un buffer, por ejemplo de 1 KB (`byte[] buffer = new byte[1024]`), en el que iremos escribiendo la información que leemos del fichero (te recomiendo mirar la clase `FileInputStream` para trabajar con el fichero). Evidentemente éste deberá ser un proceso iterativo hasta que hayamos leído toda la información del fichero y la hayamos escrito en el socket.

Por lo tanto, para esta escritura en el socket (sólo esta) no nos sirve el método “Escribir(String)” que ya tenemos implementado en nuestra clase `SocketManager`. Tendremos que implementar otro que nos permita escribir en el socket un determinado número de bytes (investigar en la clase `DataOutputStream`).

### ***¡Haciendo funcionar nuestro Servidor Web!***

Por último sólo queda echar a andar nuestra práctica. Es sencillo. Evidentemente el primer paso es compilar nuestras clases. Una vez hecho esto tenemos que “ejecutar” nuestro servidor web. Recuerda que la clase que tiene el método `main()` es la clase `WebServer` y que tenemos que indicarle en qué puerto va a ponerse a recibir peticiones. Más arriba propusimos indicárselo al llamar al comando `java` (`java WebServer 1800`, pero puedes pedirlo por teclado al comienzo de la “ejecución” del servidor web.

Y una vez que hayamos echado a andar nuestro servidor, solamente nos queda comenzar a hacerle peticiones con nuestro cliente web (cualquier navegador). Abriremos nuestro navegador e indicaremos la dirección a la que queremos hacer una petición web. En la mayoría de los casos realizaremos la petición web desde el mismo ordenador en el que está ejecutándose el servidor. Tenemos que recordar que para hacer referencia a la dirección IP del ordenador local tenemos dos opciones: la dirección IP `127.0.0.1` o el nombre de host “localhost”.

Por lo tanto nos quedarían las siguientes posibilidades para realizar la petición HTTP:

- `http://127.0.0.1:<número_de_puerto_del_servidor>/<nombre_del_fichero>`
- `http://localhost:<número_de_puerto_del_servidor>/<nombre_del_fichero>`

Y por último, no olvides colocar el fichero que vayas a pedir al servidor en el mismo directorio en el que esté “ejecutándose” el servidor web.

### ***Últimos detalles***

Por último no olvides tratar las distintas excepciones que pueda lanzar nuestro código mostrando por pantalla mensajes de error o realizando el proceso que consideres oportuno.

Aplica siempre que puedas la filosofía “divide y vencerás”. Crea métodos lo más pequeños que puedas. Quizás sea bueno que el proceso de lectura del fichero y escritura al socket lo implementes en un método (y quizás también el proceso de determinación del `content-type`,...).

Y, por supuesto, no olvides cerrar el fichero, el socket,...