

Ejercicio

Crearemos un servidor HTTP similar al del ejemplo del servidor HTTP (*Proyecto java ServerHTTP, apartado 5.1 de los contenidos*) Pero un servidor HTTP realista tendrá que atender varias peticiones simultáneamente. Para ello, tenemos que ser capaces de modificar su código para que pueda utilizar varios hilos de ejecución. Esto se hace de la siguiente manera:

- El hilo principal (o sea, el que inicia la aplicación) creará el socket servidor que permanecerá a la espera de que llegue alguna petición.
- Cuando se reciba una, la aceptará y le asignará un socket cliente para enviarle la respuesta. Pero en lugar de atenderla él mismo, el hilo principal creará un nuevo hilo para que la despache por el socket cliente que le asignó. De esta forma, podrá seguir a la espera de nuevas peticiones.

Por lo tanto modifica el ejemplo del servidor HTTP (*Proyecto java ServerHTTP, apartado 5.1 de los contenidos*) para que implemente multihilo, y pueda gestionar la concurrencia de manera eficiente.

- Descargamos el proyecto y abrimos la clase servidor que es donde modificaremos los datos.

- Declaramos dos variables:

-PORT: Puerto en el que el servidor estará escuchando por conexiones entrantes.

-THREAD_POOL_SIZE: Tamaño del pool de hilos, determinando el número máximo de hilos que se pueden ejecutar simultáneamente.

-private static final int PORT = 8066;

- private static final int THREAD_POOL_SIZE = 10;

- Creamos un ExecutorService con un pool de hilos fijo del tamaño definido en THREAD_POOL_SIZE.

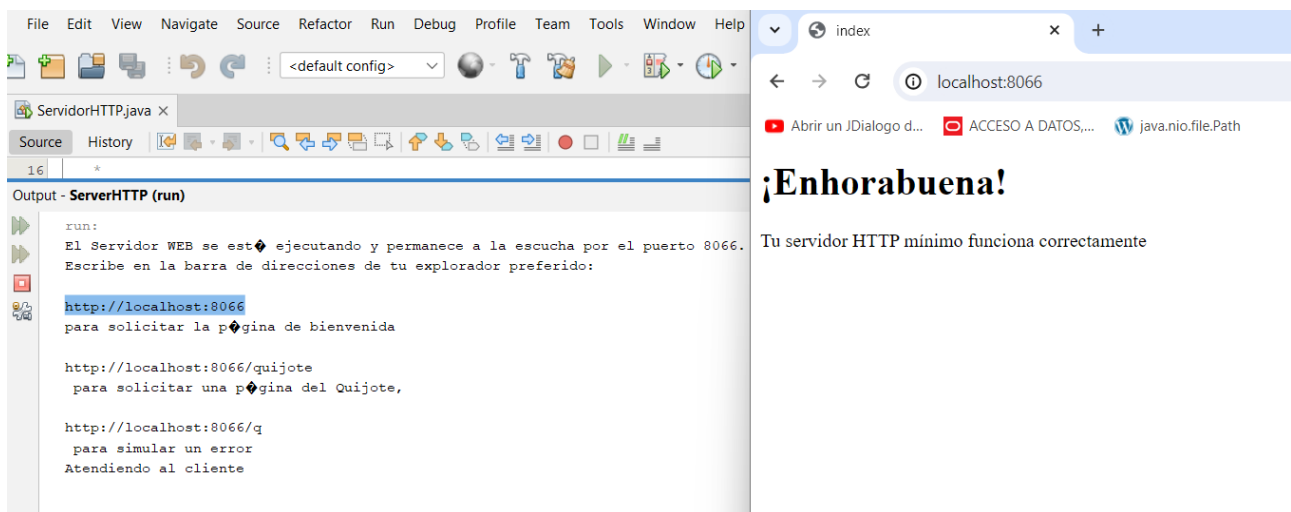
-ExecutorService threadPool = Executors.newFixedThreadPool(THREAD_POOL_SIZE);

- Creamos una instancia de ExecutorService, que se ha creado previamente con Executors.newFixedThreadPool(THREAD_POOL_SIZE)

-threadPool.execute()

- `() -> { ... }` expresión lambda, implementa la interfaz Runnable. Dentro de esta expresión lambda, definimos el código que se ejecutará en un hilo del pool en un bloque try-catch-finally para manejar las excepciones .

```
threadPool.execute() -> {
    try {
        procesaPetición(socCliente);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socCliente.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
});
```



The image shows a screenshot of an IDE (Eclipse) and a web browser. The IDE window displays the source code for `ServerHTTP.java` and the output of the program. The output shows the server running on port 8066 and handling requests. The web browser shows the `localhost:8066` address, with a message in Spanish: "¡Enhorabuena! Tu servidor HTTP mínimo funciona correctamente" (Congratulations! Your minimum HTTP server works correctly).

IDE Output - ServerHTTP (run):

```
run:
El Servidor WEB se está ejecutando y permanece a la escucha por el puerto 8066.
Escribe en la barra de direcciones de tu explorador preferido:

http://localhost:8066
para solicitar la página de bienvenida

http://localhost:8066/quijote
para solicitar una página del Quijote,

http://localhost:8066/q
para simular un error
Atendiendo al cliente
```

Web Browser: `localhost:8066`

Message: ¡Enhorabuena!
Tu servidor HTTP mínimo funciona correctamente

