Introducción a Sockets en Java

Sockets 🔌

Un **socket** es un punto de comunicación entre dos dispositivos de red, permitiendo que dos programas, generalmente en diferentes máquinas, se comuniquen entre sí.

Actúan como extremos para enviar y recibir datos a través de una red.

Sockets 🔌

Abstraen la complejidad de las conexiones de red y manejan todo lo relativo a los protocolos correspondientes (por ejemplo, TCP) ofreciendo la API general de `Streams`; de esta manera podemos tratar la conexión remota de la misma manera que leeríamos o escribiríamos de un fichero.

Streams

Los `Streams`, flujos de datos, son el concepto utilizado por la Java para manejar I/O.

Se definen dos **interfaces** principales `InputStream`, `OutputStream` que implementan distintas fuentes de datos, como para I/O con ficheros `FileInputStream`, `FileOutputStream`.

Sus métodos principales son respectivamente `read`, `write`.
Estas interfaces estan pensadas para leer **bytes**.

Streams

Si queremos leer y escribir texto existen las interfaces de `Reader` y `Writer`.

Esto es necesario ya que existen muchas codificaciones de carácteres por lo que la traducción de bytes a texto puede no ser tan sencilla.

Por ejemplo una de las codificaciones más utilizadas hoy en día UTF-8, tiene carácteres multibyte. También UNICODE, donde esto ⚡ es un caracter.

De vuelta a los sockets

Utilizaremos la clase Socket para crear una conexión y de esta conexión podemos obtener un InputStream/ OutputStream para efectuar la comunicación con un proceso remoto (o local).

- Para recibir datos tendremos que leer del InputStream. (NOTA: Si no ha llegado nada el proceso se bloquea.)
- Para enviar datos tendremos que escribir en el OutputStream.

Tipos de sockets

Distinguimos según el protocolo que utilizan.

- **Stream Sockets (TCP):** Garantizan la entrega y el orden de los datos.
- **Datagram Sockets (UDP):** Son más rápidos pero no garantizan el orden ni la entrega.

El Socket al que hacemos referencia más adelante es el que utiliza TCP.

ServerSocket

Es necesario para poder establecer una conexión que la otra parte esté receptiva. Como siempre nos referimos como cliente a la parte que incia la comunicación mientras que el servidor es quién recibe esa primera comunicación.

La clase `ServerSocket` nos permite hacer justo eso. Mediante el metodo `accept` nos quedamos a la espera de comunicaciones entrantes, y cuando una ocurre obtenemos un objeto `Socket`. La comunicación a partir de aquí ya es simétrica. (No como ocurre con HTTP)

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class ClienteTCP {
    public static void main(String[] args) {
        try (
            Socket socket = new Socket("localhost", 1234);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))
        ) {
            out.println("Hola servidor!");
            System.out.println("Respuesta del servidor: " + in.readLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(1234)) {
            System.out.println("Servidor TCP en espera de conexiones...");

            try (
                Socket clienteSocket = serverSocket.accept();
                PrintWriter out = new PrintWriter(clienteSocket.getOutputStream(), true);
                BufferedReader in = new BufferedReader(new InputStreamReader(clienteSocket.getInputStream()))
            ) {
                String mensaje = in.readLine();
                System.out.println("Cliente: " + mensaje);
                out.println("Hola, cliente!");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```