Memoria Practica VIII Sockets(II)

Que

Mediante el uso de Datagrams Sockets, realiza una pareja de programas en el cual el cliente indicará un país al servidor, y éste le devolverá la capital si la conoce y “desconocido” si no la conoce. La salida por pantalla deberá ser:

En el cliente:

```
> Obtener capital de España
> Capital de España: Madrid
…
> Obtener capital de Ruanda
> No se ha podido obtener la capital de Ruanda
```

En el servidor:

```
> DatagramServer a la espera de peticiones en puerto 5555
> Petición recibida: España
> Respuesta petición España → Madrid
…
> Petición recibida: Ruanda
> Respuesta petición Ruanda → Desconocida
```

Modifica el programa anterior, para que se utilicen Sockets Streams en lugar de Datagrams Sockets. Además, en caso de que el servidor desconozca la respuesta, el cliente podrá darle la respuesta al servidor y éste la almacenará:

Para que

Este ejercicio ha resultado útil para aprender y practicar el funcionamiento de los Sockets tanto para servidores UDP como TCP. La actividad sirve para tener una idea básica de la programación en TCP y UDP con Sockets con Java o C, dependiendo del lenguaje que queramos usar. Yo he decidido utilizar Java, ya que estoy más familiarizado con él.

Como

En primer lugar, no hablaré de las clases `ReadClient` y `Colors` porque son clases genéricas que siempre utilizo para leer datos del usuario y para sacar mensajes más coloridos por consola. Pero sí hablaré de las demás clases.

Clase Gestor

No quiero detenerme mucho con esta clase; simplemente diré que esta clase se encarga de leer y escribir en el fichero y de crearlo si no existe. Contiene dos métodos: `pull` y `push`. El método `pull` se encarga de leer el fichero y convertirlo en un `HashMap`, gestionando los posibles errores, y el método `push` se encarga de convertir un `HashMap` pasado como parámetro de entrada al formato del fichero, además de reescribir en el fichero los datos del `HashMap`.

Clase DatagramClient

Esta clase tiene dos variables `final`: la IP a la que se conecta el cliente y el puerto. Esta función solo tiene la función principal. Comienza con un bucle que no para hasta que el cliente decide escribir "/c". Si el cliente escribe un país, el programa crea un `DatagramSocket`. Es necesario crear un `socket Datagram` porque se quiere implementar la comunicación `UDP`, UDP (User Datagram Protocol) es un protocolo de comunicación sin conexión en redes, sin garantía de entrega ni orden de los datos transmitidos. Los sockets `Datagram` son adecuados para la transmisión de datos en modo no conectado, lo que significa que cada mensaje enviado es independiente y no hay un establecimiento de conexión previa, a diferencia de `TCP`, TCP (Transmission Control Protocol) es un protocolo de transporte confiable que garantiza la entrega ordenada de datos en conexiones orientadas. Se emplea en aplicaciones donde la integridad y secuencia de datos son críticas, como transferencia de archivos y navegación web. Esto prioriza la velocidad y eficiencia, pero no la entrega de la información. Luego, se convierte a un `array` de bytes para poderse enviar como un paquete `Datagram`, se crea un objeto `InetAddress` con la IP del servidor necesaria para la creación del paquete `Datagram` y se envía con una función del `socket socket.send()` este envía un paquete de datagramas desde este socket, el DatagramPacket incluye información que indica los datos que se enviarán, su longitud, la dirección IP del host remoto y el número de puerto en el host remoto. Para finalizar, necesitamos saber qué nos ha devuelto el servidor. Para ello, creamos otro array de bytes, creamos otro paquete y lo recibimos con `socket.receive()`, este método se bloquea hasta que se recibe un datagrama. Cuando este método regresa, el búfer del DatagramPacket se llena con los datos recibidos. El campo de longitud del objeto de paquete de datagrama contiene la longitud del mensaje recibido. Si el mensaje es más largo que la longitud del paquete, el mensaje se trunca. Con este paquete, creamos un `string` con la información del servidor y la mostramos por pantalla.

Clase DatagramServer

Esta clase actúa como servidor `UDP` y tiene un `HashMap` donde guarda los datos de los países, el puerto y también creo un objeto de la clase `Gestor` para poder leer y escribir en el fichero que le paso como parámetro. `DatagramServer` solo tiene el método principal. Voy a explicar brevemente cómo he desarrollado esta clase, ya que la teoría necesaria para entenderlo ya la he explicado en la clase anterior. Dentro de un `try`, declaro un `DatagramSocket` con el puerto definido, informo por pantalla el puerto que estoy usando y creo un bucle para que ejecute lo siguiente: recibo la petición del cliente en un paquete `Datagram` con `socket.receive()` que he creado. Luego creo un `String` con ese paquete y actualizo mi `HashMap` con la función `pull` de la clase `Gestor`. Con la función `getOrDefault()` de la clase `Map`, compruebo si existe y lo guardo en un `String`. Si no, en el mismo `string` se guardará "Desconocida", y le envío la respuesta al cliente creando un array de bytes de la capital, creando un objeto `InetAddress` para crear el paquete y enviárselo al cliente. Muestro un mensaje de la respuesta que le he dado al cliente.

Clase StreamClient

La clase `StreamClient` tiene las mismas variables `final` que el `DatagramClient`. Su función principal es similar al Cliente `Datagram`. Este también tiene un bucle para no dejar de pedirle al cliente un país. En él, se pide al cliente el país y, si no es la opción para salir, se establece una conexión con el servidor mediante la creación de un `socket`. Se utiliza un objeto `PrintWriter` para enviar los datos al servidor pasandole como parametros `socket.getOutputStream()`, este metodo obtiene el flujo de salida asociado al socket, que es utilizado para enviar datos desde el cliente al servidor. En otras palabras, proporciona el canal a través del cual los datos serán enviados al servidor y true para que el buffer se borre automaticamente. También se crea un `BufferedReader` para recibir datos del servidor, para crear este BufferedReader, le pasamos como parametro el flujo de entrada asociado al socket. Representa el canal a través del cual el cliente recibe datos enviados por el servidor con la funcion `socket.getInputStream()` dentro de un `InputStreamReader`.Este constructor convierte el flujo de entrada de bytes en un flujo de caracteres. InputStreamReader toma un InputStream (flujo de entrada de bytes) como argumento y proporciona un Reader (flujo de entrada de caracteres).

Hasta este punto, ya podemos enviar y recibir datos del servidor con las funciones `println()` (que envía una cadena de texto seguida de un salto de línea al flujo de salida) del `PrintWriter` o `readLine()` del `BufferedReader`, que lee una línea completa de texto desde el flujo de entrada, lee caracteres hasta encontrar un carácter de nueva línea ('\n'). El resultado es la cadena de texto completa sin incluir el carácter de nueva línea.

Luego, enviamos el nombre del país al servidor y guardamos en un `string` su respuesta. Si la respuesta es "Desconocida", le damos al cliente la opción de introducir una capital. Para ello, le pedimos amablemente que introduzca "s/n". Si la respuesta es afirmativa, le pedimos al cliente que introduzca la capital y la enviamos al servidor. Finalmente, cerramos el `socket`, el `PrintWriter` y el `BufferedReader` y volvemos al inicio del `while`.

Clase StreamServer

Esta clase es similar a la de `DatagramSocket`, tiene las mismas variables generales: el puerto, el `HashMap` y un objeto de la clase `Gestor`. Primero, crea un objeto `ServerSocket` que sirve para escuchar desde un puerto específico, en este caso, 5555. Luego, dentro de un bucle, crea un `Socket` aceptando la petición de conexión del cliente, además de un `PrintWriter` y un `BufferedReader` para enviar y leer datos. Lee la petición del cliente, actualiza el `HashMap` y con la función `getOrDefault()` busca la capital o devuelve "Desconocida", y envía la respuesta. Si la capital es desconocida, lee del cliente y comprueba si lo que ha escrito el cliente es `null`. Si es así, significa que el cliente ha dicho que no a añadir una capital, pero si no es `null`, se guarda en el `HashMap` el país y su capital y se escribe en el fichero con la ayuda de la clase `Gestor` y su función `push()`. Finalmente, informa de la actualización y cierra el socketCliente, el `PrintWriter` y el `BufferedReader`.

Conclusión

En conclusión, considero que esta actividad ha sido útil y ha contribuido significativamente a mejorar mis habilidades de programación con `Sockets`.

Además, me intriga conocer las diferentes aproximaciones que mis compañeros han tomado para abordar esta actividad, ya que soy consciente de que hay varias formas de implementarla. Consultar a mis amigos y obtener explicaciones sobre sus enfoques podría proporcionarme valiosas perspectivas y aprender nuevas técnicas. También ver cómo se hace esta actividad en `C` para poder comprender un poco más cómo funciona `C`.

En general, me ha parecido interesante descubrir como desarrollar una idea muy basica de los diferentes tipos de servidores `TCP/UDP`. Considero que la dificultad de esta práctica no ha sido elevada excepcionando el tema de los caracteres especiales ya que no he sabido como implementarlo y me gustaria ver una forma valida de hacer-lo.