**ACTIVIDAD 11**

**Enunciado**

Realizar un programa con sockets TCP que establezca un intercambio de información entre un servidor y varios clientes. El intercambio consistirá en que cada cliente le enviará al servidor una cadena de caracteres y éste se la devolverá en letras mayúsculas. La comunicación entre el cliente y el servidor finalizará cuando el primero le envíe al segundo la cadena “\*”

Con respecto al **servidor**:

* Existirá una primera clase de nombre Servidor, cuyo cometido principal será el de esperar las peticiones de conexión de los clientes
* El puerto por el que escuchará las peticiones de los clientes será el 6000
* Utilizará un bucle infinito para poder escuchar las peticiones procedentes de los clientes
* Cada vez que capture una petición de conexión de un nuevo cliente, mostrará el siguiente mensaje:

Cliente Conectado.....

* Existirá una segunda clase de nombre HiloServidor, cuyo cometido principal será el de atender las operaciones específicas de los clientes (reenviar en mayúsculas la cadena de caracteres recibida), posibilitando de esta manera que la clase Servidor se centre en la gestión de las peticiones de conexión de los clientes
* Esta clase heredará de la clase Java que corresponda, para poder así atender a cada cliente como si fuera un hilo
* Necesitará que su constructor dé un estado inicial a los objetos de la clase HiloServidor
* Cada vez que atienda una petición de un mismo cliente, mostrará un mensaje análogo al siguiente, en el que se incluyen los datos del socket:

Comunico con: Socket[addr=/127.0.0.1,port=52563,localport=6000]

* Cuando el cliente le envíe la cadena de caracteres “\*”, mostrará un mensaje análogo al siguiente, también con datos del socket:

Fin de la conexión con: Socket[addr=/127.0.0.1,port=52563,localport=6000]

* Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* En el supuesto de que 2 clientes le envíen secuencialmente al servidor 2 y 3 mensajes para poner en mayúsculas (más el correspondiente mensaje para finalizar la comunicación), los resultados que mostrará serán equivalentes a los siguientes:

Cliente Conectado.....

Comunico con: Socket[addr=/127.0.0.1,port=54109,localport=6000]

Comunico con: Socket[addr=/127.0.0.1,port=54109,localport=6000]

Comunico con: Socket[addr=/127.0.0.1,port=54109,localport=6000]

Fin de la conexión con: Socket[addr=/127.0.0.1,port=54109,localport=6000]

Cliente Conectado.....

Comunico con: Socket[addr=/127.0.0.1,port=54165,localport=6000]

Comunico con: Socket[addr=/127.0.0.1,port=54165,localport=6000]

Comunico con: Socket[addr=/127.0.0.1,port=54165,localport=6000]

Comunico con: Socket[addr=/127.0.0.1,port=54165,localport=6000]

Fin de la conexión con: Socket[addr=/127.0.0.1,port=54165,localport=6000]

Con respecto a los **clientes**:

* El nombre de su clase principal será Cliente
* Enviará los mensajes al puerto acordado con el servidor (localhost): el puerto 6000
* Cada vez que se lance la ejecución de un nuevo cliente, mostrará el siguiente mensaje:

PROGRAMA CLIENTE INICIANDO

Introduce una cadena:

* Después de haber introducido la cadena de caracteres (xxxxxxx) que se desea recibir en mayúsculas (XXXXXXX), se mostrará el siguiente mensaje:

=>Respuesta:XXXXXXX

Introduce cadena:

* Después de haber introducido el asterisco para finalizar la comunicación con el servidor, se mostrará el siguiente mensaje:

=>Respuesta:\*

Fin del envío....

* Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* En el supuesto que un determinado cliente quiera recibir en mayúsculas las cadenas *desarrollo* y *multiplataforma* y después finalice la comunicación con el servidor, los resultados que mostrará serán los siguientes:

PROGRAMA CLIENTE INICIANDO

Introduce una cadena:desarrollo

=>Respuesta:DESARROLLO

Introduce cadena:

multiplataforma

=>Respuesta:MULTIPLATAFORMA

Introduce cadena:

\*

=>Respuesta:\*

Fin del envío....

Recomendaciones a la hora de realizar las **pruebas** de esta actividad:

* Iniciar la ejecución del servidor antes que la de los clientes

**Solución**

**// Servidor \\**

**public** **class** Servidor {

**public** **static** **void** main(String[] args) {

**try** {

// CREA EL SOCKET Y LO NOTIFICA \\

ServerSocket serverSocket = **new** ServerSocket(6000);

System.***out***.println("Servidor iniciado. Esperando clientes...");

**while** (**true**) {

// CONECTA CON LOS CLIENTES \\

Socket socket = serverSocket.accept();

System.***out***.println("Cliente Conectado....");

// CREA UN HILO Y LO INICIA \\

HiloServidor hiloServidor = **new** HiloServidor(socket);

hiloServidor.start();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**// HiloServidor \\**

**public** **class** HiloServidor **extends** Thread {

**private** Socket socket;

/\*\*

\* Al crear el hilo vincula el socket con el del servidor.

\* **@param** socket

\*/

**public** HiloServidor(Socket socket) {

**this**.socket = socket;

}

@Override

**public** **void** run() {

**try** {

// NOTIFICA LA COMUNICACIÓN CON EL CLIENTE \\

System.***out***.println("Comunico con: " + socket);

// CREA EL LECTOR Y ESCRITOR \\

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

// BUCLE PARA ESCUCHAR EL CLIENTE \\

String m;

**while** ((m = br.readLine()) != **null**) {

// EN EL MOMENTO QUE EL CLIENTE ENVIA "\*" FINALIZA LA CONEXIÓN \\

**if** (m.equals("\*")) {

System.***out***.println("Fin de la conexión con: " + socket);

**break**;

}

// LEE LA CADENA ENVIADA Y LA DEVUELVE EN MAYUSCULAS \\

System.***out***.println("Recibido: " + m);

String respuesta = m.toUpperCase();

pw.println("=>Respuesta:" + respuesta);

}

// FINALIZA CERRANDO EL LECTOR, ESCRITOR Y SOCKET \\

br.close();

pw.close();

socket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**// Cliente \\**

**public** **class** Cliente {

**public** **static** **void** main(String[] args) {

**try** {

// CREA UN SOCKET QUE MIRA AL PUERTO DEL SERVIDOR \\

Socket socket = **new** Socket("localhost", 6000);

BufferedReader brServidor = **new** BufferedReader(**new** InputStreamReader(System.***in***)); // LECTOR DE LA CONSOLA \\

BufferedReader brConsola = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream())); // LECTOR DEL SERVIDOR \\

PrintWriter pwServidor = **new** PrintWriter(socket.getOutputStream(), **true**); // ESCRITOR DEL SERVIDOR \\

// NOTIFICA EL INICIO DEL PROGRAMA \\

System.***out***.println("PROGRAMA CLIENTE INICIANDO");

**while** (**true**) {

// NOTIFICA POR CONSOLA QUE SE INTRODUZCA CARÁCTERES Y LOS LEE \\

System.***out***.print("Introduce una cadena: ");

String mensaje = brConsola.readLine();

// MANDA LOS CARÁCTERES AL SERVIDOR \\

pwServidor.println(mensaje);

// LEE LA RESPUESTA DEL SERVIDOR Y LO NOTIFICA \\

String respuesta = brServidor.readLine();

System.***out***.println("=>Respuesta:" + respuesta);

// EN EL MOMENTO QUE EL USUARIO ESCRIBA "\*", FINALIZARÁ EL PROCESO \\

**if** (mensaje.equals("\*")) {

System.***out***.println("Fin del envío....");

**break**;

}

}

// FINALIZA CERRANDO LOS LECTORES, ESCRITOR Y SOCKET \\

brConsola.close();

brServidor.close();

pwServidor.close();

socket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**Resolución de problemas**

**CRITERIOS DE CORRECCIÓN**

1. Nivel de ajuste a la nomenclatura de objetos definida en el enunciado de la actividad y nivel de ajuste al almacenamiento de la información requerida en las subcarpetas definidas para la actividad y unidad didáctica correspondientes (15%).
2. Hacer lo que se indica en el enunciado (55%).
3. Claridad del código Java (15%).
4. Documentación del código Java (15%).
5. Explicación detallada de cómo se han resuelto problemas que hayan podido surgir en la realización de la actividad (hasta un 15% adicional sobre la nota obtenida).   
   Dicha explicación se incluirá al principio de los archivos de código fuente Java mediante líneas de comentarios.   
   La primera línea de comentarios sólo incluirá lo siguiente:  
    // Resolución de problemas.
6. Variables no usadas (-15%).
7. Librerías no usadas (-15%).