**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

package ejercicio11;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class Servidor

{

public static void main (String[] args) throws IOException

{

int puerto = 6000;

@SuppressWarnings("resource")

ServerSocket servidor = new ServerSocket(puerto);

while (true)

{

Socket cliente = servidor.accept();

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

HiloServidor hilo = new HiloServidor(cliente);

hilo.start();

}

}

}

package ejercicio11;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class HiloServidor extends Thread {

private Socket cliente;

public HiloServidor(Socket cliente) {

this.cliente = cliente;

}

public void run() {

try {

while (cliente.isClosed() == false) {

System.out.println("Comunico con: " + cliente.toString());

DataInputStream dis = new DataInputStream(cliente.getInputStream());

String texto = dis.readUTF();

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

System.out.println("Fin de la conexión con: " + cliente.toString());

dis.close();

cliente.close();

} else {

DataOutputStream dos = new DataOutputStream(cliente.getOutputStream());

dos.writeUTF(texto.toUpperCase());

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

package ejercicio11;

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

public class Cliente {

public static void main(String[] args) {

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

int puerto = 6000;

try {

String texto = "";

InetAddress ip = InetAddress.getLocalHost();

Socket cliente = new Socket(ip, puerto);

DataOutputStream dos;

DataInputStream dis = null;

while (true) {

System.out.println("Introduce una cadena: ");

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

try {

texto = in.readLine();

} catch (IOException e) {

e.printStackTrace();

}

dos = new DataOutputStream(cliente.getOutputStream());

dos.writeUTF(texto);

if (!texto.equals("\*")) {

dis = new DataInputStream(cliente.getInputStream());

System.out.println(" =>Respuesta: " + dis.readUTF());

} else {

dos.close();

dis.close();

cliente.close();

break;

}

}

} catch (IOException e) {

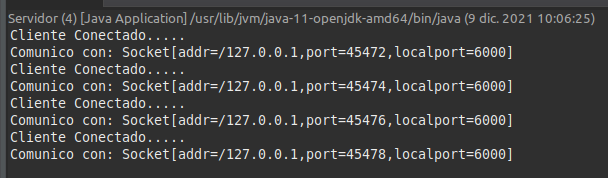
e.printStackTrace();

}

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

}

}



**REQUISITOS**

**REQUISITOS**

1. Entregar las soluciones de las actividades en un único archivo comprimido donde tendremos:
   1. Un archivo de extensión .odt o .docx que contendrá
      1. Una copia del código del programa
      2. Una captura de pantalla del resultado de la ejecución del programa
   2. Los archivos \*.java empleados
2. El único archivo con la soluciones contendrá el nombre original de la actividad, seguido de un guion bajo y del primer apellido del alumno.

**CRITERIOS DE CORRECCIÓN**

1. Cumplimiento de los requisitos (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%).