**ACTIVIDAD 6**

**Enunciado**

Realizar un programa con sockets TCP que establezca un intercambio de información entre un servidor y un único cliente. El intercambio consistirá en que el cliente le enviará al servidor un número y el servidor le devolverá al cliente dicho número elevado al cuadrado.

Con respecto al **cliente**:

* El nombre de su clase principal será Cliente
* Visualizará un mensaje de estado antes de capturar por el teclado un número de tipo primitivo *int*
* Enviará al puerto 6010 del servidor el número capturado por el teclado
* Recibirá del servidor el valor del número enviado, pero elevado al cuadrado
* Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* Suponiendo que el número enviado al servidor sea el 7, el resultado esperado de su ejecución será el siguiente:

PROGRAMA CLIENTE INICIANDO

Introduce un número

7

Recibiendo mensaje del servidor:

El cuadrado del número 7 es 49

Con respecto al **servidor**:

* El nombre de su clase principal será Servidor
* El puerto por el que escuchará las peticiones del cliente será el 6010
* Visualizará un mensaje de estado antes de crear el flujo de entrada del cliente
* Recibirá del cliente el valor de un número y lo elevará al cuadrado
* Devolverá al cliente el valor del número elevado al cuadrado, pero dentro de un stream cuyo texto completo aparece en la última línea que visualiza el cliente
* Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* El resultado esperado de su ejecución será el siguiente:

Esperando al cliente.....

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

* Iniciar la ejecución del servidor antes que la del cliente
* Variar de puerto si surgen problemas

**Solución**

Servidor:

import java.io.\*;

import java.net.\*;

public class Servidor {

public static void main(String[] args) {

final int PUERTO = 6010;

try {

ServerSocket serverSocket = new ServerSocket(PUERTO);

System.***out***.println("Esperando al cliente.....");

Socket clientSocket = serverSocket.accept();

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

// Flujo de entrada del cliente

BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

// Flujo de salida al cliente

DataOutputStream outToClient = new DataOutputStream(clientSocket.getOutputStream());

// Recibir el número del cliente

int numero = Integer.*parseInt*(inFromClient.readLine());

System.***out***.println("Recibido número del cliente: " + numero);

// Calcular el cuadrado

int cuadrado = numero \* numero;

// Enviar el resultado al cliente

outToClient.writeBytes("El cuadrado del número " + numero + " es " + cuadrado + "\n");

// Cerrar streams y sockets

inFromClient.close();

outToClient.close();

clientSocket.close();

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Cliente:

import java.io.\*;

import java.net.\*;

public class Cliente {

public static void main(String[] args) {

final String SERVER\_IP = "localhost";

final int PUERTO = 6010;

try {

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

// Flujo de entrada estándar para leer desde el teclado

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

// Conectar con el servidor

Socket clientSocket = new Socket(SERVER\_IP, PUERTO);

// Flujo de salida al servidor

DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

// Flujo de entrada del servidor

BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

System.***out***.println("Introduce un número:");

// Leer el número desde el teclado

int numero = Integer.*parseInt*(reader.readLine());

// Enviar el número al servidor

outToServer.writeBytes(numero + "\n");

// Recibir y mostrar el resultado del servidor

String resultado = inFromServer.readLine();

System.***out***.println("Recibiendo mensaje del servidor:\n\t" + resultado);

// Cerrar streams y socket

outToServer.close();

inFromServer.close();

clientSocket.close();

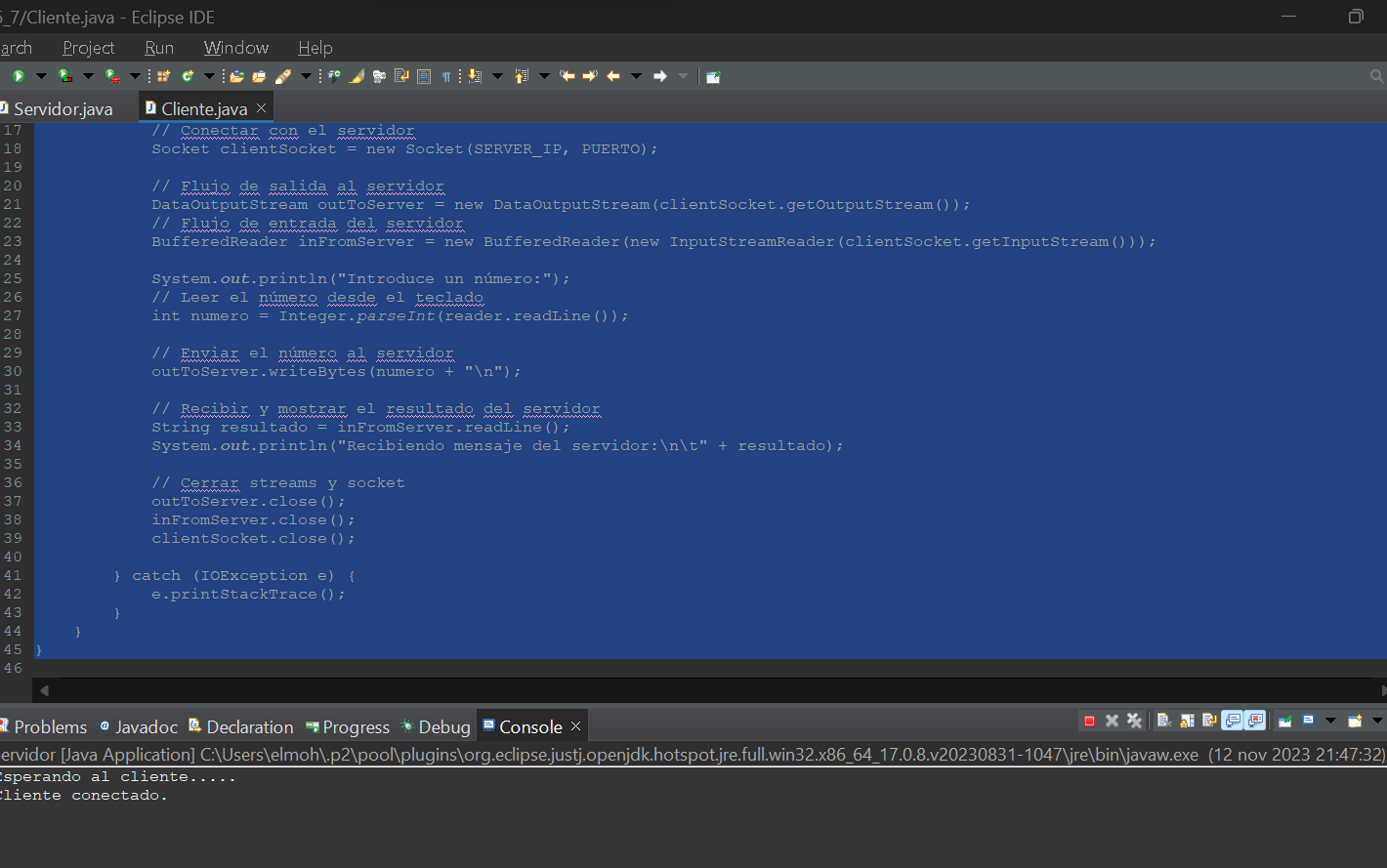
} catch (IOException e) {

e.printStackTrace();

}

}

}



Ejecutamos el servidor primero

**Resolución de problemas**

**ACTIVIDAD 7**

**Enunciado**

Realizar un programa con sockets TCP en el que un servidor atienda las peticiones de 3 clientes y les envíe un mensaje de saludo.

Con respecto a los **clientes**:

* El nombre de su clase principal será Cliente
* Visualizará un mensaje de estado antes de enviar al servidor la petición a través del puerto 6013
* Visualizará el mensaje recibido del servidor
* Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* El resultado esperado de su ejecución durante las 3 primeras peticiones será el siguiente:

PROGRAMA CLIENTE INICIANDO

Recibiendo mensaje del servidor:

Hola cliente 1

PROGRAMA CLIENTE INICIANDO

Recibiendo mensaje del servidor:

Hola cliente 2

PROGRAMA CLIENTE INICIANDO

Recibiendo mensaje del servidor:

Hola cliente 3

* El resultado esperado de su ejecución a partir de la 4ª petición al servidor es opcional, pero el siguiente es orientativo:

PROGRAMA CLIENTE INICIANDO

Connection refused: connect

Con respecto al **servidor**:

* El nombre de su clase principal será Servidor
* El puerto por el que escuchará las peticiones de los clientes será el 6013
* Visualizará un mensaje de estado antes de iniciar el tratamiento de las peticiones de los clientes
* Para cada uno de los 3 clientes:
  + Le enviará un mensaje de saludo al cliente y será este último quien lo visualice
  + Cerrará los streams y sockets en el orden recomendado
* Realizará la necesaria gestión de excepciones
* El resultado esperado de su ejecución será el siguiente:

Esperando a los clientes.....

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

* Iniciar la ejecución del servidor antes que la del cliente
* Variar de puerto si surgen problemas

**Solución**

Servidor:

package paquete\_actividad\_7;

import java.io.\*;

import java.net.\*;

public class Servidor {

public static void main(String[] args) {

final int PUERTO = 6013;

try {

ServerSocket serverSocket = new ServerSocket(PUERTO);

System.***out***.println("Esperando a los clientes.....");

for (int i = 0; i < 3; i++) {

Socket clientSocket = serverSocket.accept();

System.***out***.println("Cliente " + (i + 1) + " conectado.");

// Flujo de salida al cliente

PrintWriter outToClient = new PrintWriter(clientSocket.getOutputStream(), true);

// Enviar mensaje de saludo al cliente

outToClient.println("Hola cliente " + (i + 1));

// Cerrar streams y socket para este cliente

outToClient.close();

clientSocket.close();

}

// Cerrar el socket del servidor

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Cliente:

package paquete\_actividad\_7;

import java.io.\*;

import java.net.\*;

public class Cliente {

public static void main(String[] args) {

final String SERVER\_IP = "localhost";

final int PUERTO = 6013;

try {

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

// Conectar con el servidor

Socket clientSocket = new Socket(SERVER\_IP, PUERTO);

// Flujo de entrada del servidor

BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

// Recibir y mostrar el mensaje del servidor

String mensaje = inFromServer.readLine();

System.***out***.println("Recibiendo mensaje del servidor:\n\t" + mensaje);

// Cerrar stream y socket

inFromServer.close();

clientSocket.close();

} catch (ConnectException e) {

System.***out***.println("Connection refused: connect");

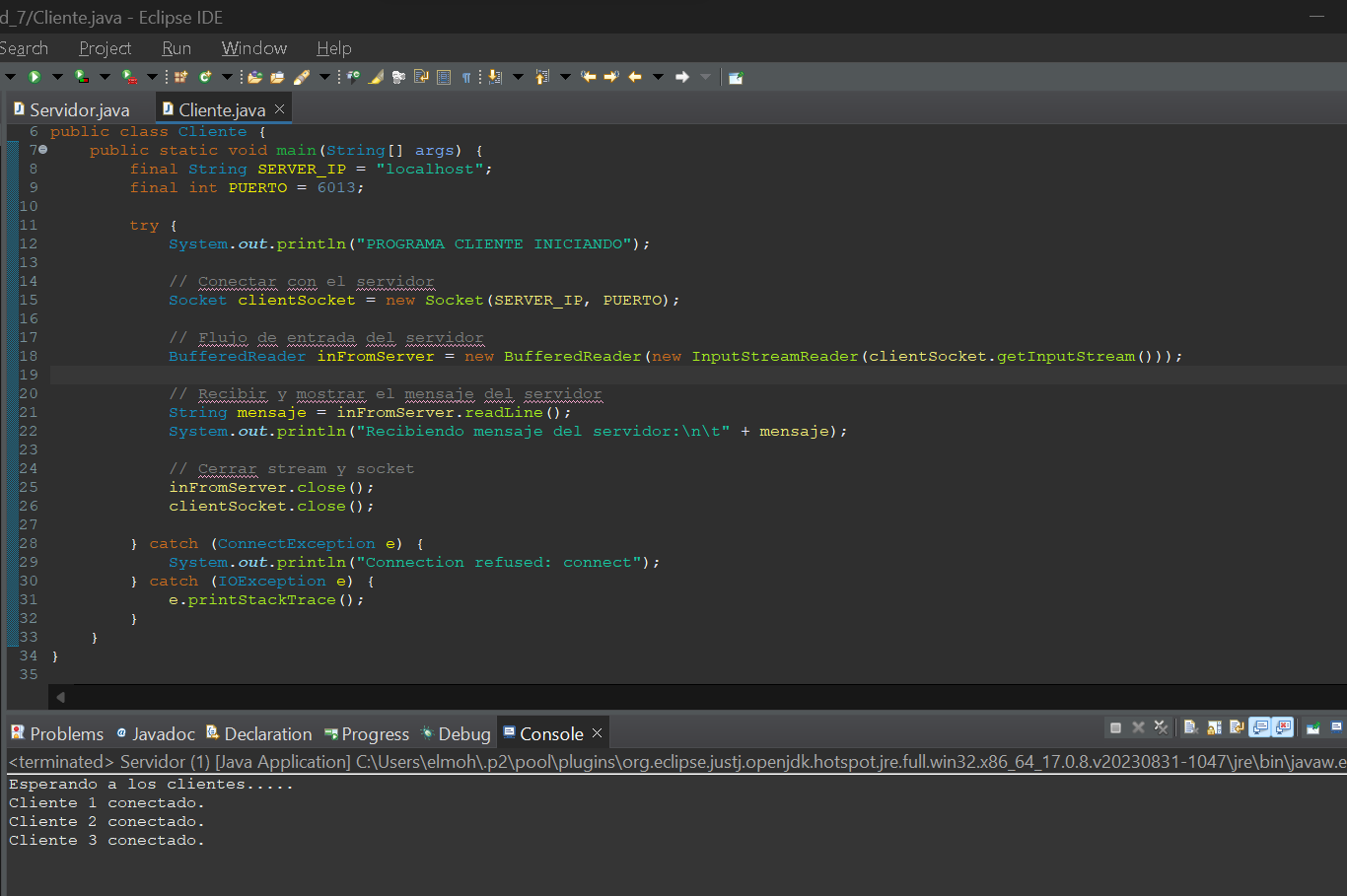
} catch (IOException e) {

e.printStackTrace();

}

}

}



Ejecutamos el servidor primero

**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%).