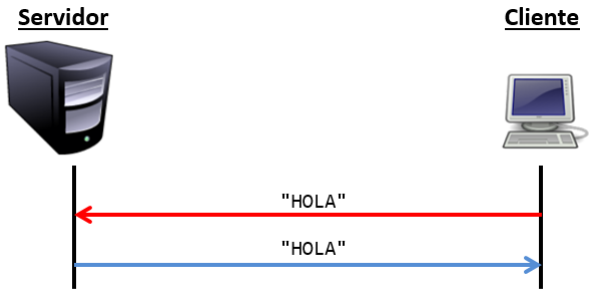
Taller 5 – Servidores Concurrentes

El propósito de este taller es entender la concurrencia en servidores a través de la construcción de una aplicación cliente – servidor, en la cual vamos a construir un servidor concurrente que responda a múltiples pedidos de varios clientes. El servidor recibe los pedidos y envía las respuestas por medio de sockets, uno de los medios tradicionales de comunicación entre aplicaciones en Internet. Esta aplicación está basada en el protocolo TCP.

Parte 1: Aplicación Cliente – Servidor de Eco

Un servidor de eco es una aplicación en la que el servidor está a la espera de que uno o más clientes le envíen mensajes. El servidor recibe un mensaje y lo devuelve al cliente que se lo envió.



a) Monothread

Para desarrollar 1: Implementar la aplicación de Eco monothread en Java

Vamos a desarrollar dos aplicaciones.

La aplicación cliente tiene dos clases:

(1) La clase **`Cliente`**
(2) La clase **`ProtocoloCliente`**

La aplicación servidor tiene dos clases:

(1) La clase **`Servidor`**
(2) La clase **`ProtocoloServidor`**

`Cliente.java`

```
public static final int PUERTO = 3400;
public static final String SERVIDOR = "localhost";

public static void main(String args[]) throws IOException {
    Socket socket = null;
    PrintWriter escritor = null;
    BufferedReader lector = null;

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

    try {
        // crea el socket en el lado cliente
        socket = new Socket(SERVIDOR, PUERTO);
```

```
        // se conectan los flujos, tanto de salida como de entrada
        escritor = new PrintWriter(socket.getOutputStream(), true);
        lector = new BufferedReader(new InputStreamReader(
                socket.getInputStream()));
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }

    // crea un flujo para leer lo que escribe el cliente por el teclado
    BufferedReader stdIn = new BufferedReader(new InputStreamReader(
            System.in));

    // se ejecuta el protocolo en el lado cliente
    ProtocoloCliente.procesar(stdIn, lector, escritor);

    // se cierran los flujos y el socket
    stdIn.close();
    escritor.close();
    lector.close();
    socket.close();
}
```

ProtocoloCliente.java

```
public static void procesar(BufferedReader stdIn, BufferedReader pIn, PrintWriter pOut)
        throws IOException {

    // lee del teclado
    System.out.println("Escriba el mensaje para enviar: ");
    String fromUser = stdIn.readLine();

    // envía por la red
    pOut.println(fromUser);

    String fromServer = "";

    // lee lo que llega por la red
    // si lo que llega del servidor no es null
    // observe la asignación  luego la condición
    if ((fromServer = pIn.readLine()) != null) {
        System.out.println("Respuesta del Servidor: " + fromServer);
    }
}
```

Servidor.java

```
public static final int PUERTO = 3400;

public static void main(String args[]) throws IOException {
    ServerSocket ss = null;
    boolean continuar = true;

    System.out.println("Main Server ...");

    try {
        ss = new ServerSocket(PUERTO);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }
    while (continuar) {
        // crear el socket en el lado servidor
        // queda bloqueado esperando a que llegue un cliente
        Socket socket = ss.accept();


        try {
            // se conectan los flujos, tanto de salida como de entrada
            PrintWriter escritor = new PrintWriter(
                    socket.getOutputStream(), true);
            BufferedReader lector = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));

            // se ejecuta el protocolo en el lado servidor
            ProtocoloServidor.procesar(lector, escritor);

            // se cierran los flujos y el socket
            escritor.close();
            lector.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ProtocoloServidor.java

```
public class ProtocoloServidor {
    // observe que es un método estático.
    // observe que lanza excepciones de tipo IOException
    public static void procesar(BufferedReader pIn, PrintWriter pOut)
            throws IOException {
        String inputLine;
        String outputLine;

        // lee del flujo de entrada
        inputLine = pIn.readLine();
        System.out.println("Entrada a procesar: " + inputLine);

        // procesa la entrada
        outputLine = inputLine;

        // escribe en el flujo de salida
        pOut.println(outputLine);
        System.out.println("Salida procesada: " + outputLine);
    }
}
```

Ejecución

Para la ejecución de la aplicación, no olvide primero lanzar el **`Servidor`** y luego el **`Cliente`**.

Lado del Servidor	Lado del Cliente

`Main Server ...`
`Entrada a procesar: Hola`
`Salida procesada: Hola`
	`Cliente ...`
	`Escriba el mensaje para enviar:`
	`Hola`
	`Respuesta del Servidor: Hola`

Realice las siguientes actividades y responda las preguntas:

1. Teniendo el servidor en ejecución, ejecútelo nuevamente. Como resultado, se lanza una excepción. ¿Qué tipo de excepción es lanzada? ¿Cuál es el mensaje asociado con esa excepción? ¿Qué pasó? ¿Cómo puede solucionarse?

2. Cambie la línea **`e.printStackTrace();`** justo después de crear el socket **`ss`** en el servidor. En su lugar, escriba **`System.err.println("No se pudo crear el socket en el puerto: " + PUERTO);`**. Eejecute otra vez el servidor. Como resultado, se lanza una excepción. ¿Cuál es el mensaje asociado?

3. Qué pasa si el cliente intenta conectarse a la máquina correcta en el puerto equivocado. Como resultado, se lanza una excepción. ¿Qué tipo de excepción es lanzada? ¿Cuál es el mensaje asociado con esa excepción?

4. ¿Por qué el servidor está ejecutando en el puerto 3400? ¿Por qué es ese número y no otro?

5. Suponga que se desea enviar dos mensajes al servidor. ¿Qué modificaciones haría? Considere diferentes opciones.

6. Utilizando el programa cliente, conéctese con el servidor de un compañero. ¿Qué tiene que hacer para que se pueda realizar la comunicación?

b) Multithread

Se trata de la misma aplicación, con el mismo protocolo, pero esta vez, el servidor será multithread. Usaremos un esquema de servidor construido sobre dos clases: (1) el servidor principal, el cual coordina la recepción de mensajes sobre un socket y la creación de threads para atender las solicitudes correspondientes, y (2) el servidor delegado que se ejecuta en un thread y atiende un cliente, debe además implementar el protocolo de comunicación con el cliente.

Para desarrollar 2: Implementar la aplicación de Eco multithread en Java

Vamos a desarrollar dos aplicaciones.

La aplicación cliente tiene dos clases:

(1) La clase **`Cliente`**
(2) La clase **`ProtocoloCliente`**

La aplicación servidor tiene tres clases:

(1) La clase **`Servidor`**
(2) La clase **`ProtocoloServidor`**
(3) La clase **`ThreadServidor`**

Dado que el cambio se realiza únicamente en el lado servidor, las clases **`Cliente`** y **`ProtocoloCliente`** son las mismas de la aplicación monothread. Lo mismo sucede con la clase **`ProtocoloServidor`**, dado que se trata del mismo protocolo.

Servidor.java

Definición de una variable para controlar el número de threads:

```
int numeroThreads = 0;
```

Dado que se trata de la misma aplicación, hay algunos los cambios en el método **main()** de la clase **Servidor**, especialmente en el ciclo **while**:

```
while (continuar) {
    // crear el socket
    Socket socket = ss.accept();

    // crear el thread con el socket y el id
    ThreadServidor thread = new ThreadServidor(socket, numeroThreads);
    numeroThreads++;

    // start
    thread.start();
}
ss.close();
```

ThreadServidor.java

```
public class ThreadServidor extends Thread {
    private Socket sktCliente = null;
    private int id;

    public ThreadServidor(Socket pSocket, int pId) {
        this.sktCliente = pSocket;
        this.id = pId;
    }
public void run() {
    System.out.println("Inicio de un nuevo thread: " + id);

    try {
        PrintWriter escritor = new PrintWriter(
                sktCliente.getOutputStream(), true);
        BufferedReader lector = new BufferedReader(new InputStreamReader(
                sktCliente.getInputStream()));

        ProtocoloServidor.procesar(lector, escritor);

        escritor.close();
        lector.close();
        sktCliente.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Ejecución

Recuerde lanzar primero el **`Servidor`**.

Lado del Servidor	Lado del Cliente

`Main Server ...`
`Inicio de un nuevo thread: 0`
`Entrada a procesar: Hola`
`Salida procesada: Hola`
	`Cliente ...`
	`Escriba el mensaje para enviar:`
	`Hola`
	`Respuesta del Servidor: Hola`

Realice las siguientes actividades:

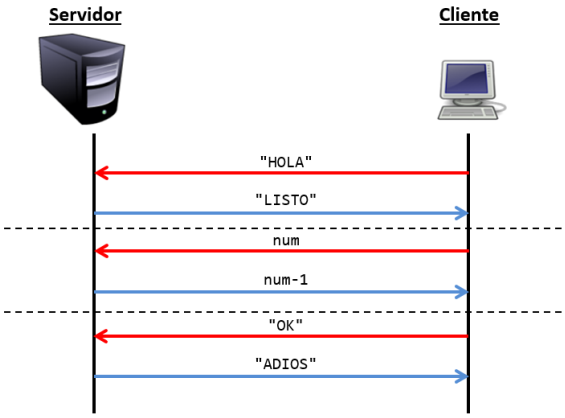
1. Ejecute al mismo tiempo el servidor monothread y el servidor multithread en el mismo computador. Como resultado, se lanza una excepción. ¿Qué tipo de excepción es lanzada? ¿Cuál es el mensaje asociado con esa excepción? ¿Qué pasó? ¿Cómo puede solucionarse? ¿Qué debe tener en cuenta para que funcione?

2. Qué se puede hacer para que el programa cliente pueda enviar 100 mensajes al servidor, sin modificar el código del lado servidor.

3. Qué se puede hacer para que el programa cliente para que pueda enviar mensajes a tres servidores diferentes.

Parte 2: Servidor concurrente

En esta aplicación, el servidor recibe mensajes de múltiples clientes y envía las respuestas por medio de sockets. En este caso, tendremos un servidor principal que se encarga de recibir las solicitudes de conexión de los clientes y crear un servidor delegado por cada solicitud. Cada servidor delegado termina cuando el cliente termina la conexión. La siguiente figura ilustra el protocolo de comunicación entre el servidor y el cliente.



Ejecución

Lado del Servidor	Lado del Cliente
Main Server ... Inicio de un nuevo thread: 0 Entrada a procesar: Hola Entrada a procesar: 100 Entrada a procesar: OK	Cliente ... Escriba el mensaje para enviar: Hola El usuario escribió: Hola Respuesta del Servidor: Listo Escriba el mensaje para enviar: 100 El usuario escribió: 100 Respuesta del Servidor: 99 Escriba el mensaje para enviar: OK El usuario escribió: OK Respuesta del Servidor: ADIOS

Para desarrollar 3: Implementar la aplicación de servidores concurrentes en Java

Vamos a desarrollar dos aplicaciones.

La aplicación cliente tiene dos clases:

(1) La clase **`Cliente`**
(2) La clase **`ProtocoloCliente`**

La aplicación servidor tiene tres clases:

(1) La clase **`Servidor`**
(2) La clase **`ProtocoloServidor`**
(3) La clase **`ThreadServidor`**

A continuación, se incluye parte del código de las clases de este proyecto. Usted debe entender el código y completarlo. Para ello, revise los comentarios. Todo lo que aparece entre los signos **<>**, es codigo que debe completar.

Cliente.java

```
public class Cliente {
    public static final int PUERTO = 3400;
    public static final String SERVIDOR = "localhost";

    public static void main(String args[]) throws IOException {
        Socket socket = null;
        PrintWriter escritor = null;
        BufferedReader lector = null;

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

        try {
            <aquí va el socket> = new Socket(<aquí va la ubicación del servidor>,
                    <aquí va el puerto>);

            escritor = <complete>
            lector = <complete>

        } catch (IOException e) {
            System.err.println("Exception: " + e.getMessage());
            System.exit(1);
        }

        BufferedReader stdIn = <complete>

        // se ejecuta el protocolo en el lado cliente
        ProtocoloCliente.procesar(stdIn, lector, escritor);

        escritor.close();
        lector.close();
        socket.close();
        stdIn.close();
    }
}
```

ProtocoloCliente.java

```
public class ProtocoloCliente {
    public static void procesar(BufferedReader stdIn, BufferedReader pIn,
            PrintWriter pOut) throws IOException {

        String fromServer;
        String fromUser;

        boolean ejecutar = true;

        while (ejecutar) {
            // lee del teclado
            System.out.println("Escriba el mensaje para enviar: ");
            fromUser = stdIn.readLine();

            // si lo que ingresa el usuario no es null y es diferente de "-1"
            // if (fromUser != null && !fromUser.equals("-1")) {
            if (fromUser != null) {
                System.out.println("El usuario escribió: " + fromUser);
                // si lo que ingresa el usuario es "OK"
                if (fromUser.equalsIgnoreCase("OK")) {
                    ejecutar = false;
                }

                // envía por la red
                pOut.println(fromUser);
            }

            // lee lo que llega por la red
            // si lo que llega del servidor no es null
            // observe la asignación luego la condición
            if ((fromServer = pIn.readLine()) != null) {
                System.out.println("Respuesta del Servidor: " + fromServer);
            }
        }
    }
}
```

Servidor.java

```
public static void main(String args[]) throws IOException {
    ServerSocket ss = null;
    boolean continuar = true;
    <aquí va una variable para controlar los identificadores de los threads>

    System.out.println("Main Server ...");

    try {
        ss = new ServerSocket(<aquí va el número de puerto>);
    } catch (IOException e) {
        System.err.println("No se pudo crear el socket en el puerto: "
                + <aquí va el número de puerto>);
        System.exit(-1);
    }

    while (continuar) {
        // crear el thread y lanzarlo.

        // crear el socket
        Socket socket = ss.accept();

        // crear el thread con el socket y el id
        ThreadServidor thread = new ThreadServidor(<aquí va el socket>,
                <aquí va el identificador del thread>);
        <aquí debe hacer una modificación porque todos los threads
        tienen un identificador diferente>

        // start
        thread.start();
    }
    ss.close();
}
```

ThreadServidor.java

```
public class ThreadServidor extends Thread {
    private Socket sktCliente = null;
    <aquí va un atributo para identificar el thread>

    public ThreadServidor(Socket pSocket, int pId) {
        <complete>
    }

    public void run() {
        System.out.println("Inicio de un nuevo thread: " + id);

        try {
            PrintWriter escritor = <complete>
            BufferedReader lector = <complete>

            ProtocoloServidor.procesar(lector, escritor);

            escritor.close();
            lector.close();
            sktCliente.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ProtocoloServidor.java

```
public static void procesar(BufferedReader pIn, PrintWriter pOut)
        throws IOException {
    String inputLine;
    String outputLine;
    int estado = 0;

    while (estado < 3 && (inputLine = pIn.readLine()) != null) {

        System.out.println("Entrada a procesar: " + inputLine);
        switch (estado) {
        case 0:
            if (inputLine.equalsIgnoreCase("Hola")) {
                outputLine = "Listo";
                estado++;
            } else {
                outputLine = "ERROR. Esperaba Hola";
                estado = 0;
            }
            break;

       case 1:
            try {
                int val = Integer.parseInt(inputLine);
                val--;
                outputLine = "" + val;
                estado++;
            } catch (Exception e) {
                outputLine = "ERROR en argumento esperado";
                estado = 0;
            }
            break;
       case 2:
            if (inputLine.equalsIgnoreCase("OK")) {
                outputLine = "ADIOS";
                estado++;
            } else {
                outputLine = "ERROR. Esperaba OK";
                estado = 0;
            }
            break;
```

```
            default:
                outputLine = "ERROR";
                estado = 0;
            }
            pOut.println(outputLine);
        }
    }
```

Para desarrollar en casa: Pool de threads

Vamos a utilizar un pool de threads para responder las solicitudes de los clientes. Es necesario modificar el método main() de la clase Servidor para esto. Agregue el siguiente código. No solo copie el código, intente analizarlo y consultar Clases y Objetos que no conozca.

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

    int N_THREADS = 10;

    final ExecutorService pool = Executors.newFixedThreadPool(N_THREADS);

    ServerSocket servSock = null;
    try{
        servSock = new ServerSocket(PUERTO);
        System.out.println("Listo para recibir conexiones");
        while(true){
            Socket cliente = servSock.accept();

            pool.execute(new ProtocoloServidor(cliente));
        }
    }catch(Exception e){
        System.err.println("Ocurrio un error");
        e.printStackTrace();
    }finally{
        try{
        servSock.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

De ser necesario, modifique la clase ProtocoloServidor para que luzca de esta manera.

```
public ProtocoloServidor(Socket s){
    this.sockCliente = s;
    try {
        escritor = new PrintWriter(
                sockCliente.getOutputStream(), true);
        lector = new BufferedReader(new InputStreamReader(
                sockCliente.getInputStream()));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void procesar(BufferedReader pIn, PrintWriter pOut)
        throws IOException {


    // lee del flujo de entrada
    String inputLine = pIn.readLine();
    System.out.println("Entrada a procesar: " + inputLine);

    // procesa la entrada
    String outputLine = inputLine;

    // escribe en el flujo de salida
    pOut.println(outputLine);
    System.out.println("Salida procesada: " + outputLine);
}
```

Para desarrollar en casa: Servidor concurrente en Lenguaje C

Trabajaremos con procesos para implementar servidores delegados. Usaremos un tutorial disponible en Internet: http://www.linuxhowtos.org/C_C++/socket.htm. Esta es una recomendación, usted puede usar otro tutorial si lo considera conveniente.

Servidor

En el tutorial encontrará el código del servidor (**`servidor.c`**), pero el código disponible no es concurrente. Usted debe hacer modificaciones para transformarlo de forma apropiada. En el mismo sitio, hacia el final, encontrará una versión concurrente (en la sección Enhancements to the server code).

Cliente

En el tutorial encontrará el código del cliente (**`cliente.c`**)