

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

UF3 PAC DESARROLLO

1. Realiza una comparación entre UDP y TCP.

El **protocolo UDP (User Datagram Protocol)**, es un protocolo no orientado a conexión, cuando una máquina envía paquetes a otra el flujo es unidireccional. La transferencia se realiza sin prevenir al destinatario y éste recibe los datos sin enviar una confirmación al emisor.

Los datos enviados por este protocolo no permiten transmitir la información relacionada al emisor, por ello el destinatario no conocerá al emisor de los datos, excepto su IP.

Por otro lado el **protocolo TCP (Transmission Control Protocol)** está orientado a conexión. Cuando una máquina enviada datos a otra, esta segunda es informada de la llegada de estos y confirma su recepción.

Interviene pues el control de datos que se basa en una ecuación matemática que permite verificar la integridad de los datos transmitidos. Así si los datos recibidos son corruptos este protocolo permite que los destinatarios soliciten al emisor que los vuelva a enviar.

2. Responde a este test.

1. Las capas de abstracción de TCP/IP son:

a. Nivel de aplicación, nivel de transporte y nivel de red.

b. Nivel de aplicación, nivel de transporte, nivel de Internet y nivel de red.

c. Nivel de aplicación, nivel de transporte y nivel de enlace.

d. Nivel de aplicación, nivel de transporte, nivel de enlace y nivel de comunicación.

2. ¿Cuál es el protocolo basado en la conexión?

a. TCP.

b. UDP.

c. Los dos.

d. Ninguno de estos dos.

3. UDP

- a. El orden de entrega es importante.
- b. Se garantiza la recepción de los paquetes enviados.

c. Envía paquetes independientes.

- d. Envía paquetes dependientes.

4. El API de bajo nivel de Java se ocupa de

- a. Las direcciones y las URIs.
- b. Las URIs y las URLs.

c. Los sockets y las interfaces.

- d. Los sockets y las conexiones.

5. ¿Cuál de estas subclases no pertenece a la clase InetAddress?

- a. Inet4Address.

b. Inet8Address.

- c. Inet6Address.

- d. Todas son subclases de InetAddress.

6. Si tenemos la siguiente dirección: <http://www.ucm.es:80/BUCM/servicios/5760.php> ¿Cuál es el puerto de conexión?

- a. 5760

- b. 57

- c. 60

d. 80

7. Señala la respuesta correcta:

- a. Los sockets utilizan la abstracción de los protocolos TCP y UDP.

- b. Los protocolos TCP utilizan la abstracción de los sockets.

- c. Los protocolos UDP utilizan la abstracción de los sockets.

d. Tanto los protocolos UDP como los TCP utilizan la abstracción de los sockets.

8. Para conectarnos a un servidor SMTP

- a. Necesitamos conocer su puerto.
- b. Es suficiente con conocer el nombre del servidor.
- c. Necesitamos tanto el nombre del servidor como el puerto.**
- d. Es imposible conectar con un servidor SMTP.

9. Los sockets...

- a. Pueden ser orientados a conexión.
- b. Pueden ser no orientados a conexión.
- c. Tenemos ambos tipos.**
- d. No se distinguen por tipos.

3. Realiza una comparación entre SOAP y REST.

Los servicios **SOAP**, son servicios que basan su comunicación bajo el protocolo SOAP (Simple Object Access Protocol), que es el protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Los servicios SOAP funcionan por lo general por el protocolo HTTP, que es lo más común cuando invocamos un Web Service, sin embargo SOAP no está limitado a este protocolo, si no que puede ser enviado por FTP, POP3, TCP. Es un protocolo robusto con un tipado fuerte, que permite agregar metadatos mediante atributos. Pero por ello es más pesado, tanto en tamaño como en procesamiento, pues los XML tienen que ser parseados a un árbol DOM.

REST por el contrario es una tecnología flexible que transporta datos por medio del protocolo HTTP, pero permite utilizar los diversos métodos de HTTP para comunicarse, como, get, post, put, delete, patch, y a la vez utiliza códigos de respuesta nativos de HTTP (404, 200, 204, 409).

En REST podemos mandar cualquier tipo de dato, por lo que además de mandar XML, podemos mandar JSON, y binarios, text, etc... Mientras que en SOAP solo podemos mandar XML.

La mayoría son transmitidos por JSON puesto que es interpretado de forma natural por JavaScript.

4. Escribe un programa que cuente el número de conexiones que vaya recibiendo. Este programa dispondrá de un socket stream servidor. Cada vez que un socket cliente se conecte, este le enviará un mensaje con el número de clientes conectados hasta ahora. Así pues, el primer cliente que se conecte recibirá un 1, el segundo un 2, el tercero un 3, etc.

En primer lugar deberemos ejecutar la clase servidor para poder ejecutar la clase cliente, de lo contrario no nos funcionarían los programas.

Por una parte tenemos la clase Servidor:

```
package UF3_Sockets;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

//EJECUTAMOS PRIMERO EL SERVIDOR ANTES DE CONECTAR CLIENTES

public class Server {
    public static void main(String[] args) {

        //Creamos un objeto servidor y ejecutamos el mismo.

        Server server = new Server();
        server.run();
    }

    //Variables y objetos para establecer el contador de clientes:

    //1 - Contador de clientes

    int counter;

    //2 - Socket servidor - Socket cliente

    ServerSocket ssckt;
    Socket clientSckt;

    //3 - Flujo output --> datos al cliente

    DataOutputStream output;

    public Server() {
        try {

            //Fijamos el puerto en el 5000

            ssckt = new ServerSocket(5000);
```

```
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    counter = 0;  
    clientSckt = null;  
}
```

//Ejecución

```
public void run() {  
    System.out.println("El servidor se encuentra escuchando en el puerto 5000.");  
  
    while (true) {  
        try {  
  
            // Escuchando hasta que entra cliente.  
  
            clientSckt = ssckt.accept();  
            System.out.println("Ha accedido un nuevo cliente");  
  
            //Intercambio de datos con el cliente:  
  
            output = new DataOutputStream(clientSckt.getOutputStream());  
  
            //Sumamos un cliente, enviamos contador al cliente (que es su identificador)  
  
            output.writeInt(++counter);  
  
            //Cerramos el output y el socket cliente  
  
            output.close();  
            clientSckt.close();  
  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

package UF3_Sockets;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

//EJECUTAMOS PRIMERO EL SERVIDOR ANTES DE CONECTAR CLIENTES

public class Server {
    public static void main(String[] args) {

        //Creamos un objeto servidor y ejecutamos el mismo.

        Server server = new Server();
        server.run();
    }

    //Variables y objetos para establecer el contador de clientes:

    //1 - Contador de clientes

    int counter;

    //2 - Socket servidor - Socket cliente

    ServerSocket ssckt;
    Socket clientSckt;

    //3 - Flujo output --> datos al cliente

    DataOutputStream output;

    public Server() {
        try {

            //Fijamos el puerto en el 5000

            ssckt = new ServerSocket(5000);

        } catch (IOException e) {
            e.printStackTrace();
        }
        counter = 0;
        clientSckt = null;
    }

    //Ejecución

    public void run() {
        System.out.println("El servidor se encuentra escuchando en el puerto 5000.");

        while (true) {
            try {

                // Escuchando hasta que entra cliente.

                clientSckt = ssckt.accept();
            }

```

```

//3 - Flujo output --> datos al cliente
DataOutputStream output;

public Server() {
    try {

        //Fijamos el puerto en el 5000

        ssckt = new ServerSocket( port: 5000);

    } catch (IOException e) {
        e.printStackTrace();
    }
    counter = 0;
    clientSckt = null;
}

//Ejecución

public void run() {
    System.out.println("El servidor se encuentra escuchando en el puerto 5000.");

    while (true) {
        try {

            // Escuchando hasta que entra cliente.

            clientSckt = ssckt.accept();
            System.out.println("Ha accedido un nuevo cliente");

            //Intercambio de datos con el cliente:

            output = new DataOutputStream(clientSckt.getOutputStream());

            //Sumamos un cliente, enviamos contador al cliente (que es su identificador)

            output.writeInt(++counter);

            //Cerramos el output y el socket cliente

            output.close();
            clientSckt.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```


Por otra parte tenemos la clase Cliente:

```
package UF3_Sockets;

import java.io.DataInputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

//EJECUTAMOS PRIMERO EL SERVIDOR ANTES DE CONECTAR CLIENTES

public class Client {

    public static void main(String[] args) {

        //Creamos un objeto cliente y ejecutamos el mismo.

        Client client = new Client();
        client.run();
    }

    //Objetos Socket y flujo:

    //1 - Socket cliente que se conecta al servidor

    Socket clientSckt;

    //2 - Stream de entrada que recibe los datos del servidor

    DataInputStream input;

    public Client() {

        try {

            //Conexion al puerto donde escucha el servidor

            clientSckt = new Socket("localhost", 5000);

            //Flujo de entrada, obteniendo datos del servidor

            input = new DataInputStream(clientSckt.getInputStream());

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Ejecución

    public void run() {
        try {
```

```
//Imprimimos los datos que nos llegan del servidor

System.out.println("Tu código de cliente es: " + input.readInt());

//Se cierra el stream y el socket

input.close();

clientSckt.close();

} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
```

```

package UF3_Sockets;

import java.io.DataInputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

//EJECUTAMOS PRIMERO EL SERVIDOR ANTES DE CONECTAR CLIENTES

public class Client {

    public static void main(String[] args) {

        //Creamos un objeto cliente y ejecutamos el mismo.

        Client client = new Client();
        client.run();
    }

    //Objetos Socket y flujo:

    //1 - Socket cliente que se conecta al servidor
    Socket clientSckt;

    //2 - Stream de entrada que recibe los datos del servidor
    DataInputStream input;

    public Client() {

        try {

            //Conexion al puerto donde escucha el servidor
            clientSckt = new Socket( host: "localhost", port: 5000);

            //Flujo de entrada, obteniendo datos del servidor
            input = new DataInputStream(clientSckt.getInputStream());

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    //Ejecución

    public void run() {

        try {

            //Imprimimos los datos que nos llegan del servidor
            System.out.println("Tu código de cliente es: " + input.readInt());

            //Se cierra el stream y el socket

```

```

//Ejecución
public void run() {
    try {
        //Imprimimos los datos que nos llegan del servidor
        System.out.println("Tu código de cliente es: " + input.readInt());

        //Se cierra el stream y el socket
        input.close();

        clientSckt.close();
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

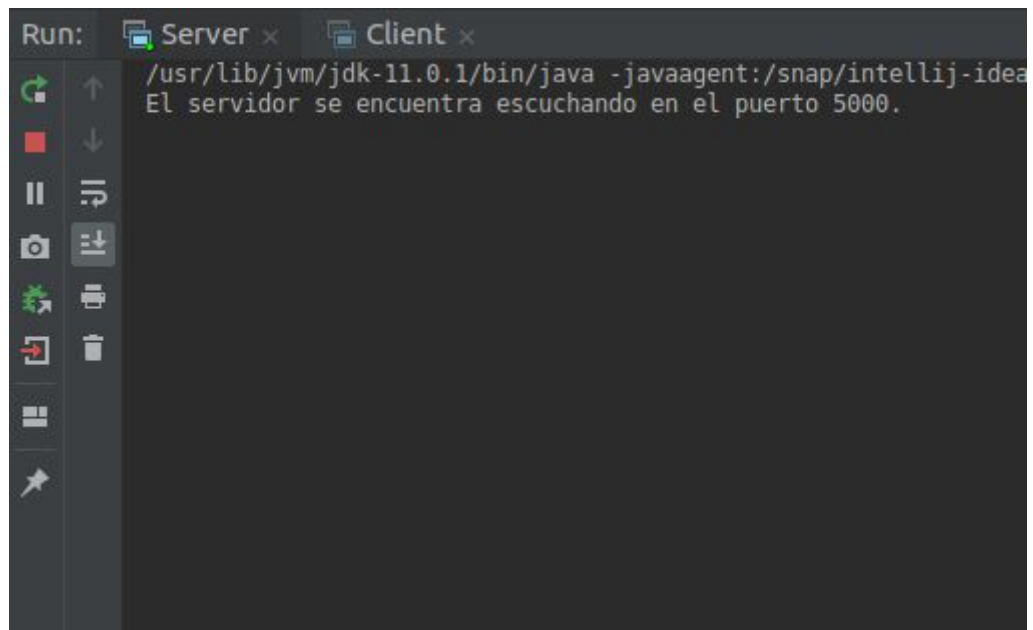
```

En primer lugar deberemos ejecutar la clase servidor para poder ejecutar la clase cliente, de lo contrario no nos funcionarían los programas. El procedimiento en ambas clases es muy parecido.

Primero se crea un objeto servidor o cliente. Después creamos un objeto Socket cliente o servidor (y un contador en el caso de servidor). Después generamos el objeto stream bien de input para cliente, bien de output para servidor.

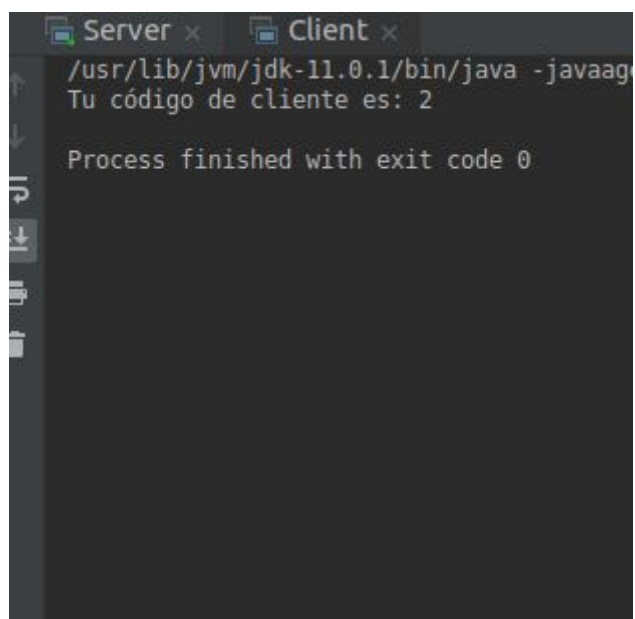
Después fijamos las conexiones en los mismos puertos a través de los sockets de servidor y cliente en cada clase.

Y por último llevamos a cabo el método de ejecución que se lleva a cabo en el main.



The screenshot shows the 'Run' console of an IDE with two tabs: 'Server' and 'Client'. The 'Server' tab is active. The console output shows the command `/usr/lib/jvm/jdk-11.0.1/bin/java -javaagent:/snap/intellij-idea` and the message 'El servidor se encuentra escuchando en el puerto 5000.' The left sidebar contains standard IDE icons for running, debugging, and testing.

```
Run: Server x Client x
/usr/lib/jvm/jdk-11.0.1/bin/java -javaagent:/snap/intellij-idea
El servidor se encuentra escuchando en el puerto 5000.
```



The screenshot shows the 'Run' console of an IDE with two tabs: 'Server' and 'Client'. The 'Client' tab is active. The console output shows the command `/usr/lib/jvm/jdk-11.0.1/bin/java -javaag`, the message 'Tu código de cliente es: 2', and 'Process finished with exit code 0'. The left sidebar contains standard IDE icons for running, debugging, and testing.

```
Server x Client x
/usr/lib/jvm/jdk-11.0.1/bin/java -javaag
Tu código de cliente es: 2
Process finished with exit code 0
```