DAM - Programación de servicios y procesos

Tema 4 - Programación de comunicaciones en red

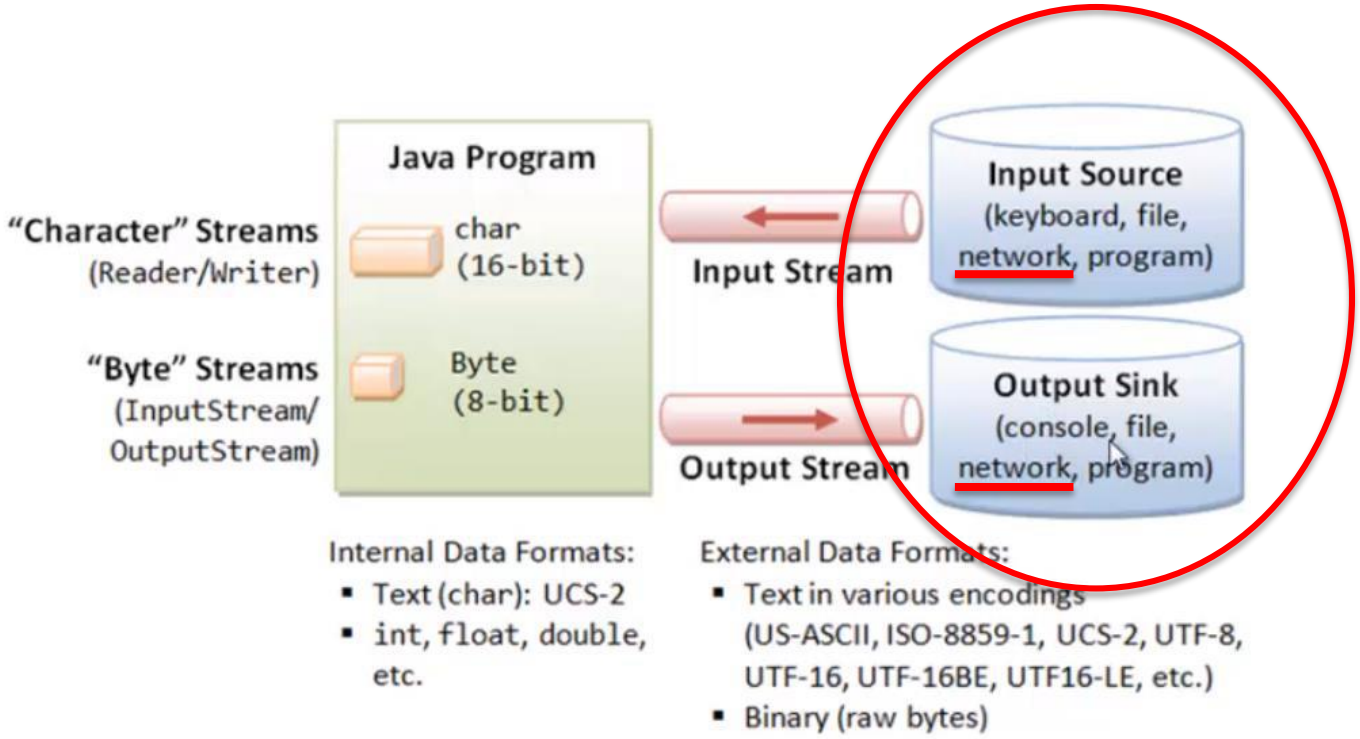
Roberto Sanz Requena

rsanz@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción



Java: lenguaje pensado y diseñado pensando en las comunicaciones en red.

1. Introducción

Roles cliente y servidor:

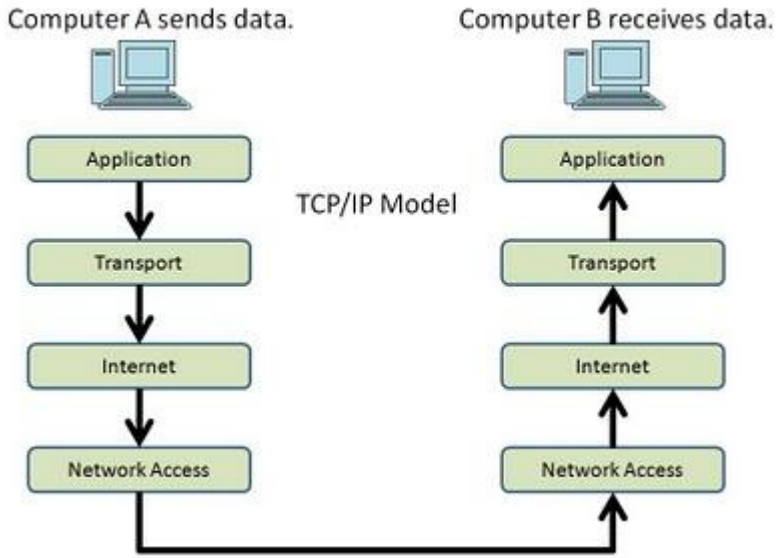
- **Servidor**: espera peticiones, recibe datos de entrada y devuelve respuestas.
- **Cliente**: genera peticiones, las envía a un servidor y espera respuestas.

Un factor fundamental en los servidores es que tienen que ser capaces de procesar varias peticiones a la vez: deben ser multihilo.

```
while (true){
     Peticion = esperarPeticion();
     hiloAsociado = new Hilo();
     hiloAsociado.atender(peticion);
}
```

1. Introducción

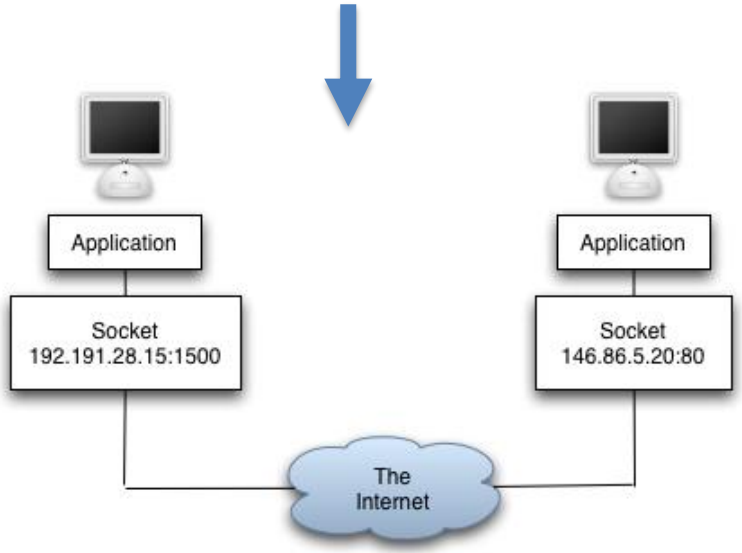
Arquitectura TCP / IP



Para programar aplicaciones, nos interesa…

- **IP**: dirección universal del cliente / servidor
- **TCP**: puerto de escucha (qué aplicación)

Establecimiento de **sockets** para comunicación



https://www.c-sharpcorner.com/UploadFile/433c33/networking-in-java/

2. Clases Java para comunicaciones en red

Paquete `java.net`

Clase `URL`: representa un puntero a un recurso de Internet.

Método	Descripción
`URL (String url)`	Constructor
`String getHost()`	Devuelve el nombre de la máquina
`InputStream openStream()`	Abre una conexión al objeto URL y devuelve un `InputStream` para la lectura de esa conexión
`URLConnection openConnection()`	Devuelve un objeto `URLConnection` que representa la conexión a un objeto remoto referenciado por la URL

- Requiere gestionar la excepción MalformedURLException.
- **Limitación**: sólo permite acceder a recursos disponibles en las URLs.

https://docs.oracle.com/javase/7/docs/api/java/net/URL.html

```java
public class GestorDescargas {
    public void descargarArchivo(
        String url_descargar,
        String nombreArchivo){
        System.out.println(" Descargando " + url_descargar);
        try {
            URL laUrl=new URL(url_descargar);
            InputStream is=laUrl.openStream();
            InputStreamReader reader=new InputStreamReader(is);
            BufferedReader bReader=new BufferedReader(reader);
            FileWriter escritorFichero=new FileWriter(nombreArchivo);
            String linea;
            while ((linea=bReader.readLine())!=null){
                escritorFichero.write(linea);
            }
            escritorFichero.close();
            bReader.close();
            reader.close();
            is.close();
        } catch (MalformedURLException e) {
            System.out.println("URL mal escrita!");
        } catch (IOException e) {
            System.out.println("Fallo en la lectura del fichero");
        }
    }
```

```java
public static void main (String[] args){
    GestorDescargas gd=new GestorDescargas();
    String url =
        "http://localhost:80"+
        "/web/recurso.txt";
    String fichero = "recurso.txt";
    gd.descargarArchivo(url,fichero);

}
}
```

3. Sockets

Clase `Socket`: cliente

- Permite contactar con un programa o servidor remoto.
- Proporciona flujos de entrada y/o salida para la comunicación.
- `Socket(InetAddress direccion, int port)`

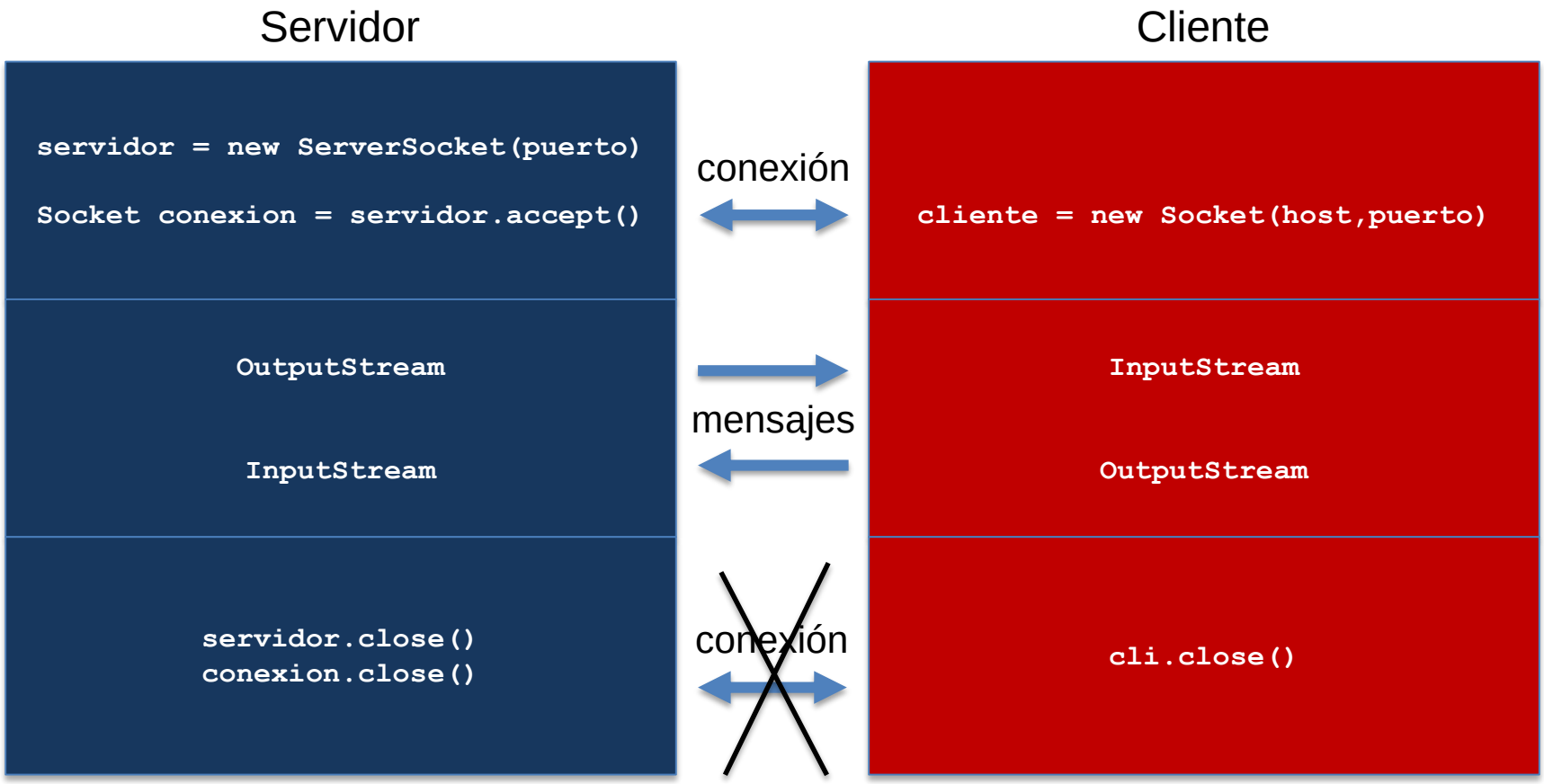
Clase `ServerSocket`: servidor

- Permite crear programas que acepten conexiones o peticiones.
- `ServerSocket(int port)`

Orientados a conexión (TCP): `Socket` y `ServerSocket`.
No orientados a conexión (UDP): `DatagramSocket` y `DatagramPacket`.

https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html

4. Enlazado y establecimiento de conexiones



5. Transmisión y recepción de información

Ejemplo de cliente

```
public class Conector {
     public static void main(String[] args) {
          String destino = "www.google.com";
          int puertoDestino = 80;
          Socket socket = new Socket();
          InetSocketAddress direccion = new InetSocketAddress(destino,puertoDestino);
          try {
               socket.connect(direccion);
               InputStream is = socket.getInputStream();
               OutputStream os = socket.getOutputStream();
               System.out.println("Conexión realizada");
          } catch (IOException e) {
               System.out.println("Error en la conexión");
               e.printStackTrace();
          }
     }
}
```

¿IP o DNS?

5. Transmisión y recepción de información

Envío/recepción de objetos

Los streams soportan el envío/recepción de bytes, tipos primitivos, caracteres y objetos.

```
ObjectOutputStream outObjeto = new
    ObjectOutputStream(socket.getOutputStream());

ObjectInputStream inObjeto = new
    ObjectInputStream(socket.getInputStream());
```

Las clases de los objetos que se desee enviar/recibir a través de la conexión deben implementar la interfaz `Serializable`.

5. Transmisión y recepción de información

Envío/recepción de objetos

```
import java.io.Serializable;
@SuppressWarnings ("serial")
public class Persona implements Serializable {
	String nombre;
	int edad;
	public Persona(String nombre, int edad) {
		this.nombre = nombre;
		this.edad = edad;
	}
	public Persona() {super();}
	public String getNombre() {return nombre;}
	public void setNombre(String nombre) {this.nombre = nombre;}
	public int getEdadO {return edad;}
	public void setEdad(int edad) {this.edad = edad;}
}
```

Servidor: crea objeto persona “anónimo” y lo envía a un cliente.
Cliente: modifica el objeto persona y lo devuelve al servidor.

5. Transmisión y recepción de información

Envío/recepción de objetos

```
public class ServidorObjeto {

	public static void main(String[] arg) throws IOException, ClassNotFoundException {
		int numeroPuerto = 5000;
		ServerSocket servidor = new ServerSocket(numeroPuerto);
		System.err.println("SERVIDOR >> Escuchando...");
		Socket cliente = servidor.accept();
		ObjectOutputStream outObjeto = new ObjectOutputStream(cliente.getOutputStream());
		Persona p = new Persona("Nombre",0);
		outObjeto.writeObject(p);
		System.err.println("SERVIDOR >> Envio a cliente: " + p.getNombre() + " - " + p.getEdad());
		ObjectInputStream inObjeto = new ObjectInputStream(cliente.getInputStream());
		Persona pMod = (Persona) inObjeto.readObject();
		System.err.println("SERVIDOR >> Recibo de cliente: " + pMod.getNombre()+ " - " +
		pMod.getEdad());
		outObjeto.close();
		inObjeto.close ();
		cliente.close ();
		servidor.close();
	}
}
```

5. Transmisión y recepción de información

Envío/recepción de objetos

```
public class ClienteObjeto {
	public static void main(String[] arg) throws UnknownHostException, IOException,
	ClassNotFoundException {
		String host = "localhost";
		int puerto = 5000;
		System.out.println("CLIENTE >> Arranca cliente");
		Socket cliente = new Socket(host,puerto);
		ObjectInputStream inObjeto = new ObjectInputStream(cliente.getInputStream());
		Persona p = (Persona) inObjeto.readObject();
		System.out.println("CLIENTE >> Recibo del servidor: "+p.getNombre() + " - " +
		p.getEdad());
		p.setNombre("Jose Garcia");
		p.setEdad(30);
		ObjectOutputStream pMod = new ObjectOutputStream(cliente.getOutputStream());
		pMod.writeObject(p);
		System.out.println("CLIENTE >> Envio al servidor: "+p.getNombre() + " - " +
		p.getEdad());
		inObjeto.close();
		pMod.close();
		cliente.close();
	}
}
```

6. Aplicaciones cliente y servidor

Ejemplo cliente/servidor: servidor de cálculo

Crear un servidor de operaciones con estas especificaciones:

- Cualquier parámetro que envíe el usuario debe ir terminado con el carácter fin de línea tipo “\n”.
- El usuario enviará primero un símbolo “+”.
- Después se puede enviar un positivo de 1 a 8 cifras.
- Después se envía un segundo positivo de 1 a 8 cifras igual que el anterior.
- El usuario podría equivocarse y enviar algo que no fuera un número. En ese caso se asume que el valor introducido es 0.
- Cuando el servidor haya recogido todos los parámetros contestará al cliente con un positivo de 1 a 16 cifras.

6. Aplicaciones cliente y servidor

```
public class ServidorCalculo {

	public static int extraerNumero(String linea) {
		int numero;
		try {
		numero = Integer.parseInt(linea);
		} catch (NumberFormatException e) {
		numero = 0;
		}
		if (numero >= 100000000) { numero = 0; }
		return numero;
	}

	public static int calcular(String op,String
	n1,String n2) {
		int resultado = 0;
		char simbolo = op.charAt(0);
		int num1 = extraerNumero(n1);
		int num2 = extraerNumero(n2);
		if (simbolo == '+') {
			resultado = num1 + num2;
		}
		return resultado;
	}
```

```
public static void main(String[] args) throws IOException {
	System.err.println("SERVIDOR >>> Arranca el servidor, espera
	peticion");
	ServerSocket socketEscucha = null;
	try {
		socketEscucha = new ServerSocket(9876);
	} catch (IOException e) {
		System.err.println("SERVIDOR >>> Error");
		return;
	}
	while (true) {
		Socket conexion = socketEscucha.accept();
		System.err.println("SERVIDOR >>> Conexion recibida!");
		InputStream is = conexion.getInputStream();
		InputStreamReader isr = new InputStreamReader(is);
		BufferedReader bf = new BufferedReader(isr);
		System.err.println("SERVIDOR >>> Lee datos para la
		operacion");
		String linea = bf.readLine();
		String num1 = bf.readLine();
		String num2 = bf.readLine();
		System.err.println("SERVIDOR >>> Realiza la
		operacion");
		Integer result = calcular(linea, num1, num2);
		System.err.println("SERVIDOR >>> Devuelve resultado");
		OutputStream os = conexion.getOutputStream();
		PrintWriter pw = new PrintWriter(os);
		pw.write(result.toString() + "\n");
		pw.flush();
		System.err.println("SERVIDOR >>> Espera nueva
		peticion");
		}
	}

}
```

6. Aplicaciones cliente y servidor

```
public class ClienteCalculo {

	public static void main(String[] args) throws IOException {
		System.out.println("CLIENTE >>> Arranca cliente");
		System.out.println("CLIENTE >>> Conexion al servidor");
		InetSocketAddress direccion = new InetSocketAddress("localhost", 9876);
		Socket socket = new Socket();
		socket.connect(direccion);

		System.out.println("CLIENTE >>> Preparado canal para recibir resultado");
		InputStream is = socket.getInputStream();
		InputStreamReader isr = new InputStreamReader(is);
		BufferedReader bfr = new BufferedReader(isr);

		System.out.println("CLIENTE >>> Envio de datos para el calculo");
		PrintWriter pw = new PrintWriter(socket.getOutputStream());
		pw.print(“+\n");
		pw.print("100\n");
		pw.print("50\n");
		pw.flush();

		String resultado = bfr.readLine();
		System.out.println("CLIENTE >>> Recibe resultado: " + resultado);
	}

}
```

7. Hilos y programación de aplicaciones en red

Cada conexión al servidor será gestionada por este como un nuevo hilo.

```
public class ServidorCalculoHilos {

	public static void main(String[] args) throws IOException {
		System.err.println("SERVIDOR >>> Arranca el servidor, espera peticion");
		ServerSocket socketEscucha = null;
		try {
			socketEscucha = new ServerSocket(9876);
		} catch (IOException e) {
			System.err.println("SERVIDOR >>> Error");
			return;
		}
		while (true) {
			Socket conexion = socketEscucha.accept();
			System.err.println("SERVIDOR >>> Conexion recibida --> Lanza hilo clase
			Peticion");
			Peticion p = new Peticion(conexion);
			Thread hilo = new Thread(p);
			hilo.start();
		}
	}

}
```

+

Clase `Peticion`

```java
public class Peticion implements Runnable {
	BufferedReader bfr;
	PrintWriter pw;
	Socket socket;
	public Peticion(Socket socket){
		this.socket = socket;
	}
	public int extraerNumero(String linea) {
		int numero;
		try { numero = Integer.parseInt(linea); }
		catch (NumberFormatException e) { numero = 0; }
		if (numero >= 100000000) { numero = 0; }
		return numero;
	}
	public int calcular(String op, String n1, String n2) {
		int resultado = 0;
		char simbolo = op.charAt(0);
		int num1 = extraerNumero(n1);
		int num2 = extraerNumero(n2);
		if (simbolo == '+') { resultado = num1 + num2; }
		return resultado;
	}
	public void run() {
		try {
			InputStream is = socket.getInputStream();
			InputStreamReader isr = new InputStreamReader(is);
			bfr = new BufferedReader(isr);
			OutputStream os = socket.getOutputStream();
			pw = new PrintWriter(os);
			String linea = bfr.readLine();
			System.err.println("SERVIDOR >>> Lee datos para la operacion");
			String num1 = bfr.readLine();
			String num2 = bfr.readLine();
			System.err.println("SERVIDOR >>> Realiza la operacion");
			Integer result = calcular(linea, num1, num2);
			System.err.println("SERVIDOR >>> Devuelve resultado");
			pw.write(result.toString() + "\n");
			pw.flush();
			System.err.println("SERVIDOR >>> Espera nueva peticion");
		} catch (IOException e) {
			e.printStackTrace();
			System.err.println("SERVIDOR >>> Error.");
		}
	}
}
```

Actividad Entregable 4 - Sockets

Presentación de la Actividad Entregable 4 (AE4_T4_Sockets)