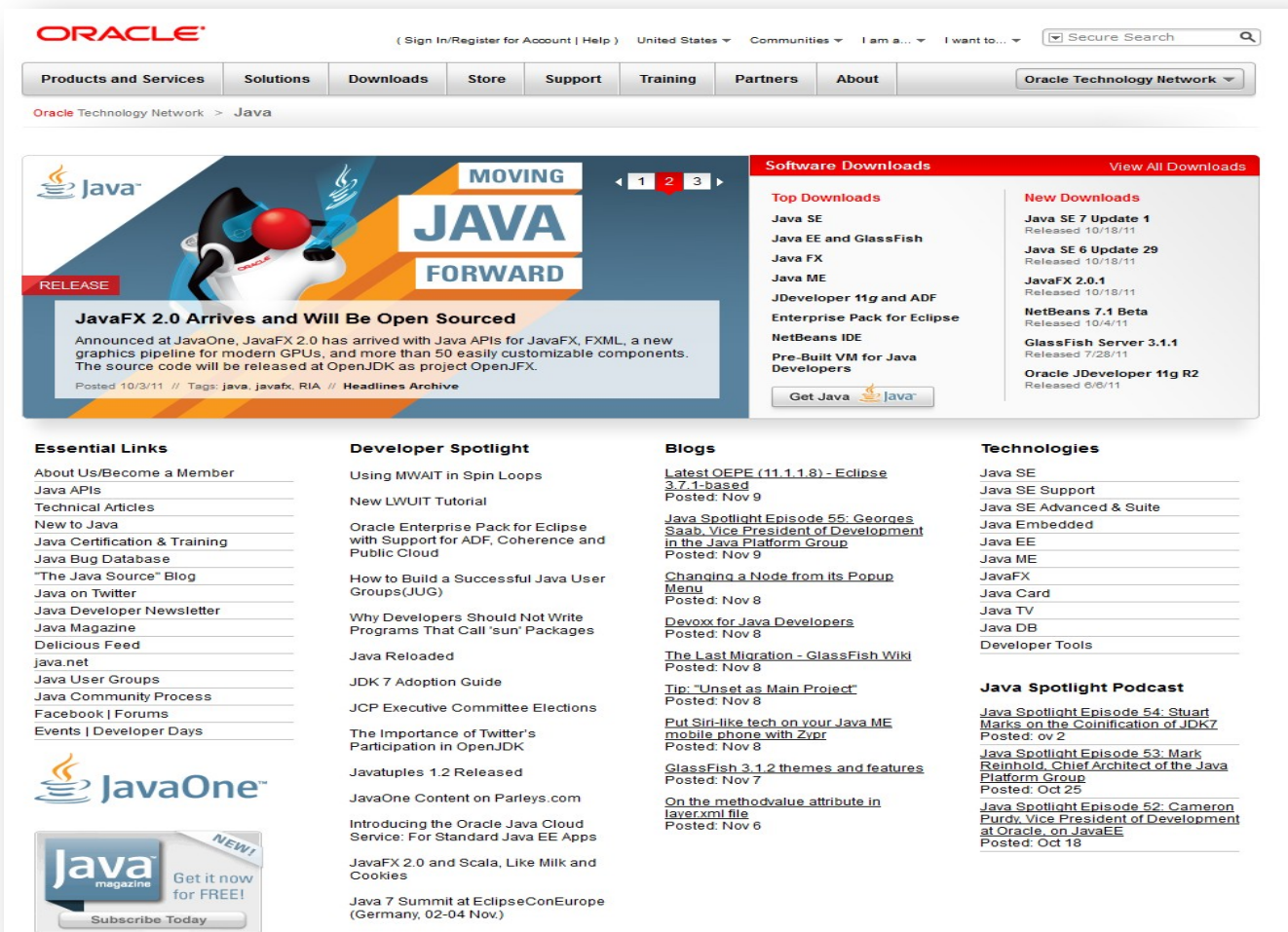


Tema 3: Java y los Threads



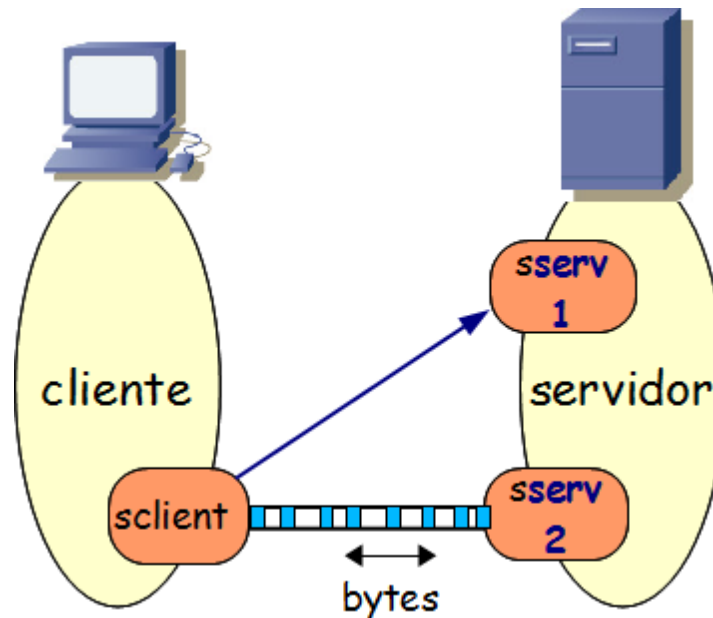
The screenshot shows the Oracle Java website. At the top, there's a navigation bar with links like "Products and Services", "Solutions", "Downloads", "Store", "Support", "Training", "Partners", and "About". Below this, a banner for "MOVING JAVA FORWARD" features the Java logo and a cartoon character. A prominent announcement states "JavaFX 2.0 Arrives and Will Be Open Sourced". To the right, a "Software Downloads" section lists various Java versions and tools. Below the banner, there are sections for "Essential Links", "Developer Spotlight", "Blogs", and "Technologies". At the bottom left, there's a "JavaOne" logo and a "Java magazine" subscription offer.

Bibliografía:

- ❑ [Kurose 10] Apartados 2.1, 2.7, 2.8
- ❑ <https://download.oracle.com/javase/8/docs/api/index.html>



- Cliente:
 - Cuando crea un socket (**sclient**) establece la conexión con el servidor
- Servidor:
 - Debe haber creado un socket (**sserv1**) donde espera a los clientes que conectan con él
 - Cuando un cliente se conecta con un servidor:
 - El servidor crea un nuevo socket (**sserv2**) para que el proceso servidor se comuniquen con el cliente
 - De esta forma es posible que un servidor se comuniquen con varios clientes simultáneamente



- Clase **ServerSocket**
- Constructores
 - **ServerSocket(int puerto) throws IOException**
 - Abre un socket en el puerto indicado en modo de escucha
 - Si `port=0`, entonces se elige cualquier puerto libre
 - **ServerSocket(int puerto, int backlog) throws IOException**
 - Abre un socket en el **puerto** indicado en modo de escucha
 - **backlog** indica la longitud máxima de la cola de conexiones en espera
 - Cuando llega una solicitud de conexión y la cola está llena, se rechaza la conexión



- Clase **ServerSocket**
- Algunos métodos importantes
 - **Socket accept() throws IOException**
 - Acepta una conexión de un cliente y devuelve un socket asociado a ella
 - El proceso se bloquea hasta que se realiza una conexión
 - El diálogo con el cliente se hace por el nuevo socket
 - El ServerSocket puede atender nuevas conexiones
 - **close() throws IOException**
 - Cierra el socket servidor
 - ✓ No se utiliza frecuentemente



- Normalmente, un servidor debe estar preparado para atender muchos clientes
- Se puede hacer de dos maneras:
 - **Secuencial**: un cliente detrás de otro
 - Hasta que no acaba con un cliente no vuelve a ejecutar otro accept
 - **Concurrente**: varios clientes al mismo tiempo



- En Java, la concurrencia la conseguimos usando hilos de ejecución
- Clase **Thread**
 - Se define una clase derivada de Thread
 - Código a ejecutar en cada hilo dentro del método **run()**
 - Se lanza el hilo con **start()**

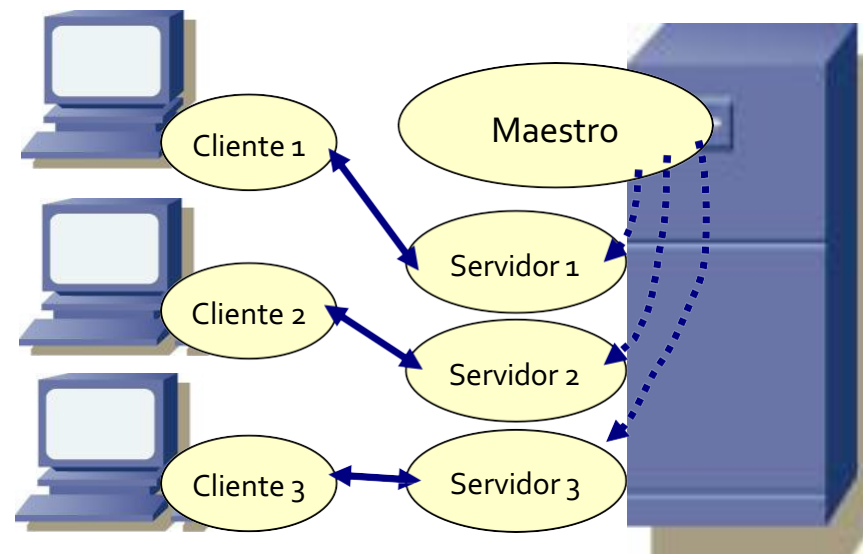
```
class Hilos extends Thread {  
    int id;  
    public Hilos(int i)  
    {  
        id=i;  
    }  
  
    public void run() {  
        for(int i=0;i<100;i++) {  
            System.out.print(id);  
            try {sleep(100);}   
            catch (InterruptedException e) {}  
        }  
    }  
  
    public static void main(String args[])  
    {  
        for(int i=0;i<3;i++) {  
            Hilos h = new Hilos(i);  
            h.start();  
        }  
    }  
}
```

Este programa no es un servidor ... pero nos permite:

- Ver cómo se crea y lanza un hilo
- Ver cómo se le pasa un parámetro al nuevo hilo

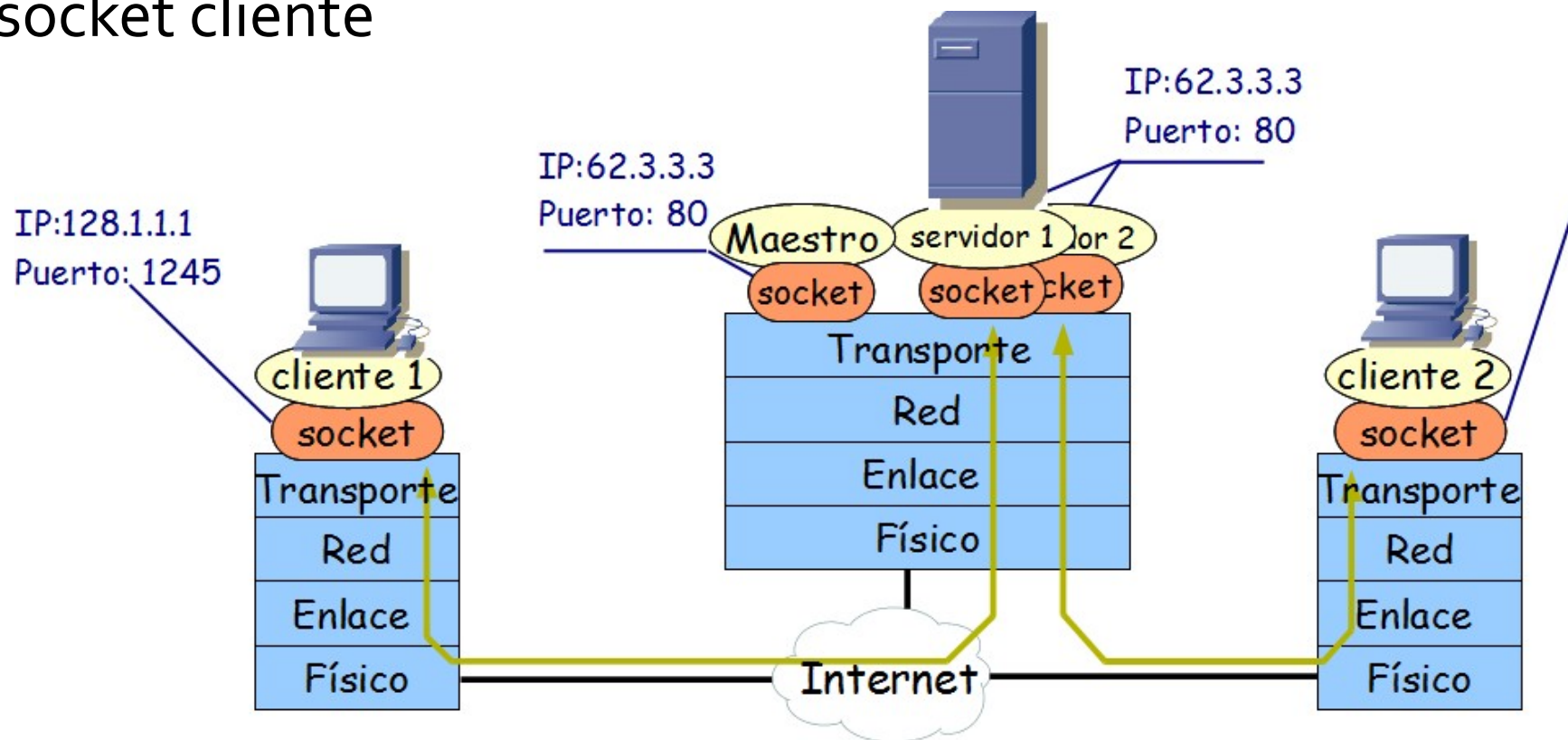


- Diversos hilos de ejecución:
 - En el hilo principal se ejecuta permanentemente el método **accept()**
 - Espera el establecimiento de nuevas conexiones
 - Para **cada cliente** que se conecta, se lanza un **nuevo hilo** de ejecución para gestionar esa conexión



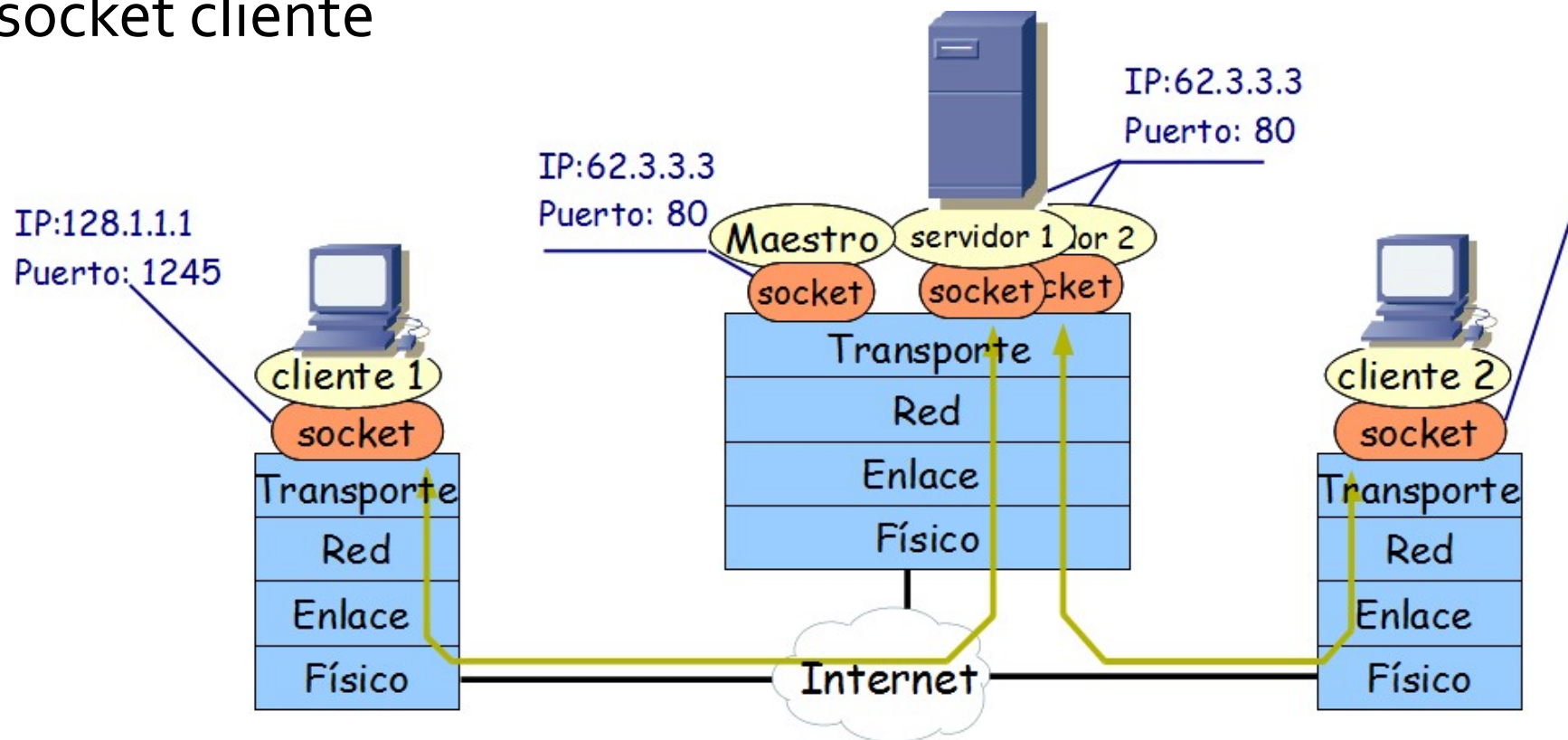
Identificación de los sockets

- Ahora tenemos varios sockets asociados al mismo puerto del servidor
 - Todos tienen la misma dirección de socket local
- Para distinguir cada conexión hay que tener en cuenta la dirección (dir. IP + puerto) del socket servidor y del socket cliente



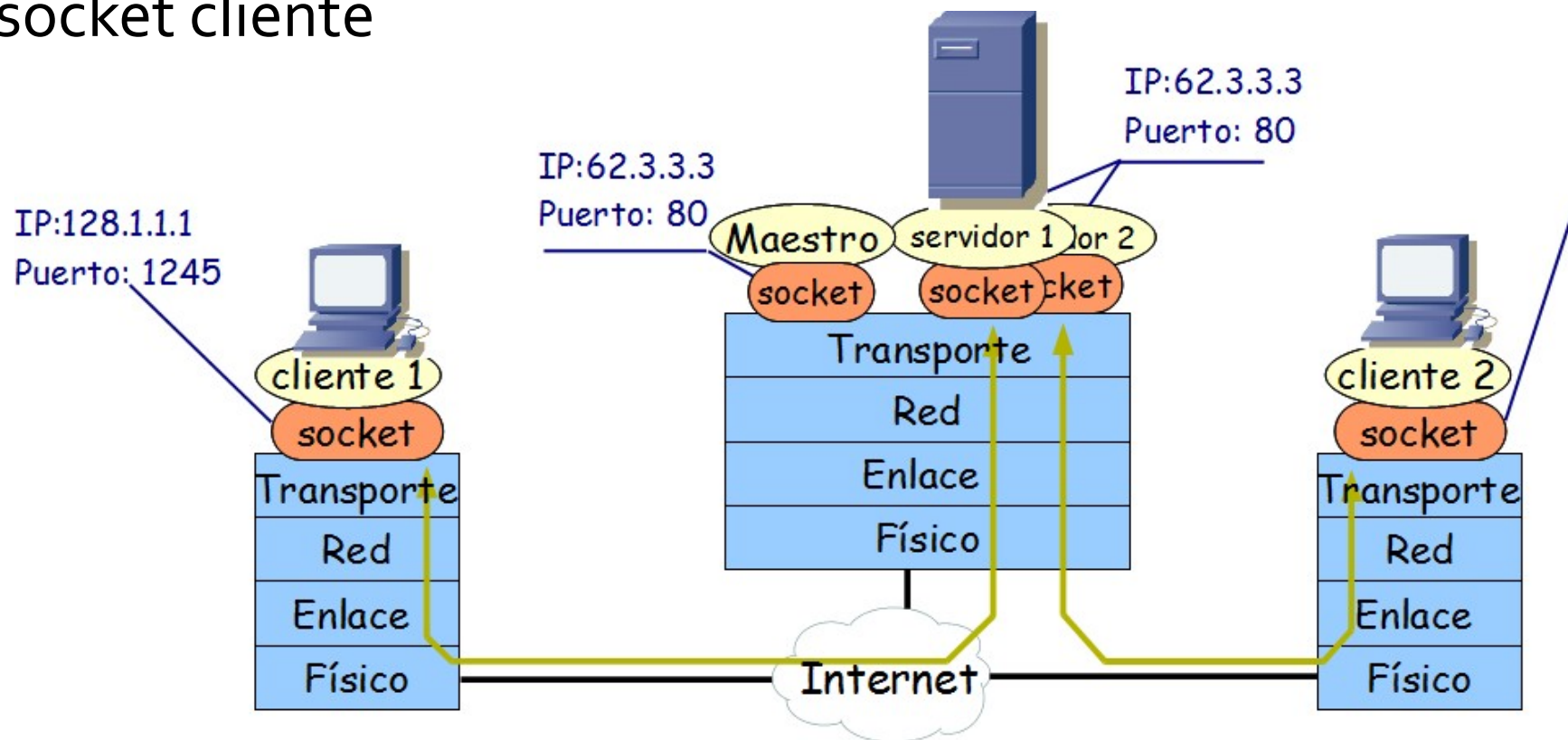
Identificación de los sockets

- Ahora tenemos varios sockets asociados al mismo puerto del servidor
 - Todos tienen la misma dirección de socket local
- Para distinguir cada conexión hay que tener en cuenta la dirección (dir. IP + puerto) del socket servidor y del socket cliente



Identificación de los sockets

- Ahora tenemos varios sockets asociados al mismo puerto del servidor
 - Todos tienen la misma dirección de socket local
- Para distinguir cada conexión hay que tener en cuenta la dirección (dir. IP + puerto) del socket servidor y del socket cliente



Implementación de servidores concurrentes TCP en java

- Definimos una clase derivada de la clase **Thread**
- En el hilo principal (método main)
 - Ejecutamos el **accept()** y conectamos con el cliente (**Socket s**)
 - Lanzamos el nuevo hilo, pasándole como parámetro el **Socket s** conectado con el cliente
- En el método **run()**
 - Incluimos el diálogo cliente-servidor
 - Cerramos el **Socket s**
- Hay que definir un constructor de la clase que acepte como parámetro un objeto **Socket**



Servidor concurrente TCP

```
import java.net.*;
import java.io.*;

class SCTCP extends Thread {
    Socket id;
    public SCTCP(Socket s) {id=s;}
    public void run() {
        try {
            PrintWriter salida=new PrintWriter(id.getOutputStream(),true);
            while(true){
                salida.println(System.currentTimeMillis());
                sleep(100);
            }
        } catch (Exception e) {System.out.println("Error en run(): " + e);}
    }

    public static void main(String args[]) throws IOException{
        ServerSocket ss=new ServerSocket(8888);
        while(true) {
            Socket s = ss.accept();
            SCTCP t = new SCTCP(s);
            t.start();
        }
    }
}
```

En el método run, ¡hay que capturar las excepciones!



- Además de los servidores concurrentes, otras aplicaciones pueden requerir hilos de ejecución
- En general, siempre que se requieran dos acciones en paralelo:
 - Clientes HTTP concurrentes
 - Aplicaciones de chat:
 - Un hilo maneja la entrada del teclado
 - El otro lee los nuevos mensajes que llegan de la red
 - Servidores multiprotocolo o multiservicio

