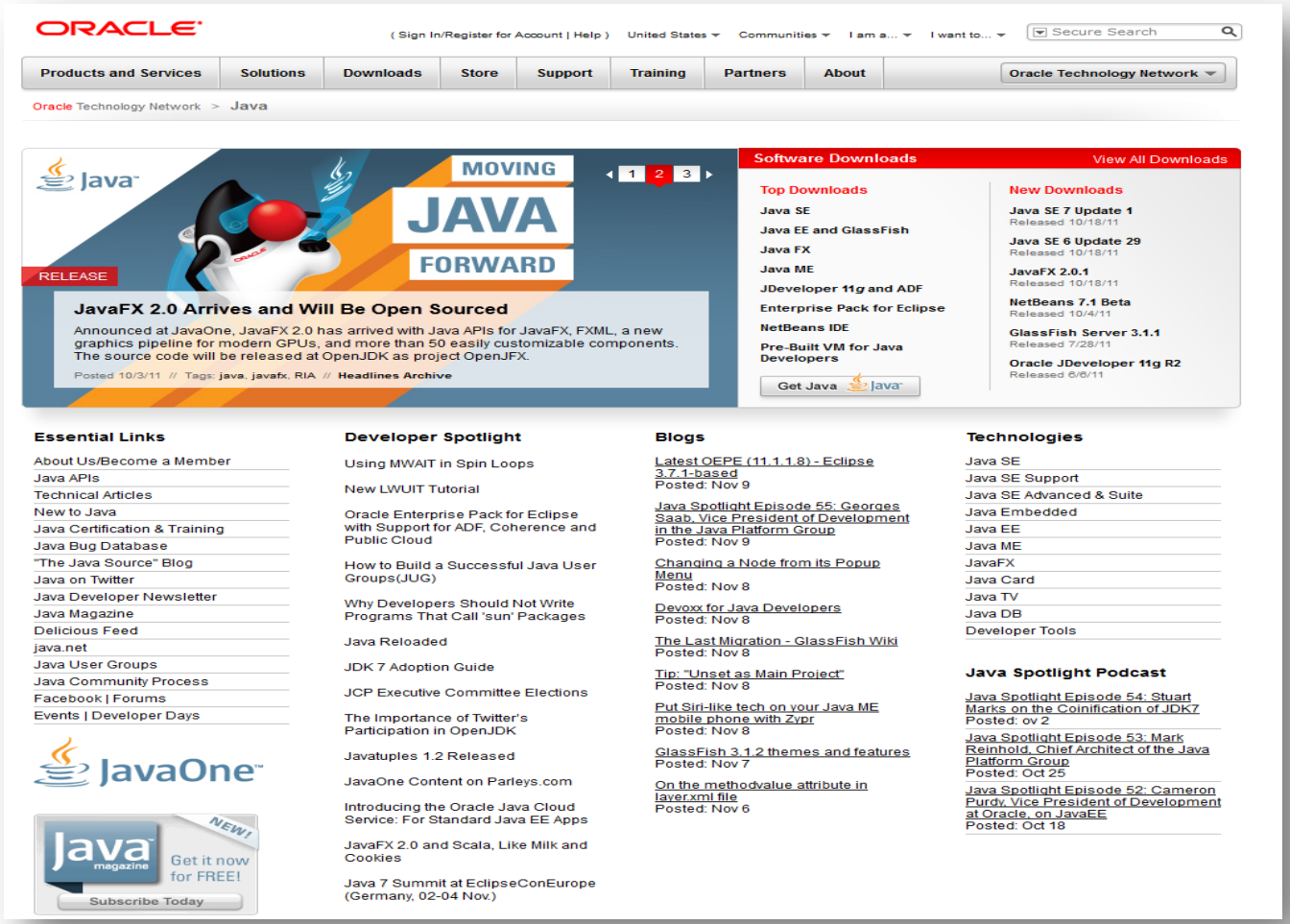
Tema 3: Java y los Threads

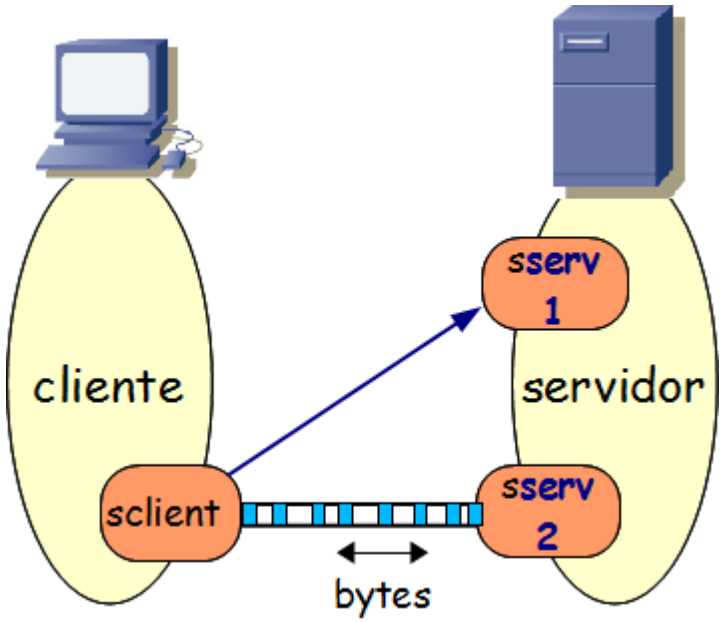


Bibliografía:

- [Kurose10] Apartados 2.1, 2.7, 2.8
- http://www.oracle.com/technetwork/java/index.html
- http://java.com/es/about/
- http://download.oracle.com/javase/6/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 comunique con el cliente
- De esta forma es posible que un servidor se comunique 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

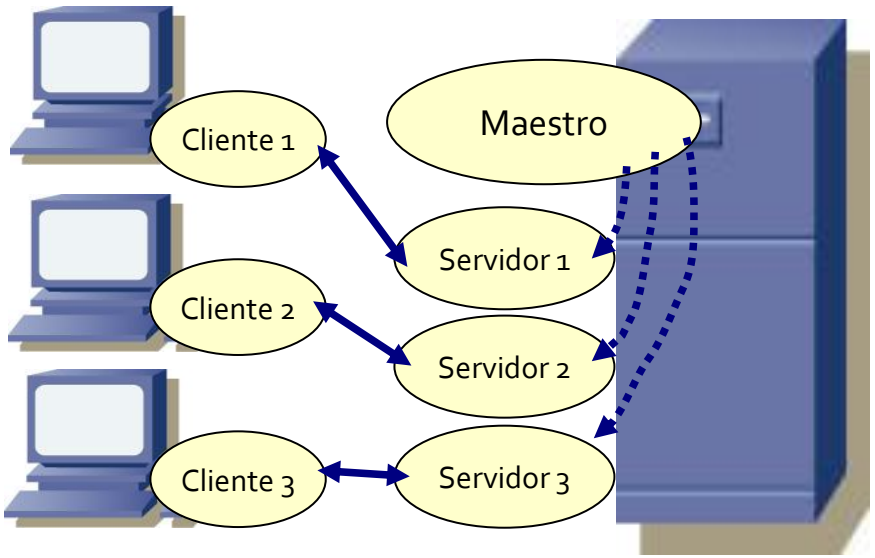
Tipos de servidores

- Normalmente, un servidor debe estar preparado para atender muchos clientes
- Se puede hacer de dos maneras:
  - Secuencial: un cliente detrás de otro
  - 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()

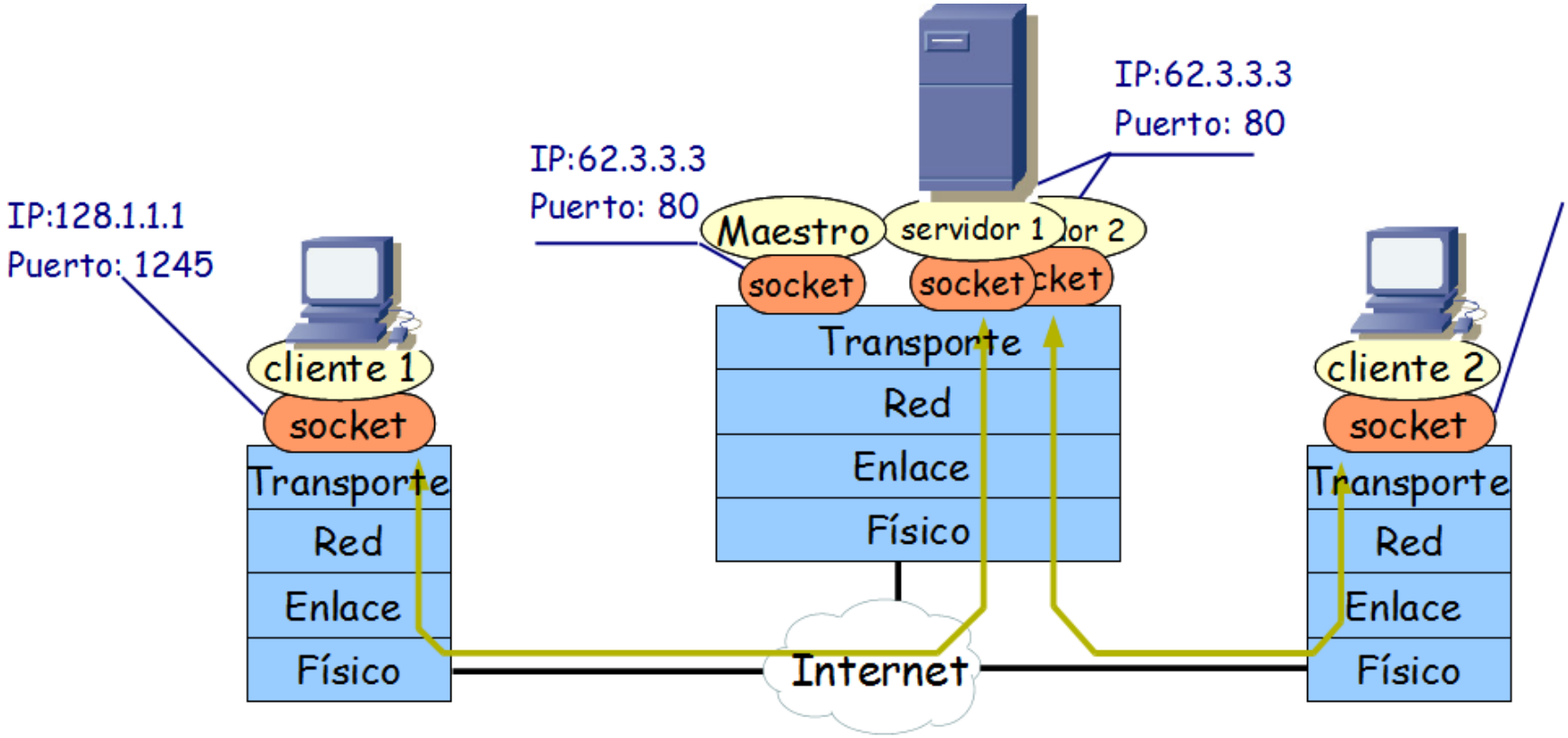
```
class Hilos extends Thread {
 int id;
 public Hilos(int i)
 {
    id=i;
 }

 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();
  }
 }
}
```

- 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
- Para identificar al socket destino hay que tener en cuenta la dirección (dir. IP + puerto) del socket origen



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) {}
 }

 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!