# Sockets para servidor

**Daniel Díaz Sánchez**
**Florina Almenárez**
**Andrés Marín**
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
dds@it.uc3m.es

# Contexto

- **Objetivos**
  - Conocer las particularidades de las comunicaciones en Java, desarrollando pequeños programas que interactúen con protocolos conocidos
  - Conocer la perspectiva de servidor del paquete java de sockets

# Índice

- Introducción a los sockets de servidor
- Conceptos básicos
- Sockets servidor
  - clase `ServerSocket`

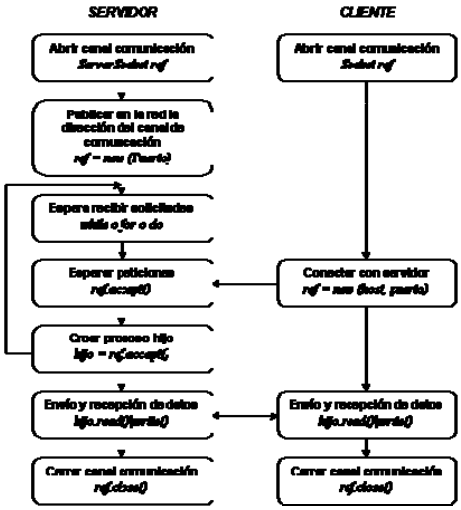


## Introducción a los sockets de servidor

- Hasta ahora hemos visto los sockets de cliente pero un cliente
  - no es nada si no puede conectarse al servidor
  - conoce el host/puerto al que quiere conectar
  - sabe cuando realiza la conexión
- En cambio, un servidor
  - No conoce de antemano quien va a conectarse
  - Ni cuando
- Un servidor es como un telefonista, siempre está esperando a que lleguen llamadas entrantes



# Introducción a los sockets de servidor






# Introducción a los sockets de servidor

- En esta sesión veremos como hacer que un servidor espere una conexión entrante
- Cómo manejar dichas conexiones de forma concurrente
  - Muchos clientes pueden conectarse simultáneamente al servidor
- Veremos algunos ejemplos

# Conceptos básicos: threads

- Hasta ahora hemos trabajado con clientes, programas secuenciales con un solo hilo
- Los servidores soportan múltiples conexiones al mismo tiempo. Por tanto, debemos usar threads o hilos
- Para hacer un programa multihilo podemos hacer dos cosas:
  - **Extender la clase java.lang.Thread**
  - **Implementar el interfaz Runnable**
- En ambos casos, la funcionalidad del thread se implementa en el método run.
- Veamos un ejemplo



# Conceptos básicos: threads

Ejemplo de threads con java.lang.Thread

```
public class EjemploConcurrencia1 extends Thread{

     /* en los atributos, tendremos todo aquello que necesitamos para que
      * nuestro hilo haga lo que deseamos. En este caso, usamos un string y un entero
      */
     String datoA;
     int datoB;
     public EjemploConcurrencia1(String nombreThread, String datoA, int datoB)
     {
             super(nombreThread); //El nombre del Thread
             this.datoA =datoA;
             this.datoB = datoB;
    }

     /* Todo thread debe tener un método run.
      * Este método se llama cuando comienza el thread*/

    public void run() {
        for (int i = datoB; i < 1000 ; i++)
            System.out.println("El thread de nombre " + getName() + " con el mensaje " + datoA + " está contando
     " + i);
        // cuando termina, escribimos un mensaje
        System.out.println("El thread con nombre " + getName() + " ha terminado ***************");
    }
    public static void main (String [] args) {
        /* Creamos varios hilos para ver el funcionamiento */
     new EjemploConcurrencia1("uno", "mensaje1", 10).start();
     new EjemploConcurrencia1("dos", "mensaje2", 20).start();
     new EjemploConcurrencia1("tres", "mensaje3", 30).start();
        System.out.println("Termina thread main");
    }
```



# Conceptos básicos: threads

## Ejemplo de threads con java.lang.Thread

```
dds@saml:~/workspace/ComputacionEnLaRed/bin$ java -cp .
   es.uc3m.it.gast.pervasive.teaching.networkComputing.ServerSockets.ex
   amples.EjemploConcurrencia1
```



# Conceptos básicos: threads

## Ejemplo de threads con java.lang.Thread

```
//Hacemos que la clase implemente Runnable
public class EjemploConcurrencia2 implements Runnable{

     /* en los atributos, tendremos todo aquello que necesitamos para que
      * nuestro hilo haga lo que deseamos. En este caso, usamos un string y un entero
      */
     String nombreThread;
     String datoA;
     int datoB;
     public EjemploConcurrencia2(String nombreThread, String datoA, int datoB)
     {
             this.nombreThread = nombreThread;
             this.datoA =datoA;
             this.datoB = datoB;
    }

     /* Todo thread debe tener un método run.
      * Este método se llama cuando comienza el thread*/

    public void run() {
     Thread.currentThread().setName(nombreThread);
        for (int i = datoB; i < 1000 ; i++)
            System.out.println("El thread de nombre " + Thread.currentThread().getName() + " con el mensaje " + datoA + "
     está contando " + i);
        // cuando termina, escribimos un mensaje
        System.out.println("El thread con nombre " + Thread.currentThread().getName() + " ha terminado **************");
    }
    public static void main (String [] args) {
        /* Creamos varios hilos para ver el funcionamiento */
     new Thread( new EjemploConcurrencia2("uno", "mensaje1", 10)).start();
     new Thread( new EjemploConcurrencia2("dos", "mensaje2", 20)).start();
     new Thread( new EjemploConcurrencia2("tres", "mensaje3", 30)).start();
        System.out.println("Termina thread main");
    }
```



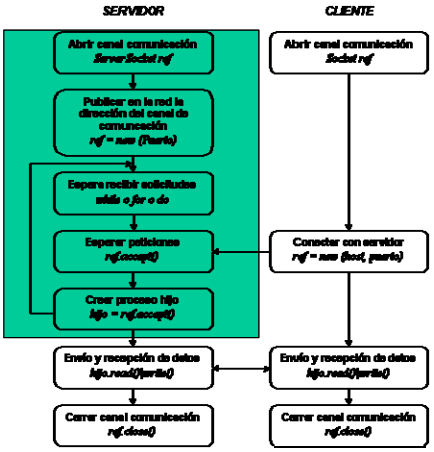
# La clase ServerSocket

- La clase `Socket` está en el paquete `java.net`
  - Proporciona soporte de TCP/IP
  - Los constructores toman como parámetros:
    - El PUERTO consisten en un entero (`int`) entre 0-65535
    - Si la máquina donde se ejecuta el programa dispone de más de un interfaz de red (ej. está conectada a la vez por cable y wifi), se puede especificar el interfaz que se debe usar para enviar/recibir datos (*multihoming*)
  - Las excepciones son de un tipo:
    - `IOException`: por errores de entrada y salida, falta de permisos, interfaz de red erroneo…
    - BindException: cuando no se pude asociar a un puerto



# La clase ServerSocket

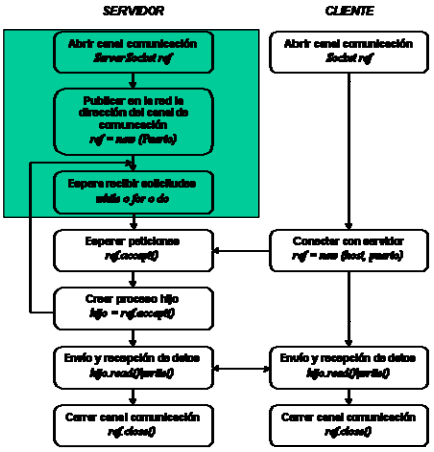
- La clase ServerSocket nos ayuda a realizar el proceso de conexión
  - Instanciar ServerSocket
  - Escuchar en el puerto (bind)
  - ServerSocket escucha hasta que llega una conexión (accept)
  - Cuando llega una conexión devuelve un objeto **Socket**






# La clase ServerSocket

- En java el constructor hace las tareas de configurar el ServerSocket y escuchar en el puerto
  - Instanciar ServerSocket
  - Escuchar en el puerto (bind)






# La clase ServerSocket : Constructores

`Java.net.ServerSocket`

Constructor

```
public ServerSocket(int port) throws BindException, IOException
```

Descripción/Parámetros

El constructor crea un socket de escucha (ServerSocket) en el puerto indicado **(creación + bind)**. Si el parámetro es 0 java seleccionará un puerto aleatorio. Si el constructor arroja una excepción de tipo BindException (IOException) puede ser debido a que el puerto está en uso o a que no tenemos suficientes permisos (puertos de 0 a 1024)

Ejemplo

```
try {
  ServerSocket servidorWeb = new ServerSocket(80);
}
catch (IOException ex) {
  System.err.println(ex);
}
```



# La clase ServerSocket : Constructores

Escanear el PC para ver los puertos de escucha usados

```
package es.uc3m.it.gast.pervasive.teaching.networkComputing.ServerSockets.examples;

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

public class EscanerDeServidoresLocales {

    public static void main(String[] args) {
          /* comprueba los puertos desde el 1 al 65535 */
          ServerSocket test = null;
          for (int puerto = 1; puerto <= 65535; puerto++) {

                    try {
                              // Si el servidor falla en escuchar en el puerto significa que
                              // ya hay un servidor en dicho puerto
                              test = new ServerSocket(puerto);
                              if (test != null)
                                        test.close();
                    } catch (IOException ex) {
                              System.out.println("El puerto " + puerto
                                                  + " está ocupado por un servidor.");
                    }
          }
    }
```



# La clase ServerSocket : Constructores

`Java.net.ServerSocket`

Constructor

**`public ServerSocket(int port, int queueLength) throws IOException, BindException`**

Descripción/Parámetros

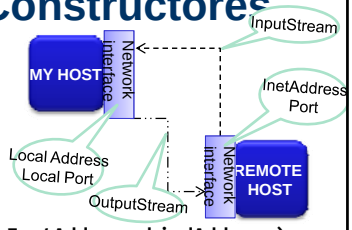
El constructor crea un socket de escucha (ServerSocket) en el puerto indicado. Si el parámetro es 0 java seleccionará un puerto aleatorio. Permite indicar el número de conexiones que se pueden guardar en la cola (sin aceptar) hasta que el servidor comienza a rechazar conexiones.

Ejemplo

```
try {
  ServerSocket glassfish = new ServerSocket(4848, 100);
}
catch (IOException ex) {
  System.err.println(ex);
}
```



# La clase ServerSocket : Constructores




Java.net.ServerSocket

Constructor

```
public ServerSocket(int port, int queueLength, InetAddress bindAddress)
   throws BindException, IOException
```

Descripción/Parámetros

Igual que el anterior pero permite seleccionar el interfaz de red (dirección local) en el que escuchar. Supongamos que tenemos varias tarjetas de red, una wifi y otra ethernet.

Ejemplo

```
try {
  ServerSocket glassfish = new ServerSocket(4848, 100);
}
catch (IOException ex) {
  System.err.println(ex);
}
```



# La clase ServerSocket : Constructores

java.net.ServerSocket

Constructor

```
public ServerSocket() throws IOException
public void bind(SocketAddress endpoint) throws IOException
public void bind(SocketAddress endpoint, int queueLength) throws
   IOException
```

Descripción/Parámetros

En este caso, simplemente crea la clase. Habría que hacer bind manualmente (usando bind) para que escuchara en el puerto.

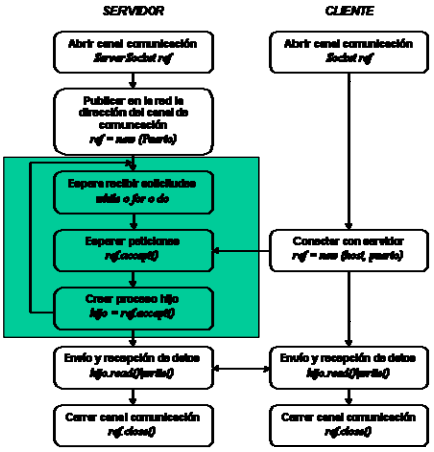
Ejemplo

```
ServerSocket ss = new ServerSocket( );
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```



# La clase ServerSocket

- El siguiente paso es esperar a una conexión y cuando llegue aceptarla
- El método accept es **bloqueante,** es decir, una vez se llama, el programa se detiene hasta que llega una conexión






# La clase ServerSocket : método accept

`java.net.ServerSocket`

Método

**`public Socket accept( ) throws IOException`**

Descripción/Parámetros

Una vez el socket está configurado y escuchando (bind) el método accept bloquea el flujo del programa hasta que llega una conexión.
El método accept devuelve un socket cuando recibe una conexión. A partir de ahí se puede llamar a getInputStream y getOutputStream para obtener los streams de comunicación con el cliente.



# La clase ServerSocket : método accept

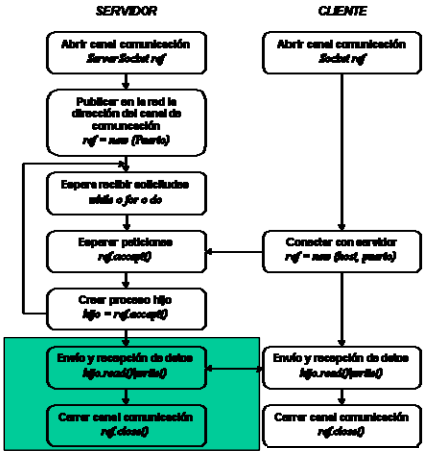
Ejemplo

```
//Creo un ServerSocket en el puerto 8080
ServerSocket server = new ServerSocket(8080);
//Llegados a este punto, si no se lanzan excepciones el ServerSocket
   está configurado y escuchando (bind)
//Bucle para escuchar conexiones entrantes
while (true) {
  //al llamar a accept se bloquea. Cuando llega una conexión devuelve
   un socket
  Socket conexion = server.accept( );
  //enviamos datos
  OutputStreamWriter out
   = new OutputStreamWriter(conexion.getOutputStream( ));
  out.write("Te has conectado al servido, hasta luego\r\n");
  //ya hemos atendido al cliente, cerramos el socket (no el
   serverSocket)
  connection.close( );
}
```



# La clase ServerSocket

- Cuando se acepta, se obtiene un Socket
- Para enviar y recibir datos se procede como en la clase socket






# La clase `ServerSocket`: métodos

- **'Getters'**: obtener información sobre el ServerSocket
- **Cierre** del ServerSocket
- **'Setters'**: Opciones de ServerSockets



# La clase `ServerSocket`: métodos 'getters'

`java.net.ServerSocket`

| Métodos |
| --- |
| **`public InetAddress getInetAddress( )`**<br>**`public int getLocalPort( )`** |
| Descripción/Parámetros |
| `getInetAddress` devuelve la dirección local (interfaz de red) en la que escucha el servidor.<br>`getLocalPort` devuelve el puerto en el que escucha el servidor (util si se ha solicitado un puerto aleatorio) |



## La clase ServerSocket: métodos 'getters'

Ejemplo

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

public class RandomPort {

  public static void main(String[] args) {

    try {
      //solicitamos un puerto aleatorio
       ServerSocket server = new ServerSocket(0);
       System.out.println("El sistema nos ha dado el puerto " +
   server.getLocalPort( ));
    }
    catch (IOException ex) {
      System.err.println(ex);
    }

  }

}
```



## La clase ServerSocket: métodos 'getters'

`java.net.ServerSocket`

Métodos avanzados

```
public int  getSoTimeout( ) throws IOException
public boolean getReuseAddress( ) throws SocketException
public int  getReceiveBufferSize( ) throws SocketException
```

Descripción/Parámetros

getSoTimeout devuelve el valor de la variable de sistema TCP **SO_TIMEOUT** que corresponde al tiempo esperado por accept antes de devolver java.io.InterruptedIOException. Por lo general es infinito.

getReuseAddress devuelve el true/false (variable de sistema TCP **SO_TIMEOUT)** indicando si la dirección/pruerto puede reutilizarse si aun existe tráfico atravesando la red (de la antigua conexión que usaba el puerto)

getReceiveBufferSize devuelve el valor de la variable de sistema TCP **SO_RCVBUF** que corresponde al tamaño del buffer de recepción asignado a cada soket devuelto por **accept**. Depende de la plataforma.



# La clase ServerSocket : método Close

java.net.ServerSocket

Método

**public void close( ) throws IOException**

Descripción/Parámetros

Cierra el socket de servidor (ServerSocket). De esta manera el servidor deja de escuchar en el puerto.
**No debe confundirse con cerrar el socket devuelto por accept.**

Ejemplo

```
ServerSocket server = new ServerSocket(80);
    server.close( );
```



# La clase ServerSocket: métodos 'setters'

java.net.ServerSocket

Métodos

**public void setSoTimeout(int timeout) throws SocketException**
**public void setReuseAddress(boolean on) throws SocketException**
**public void setReceiveBufferSize(int size) throws SocketException**

Descripción/Parámetros

setSoTimeout asigna el valor de la variable de sistema TCP **SO_TIMEOUT** que corresponde al tiempo esperado por accept antes de devolver java.io.InterruptedIOException.
setReuseAddress establece el valor (true/false) de la variable de sistema TCP **SO_TIMEOUT.** Si el valor es true, la dirección/pruerto puede reutilizarse aun en presencia de tráfico de la antigua conexión que usaba el puerto.
setReceiveBufferSize configura el tamaño del buffer de recepción del socket devuelto por accept. Corresponde a la variable de sistema TCP **SO_RCVBUF.**



# La clase ServerSocket: ejemplo

## Ejemplo: servidor de eco no concurrente

Un servidor de eco responde al cliente con los mismos datos que éste ha enviado.

Vamos a construir un servidor de eco no concurrente, es decir solo soporta un cliente simultáneo

El servidor soporta únicamente texto.

Pasos:

1.-Crear el ServerSocket
2.-Escuchar en el puerto
3.-Aceptar conexión
4.-Recibir datos
5.-Devolver los datos recibidos