Sockets

Víctor Custodio, 2014

¿Qué es un Socket?

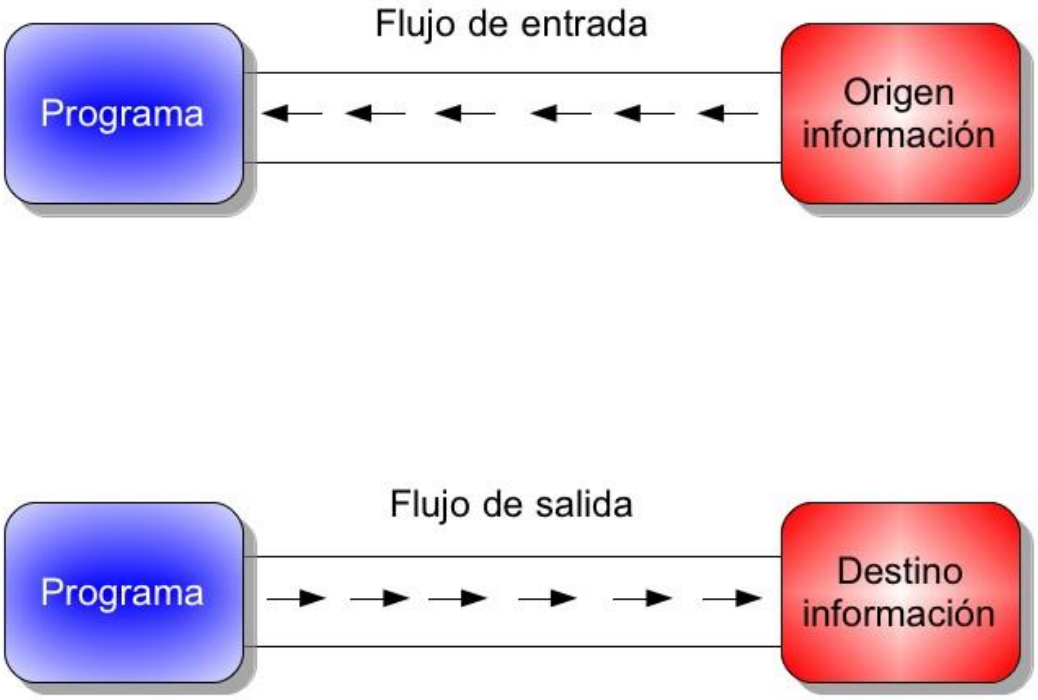
Un *socket* es una abstracción a través de la cual una aplicación pueden mandar y recibir datos.
Un *socket* permite a una aplicación conectarse a una red y comunicarse con otras aplicaciones que están conectadas a la misma red.
Existen varios tipos de *sockets*, pero los más utilizados son los *Stream Sockets* y los *Datagram Sockets,* ambos en java.net.

- Los *Stream Sockets* funcionan sobre el protocolo TCP, con el protocolo IP por debajo. Ofrecen un servicio fiable.
- Los *Datagram Sockets* funcionan sobre el protocolo UDP, con el protocolo IP por debajo.

¿Qué es un Socket?

- Un *socket IP* está identificado por una dirección única, un protocolo (TCP o UDP) y un número de puerto.
- Un socket es un punto final de un enlace de comunicación de dos vías entre dos programas que se ejecutan a través de la red.
- El cliente y el servidor deben ponerse de acuerdo sobre el protocolo que utilizarán.
- El paquete java.net de la plataforma Java proporciona las clases necesarias para trabajar con sockets

Sockets - Streams



Sockets UDP

- No orientado a conexión:
  - Envío de datagramas de tamaño fijo. No es fiable, puede haber pérdidas de información y duplicados, y la información puede llegar en distinto orden del que se envía.
  - No se guarda ningún estado del cliente en el servidor, por ello, es más tolerante a fallos del sistema.
- Tanto el cliente como el servidor utilizan la clase DatagramSocket.

Sockets UDP - DatagramSocket

- Constructores:
  - public DatagramSocket ()
  - public DatagramSocket (int port)
  - public DatagramSocket (int port, InetAddress laddr)
- port: puerto de la máquina.
- laddr: dirección IP local que se hará pública mediante el bind.
- El constructor DatagramSocket se encarga de hacer el bind.
- El primer constructor coge un puerto libre.

Sockets UDP - DatagramSocket

- public void connect(InetAddress address, int port):
  - Conecta el socket a la máquina remota con la IP address y puerto port.
- public void close()
  - Cierra el canal de comunicación.
- public void disconnect():
  - Desconecta el socket.
- public InetAddress getInetAddress():
  - Devuelve la IP de la máquina remota.
- public int getPort():
  - Devuelve el puerto de la máquina remota.

Tambien hay dos llamadas para saber la IP y puerto local (getLocalAddress() y getLocalPort()).

Sockets UDP - DatagramSocket

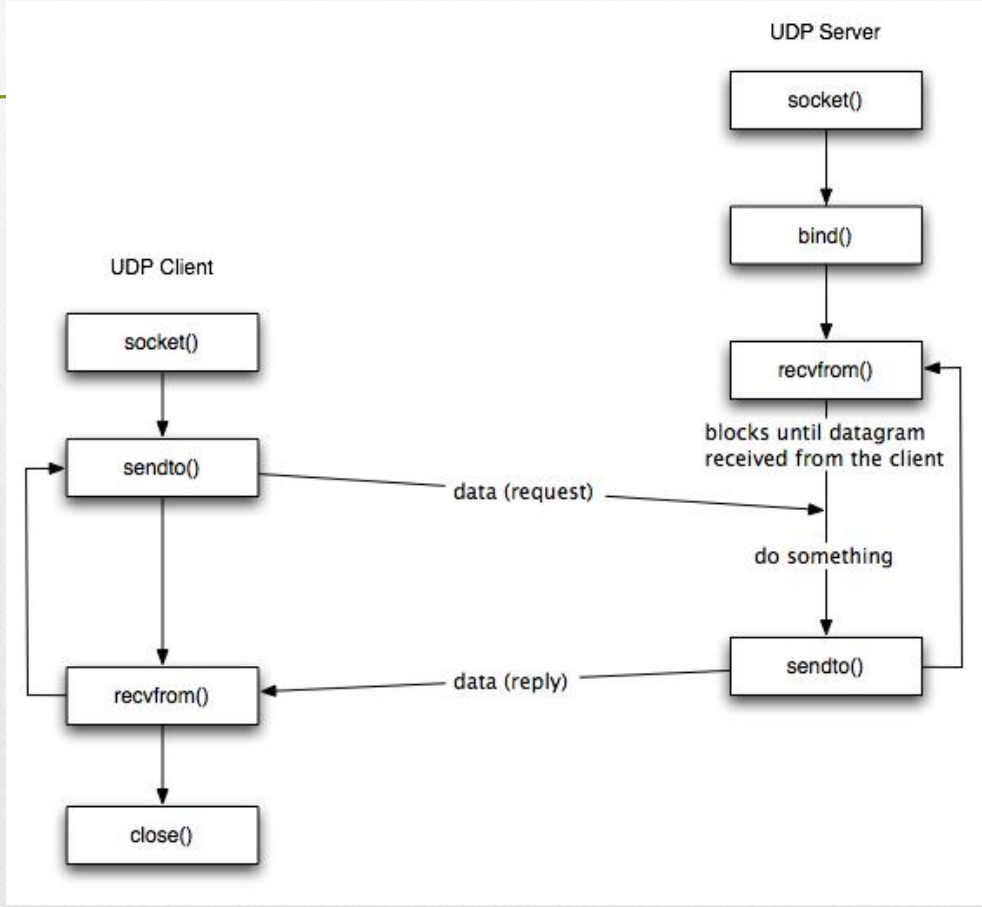
- public void send(DatagramPacket p):
  - Envía un datagrama a la máquina remota, por el socket asociado.
- public void receive(DatagramPacket p):
  - Recibe un datagrama de otra máquina, por el socket asociado.

Sockets UDP - DatagramPacket

Constructores:

- public DatagramPacket (byte[] buff, int length)
  - Construye un DatagramPacket para recibir paquetes en el buffer buff, de longitud length
- public DatagramPacket (byte[] buff, int length, InetAddress address, int port)
  - Construye un DatagramPacket para envíar paquetes con datos del buffer buff, de longitud length, a la IP address y el puerto port.
- Servicios:
  - Para la actualización y consulta de los diferentes campos de un DatagramPacket disponemos de los siguientes métodos: set/getAddress, set/getData, set/getLength, set/getPort.

Esquema conexión UDP



Ejemplo Cliente UDP

```
import java.net.*;
import java.io.*;
public class UDPClient {
  public static void main(String args[]) {
    DatagramSocket socket = null;
    try {
        socket = new DatagramSocket();
      byte[] m = args[0].getBytes();
      InetAddress host = InetAddress.getByName(args[1]);
      int port = 6789;
      DatagramPacket req = new DatagraPacket(m,
                          args[0].length(), host, port);
      socket.send(req);
      byte[] n = new byte[1000];
      DatagramPacket rep = new
                          DatagraPacket(n, n.length);
      socket.receive(rep);
      System.out.println("Received "+ new String
                           (rep.getData()));
    } catch (SocketException e) {
        System.out.println("Socket: "+e.getMessage());
   } catch (IOException e) {
        System.out.println("IO: "+e.getMessage());
   } finally { if (socket != null) socket.close(); }
  }
}
```

Ejemplo Servidor UDP

```
import java.net.*;
import java.io.*;
public class UDPServer {
  public static void main(String args[]) {
    DatagramSocket socket = null;
    try {
        socket = new DatagramSocket(6789);
       byte[] n = new byte[1000];
       while (true) {
            DatagramPacket req = new
                        DatagramPacket(n, n.length);
            socket.receive(req);
            DatagramPacket rep = new
                        DatagramPacket(req.getData(),
                        req.getLength(), req.getAddress(),
                        req.getPort());
            socket.send(rep);
       }
    } catch (SocketException e) {
        System.out.println("Socket: "+e.getMessage());
   } catch (IOException e) {
        System.out.println("IO: "+e.getMessage());
   } finally { if (socket != null) socket.close(); }
  }
}
```

Particularidades de UDP

- Tamaño del mensaje: el recibidor debe establecer el largo del mensaje a recibir, si es más pequeño que el que se mandó se trunca (se puede hasta 8 kilobytes)
- Bloqueo: la instrucción de send no bloquea hasta que el destino recibe los datos. Los datagramas son almacenados en una cola en el destino. Si no hay proceso esperándolos se descartan. Receive bloquea hasta que hay algo que leer de la cola o hasta el timeout
- Timeouts: se pueden definir sobre el socket, por default no hay *setSoTimeout*.
- Recibe de cualquiera: el receive no especifica de quién, así que el origen se saca del datagrama. Se puede abrir un DatagramSocket que sólo pueda mandar a una dirección y a un port *connect*

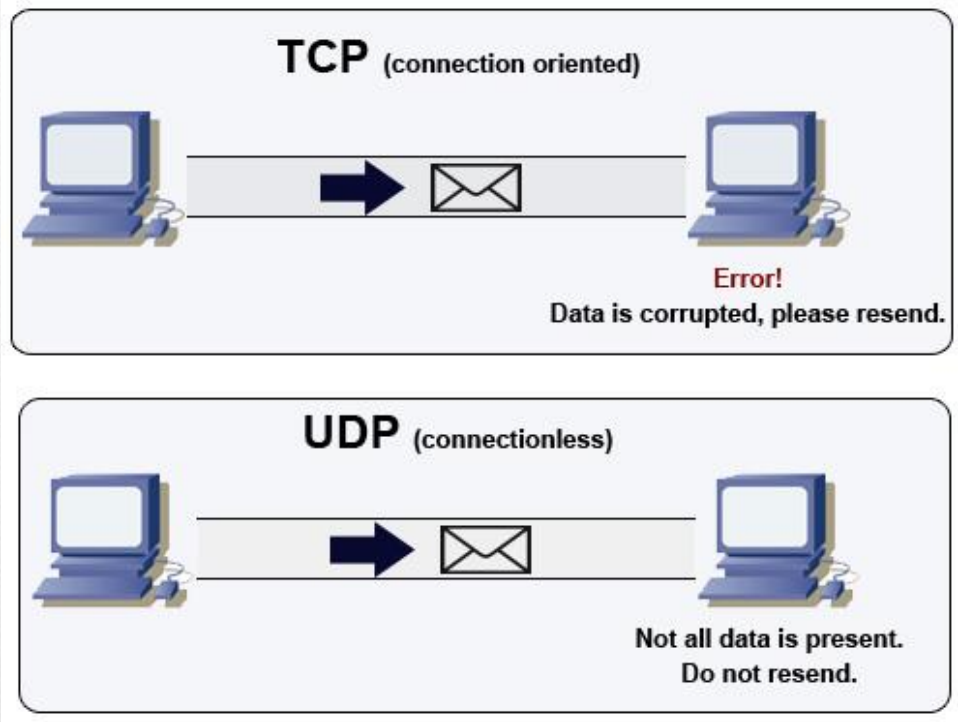
Sockets TCP

- El servicio *TCP :*
  - Es orientado a conexión y proporciona un transporte fiable.
  - Es full-dúplex. (enviar/recibir)
  - Para que una conexión se pueda establecer, uno de los sockets debe estar a la espera de recibir conexiones y el otro socket debe iniciar una conexión, y para que una conexión se cierre, basta con que uno de los sockets cierre la conexión

Sockets TCP

- Establece un camino virtual entre servidor y cliente, fiable, sin pérdidas de información ni duplicados, la información llega en el mismo orden que se envía.
- El cliente abre una sesión en el servidor y este guarda un estado del cliente.
  - El cliente utiliza la clase Socket
  - El servidor utiliza la clase ServerSocket

Sockets TCP


TCP (connection oriented)
Error!
Data is corrupted, please resend.
UDP (connectionless)
Not all data is present.
Do not resend.

Sockets TCP

- **ServerSocket** proporciona un socket dispuesto a aceptar conexiones TCP de forma pasiva. El proceso que posee un ServerSocket se le denomina Servidor.
- **Socket** proporciona un socket por el que se pueden intercambiar datos con otro socket. El proceso que posee un Socket se le denomina Cliente.

Clase Socket

- Constructores:
  - public Socket ()
  - public Socket (InetAddress address, int port)
  - public Socket (String host, int port)
  - public Socket (InetAddress address, int port, InetAddress localAddr, int localPort)
  - public Socket (String host, int port, InetAddress localAddr, int localPort)
- address / localAddr: dirección IP de la máquina remota / local.
- port / localPort: puerto de la máquina remota / local.
- host: nombre de la máquina remota

Nota: En el caso del primer constructor crea un socket sin conexión

Clase Socket

- public InetAddress getInetAddress():
  - Devuelve la dirección IP de la máquina a la que estamos conectados.
- public int getPort():
  - Devuelve el puerto de la máquina remota.
- public void close()
  - Cierra el canal de comunicación.
- public InputStream getInputStream():
  - Devuelve el canal de lectura del socket.
- public OutputStream getOutputStream():
  - Devuelve el canal de escritura del socket.

Ademas JAVA proporciona dos llamadas para saber la IP y puerto local (getLocalAddress() y getLocalPort())

Clase ServerSocket

- Constructores:
  - public ServerSocket (int port)
  - public ServerSocket (int port, int backlog)
  - public ServerSocket (int port, int backlog, InetAddress bindAddr)
- port: puerto de la máquina servidora.
- backlog: tamaño de la cola de espera, en el primero es 50.
- bindAddr: dirección IP local que se hará pública mediante el bind.
- •El constructor ServerSocket se encarga de hacer el bind y el listen.

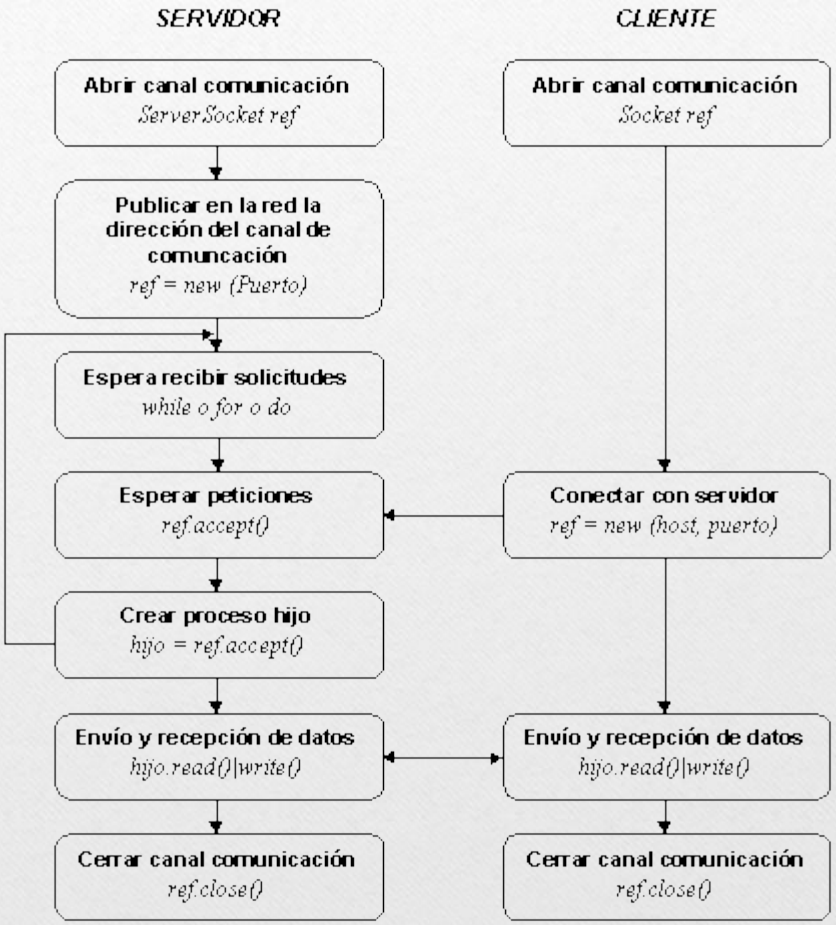
Clase ServerSocket

- public Socket accept():
  - Devuelve el socket resultado de aceptar una petición, para llevar a cabo la comunicación con el cliente.
- public void close()
  - Cierra el canal de comunicación.
- public InetAddress getInetAddress():
  - Devuelve la dirección IP de la máquina local.
- public int getLocalPort():
  - Devuelve el puerto de la máquina local

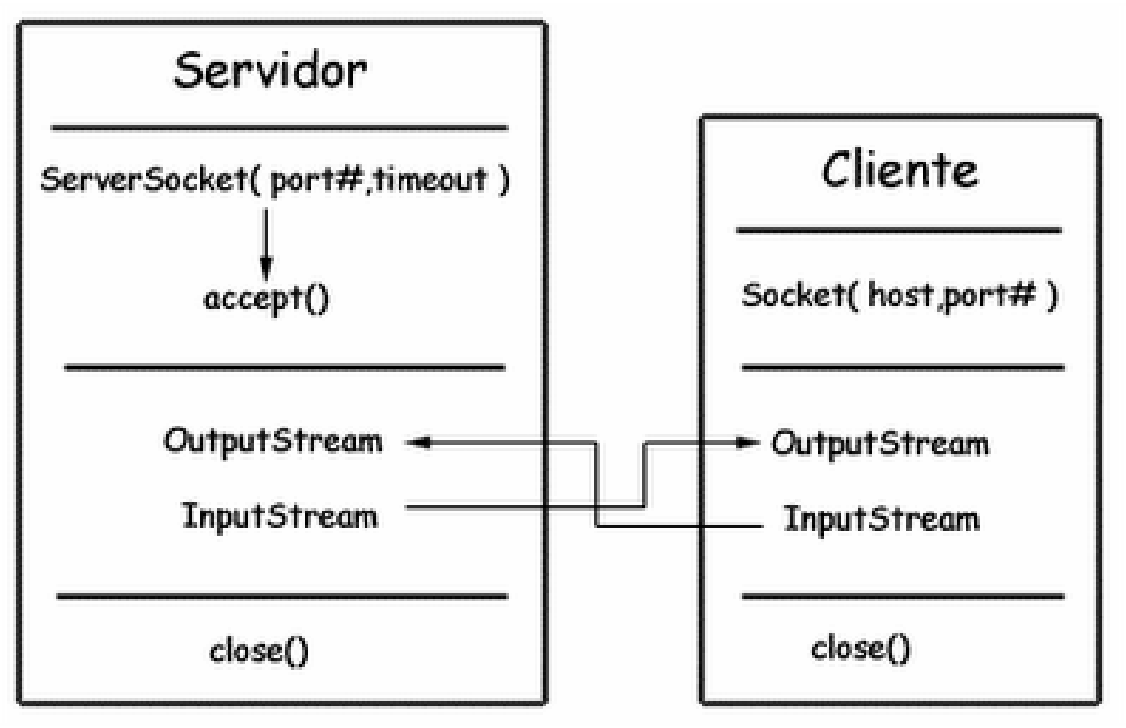
Comunicación cliente servidor

- Para la transmisión de datos entre cliente y servidor se utilizarán las clases DataInputStream (recibir datos) y DataOutputStream (enviar datos)
- Estas clases disponen de métodos para leer y escribir datos en el socket:
  - read/writeBoolean
  - read/writeChar
  - read/writeDouble, read/writeFloat, read/writeInt, read/writeLong, read/writeShort
  - read/writeUTF (leer/escribir cadenas de caracteres)
- Para envíar los datos se utiliza el método flush() de la clase DataOutputStream

Conexión de sockets TCP en java



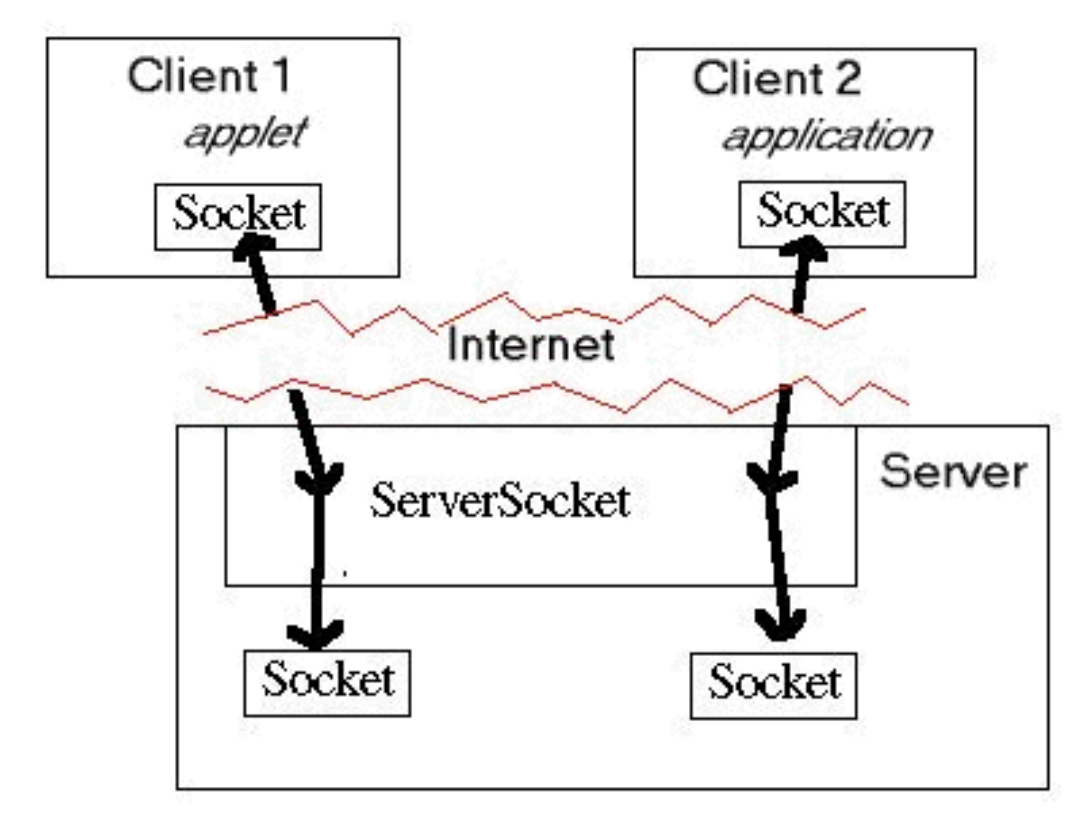
Conexión de sockets TCP en java



Conexión de sockets TCP en java

Proceso Servidor	Proceso Cliente
1. El proceso servidor crea un ServerSocket y se ata a un puerto con él.	
2. El proceso servidor se pone a escuchar en ese puerto para recibir peticiones, para ello utiliza el método accept() de la clase ServerSocket. Este método devuelve un Socket.	
	3. El proceso cliente crea un Socket y se conecta al servidor con el nombre de la máquina y el puerto.
4. El proceso servidor va a utilizar el Socket devuelto por el método accept() para intercambiar datos con el proceso cliente. Para el intercambio de datos se utilizan Streams.	
	5. El proceso cliente envía la información necesaria para que el proceso servidor pueda trabajar.
6. El proceso servidor envía al cliente la información requerida.	
NOTA el paso 5 y 6 puede repetirse varias veces.	
7. Se cierra la conexión.	7. Se cierra la conexión.

Conexión de sockets TCP en java



Ejemplos Sockets (Servidor)

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

class Serv1 {
  static final int PUERTO=6000;

  public Serv1( ) {
    try {
      ServerSocket skServidor = new ServerSocket( PUERTO );
      System.out.println("Escucho el puerto " + PUERTO );
      for ( int numCli = 0; numCli < 3; numCli++) {
        Socket skCliente = skServidor.accept(); // Crea objeto
        System.out.println("Sirvo al cliente " + numCli);
        OutputStream aux = skCliente.getOutputStream();
        DataOutputStream flujo= new DataOutputStream( aux );
        flujo.writeUTF( "Hola cliente " + numCli );
        skCliente.close();
      }
      System.out.println("Clientes Atendidos");
     } catch( Exception e ) {
     System.out.println( e.getMessage() );
   }
  }
  public static void main( String[] arg ) {
    new Serv1();
    } }
```

Ejemplos Sockets (Cliente)

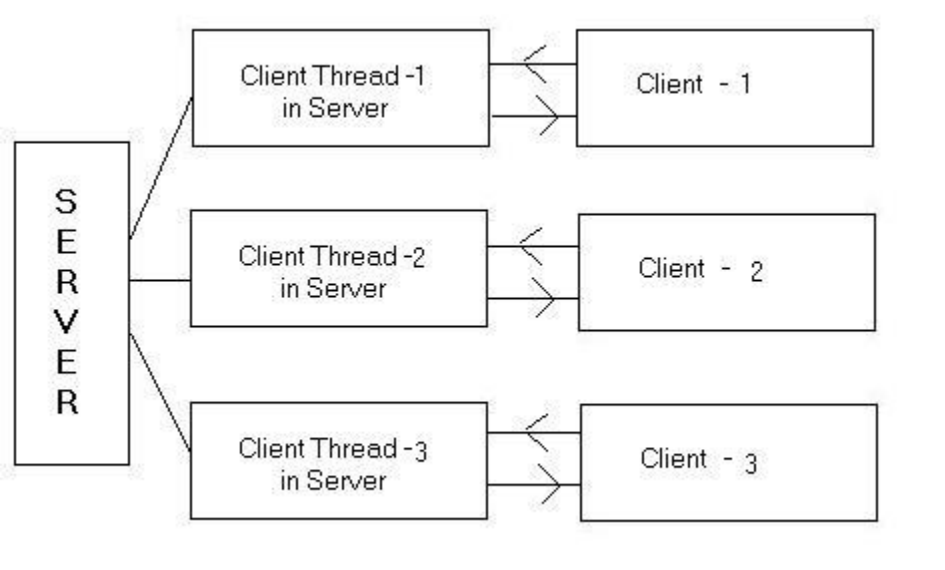
```
import java.io.*;
import java.net.*;

class Clien1 {

  static final String HOST = "localhost";
  static final int Puerto=6000;

  public Clien1( ) {
    try{
     Socket skCliente = new Socket( HOST , Puerto );
     InputStream aux = skCliente.getInputStream();
     DataInputStream flujo = new DataInputStream( aux );
     System.out.println( flujo.readUTF() );
     skCliente.close();
    } catch( Exception e ) {
      System.out.println( e.getMessage() );
    }
  }

  public static void main(String args[]){
    new Clien1();
  }
}
```

Particularidades TCP

- Tamaño del mensaje: Las aplicaciones deciden cuánto leer y cuánto escribir. El sistema subyacente decide cómo transmitirlo.
- Mensajes Perdidos: hay chequeo (ack)
- Control de flujo: TCP trata de hacer coincidir las velocidades de escritura y lectura. Si el escribidor es muy rápido, trata de bloquearlo hasta que el lector haya consumido suficiente.
- Duplicación y orden de mensajes: los paquetes ip contienen identificadores correlativos que permiten al recibidor detectar duplicados o cambiados de orden
- Destino de los mensajes: como se abre una conexión virtual entre ambos extremos, no es necesario especificar a quién va ya que el socket se abre con un connect

Sockets e Hilos

- La aplicación servidora crea un hilo por cada socket TCP creado.
- Objetivo :Servir a cada cliente de forma concurrente



Ejercicio

- Crea un Chat "grupal"
  - Cuando un usuario ejecute el cliente de tu programa deberá indicar la dirección IP del servidor y su Nick. Posteriormente a través de una GUI podrá comunicarse con todos los usuarios conectados al mismo servidor así como ver todos los mensajes de los otros usuarios
  - Está práctica será evaluable y se realizará por parejas.