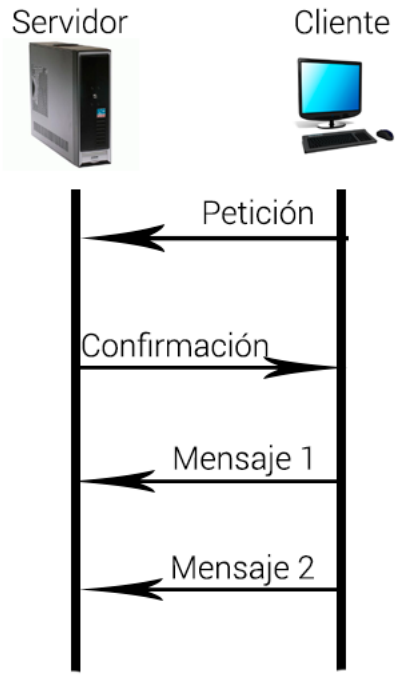
# Usando sockets en Java. Una simple aplicación cliente servidor usando sockets

Los sockets son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor) Java por medio de la librería *java.net* nos provee dos clases: *Socket* para implementar la conexión desde el lado del cliente y *ServerSocket* que nos permitirá manipular la conexión desde el lado del servidor.

Antes de comenzar a ver código (que es lo que todos queremos) explicaré cómo será el funcionamiento de nuestra aplicación cliente servidor usando sockets en Java, cabe resaltar que tanto el cliente como el servidor no necesariamente deben estar implementados en Java, solo deben conocer sus direcciones IP y el puerto por el cual se comunicarán. Para nuestro ejemplo de sockets implementaremos ambos (cliente y servidor) usando Java y se comunicarán usando el puerto 1234 (es bueno elegir los puertos en el rango de 1024 hasta 65535). La dinámica del ejercicio será como se ve:




El servidor estará a la espera de una conexión, en cuanto el cliente inicie enviará un mensaje de petición al servidor, éste le responderá afirmativamente y una vez recibida la confirmación, el cliente enviará un par de mensajes y la conexión finalizará.

Como verás es un ejemplo simple, sin muchas complicaciones, así que manos a la obra, veamos el código:

## Clase Conexion, usando sockets en java

Crearemos una clase llamada *Conexion* en el paquete *sockets* que nos dará los datos que necesitaremos en el cliente y en el servidor

```
package sockets.conexion; import java.io.DataOutputStream; import java.io.IOException; importjava.net.ServerSocket; import java.net.Socket; public class C
= 1234;//Puerto para la conexión private final String HOST = "localhost"; //Host para la conexión protected String mensajeServidor; //Mensajes entrantes
(recibidos) en el servidor protected ServerSocket ss; //Socket del servidor protected Socket cs; //Socket del cliente protected DataOutputStream
salidaServidor, salidaCliente;//Flujo de datos de salida public Conexion(String
tipo) throws IOException //Constructor {if(tipo.equalsIgnoreCase("servidor")) { ss = new ServerSocket(PUERTO);//Se crea el socket para el servidor en
puerto 1234 cs = new Socket(); //Socket para el cliente } else { cs = new Socket(HOST, PUERTO); //Socket para el cliente en localhost en puerto 1234 } }
}
```

La clase *Conexion* simplemente nos brinda los datos que necesitados, mensajes de entrada, flujo de salida socket para el Cliente y socket para el servidor, estos últimos respectivamente inicializados desde el constructor. Las clases *Cliente* y *Servidor* que veremos en breve van a heredar de la clase *Conexion* para tener acceso a los atributos y a los sockets sin problemas. Ahora vemos nuestro servidor:

## La clase Servidor

La clase *Servidor* básicamente estará a la espera de que un cliente se conecte a él usando el socket en el puerto 1234, recibirá los mensajes, los mostrará y cerrará la conexión, es todo.

```
package sockets.servidor; import java.io.BufferedReader; import java.io.DataOutputStream; importjava.io.IOException; import java.io.InputStreamReader; imp
usa el constructor para servidor de Conexion public voidstartServer()//Método para iniciar el servidor { try {
System.out.println("Esperando..."); //Esperando conexión cs = ss.accept(); //Accept comienza el socket y espera una conexión desde un
clienteSystem.out.println("Cliente en línea"); //Se obtiene el flujo de salida del cliente para enviarle mensajessalidaCliente
= new DataOutputStream(cs.getOutputStream()); //Se le envía un mensaje al cliente usando su flujo de salida salidaCliente.writeUTF("Petición recibida y
aceptada"); //Se obtiene el flujo entrante desde el cliente BufferedReader entrada
= new BufferedReader(new InputStreamReader(cs.getInputStream()));while((mensajeServidor = entrada.readLine()) != null) //Mientras haya mensajes desde el
cliente { //Se muestra por pantalla el mensaje recibido System.out.println(mensajeServidor); } System.out.println("Fin de la conexión"); ss.close();//Se
finaliza la conexión con el cliente } catch (Exception e) { System.out.println(e.getMessage()); } } }
```

Tenemos en este código entonces al servidor que espera la conexión desde el cliente en el método *accept()*, una vez ésta sucede, envía un mensaje de confirmación con *writeUTF("mensaje")*, lee todos los mensajes enviados por el cliente con *readLine()* y cierra la conexión con el método *close()*. Veamos ahora la clase Cliente

## La clase Cliente

La clase Cliente, establecerá la conexión con el servidor usando un socket en localhost y el puerto 1234, una vez establece la conexión escribe dos mensajes en el servidor usando un ciclo for y cierra la conexión. Veamos:

```
package sockets.cliente; import java.io.DataOutputStream; import java.io.IOException; importsockets.conexion.Conexion; public class Cliente extends Conexi
usa el constructor para cliente de Conexion public void startClient()//Método para iniciar el cliente { try { //Flujo de datos hacia el
servidor salidaServidor = newDataOutputStream(cs.getOutputStream()); //Se enviarán dos mensajes for (int i = 0; i < 2; i++) { //Se escribe en el servidor
usando su flujo de datos salidaServidor.writeUTF("Este es el mensaje número " + (i+1) + "\n"); } cs.close();//Fin de la conexión } catch (Exception e) {
System.out.println(e.getMessage()); } } }
```

Vemos entonces que el cliente obtiene el flujo de salida de datos hacia el Servidor con el método *getOutputStream* y lo usa para enviarle un par de mensajes con el método *writeUTF("mensaje...")*, finalmente cierra la conexión con *close()*. Notar que si se desea enviar más de dos mensajes bastará con cambiar el límite superior del ciclo for *i < 100* por ejemplo.

Pues muy bien ya tenemos todo, aunque los segmentos de código con un poco extensos (no demasiado) son bastante simples y están bien comentados, a continuación pondré los códigos para los respectivos main que harán uso del servidor y del cliente.

## Código para usar el servidor con Sockets en Java

```
package sockets.main; import java.io.IOException; import sockets.servidor.Servidor; //Clase principal que hará uso del
servidor public class MainServidor { public static void main(String[] args) throws IOException { Servidor serv = new Servidor(); //Se crea el
servidor System.out.println("Iniciando servidor\n"); serv.startServer(); //Se inicia el servidor } }
```

## Código para usar el cliente con Sockets en Java

```
package sockets.main; import java.io.IOException; import sockets.cliente.Cliente; //Clase principal que hará uso del
cliente public class MainCliente { public static void main(String[] args) throws IOException { Cliente cli = new Cliente(); //Se crea el
cliente System.out.println("Iniciando cliente\n"); cli.startClient(); //Se inicia el cliente } }
```