

# Práctica 1: INTRODUCCIÓN A LOS SOCKETS

## 1. Objetivos

El objetivo de la siguiente práctica es dar solución a las siguientes preguntas:

- ¿Qué es un socket? ¿Para qué se utiliza?
- Cómo utilizar sockets en Java: clases de sockets
- Arquitectura Cliente-Servidor: Implementación de una aplicación basada en dicho paradigma

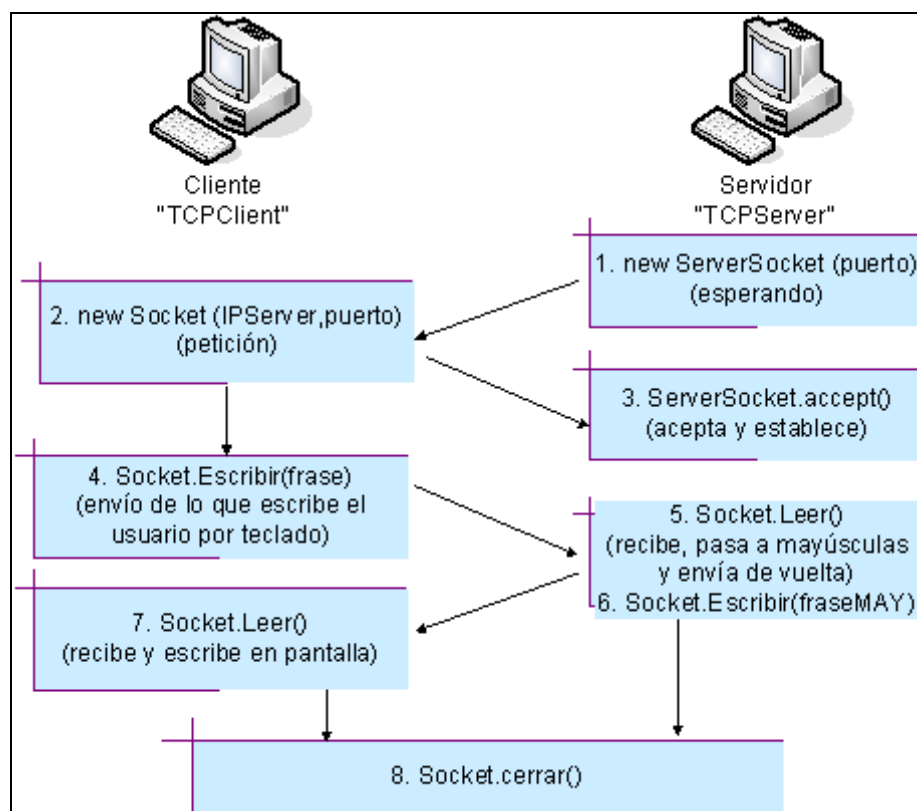
Al finalizar la práctica el alumno deberá ser capaz de:

- Entender el funcionamiento y manejo de las clases Socket y ServerSocket
- Implementar una aplicación básica de Cliente-Servidor

## 2. Descripción de la práctica

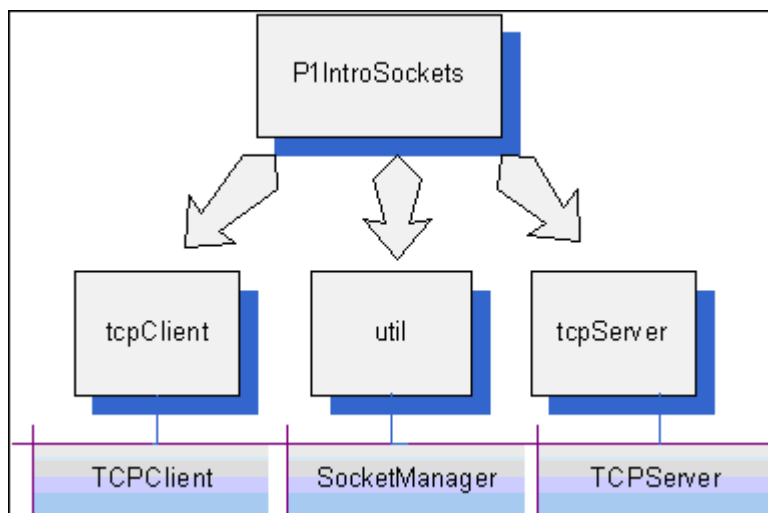
Se desea implementar una arquitectura Cliente-Servidor, en la cual, la aplicación cliente recogerá una frase introducida por teclado por el usuario, la enviará a la aplicación servidora a través de un socket, y ésta se la devolverá al cliente después de convertir los caracteres a mayúsculas, quien finalmente, mostrará por pantalla el resultado.

De forma abreviada, podemos ver el proceso básico en el siguiente esquema simplificado:




Especificaciones requeridas para la realización de la práctica:

- a) Diseñar la estructura del proyecto en 3 paquetes de clases diferenciados, siguiendo este diagrama:



Dentro del paquete **util** se habrá de implementar la clase **SocketManager** (para esta práctica; en adelante el paquete **util** contendrá todas las clases comunes para las aplicaciones cliente y servidora). Esta clase debe implementar los métodos básicos de uso por un socket: **SocketManager**, como su nombre sugiere, será un objeto que haga transparente al programador el uso de los métodos principales en el manejo de sockets. Contará con 3 atributos (*Socket*, *BufferedReader* y *DataOutputStream*) y al menos, los métodos siguientes:

- Constructores **SocketManager(parámetros)**: uno de ellos recibe un socket como parámetro, otro la dirección IP y puerto del servidor y un último que recibe el nombre del servidor y el puerto de conexión.
  - InicializaStreams()**: crea los buffers de lectura y escritura que se asociarán como canales de entrada y salida en el socket.
  - CerrarStreams()**: libera de la memoria el espacio ocupado por los streams de lectura y escritura en los buffers.
  - CerrarSocket()**: se encarga de cerrar el socket creado cuando termina la comunicación.
  - Leer()**: lee línea a línea lo que se va recibiendo en el buffer de lectura a través del socket.
  - Escribir(parámetros)**: escribe a través del buffer para escritura el objeto que se pasa como parámetro, enviándolo a través del socket.
- b) Implementar la clase **TCPClient** contenida dentro del paquete **tcpClient**. Esta clase, además de importar el paquete **util**, debe implementar un main que contenga al menos, esta secuencia de acciones:
- Instancia de objeto **SocketManager**.
  - Instancia de objeto **BufferedReader**, cuya entrada asociada sea el teclado y leer así la frase introducida por el usuario.
  - Enviar al servidor dicha frase a través del socket establecido, y leer a continuación los datos reconvertidos a mayúsculas que se espera recibir del servidor.

	PROGRAMACIÓN CON SOCKETS	Práctica 1
---	--------------------------	------------

- d. Presentar al usuario por pantalla la frase leída desde el socket.
- c) Implementar la clase **TCPServer** contenida dentro del paquete **tcpServer**. Esta clase, además de importar el paquete **util** para hacer uso de los métodos definidos en la clase **SocketManager**, debe implementar un main que contenga esta secuencia de acciones:
  - a. Crear un **ServerSocket** para atender peticiones de conexión del cliente.
  - b. Aceptar la conexión solicitada por el cliente al puerto especificado, creando un **SocketManager** asociado al socket con el cliente.
  - c. Leer los datos enviados por el cliente.
  - d. Convertir los datos leídos a mayúsculas.
  - e. Escribir los datos en el socket para que los pueda leer el cliente.
  - f. Cerrar el socket.