PROGRAMACIÓN DE SERVICIOS Y PROCESOS – TAREA 4

📖 **Descripción del Problema:**

- Desarrollar una aplicación cliente-servidor en Java para gestionar pedidos realizados desde diferentes almacenes.
- Los clientes representan almacenes que pueden realizar pedidos de diferentes productos.
- El servidor procesa los pedidos, comprueba el stock y responde si el pedido se puede atender o si hay falta de stock.
- La comunicación es bidireccional, y el servidor maneja múltiples almacenes simultáneamente usando un manejador de hilos.
- El stock y los productos disponibles se configuran mediante un archivo de configuración.

⚖️ **Requisitos:**

Servidor Socket:

- Escuchar en un puerto especificado y aceptar conexiones de múltiples almacenes.
- Recibir los pedidos enviados por los almacenes (producto y cantidad).
- Comprobar el stock y responder si el pedido se acepta o se rechaza por falta de stock.
- Guardar un registro de cada pedido procesado en un archivo de log.
- Actualizar dinámicamente el stock si se modifica el archivo de configuración.

Clientes Socket:

- Cada almacén puede realizar pedidos aleatorios cada 5 o 10 segundos.
- Los pedidos incluyen el nombre del producto y una cantidad aleatoria.
- Recibir la confirmación o rechazo del pedido y mostrarlo en la consola.

💾 Archivos Utilizados:

Archivo de Configuración (config_stock.properties):

producto1=100
producto2=50
producto3=200
producto4=150
producto5=300

Archivo de Log (pedidos.log):

[2025-02-21 11:30:45] PEDIDO - Almacén: 192.168.1.10 - Producto: producto1 - Cantidad: 20 - ACEPTADO
[2025-02-21 11:35:20] PEDIDO - Almacén: 192.168.1.11 - Producto: producto2 - Cantidad: 60 - RECHAZADO (Stock insuficiente)

🌐 **Comunicación Bidireccional:**

- El cliente envía el nombre del producto y la cantidad al servidor.
- El servidor procesa el pedido y responde indicando si ha sido aceptado o rechazado.
- La conexión permanece abierta para realizar pedidos periódicamente.

💡 Ejemplo de Ejecución:

Servidor:

Servidor de pedidos iniciado en el puerto 12345
Almacén conectado: /192.168.1.10
Almacén conectado: /192.168.1.11
Pedido procesado: producto1, cantidad: 20 - ACEPTADO
Pedido procesado: producto2, cantidad: 60 - RECHAZADO (Stock insuficiente)

Cliente Almacén 1:

Conectado al servidor de pedidos.
Ingrese su pedido: producto1,20
Respuesta del servidor: Pedido aceptado: producto1 - Cantidad: 20

Cliente Almacén 2:

Conectado al servidor de pedidos.
Ingrese su pedido: producto2,60
Respuesta del servidor: Pedido rechazado: Stock insuficiente.

🛠️ Indicaciones de entrega

- Crear un fichero comprimido que incluya:
  - El proyecto Java completo.
  - Un documento explicativo con:
    - Descripción de la solución implementada.
    - Capturas de pantalla del programa en ejecución.
    - Resultados obtenidos.
- **Nomenclatura del archivo:** PSPXX_TareaYY_apellido1_apellido2_nombre.
- Ejemplo: PSP02_Tarea01_Perez_Lopez_Juan.zip.

✨ Consejos y Recomendaciones

- Usar estructuras como Queue para gestionar clientes.
- Manejar concurrencia con hilos y sincronización.
- Implementar un sistema de logs para facilitar la depuración.

📊 Criterios de Calificación: Total 10 puntos

- (2 puntos): Implementación correcta del servidor para aceptar peticiones de clientes.
- (1 punto): Correcto manejo del archivo de configuración.
- (2 puntos): Comunicación bidireccional entre servidor y clientes.
- (1 punto): Registro adecuado de pedidos en el archivo log.
- (1 punto): Generación correcta de pedidos simulados en los clientes.
- (1 punto): Manejo correcto de errores y desconexiones inesperadas de los clientes.
- (1 punto): Actualización dinámica del stock.
- (1 punto): Buenas prácticas en el código, con comentarios y estructura clara.

🔧 Resultados de aprendizaje y criterios de evaluación

Programa mecanismos de comunicación en red empleando sockets y analizando el escenario de ejecución:

- Se han identificado escenarios que precisan establecer comunicación en red.
- Se han identificado los roles de cliente y servidor.
- Se han reconocido librerías y mecanismos del lenguaje para la comunicación en red.
- Se ha analizado el concepto de socket, sus tipos y características.
- Se han utilizado sockets para programar aplicaciones cliente-servidor.
- Se ha desarrollado una aplicación servidor y verificado su funcionamiento.
- Se han utilizado hilos para gestionar la concurrencia y la comunicación en red.

Resolución:

Para la resolución de esta tarea, se han creado dos proyectos uno para el Almacén principal (Server) y otro para los almacenes de venta al público (Client). Cada proyecto contendrá un fichero .properties que nos permitirá configurar el servidor y el cliente respectivamente.

Los ficheros .properties contendrán una lista de productos que se podrán actualizar de forma dinámica para actualizar el catálogo de datos.

El fichero del servidor se diferenciará del de cliente. EL de servidor contendrá el producto, y la cantidad que existe en stock, y el de cliente solo contendrá el producto.

Se adjunta un ejemplo del config.properties de ambos programas.

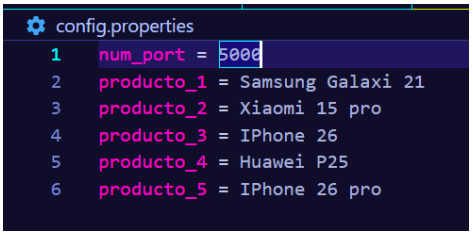
Ejemplo de fichero de servidor:

En la siguiente imagen se ve que la primera key es el num_port, que nos permitirá configuar el puerto de conexión del programa. Seguidamente se puede observar los diferentes productos que existen en el almacen.

Se puede observar la key que será el numero de producto y su valor será el nombre del producto y la cantidad en stock, ambos estarán separados por una coma, que más adelante nos permitirá separa estos datos.

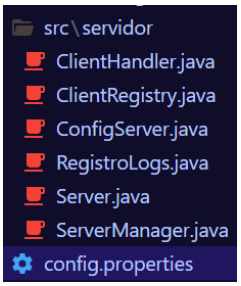
Ejemplo de fichero del cliente:

Este fichero es similar al fichero de Servidor, la diferencia radica en que el cliente no tendrá la información de la cantidad que existe en el almacén principal, es decir, solo sabrá los productos que existen y en el almacén principal.



Descripción del Servidor (Almacén Principal).

El servidor está compuesto por seis clase, la que nos permiten crear la funcionalidad desea. Desglosemos las clases para comprender su funcionamiento.



Clase Server (main).

Es la clase principal del programa y permitirá ejecutar el servidor. No contiene ninguna funcionalidad más.

```java
public class Server {

    /**
     * Método main del servidor
     */
    Run | Debug
    public static void main(String[] args) throws InterruptedException {
        // Se instancia un objeto de la clase ServerManager para crear el servidor
        ServerManager serverManager = new ServerManager();
        serverManager.start(); // Se ejecuta el servdor.
    }

}
```

Clase ServerManager:

Esta clase construirá el servidor y permitirá que se inicie el servidor correctamente. A continuación, se explicará la lógica que se a usado para dar funcionalidad a la clase.

Para el funcionamiento de esta clase se ha instanciado las siguientes variables o clases:

```java
// Se crea la lista de clientes
private ClientRegistry listaClientes;
// Se instancia un objeto Cliente
private ClientHandler cliente;
// Se instancia una variable que almacenará el puerto
private final int PORT;
// Se instancia un serversocket
private ServerSocket serversocket;
// Se instancia un objeto de la clase RegistroLogs
private RegistroLogs logger;
// Se almacenan los productos
private Map<String, Integer> productos;
// Se instancia una cola
private final ExecutorService threadPool;
// Se instancian para actualizar los pedidos
private boolean actuliza;
// Variable para evitar que se realice dos veces la actulización.
private static long ultimaActualización = 0;
```

Métodos de la clase:

- **Método constructor():**

El constructor instanciará las variables y clases para poder usarlas más adelante. Así podremos asignales los valores correspondientes o llamara a métodos de otras clases que permitirán darle una mayor funcionalidad a la clase ServerManager.

- **Método start():**

Este método nos permite ejecutar el servidor, y inicia un bucle infinito mientras el servidor este ejecutado. En este bucle se realizarán las conexiones con los clientes, y a la vez se registrarán en una lista, que permitirá controlar a quien van los mensajes de respuesta y así evitar que no se envíen mensajes a los clientes que no han realizado el pedido. Se ejecutará un pool para controlar las conexiones de los clientes y así interactuar con los mensajes que se reciban. También ira controlando si se ha realizado algún cambio en el fichero config.properties, en caso de que se modifique el fichero se actualizará la lista de productos. En caso que se finalice la conexión del servidor se llama al método para finalizar el proceso.

También se controlarán los errores, registrando cualquier incidencia.

- **Método realizarPedido():**

El método realizarPedido() recibirá como parámetros un String con el producto y la cantidad solicita, y otro String con el identificador del almacén que realiza el pedido.

Lo primero que vamos ha hacer es separar el String pedido, que contiene el producto solicitado y la cantidad.

Para realizar esto utilizamos a función Split() y le indicamos que nos separe el contenido desde la coma que se contiene. Y lo almacenaremos en un array.

Se llamará al método para comprobarStock() para ver si se puede realizar el pedido o no.

Si se recibe “ACEPTADO”, se enviará el mensaje al cliente de que su pedido ha sido tramitado y se registrará.

Si se recibe “RECHAZADO”, se enviará el mensaje al cliente de que su pedido no ha sido tramitado y se registrará.

Si se recibe cualquier otra cosa, en este caso se recibirá “NO EXISTE”, se indicará al cliente que el pedido que ha realizado no puede realizar ya que el producto indicado no existe.

- **Método comprobarStock():**

Con este método se comprobará si el pedido que ha realizado el cliente, puede realizar.

Primero se comprueba si existe el producto solicitado, de este modo controlamos cualquier error que pueda haberse producido en el almacén cliente, y además sirve para identificar si la lista de productos no esta actualizada.

En caso de que existe se comprobará si en el almacén existe stock de producto solicitado para realizar el pedido, en caso de haber stock suficiente se devuelve “ACEPTADO”, y si no hay stock suficiente se devuleve “RECHAZADO”.

Si por algún caso el producto solicitado no existiera se enviría el mensaje “NO EXISTE”.

- **Método existeProducto():**

Con este método se comprueba que si el producto existe o no en la lista de productos. En caso de que exista se devolverá un valor booleano true, si no existe se devolverá false.

- **Método serviceWatch():**

Este método construye un hilo que estará a la escucha de los cambios que se produzcan en el fichero config.properties. este hilo gracias el método tacke(), de la clase WatchService estará el hilo bloqueado hasta que se realice el cambio en el fichero.

Como se controla el cambio de todos los fichero que se contengan dentro de la carpeta indicada, debemos controlar que el cambio ser haya producido en el fichero config.properties, además para evitar que se actualice varias veces, también se controla el tiempo de actualización. Si el tiempo transcurrido es menor a 500

milisegundos, no se realizará el cambio. Esto problema surge por el sistema operativo y como gestiona los cambios en las carpetas.

Se llamará al método actProductos() de la clase ConfigServer. Cuando se haya cargado la nueva lista de productos se llamará al método actualizarLista() que actualizara la lista de productos.

- **Metodo actualizarLista():**

Primero se vacía la lista, ya que en el .properties se especificará el nuevo tock actualizado de productos.

Este método llama al método de getProductos() de la Clase ConfigServer para cargar el nuevo stock de productos.

Seguidamente, se informará de la actualización y de los nuevos productos.

- **Método shutdown():**

Método que finaliza el servidor. Servirá para finalizar el servidor y el hilo de los clientes.

Clase ConfigServer():

Esta clase configurará el servidor, obteniendo del fichero config.properties los siguientes datos: número del puerto de conexión, y los productos con el stock que tiene el almacén principal.

En esta clase se instanciarán los siguientes atributos:

```
// Se instancia los atributos
private int port; // almacenará el número del puerto de conexión.
private Map<String, Integer> listaProductos; // Se almacenarán los productos y sus cantidades
private Properties properties; // Se contedrá los parámetros del propertie
private RegistroLogs logger; // Se para almacenar el logger
```

Para el registro de productos como tiene dos valores, he decidido crear un Map que registre como clave el nombre del producto y su valor será un entero, para almacenar la cantidad de stock que tiene ese producto.

Para obtener los datos del fichero config.properties, necesitaremos los siguientes métodos:

- **Constructor ConfigServer:**

Se recibe como parámetro el logger de la clase RegistroLogs, para registrar en el mismo fichero todos los acontecimientos que hayan sucedido durante la ejecución del servidor.

Se instancia un Properties para poder obtener los datos del config.properties.

Se instanciará un nuevo Map que almacenará los productos.

- **Método ConfigServer:**

Este método abrirá el archivo config.properties y obtendrá los datos que contiene, lo primero que hará es obtener el número del puerto de conexión.

En caso de querer configurar el host del servidor también se podría agregar en el config.propertie y luego mediante este método.

Una vez, tengamos configurado la parte técnica del servidor se llamará al método actProductos, este método obtendrá los productos y la cantidad de stock que hay de esos productos.

Seguidamente se informará que se ha configurado el servidor y mostrará la lista de productos que se han obtenido del método actProductos.

- **Metodo actProductos():**

Este método abrirá el fichero config.propertie y obtendrá los productos y las cantidades que hay en stock de los mismos.

Primero se limpia el Map con los productos en caso de que tenga información. Para ello se instanciará de nuevo el Map

una vez este el Map limpio, mediante un bucle for obtenemos el key del propertie para obtener la información.

Si el key es distinto a “num_port”, se cargarás los productos.

Primero se tiene se obtiene el valor completo del key que hemos obtenido. Seguidamente utilizamos Split para separar el producto y la cantidad. Tal y como se explico anteriormente, el producto y la cantidad están separados por una “,”. Gracias a esto se facilita la separación.

Como se almacena el producto y la cantidad en un Array, esto nos facilitará trabajar con los datos.

Antes de registrar el producto vamos a verificar si la cantidad esta bien configurada para la poder convertirlo en un valor entero.

Para realizar esto, lo primero que se va ha hacer es eliminar los posibles espacios en blancos que tenga. Ahora se comprobará si el valor es un número. En caso de que no sea un número se mostrará un mensaje informando de que se debe de verificar el producto y su cantidad.

Si por lo contrarío el valor es numérico, se registra el producto y la cantidad en el Map, se asignará el producto como Key, y la cantidad como valor.

- **Método getPort():**

Este método devolverá un valor entero, para configurar el puerto del servidor.

- **Método getProductos():**

Este método devuelve un Map con los productos y la cantidad.

Clase RegistroLogs:

Está clase nos permite registrar la actividad que se produzca en el servidor y así por consultarla más adelante.

Para mostrar el documento con un formato concreto, vamos a crear una clase interna que nos permitirá eliminar que muestre la clase que llama al método que registra los datos.

También, se instanciará un formato de fecha concreto, este será el siguiente: ejemplo.

“Lunes, 30 de marzo de 2025, 18:00:24”

La hora que se muestre tendrá el formato 24 horas.

- **Constructor:**

El constructor, verificará si existe el fichero donde se almacenarán los registros y en caso de que no exista se creará el fichero.

Una vez tengamos el fichero creado o exista ya, se abrirá para registrar los acontecimientos que sucedan en el servidor.

- **Método regInicioServidor:**

Este método registrará el inicio del servidor

- **Método alertaServer:**

Este método registrará los errores que sucedan en el servidor.

- **Método regAlmacen:**

Método que registrará la conexión del cliente con el servidor.

- **Método regPedido:**

Método que registrará el pedido realizado por el cliente.

- **Método pedidoError:**

Método que registrará cuando suceda algún error con el pedido.

Clase ClientRegistry:

Esta clase creará una cola que gestionará los múltiples hilos de la clase ClientHandler. Se creará una variable de tipo **QUEUE<ClientHandler>** almacenará los hilos, para ejecutar los clientes de una forma adecuada.

- **Método addCliente:**

Este método añadirá los nuevos hilos de los clientes que se conecten.

- **Método removeClietne:**

Este método eliminará los clientes de la lista cuando se desconecten.

- **Método responderCliente:**

Este método comprobará a que cliente se ha de responder y que el servidor mande la información correcta a cada cliente. De este modo se nunca se enviará un mensaje equivocado a los clientes.

- **Método getClientesCount():**

Método que devuelve la cantidad de clientes conectados.

Clase ClientHandler:

Esta última clase representa al cliente y será el hilo que interactuará con el cliente.

Además, se implementará un Runnable para poder ejecutar el hilo.

Está clase contendrá los siguientes atributos:

```
// Se instancia los atributos de la clase.
private final Socket socket; // Socket del cliente
private ClientRegistry listaClientes; // Lista que recibe el la lista donde se almacenan los clientes
private ServerManager server; // Se almacenará el servidor
private String idAlmacen; // Variable para almacenar el id del cliente
private BufferedReader input; // Se instancia un input para obtener los datos que envie el cliente
private PrintWriter output; // Se instanacia un output para enviar los mensajes al cliente
private RegistroLogs logger;
```

- **Constructor:**

Con esta clase recibirá el socket, la lista de clientes conectados, el logger para registrar las conexiones, y el ServerManagener para interactuar entre el cliente y el servidor.

También, se instanciarán las variables que permitirán recibir y enviar información entre el cliente y el servidor.

- Método generaID:

Este método generará un ID para cada cliente.

- Método sendMessage:

Este método envía el mensaje al servidor.

- Método getIdAlmacen.

Este método para devolver el identificador del almacén.

- Método run():

Se modifica el método run, para generar nuestro hilo y comunicarnos con el cliente.

Se registrará en la lista el cliente, y se indicará que el cliente con el id indicado se ha conectado. Además, se registrará en el "logger".

Utilizamos una variable auxiliar para recibir el pedido del cliente.

Si el cliente esta conectado ejecutamos un bucle while que recibirá el pedido del cliente.

Se realiza el pedido utilizando el método realizarPedido de la clase SErverManager, este deberá devolver un mensaje, pero para evitar errores, se utiliza un if que comprobará si se recibe o no un mensaje.

Si se recibe un mensaje se enviará al cliente utilizando el método responderCliente de la clase ClientRegistry.

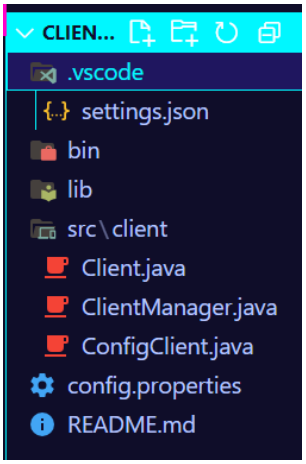
De lo contrario se mostrará en la consola de servidor el mensaje de un error al realizar el pedido.

Proyecto ClientSocket

Para la realización de esta tarea ya se indicó que se realizaron dos proyectos uno para el servidor y otro para el cliente.

A continuación, se mostrará como se ha desarrollado el proyecto para el cliente.

Para la estructura del proyecto hemos creado 3 clases, para hacer que sea más fácil actualizarlo en un futuro.



Clase Client:

Esta clase es la clase que ejecuta el cliente, contiene el método main, que llama al método star() de la clase que construye el cliente ClientManager.

Clase ClientManager:

Esta clase es la que construirá la comunicación entre el cliente y el servidor. Además, contendrá la lógica para generar el pedido al servidor.

Para esta clase se necesitan las siguientes variables.

```
// Se instancia la configuración del servidor.
private ConfigClient configClient = new ConfigClient();
// Se instancia una variable para controlar los cambios y evitar que se repita
// la actualización debido al SO.
private long ultimaActualización;
private List<String> productos;
private final int PORT;
private final String HOST;
private Random random;
private String producto;
```

- **Constructor:**

El constructor ejecutará la configuración del cliente, que permitirá configurar el puerto y la lista de productos que se podrán solicitar al almacén principal.

Una vez se han instanciado todas la variables se ejecuta el servicewatch, para mantenerse a la escucha de los posibles cambios en el fichero config.properties del proyecto de cliente. Hay que recordar que en esto fichero no contiene la cantidad de stock, que es la diferencia que hay con el del servidor.

- **Método start():**

Este método realiza la conexión con el servidor, si el servidor no está activo mandará un mensaje informado de que no se puede conectar y finalizará la ejecución del cliente.

También se instancia el bufferedReader y el PrintWriter, para enviar y recibir los datos entre el cliente y el servidor.

Si se conecta con el servidor se ejecuta un hilo para recibir los mensajes del servidor.

Por último, se utiliza un bucle while infinito que generará los pedidos y los enviará al servidor.

En el while se ha utilizado un thread.sleep() para congelar el proceso durante 5 o 10 segundos, este tiempo se generá de forma aleatoria también.

- **Método serviceWatch():**

Este método contine un hilo que estará dormido hasta que se realice un cambio en la ruta donde se almacena el config.properties. Para que se actualice solo cuando se cambia el fichero config.properties se utiliza un bucle que comprueba si el fichero que se ha modifica es el config.properties.

Cuando se confirma que es ese el fichero que se ha modificado, se llama al método actProductos() de la clase configClient, y seguidamente se llama al método actProductos():

- **Método actProductos():**

Se actualizan la lista de productos.

Clase ConfigClient:

Esta clase contiene los método para abrir el fichero config.properties, que contiene los datos para configurar el puerto de conexión y la lista de productos que se pueden solicitar.

- **Constructor:**

El constructor instanciará las variables Properties y la lista.

- **Método configurar:**

Este método abrirá el fichero config.properties y obtendrá los datos para configurar el fichero.

Lo primero que hará es obtener el número del puerto de conexión, y seguidamente añadirá los productos a la lista.

Seguidamente se devolverán dos mensajes informando que se ha configurado el servidor y se mostrarán los productos que se pueden solicitar.

- **Método actProductos:**

Este método permitirá actualizar los productos cuando se realice algún cambio en el fichero config.properties. Este método se llamará desde la clase ClientManager.

Primero se vacía la lista, y seguidamente se añadirán los nuevos productos a la lista. Cuando se finalice la actualización, se mostrará por consola los cambios realizados.

- **Getters:**

Hay dos métodos que devolverán el número del puerto y otro para devolver la lista de productos.

Funcionamiento del proyecto

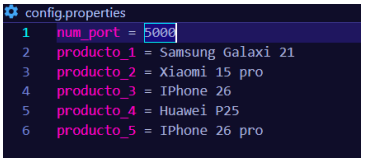
A continuación, se va a ejecutar el proyecto y vamos a ver el funcionamiento.

Lo primero que se va a mostrar es la ejecución del cliente antes de ejecutar el cliente para mostrar el resultado que devuelve el cliente.

Como se puede observar el cliente se ha configurado y ha mostrado todos los productos que se podrán solicitar, pero al no estar conectado el servidor se finaliza la aplicación.

```
PS C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSocket>  & 'C:\Pro
Messages' '-cp' 'C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSock
Se ha configurado el servidor
El dispositivo Huawei P25 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 pro se encuentra disponible para ser solicitado
El dispositivo Xiaomi 15 pro se encuentra disponible para ser solicitado
El dispositivo Samsung Galaxi 21 se encuentra disponible para ser solicitado
Estoy a pendiente de que modifiques el fichero
Error al conectarse al servidor -> Connection refused: connect
```

Vamos a aprovechar, a comprobar si ha cargado correctamente la lista de dispositivos.



Así es, se han cargado todos los productos que contiene el properties.

Ahora se ejecutará el servidor.

```
PS C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\AlmacenServer\AlmacenServer>  & 'C:\Pr
MACION\PSP\TAREAS\tarea_4\AlmacenServer\AlmacenServer\bin' 'servidor.Server'
Se ha configurado el servidor
El dispositivo Xiaomi 15 pro tiene un Stock de 60 unidades
El dispositivo Samsung Galaxi 21 tiene un Stock de 154 unidades
El dispositivo Huawei P25 tiene un Stock de 30 unidades
El dispositivo IPhone 26 pro tiene un Stock de 16 unidades
El dispositivo IPhone 26 tiene un Stock de 60 unidades
Estoy a pendiente de que modifiques el fichero
El servidor esta a inciado en el puerto 5000
```

Como se puede observar en la imagen anterior, se ha conectado correctamente el servidor y esta a la espera de que se conecten los clientes para realizar los pedidos.

Comprobemos si se han cargado todos los datos tal y como están en el fichero config.properties.

config.properties

```
num_port = 5000
producto_1 = Samsung Galaxi 21,154
producto_2 = Xiaomi 15 pro,60
producto_3 = IPhone 26,60
producto_4 = Huawei P25,30
producto_5 = IPhone 26 pro,16
```

Al igual que con el servidor esta todo correcto, y además se ha conseguido separar la cantidad del producto y se muestra de forma independiente.

Volvemos a ejecutar el cliente.

```
PS C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSocket>  & 'C:\Program
MACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSocket\bin' 'client.Client'
Se ha configurado el servidor
El dispositivo Huawei P25 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 pro se encuentra disponible para ser solicitado
El dispositivo Xiaomi 15 pro se encuentra disponible para ser solicitado
El dispositivo Samsung Galaxi 21 se encuentra disponible para ser solicitado
Estoy a pendiente de que modifiques el fichero
Me he conectado al Almacén principal 5000
Se han solicitado '33' unidades del producto 'Xiaomi 15 pro'
Su pedido ha sido realizado
```

Como se puede ver a hora esta funcionando y realizando ya las primeras solicitudes de pedido. Además, se puede observar que el almacén le está respondiendo.

Como se puede observar en la siguiente imagen el almacén se ha quedado sin stock y le esta respondiendo que ya no se puede realizar el pedido.

```
Su pedido ha sido realizado
Se han solicitado '14' unidades del producto 'Huawei P25'
Su pedido ha sido rechazado por falta de stock
Se han solicitado '30' unidades del producto 'Huawei P25'
Su pedido ha sido rechazado por falta de stock
Se han solicitado '35' unidades del producto 'IPhone 26 pro'
Su pedido ha sido rechazado por falta de stock
```

Vamos a finalizar los programas para ver si el logger a registrado correctamente lo que ha sucedido en el programa.

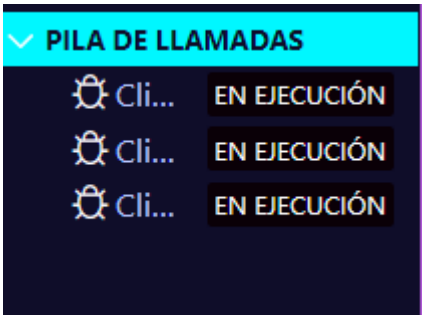
```
-------------------------------------------------------
[viernes, 04 de abril de 2025, 19:19:24 ] - Inicio del servidor en el puerto, 5000
-------------------------------------------------------
[viernes, 04 de abril de 2025, 19:21:53 ] - Almac◆n almacenIP:_/127.0.0.1_53585se ha conectado.
[viernes, 04 de abril de 2025, 19:21:53 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: Xiaomi 15 pro - Cantidad
solicitada: 33 - ACEPTADO
[viernes, 04 de abril de 2025, 19:21:59 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: IPhone 26 pro - Cantidad
solicitada: 18 - RECHAZADO (Stock insuficiente)
[viernes, 04 de abril de 2025, 19:22:09 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: IPhone 26 - Cantidad solicitada: 41
- ACEPTADO
[viernes, 04 de abril de 2025, 19:22:15 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: Xiaomi 15 pro - Cantidad
solicitada: 15 - ACEPTADO
```

Como se puede observar en la imagen anterior, se han registrado las solicitudes y la conexión tanto del inicio del servidor como la conexión del cliente al servidor.

Además, también a registrado los pedidos que han sido rechazados.

```
solicitada: 29 - RECHAZADO (Stock insuficiente)
[viernes, 04 de abril de 2025, 19:24:45 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: IPhone 26 pro - Cantidad
solicitada: 20 - RECHAZADO (Stock insuficiente)
[viernes, 04 de abril de 2025, 19:24:53 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: Samsung Galaxi 21 - Cantidad
solicitada: 6 - ACEPTADO
[viernes, 04 de abril de 2025, 19:24:58 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: IPhone 26 pro - Cantidad
solicitada: 42 - RECHAZADO (Stock insuficiente)
[viernes, 04 de abril de 2025, 19:25:08 ] Pedido realizado por Almac◆n:  almacenIP:_/127.0.0.1_53585 - Producto solicitado: IPhone 26 pro - Cantidad
solicitada: 36 - RECHAZADO (Stock insuficiente)
Error de proceso: viernes, 04 de abril de 2025, 19:25:13  Error en el hilo del cliente almacenIP:_/127.0.0.1_53585: ->Connection reset
```

Se volverá a ejecutar el servidor y varios clientes para ver si se realiza correctamente la ejecución. Como se puede observar en la siguiente imagen se han ejecutado 3 clientes.

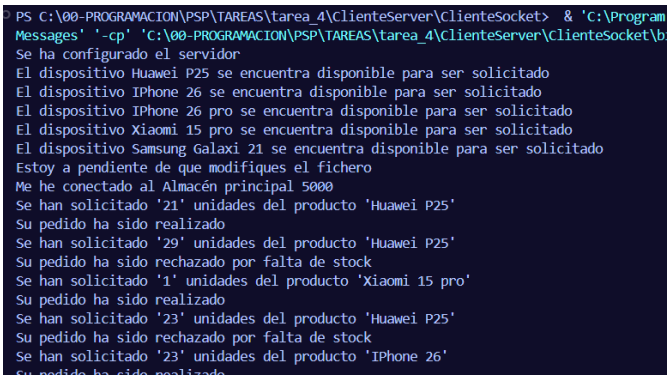


A continuación, se mostrarán las imágenes de los diferentes clientes.

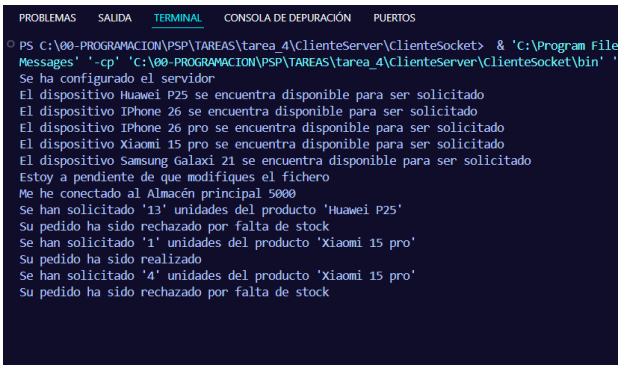
Clietne 1:

```
PS C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSocket>  & 'C:\Prog
Messages' '-cp' 'C:\00-PROGRAMACION\PSP\TAREAS\tarea_4\ClienteServer\ClienteSocke
Se ha configurado el servidor
El dispositivo Huawei P25 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 se encuentra disponible para ser solicitado
El dispositivo IPhone 26 pro se encuentra disponible para ser solicitado
El dispositivo Xiaomi 15 pro se encuentra disponible para ser solicitado
El dispositivo Samsung Galaxi 21 se encuentra disponible para ser solicitado
Estoy a pendiente de que modifiques el fichero
Me he conectado al Almacén principal 5000
Se han solicitado '2' unidades del producto 'IPhone 26 pro'
Su pedido ha sido realizado
Se han solicitado '30' unidades del producto 'Huawei P25'
Su pedido ha sido rechazado por falta de stock
Se han solicitado '18' unidades del producto 'Samsung Galaxi 21'
Su pedido ha sido realizado
Se han solicitado '12' unidades del producto 'IPhone 26 pro'
Su pedido ha sido realizado
Se han solicitado '38' unidades del producto 'IPhone 26'
Su pedido ha sido rechazado por falta de stock
Se han solicitado '0' unidades del producto 'Huawei P25'
Su pedido ha sido realizado
```

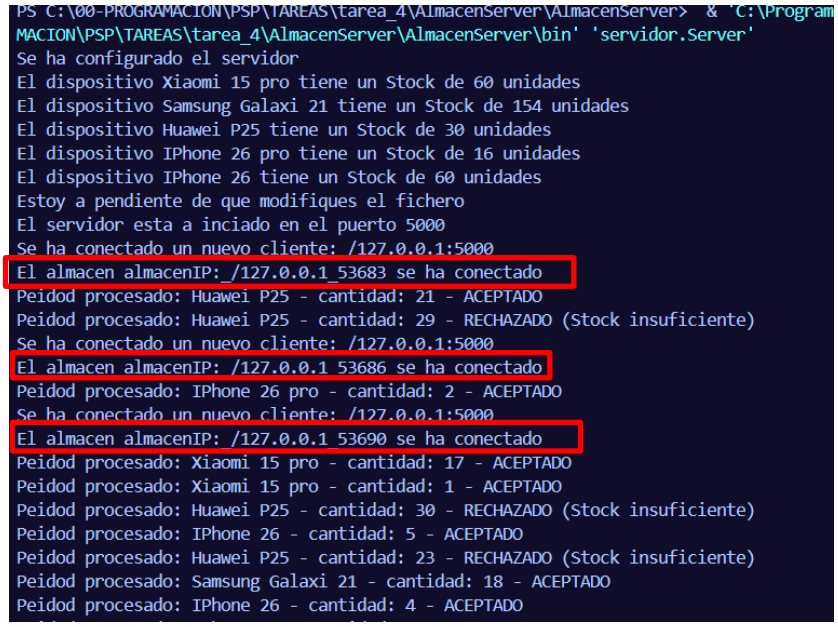
Cliente 2:



Cliente 3:



Como se puede observar en las imágenes anteriores se ha conectado correctamente los tres clientes al servidor. Veamos que muestra el servidor.



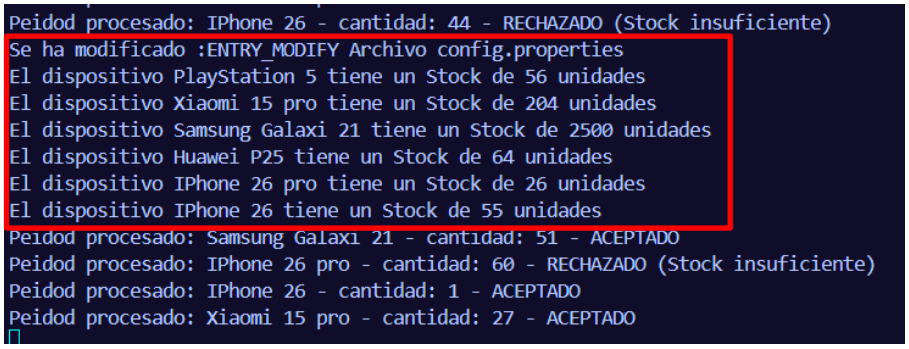
En la imagen anterior se muestran los clientes conectados.

Vamos a actualizar los datos, tanto la lista de productos, como la cantidad, para poder seguir gestionando los pedidos. Primero vamos a actualizar el servidor.

config.properties

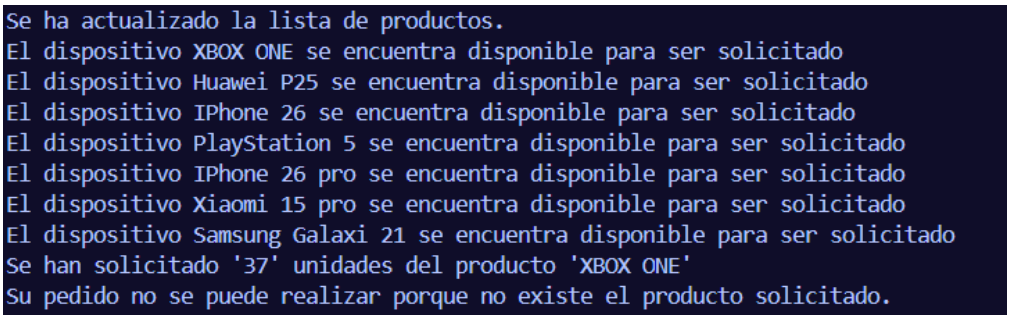
```
num_port = 5000
producto_1 = Samsung Galaxi 21,2500
producto_2 = Xiaomi 15 pro,204
producto_3 = IPhone 26,55
producto_4 = Huawei P25,64
producto_5 = IPhone 26 pro,26
producto_6 = PlayStation 5, 56
```

Como se puede observar en la imagen anterior se han agregado nuevos productos y además se han cambiado las cantidades de stock. También hay que destacar que se ha agregado un espacio entre la "," y el valor de la cantidad del producto_6, y de este modo también se comprueba si se realiza correctamente la eliminación del espacio en blanco.



Como se puede observar en la imagen anterior se ha actualizado la lista de producto y se siguen gestionando los productos que nos solicitan los otros almacenes.

Ahora, se va a actualizar la lista de productos de los clientes y vamos a aprovechar para agregar un producto que no esta en la lista del almacén principal y ver si nos devuelve el mensaje de que el producto no existe.



Como se puede observar se ha actualizado correctamente la lista de productos y además podemos observar que también no devuelve que no existe el producto solicitado.

Como conclusión de la tarea, se puede observar que las aplicaciones realizan correctamente su tarea, se realizan los pedidos tal y como se ha especificado.