

**Título del trabajo**

**Trabajo Colaborativo UD5-6**

Alumnos: David Fernández, Iker Blanco, Rodrigo López-Sastre, Jose Luis López

Asignatura: Programación Concurrente y Distribuida

**Grado en Ingeniería Informática**

**curso 2023-2024**

**Índice**

Tabla de contenido

[1. Introducción. 4](#_Toc153113482)

[2. StoreInterfaz 4](#_Toc153113483)

[2.1. Métodos de la Interfaz: 4](#_Toc153113484)

[2.2. registerNewUser 4](#_Toc153113485)

[2.3. doUserLogin 5](#_Toc153113486)

[2.4. insertProductInShop 5](#_Toc153113487)

[2.5. showProductsInShop 5](#_Toc153113488)

[2.6. deleteProductInShop 5](#_Toc153113489)

[3. StoreImpl 6](#_Toc153113490)

[3.1. Estructura del Código: 6](#_Toc153113491)

[3.2. Métodos: 6](#_Toc153113492)

[3.3. Métodos de Validación: 7](#_Toc153113493)

[4. Product 7](#_Toc153113494)

[4.1. Estructura del Código: 7](#_Toc153113495)

[5. StoreServer 8](#_Toc153113496)

[5.1. Funcionamiento: 8](#_Toc153113497)

[5.2. Método showProductsInShop 8](#_Toc153113498)

[5.3. Método insertProductInShop 9](#_Toc153113499)

[5.4. Método registerNewUser: 9](#_Toc153113500)

[6. StoreClient 10](#_Toc153113501)

[6.1. Funcionamiento: 10](#_Toc153113502)

[6.2. Menú Principal de usuarios: 10](#_Toc153113503)

[6.3. Método newUser: 10](#_Toc153113504)

[6.4. Método doUserLogin: 11](#_Toc153113505)

[6.5. Menú de Usuario Autenticado: 11](#_Toc153113506)

[6.6. Método insertProductInShop: 11](#_Toc153113507)

[6.7. Método showProductsInShop: 12](#_Toc153113508)

[6.8. Método deleteProductInShop: 12](#_Toc153113509)

[7. CustomLogFormatter 12](#_Toc153113510)

[7.1. Estructura del Código: 12](#_Toc153113511)

[8. Documentación del Código: CustomLogger 13](#_Toc153113512)

[8.1. Estructura del Código: 13](#_Toc153113513)

[9. Documentación del Código: JsonUtil 14](#_Toc153113514)

[9.1. Estructura del Código: 14](#_Toc153113515)

[10. Bibliografía 15](#_Toc153113516)

# Introducción.

En el contexto de la evolución del comercio y las cambiantes tendencias de compra, surge la necesidad de adaptarse a las demandas del mercado actual. En este escenario, nos enfrentamos al desafío de revitalizar un comercio dedicado a la venta de artículos diversos, cuya clientela tradicional ha experimentado una disminución significativa, mientras que las operaciones a través de la red han ido en constante crecimiento.

Este proyecto aborda las cuestiones solicitadas que son fundamentales, desde la autenticación de usuarios y el registro de nuevos clientes hasta la gestión eficiente de productos. La creación de una interfaz remota robusta y la interacción con un archivo JSON para el almacenamiento de datos son aspectos esenciales de este desarrollo. Además, se nos presenta la oportunidad de diseñar un proyecto de pruebas que permitirá verificar y validar la funcionalidad de cada componente, garantizando un despliegue exitoso de la tienda virtual.

# StoreInterfaz

La interfaz remota StoreInterface se ha diseñado para facilitar la interacción entre clientes y el servicio de gestión de la tienda. Esta interfaz define métodos remotos que abarcan desde la gestión de usuarios hasta la manipulación de productos en la tienda virtual. Esta interfaz establece la base para la implementación de objetos remotos que cumplirán con los requisitos especificados en el enunciado del proyecto.

## Métodos de la Interfaz:

## registerNewUser

**Descripción**: Este método permite registrar a un nuevo usuario proporcionando información como nombre de usuario, contraseña, correo electrónico, confirmación de correo electrónico y número de teléfono.

**Parámetros**:

*username*: Nombre de usuario alfanumérico.

*password*: Contraseña que cumple con ciertos criterios de complejidad.

*email*: Dirección de correo electrónico del usuario.

*checkEmail*: Confirmación de la dirección de correo electrónico.

*phone*: Número de teléfono del usuario.

**Tipo de Retorno**: boolean

**Excepciones**: RemoteException - Lanzada en caso de un problema durante la ejecución remota.

## doUserLogin

**Descripción**: Este método autentica a un usuario en el sistema verificando el nombre de usuario y la contraseña proporcionados.

**Parámetros**:

*username*: Nombre de usuario registrado.

*password*: Contraseña asociada al nombre de usuario.

**Tipo de Retorno**: boolean

**Excepciones**: RemoteException - Lanzada en caso de un problema durante la ejecución remota.

## insertProductInShop

**Descripción**: Este método permite la inserción de un nuevo producto en la tienda virtual proporcionando el nombre del producto y su precio.

**Parámetros**:

*productName*: Nombre del nuevo producto.

*price*: Precio del nuevo producto.

**Tipo de Retorno**: boolean

**Excepciones**: RemoteException - Lanzada en caso de un problema durante la ejecución remota.

## showProductsInShop

**Descripción**: Este método devuelve una lista de todos los productos disponibles en la tienda virtual.

**Tipo de Retorno**: List<Product>

**Excepciones**: RemoteException - Lanzada en caso de un problema durante la ejecución remota.

## deleteProductInShop

**Descripción**: Este método permite la eliminación de un producto de la tienda virtual mediante su identificador único.

**Parámetros**:

*productId*: Identificador único del producto a eliminar.

**Tipo de Retorno**: boolean

**Excepciones**: RemoteException - Lanzada en caso de un problema durante la ejecución remota.

# StoreImpl

La clase StoreImpl implementa la interfaz remota StoreInterface y proporciona funcionalidades clave para la gestión de la tienda virtual (usuarios, productos y persistencia de datos.). Este código aborda la autenticación de usuarios, la manipulación de productos y la persistencia de datos utilizando archivos JSON.

## Estructura del Código:

1. **Importaciones:**

Se importan las clases necesarias de java.rmi y org.json.simple para la implementación de la interfaz remota y la manipulación de archivos JSON, respectivamente.

1. **Serializable y serialVersionUID:**

La clase implementa la interfaz Serializable y proporciona un número de versión (serialVersionUID) para asegurar la consistencia de la serialización.

1. **Logger:**

Se utiliza el framework de registro personalizado (CustomLogger) para registrar eventos y mensajes importantes durante la ejecución.

1. **Constantes:**

Se definen constantes para los nombres de archivos JSON relacionados con usuarios y productos.

1. **Constructor:**

El constructor inicializa la tienda asegurándose de que los archivos JSON necesarios existan.

## Métodos:

* + **doUserLogin:**

Autentica a un usuario comparando el nombre de usuario y la contraseña proporcionados con los almacenados en el archivo users.json.

* + **registerNewUser:**

Registra a un nuevo usuario verificando y validando la información proporcionada, como nombre de usuario, contraseña, correo electrónico y número de teléfono.

* + **insertProductInShop:**

Inserta un nuevo producto en la tienda, generando un ID único y almacenando los detalles del producto en el archivo products.json.

* + **showProductsInShop:**

Muestra todos los productos de la tienda leyendo el archivo products.json y convirtiendo los objetos JSON en objetos Product.

* + **deleteProductInShop:**

Elimina un producto de la tienda según el nombre proporcionado, actualizando el archivo products.json.

* + **initializeFile:**

Método privado que inicializa un archivo si aún no existe.

## Métodos de Validación:

* + isValidUsername, isValidPassword, isValidEmail, y isValidPhone:

Métodos privados que validan los datos del usuario (nombre de usuario, contraseña, correo electrónico y número de teléfono) según ciertos criterios.

# Product

La clase Product también forma parte del sistema de gestión de una tienda virtual. Es esencial en la representación de productos en la tienda virtual. Esta clase modela los productos disponibles, con atributos como identificación única (id), nombre (name), y precio (price). Implementa métodos como equals y hashCode que facilitan la comparación y gestión de productos en el sistema. Además, implementa la interfaz Serializable para permitir la serialización de objetos.

## Estructura del Código:

1. **Serializable y serialVersionUID:**

La clase implementa la interfaz Serializable para permitir que los objetos de esta clase sean serializables, lo que es crucial para el almacenamiento y la transmisión de objetos a través de la red.

1. **Atributos:**

*id*: Identificación única del producto.

*name*: Nombre del producto.

*price*: Precio del producto.

1. **Constructor:**

La clase tiene un constructor que permite la creación de objetos Product con valores específicos para id, name, y price.

1. **Getters y Setters:**

Métodos para acceder y modificar los atributos de la clase (getId, setId, getName, setName, getPrice, setPrice).

1. **equals Method:**

Se sobrescribe el método equals para comparar objetos Product basándose en sus identificadores únicos (id). Esto facilita la comparación de productos en operaciones como la eliminación de un producto de la tienda.

1. **hashCode Method:**

Se sobrescribe el método hashCode para generar un código hash basado en el identificador único del producto (id).

1. **toString Method:**

Se sobrescribe el método toString para proporcionar una representación en cadena detallada del objeto Product, incluyendo su id, name, y price.

# StoreServer

## Funcionamiento:

## Método showProductsInShop

El método showProductsInShop en la clase StoreImpl realiza la tarea de mostrar todos los productos disponibles en la tienda. A continuación, se explica cómo funciona este método:

El método comienza registrando un mensaje en el sistema de logs para indicar que se están mostrando todos los productos.

A continuación, el método lee un archivo JSON que contiene la lista de productos. Este archivo sirve como base de datos para almacenar la información de los productos en un formato estructurado.

Se crea una lista vacía para almacenar los objetos Product. Esta lista servirá para almacenar los productos que se van a mostrar.

El método itera sobre cada objeto JSON en el arreglo de productos. Durante cada iteración, extrae los datos relevantes de cada producto, como el ID, nombre y precio.

Con estos datos, crea una instancia del objeto Product y lo añade a la lista de productos.

Al final del método, se tiene una lista completa de todos los productos, cada uno representado como un objeto Product. Esta lista se devuelve a quien haya llamado al método.

## Método insertProductInShop

El método insertProductInShop en la clase StoreImpl se encarga de insertar un nuevo producto en la tienda. A continuación, se explica cómo funciona este método:

El método comienza generando un ID único para el nuevo producto, utilizando UUID.randomUUID().toString(). Este ID es importante para identificar de manera única cada producto en la tienda.

Se registra un mensaje en el sistema de logs, indicando que se está añadiendo un nuevo producto, incluyendo detalles como el nombre y precio del producto, así como el ID generado.

A continuación, se crea un nuevo objeto Product con el ID, nombre y precio proporcionados. Este objeto representa el producto que se está añadiendo a la tienda.

El objeto Product se convierte en un objeto JSONObject. Esto implica asignar los valores de id, nombre y precio del producto a un objeto JSON para su almacenamiento.

Se lee el archivo JSON actual que contiene la lista de productos existentes. Este archivo actúa como una base de datos para almacenar los productos.

El objeto JSON del nuevo producto se añade al arreglo JSON que representa la lista de productos en la tienda.

Se escribe el arreglo JSON actualizado de vuelta al archivo Products.json. Este proceso implica actualizar la "base de datos" de productos con el nuevo producto.

Finalmente, el método verifica si la escritura del archivo fue exitosa. Si lo fue, registra un mensaje de éxito y devuelve true, indicando que el producto se añadió correctamente. Si no, registra un mensaje de error y devuelve false.

## Método registerNewUser:

El método registerNewUser en la clase StoreImpl es responsable de registrar a un nuevo usuario en la tienda. A continuación se presenta una explicación de cómo se implementa este método:

* Validación previa de la información del usuario:

Antes de proceder con el registro de los usuarios, el método llama a varios métodos (isValidUsername, isValidPassword, isValidEmail, isValidPhone) para validar la información aportada por el usuario (nombre de usuario, contraseña, correo electrónico y teléfono), verificando que cumple los criterios especificados en el enunciado del trabajo. Si falla alguna validación, imprime un mensaje de error y devuelve false, indicando que el registro no tuvo éxito.

* Registro de Usuario:

Si todas las validaciones son correctas, el método procede a registrar al nuevo usuario. Primero crea un nuevo objeto JSON (newUser) para almacenar la información del usuario. La información del usuario (nombre de usuario, contraseña, correo electrónico, teléfono) se añade al objeto newUser mediante el método put.

A continuación, se lee el archivo USERS\_JSON y lo asigna a la variable users. La información del nuevo usuario se añade a la lista existente de usuarios mediante esta variable users.

Por último, se imprime un mensaje de éxito, indicando que el usuario ha sido registrado con éxito.

# StoreClient

## Funcionamiento:

El programa comienza creando una instancia del registro RMI (Remote Method Invocation) para ubicar el servicio remoto en el servidor local en el puerto 1099.

A continuación, obtiene la referencia al objeto remoto “StoreInterface” del registro utilizando el nombre "StoreService".

Despues entra en un menú principal de registro e inicio de sesión de usuarios, cuando un usuario se registra correctamente aparecerá el menú de Usuario Autenticado para añadir, modificar o borrar productos.

## Menú Principal de usuarios:

Se presenta un bucle que representa el menú principal del cliente, donde el usuario puede seleccionar diferentes opciones:

* **Registrarse** como un nuevo usuario (opción 1). Se llama al método newUser que recopila la información del usuario y llama al método remoto registerNewUser del servicio para registrar al nuevo usuario si los valores son correctos.
* **Iniciar sesión** con un usuario existente (opción 2). Se llama al método doUserLogin que solicita al usuario que ingrese su nombre de usuario y contraseña, y luego llama al método remoto doUserLogin para realizar la autenticación.
* **Salir del programa** (opción 3).

## Método newUser:

Este método solicita al usuario que ingrese la siguiente información: nombre de usuario, contraseña, correo electrónico, confirmación de correo electrónico y número de teléfono.

A continuación, llama al método remoto registerNewUser para intentar registrar al nuevo usuario.

Imprime mensajes de registro exitoso o fallido según el resultado.

## Método doUserLogin:

Este método solicita al usuario (ya registrado) que ingrese su nombre de usuario y contraseña.

Luego, llama al método remoto doUserLogin para autenticar al usuario.

Retorna un valor booleano indicando si el inicio de sesión fue exitoso o no.

## Menú de Usuario Autenticado:

* **Insertar un Producto en la Tienda** (opción 1): Esta opción permite al usuario agregar un nuevo producto a la tienda. Al seleccionarla, se pide al usuario que introduzca detalles como el nombre y precio del producto. Estos detalles son enviados al servidor mediante una llamada al método remoto insertProductInShop. El servidor procesa la solicitud y devuelve un resultado indicando si el producto se ha añadido con éxito. El cliente muestra un mensaje correspondiente, informando al usuario sobre el resultado de la operación.
* **Mostrar todos los Productos en la Tienda** (opción 2): Al elegir esta opción, el cliente solicita la lista de todos los productos disponibles en la tienda. Esto se logra mediante la llamada al método remoto showProductsInShop. El servidor retorna una lista de productos que el cliente muestra en la pantalla. Si ocurre un error en la recuperación de los productos, se muestra un mensaje de error al usuario.
* **Borrar un Producto de la Tienda** (opción 3): Esta opción brinda al usuario la capacidad de eliminar un producto existente de la tienda. El cliente solicita al usuario que proporcione el ID del producto que desea eliminar y envía esta información al servidor a través del método remoto deleteProductInShop. El servidor intenta eliminar el producto y retorna un valor booleano para confirmar si la eliminación fue exitosa. El cliente informa al usuario sobre el resultado de la operación con un mensaje apropiado.
* **Cerrar Sesión** (opción 4): Esta opción del menú de usuario autenticado permite al usuario cerrar su sesión. Al seleccionar esta opción, el cliente termina la sesión actual y vuelve al menú principal, permitiendo a un nuevo usuario iniciar sesión o registrarse. Esta funcionalidad es esencial para mantener la seguridad de la cuenta del usuario y garantizar que el acceso a las funciones de gestión de la tienda sea restringido a usuarios autenticados.

## Método insertProductInShop:

Este método se activa cuando el usuario autenticado elige la opción de insertar un producto en la tienda (opción 1 en el menú de usuario autenticado). Solicita al usuario que ingrese el nombre y el precio del producto. Luego, llama al método remoto insertProductInShop del servicio, pasando el nombre y precio del producto como argumentos. El método remoto procesa la solicitud y devuelve un valor booleano para indicar si el producto fue añadido con éxito. El cliente, a su vez, imprime un mensaje de confirmación o error basado en la respuesta del servidor.

## Método showProductsInShop:

Esta opción del menú de usuario autenticado (opción 2) permite al usuario ver todos los productos disponibles en la tienda. Al seleccionar esta opción, el cliente llama al método remoto showProductsInShop. Este método devuelve una lista de productos, que el cliente recorre y muestra en la consola. Si ocurre algún error durante el proceso, el cliente imprime un mensaje de error.

## Método deleteProductInShop:

Al elegir la opción 3 en el menú de usuario autenticado, el cliente activa el método deleteProductInShop. Este método solicita al usuario que ingrese el ID del producto que desea eliminar. Luego, pasa este ID al método remoto deleteProductInShop del servicio. El método remoto intenta eliminar el producto y devuelve un valor booleano indicando si la operación fue exitosa. El cliente muestra un mensaje correspondiente al resultado de esta operación.

# CustomLogFormatter

La clase CustomLogFormatter define un formato personalizado para los registros de log o eventos generados por el sistema. Este formato proporciona información clave, como el nombre de la clase fuente y el mensaje de log. La clase hereda de la clase Formatter en el paquete java.util.logging, que se utiliza para formatear los registros de log. Mejora la claridad y la comprensión de los registros de log generados.

## Estructura del Código:

1. **Importaciones:**

La clase importa Formatter y LogRecord del paquete java.util.logging para implementar el formato personalizado de los registros.

1. **Clase CustomLogFormatter:**

La clase CustomLogFormatter extiende la clase Formatter para personalizar el formato de los registros de log.

1. **Método format:**

La clase implementa el método format de la clase base Formatter. Este método recibe un objeto LogRecord y devuelve una cadena formateada.

1. **Contenido del Método format:**

El método utiliza la información del LogRecord para construir una cadena formateada.

* + - record.getSourceClassName(): Obtiene el nombre de la clase fuente que generó el log.
    - record.getMessage(): Obtiene el mensaje del log.

1. **Formato de Salida:**

La cadena de salida tiene el formato: <Nombre de la Clase Fuente>: <Mensaje del Registro>\n.

# Documentación del Código: CustomLogger

La clase CustomLogger proporciona un método estático getLogger para obtener una instancia de un objeto Logger personalizado. Facilita la obtención de instancias personalizadas de Logger con configuraciones específicas. Este logger incluye un ConsoleHandler configurado con un nivel de log Level.ALL y un CustomLogFormatter personalizado para formatear los registros de log. La configuración flexible permite adaptarse a diversos niveles de detalle en los registros según las necesidades del desarrollo y la depuración. Además, se deshabilitan los handlers heredados de los padres del logger.

## Estructura del Código:

1. **Importaciones:**

Se importa Logger, ConsoleHandler, y Level del paquete java.util.logging y Class del paquete java.lang.

1. **Clase CustomLogger:**

La clase CustomLogger contiene un único método estático getLogger.

1. **Método Estático getLogger:**

Este método toma como parámetro una clase (Class<?> className) y devuelve una instancia personalizada de Logger asociada con la clase proporcionada.

1. **Creación del Logger:**

Se instancia un Logger utilizando el nombre completo de la clase (className.getName()).

1. **Verificación de Handlers:**

Se verifica si ya se han agregado handlers al logger (if (logger.getHandlers().length == 0)). Si no hay handlers, se realiza la configuración personalizada.

1. **Configuración del ConsoleHandler:**

Se crea un ConsoleHandler y se establece su nivel de log en Level.ALL.

1. **Configuración del Formatter:**

Se establece el formatter del ConsoleHandler como una instancia de CustomLogFormatter.

1. **Añadir el ConsoleHandler al Logger:**

Se agrega el ConsoleHandler al logger.

1. **Deshabilitar Handlers Padres:**

Se deshabilitan los handlers heredados de los padres (logger.setUseParentHandlers(false)).

1. **Establecer Nivel de Log del Logger:**

Se establece el nivel de log del logger en Ldevel.ALL.

1. **Retorno del Logger Configurado:**

Se retorna el logger con todas las configuraciones personalizadas.

# Documentación del Código: JsonUtil

La clase JsonUtil proporciona métodos estáticos para la lectura y escritura de archivos JSON. Utiliza la biblioteca json.simple para el manejo de JSON y ofrece funcionalidades simples para leer un archivo JSON y escribir un JSONArray en un archivo. **Conclusiones:** El manejo de excepciones mejora la robustez del código. Esta clase es útil en contextos donde se necesita interactuar con archivos JSON de manera sencilla, como en la persistencia de datos o la configuración de la aplicación.

## Estructura del Código:

1. **Importaciones:**

La clase importa las clases necesarias de json.simple para el manejo de JSON, como JSONArray, JSONParser, y ParseException.

1. **Método Estático readJsonArrayFromFile:**

Este método toma la ruta de un archivo JSON como parámetro (filePath) y devuelve un JSONArray leído desde ese archivo.

Se utiliza un JSONParser para analizar el contenido del archivo y se manejan las excepciones IOException y ParseException.

1. **Método Estático writeJsonArrayToFile:**

Este método toma un JSONArray y una ruta de archivo como parámetros y escribe el contenido del JSONArray en el archivo.

Se utiliza un FileWriter para escribir en el archivo y se maneja la excepción IOException.

1. **Manejo de Excepciones:**

En ambos métodos, las excepciones (IOException y ParseException) se manejan imprimiendo la traza de la excepción y devolviendo un nuevo JSONArray vacío en caso de error.

# Bibliografía

[1] J. M., T. A. M., K. Saira, y M. Y. Wani, «DDoS SYN Flooding; Mitigation and Prevention», *Int. J. Sci. Eng. Res.*, vol. 5, n.o 12, pp. 484-490, 2015.

[2] Rapid7, «Metasploitable 2». [En línea]. Disponible en: https://docs.rapid7.com/metasploit/metasploitable-2/. [Accedido: 29-ene-2023].

[3] D. Feng, S. He, Z. Zhou, y Y. Zhang, «A Finger Vein Feature Extraction Method Incorporating Principal Component Analysis and Locality Preserving Projections», *Sensors*, vol. 22, n.o 10, may 2022.

[4] V. Rajagopalan, «Traffic analysis of Secure Shell (SSH)», *Trisul Network Analytics*, 2017. [En línea]. Disponible en: https://www.trisul.org/blog/traffic-analysis-of-secure-shell-ssh/. [Accedido: 29-ene-2023].