Proyecto RMI: OrderSL

Jesús Guerrero Martín
Felipe Bueno Carranza
José Luis Pérez Panadero
Jaime Palomo Sivianes

ÍNDICE

1. Introducción

En este documento se va a presentar y explicar detalladamente el trabajo orientado a la utilización de objetos remotos a través de RMI.
Nuestro trabajo se va a basar en una empresa de pedidos online ficticia llamada OrderSL.

Nuestro programa se basa en una aplicación en sistemas distribuidos, donde el servidor funciona como una fábrica de 3 tipos de objetos (Usuario, Producto y Pedido) que posee 3 LinkedList para guardarlos(Usuarios, Productos y Pedidos).

Desde el cliente se podrán realizar distintas acciones para modificar dichos objetos(o lista de objetos) en el servidor y posteriormente obtener una copia del mismo objeto (o lista de objetos) actualizado para seguir modificándolo o mostrarlo por pantalla. El cliente por ejemplo podrá crear un usuario, realizar un pedido, comprobar los pedidos vinculados a la cuenta, añadir saldo, etc.

Además, los usuarios podrán ser administradores lo cual les permitirá acceder a más funciones como añadir productos nuevos al catálogo (Lista de objetos tipo Producto) o ver todos los pedidos (Lista de objetos tipo Pedido) realizados por todos los usuarios (Lista de objetos tipo Usuario).

El servidor implementará los métodos de los 3 tipos de objetos existentes, así como los métodos para crearlos y modificarlos, todos estos métodos serán llamados por el cliente cuando lo estime oportuno.

A continuación se desglosará más detalladamente el funcionamiento del programa desde el punto de vista del servidor y desde el punto de vista del cliente.

Para la realización de este proyecto y debido a la imposibilidad de reunirnos físicamente para poner en común los avances del programa hemos hecho uso de la plataforma github.com.

Enlace al proyecto: *https://github.com/JaimePalomo/RMISSDD*

2. Servidor

El servidor será el encargado de recibir las peticiones enviadas desde el cliente, procesarlas y dar una respuesta.

Este intercambio de mensajes se realizará en torno a una serie de funcionalidades implementadas en el servidor que permitirán el correcto funcionamiento del sistema.

Para el almacenamiento de datos en el servidor, éste hará uso de una serie de ficheros .dat. Estos archivos almacenarán las listas de los objetos manejados por el sistema entre los que se encuentran los productos, los usuarios y los pedidos.

El servidor por tanto, se encargará de almacenar y actualizar las listas presentes en los ficheros y enviar una copia al cliente conforme éste vaya modificando estos objetos.

El servidor estará formado por esta serie de ficheros:

- **ServidorComercio.java**: El fichero base en la ejecución del servidor. Inicia por línea de comandos funcionando de una manera similar al de las prácticas y espera que los cliente se suscriban.
- **OrderSL.java**: Contendrá la interfaz que implementará OrderSLImpl.java.
- **OrderSLImpl.java**: Implementa la interfaz OrderSL.java, es el fichero base en el funcionamiento del servidor, por lo que procedemos a explicar más detalladamente sus métodos:
    - crearUsuario: Recibe los parámetros necesarios y llama al constructor de la clase Usuario, posteriormente añade el objeto creado a la lista Usuarios y le da una copia al cliente.
    - crearProducto: Funcionamiento similar a crearUsuario, con la salvedad de que utilizamos la función Random() para añadir el id, generando un número aleatorio y comprobando que no sea negativo.
    - realizarPedido: Su funcionamiento es similar a los 2 métodos anteriores: recibe una lista de ids en forma de Array de integers,

busca una coincidencia en la lista Productos y la añade a una lista de objetos tipo Producto que será el Carrito (atributo de pedido), generará el id con el método descrito anteriormente, creará el objeto tipo Pedido llamando a su constructor y lo añadirá a la lista Pedidos.

- iniciarSesion: Esta función busca un usuario en la lista Usuarios, y comprueba que las contraseñas coinciden, devuelve al cliente el usuario o null si no encuentra la coincidencia(el usuario no existe o la sesión ha caducado).
- leerDatosBBDD: Esta función es llamada cada vez que se inicia el servidor, y lee mediante ObjectInputStream el contenido de 3 archivos .dat que contienen los datos de las 3 listas de objetos, si no encuentra algún archivo, lo creará.
- escribirDatosBBDD: Esta función limpia la base de datos, crea 3 archivos .dat y posteriormente los escribe mediante ObjectOutputStream con las 3 listas de objetos, debido a que con ObjectOutputStream sobreescribir genera un problema de cabeceras.
- existeUsuario: Comprueba si ya existe ese nombre de usuario en la lista Usuarios.
- existeProducto: Comprueba si ya existe ese nombre de producto en la lista Productos.
- obtenerProductos: Devuelve la lista Productos.
- obtenerPedidos: Devuelve la lista Pedidos.
- obtenerUsuarios: Devuelve la lista Usuarios.
- añadirSaldo: Función a la que llama el cliente para añadir saldo a un usuario. El servidor buscará el Usuario en la lista Usuarios e invocará su método añadirSaldo, posteriormente devuelve al cliente una copia del Usuario actualizada.
- cambiarNombre: Función a la que llama el cliente para cambiar el nombre de un usuario. El servidor buscará el Usuario en la lista

Usuarios e invocará su método cambiarNombre, posteriormente devuelve al cliente una copia del Usuario actualizada.
    - cambiarDireccion: Función a la que llama el cliente para cambiar la dirección de un usuario. El servidor buscará el Usuario en la lista Usuarios e invocará su método cambiarDireccion, posteriormente devuelve al cliente una copia del Usuario actualizada.
    - cambiarContraseña: Función a la que llama el cliente para cambiar la contraseña de un usuario. El servidor buscará el Usuario en la lista Usuarios e invocará su método cambiarContraseña, posteriormente devuelve al cliente una copia del Usuario actualizada.
    - obtenerSaldo: Función a la que llama el cliente para obtener el saldo de un usuario. El servidor buscará el Usuario en la lista Usuarios e invocará su método obtenerSaldo, posteriormente devuelve al cliente una copia del Usuario actualizada.
    - modificarPedido: Función a la que llama el cliente para modificar el usuario asociado a un pedido cuando éste se haya modificado. El servidor buscará una coincidencia con el nombre de usuario antiguo en la lista Pedidos y cambiará éste usuario por el nuevo mediante la función cambiarUsuario que implementa la clase Pedido.

- **Pedido.java**: El fichero que implementa la clase Pedido, tendrá como atributos un identificador, una fecha, una lista de objetos tipo Producto (Carrito), un objeto Usuario, un precio y una dirección. Presenta distintos métodos para el acceso y modificación de sus atributos.
- **Producto.java**: El fichero que implementa la clase Producto, tendrá un nombre, un identificador y un precio. Presenta distintos métodos para el acceso y modificación de sus atributos.
- **Usuario.java**: El fichero que implementa la clase Usuario, tendrá nombre, contraseña, saldo, dirección y si es admin o no. Presenta distintos métodos para el acceso y modificación de sus atributos.

3. Cliente

En el cliente se mostrarán las distintas funciones que podrá realizar un cliente. Se imprimirá un menú en el terminal donde se podrán identificar estas funcionalidades. Hay funcionalidades en el menú de las que solo se podrá hacer uso si este cliente es administrador.
Las funcionalidades disponibles para todos los clientes, tanto usuarios como administradores, son estas:

- Comprobar pedidos
- Mostrar catálogo
- Realizar pedido
- Modificar datos de usuario
- Añadir saldo

Y las funcionalidades exclusivas de los administradores serán:

- Añadir producto al catálogo
- Mostrar todos los pedidos realizados

En la carpeta del cliente se encuentran los siguientes archivos: ClienteComercio.java, menu.txt y OrderSL.txt, también están los archivos .class necesarios para la ejecución.

- **ClienteComercio.java:** en el se encuentra la implementación completa del código utilizado para el cliente.
- **menu.txt:** en este se encuentra el texto que se imprimirá por pantalla para visualizar el menú.
- **OrderSL.txt:** en este se encuentra texto decorativo para cuando se inicia sesión. El texto que incluye es “Welcome to OrderSL”.

Como hemos dicho, ClienteComercio.java es el archivo que contiene todo el código utilizado para el cliente, en este hemos desarrollado una serie de métodos para poder llevar a cabo todas las funcionalidades que antes hemos listado, además de el inicio de sesión y el registro de usuarios.
Por tanto, vamos a explicar el contenido de este archivo para comprender el funcionamiento completo del cliente.

Por tanto, ahora expliquemos todas los métodos de los que se hace uso en el código del cliente:

- **main:** el inicio de sesión o registro de usuario es lo primero que realizaremos al iniciar el cliente. Para ello, pedimos por pantalla que se

introduzca el nombre de usuario. Una vez introducido, se llama al método existeUsuario de la clase OrderSL, previamente hemos obtenido un objeto de esta clase a través del servidor.

Si el usuario existe, pedimos por pantalla que se introduzca la contraseña de dicho usuario. Una vez introducida, llamamos al método iniciarSesion de la clase OrderSL, a este le pasamos como parámetros el usuario y la contraseña introducido. Si los datos son correctos, la contraseña corresponde al usuario, nos devuelve un objeto de tipo usuario, si no son correctos el objeto de tipo usuario devuelto estará vacío, por lo que se vuelve a pedir el nombre de usuario por pantalla y se repite el proceso.

Si el usuario no existe, se procede al registro de un nuevo usuario.
Primero, pedimos la contraseña que quiere asociar a este usuario.
A continuación, pedimos que introduzca la dirección de envío del usuario y el saldo inicial del que dispondrá.
Y por último, hay que elegir si el usuario va a ser administrador.
Una vez introducidos todos estos datos, se llama al método crearUsuario de la clase OrderSL, a la que le pasamos el nombre de usuario, la contraseña, la dirección de envío, el saldo inicial y si el usuario es administrador, nos devuelve un objeto de tipo usuario. Con esto completamos el registro del usuario.

Una vez realizado el inicio de sesión o el registro de usuario correctamente.Obtenemos el catálogo, que es una lista de objetos productos, mediante la llamada al método verProductos de la clase OrderSL e imprimimos por pantalla el menú principal, mostramos este gráficamente en la siguiente figura.

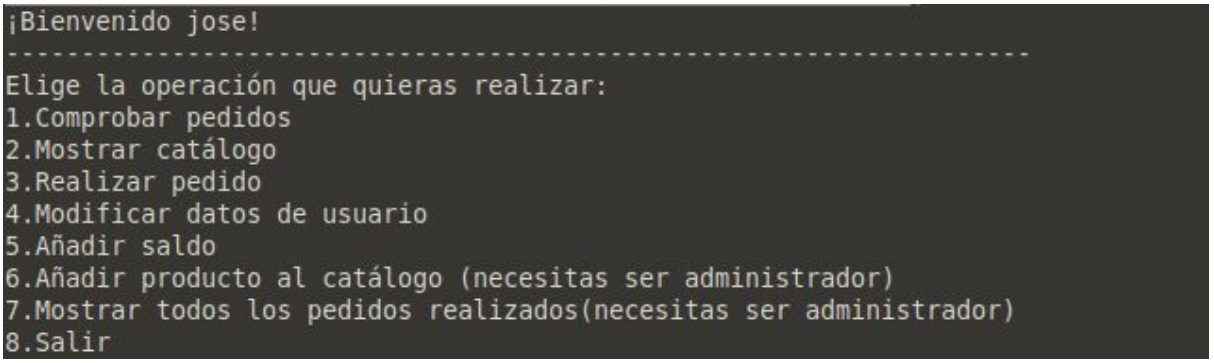

Figura 1: menú principal.

Si el usuario accede a la opción “Comprobar pedidos” se llama al método verPedidos de la claseComercio al que le pasa el objeto usuario y la referencia al objeto OrderSL, nos devuelve el número total de pedidos.
Si el número de pedidos es 0, imprimimos por pantalla que no existe ningún pedido vinculado a este usuario.

Si el usuario accede a la opción “Mostrar catálogo” imprimimos por pantalla el catálogo si el tamaño de este no es 0.

Si el usuario accede a la opción “Realizar pedido” se llama al método realizarPedido al que le pasamos como parámetros el objeto usuario, el catálogo y el objeto OrderSL, nos devuelve un objeto usuario.

Si el usuario accede a la opción “Modificar datos de usuario” llamamos al método modificarUsuario y le pasamos el nombre de usuario y el objeto OrderSL, nos devuelve un objeto usuario. Y llamamos al método modificarPedido de la clase OrderSL que modifica los pedidos del usuario por si se ha modificado el nombre.

Si el usuario accede a la opción “Añadir saldo” llamamos al método añadirSaldo al que le pasamos como parámetros el nombre de usuario y el objeto OrderSL.

Si el usuario accede a la opción “Añadir producto”, comprobamos si es administrador llamando al método isAdmin de la clase Usuario. Si es administrador llamamos al método añadirProducto y le pasamos el objeto OrderSL.

Si el usuario accede a la opción “Obtener todos los pedidos realizados”, comprobamos si es administrador llamando al método isAdmin de la clase Usuario. si es administrador llamamos al método obtenerUsuarios de la clase Usuario para obtener todos los usuarios, nos devuelve una lista de objetos Usuario. Una vez obtenidos realizamos un bucle recorriendo la lista de usuarios y obtenemos todos los pedidos realizados por cada usuario llamando a la función verPedidos y le pasamos un usuario por cada iteración.

Si el usuario accede a la opción “Salir” cerraremos la sesión. Cada vez que realizamos una acción nos devuelve de nuevo al menú, ya que nos encontramos dentro de un bucle y solo salimos de este al acceder a esta opción.

- **verPedidos:** primero llama al método obtenerPedidos de la clase OrderSL que devuelve una lista de objetos pedido con todos los pedidos. Recorremos la lista de pedidos mediante un bucle for y comprobamos los pedidos realizados por el usuario que se ha recibido como parámetro.

Además, obtenemos los productos de cada pedido llamando al método obtenerCarrito de la clase pedido y imprimimos por pantalla los datos sobre este pedido y sus productos. Y devuelve el número total de pedidos realizados.

Figura 2: ejemplo de impresión del método verPedidos.

- **realizarPedido:** imprime un menú para realizar una serie de opciones, además nos muestra el catálogo. Podemos verlo en la siguiente figura.

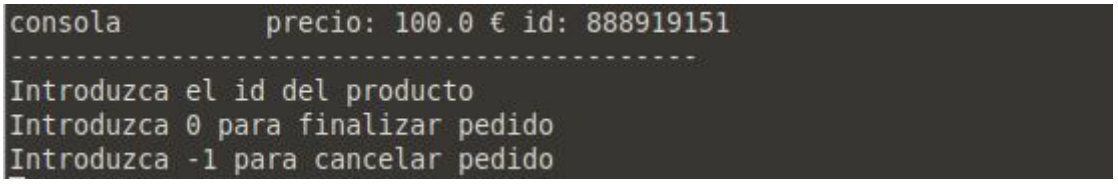

Figura 3: opciones realizarPedido

Si elegimos la opción “Introduzca el id del producto”, introducimos el id del producto que queramos adquirir se comprueba que este id coincide con un producto recorriendo con un bucle el catalogo(lista de objetos Producto). Si existe se obtiene el precio de este producto llamando al método obtenerPrecio de la clase Producto, y se almacena en un entero para posteriormente obtener el precio total del pedido y este id se guarda en una lista de enteros. Y se vuelve a mostrar el menú con las opciones.

Si elegimos la opción “Introduzca 0 para finalizar pedido”, salimos del bucle en el que nos encontramos en este menú. Antes, comprueba si la lista de enteros está vacía. Si no está vacía comprueba si el precio total del pedido es superior al saldo del usuario, si es superior a este cancela el pedido y muestra por pantalla “Saldo insuficiente”. Si no es superior, llama al método realizarPedido de la clase OrderSL al que le pasa la lista de enteros donde se han almacenado todos los id de productos introducidos y el usuario. Además, llama al método añadirSaldo de la clase OrderSL al que le pasa el nombre de usuario y el precio total del pedido para restarlo al saldo del usuario.

Si elegimos la opción “Introduzca -1 para cancelar pedido”, salimos del bucle en el que nos encontramos en este menú. Y volvemos al menú principal.

- **añadirSaldo:** imprimimos por pantalla el saldo actual y pedimos que se introduzca el saldo a añadir. Llamamos al método añadirSaldo de la clase OrderSL y le pasamos el nombre de usuario y el saldo a añadir, nos devuelve un objeto usuario. Imprime el nuevo saldo del usuario y devuelve este usuario recibido.

- **modificarUsuario:** imprime una serie de opciones para modificar al usuario. Podemos ver cuales son estas opciones en la siguiente figura.

```
1. Modificar nombre
2. Modificar contraseña
3. Modificar direccion
4. Volver al menu
```

Figura 3: opciones para modificar usuario.

Si elegimos la opción “Modificar nombre”, nos pide que introduzcamos el nuevo de usuario. Si el nombre de usuario ya existe, imprime que ese usuario ya existe y nos pide de nuevo que introduzcamos un nombre de usuario. Si no existe, llama al método cambiarNombre de la clase OrderSL y le pasa como parámetros el nombre de usuario nuevo y el antiguo, nos devuelve un objeto de tipo usuario.

Si elegimos la opción “Modificar contraseña”, nos pide que introduzcamos la nueva contraseña. Una vez introducida llama al método cambiarContraseña de la clase OrderSL al que le pasamos el nombre de usuario y la nueva contraseña, nos devuelve un objeto usuario.

Si elegimos la opción “Modificar dirección”, semejante al anterior aunque en este llama al método cambiarDireccion de la clase OrderSL al que le pasa la nueva dirección introducida y el nombre de usuario.

Si elegimos la opción “Volver al menu” volvemos al menú principal. Al realizar el resto de acciones se nos vuelve a mostrar las opciones ya que estamos dentro de un bucle y solo salimos de este al elegir esta opción.

- **añadirProducto:** nos pide por pantalla que introduzcamos el nombre del producto que queremos añadir. Llama al método existeProducto de la clase OrderSL a la que le pasamos el nombre de este producto.

Si existe nos vuelve a pedir que introduzcamos el nombre del producto. Si no existe nos pide que introduzcamos el precio del producto y llama al método crearProducto de la clase OrderSL a la que pasamos el nombre y el precio introducidos.

NOTA: Las listas de objetos(Usuarios, Productos y Pedidos) están inicialmente vacías, para la primera ejecución del programa se deberán crear algunos objetos.