

T2 - HITO INDIVIDUAL



Sergio Camino Saiz

28/01/2023



ÍNDICE

Descripción del proyecto	2
Aplicación servidor	2
Tecnologías utilizadas	5
Ejecución del programa	6
Explicación de código relevante	7
Clase ‘Servidor’	7
Clase ‘Cliente’	8
Clase ConexionDB	10
Clase LogicaServidor	11
Conclusiones	13

Descripción del proyecto

Es un programa con arquitectura cliente/servidor. El cliente consume datos de un fichero, el servidor se encarga de conectarse a la base de datos en MySQL y guardar sus datos dentro de un fichero el cual más tarde transformaremos los datos en un objeto de tipo 'coche' para que el usuario entienda mejor los datos.

Aplicación servidor

- Origen de los datos :

El cliente podrá consumir los datos directamente de la base de datos o desde un fichero. Otra opción de la que disponemos es que el usuario pueda consumir los datos mediante el uso de JPA en Java.

En mi caso he decidido que los datos estén alojados en una base de datos y que esos datos se guarden en un fichero más tarde el cual el cliente consumirá, es más complejo pero así podrían evitarse errores relacionados con la conexión entre base de datos y cliente. De esta manera también si tenemos un error sabremos específicamente en que parte y podremos solucionarlo de manera más exacta y rápida. De esta manera me resulta más fácil utilizar tecnología 'RMI' en Java la cual es la que me solicitan para el proyecto.

- Tecnologías para servicios remotos en Java accediendo a datos:

Podemos utilizar tecnologías como :

Las tecnologías que conozco para acceder a datos de forma remota son las siguientes :

	Sockets	RMI
Ventajas	<ul style="list-style-type: none"> - Buen rendimiento y poca carga de trabajo. - Flexibilidad para desarrollar aplicaciones. - Compatible con diferentes S.O. y lenguajes de programación. 	<ul style="list-style-type: none"> - Permite gestionar una aplicación sin tener que crear objetos. - Mayor rapidez y eficiencia. - Menor tiempo de carga al consumir datos remotos.
Desventajas	<ul style="list-style-type: none"> - Necesita la definición de un protocolo propio. 	<ul style="list-style-type: none"> - Mayor carga de trabajo cuando se tiene que grabar datos en disco, enviarlos por la red y recomponer los objetos de la red.

- Temática de la base de datos :

Como temática he utilizado el mundo del automovilismo. La base de datos se llama 'garaje' y dentro de ella tenemos objetos 'coche' los cuales utilizaré en mi programa. Estos coches disponen de atributos como 'marca', 'modelo' y 'motor'. Aquí específico el Script de la base de datos.

```
CREATE DATABASE garaje;

USE garaje;

CREATE TABLE Coche(
id_coche int PRIMARY KEY,
marca_coche varchar(50),
modelo_coche varchar(50),
motor_coche varchar(30)
);

INSERT INTO Coche VALUES (1,"Mazda","RX-7 Spirit R","276hp");
INSERT INTO Coche VALUES (2,"Ferrari","F40","478hp");
INSERT INTO Coche VALUES (3,"Ferrari","312T","515hp");
INSERT INTO Coche VALUES (4,"Ford","Focus","110hp");
```



Tecnologías utilizadas

Lenguaje de programación : Java.

IDE : IntelliJ.

Gestor de dependencias : Maven.

Tecnologías como : 'Registry', 'Stubs', 'Interface', 'RMI', entre otras.

Conexión a base de datos MySQL mediante 'JDBC'.

Escritura y lectura de ficheros mediante la clase 'FileWriter' y 'Scanner'.

Versiones :

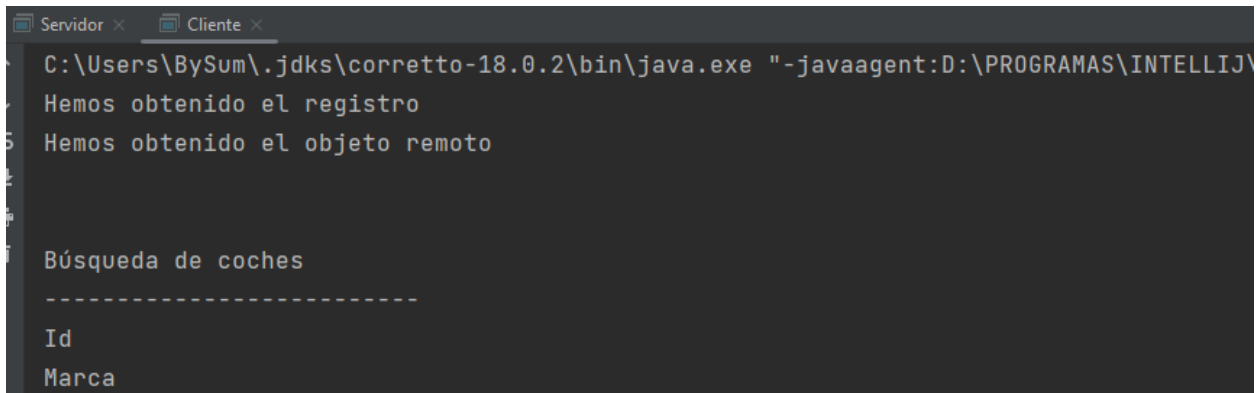
- SDK : 18.
- Módulos :
 - JDBC : 8.0.32.

Interfaces : 'Registry', 'RemoteRMI', 'Connection', 'PreparedStatement' y 'ResultSet'.

Clases : 'Scanner', 'DriverManager', 'FileWriter' y 'ArrayList'.

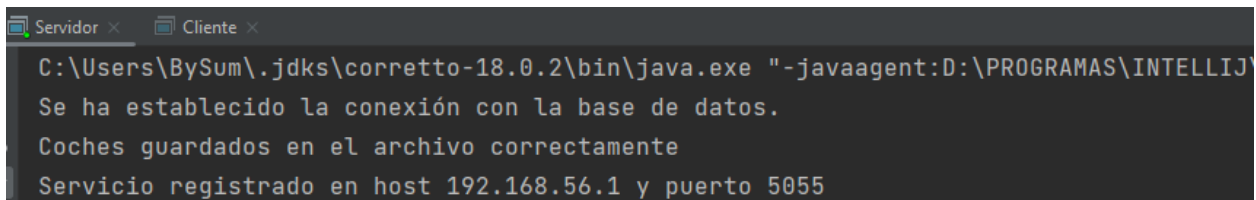
Ejecución del programa

Las siguientes capturas de pantalla demuestran cómo la aplicación ‘Servidor’ funciona correctamente (utilizando las clases e interfaces ‘LogicaServidor’, ‘ConexionDB’, ‘RemoteRMI’ y ‘Coche’) junto a la aplicación ‘Cliente’.



```
Servidor x Cliente x
C:\Users\BySum\.jdk\corretto-18.0.2\bin\java.exe "-javaagent:D:\PROGRAMAS\INTELLIJ\
Hemos obtenido el registro
Hemos obtenido el objeto remoto

Búsqueda de coches
-----
Id
Marca
```



```
Servidor x Cliente x
C:\Users\BySum\.jdk\corretto-18.0.2\bin\java.exe "-javaagent:D:\PROGRAMAS\INTELLIJ\
Se ha establecido la conexión con la base de datos.
Coches guardados en el archivo correctamente
Servicio registrado en host 192.168.56.1 y puerto 5055
```

Explicación de código relevante

Clase 'Servidor'

La clase servidor utiliza la interfaz 'Registry' para registrar los objetos remotos, que más tarde, serán consumidos por el programa 'Cliente'. Utiliza la clase 'LogicaServidor' que contiene la lógica la cual hace que el programa 'Cliente' pueda consumir los datos. Después, 'rebind' nos permite vincular un objeto remoto con el registro creado anteriormente.

```
public class Servidor {
    private static final long serialVersionUID = 1L;
    public static void main(String[] args) {
        String host;
        int puerto = 5055;
        try {
            host = InetAddress.getLocalHost().getHostAddress();
        } catch (UnknownHostException e) {
            System.out.println("No se ha podido obtener la dirección IP");
            System.out.println(e.getMessage());
            return;
        }
        try {
            Registry registro = LocateRegistry.createRegistry(puerto);
            LogicaServidor cliente = new LogicaServidor();
            registro.rebind("nuevoCoche", cliente);
            System.out.println("Servicio registrado en host " + host + " y
puerto " + puerto);
        } catch (SQLException | IOException e) {
            throw new RuntimeException(e);
        }
    }
} // CIERRA MAIN
} // CIERRA SERVIDOR
```


Clase 'Cliente'

La clase 'Cliente' es muy parecida a la clase servidor pero contiene ciertos aspectos que son diferentes. Para empezar nosotros no hacemos un 'rebind' como en el servidor, si no que hacemos 'lookup' en busca de un objeto remoto. Después de conectarse al servidor, el cliente dispone de un menú con ciertas opciones las cuales invocan a métodos remotos en el servidor.

```
public class Cliente{
    private static final long serialVersionUID = 1L;
    public static void main(String[] args) {
        Scanner lector = new Scanner(System.in);
        String host;
        try {
            host= InetAddress.getLocalHost().getHostAddress();
            Registry registro = LocateRegistry.getRegistry(host, 5055);
            System.out.println("Hemos obtenido el registro");
            RemoteRMI coche = (RemoteRMI) registro.lookup("nuevoCoche");
            System.out.println("Hemos obtenido el objeto remoto");
            System.out.println(); // Retorno de carro.
            String buscado;
            String opcion;
            do {
                escribirMenu();
                opcion = lector.nextLine().toLowerCase();
                switch (opcion) {
                    case "marca":
                        System.out.println("Escribe la marca del coche: ");
                        buscado = lector.nextLine();
                        try {
                            System.out.println(coche.buscarMarca(buscado));
                        }catch (RemoteException e){
                            System.out.println("Marca del coche no encontrada"
+ e.getMessage());
                        }
                        break;
                    case "modelo":
                        System.out.println("Escribe el modelo del coche: ");
                        buscado = lector.nextLine();
                        try{
                            System.out.println(coche.buscarModelo(buscado));
                        }catch (RemoteException e){
                            System.out.println("Modelo del coche no encontrado"
+ e.getMessage());
                        }
                        break;
                    case "motor":
```

```

        System.out.println("Escribe los cv del coche: (EJ :
'110cv')");
        buscado = lector.nextLine();
        try{
            System.out.println(coche.buscarMotor(buscado));
        }catch (RemoteException e){
            System.out.println("Motor del coche no encontrados"
+ e.getMessage());
        }
        break;
    case "salir":
        System.out.println("Saliendo del programa");
        break;
    default:
        System.out.println("Opción incorrecta");
    }
    } while (!opcion.equals("salir"));
} catch (RemoteException | NotBoundException e) {
    System.out.println(e.getMessage());
} catch (UnknownHostException e) {
    throw new RuntimeException(e);
}
lector.close();
} //CIERRA MAIN
private static void escribirMenu() {
    System.out.println();
    System.out.println("Búsqueda de coches");
    System.out.println("-----");
    System.out.println("Marca");
    System.out.println("Modelo");
    System.out.println("Motor");
    System.out.println("Salir");
    System.out.println("-----");
} //CIERRA ESCRIBIRMENU
} //CIERRA CLASE

```

Clase ConexionDB

Esta clase utiliza la dependencia 'JDBC' para conectarse a MySQL y leer los datos de la tabla. Una vez los ha leído guarda los datos en un fichero el cual consumirá el cliente.

```
public class ConexionDB {
    private static final long serialVersionUID = 1L;
    public void consultarDatos() throws SQLException, IOException {
        try {
            Connection con = null;
            String conexionbd = "jdbc:mysql://localhost:3306/garaje";
            String user = "root";
            String pass = "curso";
            try {
                con = DriverManager.getConnection(conexionbd, user, pass);
            } catch (SQLException e) {
                System.out.println("No se ha podido establecer la conexión con la base de datos.");
                System.out.println(e.getMessage());
            }
            System.out.println("Se ha establecido la conexión con la base de datos.");
            FileWriter fw = new FileWriter("coches.txt");
            PreparedStatement stmt = con.prepareStatement("SELECT * FROM coche");
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                fw.write(rs.getString(1));
                fw.write(",");
                fw.write(rs.getString(2));
                fw.write(",");
                fw.write(rs.getString(3));
                fw.write(",");
                fw.write(rs.getString(4));
                fw.write("\n");
            }
            System.out.println("Coches guardados en el archivo correctamente");
            fw.close();
            rs.close();
            stmt.close();
            con.close();
        } catch (Exception e) {
            e.getMessage();
        }
    }
} // CIERRA CONSULTARDATOS
} // CIERRA CLASE
```

Clase LogicaServidor

La clase LogicaServidor contiene toda la lógica de ambos programas. Esta clase contiene un 'ArrayList' el cual contiene los objetos que el cliente consumirá. Cuando la clase 'ConexionDB' hace su trabajo crea un fichero con los datos a consumir, la clase 'LogicaServidor' lo que hace es leer los datos del fichero y almacenarlos en el 'ArrayList' mencionado anteriormente. Esta clase contiene la conexión a la base de datos y la lectura del fichero creado, todo en su constructor para que cada vez que se llame a la clase realice estos pasos. La clase contiene los métodos creados en la interfaz 'RemoteRMI' y los desarrolla para que funcione todo correctamente.

```
public class LogicaServidor extends UnicastRemoteObject implements RemoteRMI {
    private static final long serialVersionUID = 1L;
    private static ArrayList<Coche> coches=new ArrayList<>();

    protected LogicaServidor() throws IOException, SQLException {
        ConexionDB db = new ConexionDB();
        db.consultarDatos();
        leerFichero();
    }

    public static Coche leerFichero() throws IOException {
        Scanner sc = new Scanner(new
        FileReader("coches.txt")).useDelimiter(",");
        Coche car=null;
        String[] linea;
        while(sc.hasNext()) {
            linea= sc.nextLine().split(",");
            int idCoche= Integer.parseInt(linea[0]);
            String marcaCoche=linea[1];
            String modeloCoche=linea[2];
            String motorCoche=linea[3];
            car = new Coche(idCoche,marcaCoche,modeloCoche,motorCoche);
            coches.add(car);
        }
        return car;
    }

    @Override
    public String buscarMarca(String marca) throws RemoteException {
        String resultado="";
        System.out.println(coches);
        for(Coche coche : coches){
```

```

        if(coche.getMarca_coche().contains(marca)){
            System.out.println(coche+"\n");
            resultado+=coche+"\n";
        }
    }
    return resultado;
}

@Override
public String buscarModelo(String modelo) throws RemoteException {
    String resultado="";
    for(Coche coche : coches){
        if(coche.getModelo_coche().contains(modelo)){
            resultado+=resultado+coche+"\n";
        }else{
            System.out.println("Modelo no encontrado");
        }
    }
    return resultado;
}

@Override
public String buscarMotor(String motor) throws RemoteException {
    String resultado="";
    for(Coche coche : coches){
        if(coche.getMotor_coche().contains(motor)){
            resultado+=resultado+coche+"\n";
        }else{
            System.out.println("Motor no encontrado");
        }
    }
    return resultado;
}
} // CIERRA CLASS

```

o

Conclusiones

Las conexión remota y utilizar objetos de forma remota son un método de conexión importante si queremos utilizar el menor código posible en el programa 'Servidor' y 'Cliente'.

Cuando tenemos que consumir objetos de una base de datos, utilizar una conexión de forma remota es muy buena idea debido a que el servidor no se encarga de crear objetos, tan solo se dedica a obtenerlos desde la base de datos cosa que aumenta el rendimiento de la aplicación.

Cuando el programa 'Cliente' busca un objeto mediante el registro, esto hace que aumente el rendimiento en lugar de buscar todos los objetos que contenga el servidor y así no se demora mucho a la hora de enviar la respuesta al cliente.

La parte que me ha resultado más complicada ha sido la lectura del fichero para pintarle los datos al cliente debido a que utilizó un tipo de dato 'int' para el 'id' en lugar de 'string' como sabía yo.