

AE-2. Manejo de conectores de BBDD



ÍNDICE

Integrantes.....	pág 1
Metodología de trabajo.....	pág. 2
Enunciado.....	pág. 3
Memoria.....	pág. 4 - 9

INTEGRANTES

Adalberto Martínez
Víctor Manuel Monzón
Patricia Calzado

METODOLOGÍA DE TRABAJO

En esta práctica cada uno realizo por su lado la practica y se puso en común para poder tener todas las formas posibles. Decidimos quedarnos con la mejor y sobre ella la comentamos y realizamos la memoria en conjunto.

ENUNCIADO

Requerimiento 1

Se desea hacer un CRUD completo de la entidad Coche. Esta vez no se trabajará con ningún fichero y se deberá realizar la opción de modificar coche por ID. Es muy importante usar el patrón DAO visto en clase. El menú mostrado será de la siguiente forma:

- Añadir nuevo coche (El ID lo incrementará automáticamente la base de datos)
- Borrar coche por ID
- Consulta coche por ID
- Modificar coche por ID
- Listado de coches
- Terminar el programa

Valoración: 5 puntos sobre 10

Requerimiento 2

Se pide añadir la siguiente funcionalidad.

Los coches, tendrán asociados N pasajeros en él (habrá que crear la tabla pasajeros y hacer la relación pertinente). Los pasajeros tendrán los siguientes atributos, id, nombre, edad y peso. Se añadirá la opción “gestión de pasajeros” al programa principal, dicha opción nos mostrará un submenú como el que sigue

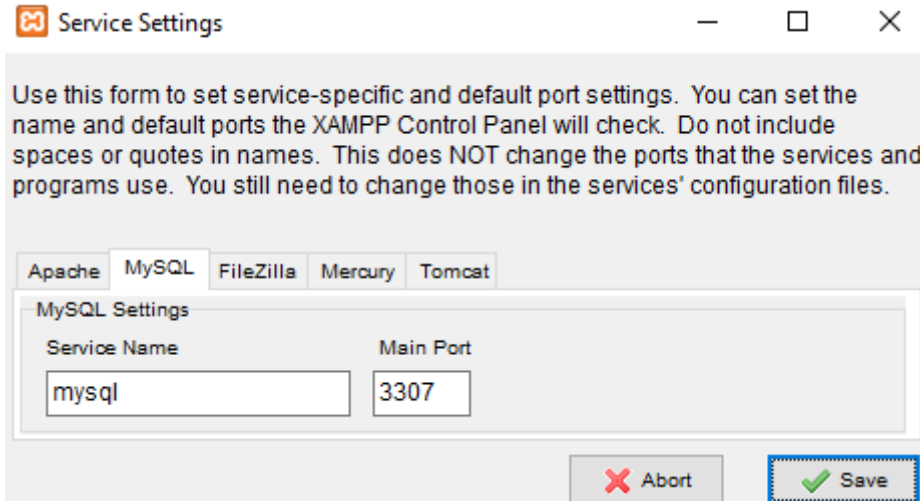
- Añadir nuevo pasajero
- Borrar pasajero por id
- Consulta pasajero por id
- Listar todos los pasajeros
- Añadir pasajero a coche, el programa nos pedirá un id de un pasajero y el id de un coche, y lo añadirá al coche a nivel de base de datos. Sería una buena opción mostrar todos los coches disponibles.
- Eliminar pasajero de un coche, el programa nos pedirá un id de un pasajero y el id de un coche, y lo eliminará del coche a nivel de base de datos. Sería una buena opción mostrar todos los coches y sus pasajeros asociados.
- Listar todos los pasajeros de un coche, el programa pedirá el id de un coche, y nos mostrará todos los pasajeros asociados a él.

Valoración: 5 puntos sobre 10

MEMORIA

Por problemas de conectividad con el Puerto por defecto se optó por el cambio de puerto principal de acceso a la BBDD de MySQL en XAMP al 3307.

Una vez cambiado los archivos y configurado correctamente el puerto principal, se procede a ejecutar en PHPmyAdmin e iniciar la BBDD con las tablas correspondientes: en este caso COCHES y PASAJEROS:



```
CREATE TABLE coches(  
id int(11) NOT NULL AUTO_INCREMENT,  
matricula (7) NOT NULL UNIQUE,  
marca varchar(20) DEFAULT NULL,  
modelo varchar(20) (3) DEFAULT NULL,  
color varchar(20) DEFAULT NULL,  
PRIMARY KEY (id)  
);
```

```
CREATE TABLE pasajeros(  
id int(11) NOT NULL AUTO_INCREMENT,  
nombre varchar(45) DEFAULT NULL,  
edad int(3) DEFAULT NULL,  
peso double DEFAULT NULL,  
cocheid int(11) DEFAULT NULL,  
PRIMARY KEY (id),  
FOREIGN KEY (cocheid) REFERENCES coches(id)  
);
```

```
ALTER TABLE `pasajeros` DROP FOREIGN KEY `pasajeros_ibfk_1`;  
ALTER TABLE `pasajeros` ADD CONSTRAINT `pasajeros_ibfk_1` FOREIGN KEY (`cocheid`)  
REFERENCES `coches`(`id`) ON DELETE SET NULL ON UPDATE CASCADE;
```




Ya que se pide que cada usuario pueda tener asociado un coche al que montar, se define en la tabla pasajero la FOREIGN KEY COCHEID que tendrá como referencia la clave primaria de la tabla Coches ID.

Además, nos adelantamos a la eliminación del coche con un pasajero asignado en este punto con la sentencia ON DELETE SET NULL, así al eliminar el coche, quedará vacío ese campo.

Para la estructura del proyecto se utilizó de la siguiente manera:

MODELO ENTIDAD: que reúne los objetos Coche y Pasajero que utilizaremos




MODELO PERSISTENCIA: Un DAO por cada objeto necesario orientado para MySQL

▼  modelo.entidad
 >  Coche.java
 >  Pasajero.java

```
public class Coche{  
  
    private int id;  
    private String matricula;  
    private String marca;  
    private String modelo;  
    private String color;
```

```
public class Pasajero {  
  
    private int id;  
    private String nombre;  
    private String apellidos;  
    private int edad;  
    private double peso;  
    private Coche coche;
```




DAOPasajero creará una instancia de DAOCoche para utilizar algunos de sus métodos y agilizar la inserción de Objetos Coche en sus atributos como se puede ver en el objeto Pasajero.

▼  modelo.persistencia
 >  DaoCocheMySQL.java
 >  DaoPasajeroMySQL.java



INTERFACES:

Estos DAO implementarán las interfaces ya creadas para facilitar el proceso de creación de los DAO para tener en cuenta todos los métodos que se deben utilizar en caso de necesitar crear DAOs para otras BBDD.

Los DAO su función será el envío de las Queries de la forma apropiada a la base de datos según el método utilizado. El control de la inserción de datos se realizará en el

▼  modelo.persistencia.interfaces
 >  DaoCoche.java
 >  DaoPasajero.java

MODELO DE NEOGOCIO:

▼  modelo.negocio
 >  Controlador.java

En este objeto se construirán los métodos que chequeen la validez de los datos introducidos y la conexión con la vista la cual ayudará a transmitir los errores surgidos en el programa durante su ejecución.

Por ejemplo:

El alta de los coches comprobará que la matrícula sea de formato válido es decir, que esté compuesta por 4 números y seguido de 3 letras.

```
public int altaCoche(Coche c){
    if(c.getMatricula().length() <= 7
        && c.getMatricula().matches("^\\d{4}?[ -]*([A-Z]{3})$")) {
        boolean alta = daoCoche.altaCoche(c);
        if(alta) {
            return 0;
        }else {
            return 1;
        }
    }else {
        return 2;
    }
}
```

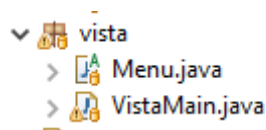
0 al montar pasajero que comprobará si el coche ya tiene 4 ocupantes.

VISTA:

```
public boolean montarPasajero(Pasajero p, Coche c){
    boolean alta = false;
    if (listarPasajerosID(c.getId()).size() < 4) {
        alta = daoPasajero.montarPasajero(p, c);
        return alta;
    }else {
        return alta;
    }
}
```

Esto será la GUI del programa (en este caso texto por consola).

En ella se ejecutará el menú y submenú de pasajeros:



MENU

1. Añadir nuevo coche.
2. Borrar coche por id.
3. Consulta coche por id.
4. Modificar coche por id
5. Listado de coches.
6. Gestionar pasajeros.
0. Salir.

6

MENU PASAJEROS

1. Añadir nuevo pasajero.
2. Borrar pasajero por id.
3. Consulta pasajero por id.
4. Listado de pasajeros.
5. Añadir pasajero a coche.
6. Eliminar pasajero de coche
7. Listar pasajeros de un coche.
0. Salir.

Método para establecer conexión con la base de datos y enviar las queries:

```
private Connection conexion;

public boolean abrirConexion(){
    String url = "jdbc:mysql://localhost:3307/bbdd";
    String usuario = "root";
    String password = "";
    try {
        conexion = DriverManager.getConnection(url,usuario,password);
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return false;
    }
    return true;
}

public boolean cerrarConexion(){
    try {
        conexion.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Añadir un coche:

```
@Override
public boolean altaCoche(Coche c) {
    if(!abrirConexion()){
        return false;
    }
    boolean alta = true;

    String query = "INSERT INTO coches(matricula, marca, modelo, color) VALUES(?,?,?,?)";
    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setString(1, c.getMatricula());
        ps.setString(2, c.getMarca());
        ps.setString(3, c.getModelo());
        ps.setString(4, c.getColor());

        int numeroFilasAfectadas = ps.executeUpdate();
        if(numeroFilasAfectadas == 0) {
            alta = false;
        }
    } catch (SQLException e) {
        System.out.println("alta -> Error al insertar: " + c);
        alta = false;
        e.printStackTrace();
    } finally{
        cerrarConexion();
    }

    return alta;
}
```

MENU

1. Añadir nuevo coche.
2. Borrar coche por id.
3. Consulta coche por id.
4. Modificar coche por id
5. Listado de coches.
6. Gestionar pasajeros.
0. Salir.

1

Introduzca matricula:

5555DDD

Introduzca marca:

Toyota

Introduzca modelo:






Corola

Introduzca color:

Blanco

Ingreso OK

.....

				id	matricula	marca	modelo	color
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	4	3333CCC	Ford	Fiesta	Azul
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	5	4444DDD	Peugeot	207	Azul
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	6	1111AAA	Seat	Panda	Rojo
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	7	5555DDD	Toyota	Corola	Blanco

Borrar coche:

```
@Override
public boolean bajaCoche(int id) {
    if(!abrirConexion()){
        return false;
    }
    boolean baja = true;

    String query = "DELETE FROM coches WHERE id=?";
    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setInt(1, id);
        int numeroFilasAfectadas = ps.executeUpdate();
        if(numeroFilasAfectadas == 0) {
            baja = false;
        }
    } catch (SQLException e) {
        System.out.println("baja -> Error al eliminar: " + id);
        baja = false;
        e.printStackTrace();
    } finally {
        cerrarConexion();
    }
    return baja;
}
```

MENÚ

1. Añadir nuevo coche.
2. Borrar coche por id.
3. Consulta coche por id.
4. Modificar coche por id.
5. Listado de coches.
6. Gestionar pasajeros.
0. Salir.

2
Introduzca ID para borrar:

7
Borrado correcto

<div><div>←T→</div><div>▼</div></div>				id	matricula	marca	modelo	color
<input type="checkbox"/>		Editar	 Copiar	 Borrar	4 3333CCC	Ford	Fiesta	Azul
<input type="checkbox"/>		Editar	 Copiar	 Borrar	5 4444DDD	Peugeot	207	Azul
<input type="checkbox"/>		Editar	 Copiar	 Borrar	6 1111AAA	Seat	Panda	Rojo

Listado y consultas:

Para el listado de coches se utilizará la query:

Introduzca ID para consultar:

5
[Coches [id=5, matricula=4444DDD, marca=Peugeot, modelo=207, color=Azul]

5
[Coches [id=4, matricula=3333CCC, marca=Ford, modelo=Fiesta, color=Azul], Coches [id=5, matricula=4444DDD, marca=Peugeot, modelo=207, color=Azul], Coches [id=6, matricula=1111AAA, marca=Seat, modelo=Panda, color=Rojo]]

Utilizan similar fórmula pero gracias al método next de la clase ResultSet se almacena cada campo y se guarda en un nuevo objeto coche. En el caso del listado se va almacenando cada coche en una lista a su vez.

```

@Override
public Coche consultarCoche(int id) {
    if(!abrirConexion()){
        return null;
    }
    Coche c = null;

    String query = "SELECT * FROM coches WHERE id=?";
    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setInt(1, id);

        ResultSet rs = ps.executeQuery();
        while(rs.next()) {
            c = new Coche();
            c.setId(rs.getInt(1));
            c.setMatricula(rs.getString(2));
            c.setMarca(rs.getString(3));
            c.setModelo(rs.getString(4));
            c.setColor(rs.getString(5));
        }
    } catch (SQLException e) {
        System.out.println("Consultar -> Error al obtener: " + id);
        e.printStackTrace();
    } finally {
        cerrarConexion();
    }
    return c;
}

@Override
public List<Coche> listarCoches() {
    if(!abrirConexion()){
        return null;
    }
    List<Coche> listaCoches = new ArrayList<>();

    String query = "select * from coches";
    try {
        PreparedStatement ps = conexion.prepareStatement(query);

        ResultSet rs = ps.executeQuery();

        while(rs.next()) {
            Coche c = new Coche();
            c.setId(rs.getInt(1));
            c.setMatricula(rs.getString(2));
            c.setMarca(rs.getString(3));
            c.setModelo(rs.getString(4));
            c.setColor(rs.getString(5));
            listaCoches.add(c);
        }
    } catch (SQLException e) {
        System.out.println("Consultar listado -> Error al obtener listado");
        e.printStackTrace();
    } finally {
        cerrarConexion();
    }
    return listaCoches;
}

```

Modificar:

```
@Override
public boolean modificarCoche(Coche c) {
    if(!abrirConexion()){
        return false;
    }
    boolean modificar = true;
    String query = "UPDATE SET matricula=?, marca=?, modelo=?, color=? WHERE id=?";

    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setString(1, c.getMatricula());
        ps.setString(2, c.getMarca());
        ps.setString(3, c.getModelo());
        ps.setString(4, c.getColor());
        ps.setInt(5, c.getId());

        int numeroFilasAfectadas = ps.executeUpdate();
        if(numeroFilasAfectadas == 0)
            modificar = false;
        else
            modificar = true;
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println("modificar -> error al modificar" + c);
        modificar = false;
        e.printStackTrace();
    } finally{
        cerrarConexion();
    }

    return modificar;
}
```

Utiliza la misma estructura que añadir un nuevo coche, lo único que la query es la apropiada con el uso de UPDATE SET y los campos adecuados para el id adecuado.

PASAJEROS:

Pasajeros lleva el mismo proceso que los mostrado anteriormente, por lo que nos centraremos en los métodos del DAO los cuales son diferentes.

MONTAR PASAJERO EN COCHE,

DEMONTAR PASAJERO DE COCHE

y

LISTAR PASAJEROS DE X COCHE.

Montar pasajero:

```
@Override
public boolean montarPasajero(Pasajero p, Coche c) {
    if(!abrirConexion()){
        return false;
    }
    boolean modificar = true;
    String query = "UPDATE pasajeros SET cocheid=? WHERE id=?";

    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setInt(1, c.getId());
        ps.setInt(2, p.getId());

        int numeroFilasAfectadas = ps.executeUpdate();
        if(numeroFilasAfectadas == 0)
            modificar = false;
        else
            modificar = true;
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println("modificar -> error al modificar" + p);
        modificar = false;
        e.printStackTrace();
    } finally{
        cerrarConexion();
    }

    return modificar;
}
```

Introduzca ID Persona

3

Pasajero [id=3, nombre=Filemón, apellidos=Pi, edad=55, peso=65.0, coche=null]

¿Este es el pasajero correcto? S/N

S

Introduzca ID Coche

5

Coches [id=5, matricula=4444DDD, marca=Peugeot, modelo=207, color=Azul]

¿Este es el coche correcto? S/N

S

		id	nombre	apellidos	edad	peso	cocheid
<input type="checkbox"/>	 Editar  Copiar  Borrar	1	Pepito	Pérez	45	80	NULL
<input type="checkbox"/>	 Editar  Copiar  Borrar	3	Filemón	Pi	55	65	5

Desmontar:

Mismo funcionamiento salvo que la query será esta:

```
---
"UPDATE pasajeros SET cocheid=NULL WHERE id=?";
```

Listar pasajero de un coche:

Introduzca ID Coche

```
5
[[Pasajero [id=1, nombre=Pepito, apellidos=Pérez, edad=45, peso=80.0, coche=Coche [id=5, matricula=4444DDD, marca=Peugeot, modelo=207, color=Azul]],
Pasajero [id=3, nombre=Filemón, apellidos=Pi, edad=55, peso=65.0, coche=Coche [id=5, matricula=4444DDD, marca=Peugeot, modelo=207, color=Azul]]]
```

En este caso será similar a listado salvo que se comprobará el id del coche en la query:

```
@Override
public List<Pasajero> listarPasajeros(int cocheid) {
    if(!abrirConexion()){
        return null;
    }
    List<Pasajero> listaPasajeros = new ArrayList<>();

    String query = "SELECT * FROM pasajeros WHERE cocheid=?";
    try {
        PreparedStatement ps = conexion.prepareStatement(query);
        ps.setInt(1, cocheid);
        ResultSet rs = ps.executeQuery();

        while(rs.next()) {
            Pasajero p = new Pasajero();
            p.setId(rs.getInt(1));
            p.setNombre(rs.getString(2));
            p.setApellidos(rs.getString(3));
            p.setEdad(rs.getInt(4));
            p.setPeso(rs.getDouble(5));
            p.setCoche(daoCoche.consultarCoche(rs.getInt(6)));
            listaPasajeros.add(p);
        }
    } catch (SQLException e) {
        System.out.println("Consultar listado -> Error al obtener listado");
        e.printStackTrace();
    } finally {
        cerrarConexion();
    }
    return listaPasajeros;
}
```

Como ya se mencionó anteriormente el coche no tendrá más de 4 ocupantes gracias al modelo.negocio.

VISTA:

La vista consta de un menú como ya hemos visto en ocasiones anteriores. En este caso se instancia el objeto de la clase Controlador (modelo negocio) que se utilizará para ejecutar los métodos y transferir a la base de datos las queries apropiadas por su comunicación con cada DAO:

```
Controlador control = new Controlador();
Scanner scan = new Scanner(System.in);

//INICIO MENÚ
boolean continuar = true;

while (continuar) {
    Menu.menu1();
    scan = new Scanner(System.in);
    int instruction = Integer.parseInt(scan.nextLine());
    switch(instruction) {
        //Opción 1: añadir
        case 1:

            Coche coche = ingresarCoche(scan);
            if (control.controlMatricula(coche.getMatricula())) {

                int altaCoche = control.altaCoche(coche);
                controlErroresCoche(altaCoche);

            }else {
                System.out.println("Matrícula repetida");
            }
            break;

        //Opción 2: Borrar
        case 2:
            System.out.println("Introduzca ID para borrar:");
            int id = Integer.parseInt(scan.nextLine());
            if(control.bajaCoche(id)) {
                System.out.println("Borrado correcto");
            }else {
                System.out.println("Error ID no encontrado");
            }
            break;

        //Opción 3: Consultar
        case 3:
            System.out.println("Introduzca ID para consultar:");
            id = Integer.parseInt(scan.nextLine());
            System.out.println(control.consultaCoche(id));
            break;
    }
}
```

Para la optimización del código se ha optado por factorizar ciertos métodos, pero aún sigue siendo un método para mostrar un menú largo y poco claro debido a la falta de tiempo:

```
public static void controlErroresCoche(int i) {
    switch (i) {
        case 0:
            System.out.println("Ingreso OK");
            break;
        case 1:
            System.out.println("Error BBDD");
            break;
        case 2:
            System.out.println("Matrícula errónea");
            break;
    }
}

public static void controlErroresPasajero(int i) {
    switch (i) {
        case 0:
            System.out.println("Ingreso OK");
            break;
        case 1:
            System.out.println("Error BBDD");
            break;
        case 2:
            System.out.println("Pasajero errónea");
            break;
    }
}

public static Coche ingresarCoche(Scanner scan) {

    System.out.println("Introduzca matricula:");
    String matricula = scan.nextLine();
    System.out.println("Introduzca marca:");
    String marca = scan.nextLine();
    System.out.println("Introduzca modelo:");
    String modelo = scan.nextLine();
    System.out.println("Introduzca color:");
    String color = scan.nextLine();

    Coche coche = new Coche(matricula, marca, modelo,color);

    return coche;
}
```