índice

Enunciado Tarea

En esta actividad, se debe crear un programa en Java que demuestre su capacidad para establecer una conexión efectiva a una base de datos utilizando el driver JDBC. Posteriormente, deberá ejecutar una consulta SQL en la base de datos y mostrar los resultados en la terminal. El programa debe conectarse a una base de datos y proporcionar un menú pequeño interactivo para realizar consultas SQL. Los resultados de las consultas se deben mostrar en la terminal o por interfaz gráfica.

Descripción del menú interactivo (Ejemplo):

1. **Opción 1: Mostrar todos los clientes**
   - Muestra la lista de todos los clientes en la base de datos.
2. **Opción 2: Mostrar pedidos de un cliente por ID**
   - Solicita al usuario el **ID del cliente** y luego muestra los pedidos asociados a ese cliente.
3. **Opción 3: Mostrar suma total de pedidos de un cliente por DNI**
   - Solicita al usuario el **DNI del cliente** y muestra la **suma total** de los pedidos que ha realizado ese cliente.
4. **Opción 4: Salir**
   - Finaliza el programa.

Detalles adicionales:

- El programa usa un ciclo `do-while` para mostrar el menú repetidamente hasta que el usuario elija salir (opción 4).
- El programa maneja la conexión con la base de datos usando **JDBC**.
- El método `statement.executeQuery` ejecuta las consultas SQL y los resultados se manejan a través de `ResultSet`.

Ejemplo de salida:
Menú:
1. Mostrar todos los clientes
2. Mostrar pedidos de un cliente por ID
3. Mostrar suma total de pedidos de un cliente por DNI
4. Salir
Elija una opción: 3
Ingrese el DNI del cliente: 12345678A
Suma total de los pedidos del cliente con DNI 12345678A: 400.50

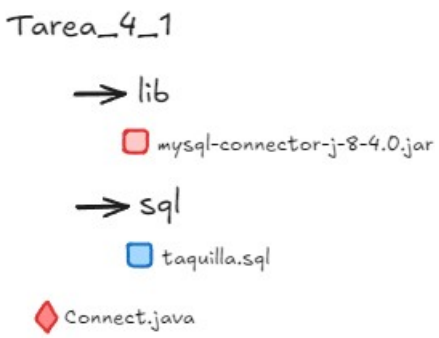
Entregables:

1. **Proyecto Java**: Debes entregar el proyecto Java con el código que establece la conexión a la base de datos y ejecuta las consultas SQL.
2. **Archivo SQL**: Incluye un archivo `.sql` que contenga la estructura de la base de datos.
3. **Memoria:** Incluye una pequeña memoria con las pruebas realizadas o explicaciones necesarias.
4. **Demostración**: Deberás mostrar al profesor que el programa se conecta correctamente a la base de datos y ejecuta las consultas

Estructura de carpetas

Para tener el ejercicio mejor organizado.

En la carpeta lib se encontrará el conector y en la carpeta sql la base de datos y en la carpeta del proyecto el .java

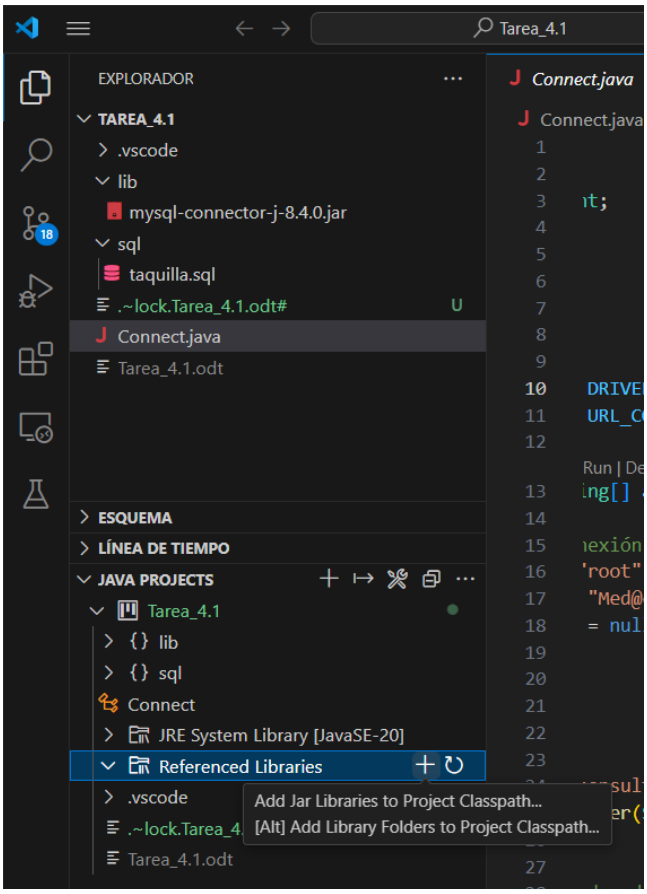


Introducir Driver de jdbc en visual studio code

En visual studio code, añadimos una vez descargado el driver y descomprimido el .jar en

JAVA PROJECTS>Referenced Libraries>+

y nos abre el explorador de archivos y ahí buscamos el .jar en la carpeta descomprimida y listo.



Crear la base de datos en Workbench con mysql

Borrar la base de datos si existe, crear la base de datos y usarla

```
DROP DATABASE IF EXISTS taquilla;

CREATE DATABASE taquilla;
USE taquilla;
```

Crear la tablas películas actores y directores

```
CREATE TABLE peliculas (

 id int auto_increment primary key,
nombre varchar(100) not null,
anio int not null
);

CREATE TABLE actores (
id int auto_increment primary key,
nombre varchar(50) not null,
apellido varchar(50) not null
);

CREATE TABLE directores (
id int auto_increment primary key,
nombre varchar(50) not null
);
```

Crear las tablas de unión mucho a muchos

```
CREATE TABLE peliculas_actores (

    id_pelicula int,
    id_actor int,

    PRIMARY KEY(id_pelicula, id_actor),
    FOREIGN KEY (id_pelicula) REFERENCES peliculas(id),
    FOREIGN KEY (id_actor) REFERENCES actores(id)
);

CREATE TABLE directores_peliculas (
    id_director int,
    id_pelicula int,

    PRIMARY KEY(id_director, id_pelicula),
    FOREIGN KEY (id_director) REFERENCES directores(id),
    FOREIGN KEY (id_pelicula) REFERENCES peliculas(id)
);
```

Hacemos registros en las tablas:

```
-- Registros peliculas

INSERT peliculas(nombre, anio) VALUES("Titanic", 1997);
INSERT peliculas(nombre, anio) VALUES("La maldición de la Perla Negra", 2003);
INSERT peliculas(nombre, anio) VALUES("Piratas del Caribe: La venganza de Salazar", 2017);
INSERT peliculas(nombre, anio) VALUES("Fast & Furious", 2001);
INSERT peliculas(nombre, anio) VALUES("Fast X", 2003);
INSERT peliculas(nombre, anio) VALUES("Crepúsculo", 2008);
INSERT peliculas(nombre, anio) VALUES("La Saga Crepúsculo: Amanecer - Parte 2", 2012);
INSERT peliculas(nombre, anio) VALUES("Iron Man 3", 2013);
INSERT peliculas(nombre, anio) VALUES("Capitán América: El Soldado de Invierno", 2014);
INSERT peliculas(nombre, anio) VALUES("Deadpool", 2016);
INSERT peliculas(nombre, anio) VALUES("Deadpool 3", 2024);
INSERT peliculas(nombre, anio) VALUES("Top Gun", 1986);
INSERT peliculas(nombre, anio) VALUES("Misión: Imposible", 1996);
INSERT peliculas(nombre, anio) VALUES("Misión: Imposible - Fallout", 2018);

-- Actores
INSERT actores(nombre, apellido) VALUES("Leonardo", "DiCaprio");
INSERT actores(nombre, apellido) VALUES("Kate", "Windslet");
INSERT actores(nombre, apellido) VALUES("Johnny Depp", "Depp");
INSERT actores(nombre, apellido) VALUES("Orlando", " Bloom");
INSERT actores(nombre, apellido) VALUES("Vin", "Diesel");
INSERT actores(nombre, apellido) VALUES("Paul", "Walker");
INSERT actores(nombre, apellido) VALUES("Kristen", "Stewart");
INSERT actores(nombre, apellido) VALUES("Robert", " Pattinson");
INSERT actores(nombre, apellido) VALUES("Robert", "Downey");
INSERT actores(nombre, apellido) VALUES("Gwyneth", "Paltrow");
INSERT actores(nombre, apellido) VALUES("Chris", "Evans");
INSERT actores(nombre, apellido) VALUES("Scarlett", "Johansson");
INSERT actores(nombre, apellido) VALUES("Ryan", "Reynolds");
INSERT actores(nombre, apellido) VALUES("Morena", "Baccarin");
INSERT actores(nombre, apellido) VALUES("Tom", "Cruise");
INSERT actores(nombre, apellido) VALUES("Kelly", "McGillis");
INSERT actores(nombre, apellido) VALUES("Tom", "Cruise");
INSERT actores(nombre, apellido) VALUES("Henry", "Cavill");

-- Directores
INSERT directores(nombre) VALUES("James Cameron");
INSERT directores(nombre) VALUES("Gore Verbinski");
INSERT directores(nombre) VALUES("Joachim Rønning");
INSERT directores(nombre) VALUES("Rob Cohen");
INSERT directores(nombre) VALUES("Louis Leterrier");
INSERT directores(nombre) VALUES("Catherine Hardwicke");
INSERT directores(nombre) VALUES("Bill Condon");
INSERT directores(nombre) VALUES("Shane Black");
INSERT directores(nombre) VALUES("Anthony y Joe Russo");
INSERT directores(nombre) VALUES("Tim Miller");
```

```
INSERT directores(nombre) VALUES("Shawn Levy");
INSERT directores(nombre) VALUES("Tony Scott");
INSERT directores(nombre) VALUES("Brian De Palma");
INSERT directores(nombre) VALUES("Christopher McQuarrie");

-- Insert peliculas actores
-- Titanic
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (1, 1);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (1, 2);
-- La maldición de la Perla Negra
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (2, 3);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (2, 4);
-- Piratas del Caribe: La venganza de Salazar
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (3, 3);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (3, 4);
-- Fast & Furious: Vin Diesel
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (4, 5);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (4, 6);
-- Fast X
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (5, 5);
 -- Crepúsculo
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (6, 7);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (6, 8);
-- La Saga Crepúsculo: Amanecer
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (7, 7);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (7, 8);
-- Iron Man 3
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (8, 9);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (8, 10);
-- Capitán América
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (9, 11);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (9, 12);
-- Deadpool
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (10, 13);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (10, 14);
-- Deadpool 3
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (11, 13);
-- Top Gun
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (12, 15);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (12, 16);
-- Misión: Imposible
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (13, 15);
INSERT INTO peliculas_actores (id_pelicula, id_actor) VALUES (14, 15);

-- Relación entre directores y
-- Titanic
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (1, 1);
-- La maldición de la Perla Negra
```

```
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (2, 2);
-- Piratas del Caribe: La venganza de Salazar
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (3, 3);
-- Fast & Furious
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (4, 4);
-- Fast X
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (5, 5);
-- Crepúsculo
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (6, 6);
-- La Saga Crepúsculo: Amanecer - Parte 2
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (7, 7);
-- Iron Man 3
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (8, 8);
-- Capitán América: El Soldado de Invierno
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (9, 9);
-- Deadpool
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (10, 10);
-- Deadpool 3
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (11, 11);
-- Top Gun
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (12, 12);
-- Misión: Imposible
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (13, 13);
-- Misión: Imposible - Fallout
INSERT INTO directores_peliculas (id_director, id_pelicula) VALUES (14, 14);
```

y guardamos el script en la carpeta sql dentro de nuestro trabajo

Driver y consultas a la base de datos .java

Variables con las credenciales de la base de datos

Estas credenciales deben ser las que ya tenemos en mysql workbench el usuario y la clave, si no no funciona. También creamos el objeto de la clase predefinida Connection para más adelante usarlo.

```java
// Variables para la conexión a la BDD
        final String USUARIO = "root";
        final String PASSWORD = "Med@c";
        Connection dbConnection = null;
```

Try-Catch, cargar controlador, conexión a BDD

Dentro del try:

-Muestro el menú al usuario:

-Le pido una opción del menú con Scanner:

-Cargamos el controlador de la base de datos: (lo introducido anteriormente hay que declararlo en código y el try-catch se usa porque si no no funciona)

-Establecemos conexión a la BDD: con el objeto declarado anteriormente la variable se llama dbConnection y como parámetros introducimos las variables creadas anteriormente para la conexión a la BDD.

```
try {
            // Crear menú
            System.out.println(
                    "Menú de consultas:\n Opción 1: Mostrar todas las películas\n Opción 2: Mostrar todas
            Scanner sc = new Scanner(System.in);
            int opcion;

            // Carga el controlador de la base de datos
            Class.forName(DRIVER);
            // Establecer conexión a la base de datos
            dbConnection = DriverManager.getConnection(URL_CONEXION, USUARIO, PASSWORD);
```

-Un bucle Do-While: En este bucle he realizado un switch y dependiendo la opción del menú que escoja el usuario se hará un caso u otro. Y Cada caso llamará a un método para que el código este mejor estructurado.

```
do {
                System.out.println("Introduzca una opción del menú (1-5): ");
                opcion = sc.nextInt();
                sc.nextLine(); // Quitar el salto de línea

                switch (opcion) {
                    case 1:
                        mostrarPeliculas(dbConnection); // pasamos la conexión como parámetro
                        break;

                    case 2:
                        mostrarPeliculasPorActor(dbConnection, sc);
                        break;

                    case 3:
                        mostrarPeliculasActoresDirectores(dbConnection);
                        break;

                    case 4:
                        contarPeliculasPorActor(dbConnection, sc);
                        break;

                    case 5:
                        System.out.println("Ha elegido salir del menú");
                        break;

                    default:
                        System.out.println("Opción no válida. Intente de nuevo.");
                        break;
                }

            } while (opcion != 5);

            sc.close();

        }
```

-El bucle se ejecuta todo el tiempo hasta que el usuario no introduzca la opción 5 que es la de salir del menú.

-Y por último en el try cerramos el Scanner.

En el catch: manejo de excepciones

```
} catch (SQLException e) {
            System.out.println("Error de SQL: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println("Controlador no encontrado: " + e.getMessage());
        } finally {
            try {
                if (dbConnection != null) {
                    dbConnection.close();
                }
            } catch (SQLException e) {
                System.out.println("Error cerrando la conexión: " + e.getMessage())
            }
        }
    }
```

El bloque finally se ejecuta siempre salten excepciones o no y ahí cerramos la conexión a la base de datos.

Métodos

Todos los métodos reciben como parámetro El objeto dbConnection de tipo Connection. Que utilizamos antes para hacer la conexión a la base de datos. Para que el método pueda acceder a la base de datos y hacer la consulta. Si no no podría y no funcionaria y así nos ahorramos también tener que crear en cada método una conexión.

-mostrarPeliculas, mostrarPeliculasActoresDirectores son casi iguales pero con diferentes consultas

recibe parámetro dbConnection de tipo Connection.

Variable con la consulta que quiero realizar

Try-catch con la conexión preparedStatement y la variable de la consulta y el executeQuery

Son métodos de XPath para hacer consultas en la base de datos y son necesarios los dos.

Y un while que no termina hasta que no haya más registros con el método .next(). Y un System.out.println() que dentro tienen el método .getString(). Obtienen la información de la base de datos y la imprime por consola.

```
private static void mostrarPeliculas(Connection dbConnection) throws SQLException { // Pasamos el pa
    String mostrarPeliculas = "SELECT * FROM peliculas";
    try (PreparedStatement statement = dbConnection.prepareStatement(mostrarPeliculas);
        ResultSet rs = statement.executeQuery()) {

        while (rs.next()) { // Recorrer todos los registros de la tabla
            System.out.println(x:"-----------------------------------");
            System.out.println("id: " + rs.getString(columnLabel:"id"));
            System.out.println("Nombre: " + rs.getString(columnLabel:"nombre"));
            System.out.println("Año: " + rs.getString(columnLabel:"anio"));
        }
```

```java
private static void mostrarPeliculasActoresDirectores(Connection dbConnection) throws SQLException {
    String actorPeliculaDirector = "SELECT d.nombre AS director, p.nombre AS pelicula, " +
                                   "a.nombre AS actor_nombre, a.apellido AS actor_apellido " +
                                   "FROM peliculas p " +
                                   "JOIN directores_peliculas dp ON p.id = dp.id_pelicula " +
                                   "JOIN directores d ON dp.id_director = d.id " +
                                   "JOIN peliculas_actores pa ON p.id = pa.id_pelicula " +
                                   "JOIN actores a ON pa.id_actor = a.id";

    try (PreparedStatement statement = dbConnection.prepareStatement(actorPeliculaDirector);
         ResultSet rs2 = statement.executeQuery()) {

        while (rs2.next()) {
            System.out.println("Director: " + rs2.getString(columnLabel:"director") +
                               ", Película: " + rs2.getString(columnLabel:"pelicula") +
                               ", Actor: " + rs2.getString(columnLabel:"actor_nombre") + " " +
                               rs2.getString(columnLabel:"actor_apellido"));
        }
    }
}
```

-mostrarPeliculasPorActor, contarPeliculasPorActor son el método prácticamente igual cambia la consulta.

Este método a diferencia del anterior pide por consola también el nombre del actor para mostrar las películas de ese actor.

Para eso introducimos por parámetro la conexión a la BDD como todos los métodos y también Scanner para no tener que crear un Scanner de nuevo en el método y reutilizar el ya creado en el main.

Y para eso usamos también try-catch con PreparedStatement esto es una clase predefinida que se encarga de evitar el SQL Injection (poner consultas maliciosas en lugar de el nombre y que ocurra algo malo en la base de datos) En la consulta en el lugar donde ira el texto que pedimos al usuario ponemos una interrogación ?

Creamos una variable booleana y la inicializamos en false si hay resultado (nombre introducido esta en la BDD) cambia a true.

Y Hacemos un while y imprimimos con getString() la tabla "pelicula" que es la de la consulta de este método que hemos realizado.

Añadimos un if en el que indicamos que si la variable booleana es falsa entonces imprima que no se encontró ese nombre.

```java
private static void mostrarPeliculasPorActor(Connection dbConnection, Scanner sc) throws SQLException {
    System.out.println(x:"Introduzca el nombre del actor: ");
    String nombreActorInput = sc.nextLine();
    String peliculasActor = "SELECT p.nombre AS pelicula FROM peliculas p " +
                            "JOIN peliculas_actores pa ON p.id = pa.id_pelicula " +
                            "JOIN actores a ON pa.id_actor = a.id WHERE a.nombre = ?";

    try (PreparedStatement preparedStatement = dbConnection.prepareStatement(peliculasActor)) { //Evitar
        preparedStatement.setString(parameterIndex:1, nombreActorInput);
        try (ResultSet rs1 = preparedStatement.executeQuery()) {
            boolean hayResultado = false;

            while (rs1.next()) { // Seguir leyendo filas hasta que no haya más
                hayResultado = true;
                System.out.println(rs1.getString(columnLabel:"pelicula"));
            }

            if (!hayResultado) {
                System.out.println("No se encontraron películas para el actor: " + nombreActorInput);
            }
        }
    }
}
```

```java
private static void contarPeliculasPorActor(Connection dbConnection, Scanner sc) throws SQLException {
    System.out.println(x:"Introduzca el nombre del actor: ");
    String nomActorInput = sc.nextLine();
    String peliculasActorCount = "SELECT COUNT(p.id) AS cantidad_peliculas FROM peliculas p " +
                                 "JOIN peliculas_actores pa ON p.id = pa.id_pelicula " +
                                 "JOIN actores a ON pa.id_actor = a.id WHERE a.nombre = ?";

    try (PreparedStatement preparedStatement = dbConnection.prepareStatement(peliculasActorCount)) {
        preparedStatement.setString(parameterIndex:1, nomActorInput);
        try (ResultSet rs3 = preparedStatement.executeQuery()) {
            if (rs3.next()) {
                int cantidadPeliculas = rs3.getInt(columnLabel:"cantidad_peliculas");
                System.out.println("Cantidad de películas del actor " + nomActorInput + ": " + cantidadPeliculas);
            } else {
                System.out.println(x:"No se encontraron resultados.");
            }
        }
    }
```