



Práctica 2.02	Acceso a Datos
JDBC con patrones DAO, Singleton y Repositorio	23-24

## Table of Contents

Introducción	.....
Base de datos	.....
Java Database Connection	.....
Clase Doctor	.....
Database Access Object (DAO)	.....
Entidad Doctor	.....
Entidad Patient	.....
Patrón Repositorio	.....
Tareas a realizar	.....
Tarea 1	.....
Tarea 2	.....
Tarea 3 (opcional)	.....
Conclusiones	.....

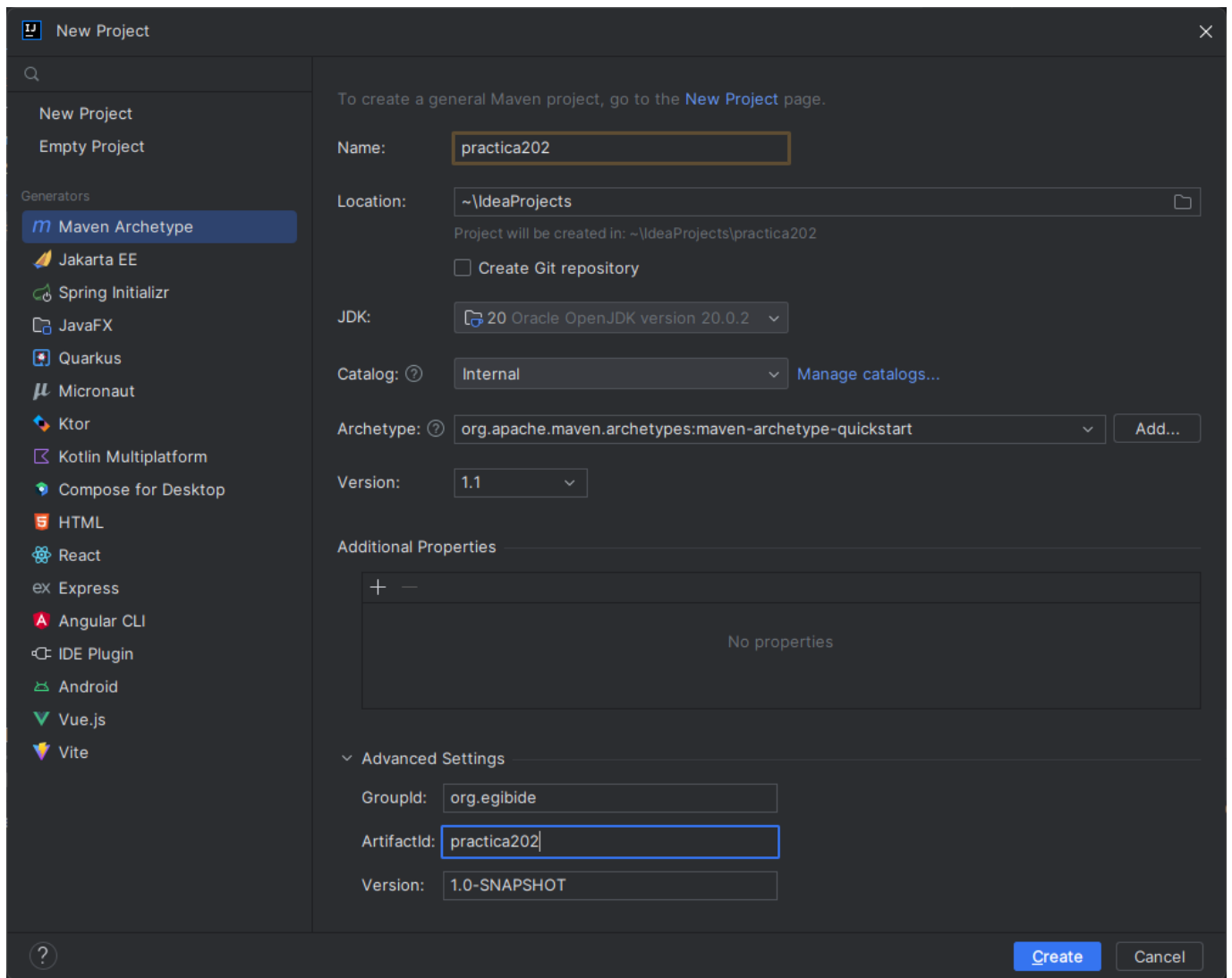
## Introducción

En esta práctica vamos a hacer un pequeño ejemplo de cómo hacer operaciones JDBC usando objetos que representan nuestras entidades de negocio mediante el patrón DAO y usaremos también una conexión que seguirá el patrón Singleton para conectarnos a la BBDD de MySQL en este caso. Además de ello, en la segunda parte implementaremos el patrón Repositorio.

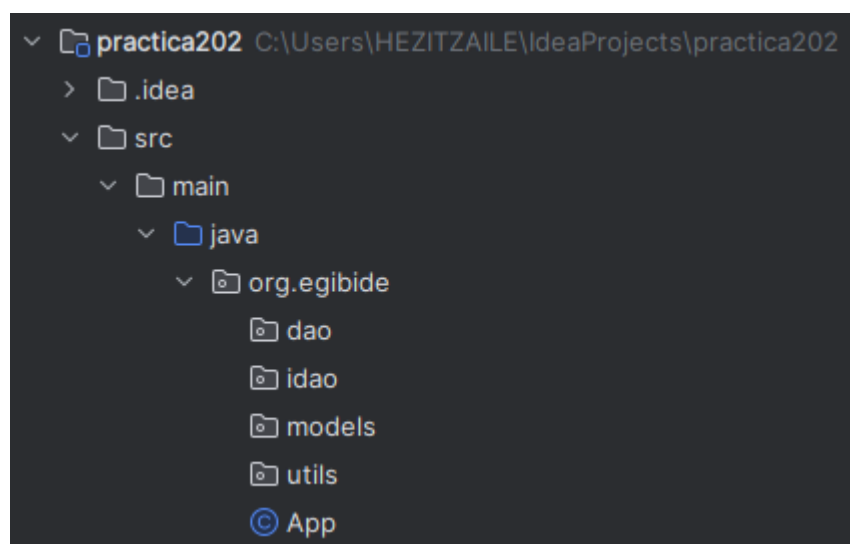
Por otro lado, sabemos que JDBC es una API de Java para hacer operaciones contra distintos motores de BBDD.

Disponéis del código empleado del que partiréis en un repositorio de [Github Classroom](#) donde además deberéis hacer la entrega. Recordad: *commit early, commit often*.

En caso de que no partamos de dicho repositorio, comenzaremos por crear un proyecto Java Maven con arquetipo `archetype-simple`, llamado `practica202`:

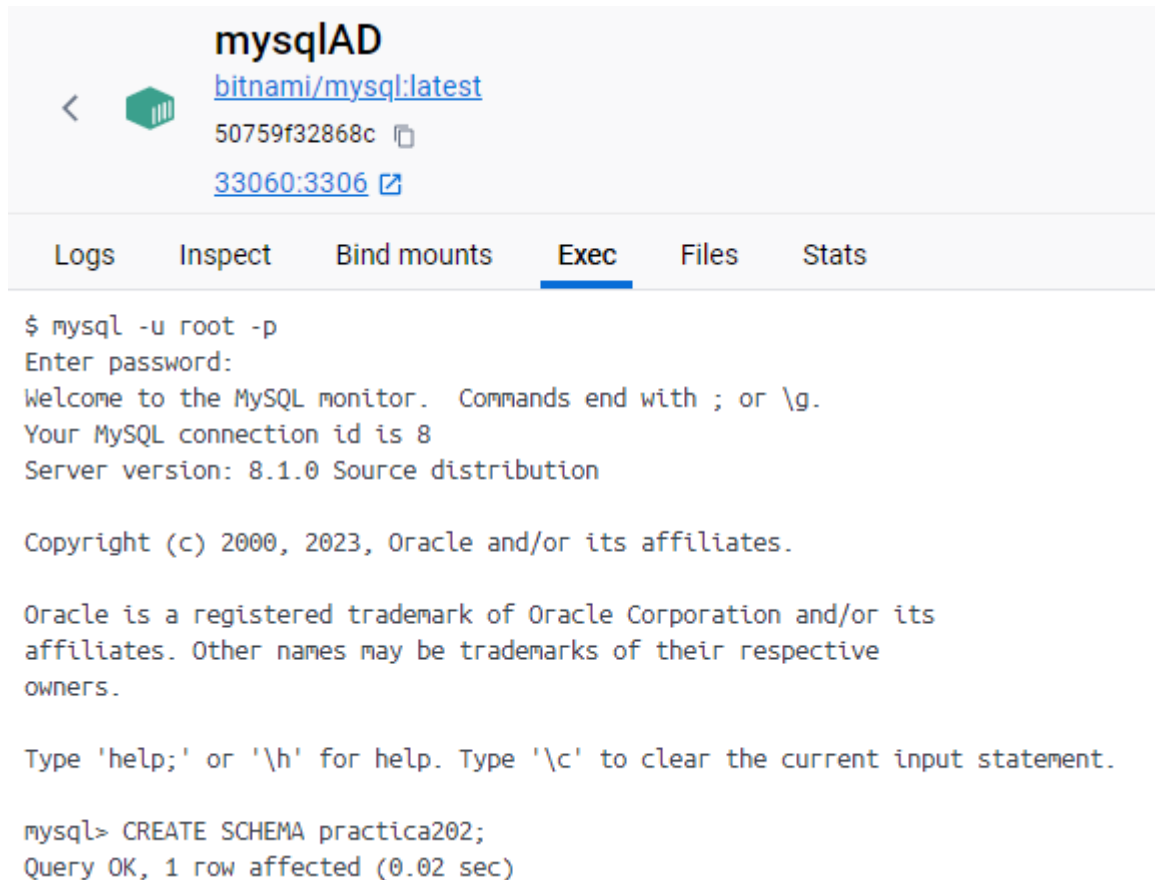


Crear la siguiente estructura de carpetas:



## Base de datos

Ahora a través del cliente que useis habitualmente crear una BBDD nueva en el servidor de MySQL, tenéis que crear una BBDD llamada `practica202` :



The screenshot shows the mysqlAD web interface. At the top, there's a header with a back arrow, a green cube icon, the text 'mysqlAD', a link to 'bitnami/mysql:latest', a container ID '50759f32868c', and a port mapping '33060:3306'. Below the header is a navigation bar with tabs: 'Logs', 'Inspect', 'Bind mounts', 'Exec' (which is selected and underlined), 'Files', and 'Stats'. The main area displays a terminal session. The session starts with a shell prompt '\$ mysql -u root -p', followed by 'Enter password:', a welcome message 'Welcome to the MySQL monitor. Commands end with ; or \g.', connection details 'Your MySQL connection id is 8' and 'Server version: 8.1.0 Source distribution', a copyright notice, and a help message. The user then enters the command 'mysql> CREATE SCHEMA practica202;', which results in 'Query OK, 1 row affected (0.02 sec)'.

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.1.0 Source distribution

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE SCHEMA practica202;
Query OK, 1 row affected (0.02 sec)
```

Tendréis que crear y rellenar las tablas según vayais desarrollando la práctica.

## Java Database Connection

Dentro de la carpeta `utils` crea la clase `DatabaseConnection` :

```

public class DatabaseConnection {
    private static DatabaseConnection instance;
    private Connection connection;
    private String url = "jdbc:mysql://localhost:33060/practica202";
    private String driver = "com.mysql.cj.jdbc.Driver";
    private String user = "root";
    private String password = "12345Abcde";

    private DatabaseConnection() throws SQLException {
        try {
            Class.forName(driver);
            this.connection = DriverManager.getConnection(url, user, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public Connection getConnection() {
        return connection;
    }

    public static DatabaseConnection getInstance() throws SQLException {
        if (instance == null) {
            instance = new DatabaseConnection();
        } else if (instance.getConnection().isClosed()) {
            instance = new DatabaseConnection();
        }

        return instance;
    }
}

```

Desde la clase ejecutable `App.java` comprobar que conectáis con vuestra BBDD.

## Clase Doctor

Ahora dentro del paquete `models` crea una clase `Doctor` que representará dicha entidad de negocio en nuestra aplicación orientada a objetos:

```
public class Doctor {  
    private int id;  
    private String name;  
    private String lastname;  
    private String dni;  
    private double salary;  
    private String speciality;  
    // Constructors, getters, setters and toString  
}
```

# Database Access Object (DAO)

---

## Entidad Doctor

---

Una vez tenemos el modelo de `Doctor`, podemos crear la interfaz `DoctorDao` y su implementación `DoctorDaoImpl`. Esta clase será la encargada de acceder a base de datos para trabajar con la entidad `Doctor`.

En concreto usaremos la clase `PreparedStatement` para ejecutar las diferentes operaciones contra la BBDD. En concreto `PreparedStatement` tiene 2 métodos importantes:

- `executeQuery()` : se utiliza para obtener datos de la BBDD.
- `executeUpdate()` : se usa para insertar, actualizar o eliminar registros.

```
public interface DoctorDao {  
    int add(Doctor doctor);  
    void delete(int id);  
    Doctor getDoctor(int id);  
    List<Doctor> getDoctors();  
    boolean update(Doctor doctor);  
}
```

En cuanto a la implementación `DoctorDaoImpl`, os muestro dos de los métodos que pertenecen a esta clase:

```

@Override
public Doctor getDoctor(int id) {
    String query = "select * from doctors where id=?";
    PreparedStatement ps = null;

    Doctor doctor = null;

    try {
        ps = DatabaseConnection.getInstance().getConnection().prepareStatement(query);
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            doctor = new Doctor(rs.getInt("id"),
                                rs.getString("name"),
                                rs.getString("lastname"),
                                rs.getString("dni"),
                                rs.getDouble("salary"),
                                rs.getString("speciality"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return doctor;
}

@Override
public boolean update(Doctor doctor) {
    if (doctorExists(doctor.getId())) {
        String query = "update doctors set name=?, lastname=?, dni=?, salary=?,
speciality=? where id=?";
        PreparedStatement ps;
        int rs = 0;

        try {
            ps =
DatabaseConnection.getInstance().getConnection().prepareStatement(query);
            ps.setString(1, doctor.getName());
            ps.setString(2, doctor.getLastname());
            ps.setString(3, doctor.getDni());
            ps.setDouble(4, doctor.getSalary());
            ps.setString(5, doctor.getSpeciality());
            ps.setInt(6, doctor.getId());

            rs = ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return (rs > 0);
    }
}

```

```
return false;
}
```

Como ejercicio hay que completar el resto de métodos de `DoctorDaoImpl` y probarlos adecuadamente.

## Entidad Patient

---

Ahora que ya tenéis el `Dao` de `Doctor` completo, hay que hacer lo mismo para el objeto de dominio `Patient`. Eso implica la creación de las siguientes clases, con sus correspondientes métodos y probarlos adecuadamente, además de la tabla correspondiente en la base de datos:

- `Patient`
- `PatientDao`
- `PatientDaoImpl`

En concreto para la clase `Patient` tendremos los siguientes atributos:

```
public Class Patient {
    private int id;
    private String name;
    private String lastname;
    private String dni;
    private int age;
    private String phone;
    private String disease;
    // Constructors, getters, setters and toString
}
```

## Patrón Repositorio

---

A nivel de BBDD, tenéis que crear un campo en la tabla `patients` llamado `doctor_id`. Este campo es FK de la tabla `doctors`.

Es decir entre las dos tablas existe una relación 1-N entre pacientes y doctores: un paciente es atendido por un doctor; y un doctor atiende muchos pacientes.

Esa relación 1-N se establece en una BBDD relacional a través del campo `doctor_id` indicado. Sin embargo, dentro de una aplicación OO (orientada a objetos), las relaciones entre clases se

establecen a través de los atributos de las mismas usando distintas estructuras de datos y definiendo esas relaciones.

En concreto, si para un objeto paciente yo quiero obtener el doctor que lo ha atendido, tendré que añadir un atributo de tipo `Doctor` a la clase `Patient` :

```
private Doctor doctor;  
//add the getter and the setter
```

Además de lo anterior, vamos a meter un nuevo método en la interfaz `DoctorDao` :

```
public Doctor getDoctorByPatientId(int patient_id);
```

Implementad dicho método, que será encargado de devolver un objeto de tipo `Doctor` a partir de id del paciente atendido. Dicha implementación irá en la clase `DoctorDaoImpl` . Por tanto, para este caso tendréis que unir la información de la tabla `patients` con la tabla `doctors` . Probarlo convenientemente.

Finalmente, para entender y terminar de implementar el patrón repositorio, cread un nuevo paquete llamado `repositories` . Dentro crear una interfaz llamada `PatientRepository` como sigue:

```
public interface PatientRepository {  
    Patient getPatient(int id);  
    //void add(Patient patient);  
    //void update(Patient patient);  
    //void remove(Patient patient);  
}
```

Añadid también una clase con su implementación:



```
public class PatientRepositoryImpl implements PatientRepository {

    private PatientDaoImpl patientDao = new PatientDaoImpl();
    private DoctorDaoImpl doctorDao = new DoctorDaoImpl();

    @Override
    public Patient getPatient(int id) {
        Patient patient = patientDao.getPatient(id);
        Doctor doctor = doctorDao.getDoctorByPatientId(patient.getId());
        patient.setDoctor(doctor);
        return patient;
    }
}
```

Ahora, como veis, a través del repositorio, somos capaces de obtener un objeto `Patient` con el `Doctor` que lo atendió. Es decir, estamos agregando los dos sets de información en el mismo objeto.

Probar convenientemente el método que acabamos de crear.

## Tareas a realizar

### Tarea 1

Ahora vamos a crear la relación inversa, es decir si antes para un objeto `Patient` le asociábamos un atributo con el `Doctor` que lo atendía, estaría bien si en la clase `Doctor` creáramos un atributo con la lista de pacientes que ha atendido:

```
private List<Patient> attendedPatients = null;
//add the corresponding getter/setter/toString
```

Después crearemos una interfaz `DoctorRepository` con el siguiente método, que devolverá todo el objeto `Doctor` completo incluyendo la lista de pacientes atendidos:

```
public interface DoctorRepository {
    Doctor getDoctor(int doctor_id);
}
```

Para poder realizar la implementación del anterior repositorio, necesitaremos primero añadir un nuevo método en la clase `PatientDao`:

```
public List<Patient> getPatientsByDoctorId(int doctor_id);
```

Realizad primero la implementación del método `getPatientsByDoctorId` y después completad la implementación del `DoctorRepository`. Probadlo convenientemente.

## Tarea 2

---

Dentro de la interfaz del repositorio de pacientes, crea un método con la siguiente firma:

```
public boolean isPatientAttendedByDoctor(int patient_id, int doctor_id);
```

Implementadlo dentro del repositorio correspondiente. El método devolverá si un paciente determinado ha sido atendido por un determinado doctor. No tienes que crear ningún método en los DAO. Con los métodos que tienes ahora mismo es suficiente.

## Tarea 3 (opcional)

---

Después de subir todo lo realizado hasta ahora a vuestro repositorio, cread una nueva rama de todo el código llamada `sqlite`, en la que tendréis que crear una nueva implementación de la interfaz `DatabaseConnection` para que la aplicación trabaje con dicho sistema de bases de datos.

## Conclusiones

---

Ahora que hemos implementado los patrones DAO y de repositorio, podríamos concluir lo siguiente:

- DAO es una abstracción de los datos que persisten en BBDD. Sin embarlo, los repositorios son una abstracción de una colección de objetos (de DAOs).
- DAO es un concepto a más bajo nivel, más cercano a las BBDD. Y los repositorios son más cercanos a los objetos de dominio.
- DAO trabaja mapeando datos a objetos, ocultando las queries. Y los respositorios, ocultan la complejidad de manejar relaciones entre objetos de dominio.
- Se podría decir que los repositorios son manejadores de DAOs y los usan para acceder a través de ellos a la persistencia de los datos.