Para el siguiente programa de JDBC necesito que me confecciones un texto que explique en profundidad:

- Las funciones que ejecuta cada módulo

- La relación funcional entre los distintos módulos que lo componen  
  
// AdministradorDeConexiones.java

// esta clase se encarga de administrar las conexiones con la base de datos

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.DriverManager;

public class AdministradorDeConexiones {

// atributo

// obtener una conexión, hago el método estático porque no quiero crear una instancia para usarlo

public static Connection conectar() {

// logica para obtener la conexión

String url = "jdbc:mysql://localhost:3306/cac-24100?serverTimeZone=UTC&userSSL=false";

String user = "root";

String password = "Hammil01";

// como usamos JDBC tenemos que decirle que driver vamos a usar

String driver = "com.mysql.cj.jdbc.Driver";

// tengo que crear una instancia de Connection para poder hacer la consulta

Connection connection = null;

// como puede dar un error entonces try/catch

try {

// si va todo bien

// forName es un metodo estatico: que no necesito una instancia para usar dicho

// metodo

Class.forName(driver);

// me conecto

connection = DriverManager.getConnection(url, user, password);

} catch (Exception e) {

// si falla

System.err.println("No se ha podido conectar al la db:" + e.getMessage());

}

return connection;

}

public static void desconectar(Connection connection) {

//controlo el posible error del metodo close

try{

connection.close();

}catch(Exception e) {

System.err.println(e);

}

}

}

// CrearTurnosController.java

package ar.com.codo24100.practica;

public class CrearTurnosController {

public static void main(String[] args) {

// Atributos

TurnoService turnoService = new TurnoService();

Turno nuevoTurno = new Turno("Test 56");

// Metodos

// turnoService.crearTurno(nuevoTurno);

System.out.println(turnoService.listarTurnos());

}

}

// Turno.java

package ar.com.codo24100.practica;

import java.time.LocalDate;

public class Turno {

//atributos

private Long id;

private LocalDate fecha;

private Long tipo;

private String motivo;

//constructor

public Turno(String motivo) {

this.motivo = motivo;

}

//metodos (get/set u otros)

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public LocalDate getFecha() {

return fecha;

}

public void setFecha(LocalDate fecha) {

this.fecha = fecha;

}

public Long getTipo() {

return tipo;

}

public void setTipo(Long tipo) {

this.tipo = tipo;

}

public String getMotivo() {

return motivo;

}

public void setMotivo(String motivo) {

this.motivo = motivo;

}

public String toString() {

return "Turno [id=" + id + ", fecha=" + fecha + ", tipo=" + tipo + ", motivo=" + motivo + "]";

}

}

// TurnoDAO.java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public interface TurnoDAO {

public void create(Turno turno);

public ArrayList<Turno> findAll();

}

// TurnoDaoMysqlImpl.java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

public class TurnoDaoMysqlImpl implements TurnoDAO{

public void create(Turno turno) {

String sql = "INSERT INTO TURNOS (MOTIVO) VALUES(?)";

Connection connection = AdministradorDeConexiones.conectar();

try {

PreparedStatement pst = connection.prepareStatement(sql);

// el signo de pregunta indica que es un parametro, el primer signo usa el parámetro del primer indice

// el segundo signo usa el parámetro del segundo indice y así sucesivamente

pst.setString(1,turno.getMotivo());

//como es un insert

pst.executeUpdate();

}catch(Exception e) {

System.err.println(e);

}finally{

AdministradorDeConexiones.desconectar(connection);

}

}

@Override

public ArrayList<Turno> findAll() {

String sql = "SELECT \* FROM TURNOS";

Connection connection = AdministradorDeConexiones.conectar();

ArrayList<Turno> turnos = new ArrayList<>();

try {

PreparedStatement pst = connection.prepareStatement(sql);

//como es un insert

ResultSet res = pst.executeQuery();

while(res.next()) {//3 veces

Long id = res.getLong(1);

String motivo = res.getString(2);

//res.getString(3)//tph... como extraer de TimeStamp a un LocalDate

Turno turno = new Turno(motivo);

turno.setId(id);

turnos.add(turno);

}

}catch(Exception e) {

System.err.println(e);

}finally{

AdministradorDeConexiones.desconectar(connection);

}

return turnos;

}

}

// TurnoService.java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public class TurnoService {

private TurnoDAO dao;//null

// tengo que instanciar la interface para poder usar los metodos.

public TurnoService () {

//Interface nombre = new ClaseQueImplementaLaInterface();

this.dao = new TurnoDaoMysqlImpl();

}

public void crearTurno(Turno turno) {

this.dao.create(turno);

}

public ArrayList<Turno> listarTurnos() {

return this.dao.findAll();

}

}

De Chat GPT  
### Explicación detallada del programa JDBC

El programa JDBC que has proporcionado se compone de varios módulos interrelacionados. Cada uno de estos módulos cumple una función específica dentro del flujo general del programa. A continuación, se describe en profundidad la función de cada módulo y cómo se relacionan entre sí.

#### 1. \*\*AdministradorDeConexiones.java\*\*

- \*\*Función:\*\*

- Esta clase es responsable de gestionar las conexiones con la base de datos. Proporciona métodos para conectar y desconectar de la base de datos.

- \*\*Métodos:\*\*

- `public static Connection conectar()`: Este método establece una conexión con la base de datos utilizando JDBC. Utiliza la clase `DriverManager` para obtener una conexión a la base de datos especificada.

- `public static void desconectar(Connection connection)`: Este método cierra la conexión a la base de datos si está abierta.

- \*\*Relación funcional:\*\*

- Este módulo es fundamental ya que proporciona la conexión necesaria para todas las operaciones de la base de datos realizadas por otros módulos. Cualquier interacción con la base de datos requiere una conexión establecida a través de este módulo.

#### 2. \*\*Turno.java\*\*

- \*\*Función:\*\*

- Esta clase representa el modelo de datos de un "Turno". Contiene atributos como `id`, `fecha`, `tipo`, y `motivo`, junto con sus respectivos métodos getters y setters.

- \*\*Constructor:\*\*

- `public Turno(String motivo)`: Inicializa un objeto `Turno` con un motivo específico.

- \*\*Métodos:\*\*

- Métodos getter y setter para cada atributo.

- `public String toString()`: Sobrescribe el método `toString` para proporcionar una representación de cadena del objeto `Turno`.

- \*\*Relación funcional:\*\*

- Esta clase se utiliza para crear y manipular objetos `Turno` que serán almacenados y recuperados de la base de datos.

#### 3. \*\*TurnoDAO.java\*\*

- \*\*Función:\*\*

- Esta es una interfaz que define los métodos para las operaciones de la base de datos relacionadas con los objetos `Turno`.

- \*\*Métodos:\*\*

- `public void create(Turno turno)`: Define el método para crear un nuevo `Turno` en la base de datos.

- `public ArrayList<Turno> findAll()`: Define el método para recuperar todos los `Turnos` de la base de datos.

- \*\*Relación funcional:\*\*

- Proporciona una abstracción para las operaciones de la base de datos, lo que permite una implementación específica en clases que implementan esta interfaz.

#### 4. \*\*TurnoDaoMysqlImpl.java\*\*

- \*\*Función:\*\*

- Esta clase implementa la interfaz `TurnoDAO` y proporciona la implementación específica para las operaciones de la base de datos utilizando MySQL.

- \*\*Métodos:\*\*

- `public void create(Turno turno)`: Implementa la lógica para insertar un nuevo `Turno` en la base de datos.

- `public ArrayList<Turno> findAll()`: Implementa la lógica para recuperar todos los `Turnos` de la base de datos.

- \*\*Relación funcional:\*\*

- Este módulo interactúa directamente con la base de datos a través de consultas SQL y utiliza `AdministradorDeConexiones` para gestionar las conexiones.

#### 5. \*\*TurnoService.java\*\*

- \*\*Función:\*\*

- Esta clase actúa como un servicio que utiliza la interfaz `TurnoDAO` para realizar operaciones relacionadas con `Turno`.

- \*\*Constructor:\*\*

- `public TurnoService()`: Inicializa el servicio con una implementación específica de `TurnoDAO`.

- \*\*Métodos:\*\*

- `public void crearTurno(Turno turno)`: Utiliza el método `create` de `TurnoDAO` para crear un nuevo `Turno`.

- `public ArrayList<Turno> listarTurnos()`: Utiliza el método `findAll` de `TurnoDAO` para listar todos los `Turnos`.

- \*\*Relación funcional:\*\*

- Este módulo sirve como intermediario entre los controladores (o cualquier otro cliente) y la capa de acceso a datos, proporcionando una capa de abstracción y facilitando la separación de preocupaciones.

#### 6. \*\*CrearTurnosController.java\*\*

- \*\*Función:\*\*

- Este es el módulo de control que gestiona la creación y la visualización de `Turnos`. Actúa como el punto de entrada del programa.

- \*\*Métodos:\*\*

- `public static void main(String[] args)`: Este método principal crea una instancia de `TurnoService` y un nuevo `Turno`, luego utiliza los métodos de `TurnoService` para realizar operaciones.

- \*\*Relación funcional:\*\*

- Este módulo interactúa con `TurnoService` para realizar operaciones específicas, como la creación y la visualización de `Turnos`. Sirve como un ejemplo de cómo otros componentes o interfaces de usuario podrían interactuar con el sistema.

### Relación funcional entre los módulos

- \*\*AdministradorDeConexiones\*\* proporciona la infraestructura básica para conectarse y desconectarse de la base de datos, lo que es esencial para todas las operaciones de datos en los módulos `TurnoDaoMysqlImpl` y `TurnoService`.

- \*\*TurnoDaoMysqlImpl\*\* utiliza \*\*AdministradorDeConexiones\*\* para obtener conexiones y ejecutar consultas SQL. Implementa la interfaz `TurnoDAO`, definiendo la lógica específica para interactuar con la base de datos.

- \*\*TurnoService\*\* actúa como una capa intermedia que utiliza `TurnoDaoMysqlImpl` (a través de la interfaz `TurnoDAO`) para realizar operaciones de base de datos. Proporciona métodos de alto nivel que pueden ser utilizados por controladores u otros módulos.

- \*\*CrearTurnosController\*\* es el punto de entrada del programa que interactúa con `TurnoService` para realizar acciones específicas, como la creación y la visualización de `Turnos`.

- \*\*Turno\*\* es el modelo de datos que se manipula y persiste en la base de datos. Los objetos `Turno` se crean en `CrearTurnosController`, se manipulan en `TurnoService` y se persisten o recuperan en `TurnoDaoMysqlImpl`.

Esta arquitectura modularizada permite una clara separación de preocupaciones, facilita el mantenimiento y la escalabilidad del sistema, y proporciona una estructura robusta para el manejo de datos utilizando JDBC en Java.

¿Hay alguno de estos módulos que podría obviarse y usar el código de ese módulo en otro?

Sí, algunos de los módulos en el programa pueden ser combinados para simplificar el código, especialmente si la aplicación es pequeña y no se anticipa que crezca significativamente en complejidad. A continuación, se analiza cómo podría hacerse esto y las implicaciones de dicha combinación.

### 1. \*\*Combinación de `TurnoDAO` y `TurnoDaoMysqlImpl`\*\*

Dado que `TurnoDaoMysqlImpl` es la única implementación de `TurnoDAO`, se podría combinar la interfaz y su implementación en una sola clase. Esto simplifica la estructura eliminando la necesidad de una interfaz separada cuando no se prevén múltiples implementaciones.

```java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

public class TurnoDao {

public void create(Turno turno) {

String sql = "INSERT INTO TURNOS (MOTIVO) VALUES(?)";

Connection connection = AdministradorDeConexiones.conectar();

try {

PreparedStatement pst = connection.prepareStatement(sql);

pst.setString(1, turno.getMotivo());

pst.executeUpdate();

} catch(Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexiones.desconectar(connection);

}

}

public ArrayList<Turno> findAll() {

String sql = "SELECT \* FROM TURNOS";

Connection connection = AdministradorDeConexiones.conectar();

ArrayList<Turno> turnos = new ArrayList<>();

try {

PreparedStatement pst = connection.prepareStatement(sql);

ResultSet res = pst.executeQuery();

while(res.next()) {

Long id = res.getLong(1);

String motivo = res.getString(2);

Turno turno = new Turno(motivo);

turno.setId(id);

turnos.add(turno);

}

} catch(Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexiones.desconectar(connection);

}

return turnos;

}

}

```

### 2. \*\*Integración de `TurnoService` en `CrearTurnosController`\*\*

Si la aplicación es simple y no se anticipa la necesidad de una lógica de negocio compleja, `TurnoService` puede ser combinado directamente en `CrearTurnosController`. Esto reducirá la cantidad de clases, pero a costa de mezclar la lógica de negocio con la lógica de presentación.

```java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public class CrearTurnosController {

public static void main(String[] args) {

TurnoDao turnoDao = new TurnoDao();

Turno nuevoTurno = new Turno("Test 56");

// Crear un nuevo turno

turnoDao.create(nuevoTurno);

// Listar todos los turnos

ArrayList<Turno> turnos = turnoDao.findAll();

System.out.println(turnos);

}

}

```

### 3. \*\*Reevaluación de `AdministradorDeConexiones`\*\*

Aunque `AdministradorDeConexiones` es esencial para gestionar las conexiones, su código es bastante pequeño y se podría integrar directamente en `TurnoDao` para simplificar aún más la estructura. Sin embargo, es recomendable mantenerla separada para una mejor separación de responsabilidades y facilidad de mantenimiento.

### Beneficios y Desventajas de la Simplificación

\*\*Beneficios:\*\*

- Menor cantidad de archivos y clases, lo que puede hacer que el proyecto sea más fácil de entender y navegar para proyectos pequeños.

- Menor complejidad inicial.

\*\*Desventajas:\*\*

- Menos flexibilidad para escalar o cambiar la implementación en el futuro.

- Potencial para mezclar diferentes responsabilidades en una sola clase, lo que puede llevar a un código más difícil de mantener.

- Pérdida de la separación de preocupaciones, lo que puede complicar la depuración y el mantenimiento del código.

En resumen, si tu aplicación es pequeña y no se espera que crezca significativamente en complejidad, puedes simplificar la estructura combinando algunos de los módulos. Sin embargo, es importante equilibrar la simplicidad con la necesidad de mantener una separación clara de responsabilidades para facilitar el mantenimiento y la escalabilidad en el futuro.

¿Por que tengo un módulo turnoservice.java? ¿Se usa siempre en JDBC?

El módulo `TurnoService.java` en tu programa se utiliza para implementar una capa de servicio que actúa como intermediario entre el controlador (`CrearTurnosController`) y la capa de acceso a datos (`TurnoDao`). Esta capa de servicio no es una obligación en JDBC, pero es una práctica común en el desarrollo de software para seguir el principio de separación de preocupaciones (SoC) y para organizar mejor el código.

### Función de `TurnoService.java`

El propósito de `TurnoService` es encapsular la lógica de negocio. Aquí están sus funciones y ventajas en detalle:

1. \*\*Encapsulación de la Lógica de Negocio:\*\*

- `TurnoService` contiene la lógica relacionada con los "turnos". Por ejemplo, podría validar los datos antes de enviarlos al DAO para su almacenamiento o transformar datos recuperados de la base de datos antes de devolverlos al controlador.

- Permite modificar la lógica de negocio sin afectar al controlador ni al DAO.

2. \*\*Intermediario entre Controlador y DAO:\*\*

- El controlador (`CrearTurnosController`) se comunica con la capa de servicio en lugar de interactuar directamente con el DAO.

- Facilita la centralización de la lógica de negocio, haciendo que el controlador sea más sencillo y centrado en la lógica de presentación.

3. \*\*Facilita el Testeo:\*\*

- Las pruebas unitarias pueden ser más fáciles de manejar porque se puede probar la lógica de negocio aislada de la lógica de acceso a datos.

### ¿Es Necesario Siempre `TurnoService` en JDBC?

No es necesario tener un `TurnoService` para todas las aplicaciones que usan JDBC. Sin embargo, su uso puede ser beneficioso en los siguientes casos:

- \*\*Aplicaciones de Tamaño Mediano a Grande:\*\*

- Cuando la aplicación crece, la capa de servicio ayuda a organizar el código y facilita su mantenimiento y escalabilidad.

- \*\*Lógica de Negocio Compleja:\*\*

- Si hay una lógica de negocio significativa que necesita ser aplicada a los datos antes de almacenarlos o después de recuperarlos, la capa de servicio es útil para encapsular esta lógica.

- \*\*Separación de Preocupaciones:\*\*

- Ayuda a mantener la separación de preocupaciones, haciendo que el código sea más modular y fácil de entender.

### ¿Cuándo Podría No Ser Necesario?

- \*\*Aplicaciones Muy Pequeñas:\*\*

- En aplicaciones pequeñas o muy simples, puede ser innecesario agregar una capa adicional de abstracción.

- \*\*Prototipos o Aplicaciones de Prueba:\*\*

- Si estás construyendo un prototipo o una aplicación de prueba rápida, puedes omitir la capa de servicio para acelerar el desarrollo.

### Resumen del Código

Vamos a hacer un resumen de cómo se interrelacionan estos módulos y cuál es su función:

1. \*\*AdministradorDeConexiones.java\*\*

- Se encarga de administrar las conexiones con la base de datos.

- Proporciona métodos para conectar y desconectar la base de datos.

2. \*\*Turno.java\*\*

- Representa la entidad `Turno`.

- Contiene atributos y métodos relacionados con los datos de un turno.

3. \*\*TurnoDAO.java\*\*

- Define una interfaz para las operaciones CRUD relacionadas con `Turno`.

- Separa la definición de las operaciones de su implementación.

4. \*\*TurnoDaoMysqlImpl.java\*\*

- Implementa `TurnoDAO` usando JDBC para interactuar con MySQL.

- Contiene los métodos concretos para crear y encontrar turnos en la base de datos.

5. \*\*TurnoService.java\*\*

- Actúa como una capa de servicio que encapsula la lógica de negocio relacionada con los turnos.

- Se comunica con `TurnoDaoMysqlImpl` para realizar operaciones de negocio.

6. \*\*CrearTurnosController.java\*\*

- Controlador que actúa como punto de entrada para la aplicación.

- Interactúa con `TurnoService` para realizar operaciones y manejar la lógica de presentación.

### Ejemplo Simplificado sin `TurnoService`

Para un caso más simple, puedes eliminar `TurnoService` y realizar las operaciones directamente en el controlador:

```java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public class CrearTurnosController {

public static void main(String[] args) {

TurnoDaoMysqlImpl turnoDao = new TurnoDaoMysqlImpl();

Turno nuevoTurno = new Turno("Test 56");

// Crear un nuevo turno

turnoDao.create(nuevoTurno);

// Listar todos los turnos

ArrayList<Turno> turnos = turnoDao.findAll();

System.out.println(turnos);

}

}

```

En este ejemplo, `CrearTurnosController` se comunica directamente con `TurnoDaoMysqlImpl`, eliminando la necesidad de `TurnoService`.

### Conclusión

El módulo `TurnoService` no es un requerimiento absoluto para trabajar con JDBC, pero su inclusión puede aportar beneficios significativos en términos de organización y mantenimiento del código. Su necesidad depende del tamaño y la complejidad de la aplicación, así como de las mejores prácticas que se deseen seguir en el desarrollo del software.

Supongamos que en este programa voy a usar 2 bases de datos: MySQL y Posgrest. Que debería agregar/modificar en los modulos de para usar estas dos bases de datos? Supongamos para simplificar que en CrearTurnosController.java creo 2 métodos adicionales para guardar turnos y leer todos los turnos guardados de la base de Posgres. Escribe nuevamente todos los módulos que componen el programa para hacer esta implementación, inclusive el pom.xml

Para implementar el uso de dos bases de datos (MySQL y PostgreSQL) en tu aplicación, necesitas crear implementaciones específicas del DAO para cada base de datos y ajustar el controlador y la configuración del proyecto para soportar ambas conexiones.

### Estructura del Proyecto

1. \*\*AdministradorDeConexiones\*\*: Dos clases separadas para MySQL y PostgreSQL.

2. \*\*TurnoDAO\*\*: La interfaz que define las operaciones CRUD.

3. \*\*TurnoDaoMysqlImpl\*\*: Implementación de `TurnoDAO` para MySQL.

4. \*\*TurnoDaoPostgresImpl\*\*: Implementación de `TurnoDAO` para PostgreSQL.

5. \*\*TurnoService\*\*: Dos servicios, uno para cada base de datos.

6. \*\*CrearTurnosController\*\*: Controlador que interactúa con ambos servicios.

7. \*\*pom.xml\*\*: Configuración para incluir las dependencias de MySQL y PostgreSQL.

### Archivos del Proyecto

#### 1. AdministradorDeConexionesMysql.java

```java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.DriverManager;

public class AdministradorDeConexionesMysql {

public static Connection conectar() {

String url = "jdbc:mysql://localhost:3306/cac-24100?serverTimezone=UTC&useSSL=false";

String user = "root";

String password = "Hammil01";

String driver = "com.mysql.cj.jdbc.Driver";

Connection connection = null;

try {

Class.forName(driver);

connection = DriverManager.getConnection(url, user, password);

} catch (Exception e) {

System.err.println("No se ha podido conectar a la DB MySQL: " + e.getMessage());

}

return connection;

}

public static void desconectar(Connection connection) {

try {

connection.close();

} catch (Exception e) {

System.err.println(e);

}

}

}

```

#### 2. AdministradorDeConexionesPostgres.java

```java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.DriverManager;

public class AdministradorDeConexionesPostgres {

public static Connection conectar() {

String url = "jdbc:postgresql://localhost:5432/cac-24100";

String user = "postgres";

String password = "password";

String driver = "org.postgresql.Driver";

Connection connection = null;

try {

Class.forName(driver);

connection = DriverManager.getConnection(url, user, password);

} catch (Exception e) {

System.err.println("No se ha podido conectar a la DB PostgreSQL: " + e.getMessage());

}

return connection;

}

public static void desconectar(Connection connection) {

try {

connection.close();

} catch (Exception e) {

System.err.println(e);

}

}

}

```

#### 3. Turno.java

```java

package ar.com.codo24100.practica;

import java.time.LocalDate;

public class Turno {

private Long id;

private LocalDate fecha;

private Long tipo;

private String motivo;

public Turno(String motivo) {

this.motivo = motivo;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public LocalDate getFecha() {

return fecha;

}

public void setFecha(LocalDate fecha) {

this.fecha = fecha;

}

public Long getTipo() {

return tipo;

}

public void setTipo(Long tipo) {

this.tipo = tipo;

}

public String getMotivo() {

return motivo;

}

public void setMotivo(String motivo) {

this.motivo = motivo;

}

@Override

public String toString() {

return "Turno [id=" + id + ", fecha=" + fecha + ", tipo=" + tipo + ", motivo=" + motivo + "]";

}

}

```

#### 4. TurnoDAO.java

```java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public interface TurnoDAO {

void create(Turno turno);

ArrayList<Turno> findAll();

}

```

#### 5. TurnoDaoMysqlImpl.java

```java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

public class TurnoDaoMysqlImpl implements TurnoDAO {

public void create(Turno turno) {

String sql = "INSERT INTO TURNOS (MOTIVO) VALUES(?)";

Connection connection = AdministradorDeConexionesMysql.conectar();

try {

PreparedStatement pst = connection.prepareStatement(sql);

pst.setString(1, turno.getMotivo());

pst.executeUpdate();

} catch (Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexionesMysql.desconectar(connection);

}

}

@Override

public ArrayList<Turno> findAll() {

String sql = "SELECT \* FROM TURNOS";

Connection connection = AdministradorDeConexionesMysql.conectar();

ArrayList<Turno> turnos = new ArrayList<>();

try {

PreparedStatement pst = connection.prepareStatement(sql);

ResultSet res = pst.executeQuery();

while (res.next()) {

Long id = res.getLong(1);

String motivo = res.getString(2);

Turno turno = new Turno(motivo);

turno.setId(id);

turnos.add(turno);

}

} catch (Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexionesMysql.desconectar(connection);

}

return turnos;

}

}

```

#### 6. TurnoDaoPostgresImpl.java

```java

package ar.com.codo24100.practica;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

public class TurnoDaoPostgresImpl implements TurnoDAO {

public void create(Turno turno) {

String sql = "INSERT INTO TURNOS (MOTIVO) VALUES(?)";

Connection connection = AdministradorDeConexionesPostgres.conectar();

try {

PreparedStatement pst = connection.prepareStatement(sql);

pst.setString(1, turno.getMotivo());

pst.executeUpdate();

} catch (Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexionesPostgres.desconectar(connection);

}

}

@Override

public ArrayList<Turno> findAll() {

String sql = "SELECT \* FROM TURNOS";

Connection connection = AdministradorDeConexionesPostgres.conectar();

ArrayList<Turno> turnos = new ArrayList<>();

try {

PreparedStatement pst = connection.prepareStatement(sql);

ResultSet res = pst.executeQuery();

while (res.next()) {

Long id = res.getLong(1);

String motivo = res.getString(2);

Turno turno = new Turno(motivo);

turno.setId(id);

turnos.add(turno);

}

} catch (Exception e) {

System.err.println(e);

} finally {

AdministradorDeConexionesPostgres.desconectar(connection);

}

return turnos;

}

}

```

#### 7. TurnoService.java

```java

package ar.com.codo24100.practica;

import java.util.ArrayList;

public class TurnoService {

private TurnoDAO mysqlDao;

private TurnoDAO postgresDao;

public TurnoService() {

this.mysqlDao = new TurnoDaoMysqlImpl();

this.postgresDao = new TurnoDaoPostgresImpl();

}

public void crearTurnoMysql(Turno turno) {

this.mysqlDao.create(turno);

}

public ArrayList<Turno> listarTurnosMysql() {

return this.mysqlDao.findAll();

}

public void crearTurnoPostgres(Turno turno) {

this.postgresDao.create(turno);

}

public ArrayList<Turno> listarTurnosPostgres() {

return this.postgresDao.findAll();

}

}

```

#### 8. CrearTurnosController.java

```java

package ar.com.codo24100.practica;

public class CrearTurnosController {

public static void main(String[] args) {

TurnoService turnoService = new TurnoService();

Turno nuevoTurno = new Turno("Test 56");

// Guardar y listar turnos en MySQL

turnoService.crearTurnoMysql(nuevoTurno);

System.out.println("Turnos en MySQL:");

System.out.println(turnoService.listarTurnosMysql());

// Guardar y listar turnos en PostgreSQL

turnoService.crearTurnoPostgres(nuevoTurno);

System.out.println("Turnos en PostgreSQL:");

System.out.println(turnoService.listarTurnosPostgres());

}

}

```

#### 9. pom.xml

```xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0

</modelVersion>

<groupId>ar.com.codo24100.practica</groupId>

<artifactId>practica</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- MySQL Connector -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.28</version>

</dependency>

<!-- PostgreSQL Connector -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.3.1</version>

</dependency>

<!-- JUnit for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>11</source>

<target>11</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

```

### Explicación

1. \*\*AdministradorDeConexionesMysql.java\*\* y \*\*AdministradorDeConexionesPostgres.java\*\*: Clases separadas para manejar las conexiones a MySQL y PostgreSQL respectivamente.

2. \*\*TurnoDaoMysqlImpl.java\*\* y \*\*TurnoDaoPostgresImpl.java\*\*: Implementaciones separadas del DAO para interactuar con cada base de datos.

3. \*\*TurnoService.java\*\*: Contiene métodos para interactuar con ambas bases de datos.

4. \*\*CrearTurnosController.java\*\*: Controlador que muestra cómo guardar y listar turnos en ambas bases de datos.

5. \*\*pom.xml\*\*: Configuración de Maven para incluir los conectores necesarios para MySQL y PostgreSQL.

Con esta estructura, tu aplicación puede interactuar con ambas bases de datos y realizar operaciones CRUD en cada una.