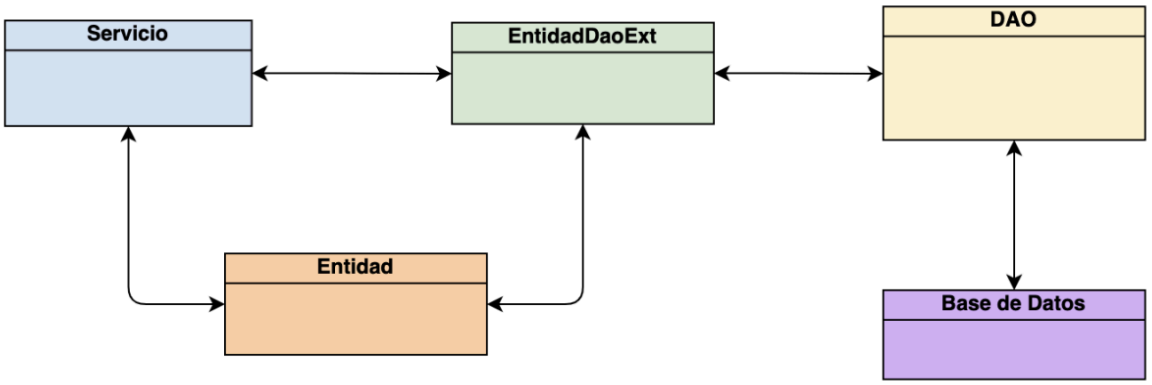
Material descargable

Java: JDBC. Patrón de diseño DAO

Repasemos, ¿Cuál es el patrón de diseño DAO?

El patrón de diseño DAO (Data Access Object) proporciona una capa de abstracción entre la lógica de negocio de una aplicación y el acceso a los datos almacenados en una base de datos. Este patrón facilita la separación de responsabilidades y la reutilización del código, permitiendo una gestión más eficiente y mantenible de los datos.



Implementación del DAO Abstracta

El DAO abstracto actúa como una superclase que define métodos comunes para interactuar con la base de datos. Las clases DAO específicas heredarán de esta clase y utilizarán estos métodos para implementar operaciones CRUD específicas del objeto al que manipulan.

Ten en cuenta que debe existir una superclase DAO que contenga los métodos para consultar, insertar, modificar y eliminar datos de la base de datos, así como los métodos necesarios para conectar y desconectar la base de datos.

En líneas generales, una clase abstracta DAO podría verse de la siguiente manera:

```java
package persistencia;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public abstract class DAO {

    protected Connection conexion = null;
    protected ResultSet resultSet = null;
    protected Statement statement = null;

    private final String HOST = "127.0.0.1";
    private final String PORT = "3306";
    private final String USER = "root";
    private final String PASSWORD = "root";
    private final String DATABASE = "vivero";
    private final String DRIVER = "com.mysql.cj.jdbc.Driver";
    private final String ZONA =
"?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverT
imezone=UTC";

    protected void connectarDataBase() throws SQLException, ClassNotFoundException {

        try {
            Class.forName(DRIVER);
            String url = "jdbc:mysql://" + HOST + ":" + PORT + "/" + DATABASE + ZONA;
            conexion = DriverManager.getConnection(url, USER, PASSWORD);
            System.out.println("Conexión exitosa a la base de datos.");

        } catch (Exception e) {
            System.out.println(e.getMessage());
            throw e;
        }

    }

    protected void desconectarDataBase() throws SQLException, ClassNotFoundException {
        try {
            if (resultSet != null) {
                resultSet.close();
            }

            if (statement != null) {
                statement.close();
            }

            if (conexion != null) {
                conexion.close();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
            throw e;
        }
```

```
    }

    protected void insertarModificarEliminarDataBase(String sql) throws Exception {
        try {
            connectarDataBase();
            statement = conexion.createStatement();
            statement.executeUpdate(sql);
            System.out.println("Dato OK en BBDD");

        } catch (SQLException | ClassNotFoundException ex) {
            System.out.println(ex.getMessage());
            throw ex;
        } finally {
            desconectarDataBase();
        }
    }

    protected void consultarDataBase(String sql) throws Exception {
        try {

            connectarDataBase();
            statement = conexion.createStatement();
            resultSet = statement.executeQuery(sql);

        } catch (SQLException | ClassNotFoundException ex) {
            System.out.println(ex.getMessage());
            throw ex;
        }
    }

}
```

Desglose de los Métodos

Vamos a repasar los puntos más importantes de los métodos definidos en el DAO abstracto:

- **connectarDataBase**:
  - **Función**: Establece la conexión con la base de datos.
  - **Detalles**: Utiliza `DriverManager` para obtener la conexión con los parámetros definidos (host, puerto, usuario, contraseña, base de datos, driver).
  - **Errores manejados**: SQLException, ClassNotFoundException.
- **desconectarDataBase**:
  - **Función**: Cierra la conexión con la base de datos, así como los `ResultSet` y `Statement` utilizados.

    - **Detalles**: Asegura que todos los recursos se liberen adecuadamente.
    - **Errores manejados**: SQLException, ClassNotFoundException.
- **insertarModificarEliminarDataBase**:
    - **Función**: Ejecuta una sentencia SQL de inserción, modificación o eliminación en la base de datos.
    - **Detalles**:
        - Establece conexión con la base de datos.
        - Ejecuta la sentencia SQL recibida como parámetro.
        - Cierra la conexión.
    - **Errores manejados**: SQLException, ClassNotFoundException.
- **consultarDataBase**:
    - **Función**: Ejecuta una consulta SQL en la base de datos y obtiene un `ResultSet` con los resultados.
    - **Detalles**:
        - Establece conexión con la base de datos.
        - Ejecuta la consulta SQL recibida como parámetro.
    - **Errores manejados**: SQLException, ClassNotFoundException.

En conclusión, continuarás implementando los métodos como lo hacías hasta ahora, simplemente migrarás la lógica a una clase encargada de dichas tareas.

Implementación del DAO Específico para una Entidad

Cada entidad en tu aplicación debe tener su propio DAO específico, el cual extiende de la clase abstracta DAO. Esto asegura que cada entidad tenga métodos personalizados para interactuar con su correspondiente tabla en la base de datos. Al hacerlo, se logra una separación clara de responsabilidades, lo que facilita el mantenimiento y la escalabilidad del código.

El DAO específico para una entidad hereda los métodos genéricos del DAO abstracto y agrega métodos específicos para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de dicha entidad. Esta estructura permite mantener un código limpio y modular, donde cada clase tiene una única responsabilidad.

Ejemplo Específico: ClienteDAO

A continuación, se presenta un ejemplo de implementación de un DAO específico para la entidad Cliente, denominado `ClienteDAO`. Este DAO extiende de la clase abstracta DAO y utiliza sus métodos para realizar operaciones CRUD específicas para los clientes.

```
package persistencia;

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import entidad.Cliente;

public class ClienteDAO extends DAO {

    public void guardarCliente(Cliente cliente) throws Exception {
        if (cliente == null) {
            throw new Exception("El cliente no puede ser nulo");
        }
        String sql = "INSERT INTO clientes (nombre, apellido, email) VALUES ('"
                    + cliente.getNombre() + "', '"
                    + cliente.getApellido() + "', '"
                    + cliente.getEmail() + "')";
        insertarModificarEliminarDataBase(sql);
    }

    public List<Cliente> listarTodosLosClientes() throws Exception {
        String sql = "SELECT id, nombre, apellido FROM clientes";
        consultarDataBase(sql);

        List<Cliente> clientes = new ArrayList<>();
        while (resultSet.next()) {
            Cliente cliente = new Cliente();
            cliente.setId(resultSet.getInt("id"));
            cliente.setNombre(resultSet.getString("nombre"));
            cliente.setApellido(resultSet.getString("apellido"));
            clientes.add(cliente);
        }
        return clientes;
    }

    public void eliminarClientePorId(int id) throws Exception {
        String sql = "DELETE FROM clientes WHERE id = " + id;
        insertarModificarEliminarDataBase(sql);
    }
}
```

Desglose de los Métodos

Vamos a repasar los puntos más importantes de los métodos definidos en ClienteDAO:

- **guardarCliente**:

    - **Función**: Inserta un nuevo cliente en la base de datos.
    - **Detalles**:
        - Verifica que el cliente no sea nulo.
        - Crea la sentencia SQL de inserción con los datos del cliente.
        - Utiliza el método `insertarModificarEliminarDataBase` de la superclase DAO para ejecutar la operación.
- **listarTodosLosClientes**:
    - **Función**: Consulta y devuelve una lista de todos los clientes en la base de datos.
    - **Detalles**:
        - Crea la sentencia SQL para seleccionar los clientes.
        - Utiliza el método `consultarDataBase` de la superclase DAO para ejecutar la consulta.
        - Procesa el `ResultSet` para crear una lista de objetos Cliente.
- **eliminarClientePorId**:
    - **Función**: Elimina un cliente de la base de datos por su ID.
    - **Detalles**:
        - Crea la sentencia SQL de eliminación con el ID del cliente.
        - Utiliza el método `insertarModificarEliminarDataBase` de la superclase DAO para ejecutar la operación.

En resumen, el patrón DAO ofrece una estructura organizada y modular para gestionar el acceso a datos en una aplicación. Entre sus beneficios destacan:

- **Mantenibilidad:** Facilita el mantenimiento del código al permitir que los cambios en el acceso a datos se realicen en una sola ubicación.
- **Flexibilidad:** Proporciona la capacidad de cambiar fácilmente la implementación de acceso a datos sin afectar otras partes de la aplicación.
- **Claridad:** Mejora la claridad del código al separar claramente las responsabilidades de la lógica de negocio y el acceso a datos.
- **Reutilización:** Promueve la reutilización de código al centralizar las operaciones de acceso a datos en objetos DAO.
- **Pruebas Unitarias:** Simplifica la creación de pruebas unitarias para la lógica de negocio al eliminar la dependencia de una base de datos real.

En resumen, la implementación de DAOs específicos para cada entidad permite gestionar las operaciones de base de datos de forma modular y eficiente. Esto no solo

mejora la organización del código, sino que también facilita la extensibilidad y el mantenimiento a largo plazo, asegurando que cada entidad maneje sus propias operaciones CRUD de manera independiente y coherente.