Material descargable

Java Database Connectivity (JDBC)

¿Qué es un patrón de diseño?

Un patrón de diseño es una solución probada que resuelve un tipo específico de problema en el desarrollo de software referente al diseño. Las ventajas de usar un patrón de diseño son, que permiten tener el código bien organizado, legible y mantenible, además te permite reutilizar código y aumenta la escalabilidad en tu proyecto.

En sí, proporcionan una terminología estándar y un conjunto de buenas prácticas en cuanto a la solución en problemas de desarrollo de software.

Patrón de diseño DAO

El patrón de diseño DAO 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.

Ventajas del patrón DAO:

- **Separación de responsabilidades:** La lógica de negocio y la lógica de acceso a datos están desacopladas, facilitando el mantenimiento y la escalabilidad.
- **Reutilización de código:** Los métodos de acceso a datos pueden ser reutilizados en diferentes partes del proyecto.
- **Facilidad para realizar cambios:** Si se modifica la fuente de datos o la forma de acceso, los cambios se concentran en la capa DAO.
- **Más claridad y organización:** La separación entre capas mejora la legibilidad del código.

Clases

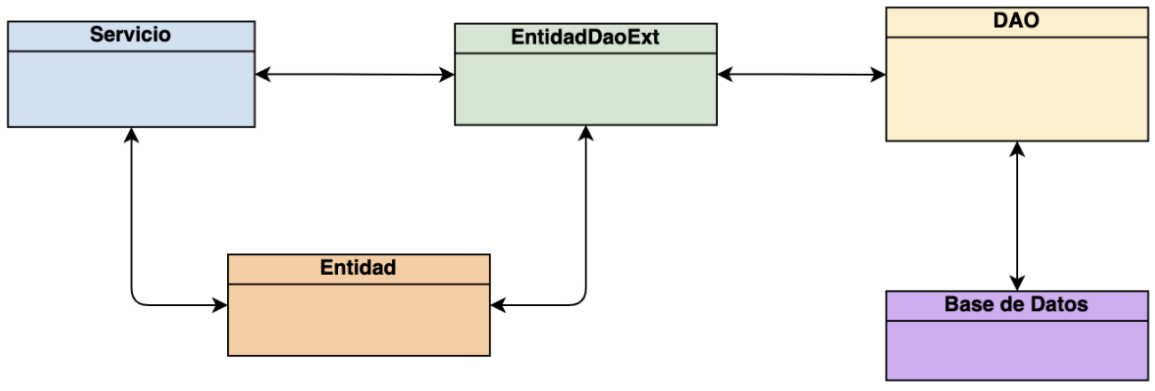
Este patrón se implementará mediante al menos cuatro modelos de clases principales:

Entidad: va a ser la clase que va a representar a la tabla que queremos trabajar de la base de datos. Va tener como atributos las columnas de la tabla de la base de datos.

Servicio o Business Service: va a tener toda la lógica de negocio del proyecto, usualmente se genera una para cada entidad. Es la que se encarga de obtener datos desde la base de datos y enviarla al cliente, o a su vez recibir la clase desde el cliente y enviar los datos al servidor, por lo general tiene todos los métodos CRUD (create, read, update y delete).

DAO: representa una capa de acceso a datos que oculta la fuente y los detalles técnicos para recuperar los datos. Esta clase va a ser la encargada de comunicarse con la base de datos, de conectarse con la base de datos, enviar las consultas y recuperar la información de la base de datos.

EntidadDaoExt: esta clase va a extender de la clase DAO y se va encargar de generar las sentencias para enviar a la clase DAO, como un insert, select, etc. Y si estuviéramos haciendo un select, sería también, la encargada de recibir la información, que recupera la clase DAO de la base de datos sobre una entidad, para después enviarla al servicio, que será la encargada de imprimir dicha información. Este es un objeto plano que implementa el patrón Data Transfer Object (DTO), el cual sirve para transmitir la información entre el DAO y el Business Service.



Implementación del DAO abstracto

El DAO abstracto actúa como una superclase que define métodos comunes para interactuar con la base de datos. Las clases DAO específicas heredan de esta clase y utilizan 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.

Ejemplo de Clase Abstracta DAO

```
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&s
erverTimezone=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;
        }

    }
```

```java
    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

1. 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.

2. 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.

3. 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.

4. 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.

Más adelante, verás cómo implementar este diseño en cada entidad específica, lo que te permitirá acceder a las tablas de la base de datos según lo necesites y realizar operaciones sobre ellas. De esta manera, podrás aplicar el patrón DAO de forma práctica y eficiente en tus proyectos.

Si deseas profundizar en el concepto de DAO o explorar más patrones de diseño relacionados, te recomiendo consultar la documentación oficial de Oracle sobre JDBC o materiales especializados en patrones de diseño de software como el libro *Design Patterns: Elements of Reusable Object-Oriented Software* de Gamma et al.