educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 1 - Java Analyst

Módulo 9

DAO

educación IT
CODE YOUR FUTURE

Introducción

Muchas veces, debemos **acceder a los datos desde diferentes fuentes** (archivos planos, bases de datos relacionales y no relacionales, servicios WEB, y otros.).

Esto representa la implementación de una lógica que se puede llegar a complicar por cada una de las fuentes, en las clases creadas para las entidades. Para solucionar este problema, contamos con el patrón de diseño **DAO**.

DAO

El patrón de diseño **DAO**, **Objeto de Acceso a Datos (*Data Access Object*)** propone separar la lógica de negocio (Entidades) de la lógica de acceso a los datos, proporcionando métodos para insertar, actualizar, eliminar y seleccionar esos datos. Este patrón de diseño viene acompañado de otro, el **DTO**, **Objeto de Transferencia de Datos (*Data Transfer Object*)**: es un patrón simple que propone la creación de un objeto que se encargue de la transferencia de datos entre el servidor y el cliente.

DAO

Todo patrón de diseño tiene una **estructura**. El **DAO** está representado de la siguiente manera:

- ***Business Object:*** representa un objeto con la lógica de negocio, “Entidades de negocio” (Persona, Documento, Alumno, y otros).
- ***DataAccess Object:*** representa una capa de acceso a datos que oculta la fuente y los detalles técnicos para recuperar los datos. Representada por lo general por una interfaz.
- ***Transfer Object:*** representa un objeto que implementa el patrón DTO.
- ***Data Source:*** representa la implementación de la interfaz.

Business Object

Representada por **cualquier entidad necesaria que ayude a resolver la problemática del negocio**.

```
public abstract class Familiar {
    private Colores color;
    private String marca;
    private Patente patente;
    public static String concesionaria = "Autos EducacionIT";
    private Integer puestos;


    // constructores

    // getters y setters
```

Data Access Object

Representada por una **interfaz que contiene la abstracción de los métodos necesarios para acceder, modificar, eliminar e insertar nuestros objetos**.

Podemos, sin ningún problema, crear una interfaz por cada entidad de negocio pero, para simplificar nuestro código, nos apoyamos en los genéricos.

```java
public interface DAO<E, K> {

    E buscarPorID(K key);

    boolean insertar(E elemento);

    boolean eliminar(E elemento);

    boolean actualizar(E elemento);

    List<E> listar();

}
```

Data Sources

Representada por un **objeto que implementa los algoritmos necesarios y el acceso a los datos**.

Para este ejemplo, separamos la conexión y la implementación para abstraer aún más:

```java
public interface ConexionMariaDB {

    default Connection getConexion() {
        Connection aux = null;

        try {
            final String DRIVER = "org.mariadb.jdbc.Driver";
            final String URL = "jdbc:mariadb://localhost:3306/autosEducacionIT";
            final String USER = "root";
            final String PASS = "";

            Class.forName(DRIVER);
            aux = DriverManager.getConnection(URL, USER, PASS);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return aux;
    }
}
```

```java
public class FamiliarImpl implements DAO<Familiar, String>, ConexionMariaDB{

    @Override
    public Familiar buscarPorID(String patente) {
        // Algoritmos
        return null;
    }

    @Override
    public boolean insertar(Familiar elemento) {
        // Algoritmos
        return false;
    }

    @Override
    public boolean eliminar(Familiar elemento) {
        // Algoritmos
        return false;
    }
```

...

```
...

    @Override
    public boolean actualizar(Familiar elemento) {
        // Algoritmos
        return false;
    }

    @Override
    public List<Familiar> listar() {
        // Algoritmos
        return null;
    }

}
```

Transfer Object

Representada por una **clase que se encarga de instanciar los objetos para interactuar con nuestro origen de datos**.

```
// creamos el objeto que transfiere la informacion entre nuestro software y la
// base de datos

FamiliarImpl familiarImpl = new FamiliarImpl();

// insertamos el objeto
Familiar autoFamiliar1 = new Familiar(Colores.MARRON, "Audi", new Patente("ZBG-999", true), 6, "SEDAN");

familiarImpl.insertar(autoFamiliar1);

// lo buscamos en la base de datos
Familiar autoFamiliarAuxiliar = familiarImpl.buscarPorID("COM-89655");
```

```
...

// cambiamos atributos y lo actualizamos en la base de datos
autoFamiliarAuxiliar.setMarca("TATA");

familiarImpl.actualizar(autoFamiliarAuxiliar);

// listamos los objetos de la base de datos y los recorremos
List<Familiar> listaFamiliar = familiarImpl.listar();

for (Familiar familiar : listaFamiliar) {
    System.out.println(familiar);
}

// eliminamos el objeto
familiarImpl.eliminar(autoFamiliar1);
```

¡Sigamos trabajando!