

# MEMORIA CIENTÍFICO TÉCNICA DEL PROYECTO

**DESARROLLADORES:** Carlos Robles, Jose Gracia

**TÍTULO DEL PROYECTO:** ADMINISTRADOR DE INCIDENCIAS CENTRO EDUCATIVO

## 1. RESUMEN DE LA PROPUESTA:

**Objetivo:**

Se precisa de un sistema para gestionar las incidencias de un centro y así llevar un seguimiento de las mismas. De esta forma se conseguirá un mejor rendimiento solucionando los problemas que surgen a lo largo del curso escolar.

**Resumen:**

La finalidad de esta aplicación es la gestión de las incidencias de un centro educativo. Con la misma se podrán crear, modificar y eliminar peticiones para solucionar problemas relacionados con los materiales del centro. Todas las incidencias se almacenarán en una base de datos, junto con las ubicaciones y usuarios que existen en el centro. Existirán 4 tipos de usuarios; profesor, jefe de departamento, mantenimiento, mensajes y administrador. Cada uno de los mismos tendrán diferentes opciones y podrán intercambiarse mensajes con el fin de solucionar las incidencias que vayan surgiendo de una forma más dinámica.

## 2. Patrones:

### Singleton

- Diseño creado para restringir la creación de objetos pertenecientes a una clase.
- Garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella.
- Este patrón se ha aplicado de la siguiente forma:
  - Se aplica en la clase raíz que se desea instanciar múltiples veces

```
package model;

import java.sql.Connection;

public class Conexion {

    private static Conexion instance;
    private Connection connect;

    private Conexion() {
        this.connect = conectar();
    }

    public static Conexion getInstance() {
        if (instance == null) {
            instance = new Conexion();
        }
        return instance;
    }
}
```

- Todas las clases que la instancien la llaman mediante *getInstance*

```
import java.sql.Connection;

public class jdbcProveedoresDAO implements proveedoresDAO {

    private Connection connect;
    private PreparedStatement ps;
    private ResultSet rs;

    public jdbcProveedoresDAO() {
        this.connect = Conexion.getInstance().conectar();
    }
}

import java.sql.Connection;

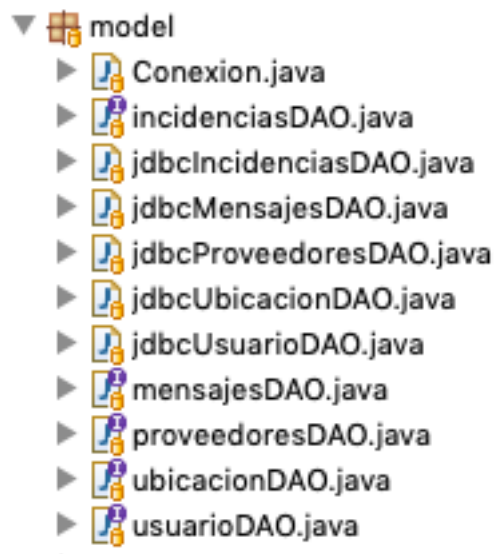
public class jdbcMensajesDAO {

    private Connection connect;
    private PreparedStatement ps;
    private ResultSet rs;

    public jdbcMensajesDAO() {
        this.connect = Conexion.getInstance().conectar();
    }
}
```

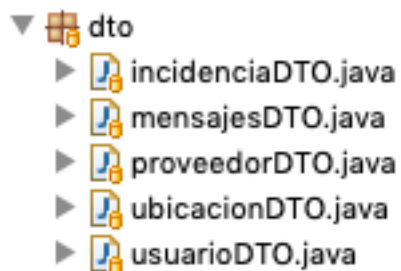
## DAO

- El problema que vine a resolver este patrón es el acceso a los datos, que básicamente tiene que ver con la gestión de diversas fuentes de datos y además abstrae la forma de acceder a ellos
- Cada clase se sigue a su correspondiente interfaz.
- En nuestro caso este patrón se aplica en la implementación de las clases de gestión de base de datos.
- Este patrón se ha aplicado de la siguiente forma:



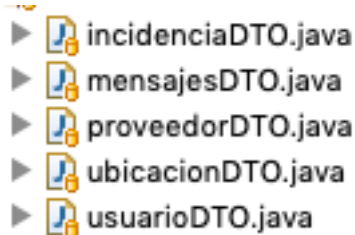
## DTO

- Crea objetos planos con una serie de atributos cuya utilidad solo puede ser de lectura; es decir han de evitar operaciones y métodos que realicen cálculos sobre los datos.

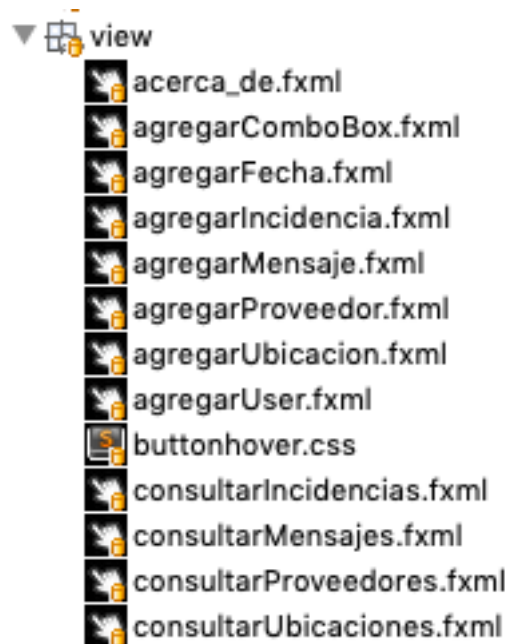


## Modelo vista controlador

- En español Modelo Vista Controlador, este patrón permite separar una aplicación en 3 capas, una forma de organizar y de hacer escalable un proyecto, a continuación una breve descripción de cada capa.
- **Modelo:** Esta capa representa todo lo que tiene que ver con el acceso a datos: guardar, actualizar, obtener datos, además todo el código de la lógica del negocio, básicamente son las clases Java y parte de la lógica de negocio.










- **Vista:** La vista tiene que ver con la presentación de datos del modelo y lo que ve el usuario, por lo general una vista es la representación visual de un modelo.



- **Controlador:** El controlador es el encargado de conectar el modelo con las vistas, funciona como un puente entre la vista y el modelo, el controlador recibe eventos generados por el usuario desde las vistas y se encarga de direccionar al modelo la petición respectiva.



Carlos Robles – Jose Gracia - DAW<sup>..</sup>

- ▶  controller
- ▶  controllerIncidencias
- ▶  controllerMensajes
- ▶  controllerProveedores
- ▶  controllerUbicaciones
- ▶  controllerUsuarios
- ▶  controllerUtilidades

### 3. Explicar una clase: *agregar\_incidencia* de *controllerIncidencias*

- Primera parte:

```

public class agregar_incidencia {
    @FXML
    private TextField descripcion;
    @FXML
    private TextField elemento;
    @FXML
    private DatePicker date;
    @FXML
    private TextField urgencia;
    @FXML
    private TextField categoria;
    @FXML
    private TextField materiales;
    @FXML
    private ComboBox<String> ubicacion;
    @FXML
    private Button agregarIncidencia;

    private Stage stage;

    private String nombreCompleto;
    private jdbcUbicacionDAO jdbcUbicacionDAO;
    private incidenciaDTO incidenciaDTO;
    private jdbcUsuarioDAO jdbcUsuarioDAO;
    private jdbcIncidenciasDAO jdbcIncidenciasDAO;

    /**
     * agregar_incidencia constructor default que inicializa variables
     */
    public agregar_incidencia() {
        this.nombreCompleto = "";
        this.jdbcUbicacionDAO = new jdbcUbicacionDAO();
        this.incidenciaDTO = new incidenciaDTO();
        this.jdbcUsuarioDAO = new jdbcUsuarioDAO();
        this.jdbcIncidenciasDAO = new jdbcIncidenciasDAO();
    }

    /**
     * inicializar inicializa los ComboBox llamando a las bases de datos de
     * ubicaciones para poner seleccionarlas en el ComboBox
     *
     * @param nombreCompleto nombre y apellidos del usuario logeado
     * @throws SQLException por si ha habido una excepción SQL
     */
    public void inicializar(String nombreCompleto) throws SQLException {
        this.nombreCompleto = nombreCompleto;
        ArrayList<String> ubicacionesArray = this.jdbcUbicacionDAO.leerNombresUbicaciones();

        ObservableList<String> ubicacionBox = FXCollections.observableArrayList(ubicacionesArray);
        this.ubicacion.setItems(ubicacionBox);
        this.ubicacion.setEditable(false);
        this.ubicacion.getSelectionModel().select(0);
        this.ubicacion.getStyleClass().add("center-aligned");// clase del css para centrar combobox
    }

```

La siguiente clase se utiliza para agregar una nueva incidencia

Declaración de atributos FXML

Declaración del stage

Declaración de jdbc's para acceder a las correspondientes bases de datos

Inicialización de jdbc's para acceder a las correspondientes bases de datos

La clase se inicializa con un parametro que recibe de la clase que la invoca

La vista se prepara para ser mostrada al usuario y se reciben las ubicaciones del combobox.

```
@FXML
/**
 * agregar incidencia tras recibir los datos introducidos por el usuario desde
 * TextFields y ComboBox, rellenamos un objeto incidenciaDTO y se lo pasamos a la
 * base de datos
 *
 * @throws SQLException por si ha habido una excepción SQL
 */
public void agregarIncidencia() throws SQLException {
    String[] nombreYapellidos = this.nombreCompleto.split(" ");
    // Se crea un array con el nombre y los apellidos del usuario

    this.incidenciaDTO.setUsuario(this.jdbcUsuarioDAO.leerUsuario(nombreYapellidos[0], nombreYapellidos[1]));
    this.incidenciaDTO.setDescripcion(this.descripcion.getText());
    this.incidenciaDTO.setElemento(this.elemento.getText());
    // Se añade un usuario descripción y elemento a la incidencia

    // Si la fecha está vacía cogerá la actual
    if (this.date.getValue() == null) {
        java.sql.Date sqlDate = new java.sql.Date(Calendar.getInstance().getTime().getTime());
        this.incidenciaDTO.setFecha(sqlDate);
    }
    // Métodos para obtener la fecha a través de una vista alternativa

    } else {
        Date date = java.sql.Date.valueOf(this.date.getValue());
        this.incidenciaDTO.setFecha((java.sql.Date) date);
        // Campo vacío = fecha actual
    }

    this.incidenciaDTO.setUrgencia(this.urgencia.getText());
    this.incidenciaDTO.setCategoria(this.categoria.getText());
    this.incidenciaDTO.setUbicacion(this.ubicacion.getValue());
    // Se añade urgencia, categoría y ubicación

    this.stage = (Stage) this.agregarIncidencia.getScene().getWindow(); // seleccionamos la escena actual
    this.stage.close(); // cerramos la ventana actual para pasar a la siguiente

    this.incidenciaDTO.setUbicacionI(this.jdbcUbicacionDAO.obtenerIdUbicacion(this.incidenciaDTO.getUbicacion()));
    this.jdbcIncidenciasDAO.crearIncidencia(this.incidenciaDTO);

    // Finalmente se crea la incidencia con todos los atributos de
    // objeto necesarios menos el de materiales(opcional)
}
```

#### 4. Problemas técnicos:

Estos son algunos de los problemas técnicos que hemos tenidos a lo largo de la creación de este proyecto:

- Paso de información entre interfaces.
- Acceso a la base de datos.
- Aplicación de los patrones DAO, DTO y Vista Controlador
- Cerrado de la base de datos.
- Distribución de clases.
- Filtrado de datos.
- Aplicación de instancias del patrón Singleton.
- Encriptación de credenciales.
- Distribución de paquetes.
- Creación del UML – Diagrama de clases
- Encriptación de credenciales



## 5. Posibles mejoras:

- **Importación de la base de datos a un hosting mysql online.**
- **Mejora de la base de datos mediante:**
  - **Triggers**
  - **Eventos**
  - **Vistas**
- **Mejorar efectos visuales de las interfaces.**
- **Compactación de código.**

## 6. Valoración personal:

En este proyecto hemos mejorado notablemente nuestros conocimientos en java. Nos hemos encontrado ante bastantes errores pero hemos sabido solucionarlos desde un principio gracias a diferentes lugares de la web como stackoverflow. Hemos estado al día de las últimas novedades de javafx, las cuales han sido aplicadas satisfactoriamente al proyecto. Esperamos la aprobación de Borja.

Jose y Carlos.