



# Práctico 2B

## Análisis del código fuente y patrones de diseño

**Grupo 7:** Barone Adrián, Hernández Agustina, Menéndez José, Pagliasso Francine.

### ¿Qué patrón identificaron?

El patrón que se utiliza es **Singleton**

### ¿Dónde y cómo se aplica en el código?

#### Donde:

Se aplica en la clase **DBConfigSingleton** (configuración de la base de datos, y administración de apertura o cierre).

#### Cómo:

##### -public final class DBConfigSingleton:

El modificador final en una declaración de clase significa que la clase no puede ser extendida o heredada. En otras palabras, ninguna otra clase puede ser una subclase de ella.

Esto se hace para que no exista la posibilidad de que una subclase modifique su comportamiento, el cual mantiene el patrón, que le da seguridad a la funcionalidad.

##### Instancia privada y estática:

Declara un campo estático y privado para asegurar que no se pueda acceder desde fuera de la clase.

```
private static DBConfigSingleton instance;
```

##### Constructor privado:

Esta visibilidad evita que la clase pueda instanciarse directamente desde afuera. Esto fuerza el uso del método de acceso estático.

```
private DBConfigSingleton() {
    // Configuraciones para SQLite
    this.driver = "org.sqlite.JDBC"; // Driver JDBC para SQLite
    this.dbUrl = System.getProperty("db.url", "jdbc:sqlite:./db/dev.db");
    this.user = ""; // SQLite no usa usuario
    this.pass = ""; // SQLite no usa contraseña
}
```

##### Método getInstance (de Acceso global y Sincronizado):

```
public static synchronized DBConfigSingleton getInstance() {
    if (instance == null) {
        instance = new DBConfigSingleton();
    }
}
```

```

        return instance;
    }
}

```

Permite acceder a la única instancia global o bien crearla si no existe. Tiene el modificador **synchronized**: esto es para que no más de un hilo pueda acceder al mismo tiempo, y crear conflictos por querer crear una instancia, u obtenerla.

#### **Uso en App.main:**

La clase principal App accede a la configuración de la DB usando este método de acceso global.

```
DBConfigSingleton dbConfig = DBConfigSingleton.getInstance();
```

#### **¿Qué problema resuelve este patrón en ese contexto?**

El **propósito** de este es que no haya más de una instancia de la clase, justamente porque debería haber solo un archivo de configuración y administración de sesión de la base de datos, para evitar inconsistencia de información y errores graves.

HU:

ID de HU	001
Título	Alta de profesor al sistema
Declaración	<b>Como</b> administrador del sistema, <b>quiero</b> registrar un nuevo profesor ingresando su información personal, <b>para</b> poder asignarlo a las asignaturas correspondientes dentro de una carrera.
Descripción Detallada	<p>Esta funcionalidad permite al usuario con rol de administrador acceder a un formulario de registro específico para profesores. El sistema debe presentar una interfaz donde el administrador ingresará los datos obligatorios: Nombre, Apellido, DNI, Correo Electrónico y Grado.</p> <p>El sistema debe verificar la integridad y formato de estos datos en tiempo real o al intentar enviar el formulario. Si los datos son válidos y no existen duplicados (DNI previamente registrado), el sistema registrará al profesor en la base de datos, habilitándolo para ser asignado posteriormente a las materias correspondientes. En caso contrario, se informará al usuario del error específico sin perder la información ya cargada, permitiendo su corrección.</p>
Criterios de Validación (Criterios de Aceptación)	<ul style="list-style-type: none"> <li>Flujo exitoso: Al completar todos los campos obligatorios (nombre, apellido, correo, DNI) con datos válidos y guardar, el sistema muestra un mensaje de éxito..</li> <li>Validaciones de Datos: El sistema debe impedir el registro si: <ul style="list-style-type: none"> <li>Faltan campos obligatorios.</li> <li>Si el formato del correo electrónico no es válido.</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>○ El correo electrónico o el DNI ya existen en la base de datos.</li> <li>● Manejo de Errores: Si alguna validación falla, el sistema debe mostrar un mensaje de error claro, sin permitir que se guarde el formulario.</li> <li>● Acción de Cancelar: El formulario debe incluir un botón "Cancelar" que elimine todos los datos ingresados y devuelva al usuario a la pantalla anterior.</li> </ul>
Tareas asociadas a la implementación	<ul style="list-style-type: none"> <li>● Crear tabla teachers en la base de datos.</li> <li>● Crear model Teacher.java.</li> <li>● Crear teachers_form.mustache.</li> <li>● Agregar rutas get y post para teacher.</li> </ul>