Bridge

1. Introducción

El **Patrón Bridge** es un patrón estructural que permite desacoplar una abstracción de su implementación, de manera que ambas puedan evolucionar de forma independiente.

En este caso, aplicamos el **Patrón Bridge** a la gestión de citas médicas, dividiendo la lógica en dos partes principales:

- **Abstracción (`FormatoCita`)**: Define los métodos para interactuar con las citas médicas.
- **Implementación (`FormatoCitaImplementacion` y `FormatoCitaBD`)**: Contiene la lógica concreta para acceder a los datos de la base de datos.

2. Problema

En el sistema de gestión de citas médicas, la clase `Formato_Citas` estaba fuertemente acoplada a la base de datos. Esto significaba que cualquier cambio en la forma en que se obtenían los datos (por ejemplo, cambiar de base de datos, usar una API externa o leer desde un archivo) requería modificar directamente la clase, lo que dificultaba la escalabilidad y el mantenimiento del sistema.

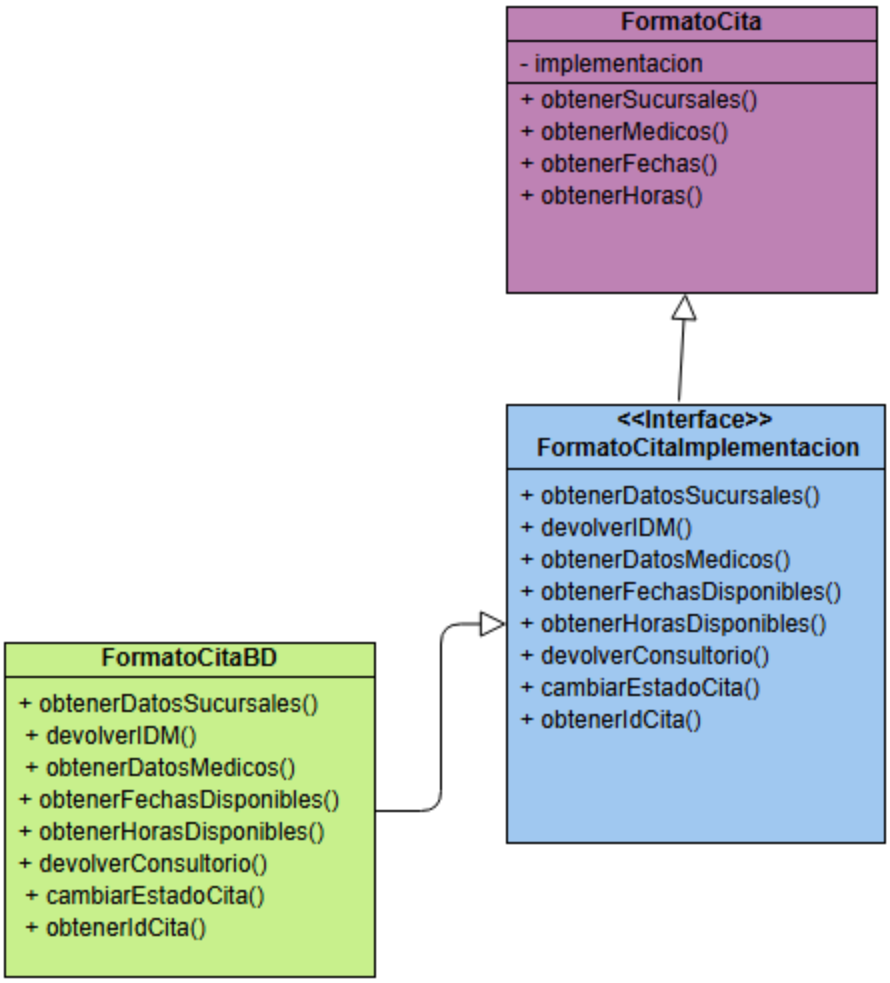
3. Participantes

1. **Abstracción (`FormatoCita`)**
   - Define la interfaz de alto nivel para la obtención y gestión de citas médicas.
   - Mantiene una referencia a una implementación concreta del puente.
2. **Implementación (`FormatoCitaImplementacion`)**

- Es la interfaz que define las operaciones que deben ser implementadas por las clases concretas.
- Permite desacoplar la abstracción de su implementación.

3. **Implementaciones Concretas (`FormatoCitaBD`)**
   - Implementa la interfaz de la implementación (`FormatoCitaImplementacion`).
   - Define la obtención de datos desde la base de datos, permitiendo gestionar las sucursales, médicos, fechas y horarios de las citas.

4. Diagrama UML



5. Código

El código sigue la estructura del **Patrón Bridge**, donde `FormatoCita` es la abstracción y `FormatoCitaBD` es la implementación que creamos.

3.1. Interfaz `FormatoCitaImplementacion`

Define los métodos que cualquier implementación de `FormatoCita` debe seguir.

```
public interface FormatoCitaImplementacion {
    void obtenerDatosSucursales(Connection conn, JComboBox<String> sucur
```

```
salComboBox);
    void devolverIDM(Connection conn, String idMedico);
    void obtenerDatosMedicos(Connection conn, JComboBox<String> medico
sComboBox);
    void obtenerFechasDisponibles(Connection conn, JComboBox<String> fec
hasComboBox);
    void obtenerHorasDisponibles(Connection conn, JComboBox<String> hora
sComboBox);
    void devolverConsultorio(Connection conn, String idMedico);
    void cambiarEstadoCita(Connection conn, String idCita, String estado);
    String obtenerIdCita(Connection conn);
}
```

3.2. Clase `FormatoCita` (Abstracción)

Esta clase mantiene una referencia a `FormatoCitaImplementacion`, permitiendo que las solicitudes de los clientes sean delegadas a la implementación específica.

```
/*
 * To change this license header, choose License Headers in Project Propertie
s.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author gatito-asesino
 */
public class FormatoCita {
    protected FormatoCitaImplementacion implementacion;

    public FormatoCita(FormatoCitaImplementacion implementacion) {
        this.implementacion = implementacion;
    }
```

```
    public void obtenerSucursales(Connection conn, JComboBox<String> sucu
rsalComboBox) {
        implementacion.obtenerDatosSucursales(conn, sucursalComboBox);
    }

    public void obtenerMedicos(Connection conn, JComboBox<String> medico
sComboBox) {
        implementacion.obtenerDatosMedicos(conn, medicosComboBox);
    }

    public void obtenerFechas(Connection conn, JComboBox<String> fechasC
omboBox) {
        implementacion.obtenerFechasDisponibles(conn, fechasComboBox);
    }

    public void obtenerHoras(Connection conn, JComboBox<String> horasCo
mboBox) {
        implementacion.obtenerHorasDisponibles(conn, horasComboBox);
    }
}
```

3.3. Clase `FormatoCitaBD` (Implementación)

Proporciona la lógica concreta para acceder a los datos de la base de datos.

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.t
xt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */
package Modelo;
import javax.swing.*;
import java.sql.*;
import java.util.ArrayList;
/**
```

```
 *
 * @author crist
 */
public class Formato_CitasBD {
    // ArrayLists para almacenar los datos de la base de datos
    private static ArrayList<Integer> idSucursales = new ArrayList<>();
    private static ArrayList<Integer> idMedicos = new ArrayList<>();
    private static ArrayList<Date> fechas = new ArrayList<>();
    private static ArrayList<Time> horas = new ArrayList<>();

    // Método para obtener los datos de las sucursales desde la base de datos
    public static void obtenerDatosSucursales(Connection conn, JComboBox<
String> sucursalComboBox) {
        try {
            String query = "SELECT id, entidad FROM Sucursales";
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                int id = rs.getInt("id");
                String entidad = rs.getString("entidad");
                idSucursales.add(id);
                sucursalComboBox.addItem(entidad);
            }
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static int devolverIDM(Connection conn, String nombre) {
        int nombre1=-1;
    try {
            String query = "SELECT id FROM Especialistas WHERE nombre = '"+n
ombre+"'";
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
```

```
            nombre1= rs.getInt("id");
          }

      } catch (SQLException e) {
          e.printStackTrace();
      }

   return nombre1;
}

   // Método para obtener los datos de los médicos desde la base de datos
   public static void obtenerDatosMedicos(Connection conn,JComboBox<Stri
ng> medicoComboBox, int Sucursal, String tipo) {
      try {
          String query = "SELECT distinct nombre FROM Especialistas WHERE e
specialidad = '"+tipo+"'";
          Statement stmt = conn.createStatement();
          ResultSet rs = stmt.executeQuery(query);
          while (rs.next()) {
              String nombre = rs.getString("nombre");
              medicoComboBox.addItem(nombre);
          }
          stmt.close();
      } catch (SQLException e) {
          e.printStackTrace();
      }
   }

   // Método para obtener las fechas disponibles desde la base de datos
   public static void obtenerFechasDisponibles(Connection conn, JComboBox
<String> fechaComboBox, int idMedico) {
   try {
      System.out.println(idMedico);
      String query = "SELECT DISTINCT fecha FROM Citas WHERE id_medico
= ? AND disponible = TRUE";
      PreparedStatement stmt = conn.prepareStatement(query);
```

```
        stmt.setInt(1, idMedico);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            System.out.println("con");
            Date fecha = rs.getDate("fecha");
            fechaComboBox.addItem(fecha.toString());
        }
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

    // Método para obtener las horas disponibles desde la base de datos
    public static void obtenerHorasDisponibles(Connection conn, JComboBox<
String> horaComboBox, String fecha, int idMedico) {
    try {
        String query = "SELECT DISTINCT hora FROM Citas WHERE fecha = ? A
ND id_medico = ? AND disponible = TRUE";
        PreparedStatement stmt = conn.prepareStatement(query);
        stmt.setString(1, fecha);
        stmt.setInt(2, idMedico);
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            Time hora = rs.getTime("hora");
            horaComboBox.addItem(hora.toString());
        }

        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
    public static String devolverConsultorio(Connection conn, String id) {
        String nombre1=null;
```

```
    try {
            String query = "SELECT consultorio FROM Especialistas WHERE id =
'"+id+"'";
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
               nombre1= rs.getString("consultorio");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }

    return nombre1;
}

    public static void cambiarEstadoCita(Connection conn, int idCita, int idAfilia
do, String nombreAfiliado, String fechaAgendada, String horaAgendada, Strin
g doctorAsignado, String fechaCita, String horaCita, String consultorio, String
tipo,int idMedico) {
    try {
        boolean captura;
        captura = ControladorVista_Historial_De_Citas.ultimaCitaYaPaso(conn, tip
o,idAfiliado+"");
        if(captura){
        String query = "UPDATE Citas SET disponible = FALSE, id_paciente = ? W
HERE id = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, idAfiliado);
        pstmt.setInt(2, idCita);
        pstmt.executeUpdate();
        pstmt.close();
        System.out.println(tipo);
        // Ahora insertamos los datos en la tabla citas_afiliado
        String insertQuery = "INSERT INTO citas_afiliado (id_cita, id_paciente, no
mbre_paciente, fecha_agendada, hora_agendada, doctor_asignado, fecha_cit
```

```
a, hora_cita, consultorio, tipo,id_medico) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement insertStmt = conn.prepareStatement(insertQuery);
        insertStmt.setInt(1, idCita);
        insertStmt.setInt(2, idAfiliado);
        insertStmt.setString(3, nombreAfiliado);
        insertStmt.setString(4, fechaAgendada);
        insertStmt.setString(5, horaAgendada);
        insertStmt.setString(6, doctorAsignado);
        insertStmt.setString(7, fechaCita);
        insertStmt.setString(8, horaCita);
        insertStmt.setString(9, consultorio);
        insertStmt.setString(10, tipo);
        insertStmt.setInt(11, idMedico);
        insertStmt.executeUpdate();
        insertStmt.close();

        JOptionPane.showMessageDialog(null, "La cita ha sido agendada correct
amente");}
        else{
            JOptionPane.showMessageDialog(null, "Tienes una cita agendada aú
n, intentalo de nuevo despues de finalizar tu cita");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

}
    public static int obtenerIdCita(Connection conn, int idDoctor, String fecha, S
tring hora) {
    int idCita = -1;
    try {
        String query = "SELECT id FROM Citas WHERE id_medico = ? AND fecha
= ? AND hora = ? AND disponible = TRUE";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, idDoctor);
        pstmt.setString(2, fecha);
```

```
        pstmt.setString(3, hora);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            idCita = rs.getInt("id");
        }
        pstmt.close();

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return idCita;
}


}
```

3.4. Uso del código

Finalmente, para utilizar el código, se instancia la abstracción con una implementación específica:

```
FormatoCitaImplementacion formatoCita = new FormatoCitaBD();
FormatoCita cita = new FormatoCita(formatoCita);

cita.obtenerSucursales(conn, sucursalComboBox);
cita.obtenerMedicos(conn, medicosComboBox);
cita.obtenerFechas(conn, fechasComboBox);
cita.obtenerHoras(conn, horasComboBox);
```

---

6. Conclusión

En la implementación del patrón **Bridge** dentro de nuetro proyecto, este permitió desacoplar la lógica de obtención de datos de la base de datos del sistema de gestión de citas médicas. Al separar la abstracción de su implementación, logramos que la estructura del código fuera más flexible y mantenible, facilitando

la posibilidad de modificar o extender la funcionalidad sin afectar otras partes del sistema.

Gracias a esta separación, ahora podemos agregar nuevas formas de gestionar citas o cambiar la fuente de datos sin alterar la lógica central. Esto es especialmente útil en un entorno donde los requisitos pueden cambiar con el tiempo o donde se pueden integrar nuevas tecnologías. Además, la aplicación de **Bridge** ayudó a reducir la dependencia directa entre las clases, permitiendo cumplir con los principios de diseño SOLID, en particular el principio de inversión de dependencias.
