

PATRON ADAPTER

El Patrón Adapter es un patrón estructural que permite que clases con interfaces incompatibles trabajen juntas sin modificar su código original. Se usa cuando:

- Queremos reutilizar código existente que no es compatible con una nueva interfaz.
- Necesitamos adaptar una API o servicio externo sin modificar el código base.

APLICACIÓN DEL PATRÓN ADAPTER AL CODIGO

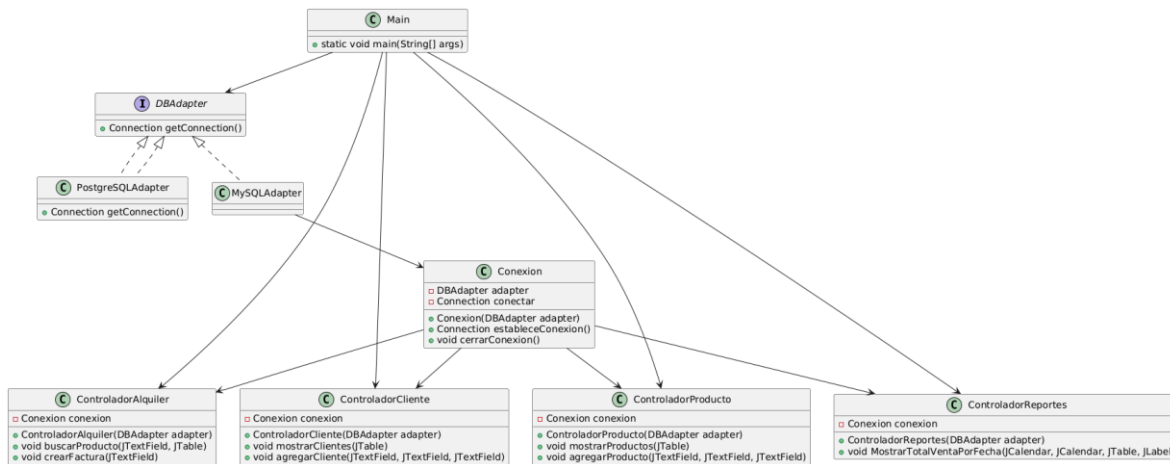
En nuestro código, el problema era que todas las clases (ControladorAlquiler, ControladorCliente, etc.) estaban directamente acopladas a la clase Conexion, la cual se conectaba exclusivamente a MySQL con DriverManager.

El objetivo del Adapter es permitir que estas clases trabajen con diferentes bases de datos sin modificar su código.

SOLUCIÓN CON EL PATRÓN ADAPTER

- Creamos una interfaz DBAdapter
Define el método getConnection(), que debe ser implementado por cualquier base de datos.
- Creamos una clase MySQLAdapter
Implementa DBAdapter y gestiona la conexión a MySQL.
- Modificamos la clase Conexion
En lugar de depender de DriverManager, ahora usa DBAdapter para obtener la conexión.
- Modificamos los Controladores (ControladorAlquiler, ControladorCliente, etc.)
Ahora reciben una instancia de Conexion basada en DBAdapter, lo que les permite conectarse a cualquier base de datos sin cambiar su código.

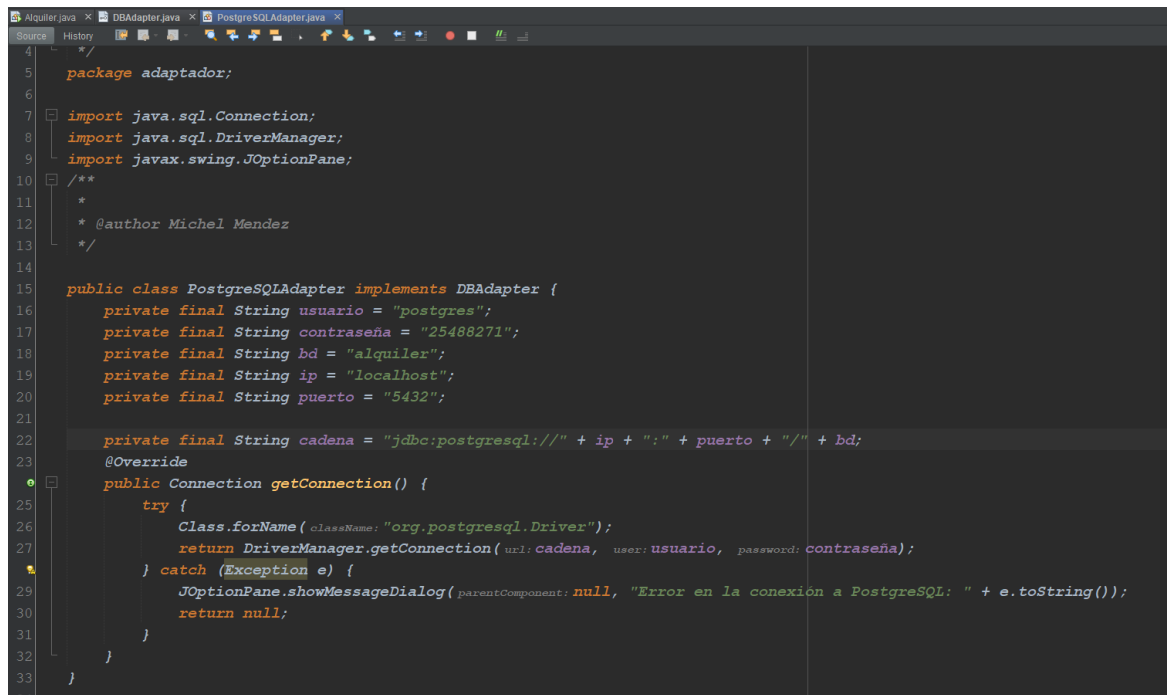
UML DE LA IMPLEMENTACION DEL PATRON ADAPTER



APLICACIÓN DEL PATRON ADAPTER AL PROGRAMA

```
Alquiler.java x DBAdapter.java x PostgreSQLAdapter.java x
Source History
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes
4  */
5  package adaptador;
6
7  import java.sql.Connection;
8
9  /**
10   *
11   * @author Michel Mendez
12   */
13
14  public interface DBAdapter {
15      Connection getConnection();
16  }
```

Creamos la interfaz DBAdapter, la cual define un contrato para que todas las bases de datos tengan el mismo método getConnection().

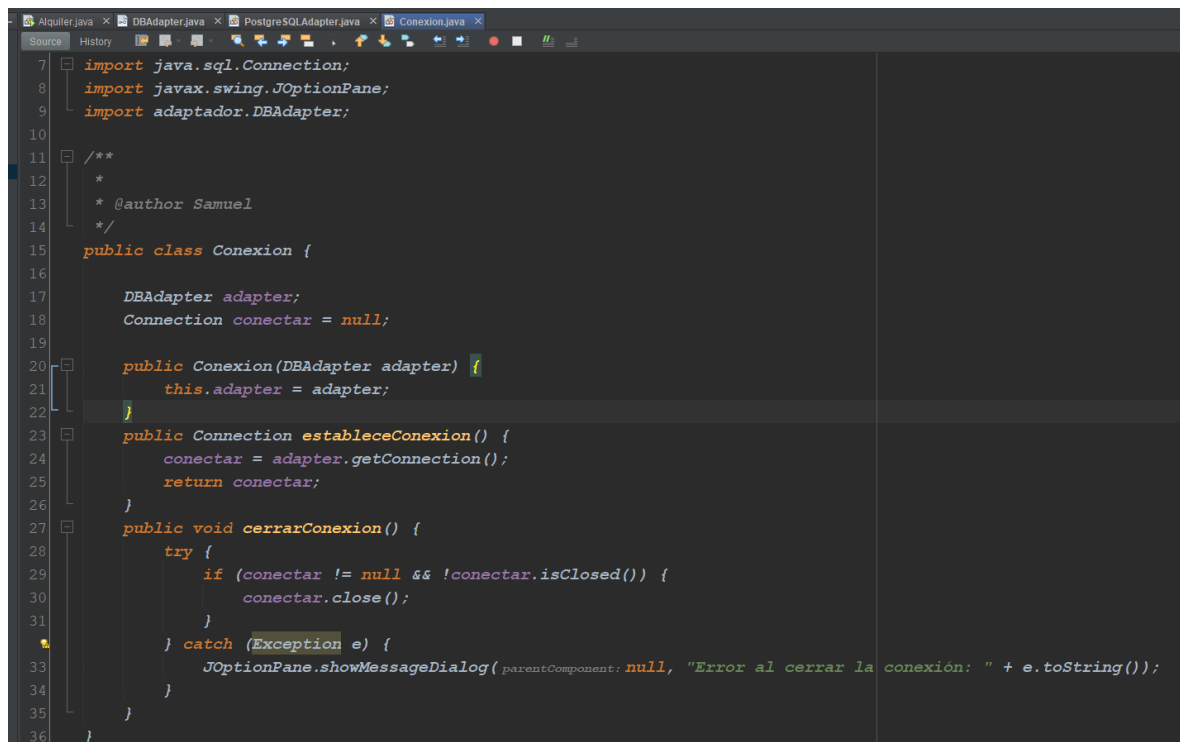


```

4  */
5  package adaptador;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import javax.swing.JOptionPane;
10
11  /**
12   *
13   * @author Michel Mendez
14   */
15  public class PostgreSQLAdapter implements DBAdapter {
16      private final String usuario = "postgres";
17      private final String contraseña = "25488271";
18      private final String bd = "alquiler";
19      private final String ip = "localhost";
20      private final String puerto = "5432";
21
22      private final String cadena = "jdbc:postgresql://" + ip + ":" + puerto + "/" + bd;
23      @Override
24      public Connection getConnection() {
25          try {
26              Class.forName("org.postgresql.Driver");
27              return DriverManager.getConnection(url: cadena, user: usuario, password: contraseña);
28          } catch (Exception e) {
29              JOptionPane.showMessageDialog(parentComponent: null, "Error en la conexión a PostgreSQL: " + e.toString());
30              return null;
31          }
32      }
33  }
34

```

Implementamos un Adaptador para PostgreSQL (PostgreSQLAdapter), para que funcione con PostgreSQL.



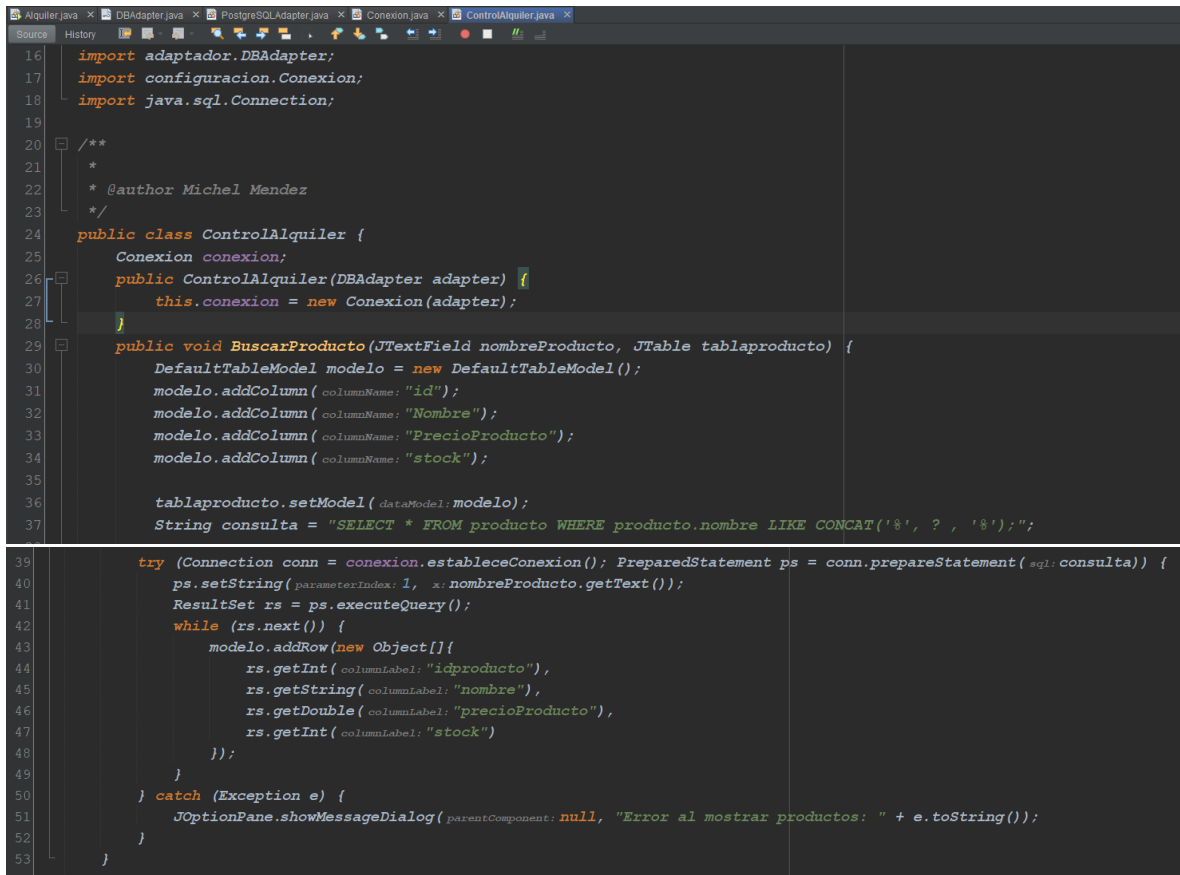
```

7  import java.sql.Connection;
8  import javax.swing.JOptionPane;
9  import adaptador.DBAdapter;
10
11  /**
12   *
13   * @author Samuel
14   */
15  public class Conexion {
16
17      DBAdapter adapter;
18      Connection conectar = null;
19
20      public Conexion(DBAdapter adapter) {
21          this.adapter = adapter;
22      }
23      public Connection estableceConexion() {
24          conectar = adapter.getConnection();
25          return conectar;
26      }
27      public void cerrarConexion() {
28          try {
29              if (conectar != null && !conectar.isClosed()) {
30                  conectar.close();
31              }
32          } catch (Exception e) {
33              JOptionPane.showMessageDialog(parentComponent: null, "Error al cerrar la conexión: " + e.toString());
34          }
35      }
36  }
37

```

Modificamos la clase Conexión para usar DBAdapter, ya que en nuestro código original Conexión dependía directamente de Mysql, ahora, se hace

usando DBAdapter. Ahora Conexión es independiente del tipo de base de datos.



```
16 import adaptador.DBAdapter;
17 import configuracion.Conexion;
18 import java.sql.Connection;
19
20 /**
21  *
22  * @author Michel Mendez
23  */
24 public class ControlAlquiler {
25     Conexion conexion;
26     public ControlAlquiler(DBAdapter adapter) {
27         this.conexion = new Conexion(adapter);
28     }
29     public void BuscarProducto(JTextField nombreProducto, JTable tablaproducto) {
30         DefaultTableModel modelo = new DefaultTableModel();
31         modelo.addColumn("id");
32         modelo.addColumn("Nombre");
33         modelo.addColumn("PrecioProducto");
34         modelo.addColumn("stock");
35
36         tablaproducto.setModel(modelo);
37         String consulta = "SELECT * FROM producto WHERE producto.nombre LIKE CONCAT('%', ?, '%')";
38
39         try (Connection conn = conexion.estableceConexion(); PreparedStatement ps = conn.prepareStatement(sql: consulta)) {
40             ps.setString(1, nombreProducto.getText());
41             ResultSet rs = ps.executeQuery();
42             while (rs.next()) {
43                 modelo.addRow(new Object[]{
44                     rs.getInt("idproducto"),
45                     rs.getString("nombre"),
46                     rs.getDouble("precioProducto"),
47                     rs.getInt("stock")
48                 });
49             }
50         } catch (Exception e) {
51             JOptionPane.showMessageDialog(null, "Error al mostrar productos: " + e.toString());
52         }
53     }
54 }
```

Ahora los adaptamos a nuestros controladores, en lugar de crear una nueva conexión cada vez, recibe una instancia de conexión.

Realiza la conexión en lugar de crear una nueva cada vez, se puede usar cualquier base de datos sin modificar el código.

Entramos a SQL Shell (psql) y creamos nuestra base de datos.

Creamos nuestra base de datos con los siguientes comandos:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]:
Port [5432]: 5432
Username [postgres]: postgres
Contraseña para usuario postgres:

psql (12.22)
ADVERTENCIA: El código de página de la consola (850) difiere del código
de página de Windows (1252).
Los caracteres de 8 bits pueden funcionar incorrectamente.
Vea la página de referencia de psql «Notes for Windows users»
para obtener más detalles.
Digite «help» para obtener ayuda.

postgres=# CREATE DATABASE alquiler;
CREATE DATABASE
postgres=# \c alquiler;
Ahora está conectado a la base de datos «alquiler» con el usuario «postgres»
.
alquiler=#
alquiler=# CREATE TABLE cliente (
alquiler(#      idcliente SERIAL PRIMARY KEY,
alquiler(#      nombres VARCHAR(100),
alquiler(#      appaterno VARCHAR(100),
alquiler(#      apmaterno VARCHAR(100)
alquiler(# );
CREATE TABLE
alquiler=# INSERT INTO cliente (nombres, appaterno, apmaterno) VALUES
alquiler-# ('Michel', 'Mendez', 'Mendoza');
INSERT 0 1
alquiler=# CREATE TABLE factura (
alquiler(#      idfactura SERIAL PRIMARY KEY,
alquiler(#      fechaFactura DATE,
alquiler(#      fkcliente INT REFERENCES cliente(idcliente)
alquiler(# );
CREATE TABLE
```

```

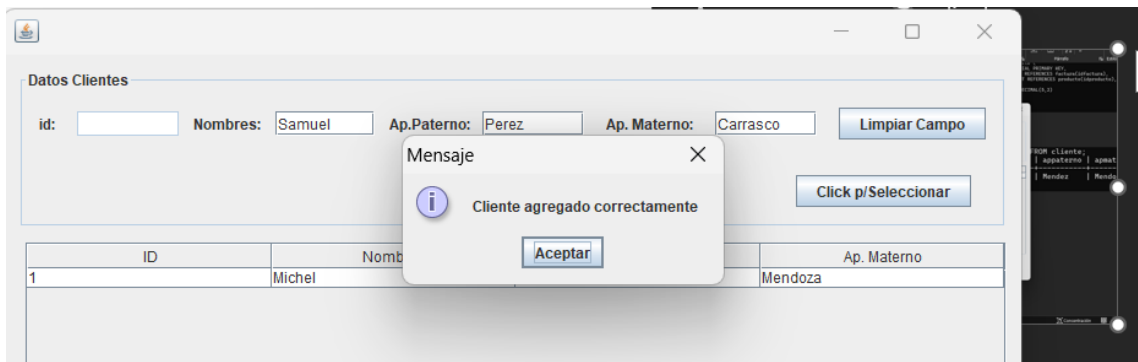
alquiler=# CREATE TABLE producto (
alquiler(#      idproducto SERIAL PRIMARY KEY,
alquiler(#      nombre VARCHAR(100),
alquiler(#      precioProducto DECIMAL(25,2),
alquiler(#      stock INT
alquiler(# );
CREATE TABLE
alquiler=# INSERT INTO producto (nombre, precioProducto, stock) VALUES
alquiler-# ('SILLAS METALICAS', 5, 1000),
alquiler-# ('SILLAS PLASTICAS', 5, 1000),
alquiler-# ('SILLAS MADERA', 15, 1000),
alquiler-# ('SILLAS TIFANY', 20, 1000),
alquiler-# ('SILLAS ACOJINADAS', 15, 1000),
alquiler-# ('SILLAS INFANTILES', 5, 1000);
INSERT 0 6
alquiler=# CREATE TABLE detalle (
alquiler(#      iddetalle SERIAL PRIMARY KEY,
alquiler(#      fkfactura INT REFERENCES factura(idfactura),
alquiler(#      fkproducto INT REFERENCES producto(idproducto),
alquiler(#      cantidad INT,
alquiler(#      precioVenta DECIMAL(5,2)
alquiler(# );
CREATE TABLE
alquiler=# |

```

PRUEBAS

Agregamos un nuevo cliente:

The screenshot displays a Java application window titled 'nú Principal - Alquileres California'. The interface includes a sidebar with buttons for 'RENTA', 'R COMPROBANTE', 'RTE POR FECHAS', 'ALMACÉN', 'CLIENTES', and 'CERRAR'. A 'Datos Clientes' dialog box is open, featuring input fields for 'id', 'Nombres', 'Ap. Paterno', and 'Ap. Materno', along with 'Limpiar Campo' and 'Click p/Seleccionar' buttons. Below these fields is a table with one row: ID 1, Nombres Michel, Ap. Paterno Mendez, Ap. Materno Mendoza. At the bottom of the dialog are 'Guardar', 'Modificar', and 'Eliminar' buttons. In the background, a code editor shows Java code for database connections and SQL queries.



Revisamos nuestra base de datos:

