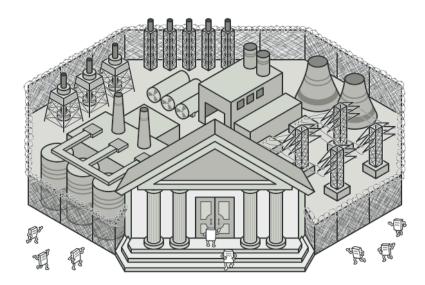
PATRÓN DE DISEÑO: FACADE

Facade es un patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases, Crea una interfaz única para acceder a un conjunto de funcionalidades complejas.



La clase de Clientesclass java no necesita cambios relevantes, lo único que tiene que hacer es seguir proporcionando los métodos para acceder a los datos de los clientes, esta clase nos servirá para seguir accediendo a los datos de los clientes.

Ahora para poder aplicar el patrón de Facade vamos a realizar una fachada que eso quiere decir Facade en español.

Creamos nuestra clase de ClientesFacade quedando de la siguiente forma.

```
package Modelo;
import java.util.List;
public class ClientesFacade {
private Clientesclass clientesclass;
  public ClientesFacade() {
    this.clientesclass = new Clientesclass();
  // Método que obtiene todos los clientes
  public List<Clientesclass> obtenerClientes() {
    return clientesclass.obtenerClientes();
  // Método que obtiene clientes por número de celular
  public List<Clientesclass> obtenerClientesPorNumero(String numero) {
    return clientesclass.obtenerClientesPorNumero(numero);
```

} // Método que obtiene clientes por ID y nombre public List<Clientesclass> obtenerClientesidnombre() { return clientesclass.obtenerClientesidnombre(); Copymax [main] Source Packages package Modelo; <default package> Conexion 3 ☐ import java.util.List: Conexion.java public class ClientesFacade { lconosproductos 8 9 Imagenes private Clientesclass clientesclass; ⊞₀Modelo public ClientesFacade() { Clientesclass.java [-/M] 10 this.clientesclass = new Clientesclass(); FiltroLetras.java 11 FiltroNumerosDecimales.java 12 🖃 public List<Clientesclass> obtenerClientes() { Filtromayusculas.java return clientesclass.obtenerClientes(); Filtronumeros.java Letraseditor.java 15 📮 public List<Clientesclass> obtenerClientesPorNumero(String numero) { Numerosdecimaleseditor.java 16 return clientesclass.obtenerClientesPorNumero(numero); Numeroseditor.java public List<Clientesclass> obtenerClientesidnombre() { 18 📮 Pedidoclass.java 19 return clientesclass.obtenerClientesidnombre(); Productoclass.java 20

Quedando una lista de Clientes por cada método, esto nos ayudará a que podamos usar los nuevos métodos de facade y reemplazarlos por los viejos que se encuentran en Clientesclass.java.

Lo que debemos hacer es implementar el Facade en nuestro programa, esto quiere decir que vamos a cambiar los métodos viejos por los nuevos, ya que facade sería nuestra interfaz simplificada que sería nuestra "intermedia" entre el cliente y la obtención de Clientes.

Como trabajamos en Netbeans nos permite buscar donde se usan los métodos que estamos usando:

```
Projects × Files | Services _ | 🔊 Clientesclass.java [-/M] × | 🖓 ClientesFacade.java [-/A] ×
                                                                                                                © Clemesclass.java | M| × © ClientesFacade.java | M| × © ClientesClass | M| × © ClientesCl
  Copymax [main]
Source Packages
Selection Conexion
Conexion, java
                                                                                                                                    □ public List<Clientesclass> obtenerClientes() {
    List<Clientesclass> clientes = new ArrayList<);
    Conexion conex = new Conexion();
    String sql = "SELECT Nombre, Apellidos, Calular, RFC, Correo FROM Cliente";
                                                                                                                                                                          try (Connection con = conex.getConnection();
    PreparedStatement pst = con.prepareStatement(sql);
    ResultSet rs = pst.executeQuery()) {

Clientescade, java [-/A]
Clientescas, java [-M]
FiltroLetras, java
FiltroLetras, java
FiltromerosDecimales, java
Filtronumeros java
Filtronumeros, java
Letraseditor, java
Numerosdecimaleseditor, java
Pedidoclass, java
Pedidoclass, java
Pedidoclass, java
                                                                                                                                                                                                while (rs.next()) {
   Clientesclass cliente = new Clientesclass();
                                                                                                                                                                                                            cliente.setNombre(rs.getString("Nombre"));
cliente.setApellidos(rs.getString("Apellidos"));
cliente.setRet(ers.getString("N=C"));
cliente.setRet(rs.getString("N=C"));
cliente.setCorreo(rs.getString("Orreo"));
                     | Productoclass.java
| Productosprecios.java
| Ticket.iava
  Usages of obtenerClientes × Usages of obtenerClientes × Usages of obtenerClientes ×
                            Usages of Clientesclass.obtenerClientes [6 occurrent

© Copymax

13: return clientesclass.obtenerClientes();
                                                                                                                                                                                                                                                                                            private void llenarTabla() {
    Clientesclass clientee = new Clientesclass();
    List<Clientesclass> clientes = clientee.obtenerClientes();
                                 for (Clientesclass cliente : clientes) {
    Object[] fila = new Object[5];
    fila[0] = cliente, getknebmer();
    fila[1] = cliente, getknellidos();
    fila[2] = cliente, getcoluar();
    fila[3] = cliente, getRecluar();
    fila[4] = cliente.getCorrec();
    modelo_andMownfila();

                                                                                                                                                                                                                                                                       © ZGZ_USINGHIGHOUSES=

Pedido,java

© 127: List<Clientesclass> clientes = clientee.obtenerClientes();
                                                                                                                                                                                                                                                                                            * @param celular Número de celular del cliente a eliminar.
```

Ahi debemos de usar nuestra facade de la siguiente forma:

Importamos nuestra clase de facade al inicio de la clase.

```
Projects × Files
              Services
Copymax [main]
                         Source Design History 🖟 🖟 📲 - 🔍 👯 🖓 🖶 🖫 🔓 😭 👙 🔮 🚇 🐞 🕒 🏰
□ Source Packages
> = <default package>
                               package Vista;
                            □ import Modelo.ClientesFacade;
Conexion

  O

   Conexion.java
                           4
                               import Modelo.Clientesclass;
> 🖺 Iconosproductos
                           5
                               import Conexion.Conexion;
                               import Modelo.Filtronumeros;
> 🏥 Imagenes
```

Y modificamos nuestro método donde se utilizaba el viejo método:

```
private void llenarTabla() {
362
           ClientesFacade clientee = new ClientesFacade();
363
           List<Clientesclass> clientes = clientee.obtenerClientes();
364
365
           for (Clientesclass cliente : clientes) {
366
               Object[] fila = new Object[5];
               fila[0] = cliente.getNombre();
367
368
               fila[1] = cliente.getApellidos();
369
               fila[2] = cliente.getCelular();
370
               fila[3] = cliente.getRfc();
               fila[4] = cliente.getCorreo();
371
372
               modelo.addRow(fila);
373
374
375
```

Usamos la nueva Fachada de ClientesFacade.

Hacemos eso en todo donde se ocupe la vieja clase de Clientesclass.java

```
public void actualizarTabla() {
97
          modelo.setRowCount(0); // Limpiar la tabla
98
99
          ClientesFacade clientee = new ClientesFacade();
00
          List<Clientesclass> clientes = clientee.obtenerClientes();
01
02
   \dot{\Box}
          for (Clientesclass cliente : clientes) {
03
              Object[] fila = new Object[5];
04
              fila[0] = cliente.getNombre();
05
              fila[1] = cliente.getApellidos();
06
              fila[2] = cliente.getCelular();
07
              fila[3] = cliente.getRfc();
08
              fila[4] = cliente.getCorreo();
09
              modelo.addRow(fila);
10
          }
11
```

Al final nuestro método de Cliente únicamente será ocupado por nuestra clase "intermediaria

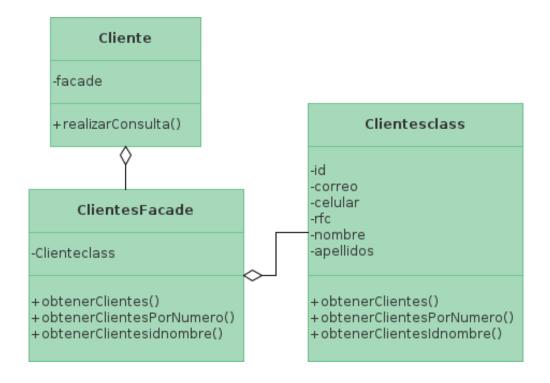
[&]quot;Facade que la usara para obtener datos, pero no será ocupada por ninguna otra clase:

```
Usages of obtenerClientes × Us
```

¿Por qué se implementó de esta forma el patrón? Bueno queríamos simplificar la interfaz de como se manejan los clientes, esto igual desacopla la clase FacadeClientes de los datos y consultas, con eso conseguimos que las clases que soliciten datos y Clientes únicamente interactúen con la Fachada pero no con la clase de las consultas.

Así mismo, si en el futuro decides agregar más métodos o lógicas complejas para la obtención de datos de clientes, puedes hacerlo dentro de la clase ClientesFacade sin modificar las clases que consumen los datos.

El patrón Facade proporciona una interfaz clara y directa para realizar acciones sobre los datos de clientes. Los usuarios de la fachada no necesitan preocuparse por la complejidad interna, lo que hace que trabajar con el código sea más intuitivo y accesible.



Ventajas	Desventajas
Simplicidad: Proporciona una interfaz simplificada y clara para acceder a funcionalidades complejas.	Dependencia de la fachada: Si la fachada se vuelve demasiado grande o compleja, podría hacer que la implementación sea difícil de mantener.
Desacoplamiento: El patrón desacopla las clases que consumen datos de la lógica de negocio y de acceso a datos.	Rendimiento: En sistemas de alto rendimiento, la capa adicional de abstracción puede introducir una pequeña sobrecarga.
Centraliza la lógica: La lógica de acceso a datos y operaciones complejas se centraliza en un único punto.	
Futuras extensiones fáciles: Nuevas funcionalidades pueden ser añadidas dentro de la fachada sin afectar el resto del sistema.	