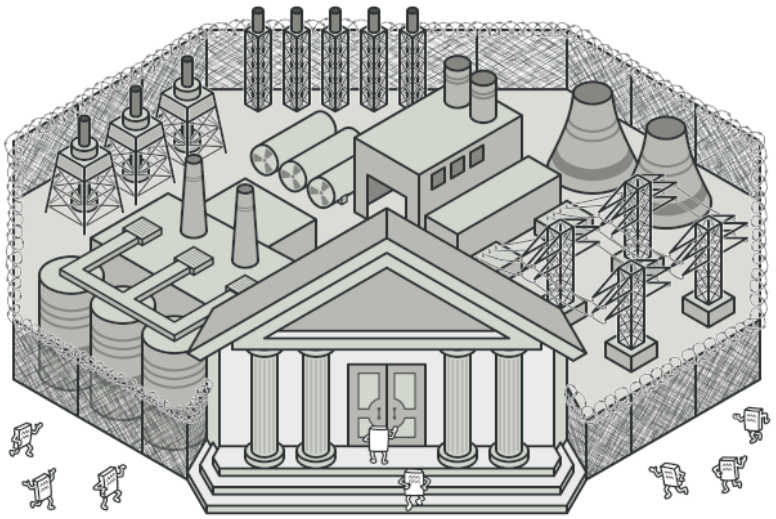
PATRÓN DE DISEÑO: FACADE

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.



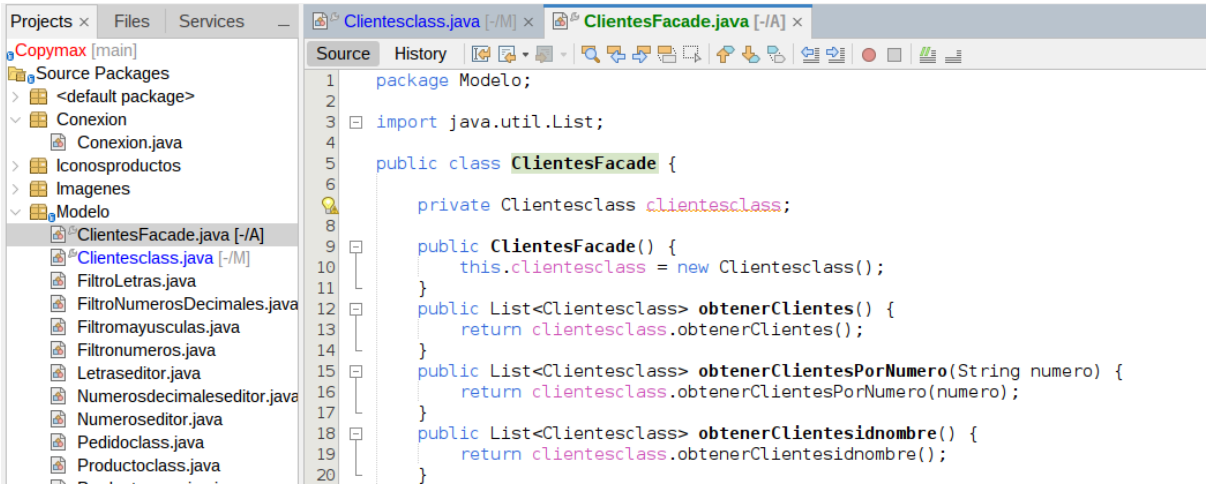
La clase de Clientesclass.java no necesita cambios relevantes, lo único que tiene que hacer es seguir proporcionando los métodos para acceder a los datos de los clientes, esta clase nos servirá para seguir accediendo a los datos de los clientes.

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);
```

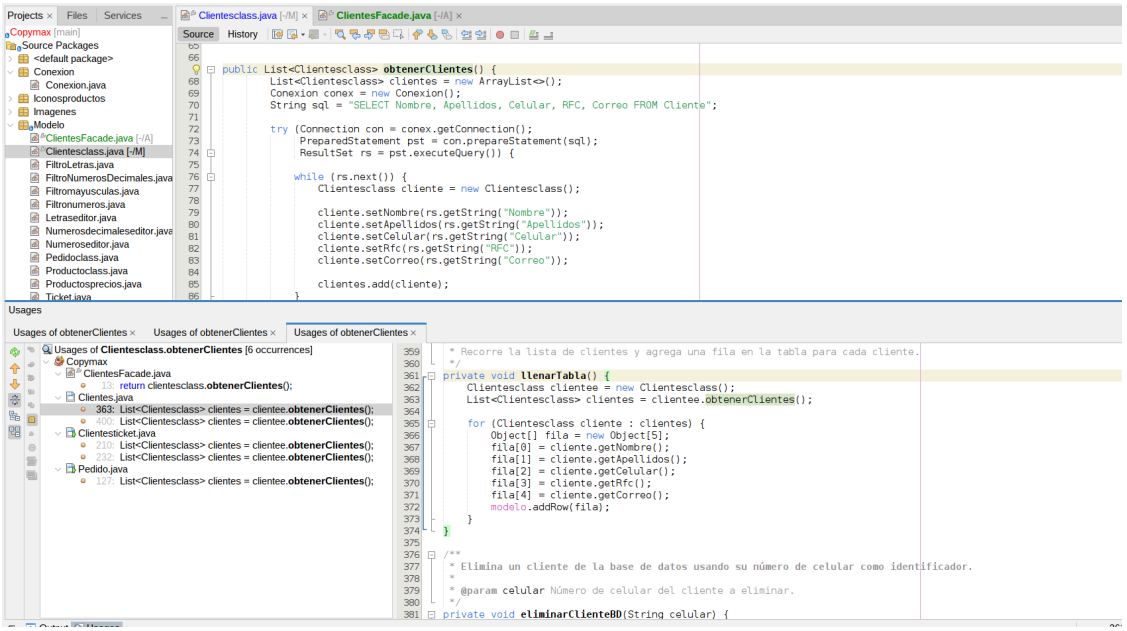
```
    }

    // Método que obtiene clientes por ID y nombre
    public List<Clientesclass> obtenerClientesidnombre() {
        return clientesclass.obtenerClientesidnombre();
    }
}
```



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.

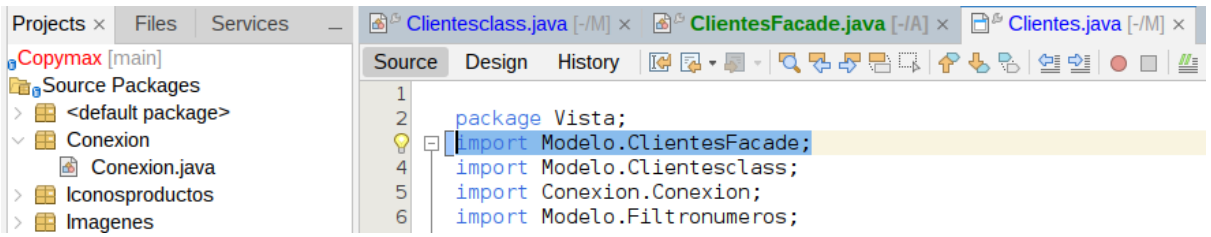
Lo que debemos hacer es implementar el Facade en nuestro programa, esto quiere decir que vamos a cambiar los métodos viejos por los nuevos, ya que facade sería nuestra interfaz simplificada que sería nuestra “intermedia” entre el cliente y la obtención de Clientes.

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

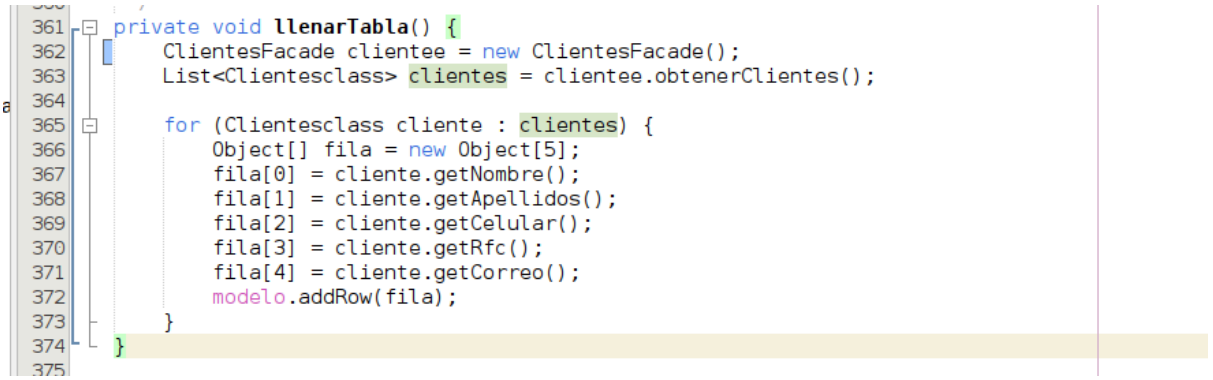


Ahi debemos de usar nuestra facade de la siguiente forma:

Importamos nuestra clase de facade al inicio de la clase.



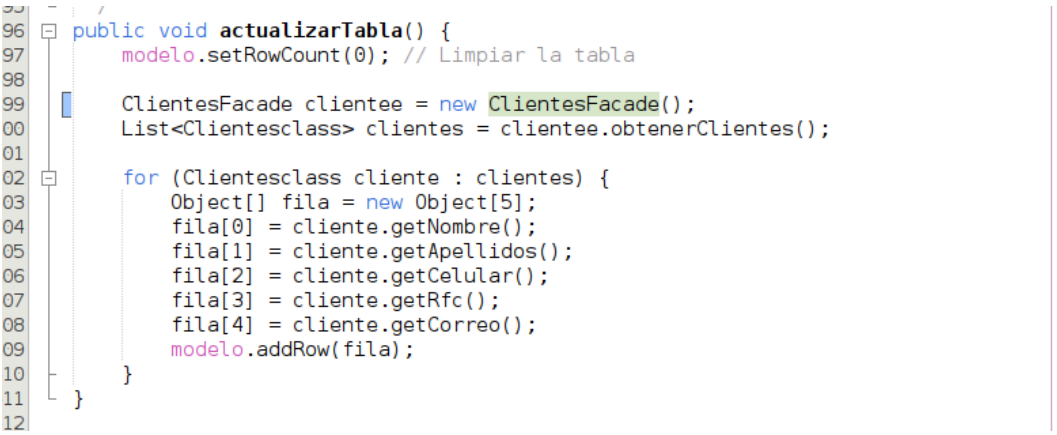
Y modificamos nuestro método donde se utilizaba el viejo método:



```
private void llenarTabla() {
    ClientesFacade clientee = new ClientesFacade();
    List<Clientesclass> clientes = clientee.obtenerClientes();

    for (Clientesclass cliente : clientes) {
        Object[] fila = new Object[5];
        fila[0] = cliente.getNombre();
        fila[1] = cliente.getApellidos();
        fila[2] = cliente.getCelular();
        fila[3] = cliente.getRfc();
        fila[4] = cliente.getCorreo();
        modelo.addRow(fila);
    }
}
```

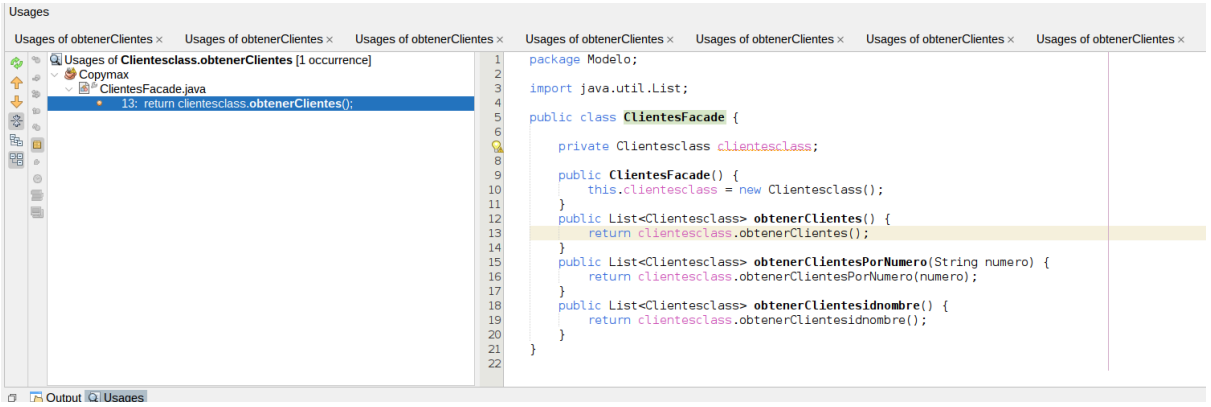
Usamos la nueva Fachada de ClientesFacade.

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



```
public void actualizarTabla() {
    modelo.setRowCount(0); // Limpiar la tabla

    ClientesFacade clientee = new ClientesFacade();
    List<Clientesclass> clientes = clientee.obtenerClientes();

    for (Clientesclass cliente : clientes) {
        Object[] fila = new Object[5];
        fila[0] = cliente.getNombre();
        fila[1] = cliente.getApellidos();
        fila[2] = cliente.getCelular();
        fila[3] = cliente.getRfc();
        fila[4] = cliente.getCorreo();
        modelo.addRow(fila);
    }
}
```

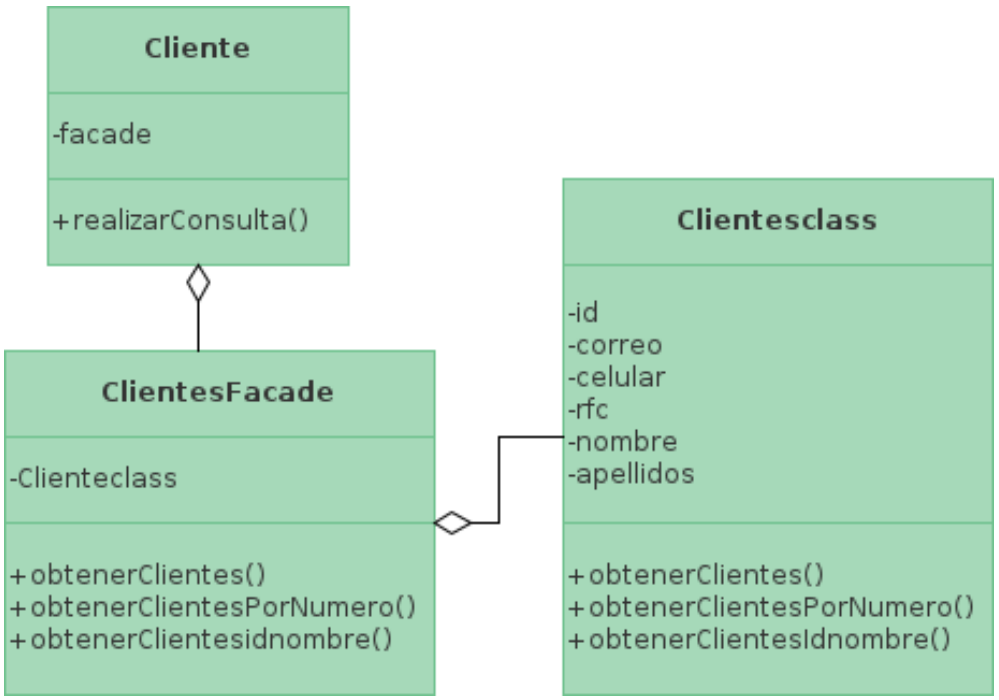
Al final nuestro método de Cliente únicamente será ocupado por nuestra clase “intermediaria ” Facade que la usara para obtener datos, pero no será ocupada por ninguna otra clase:



¿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.	