

PROXY

Proxy es un patrón de diseño estructural que te permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

Aplicabilidad

Hay decenas de formas de utilizar el patrón Proxy. Repasemos los usos más populares.

Inicialización diferida (proxy virtual). Es cuando tienes un objeto de servicio muy pesado que utiliza muchos recursos del sistema al estar siempre funcionando, aunque solo lo necesites de vez en cuando.

En lugar de crear el objeto cuando se lanza la aplicación, puedes retrasar la inicialización del objeto a un momento en que sea realmente necesario.

Control de acceso (proxy de protección). Es cuando quieres que únicamente clientes específicos sean capaces de utilizar el objeto de servicio, por ejemplo, cuando tus objetos son partes fundamentales de un sistema operativo y los clientes son varias aplicaciones lanzadas (incluyendo maliciosas).

El proxy puede pasar la solicitud al objeto de servicio tan sólo si las credenciales del cliente cumplen ciertos criterios.

Ejecución local de un servicio remoto (proxy remoto). Es cuando el objeto de servicio se ubica en un servidor remoto.

En este caso, el proxy pasa la solicitud del cliente por la red, gestionando todos los detalles desagradables de trabajar con la red.

Solicitudes de registro (proxy de registro). Es cuando quieres mantener un historial de solicitudes al objeto de servicio.

El proxy puede registrar cada solicitud antes de pasarla al servicio.

Resultados de solicitudes en caché (proxy de caché). Es cuando necesitas guardar en caché resultados de solicitudes de clientes y gestionar el ciclo de vida de ese caché, especialmente si los resultados son muchos.

El proxy puede implementar el caché para solicitudes recurrentes que siempre dan los mismos resultados. El proxy puede utilizar los parámetros de las solicitudes como claves de caché.

Referencia inteligente. Es cuando debes ser capaz de desechar un objeto pesado una vez que no haya clientes que lo utilicen.

El proxy puede rastrear los clientes que obtuvieron una referencia del objeto de servicio o sus resultados. De vez en cuando, el proxy puede recorrer los clientes y comprobar si siguen activos. Si

la lista del cliente se vacía, el proxy puede desechar el objeto de servicio y liberar los recursos subyacentes del sistema.

El proxy también puede rastrear si el cliente ha modificado el objeto de servicio. Después, los objetos sin cambios pueden ser reutilizados por otros clientes.

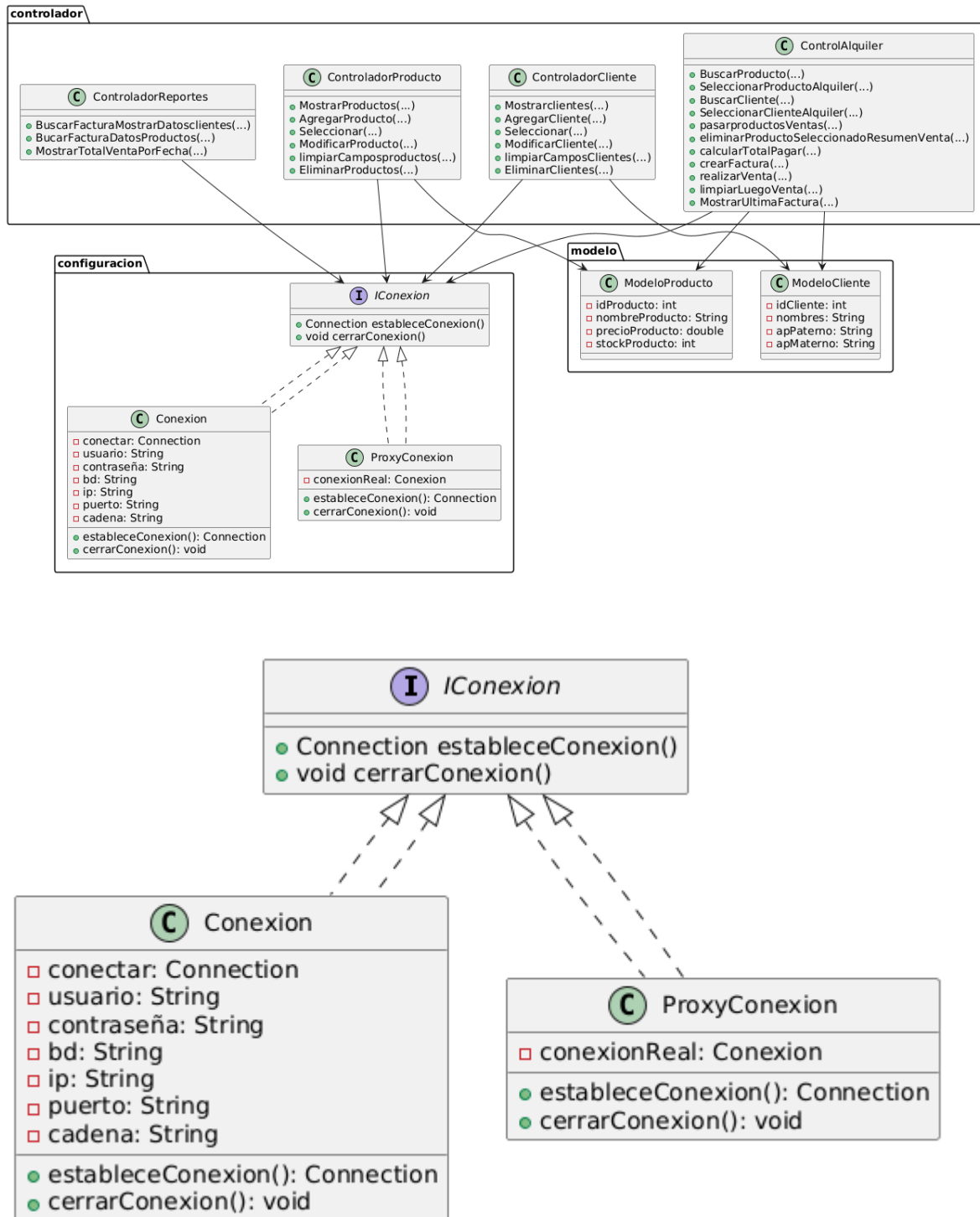
Cómo implementarlo

1. Si no hay una interfaz de servicio preexistente, crea una para que los objetos de proxy y de servicio sean intercambiables. No siempre resulta posible extraer la interfaz de la clase servicio, porque tienes que cambiar todos los clientes del servicio para utilizar esa interfaz. El plan B consiste en convertir el proxy en una subclase de la clase servicio, de forma que herede la interfaz del servicio.
2. Crea la clase proxy. Debe tener un campo para almacenar una referencia al servicio. Normalmente los proxies crean y gestionan el ciclo de vida completo de sus servicios. En raras ocasiones, el cliente pasa un servicio al proxy a través de un constructor.
3. Implementa los métodos del proxy según sus propósitos. En la mayoría de los casos, después de hacer cierta labor, el proxy debería delegar el trabajo a un objeto de servicio.
4. Considera introducir un método de creación que decida si el cliente obtiene un proxy o un servicio real. Puede tratarse de un simple método estático en la clase proxy o de todo un método de fábrica.
5. Considera implementar la inicialización diferida para el objeto de servicio.

Pros y contras

- Puedes controlar el objeto de servicio sin que los clientes lo sepan.
- Puedes gestionar el ciclo de vida del objeto de servicio cuando a los clientes no les importa.
- El proxy funciona incluso si el objeto de servicio no está listo o no está disponible.
- *Principio de abierto/cerrado.* Puedes introducir nuevos proxies sin cambiar el servicio o los clientes.
- El código puede complicarse ya que debes introducir gran cantidad de clases nuevas.
- La respuesta del servicio puede retrasarse.

IMPLEMENTACION DEL PATRON EN EL PROYECTO DE ALQUILER UML



Propósito del Proxy:

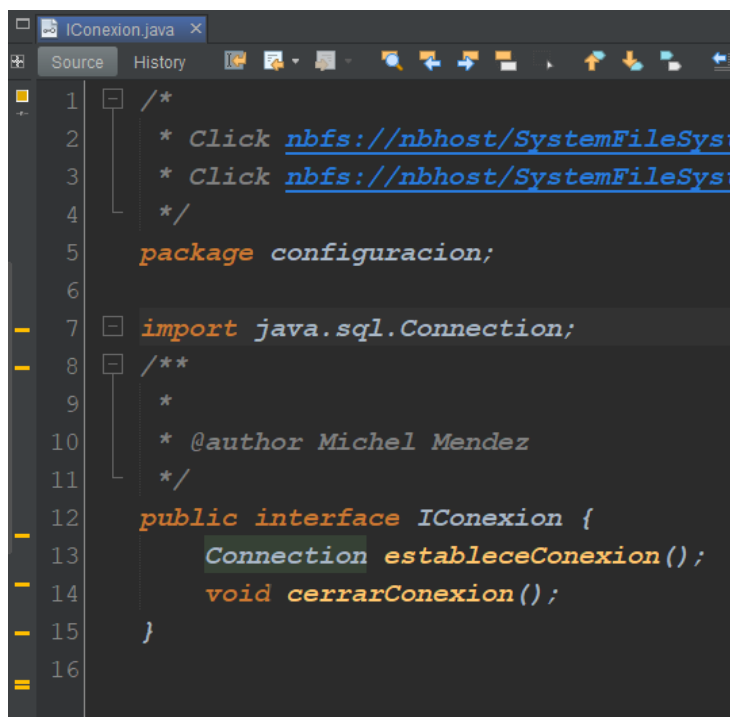
Controlar quién tiene permiso de conectarse a la base de datos, según el tipo de usuario (admin o empleado).

Cómo lo implementaste:

1. Creamos una interfaz IConexion
Define los métodos comunes que deben tener tanto la conexión real como el proxy (estableceConexion, cerrarConexion).
2. Implementamos la clase real Conexion
Se encarga de conectarse y desconectarse realmente de la base de datos.
3. implementamos ProxyConexion, el intermediario
Antes de conectarse, verifica si el usuario está
- 4.
5. ¿está autenticado:
 - Si no lo está → bloquea el acceso.
 - Si sí lo está → deja pasar y conecta.
6. Usamos el Proxy en el Login
Según el usuario que inicia sesión (admin o empleado), se crea un objeto ProxyConexion con permiso o sin permiso:

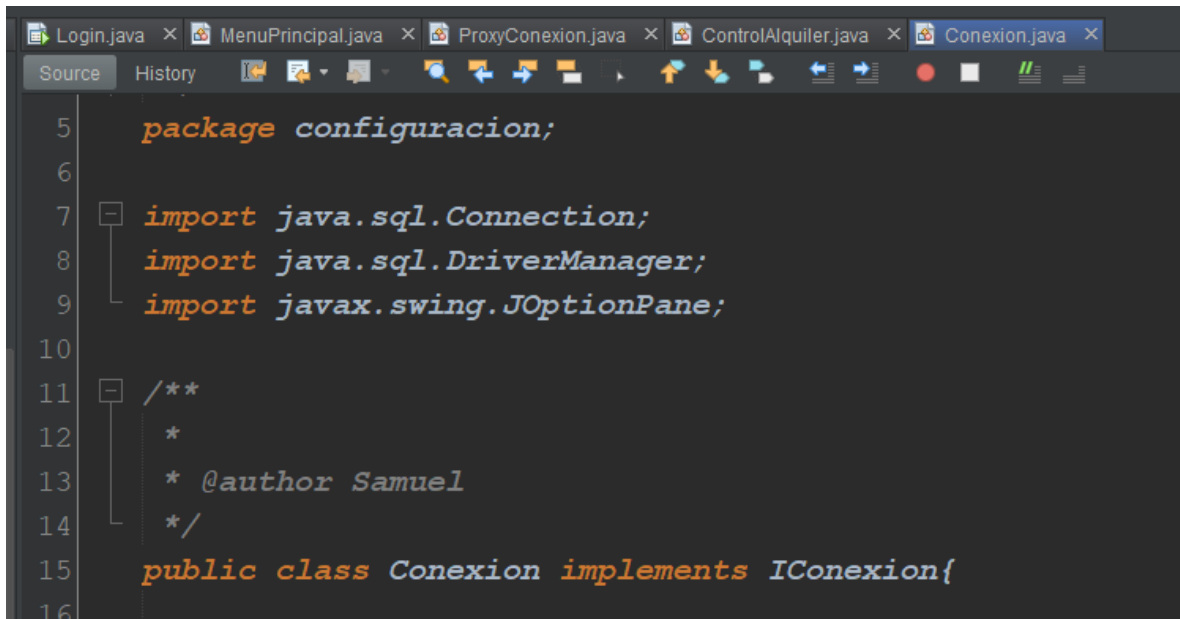
```
IConexion conexion = new ProxyConexion(true); //acceso permitido
```

IMPLEMENTACION DEL PATRON EN EL CODIGO



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem
3   * Click nbfs://nbhost/SystemFileSystem
4   */
5   package configuracion;
6
7   import java.sql.Connection;
8   /**
9    *
10   * @author Michel Mendez
11   */
12   public interface IConexion {
13       Connection estableceConexion();
14       void cerrarConexion();
15   }
16
```

Se creo una interfaz común. Esto permite que tanto el proxy como la clase real se comporten igual desde el punto de vista del cliente.



```
5 package configuracion;
6
7 import java.sql.Connection;
8 import java.sql.DriverManager;
9 import javax.swing.JOptionPane;
10
11 /**
12  *
13  * @author Samuel
14  */
15 public class Conexion implements IConexion{
16
```

Esta clase real que se conecta a la base de datos, es la que hace el trabajo real, pero no controla el acceso.

```
5 package configuracion;
6 import java.sql.Connection;
7 import javax.swing.JOptionPane;
8 /**
9  *
10  * @author Michel Mendez
11  */
12 public class ProxyConexion implements IConexion {
13     private Conexion conexionReal;
14     private boolean autenticado;
15
16     public ProxyConexion(boolean autenticado) {
17         this.autenticado = autenticado;
18     }
19     @Override
20     public Connection estableceConexion() {
21         if (!autenticado) {
22             JOptionPane.showMessageDialog(parentComponent: null, message: "Acceso denegado: usuario no autenticado");
23             return null;
24         }
25         if (conexionReal == null) {
26             conexionReal = new Conexion();
27         }
28         return conexionReal.estableceConexion();
29     }
30     @Override
31     public void cerrarConexion() {
32         if (conexionReal != null) {
33             conexionReal.cerrarConexion();
34         }
35     }
36 }
```

Esta clase controla el acceso antes de permitir la conexión real, si el usuario no esta autenticado no entra y si si, abra conexión.

```
17 import configuracion.IConexion;
18 import configuracion.ProxyConexion;
19
20 /**
21  *
22  * @author Samuel
23  */
24 public class ControlAlquiler {
25
26     public void BuscarProducto(JTextField nombreProducto, JTable tablaproducto){
27
28         //configuracion.Conexion objetoConexion = new configuracion.Conexion();
29         IConexion objetoConexion = new ProxyConexion( autenticado: true);
30     }
```

Usamos proxy en vez de la conexión directa, aquí decidimos si el usuario tiene permiso o no.

```
330 // Verificar si el usuario es administrador
331 boolean esAdmin = usuario.equals( anObject: "admin");
332 // Crear la conexión a través del Proxy
333 configuracion.IConexion conexion = new configuracion.ProxyConexion( autenticado: true);
334
```

Esto lo hacemos en el login

REFERENCIAS

Proxy. (s. f.). <https://refactoring.guru/es/design-patterns/proxy>

Observer. (s. f.). <https://refactoring.guru/es/design-patterns/observer>