Taller 5 – Patrones de Diseño

Thomas Bonnett – 202225550

DPOO – Sec. 3

URL: https://github.com/mehboobali98/Protection-Proxy-Design-Pattern.git

Propósito del Proyecto

El proyecto tiene como objetivo implementar el patrón de diseño "Proxy" proporcionando una capa de protección adicional al sistema verificando el tipo de usuario que intenta acceder a la información de la aplicación.

Estructura del proyecto

La estructura general del proyecto sigue una arquitectura basada en el patrón de diseño “Proxy”. El sistema se encarga de gestionar el acceso a cierta información y garantiza que solo los usuarios autorizados tengan permiso para interactuar con ellos. En este caso, la acción que realiza el proxy es una validación del tipo de empleado que intenta acceder a la información real, protegiendo el sistema y a su vez, evitando que algún error en el código permita que cualquier usuario pueda acceder a la información que solo debe ser visible para los empleados de tipo “Admin”. Sin embargo, la explicación detallada de la aplicación se hará al analizar la aplicación del patrón en el proyecto

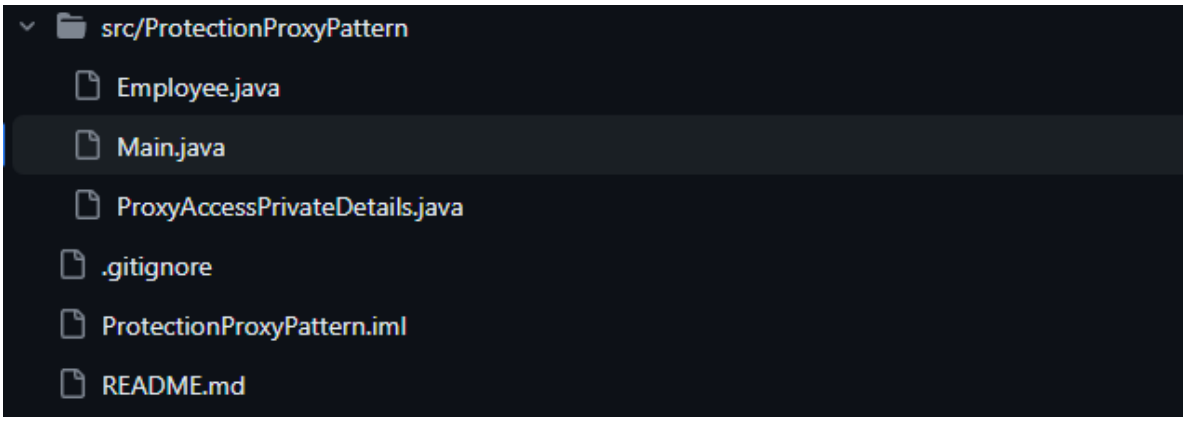

Imagen 1. Vista modo folder del proyecto estudiado.

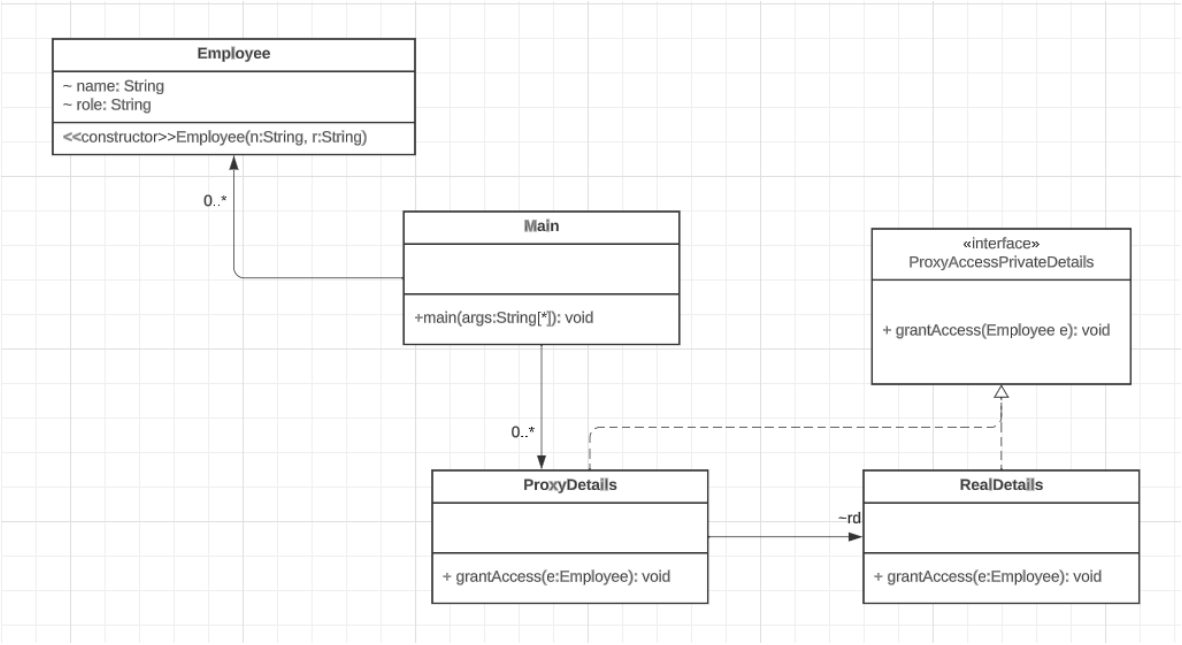

Imagen 2. Diagrama de clases del proyecto.

Retos de Diseño

El principal reto de diseño en este proyecto es la implementación de una capa de protección eficiente que evite accesos no autorizados a la información. La gestión de permisos y la autenticación adecuada son aspectos cruciales para garantizar la seguridad e integridad de un sistema.

Propósito del Patrón

El patrón de diseño "Proxy" se utiliza para controlar el acceso a un objeto. Introduce un sustituto o intermediario (proxy) que actúa como un representante del objeto real y gestiona las solicitudes de acceso. Un proxy controla el acceso al objeto original, permitiendo realizar acciones antes o después de que la solicitud llegue al objeto original.

Patrón Aplicado al Proyecto

En este proyecto, el patrón "Proxy" se utiliza para proteger ciertos objetos del acceso no autorizado de ciertos empleados. El proxy actúa como un intermediario entre la clase Employee y la clase RealDetails, verificando la autenticidad del usuario (employee) antes de permitir el acceso a la información real.

```java
package ProtectionProxyPattern;

public class Main {

    public static void main(String[] args) {
        // write your code here
        Employee e1 = new Employee("Mehboob", "Admin");
        Employee e2 = new Employee("Omer", "MD");

        ProxyAccessPrivateDetails papd = new ProxyDetails();

        papd.grantAccess(e1);

        papd.grantAccess(e2);
    }
}
```

Imagen 3. Código del archivo Main.java.

En la clase Main, se puede apreciar como se crean dos instancias de la clase Employee con su información correspondiente, los cuales para efectos prácticos difieren en la información que responde a su rol. Luego de crear estos objetos, se crea un objeto de la clase ProxyDetails y se comprueba si ambos empleados pueden tener acceso a la información del sistema.

```java
package ProtectionProxyPattern;

public class Employee {
    String name;
    String role;

    Employee(String n, String r)
    {
        name = n;
        role = r;
    }
}
```

Imagen 4. Código clase Employee.java.

La clase Employee contiene la información básica del empleado, siendo esta el nombre y el rol de este dentro del sistema.

```java
package ProtectionProxyPattern;

import java.security.cert.PolicyNode;

public interface ProxyAccessPrivateDetails {
    void grantAccess(Employee e);
}

class RealDetails implements ProxyAccessPrivateDetails {

    public void grantAccess(Employee e)
    {
        System.out.println("Welcome to the chamber of secrets " +
                "" +
                "" + e.name);
    }

}
class ProxyDetails implements ProxyAccessPrivateDetails{
    RealDetails rd;

    public void grantAccess(Employee e)
    {
        if(e.role.equalsIgnoreCase("Admin"))
        {
            rd = new RealDetails();
            rd.grantAccess(e);
        } else {
            System.out.println("You can't access it.");
        }

    }
}
```

Imagen 5. Código del archivo ProxyAccessPrivateDetails.java.

La clase ProxyDetails, la cual implementa la interfaz ProxyAccessPrivateDetails, es la que implementa el patrón proxy y contiene un único método el cual recibe por parámetro un objeto de tipo empleado para así, verificar si su rol dentro del sistema le permite acceder a la información real de la aplicación. De ser un empleado con un rol “Admin”, se le dará acceso a la clase RealDetails y, por ende, podrá ver su información, de lo contrario, al no tener un rol adecuado dentro del sistema se le informará al empleado que no tiene permitido acceder a la información de la clase RealDetails.

Ventajas de Utilizar el Patrón en este Proyecto

1. **Control de Acceso:** Este patrón permite implementar un control de acceso eficiente, asegurando que solo usuarios autorizados puedan interactuar con el objeto real.

2. **Reutilización:** Al utilizar un proxy, se facilita la reutilización del código, ya que el proxy puede ser aplicado a diferentes objetos sin modificar su lógica interna.

Desventajas de Utilizar el Patrón en Este Punto del Proyecto

1. **Complejidad Adicional:** La introducción de un proxy puede agregar complejidad al diseño general del sistema.
2. **Disminución del rendimiento:** El proceso de autenticación adicional introducido por el proxy puede tener un impacto en el rendimiento del sistema. Sin embargo, al ser un sistema tan pequeño, en este contexto no representa un factor determinante en el rendimiento de este.

Otras posibles soluciones

1. En lugar del patrón Proxy, se pudo haber utilizado el patrón Decorator para agregar funcionalidades adicionales al objeto original según lo requiera el sistema, pero sin cambiar su interfaz.
2. Se pudo haber utilizado el patrón de comportamiento Strategy para cambiar dinámicamente el comportamiento del objeto en tiempo de ejecución protegiendo la información de una manera similar a la que lo haría un proxy.