Nombres de los integrantes del grupo

Edú Ramírez Campos

Eliacid Castillo Rojas

Esteban Jesús Porras Villalobos

Patrón Propuesto

Proxy

Qué es y en qué consiste el patrón

El patrón Proxy es un concepto clave en el diseño de software. Se trata de un patrón de diseño que se utiliza para controlar el acceso a un objeto mediante la introducción de un intermediario entre el cliente y el objeto real. Este intermediario, llamado Proxy, actúa como representante del objeto real y puede realizar acciones adicionales antes o después de que la solicitud del cliente llegue al objeto real. El propósito principal del patrón Proxy es controlar el acceso al objeto real para diversos fines, como seguridad, rendimiento, encapsulamiento y manejo de excepciones.

El patrón Proxy se compone de tres elementos principales: el Sujeto, que es el objeto real que se está representando; el Proxy, que es el objeto intermediario que controla el acceso al Sujeto; y una Interfaz común que define las operaciones que tanto el Proxy como el Sujeto pueden realizar. El cliente interactúa con el Proxy, que a su vez delega las operaciones al Sujeto a través de la interfaz común.

Existen diferentes tipos de Proxies, cada uno con un propósito específico:
Proxy remoto: Representa un objeto que se encuentra en una ubicación diferente, generalmente en una red.
Proxy virtual: Crea un objeto "virtual" que no se materializa hasta que es necesario.
Proxy de protección: Controla el acceso a un objeto real para garantizar la seguridad.
Proxy de caché: Almacena los resultados de operaciones costosas para un acceso más rápido posterior.

Qué resuelve el patrón

El patrón Proxy resuelve varios problemas comunes en el diseño de software:

Control de acceso: Permite restringir el acceso al objeto real, lo que resulta útil para implementar políticas de seguridad, mejorar el rendimiento o garantizar el encapsulamiento adecuado.

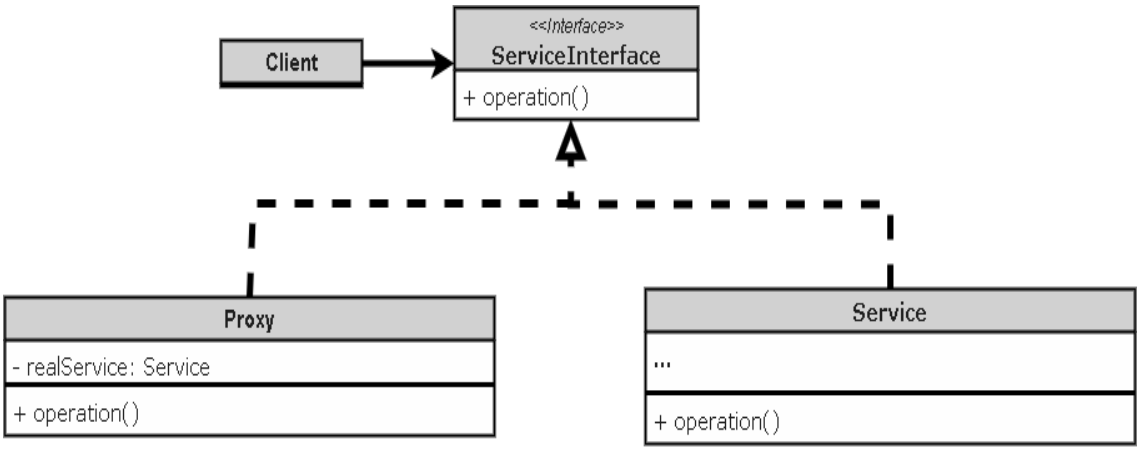
Mejora del rendimiento: Actúa como una capa de caché para almacenar los resultados de operaciones costosas, lo que permite un acceso más rápido posteriormente. Es útil en aplicaciones donde las operaciones costosas se realizan con frecuencia.

Virtualización: Permite la creación de objetos "virtuales" que no se materializan hasta que son necesarios. Esto es útil en situaciones donde la creación de objetos es costosa en términos de recursos o tiempo.

Protección contra errores: Intercepta llamadas a objetos que pueden fallar y maneja las excepciones de manera transparente. Esto ayuda a mejorar la robustez y la confiabilidad del sistema al evitar que los errores no controlados afecten al cliente directamente.

Modularidad y desacoplamiento: Permite desacoplar el cliente del objeto real, mejorando la flexibilidad y la mantenibilidad del código. Esto facilita la introducción de cambios y mejoras en el sistema sin afectar a otras partes del código.

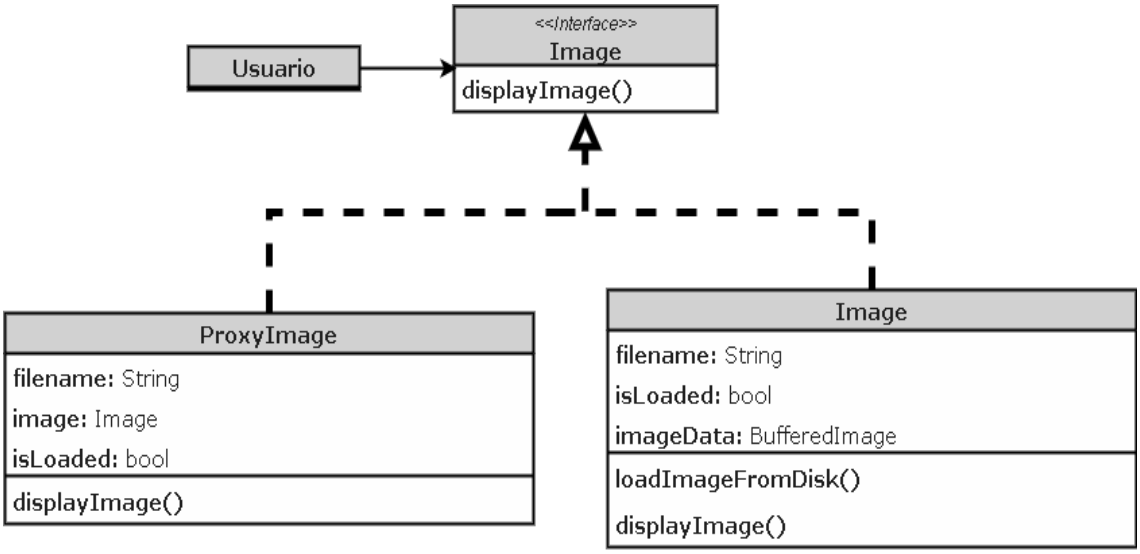
Diagrama original del patrón:



Contexto Propuesto:

En un determinado sistema, se desea visualizar imágenes que pueden llegar a ser muy grandes y afectar al rendimiento del sistema, ya que, cada vez que una imagen se quiera visualizar, esta debe ser cargada desde el disco. Por lo que se necesita hacer uso de un caché de imágenes, este permitirá cargar las imágenes desde el disco la primera vez que se acceden, las almacena en caché para accesos posteriores. De esta manera, se reduce la carga en el servidor y se mejora la velocidad de acceso para el usuario.

Diagrama del patrón propuesto en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

ImageInterface (Interfaz ImageInterface):

Esta interfaz define el contrato para las clases que representan imágenes. Contiene el método displayImage(), que se encarga de mostrar la imagen.

Image (Clase Image):

Esta clase representa una imagen e implementa la interfaz ImageInterface. Tiene campos para almacenar el nombre del archivo de la imagen, el estado de carga (isLoaded), y los datos de la imagen en sí (imageData). El constructor toma el nombre del archivo de la imagen y carga los datos de la imagen desde el disco utilizando el método loadImageFromDisk(). El método loadImageFromDisk() carga los datos de la imagen desde el disco utilizando la clase ImageIO. El método displayImage() muestra la imagen en una ventana utilizando JFrame y JLabel.

ProxyImage (Clase ProxyImage):

Esta clase actúa como un proxy para la clase Image, implementando la interfaz ImageInterface. Tiene campos para almacenar el nombre del archivo de la imagen (filename), una instancia de Image (image), y un indicador de si la imagen ha sido cargada (isLoaded). El constructor toma el nombre del archivo de la imagen, pero no carga los datos de la imagen en ese momento. El método displayImage() verifica si la imagen ya ha sido cargada. Si no lo ha sido, crea una instancia de Image y carga la imagen utilizando el método displayImage() de esa instancia. El método loadImageFromDisk() no se encuentra en la clase ProxyImage ya que se utiliza exclusivamente en la clase Image para evitar la sobrecarga de cargar la imagen desde el

disco cada vez. El propósito del ProxyImage es cargar la imagen solo una vez y luego proporcionar acceso a ella mediante la delegación a la clase Image.

Usuario (Clase Usuario):

Esta clase representa a un usuario que desea ver una imagen. Tiene un campo para almacenar una instancia de ImageInterface. El constructor toma una instancia de ImageInterface como parámetro. El método verImagen() llama al método displayImage() de la instancia de ImageInterface para mostrar la imagen. El método setProxyImage() permite cambiar o establecer manualmente la ProxyImage que el usuario va a visualizar.

ProxyExample (Clase ProxyExample):

Actúa como ejecutor del sistema.

Ejemplo de implementación bajo el lenguaje de programación Java:

```
interface ImageInterface {
    void displayImage();
}
```

```java
class Image implements ImageInterface {
    private String filename;
    private boolean isLoaded;
    private BufferedImage imageData;

    public Image(String filename) {
        this.filename = filename;
        this.isLoaded = false;
        this.imageData = null;

        loadImageFromDisk();
    }

    private void loadImageFromDisk() {
        System.out.println("Loading " + filename + "...");
        try {
            imageData = ImageIO.read(getClass().getResource(filename));
            isLoaded = true;
        } catch (IOException e) {
            System.out.println("Error loading image: " + e.getMessage());
        }
    }

    public void displayImage() {
        if (!isLoaded) {
            loadImageFromDisk();
        }

        System.out.println("Displaying " + filename);
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new JLabel(new ImageIcon(imageData)));
        frame.pack();
        frame.setVisible(b:true);
    }
}
```

```
class ProxyImage implements ImageInterface {
    private String filename;
    private Image image;
    private boolean isLoaded;

    public ProxyImage(String filename) {
        this.filename = filename;
        this.image = null;
        this.isLoaded = false;
    }

    public void displayImage() {
        if (!isLoaded) {
            image = new Image(filename);
            isLoaded = true;
        }

        image.displayImage();
    }
}
```

```
class Usuario {
    private ImageInterface proxyImage;

    public Usuario(ImageInterface proxyImage) {
        this.proxyImage = proxyImage;
    }

    public void verImagen() {
        proxyImage.displayImage();
    }

    public void setProxyImage(ImageInterface proxyImage) {
        this.proxyImage = proxyImage;
    }
}
```

```java
public class ProxyExample {
    Run | Debug
    public static void main(String[] args) {
        ImageInterface proxyImage1 = new ProxyImage(filename:"imgs/UtnLogo.png");
        ImageInterface proxyImage2 = new ProxyImage(filename:"imgs/ITI_Logo.png");

        Usuario usuario1 = new Usuario(proxyImage1);
        usuario1.verImagen();

        Usuario usuario2 = new Usuario(proxyImage2);
        usuario2.verImagen();

        usuario2.setProxyImage(proxyImage1);
        usuario2.verImagen();
    }
}
```


Refactoring.Guru. "Proxy Design Pattern." Refactoring.Guru, https://refactoring.guru/design-patterns/proxy.

Eric Freeman, Elisabeth Robson. Head First Design Patterns, 2nd Edition(December 2020), O'Reilly Media, Inc.

Kamon Ayeva, Sakis Kasampalis. Mastering Python Design Patterns - Second Edition(August 2018), Packt Publishing

El documento debe ser entregado en formato PDF bajo el nombre IngSW_EquipoX.PDF donde X corresponde al número de equipo asignado.