Proxy pattern:

- **Información general del proyecto:**

  - **¿Para qué sirve?:**

El proyecto hace la suposición de que tenemos un objeto Book el cual contiene una cadena larga, necesitando a la vez realizar algunos cálculos intensivos al crearlo. Para mejorar el rendimiento, utiliza un proxy para acceder al objeto real del libro, creándolo así solo cuando realmente se necesita (es algo similar al lazy loading).

Al utilizar Proxy para controlar el acceso a objetos y agregar funcionalidades adicionales (como control de acceso y generación de datos aleatorios) podemos ver funcionalidades que son claros ejemplos de cómo se puede utilizar dicho patrón en situaciones en las que se requiere un mayor control sobre el acceso a objetos o se desea agregar funcionalidades adicionales sin modificar directamente la clase original.

  - **¿Cuál es la estructura general del diseño?:**

La estructura general del diseño sigue el patrón Proxy, donde los proxies actúan como intermediarios entre el cliente y los objetos reales, controlando el acceso y proporcionando funcionalidades adicionales según sea necesario. Podríamos dividirlo bajo los siguientes principios:

1. **Interfaz del Sujeto:** En este caso, las interfaces "BookInterface" y "PrinterInterface" definen los métodos que deben ser implementados por las clases reales ("Book" y "Printer", respectivamente) y los proxies ("BookProxy" y "PrinterProxy", respectivamente).
2. **Sujeto Real:** Las clases "Book" y "Printer" representan los objetos reales con los que se interactúa. En el caso del "Book", contiene el contenido del libro y realiza cálculos intensivos en su inicialización, mientras que "Printer", imprime los datos recibidos.
3. **Proxy:** Las clases "BookProxy" y "PrinterProxy" actúan como intermediarios entre el cliente y los objetos reales. Estas implementan las mismas interfaces que los objetos reales ("BookInterface" y "PrinterInterface") y controlan de esta forma el acceso a estos. En el caso del "BookProxy", se crea el objeto "Book" solo cuando se solicita y se proporciona un acceso controlado a sus métodos, mientras que con "PrinterProxy" se verifica el rol del usuario antes de permitir la impresión y se muestra un mensaje de acceso denegado si no se tienen los permisos adecuados.
4. **Cliente:** El cliente interactúa con el objeto "BookProxy" y "PrinterProxy" en lugar de los objetos reales directamente. El cliente realiza las operaciones y solicita la información necesaria a través de los proxies.

  - **¿Qué grandes retos de diseño enfrenta?:**

Algunos de los grandes retos de diseño que se enfrentan en este código son los siguientes:

1. **Control de acceso y seguridad:** Uno de los desafíos es garantizar un control adecuado de acceso y seguridad en el uso de los proxies. En el caso del "PrinterProxy," se verifica el rol del usuario antes de permitir la impresión, sin embargo, es importante asegurarse de que

los roles y permisos estén correctamente implementados y que no se produzcan vulnerabilidades de seguridad.

2. **Coherencia y sincronización:** Si varios clientes acceden simultáneamente al mismo proxy, puede haber problemas de coherencia y sincronización. En consecuencia, es necesario garantizar que la creación y acceso al objeto real ("Book" en el caso del "BookProxy") se maneje de manera segura y sin conflictos, para evitar así resultados incoherentes o errores.
3. **Eficiencia y rendimiento:** Dado que el objetivo principal del uso de proxies en este código es mejorar el rendimiento, es importante diseñar el proxy de manera eficiente para minimizar el impacto en el rendimiento general del sistema. Esto incluye asegurarse de que la creación del objeto real se realice solo cuando sea necesario y que los cálculos intensivos se realicen de manera eficiente.
4. **Escalabilidad:** El diseño debe ser escalable para manejar una mayor carga de trabajo y un mayor número de solicitudes de los clientes. Esto implica tener en cuenta consideraciones de rendimiento, uso eficiente de recursos y gestión adecuada de la concurrencia.

- **URL del proyecto:**

https://github.com/kan01234/design-patterns/tree/master/proxy-pattern

- **Información y estructura del fragmento del proyecto donde aparece el patrón.**

El fragmento del proyecto donde se implementa el patrón Proxy consta de varias clases que interactúan entre sí para lograr el objetivo deseado:

1. **"Book":** Esta clase representa el objeto real del libro. Tiene atributos como "numOfChapter" (número de capítulos) y "content" (contenido del libro), además tiene métodos getter y setter para acceder y modificar estos atributos.
2. **"BookInterface":** Es una interfaz que define los métodos necesarios para acceder al contenido del libro, el número de capítulos y el número de páginas.
3. **"BookProxy":** Actúa como un proxy para el objeto "Book". Este implementa la interfaz "BookInterface" y tiene un atributo "book" que representa el objeto real "Book". En los métodos "getContent()", "getNumOfChapter()" y "getNumOfPage()" verifica si el objeto "book" es nulo, de ser así, crea una nueva instancia de "Book" con el contenido proporcionado en su construcción. Luego, devuelve la información solicitada del objeto book, esto permite crear el objeto Book solo cuando se necesita acceder a su contenido, lo que mejora el rendimiento al evitar la creación innecesaria de objetos.
4. **"Printer":** Esta clase representa el objeto real de la impresora. Tiene un método "print()" que recibe los datos a imprimir y los muestra por la consola.
5. **"PrinterInterface":** Es una interfaz que define el método "print()" para la impresora.
6. "**PrinterProxy":** Actúa como un proxy para la clase "Printer". Básicamente implementa la interfaz "PrinterInterface" y tiene un atributo "printer" que representa el objeto real "Printer". En el método "print()", verifica si el rol del usuario tiene los permisos adecuados para imprimir, si es así, llama al método para imprimir los datos, de lo contario muestra un mensaje de acceso denegado.
7. **"ProxyPatternTest":** Esta es una clase de prueba que verifica el funcionamiento de los proxies. Tiene métodos de prueba, como "test()", que crea una instancia de "BookProxy" y realiza varias pruebas para asegurarse de que el proxy funcione correctamente. Por otro lado, tiene el método "test2()", que crea una instancia de "PrinterProxy" y realiza pruebas para verificar el control de acceso a la impresora según los roles de usuario.

En conclusión, podríamos decir que el proyecto utiliza el patrón Proxy para controlar el acceso a los objetos reales “Book” y “Printer”, permitiendo la creación de los objetos y proporcionando funcionalidades adicionales, como la verificación de roles en el acceso y la impresión. Mientras que la clase “ProxyPatternTest” se utiliza para probar y validar el funcionamiento de los proxies en diferentes escenarios.

- **Información general sobre el patrón:**

  - **¿Qué patrón es?:**

El patrón que se implementa en el fragmento del proyecto es el patrón Proxy.

  - **¿Para qué se usa usualmente?:**

El patrón Proxy se usa usualmente cuando se desea controlar el acceso a un objeto y agregar funcionalidades adicionales a través de un intermediario. Algunos casos de uso comunes del patrón Proxy son:

1. **Acceso remoto:** El proxy actúa como un representante local de un objeto remoto, permitiendo que el cliente interactúe con el objeto remoto a través del proxy sin conocer su ubicación o detalles de implementación. Esto se utiliza en aplicaciones distribuidas y sistemas cliente-servidor.
2. **Protección de acceso:** El proxy controla el acceso al objeto real y puede imponer restricciones o verificar permisos antes de permitir que el cliente acceda a él. Esto es útil en escenarios donde se requiere seguridad y control de acceso.
3. **Creación perezosa (Lazy loading):** El proxy evita la creación anticipada de objetos costosos en términos de recursos o tiempo de inicialización. Es decir, el objeto real se crea solo cuando se solicita explícitamente a través del proxy, lo que mejora el rendimiento al evitar la creación innecesaria de objetos.
4. **Registro y estadísticas:** El proxy puede realizar un seguimiento de las llamadas al objeto real y recopilar datos para fines de registro, estadísticas o monitoreo. Esto permite analizar el comportamiento y el rendimiento del objeto real.
5. **Caché:** El proxy puede mantener en caché los resultados de operaciones costosas en el objeto real y devolver los resultados almacenados en lugar de volver a calcularlos en cada solicitud, gracias a esto se mejora la eficiencia y la velocidad de respuesta.

- **Información del patrón aplicado al proyecto:**

  - **¿Cómo se está utilizando el patrón dentro del proyecto?:**

El patrón Proxy se utiliza para proporcionar un intermediario (“BookProxy”) entre el cliente y el objeto real (“Book”), este controla el acceso al objeto real y crea una instancia del objeto solo cuando se necesita acceder a su contenido. Esto mejora el rendimiento al evitar la creación innecesaria de objetos y permite agregar funcionalidades adicionales en el proceso de acceso al objeto real. (anteriormente se dio una explicación más detallada de que realiza en cada clase y la relación con el patrón)

  - **¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?, ¿Qué ventajas tiene el haber usado este patrón?:**

Tiene sentido utilizar el patrón Proxy en ese punto del proyecto por varias razones:

1. **Mejora del rendimiento:** Al utilizar el patrón Proxy, se evita la creación innecesaria de objetos costosos en términos de recursos y tiempo de procesamiento. En el caso del objeto "Book", el contenido del libro puede ser grande y realizar cálculos pesados durante su creación, sin embargo al utilizar el "BookProxy", se retrasa la creación del objeto hasta que sea realmente necesario acceder a su contenido. Esto mejora el rendimiento al evitar la inicialización costosa del objeto a menos que se requiera explícitamente.
2. **Control de acceso:** El patrón Proxy permite agregar una capa adicional de control de acceso a los objetos reales. Por ejemplo, en el caso del objeto "Printer", el "PrinterProxy" verifica el rol del usuario antes de permitir la impresión. Esto garantiza que solo los roles autorizados puedan acceder a la funcionalidad de impresión, brindando seguridad y control sobre el acceso a recursos sensibles.
3. **Simplificación de la interfaz:** Este patrón proporciona una interfaz común entre el cliente y los objetos reales, esto permite al cliente interactuar de manera transparente con el proxy sin conocer los detalles internos del objeto real. En el caso del objeto "Book", el cliente solo necesita interactuar con el "BookProxy" a través de la interfaz "BookInterface", lo que simplifica el código y facilita la gestión de los objetos reales.

Al tener esto resultamos en un código más eficiente, seguro y fácil de mantener.

- **¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?:**

Aunque el patrón Proxy ofrece varias ventajas, también puede tener algunas desventajas en ciertos contextos. Algunas posibles desventajas de haber utilizado el patrón Proxy en ese punto del proyecto podrían ser:

1. **Complejidad adicional:** Puede introducir una capa adicional de complejidad en el diseño y la implementación del sistema. Al usar dicho patrón se requiere la creación de clases adicionales para el proxy y la gestión de la interacción entre este y los objetos reales, aumentando la complejidad del código y haciendo que sea más difícil de entender y mantener.
2. **Sobrecarga de memoria:** En algunos casos, el uso del patrón Proxy puede resultar en una sobrecarga de memoria. Por ejemplo, si se crean múltiples instancias de proxies para diferentes objetos reales, esto puede consumir más memoria en comparación con el enfoque directo de acceso a los objetos reales, siendo un inconveniente si el sistema tiene restricciones de memoria o si se utilizan muchos proxies.
3. **Impacto en el rendimiento:** Aunque el patrón Proxy puede mejorar el rendimiento al retrasar la creación de objetos costosos, también puede tener un impacto negativo en el rendimiento en ciertos casos. Si se accede repetidamente al objeto real a través del proxy, se incurre en el costo adicional de verificar y crear el objeto en cada llamada; afectando así el rendimiento en comparación con un enfoque directo de acceso a este.
4. **Complejidad en el manejo de errores:** Puede complicar el manejo de errores ya que si ocurre un error en la creación o en la lógica del proxy, puede ser más difícil determinar la causa raíz y depurar el problema. Esto puede aumentar la complejidad del manejo de errores y dificultar la identificación y resolución de problemas.

- **¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

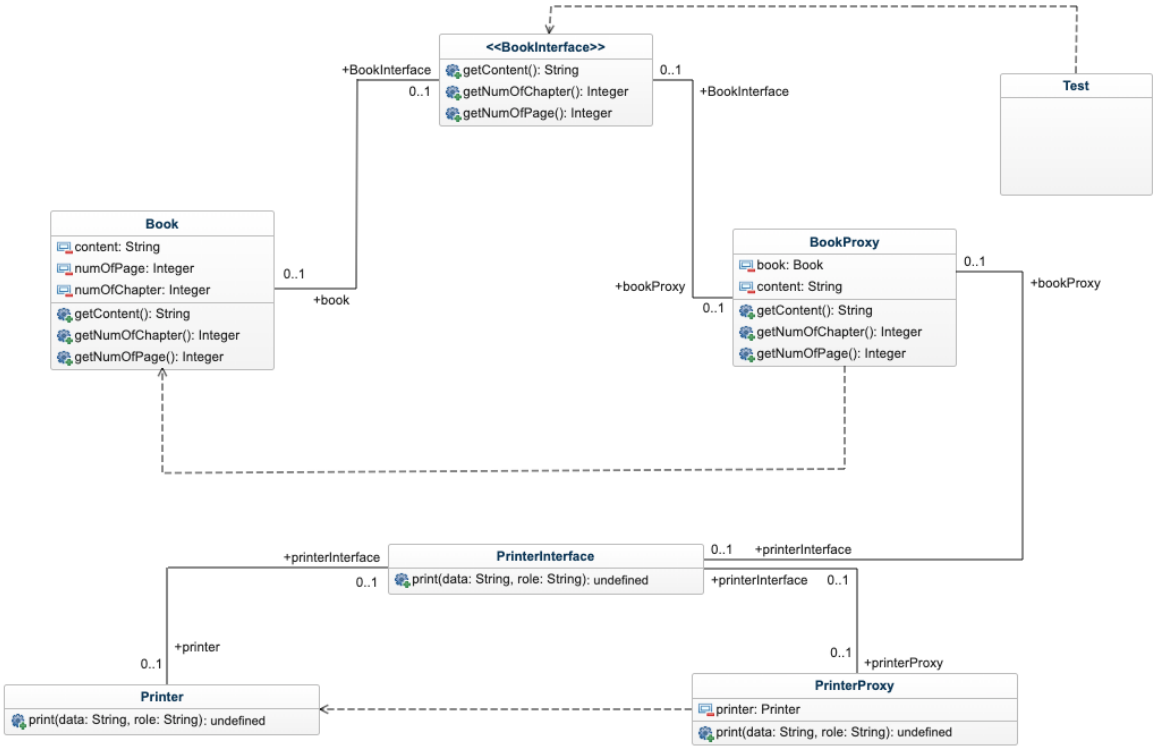
En este caso particular, aparte del patrón Proxy, existen otras formas en las que se podrían haber solucionado los problemas que resuelve el patrón. Algunas alternativas podrían ser:

1. **Lazy initialization (Inicialización perezosa):** En lugar de utilizar un proxy, se podría haber implementado la inicialización perezosa directamente en la clase "Book". Básicamente, en vez inicializar todas las propiedades del libro en el constructor, se podrían haber hecho solo cuando se acceda a ellas por primera vez, esto permitiría cargar el contenido y realizar cálculos pesados solo cuando sea necesario (en lugar de hacerlo de forma anticipada).
2. **Caching (Caché):** En el primer acceso a un libro con contenido determinado debería permitirse crear y almacenar en caché el objeto "Book", y los accesos posteriores podrían obtener el libro desde este en lugar de recrearlo, de esta forma se evitaría la creación innecesaria de múltiples objetos "Book" y mejoraría el rendimiento.
3. **División de responsabilidades:** En lugar de tener una única clase "Book" que maneje tanto el contenido como los cálculos pesados, se podría haber dividido la responsabilidad en clases separadas, permitiendo así una mejor organización y separación de preocupaciones, lo que facilitaría la gestión y el mantenimiento del código.
4. **Mejora de algoritmos:** Mediante técnicas de optimización y análisis de rendimiento, se podría haber logrado reducir el tiempo de ejecución de los cálculos y, en consecuencia, mejorar el rendimiento global del sistema.

Por otro lado, podrían considerarse otros patrones de diseño como alternativas al patrón Proxy. Algunos de los patrones que podrían ser relevantes incluyen:

1. **Factory Pattern:** Este patrón se utiliza para crear objetos sin especificar la clase concreta. En lugar de utilizar un proxy para crear objetos "Book" cuando sea necesario, se podría haber utilizado un Factory para crear instancias de "Book" de forma dinámica y flexible, esto permitiría centralizar la lógica de creación de objetos y proporcionar una forma más modular y extensible de instanciar libros.
2. **Flyweight Pattern:** Este patrón se utiliza para minimizar el uso de memoria al compartir eficientemente objetos que son similares entre sí. En lugar de utilizar un proxy para acceder a objetos "Book" individuales, se podría haber utilizado el patrón Flyweight para almacenar y reutilizar instancias de "Book" que tienen el mismo contenido, esto reduciría la duplicación de memoria y mejoraría el rendimiento en escenarios donde hay varios libros con contenido idéntico.
3. **Strategy Pattern:** Este patrón se utiliza para encapsular diferentes algoritmos en clases separadas y permitir que se intercambien dinámicamente. En lugar de realizar los cálculos pesados directamente en la clase "Book", se podría haber utilizado el patrón Strategy para encapsular diferentes estrategias de cálculo en clases separadas, esto daría la opción de intercambiar y seleccionar dinámicamente la estrategia de cálculo adecuada según sea necesario.

- **Diagrama UML:**



- **Referencias:**

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
2. Shalloway, A., & Trott, J. R. (2004). Design Patterns Explained: A New Perspective on Object-Oriented Design. Addison-Wesley.
3. kan01234. (23 de Mayo 2023). Design Patterns: Proxy Pattern. Recuperado de [https://github.com/kan01234/design-patterns/tree/master/proxy-pattern].

María Paula Estupiñan 202212331