Ejercicio 1

|  |  |
| --- | --- |
| Nombre | Proxy |
| Propósito | Objetos que controlan la creación y acceso a otros objetos. |
| Problema | Crear un sistema de control de visitas, permitiendo entrar a las personas cuando el usuario y contraseña coincidan con los que están en el archivo de texto y admitir realizar diferentes acciones dependiendo del role del usuario |
| Solución | La implementación de este patrón en este problema nos permite tener un objeto que controla el acceso a otro objecto, dando así una seguridad al sistema y obligando a los usuarios a autentificarse. |
| Participantes y colaboradores | Proxy (UserProxy): controla el acceso al sujeto real mediante los datos de nombre de usuario, contraseña y role.  Sujeto (IUser): Interface común para el Proxy y sujeto real, haciendo así que se pueda usar el Proxy cuando se espere un sujeto real.  Sujeto Real (User): Define el objecto real al cual el Proxy limita al acceso a él. |
| Consecuencia | Simplifica la interacción con elementos remotos porque el proxy implementa los mecanismos de comunicación haciéndolos transparentes al cliente, y de esta manera el cliente no se da entera de donde ni como es que se crear el objeto real por lo que se dice que da un nivel de no determinismo de donde se encuentra el objeto real. |
| Implementación | Se crea una interfaz que se implementada por nuestro Proxy y Sujeto real.  Y en nuestro proxy se crea una referencia privada del objeto real, además se crea las validaciones que son las requeridas por nuestro sistema para que podamos acceder solo por medio del Proxy. |

Ejercicio 2

|  |  |
| --- | --- |
| Nombre | Adaptador |
| Propósito | Une tipos que no fueron diseñados para funcionar entre sí, es decir adapta interfaces o abstracciones que no son compatibles o no cumplen los requisitos. |
| Problema | Adaptar código para que el reproductor pueda reproducir música en MP3. |
| Solución | Con la implementación de este patrón logramos adaptar un código que ya se encontraba diseñado, y que ocupábamos una nueva funcionalidad en nuestro reproductor para reproducir música MP3, por lo que se hace un adaptador para que el MP3 pueda ser reproducido. |
| Participantes y colaboradores | Cliente (Principal): colabora y espera con objetos que se ajustan a la interfaz Target.  Target (Reproductor): define la interface específica que se usa en el cliente.  Adaptador (ReproductorMP3Adaptador): implementa Target, usando llamadas al objeto Adaptado  Adaptado (ReproductorMP3): es usado por el adaptador para responder a las solicitudes del Cliente. |
| Consecuencia | Permite adaptar clases en dominios totalmente diferentes, es quiere decir que permite adaptar clases que no son compatibles, pero hay que tener mucho cuidado con la implementación del patrón porque para las subclases de adaptado también podría funcionar este adaptador. |
| Implementación | Para implementar este patrón se tuvo que crear la clase ReproductorMP3Adaptador y que heredara de la clase Reproductor. ReproductorMP3Adaptador tiene una referencia a ReproductorMP3 para que cuando se realicen las llamadas a este objecto. El cliente tiene que crear la instancia del objeto ya sea del ReproductorWAV o de ReproductorMP3Adaptador. |

Ejercicio 3

|  |  |
| --- | --- |
| Nombre | Decorador |
| Propósito | Agregar un nuevo estado y comportamiento a un objeto. |
| Problema | Agregar bigote, anteojos y sombreros a una imagen sin hacer los cambios directo en la foto. |
| Solución | Por medio de la implementación del patrón decorador, logramos extender la funcionalidad del sistema sin necesidad de realizar herencia, guardamos una nueva instancia con los decoradores concretos para que se guarde las modificaciones. |
| Participantes y colaboradores | Componente (BaseImage): define la clase abstracta para los objetos cuyas responsabilidades pueden aumentar en tiempo de ejecución  Componente Concreto (Image): define un objeto al cual se le pueden agregar  responsabilidades.  Decorador (ObjetoDecorador): mantiene una referencia a un objeto que implementa Componente  Decorador Concreto (Sombrero, Anteojos, Bigote): Agrega responsabilidades al objeto |
| Consecuencia | El objecto original desconoce la decoración, ya que la instancia de este objeto no la estamos modificando directamente, además todas las decoraciones son independientes entre sí, eso quiere decir que podemos decorar a la imagen con uno o varios decoradores al mismo tiempo. Se gana una flexibilidad sobre la herencia estática. |
| Implementación | Se crea una clase abstracta llamada BaseImage la cual es la que puede aumentar la responsabilidad en tiempo de ejecución, se crea el componente concreto que es la clase Image que hereda de BaseImage. Se crea el objecto decorado que hereda de BaseImage y tiene una referencia a esta clase abstracta y después de eso se crean los objectos decoradores concretos que son los que le agregan las responsabilidades en tiempo de ejecución, ya lo que queda al cliente es crear una Imagen y utilizar cualquier objeto decorador con la imagen. |

Ejercicio 4

|  |  |
| --- | --- |
| Nombre | Compositor |
| Propósito | Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que, al poseer todos ellos una interfaz común, se tratan todos de la misma manera. |
| Problema | Implementar un drive donde se pueden crear carpetas, archivos y se puede mover los archivos o carpetas dentro de otras carpetas. |
| Solución | Por medio del patrón compositor, crearemos una estructura de datos llamada árbol donde cada hoja puede ser un archivo o una carpeta y por medio de la recursividad poder obtener las diferentes hojas. |
| Participantes y colaboradores | Component (IDrive, Elemento): Es una clase abstracta que tiene las operaciones mínimas que serán utilizadas, este componente es extendido por los otros dos componentes Leaf y Composite.  Leaf (Archivo): Representa la parte más simple o pequeña de toda la estructura, el cual con se puede representar con los archivos y ésta hereda de Component.  Composite (Carpeta): Está conformado por un conjunto de Component y Leaf y hereda de component. |
| Consecuencia | Permite tratamiento uniforme de objetos simples (archivos) y complejos (carpetas) así como composiciones recursivas y además simplifica el código de los clientes, que sólo usan una interfaz. |
| Implementación | Se crea el elemento abstracto (clase abstracta) Component IDrive donde se ponen los métodos que tiene archivos y carpetas, lo que no son implementados por alguno se envía una excepción. Las clases derivadas ósea Archivos y Carpetas deben de encargarse de proporcionar la funcionalidad y el cliente le pide al componente que realice una operación y esto hace que una hoja o varias ejecuten lo que les piden y se lo devuelve al componente y este se lo devuelve al cliente. |