Práctica 2: Implementación de los patrones de diseño: Decorator y Adapter para la resolución de un problema

UNAM, Facultad De Ciencias.

Curso: Modelado y Programación
Profesor.Rosa Victoria Villa Padilla
Ayudante. Lab. Arturo Lemus Pablo
Ayudante. Fernando López Balcazar
Ayudante. Itzel Azucena Delgado Díaz

Equipo: Prietos en aprietos

SanMartin Macias Juan Daniel No. Cuenta 318181637

López Diego Gabriela No. Cuenta 318243485

Rivera Zavala Javier Alejandro No. Cuenta 311288876

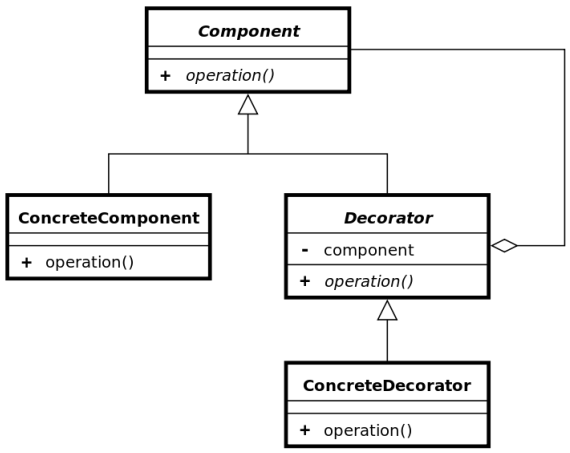
25 de septiembre de 2022.
Ciudad de México.

1. Patrón de diseño DECORATOR

Este patrón de tipo estructural nos es util cuando queremos añadir algun tipo de funcionalidad o responsabilidad a un objeto. Tambien, cuando necesitamos extender la funcionalidad de una clase pero la herencia no una solución viable. Lo anterior, lo podemos realizar cuando colocamos estos objetos dentro de objetos encapsuladores que contienen dichas funcionalidades.
El patrón lo implementamos de la siguiente manera

1. Creamos una clase abstracta o interfaz component que sera el objeto a decorar o nucleo.
2. Las clases concrete componet heredan o implementan de component. Seran objetos ya determinados al que tambien le podemos añadir resposabilidades.
3. La clase Decorator, sera abstracta. Aquí tendra alguna referencia al componente asociado. Y como se muestra en la imagen, hereda de la clase component.
4. Las clases concrete decorator seran los objetos encargados de añadir funcionalidades al componente nucleo.

Es decir, la plantilla general de dicho patron esta implementado de la siguiente manera:



Desventajas

1. Podemos generar grandes cantidades de objetos pequeños(decoradores)
2. Puede existir problemas con la identidad de los objetos, debido a que los objetos decoradores (envoltorios) se comportan como el objeto nucleo pero con referencia distinta.

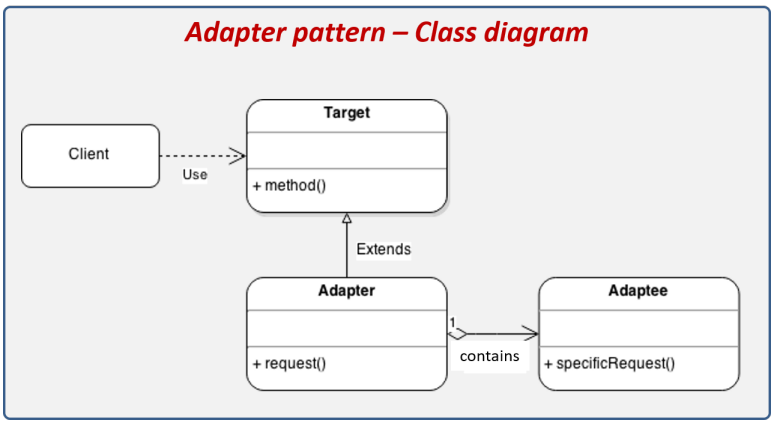
2. Patrón de diseño ADAPTER

De igual forma, se trata de un patron de diseño estructural que nos permite la colaboración entre objetos e interfaces incompatibles pero alguna funcionalidad similar. Aplicamos dicho patron cuando, deseamos usar una clase existente y su interfaz no sea igual a la requerida y/o cuando queremos crear una clase reutilizable que coopere con clases no relacionadas.
Adapter lo implementamos de la siguiente manera:

1. Nuestra interfaz target, nos permitira homogenizar ambas interfaces incompatibles.

2. La clase adapter tendra la responsabilidad de mediar entre cliente y adaptee
3. Finalmente, la clase adaptee sera nuestra clase con interface incompatible

Lo anterior, lo podemos ver de la siguiente manera: ’



Desventajas

1. añade complejidad y contribuye a la dificultad implicada en la compresión del programa.
2. si tenemos dos clases que implementen muchos metodos, sera muy dificl de adaptar.

Por ultimo,

Para compilar la practica 03 escribimos lo siguiente en terminal

javac RestauranteWaySub.java

Y para ejecutarlo

java RestauranteWaySub