Angelica Yeraldin Rodriguez Gualtero – 202215816

Taller. 5 - Patrones
Programación y diseño orientada a objetos
Angelica Yeraldin Rodriguez Gualtero

Información general del proyecto:
La URL para consultar mi proyecto seleccionado es:

https://github.com/palencia77/ejemplo-decorator.git

El enunciado del proyecto de ejemplo del uso del patrón decorador es:

Una fábrica de automóviles produce uno de sus modelos en tres variantes, llamadas sedán, coupé y familiar. Cada una tiene un precio de venta básico sin opcionales. A su vez, a cada variante se le pueden agregar opcionales como techo corredizo, aire acondicionado, sistema de frenos ABS, airbag y llantas de aleación. Cada uno de estos opcionales tiene un precio que suma al básico. En este caso, cada auto vendrá caracterizado por su variante y podrá tener ninguno, uno o más opcionales. Asumiendo los siguientes costos:

- Básico sedán 230.000
- Básico familiar 245.000
- Básico coupé 270.000
- Techo corredizo (TC) 12.000
- Aire acondicionado (AA) 20.000
- Sistemas de frenos ABS (ABS) 14.000
- Airbag (AB) 7.000
- Llantas de aleación (LL) 12.000

Diseñe una solución que permita calcular el costo final de un automóvil. El objetivo es caracterizar cada automóvil por su variante y los opcionales que tiene.

Estructura general del diseño (basada en el repositorio Ejemplo del patrón Decorador):

1. **Componente Base (Automovil):**

- Representa la clase base que queremos extender con funcionalidad adicional.
- La clase Automovil es abstracta y contiene los siguientes atributos:
    - precio: Representa el precio básico del automóvil.
    - descripcion: Representa la descripción del automóvil (por ejemplo, "Automóvil Sedán").
- Define un método abstracto calcularCostoTotal() que debe ser implementado por las subclases.

2. **Clases Concretas (AutomovilSedan, Airbag, AireAcondicionado, FrenosABS y LlantaAleacion):**

- Son clases que extienden la clase Automovil.
- Cada una de estas clases representa una variante específica de automóvil o un opcional.
- En el constructor, se establece el precio y la descripción correspondientes.
- El método calcularCostoTotal() calcula el costo total considerando el precio base y los opcionales.

3. **Decoradores (Airbag, AireAcondicionado, FrenosABS, LlantaAleacion y TechoCorredizo):**

- Son clases concretas que extienden la clase OpcionalesDecorator.
- Agregan funcionalidad específica al componente base (Automovil).
- En el método calcularCostoTotal(), suman el precio del opcional al costo total.

Los retos dentro del proyecto son:

1. **Flexibilidad y Extensibilidad**:

    - El patrón Decorator se utiliza para extender las funcionalidades de los objetos de automóviles sin afectar su estructura básica.
    - El reto es garantizar que los opcionales (decoradores) puedan agregarse o eliminarse dinámicamente durante el tiempo de ejecución sin afectar otras partes del código.

2. **Composición de Opcionales**:

    - El proyecto debe manejar la composición de múltiples opcionales (decoradores) de manera coherente.
    - Cada opcional puede afectar el costo total y la descripción del automóvil, por lo que la gestión de la composición es fundamental en el desarrollo.

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

En el proyecto se usa una carpeta "**Decoradores**" donde encontramos los archivos (Airbag.java, AireAcondicionado.java, FrenosABS.java, LlantaAleacion.java, OpcionalesDecorator.java, TechoCorredizo.java) en los que el patrón Decorator se utiliza para agregar funcionalidades a los objetos de automóviles colocándolos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

1. Airbag.java:
    - La clase Airbag es un decorador concreto que extiende la clase OpcionalesDecorator.
    - Agrega la funcionalidad de un airbag al automóvil.
    - El precio adicional del airbag es de 7,000 dólares.
    - La descripción del opcional es "AB".
2. AireAcondicionado.java:
    - La clase AireAcondicionado también es un decorador concreto que extiende la clase OpcionalesDecorator.
    - Agrega la funcionalidad de aire acondicionado al automóvil.
    - El precio adicional del aire acondicionado es de 20,000 dólares.
    - La descripción del opcional es "AA".
3. FrenosABS.java:
    - La clase FrenosABS es otro decorador concreto que extiende la clase OpcionalesDecorator.
    - Agrega la funcionalidad de frenos ABS al automóvil.
    - El precio adicional de los frenos ABS es de 14,000 dólares.
    - La descripción del opcional es "ABS".
4. LlantaAleacion.java:
    - La clase LlantaAleacion también es un decorador concreto que extiende la clase OpcionalesDecorator.
    - Agrega la funcionalidad de llantas de aleación al automóvil.
    - El precio adicional de las llantas de aleación es de 12,000 dólares.
    - La descripción del opcional es "LL".

En resumen, estos archivos representan diferentes opcionales que se pueden agregar a los automóviles básicos. Cada decorador extiende la funcionalidad del automóvil base sin afectar la estructura general.

Información general sobre el patrón: qué patrón es y para qué se usa usualmente.

El patrón que elegí para este taller es el patrón Decorador (Decorator) en este se añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad. (Design Patterns Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides).

1. **Expansión dinámica**: El patrón permite agregar o modificar funcionalidades durante el tiempo de ejecución. Esto es especialmente útil cuando se necesita ajustar el comportamiento de un objeto sin recurrir a jerarquías de herencia largas y complejas.
2. **Componente y decoradores**: En el patrón, un **componente** (objeto base) se "decora" con una o más **clases decoradoras**. Estas clases envuelven completamente al componente y tienen la misma interfaz que él. Las llamadas de método entrantes pueden delegarse fácilmente al componente adjunto mientras se realiza una funcionalidad adicional dentro del decorador.

3. **¿Cómo funciona?**

    - Tenemos un componente base (interfaz o clase abstracta) que define la funcionalidad básica.
    - Los decoradores concretos extienden el componente base y agregan funcionalidades específicas.
    - Los decoradores se pueden combinar de manera flexible para crear objetos complejos con múltiples características.

4. **¿Para qué se usa usualmente?**
   El patrón Decorator se utiliza cuando queremos agregar funcionalidades a objetos de manera dinámica sin afectar su estructura, necesitamos una alternativa a la herencia para extender funcionalidades o deseamos evitar una clase con múltiples variantes de subclases.

Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.

Clase base Automovil:
La clase abstracta Automovil define un automóvil genérico con atributos como precio y descripción que también declara un método abstracto calcularCostoTotal() que debe ser implementado por las subclases.

Subclases de Automovil:
Las clases AutomovilCoupe, AutomovilFamiliar y AutomovilSedan representan diferentes tipos de automóviles.Cada una establece su precio base y descripción específica e implementan el método calcularCostoTotal() para devolver el precio base.

Decoradores:
Los decoradores extienden la funcionalidad de los automóviles base.
Ejemplos de decoradores en el proyecto:

- Airbag: Agrega la funcionalidad de airbag al automóvil.
- AireAcondicionado: Agrega la funcionalidad de aire acondicionado.
- TechoCorredizo: Agrega la funcionalidad de techo corredizo.

Composición dinámica:
Los decoradores se pueden combinar de manera flexible.Por ejemplo, un automóvil puede tener airbags y aire acondicionado al mismo tiempo.

En el proyecto seleccionado el patrón Decorador permite que los automóviles se personalicen con opciones adicionales sin crear una jerarquía de clases compleja. Cada decorador agrega funcionalidad sin afectar la estructura original del automóvil.

Angelica Yeraldin Rodriguez Gualtero – 202215816

¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?

El uso del patrón de diseño Decorator en ese punto del proyecto tiene sentido y ofrece varias ventajas:

-**Flexibilidad y extensibilidad:** El patrón Decorator permite agregar nuevas funcionalidades de manera dinámica sin modificar la estructura de las clases existentes. Adicionalmente, en el contexto de los automóviles, esto significa que se pueden incorporar opciones adicionales (como airbags, aire acondicionado o techo corredizo) sin afectar las clases base (sedán, coupé, familiar).
-**Evita creación de múltiples subclases:** Sin el patrón Decorator, el autor podría haber creado múltiples subclases para cada combinación de opciones (por ejemplo, AutomovilSedanConAirbag, AutomovilSedanConAireAcondicionado, etc.) lo que resultaría en una jerarquía de clases compleja y difícil de mantener.

-**Mantenimiento y reutilización:** Cada decorador es independiente y se puede reutilizar en diferentes contextos.Por ejemplo, si se necesita cambiar o agregar una funcionalidad, solo se modifica el decorador correspondiente, sin afectar otras partes del código lo que facilita enormemente el mantenimiento

-**Cumple con el principio de "Open/Closed":** El patrón Decorator sigue el principio SOLID de "Open/Closed" ya que las clases están abiertas para extensión (pueden recibir nuevos decoradores), pero cerradas para modificación (no es necesario cambiar las clases existentes).

¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?
Las desventajas que más destacan al usar este patrón en el proyecto son:

-Complejidad adicional: La introducción de múltiples decoradores puede aumentar la complejidad del código, se debe tener cuidado con que cada nuevo decorador debe implementarse y probarse correctamente, además si se agregan muchos decoradores, el mantenimiento y la comprensión del sistema pueden volverse más difíciles.

-Posible sobrecarga de objetos: Cada decorador envuelve un objeto base, lo que puede resultar en una sobrecarga de objetos pues, si se aplican muchos decoradores, el rendimiento podría verse afectado debido a la creación y gestión de múltiples objetos.

Angelica Yeraldin Rodriguez Gualtero – 202215816

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

Se me ocurren dos formas de solucionar este caso en vez de usar el patrón decorador y son por medio de herencia y subclases y el patrón builder.
-Herencia y Subclases: En lugar de usar decoradores, podría haber creado una jerarquía de subclases para cada combinación de opciones. Por ejemplo, tendríamos clases como AutomovilSedanConAirbag, AutomovilSedanConAireAcondicionado, etc. Sin embargo, esto podría resultar en una jerarquía de clases extensa y difícil de mantener.

-Patrón Builder: El patrón Builder podría utilizarse para construir objetos complejos paso a paso.Cada opción (airbag, aire acondicionado, etc.) podría ser un paso en el proceso de construcción y esto proporcionaría una forma estructurada de crear objetos con diferentes configuraciones.