Mariana Pineda Miranda
202123330
TALLER 5 DEPO
Patrón: Composite
URL: https://github.com/RefactoringGuru/design-patterns-java/tree/main/src/refactoring_guru/composite/example

1. Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta (i.e. ¿qué es lo difícil?). Deben incluir la URL para consultar el proyecto.

El repositorio que se uso es un programa el cual pinta diferentes figuras y las agrupa. En primer lugar, se tiene la interfaz "Shape" en la cual se definen los métodos que debe tener un figura para que esta pueda dibujarse. Al ser una interfaz los métodos no tienen cuerpo, sin embrago, todas las clases hijas de esta interfaz deben implementar todos los métodos que se hayan declarado en la interfaz. A continuación, se tiene la clase abstacta "BaseShape" en donde se implementan cada uno de los métodos que contiene la interfaz padre, esta clase contiene atributos característicos de una figura como su ancho, alto, color y si esta seleccionada. El método *enableSelectionStyle()* funciona cuando una figura esta seleccionada y si esto ocurre se cambia el color a gris claro y se le aplica una línea en el medio, y se tiene el método *disableSelectionStyle()* para revertir lo mencionado anteriormente. Por último, en esta clase abstracta "BaseShape" se tiene el método para pintar la figura en un objeto gráfica, de igual manera este método verifica si la figura esta seleccionada y aplica el formato indicado.

Por otra parte, se tienen cada una de las figuras que se quieren dibujar, en este caso se quiere dibujar un círculo, un rectángulo y un punto. Cada una de estas figuras tiene una clase independiente en la cual se tienen métodos para encontrar los atributos característicos de cada una de las figuras.

a. "Circle": esta clase extiende a la clase BaseShape por lo que hereda todos los métodos y atributos públicos de la clase BaseShape. En esta clase el constructor entra como parámetros la coordenada x y y en donde se va a ubicar la figura, entra el radio del círculo y por último entra el color de la figura. En este caso tienen la funciones getWidth() y getHeight() relacionadas con el circulo por lo que se tiene en cuenta el radio de la figura y se multiplica este por dos para encontrar el diámetros y así el ancho y el alto. Finalmente se tiene la función paint() con la que se pinta el circulo por medio de la función drawOval() con parámetros las coordenadas x y y, y con tamaño el ancho y alto que se encontraron previamente, de igual manera se sabe que se pinta en la figura base.
b. "Rectangle" es una clase la cual extiende "BaseShape" lo que significa que esta hereda todos los métodos y parámetros públicos de la clase padre. En el constructor de esta clase entran como parámetros la coordenada de ubicación (x,y), el ancho del rectángulo y el alto del mimo ademas de su color, al ser este un rectángulo su ancho y alto son los valores que entran por parámetro y no es necesario realizar alguna modificación al respecto. Finalmente, se tiene la función paint() en donde se pinta la figura teniendo en cuenta la ubicación que entra por parámetro y las dimensiones de la figura.
c. "Dot": esta clase extiende a la clase abstracta "BaseShape" esto quiere decir que todos los métodos implementados en la clase padre se heredan a la clase "Dot". En esta clase se define el tamaño del punto con una constante y su ancho y alto son esta misma

constante. De igual manera en el constructor se llama al padre de la clase “Dot” para obtener la principal información de esta. Además se tiene el método paint() para pintar el punto en las coordenadas ingresadas junto a las dimensiones del mismo.

De igual manera, se tiene la clase “CompundShape” la cual extiende de la clase “BaseShape”, lo que significa que esta clase hereda todos los métodos y atributos públicos que tiene la clase padre. El propósito de esta clase es tener una figura compuesta de diferentes figuras. En la lista *children* se guardan las figuras que se van a dibujar en la “CompundShape”. El constructor de la clase recibe como parámetros un array de objetos de la clase “Shape” y los añade a la lista *children*.

Entre las principales dificultades que se tienen con la implementación de este proyecto está la creación de figuras con diferentes estructuras y características las cuales deben todas ubicarse en una misma figura compuesta. De igual manera, cada una de las figuras tiene métodos diferentes para encontrar sus dimensiones, sin embargo, al final todas son figuras y deben de alguna manera pertenecer a la clase “Shape”

2. Información y estructura del fragmento del proyecto donde aparece el patrón. No se limite únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente va a tener que incluir elementos cercanos que sirvan para contextualizarlo.

Entre los fragmentos en los que se puede evidenciar la presencia del patrón “Composite” se encuentra la creación de la interfaz “Shape” y la clase “BasicShape” en donde se definen los métodos necesarios para pintar una figura, sin embargo, se usa @Override para mostrar que algunos de los atributos de cada una de estas clases van a ser implementados de manera separada en las clases correspondientes a cada figura debido a que estas características son diferentes cada una. De igual manera, se sabe que a quien corra la aplicación solo quiere mostrársele la figura compuesta y no le interesa las figuras que la componen por lo que el patron “Composite” es aplicado.

De igual manera, bien se sabe que en el programa se tiene una jerarquía en donde se tienen tanto, objetos por separado (“Circle”, “Rectange”, “Dot”) como una agrupación de objetos (“CompoundShape”) y se ve que se quiere trabajar con ellos de manera uniforme.

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

El patrón “Composite” es un patrón estructural el cual permite componer objetos por medio de estructuras jerárquicas, en donde se permite tratar los elementos por separado como los grupos de manera uniforme. Con este patrón se pretende que los clientes del programa no tengan conocimiento sobre la diferencia entre objetos individuales y las composiciones de objetos. La clave de este patrón es una clase abstracta la cual represente los objetos primitivos y sus contenedores. De igual manera, es importante destacar que la jerarquía de este patrón se asemeja a la jerarquía de un árbol incluyendo elementos de este para la estructura del patrón.
La estructura de este patrón se ve de la siguiente manera:

- Componente:
    - Declara la interfaz para objetos de la misma composición.

    - Implementa un comportamiento común el cual debe ser manejado por todas las clases que hereden esta interfaz.
    - Declara una interfaz para acceder y manejar los componentes de los hijos.
- Hojas:
    - Representa cada uno de los objetos de los que hace parte la composición.
    - Define el comportamiento de los elementos que hacen parte de la composición.
- Composite:
    - Define el comportamiento de los componentes que cuentan con herencias.
    - Almacena los componentes de los hijos de la clase padre.
    - Implementa las operaciones necesarias para realizar una composición.
- Cliente:
    - Manipula los objetos de la clase por medio de la interfaz del componente.
    - No diferencia entre objetos individuales y la composición.

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

El patrón Composite se aplica en el proyecto en la medida en que se tiene cada uno de los componentes de la estructura del patrón Composite. De igual manera, se puede notar que se separan los objetos individuales de las composiciones de objetos, sin embargo, se tiene la intención de tratar ambos tipos de objetos de manera uniforme.

- Componente: En este caso el componente seria la interfaz “Shape” esta es una interfaz la cual se declara para objetos con la misma composición. Así mismo, en esta clase se declaran los métodos que se necesitan para la construcción de una figura los cuales se deben heredar para todas las clases las cuales implementen la interfaz mencionada. Por otra parte.
- Hojas: En este caso se tienen cuatro clases las cuales pueden ser consideradas como Hojas.
    - En primer lugar, la clase abstracta “BasicShape” en donde se define el comportamiento que tiene una figura básica sin alguna característica en especifica, sin embargo, esta implementa la interfaz “Shape” lo que significa que en esta clase se deben implementar todos los métodos que se declararon previamente en la interfaz “Shape”.
    - Por otra parte, se tiene la clase “Circle” en esta clase se define el comportamiento especifico de un círculo y representa la clase que se usa para dibujar un círculo.
    - Así mismo, se tiene la clase rectángulo, que al igual que la clase circulo define el comportamiento de una figura en específico, esta clase define el comportamiento de un rectángulo y representa una clase la cual pinta un rectángulo.
    - Finalmente, se tiene la clase “Dot” la cual se usa para realizar el dibujo de un punto y tiene los atributos y los métodos específicos de un punto.

- Composite: Se tiene la clase “CompoundShape” en donde se agrupan las figuras individuales de las hojas para ponerlas en una sola figura en donde este la composición de las tres figuras representadas por las hojas.

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

Tiene sentido haber usado el patrón en ese punto del proyecto debido que este patrón ayuda a que la separar los objetos individuales de la composición, sin embargo, estos se siguen usando de manera uniforme. De igual manera, el uso de este patrón ayuda a la simplificación del código ya que permite al usuario usar los objetos de manera uniforme sin diferenciar entre objetos individuales y composiciones. De igual manera, se puede evidenciar una flexibilidad en la jerarquía debido a que se pueden manejar cada una de las diferentes figuras pudiendo diferenciar las características específicas de cada una.

Otra de las ventajas del uso de “Composite” es que los objetos primitivos, en este caso las figuras pueden convertirse en componentes de objetos más complejos, en este caso un panel con diferentes figuras, además este proceso se puede realizar de manera recursiva. Es mucho más facl añadir nuevos componentes, por ejemplo, si se quiere añadir una nueva figura solo se debe crear una clase la cual contenga la información específica de la figura, y no se deben crear nuevas clases que traten a la figura de manera independiente, además añadirla al panel de figuras es bastante simple por el uso de “Composite”.

6. ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

Algunas de las desventajas del uso del patrón “Composite” puede ser el acoplamiento de clases debido a que la dependencia entre clases es bastante alta. De igual manera, se puede tener una complejidad adicional debido a que en cambio de tener una clase en la que se implemente solo un panel con las diferentes figuras en sus diferentes ubicaciones se tienen varias clases cada una con figuras con comportamientos diferentes. De igual manera, debido a que se tienen más clases el rendimiento del programa sería menor y se podría tener una sobrecarga de rendimiento. El diseño podría ser algo general y seria más difícil restringir componentes, debido a que los objetos individuales y las composiciones se tratan de la misma manera se tendría que agregar lógica adicional para operaciones específicas.

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

Este problema se pudo resolver por medio de otros patrones, por ejemplo, se pudo utilizar el patrón Bridge para separar las figuras y sus atributos de la forma en las que estas se dibujan. De igual manera, se podría tener sola una clase la cual simplemente pinte cada uno de los componentes dentro de un panel, sin embargo esto sería muy complicado para una sola clase y haría el programa muy pesado.