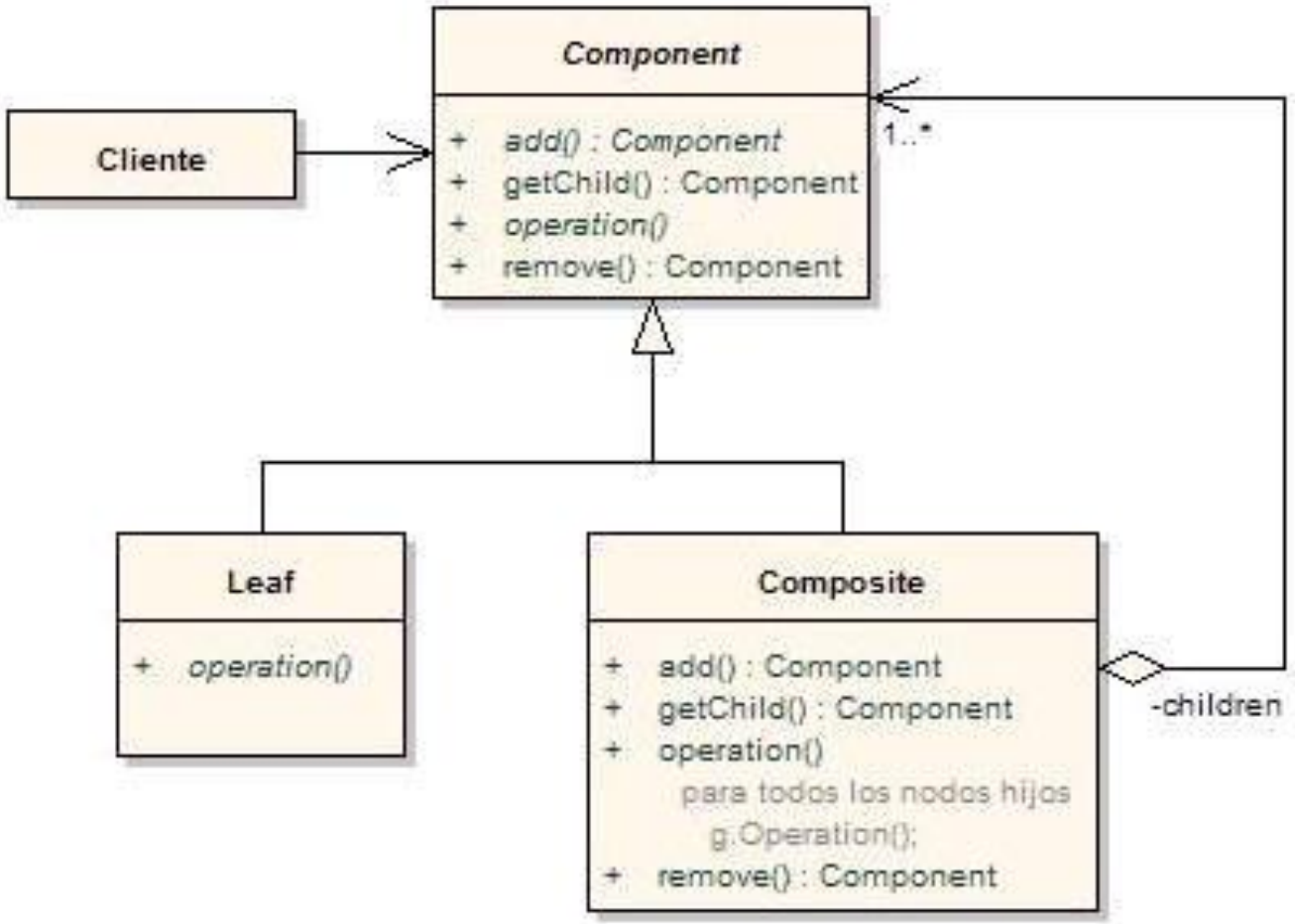
Patrón de Diseño: Composite

Integrantes: Luis Huang, Luis Choque, Felipe Morales, Agustin Zabalgoitia, Luciano Martins Louro, Maximo Bosch, Lecuona Martin Roberto, Burnowicz Alejo, Huang Marcelo, Federico Agustin, Perez Ariel, Sanson Melina, Brenda Martinez, Rodriguez Mariano

Diagrama de Clases



El patrón Composite sirve para construir algoritmos u 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.

Diagrama de Clases del Problema

Mediante el programa, cada NaveEspacial se podrá movilizar individualmente o en conjunto mediante el Escuadron.
Al igual que realizar un ataque individual o en conjunto.

Fragmentos de Código

```java
public abstract class UnidadCombate {

    protected int posX = 0;
    protected int posY = 0;
    protected String Nombre;
    protected boolean alive = true;

    public void move(int newPosX, int newPosY) {
        this.posX = newPosX;
        this.posY = newPosY;
    }

    public abstract void takeDamage(int damage);

    public void mostrarPosicion() {
        System.out.println("Posicion en x: " + this.posX + "\nPosicion en Y: " + this.posY);
    }

}
```

```java
public class NaveEspacial extends UnidadCombate {

    protected int HP;
    protected int damage;
    protected int fuel;

    public NaveEspacial(int HP, int damage, int fuel, String nombre) {
        this.HP = HP;
        this.Nombre = nombre;
        this.fuel = fuel;
        this.damage = damage;
    }

    public void attack(NaveEspacial unit) {..

    public void takeDamage(int damage) {..

    public int mostrarVida() {..
```

```java
import java.util.LinkedList;

public class EscuadronCazas extends UnidadCombate {
    private LinkedList<NaveEspacial> listaNaves = new LinkedList<NaveEspacial>();

    public EscuadronCazas(NaveEspacial nave) {
        listaNaves.add(nave);
        // TODO Auto-generated constructor stub
    }

    public EscuadronCazas(LinkedList<NaveEspacial> ListaNaves) {
        this.listaNaves.addAll(ListaNaves);
        // TODO Auto-generated constructor stub
    }

    public void takeDamage(int damage) {

    public void attack(NaveEspacial naveEnemiga) {

    public void attack(EscuadronCazas escuadronEnemigo) {


    @Override
    public void move(int posX, int posY)
    {
        for (NaveEspacial naveEspacial : listaNaves) {
            naveEspacial.move(posX, posY);
        }
    }
```

Ventajas:

- La simplificación y la abstracción de una familia de clases.
- Permite su uso de manera mas simple ya que no es necesario el conocimiento de sobre qué clase se está trabajando, pero será posible asegurar que contiene los elementos de la interfaz.
- La escalabilidad porque podríamos agregar mas módulos fácilmente sin hacer un cambio significativo en el código
- Es mucho mas prolijo el implementar cambios

Desventajas:

- Las desventajas de este patrón es que el diseño de las clases se vuelve muy genérico pudiendo llegar a generar confusión a la hora de hacer cambios en las mismas.

- Solo se puede aplicar en proyectos cuyas clases tengan una relación muy evidente.

- Sin el patrón se tendrían que crear por lo menos 2 clases distintas y no sería posible hacer referencia a ambas a la vez. Se tendrían que repetir código y el usuario de la familia de clases deberá de saber con qué clase está trabajando.

Referencias

- http://migranitodejava.blogspot.com/2011/06/composite.html (Texto e Imagenes)