Design Pattern :

COMPOSITE

LUONG Tristan
MAVOUNGOU Ken

Sommaire

Introduction

Composite est un Structural pattern.

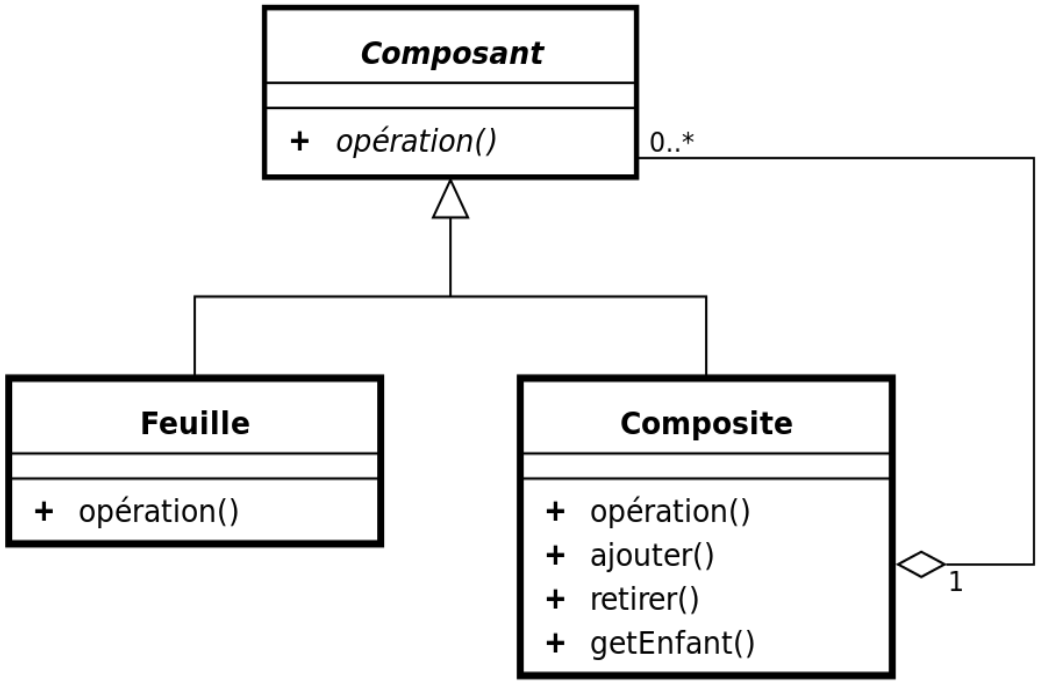
En génie logiciel, les modèles de conception structurale sont des modèles de conception qui facilitent la conception en identifiant un moyen simple de réaliser des relations entre les entités.

Ce patron permet de concevoir **une structure arborescente.**

Il existe donc le design pattern composite qui permet de gérer un ensemble d'objets en tant qu'un seul et même objet, autrement dit un **objet composé de plusieurs autres**.

Il permet d'additionner les propriétés des différents objets (par exemple un prix) pour en composer un seul et même. Il simplifie les compositions car on peut ajouter ou supprimer des éléments d'un objet composé grâce à des méthodes, sans avoir à modifier le code de leurs classes.

Structure



Composant

- est l'abstraction pour tous les composants, y compris ceux qui sont composés
- déclare l'interface pour le comportement par défaut

Feuille

- représente un composant n'ayant pas de sous-éléments
- implémente le comportement par défaut

Composite

- représente un composant pouvant avoir des sous-éléments
- stocke des composants enfants et permet d'y accéder
- implémente un comportement en utilisant les enfants

Structure

Spécificités :

+ le composant est une interface généralement
+ le composite possède les méthodes *.add()*, *.remove()*, qui permettent d'ajouter ou de supprimer plusieurs éléments à une composition/composite
+ relation many-to-one entre composite ->composants

Exemples :

a. Code (Java)

Prenons l'exemple d'un camion semi-remorque.

Ce dernier est composé d'un tracteur et d'une remorque, qui ont un poid séparé mais également un poid camion entier. Un tracteur routier doit également pouvoir rouler sans remorque

```java
public interface Composant {

    public int getPoids();

}
```

```java
public class Remorque implements Composant {

        private int poids;

    public Remorque(int poids) {
                this.poids = poids;
    }

    @Override
    public int getPoids() {
        return this.poids;
    }

}

public class Tracteur implements Composant {

        private int poids;

    public Tracteur(int poids) {
                this.poids = poids;
    }

        @Override
    public int getPoids() {
        return this.poids;
    }
}
```

Exemples :

```
public class CamionComposite implements Composant {

        private Collection children;

    public CamionComposite() {
        children = new ArrayList();
    }


    public void add(Composant composant){

        children.add(composant);
    }


    public void remove(Composant composant){
        children.remove(composant);
    }

    public Iterator getChildren() {
        return children.iterator();
    }

        @Override
    public int getPoids() {
        int result = 0;
        for (Iterator i = children.iterator(); i.hasNext(); ) {
            Object objet = i.next();

            Composant composant = (Composant)objet;

            result += composant.getPoids();
        }
        return result;
    }
}
```

Class Composite, qui possède des éléments *children*

Exemples :

```java
public class Main
{

    public static void main(String[] args)
    {
        Remorque maRemorque = new Remorque(11);
        System.out.println("Le poids de ma remorque est:");
        System.out.println(maRemorque.getPoids());
        System.out.println("tonnes");


        Tracteur monTracteur = new Tracteur(8);
        System.out.println("Le poids de mon tracteur est:");
        System.out.println(monTracteur.getPoids());
        System.out.println("tonnes");


        CamionComposite semiRemorque = new CamionComposite();
        semiRemorque.add(maRemorque);
        semiRemorque.add(monTracteur);
        System.out.println("Le poids de mon semi-remorque est:");
        System.out.println(semiRemorque.getPoids());
        System.out.println("tonnes");
    }
}
```

Exemples (output) :

```
Le poids de ma remorque est:
11
tonnes
Le poids de mon tracteur est:
8
tonnes
Le poids de mon semi-remorque est:
19
tonnes
```

Conclusion

Avantages :
Il facilite l'ajout de nouveaux types de composants
Cela simplifie la vie des clients, puisqu'ils n'ont pas à savoir s'ils ont affaire à une ***leaf*** ou à une composante ***composite***.

Désavantages :
Il est plus difficile de restreindre le type de composants d'un ***composite***.

MERCI !