LE PATTERN DECORATEUR

Intention

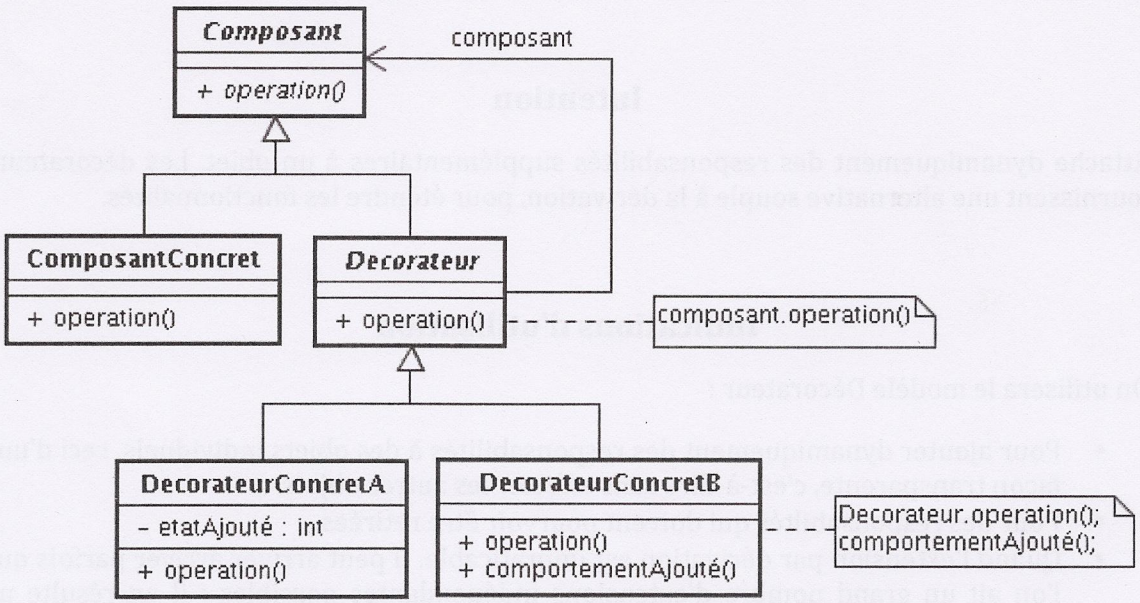
Attache dynamiquement des responsabilités supplémentaires à un objet. Les décorateurs fournissent une alternative souple à la dérivation, pour étendre les fonctionnalités.

Indications d'utilisation

On utilisera le modèle Décorateur :

- Pour ajouter dynamiquement des responsabilités à des objets individuels, ceci d'une façon transparente, c'est-à-dire sans affecter les autres objets.
- Pour des responsabiltés qui doivent pourvoir être retirées.
- Quand l'extension par dérivation est impraticable. Il peut arriver arriver parfois que l'on ait un grand nombre d'extensions indépendantes possibles ; il en résulte un prolifération explosive de sous-classes pour permettre chaque combinaison. D'autre fois, la définition de classe pourra être cachée, ou encore inaccessible pour la dérivation.

Structure



Constituants

Composant

- Composant définit l'interface des objets qui peuvent recevoir dynamiquement des responsabilités supplémentaires.

ComposantConcret

- ComposantConcret définit un objet auquel des responsabilités supplémentaires peuvent être adjointes.

Decorateur

- Décorateur gère une référence à un objet Composant et définit une interface conforme à celle du Composant.

DecorateurConcret

- DecorateurConcret ajoute des responsabilités à un composant.

Collaborations

Décorateur transmet les requêtes à son objet Composant. Il peut éventuellement effectuer des opérations supplémentaires avant et après la transmission de la requête.

```
=====================================================
                    BOISSON
=====================================================

public abstract class Boisson {
    String description = "Boisson inconnue";

    public String getDescription() {
        return description;
    }

    public abstract double cout();
}

=====================================================
                    ESPRESSO
=====================================================

public class Espresso extends Boisson {
    public Espresso() {
        description = "Espresso";
    }

    public double cout() {
        return 1.99;
    }
}

=====================================================
                    SUMATRA
=====================================================

public class Sumatra extends Boisson {
    public Sumatra() {
        description = "Sumatra corsé";
    }

    public double cout() {
        return .99;
    }
}
```



```
=====================================================
                    COLOMBIA
=====================================================

public class Colombia extends Boisson {
    public Colombia() {
        description = "Pur Colombia";
    }

    public double cout() {
        return .89;
    }
}


=====================================================
                    CAFETE
=====================================================

public class StarBuzz {
    public static void main(String args[]) {
        Boisson cafe = new Espresso();
        System.out.println(cafe.getDescription()
                + " €" + cafe.cout());

        Boisson boisson2 = new Sumatra();
        boisson2 = new Chocolat(boisson2);
        boisson2 = new Chocolat(boisson2);
        boisson2 = new Chantilly(boisson2);
        System.out.println(boisson2.getDescription()
                + " €" + boisson2.cout());

        Boisson boisson3 = new Colombia();
        boisson3 = new Caramel(boisson3);
        boisson3 = new Chocolat(boisson3);
        boisson3 = new Chantilly(boisson3);
        System.out.println(boisson3.getDescription()
                + " €" + boisson3.cout());
    }
}
```