Decorator Design Pattern

תבנית העיצוב Decorator היא תבנית עיצוב מבנית המאפשרת להוסיף פונקציונליות לאובייקטים קיימים בזמן ריצה. תבנית זו מאפשרת להוסיף פונקציונליות לאובייקטים קיימים בזמן ריצה ומחליפה ירושה ויצירה של הרבה תתי מחלקות.

אנחנו "נעטוף" את האובייקט המקורי באובייקטים נוספים שמוסיפים פונקציונליות נוספות. כל אובייקט נוסף מוסיף פונקציונליות נוספות.

התבנית בנויה מ3 רכיבים עיקריים:

1. **Component** - ממשק או מחלקה אבסטרקטית שמגדירה את הפונקציונליות הבסיסית של האובייקט.
2. **ConcreteComponent** - מחלקה שמיישמת את הממשק Component, ומכילה עוד אובייקט מסוג Component, שעליו היא מוסיפה פונקציונליות.
3. *Decorator* - ממשק או מחלקה אבסטרקטית שמיישמת את הממשק Component ומכילה עוד אובייקט מסוג Component. כל מחלקה שיורשת מהמחלקה הזו מוסיפה פונקציונליות נוספת לאובייקט המקורי.

נשתמש בתבנית זו כאשר:

- נרצה להוסיף בצורה דינמית פונקציונליות לאובייקטים קיימים בזמן ריצה.
- כשהבעיה דורשת הרבה תתי מחלקות עם פונקציונליות דומה, והירושה אינה פתרון טוב.

יתרונות

- גמישות - ניתן להוסיף פונקציונליות בזמן ריצה.
- מונע הרבה תתי מחלקות - אפשר לשלב כמה דקורטורים ולקבל תוצאה של כל הפונקציונליות מבלי להוסיף הרבה תתי מחלקות.
- הפרדת פונקציונליות - ניתן להפריד את הפונקציונליות לכמה דקורטורים ולהוסיף ולהסיר פונקציונליות בקלות.

דוגמת קוד

בדוגמה ניצור בית קפה שבו יש סוגים שונים של משקאות עם מחירים שונים. יהיה לנו את הממשק `Beverage` שמגדיר את הפונקציונליות הבסיסית של המשקה. ניצור מחלקות `Espresso` ו- `HouseBlend` שמיישמות את הממשק.

בשביל התוספות למשקאות הבסיסיים ניצור מחלקה אבסטרקטית `CondimentDecorator` שמיישמת את הממשק `Beverage` ומכילה עוד אובייקט מסוג `Beverage`. ניצור מחלקות `Mocha` , `Milk` ו- `Whip` שמיישמות את `CondimentDecorator` ומוסיפות פונקציונליות נוספות למשקאות.

עכשיו בכל פעם שנרצה להוסיף תוספת למשקה ניצור אובייקט מהמחלקה המתאימה ונעטוף את המשקה המקורית באובייקט החדש, ובמימוש של הפונקציות של ה `CondimentDecorator` נשתמש בפונקציות של המשקה המקורית ונוסיף פונקציונליות נוספת (במקרה שלנו תוספת למחיר ותוספת לתיאור).

```
// Component interface
interface Beverage {
    String getDescription();
    double cost();
}

// Concrete Components (base beverages)
class Espresso implements Beverage {
    @Override
    public String getDescription() {
        return "Espresso";
    }

    @Override
    public double cost() {
        return 1.99;
    }
}

class HouseBlend implements Beverage {
    @Override
    public String getDescription() {
```

```java
        return "House Blend Coffee";
    }

    @Override
    public double cost() {
        return 0.89;
    }
}

// Decorator (abstract add-on)
abstract class CondimentDecorator implements Beverage {
    protected Beverage beverage;

    public CondimentDecorator(Beverage beverage) {
        this.beverage = beverage;
    }

    public abstract String getDescription();
}

// Concrete Decorators (add-ons)
class Milk extends CondimentDecorator {
    public Milk(Beverage beverage) {
        super(beverage);
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Milk";
    }

    @Override
    public double cost() {
        return beverage.cost() + 0.20;
    }
}

class Mocha extends CondimentDecorator {
    public Mocha(Beverage beverage) {
        super(beverage);
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Mocha";
    }

    @Override
    public double cost() {
        return beverage.cost() + 0.30;
    }
}

class Whip extends CondimentDecorator {
    public Whip(Beverage beverage) {
        super(beverage);
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Whip";
    }

    @Override
    public double cost() {
        return beverage.cost() + 0.10;
    }
}

// Usage
public class CoffeeShop {
```

```java
    public static void main(String[] args) {
        Beverage beverage1 = new Espresso();
        beverage1 = new Milk(beverage1);
        beverage1 = new Mocha(beverage1);

        Beverage beverage2 = new HouseBlend();
        beverage2 = new Whip(beverage2);

        System.out.println(beverage1.getDescription() + " $" + beverage1.cost());
        System.out.println(beverage2.getDescription() + " $" + beverage2.cost());
    }
}
```

בעצם החיסכון כאן זה שבמקום ליצור תתי מחלקות רבות של משקאות עם תוספות שונות, אנחנו יכולים ליצור תתי מחלקות אחת לכל תוספת ולהוסיף אותם למשקה המקורי.