Composite Design Pattern

תבנית העיצוב Composite היא תבנית עיצוב מבנית המאפשרת ליצור מבנה עץ של אובייקטים, כאשר כל אובייקט בעץ יכול להיות אובייקט פשוט או קבוצת אובייקטי, ולאחר מכן לעבוד עם כל האובייקטים בעץ באותה דרך.

התבנית באה לעזור לנו כאשר יש לנו אובייקט שמורכב מאובייקטים נוספים שגם הם יכולים להיות אובייקטים מורכבים, ואנחנו רוצים לטפל בכל האובייקטים באותה דרך.

התבנית בנויה משני רכיבים עיקריים:

1. **Component** - ממשק או מחלקה אבסטרקטית שמגדירה את הפונקציונליות הבסיסית של האובייקט (בסיסי או מורכב).
2. **Composite** - מחלקה שמיישמת את הממשק Component, ומכילה רשימה של אובייקטים מסוג Component, ומטפלת בהם באותה דרך.

> קישור לשרשור בstack overflow על התבנית: Composite Design Pattern

יתרונות

- גמישות - ניתן ליצור מבנה עץ של אובייקטים ולעבוד איתם בצורה אחידה.
- משפט את הקוד - ניתן לקבץ את כל הפעולות של כל האובייקטים בעץ במקום אחד.

> צריך לשים לב - לאל כל מבנה שבנוי בצורה של עץ יכול להתאים לתבנית זו. יש מקרים שבהם יש צורך להשתמש בתבנית זו ויש מקרים שבהם יש צורך להשתמש בתבנית אחרת. נשתמש בתבנית כאשר אחידות וגמישות בעבודה עם אובייקטים בעץ היא חשובה לנו.

דוגמת קוד

הדוגמה שלנו תהיה מערכת קבצים - מערכת קבצים בנויה בצורה של עץ. כל קובץ יכול להיות קובץ פשוט או קובץ מורכב (תיקייה) שמכיל קבצים נוספים.

ניצור ממשק `FileSystemItem` ומחלקות `File` ו- `Folder` שמיישמות את הממשק. כל איבר במערכת יכול להיות קובץ או תיקייה, וכאשר נקרא לפונקציה `print` כל איבר ידפיס את השם שלו.

אצלנו תיקייה תייצג צומת (עם ילדים) בגרף, וקובץ ייצג עלה בגרף.

```java
// Component interface
interface FileSystemItem {
    String getName();
    void print();
}

// Leaf (File)
class File implements FileSystemItem {
    private String name;

    public File(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void print() {
        System.out.println("File: " + name);
    }
}

// Composite (Folder)
class Folder implements FileSystemItem {
    private String name;
    private List<FileSystemItem> children;
```

```java
    public Folder(String name) {
        this.name = name;
        this.children = new ArrayList<>();
    }

    public void add(FileSystemItem item) {
        children.add(item);
    }

    public void remove(FileSystemItem item) {
        children.remove(item);
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void print() {
        System.out.println("Folder: " + name);
        for (FileSystemItem item : children) {
            item.print();
        }
    }

}

// Usage
public class Main {
    public static void main(String[] args) {
        Folder root = new Folder("Root");
        Folder folder1 = new Folder("Folder1");
        Folder folder2 = new Folder("Folder2");
        File file1 = new File("File1");
        File file2 = new File("File2");

        root.add(folder1);
        root.add(folder2);
        folder1.add(file1);
        folder2.add(file2);

        root.print();
    }
}
```