Facade pattern

תבנית ה- Facade היא תבנית עיצוב מבנית שמאפשרת לנו להסתיר את המימוש של קוד מורכב מהמשתמש. תבנית זו מאפשרת לנו ליצור ממשק פשוט וקל לשימוש שמסתיר את המימוש המורכב מהמשתמש.

התבנית באה לעזור לנו כאשר יש לנו קוד מורכב שמכיל הרבה פעולות ואנחנו רוצים להסתיר את המימוש מהמשתמש. במקום שהמשתמש יצטרך להכיר את כל הפעולות של המחלקה, הוא יוכל להשתמש בממשק פשוט שמסתיר את המימוש.

שלד של תבנית זו מורכב משלושה חלקים:

1. ממשק פשוט: ממשק פשוט שמסתיר את המימוש המורכב מהמשתמש.
2. מחלקת Facade: מחלקה שמכילה את הממשק ומכילה את הפעולות של הממשק.
3. מחלקות מורכבות: מחלקות שמכילות את המימוש המורכב של הממשק.

דוגמה לשימוש ב- Facade

נניח שינרצה לייצג מערכת קולנוע ביתית, המערכת כולל מספר רכיבים כמו מסך, רמקולים, מקרן ועוד. כל רכיב יש לו פעולות שונות שניתן לבצע עליו, כמו להדליק את המקרן, להפעיל את הרמקולים ועוד.

```
// Complex class
class Screen {
    void turnOn() {
        // turn on the screen
    }

    void turnOff() {
        // turn off the screen
    }
}

class Speakers {
    void turnOn() {
        // turn on the speakers
    }

    void turnOff() {
        // turn off the speakers
    }
}

class AudioSystem {
    void turnOn() {
        // turn on the audio system
    }
    public void setVolume(int volume) {
        // set the volume
    }

    void turnOff() {
        // turn off the audio system
    }
}

// Facade class
public class HomeTheaterFacade {
    private Screen screen;
    private Speakers speakers;
    private AudioSystem audioSystem;

    public HomeTheaterFacade() {
        this.screen = new Screen();
        this.speakers = new Speakers();
        this.audioSystem = new AudioSystem();
```

```
    }

    public void watchMovie() {
        screen.turnOn();
        speakers.turnOn();
        audioSystem.turnOn();
        audioSystem.setVolume(50);
    }

    public void turnOff() {
        screen.turnOff();
        speakers.turnOff();
        audioSystem.turnOff();
    }
}
```

עכשיו מי שיירצה לבנות מערכת קולנוע ביתית יוכל להשתמש במחלקת HomeTheaterFacade ולהשתמש בפעולות שלה כדי להפעיל את המערכת, והוא לא יצטרך להכיר את המימוש המורכב של המערכת.

יתרונות של Facade

- משתמשים מקבלים Minimal API פשוט וקל לשימוש.
- אם קורים שינויים במימוש, ניתן לשנות את המימוש במחלקת Facade בלבד, והלקוח לא צריך להיות מודע לאותם השינויים.
- אם משתמש רוצה לבנות מערכת יותר מורכבת, הוא עדיין יכול לבנות מחלקה שתשתמש במחלקות המסובכות ולא להשתמש במחלקת Facade.

עוד שימוש ל- Facade

אנחנו יכולים להשתמש בו גם כדי להציג ממשק יותר קל לעבודה עם מחלקה שיצרנו.

דוגמה: אם יש נגן מדיה מורכב, שיכול לרבל הרבה פרמטרים (עוצמת המוזיקה בצד ימין, עוצמת המוזיקה בצד שמאל, בס ועוד), נוכל ליצור מחלקת Facade שתקבל פרמטרים פשוטים ותעביר אותם לנגן המדיה המורכב.

(בדוגמה הזאת אנחנו פשוט הופפכים מממשק מורכב של מחלקה אחת [הוא צריך לקבל הרבה פרמטרים שלא לכל המשתמשים יהיו קלים לשימוש] למשק פשוט של מחלקת Facade).

```
class ComplexMediaPlayer {
    ...
    public void play(String fileName,int rightVolume, int leftVolume, int bass) {
        // play the media
    }
}

 interface SimpleMediaPlayer {
    void play(String fileName);
}


class MediaPlayerFacade implements SimpleMediaPlayer {
    private ComplexMediaPlayer complexMediaPlayer;

    public MediaPlayerFacade() {
        this.complexMediaPlayer = new ComplexMediaPlayer();
    }

    @Override
    public void play(String fileName) {
        complexMediaPlayer.play(fileName, 50, 50, 50);
    }
}
```