Singleton Design Pattern

תבנית העיצוב של Singleton היא תבנית עיצוב מסוג יצירה, שמגבילה יצירה של אובייקטים מסוג מחלקה מסויים לאובייקט אחד בלבד. תבנית זו מבטיחה שיש רק אובייקט אחד במערכת ומספקת נקודת גישה גלובלית אליו.

למה זה טוב?

נדמיין שיש לנו מחלקה שאחראית לנהל מידע מסויים, כמו database. אנחנו לא נרצה שהמידע ינוהל על ידי אובייקטים שונים, כי זה יכול לגרום לבעיות אם נעדכן אובייקט אחד ואת השני לא, ולשימוש לא יעיל של המשאבים (לשמור את אותו מידע בכמה מקומות). תבנית העיצוב של Singleton פותרת בעיה זו על ידי יצירת אובייקט אחד בלבד שמנהל את המידע.

יתרונות

1. גישה מבוקרת לאובייקט אחד בלבד - מפשט את ניהול המשאבים ומונע התנגשויות.
2. נקודת גישה גלובלית - ניתן לגשת לאובייקט מכל מקום במערכת.
3. יעיל בזכרון - כאשר יש רק אובייקט אחד במערכת, זה חוסך מקום בזיכרון, שלא ניצור אובייקטים נוספים.

דוגמה לשימוש ב- Singleton

נניח שיש לנו מערכת שאחראית לנהל טיסה מרובה של רחפנים. נרצה שיהיה רק אובייקט אחד שמנהל את כל הרחפנים במערכת.

למשל המשימות שיכולות להיות לו הן: רישום וזהוי רחפנים, לתת משימה לקבוצת רחפנים, איסוף נתונים, מניעת הנתנגשות ועוד.

בעזרת תבנית העיצוב של Singleton נוכל ליצור מחלקה שמנהלת את כל הרחפנים במערכת ולהגביל את היצירה של אובייקטים מסוג זה לאובייקט אחד בלבד, וככה הניהול של הרחפנים יהיה נכון (אם היה לנו כמה אובייקטים שמנהלים את אותו רחפן, זה היה יכול לגרום לבעיות).

```java
public class DroneFleetManager {

    private static DroneFleetManager instance;

    private List<Drone> drones;

    // Private constructor
    private DroneFleetManager() {
        drones = new ArrayList<>();
    }

    // Public method to access the single instance
    public static DroneFleetManager getInstance() {
        if (instance == null) {
            instance = new DroneFleetManager();
        }
        return instance;
    }

    // Methods for managing drones
    public void registerDrone(Drone drone) { ... }
    public void sendCommand(int droneId, Command command) { ... }
    public void receiveTelemetryData(int droneId, TelemetryData data) { ... }
    public void manageAirspace() { ... }
}
```

דוגמא לשימוש ב- Singleton בשפת Python

בניגוד לשפת Java, בשפת Python אין לנו באמת אפשרות להגביל את הגישה לבנאי, אז הדרך שלנו לעקוף את זה, היא שכל פעם שמשתמש ייצור אובייקט נחזיר את אותו אובייקט.

ניצור משתנה סטטי שיכיל את האובייקט שהיחיד של המחלקה. התפקיד של הפונקציה `__new__` הוא להקצות מקום בזכרון ולהחזיר את האובייקט שנוצר.

אז נדרוס את המיממוש הדיפולטיבי של הפונקציה `__new__` ונבדוק:

אם האובייקט עדיין לא נוצר, ניצור אותו (באמצעות הפונקציה `__new_` הדיפולטיבית של המחלקת האב), ונחזיר את האובייקט שנוצר.

אם האובייקט כבר נוצר, נחזיר את האובייקט.

(חלק מתהליך יצירת האובייקט בפייתון הוא: קריאה לפונקציה `__new__` שהיא פונקציית מחלקה שמחזירה את האובייקט שנוצר, ואז קריאה לפונקציה `__init__` שהיא מתודת אובייקט שמאתחלת את האובייקט שנוצר.)

```
class Singleton:
    _instance = None
    _initialized = False

    def __new__(cls, *args, **kwarg):
        if cls._instance is None:
            cls._instance = super(Singleton, cls).__new__(cls)
        return cls._instance

    def __init__(self):
        if Singleton._initialized
            return
        Singleton._initialized = True
        print("Singleton instance created")
```