Strategy Design Pattern

תבנית העיצוב של Strategy היא תבנית עיצוב מסוג התנהגות, היא מאפשרת להגדיר סוגים שונים של אלגוריתמים ולהחליט בזמן ריצה איזה אלגוריתם להשתמש בו. (בעצם האיזה אסטרטגיה האובייקט יעבוד ייקבע בזמן ריצה).

התבנית מאפשרת לנו להפריד בין אופן הפעולה של האובייקט מהאובייקט עצמו, ונותן לנו דרך קלה להחליף התנהגות בזמן ריצה.

התבנית משתמשת בדל

התבנית מורכבת משלושה חלקים עיקריים:

1. Strategy - ממשק שמגדיר את הפעולה של האלגוריתם.
2. ConcreteStrategy - מחלקות שמממשות את האסטרטגיה.
3. Context - מחלקה שמשתמשת באסטרטגיה ומכילה את האסטרטגיה הנבחרת.

נשתמש בתבנית כאשר:

- יש לנו כמה אלגוריתמים שיכולים לבצע אותו סוג של פעולה, ונרצה להחליט בזמן ריצה איזה אלגוריתם להשתמש.
- כשנרצה להפריד בין האלגוריתם עצמו לאובייקט שמשתמש בו (אינקפסולציה של האלגוריתם).
- כשנרצה להוסיף בהמשך אלגוריתמים נוספים ללא שינוי בקוד הקיים.

דוגמה לשימוש ב- Strategy

נניח ויש לנו אפליקציית מסחר אלקטרונית שצריכה לתמוך בכמה סוגי תשלום שונים, כמו paypal, BitCoin, כרטיס אשראי וכו'. במקום להשתמש בתנאים רבים וקוד מסורבל, נשתמש בתבנית העיצוב של Strategy ונפריד בין סוגי התשלום לפי אסטרטגיה נפרדת.

בדוגמה הזאת יש לנו:

- `PaymentStrategy` - ממשק שמגדיר התנהגות משותפת לכל אמצעי התשלום.
- `CreditCardPaymentStrategy`, `BitCoinPaymentStrategy`, `PayPalPaymentStrategy` - מחלקות שמממשות את האסטרטגיות השונות.
- `ShoppingCart` - מחלקה שמשתמשת באסטרטגיה ומכילה את האסטרטגיה הנבחרת.

בזמן הריצה נחליף את אסטרטגיית התשלום בקלות בעזרת הפעולה `setPaymentStrategy`.

```
// Strategy interface
interface PaymentStrategy {
    void pay(double amount);
}

// Concrete strategies
class PayPalPaymentStrategy implements PaymentStrategy {
    @Override
    public void pay(double amount) {
        System.out.println("Paying " + amount + " using PayPal.");
    }
}

class BitCoinPaymentStrategy implements PaymentStrategy {
    @Override
    public void pay(double amount) {
        System.out.println("Paying " + amount + " using BitCoin.");
    }
}

class CreditCardPaymentStrategy implements PaymentStrategy {
    @Override
    public void pay(double amount) {
        System.out.println("Paying " + amount + " using Credit Card.");
    }
}
// Context
```

```java
class ShoppingCart {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }

    public void checkout(double amount) {
        paymentStrategy.pay(amount);
    }
}

// Usage
ShoppingCart cart = new ShoppingCart();

// Select payment method at runtime
cart.setPaymentStrategy(new PayPalPaymentStrategy());
cart.checkout(100); // Pays using PayPal

cart.setPaymentStrategy(new BitCoinPaymentStrategy());
cart.checkout(200); // Pays using BitCoin

cart.setPaymentStrategy(new CreditCardPaymentStrategy());
cart.checkout(300); // Pays using Credit Card
```