Plik pobrano z www.lukasz-socha.pl

Wzorce projektowe, cz. 2 – Strategy

Druga część z serii wpisów o wzorcach projektowych. Dziś omówię wzorzec **Strategii (Strategy)**.

Wstęp

Strategia jest wzorcem projektowym, który definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas. Dzięki temu umożliwia wymienne stosowanie każdego z nich w trakcie działania programu.

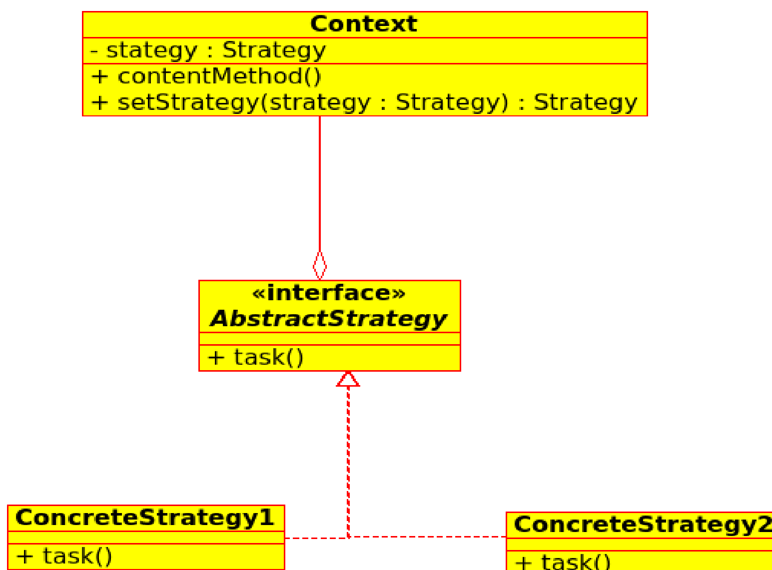

Diagram klas wzorca Strategy

Materiał przeznaczony tylko do własnego użytku. Publikacja na innych stronach wyłącznie za zgodą autora.
kontakt@lukasz-socha.pl

Przykładowa implementacja

```
<?php
interface AbstractStrategy{
    function task();
}
class ConcreteStrategy1 implements AbstractStrategy{
    public function task() {
        echo "Strategy 1";
    }
}
class ConcreteStrategy2 implements AbstractStrategy{
    public function task() {
        echo "Strategy 2";
    }
}
class Context{
    private $strategy;

    public function setStrategy(AbstractStrategy $obj) {
        $this->strategy=$obj;
    }
    public function getStrategy() {
        return $this->strategy;
    }
}
// testy
$obj = new Context();
$obj->setStrategy(new ConcreteStrategy1);
$obj->getStrategy()->task(); // wyswietla „Strategy 1”
$obj->setStrategy(new ConcreteStrategy2);
$obj->getStrategy()->task(); // wyswietla „Strategy 2”
?>
```

We wzorcu tym definiujemy wspólny interfejs dla wszystkich strategii. Następnie, w poszczególnych klasach implementujemy metody z konkretnymi już algorytmami. Za pomocą klasy ***Context*** możemy łatwo zmieniać używaną strategię w trakcie działania aplikacji.

Przykład z życia wzięty

Przypuśćmy, że tworzymy sklep internetowy oferujący swoje usługi w kilku państwach. Jak wiadomo prawo podatkowe znacząco różni się w poszczególnych krajach. Powstaje problem naliczenia odpowiedniego podatku dla klientów pochodzących z odmiennych państw. Jak to rozwiązać? Wybrać odpowiednią stawkę za pomocą licznych instrukcji warunkowych? Nie, do tego świetnie nadaje się wzorzec ***strategii***.

```
<?php

interface Tax{
    public function count($net);
}

class TaxPL implements Tax{
    public function count($net) {
        return 0.23*$net;
    }
}

class TaxEN implements Tax{
    public function count($net) {
        return 0.15*$net;
    }
}

class TaxDE implements Tax{
    public function count($net) {
        return 0.3*$net;
    }
}
```

Materiał przeznaczony tylko do własnego użytku. Publikacja na innych stronach wyłącznie za zgodą autora.
kontakt@lukasz-socha.pl

```
class Context{
    private $strategy;

    public function setCountry($country) {
        switch ($country) {
            case "PL":
                $this->strategy = new TaxPL();
                break;
            case "DE":
                $this->strategy = new TaxDE();
                break;
            case "EN":
                $this->strategy = new TaxEN();
                break;
        }
    }
    public function getTax() {
        return $this->strategy;
    }
}

// testy
$tax = new Context();
$tax->setCountry("PL");
echo $tax->getTax()->count(100); // wyswietla "23"
$tax->setCountry("EN");
echo $tax->getTax()->count(100); // wyswietla "15"
$tax->setCountry("DE");
echo $tax->getTax()->count(100); // wyswietla "30"

?>
```

Za pomocą metody ***setCountry()*** możemy w bardzo prosty sposób wybierać odpowiednią strategię obliczania podatku dostosowaną do państwa, z którego pochodzi nasz klient. Jeżeli sklep zdecyduje

Materiał przeznaczony tylko do własnego użytku. Publikacja na innych stronach wyłącznie za zgodą autora.
kontakt@lukasz-socha.pl

się na poszerzenie działalności o kolejny kraj, wystarczy tylko dopisać odpowiednią klasę implementującą ***interfejs Tax*** oraz dodać do ***switch'a*** odpowiedni warunek.

Zalety i wady

Zalety:

- Eliminacja instrukcji warunkowych – kod jest bardziej przejrzysty.
- Umożliwia wybór implementacji – algorytmy mogą rozwiązywać ten sam problem, lecz różnić się uzyskiwanymi korzyściami.
- Łatwość dołączania kolejnych strategii.
- Łatwiejsze testowanie programu – można debugować każdą strategię z osobna.

Wady:

- Dodatkowy koszt komunikacji między klientem, a strategią (wywołania metod, przekazywanie danych).
- „Rozmycie” kodu na kilka klas.

Zastosowanie

Wszędzie tam, gdzie istnieje potrzeba rozwiązania danego problemu na kilka różnych sposobów.

Oficjalna wirtualna maszyna ***Javy HotSpot*** wykorzystuje wzorzec strategii w wewnętrznej implementacji mechanizmu odśmiecania pamięci, oferując do wyboru kilka algorytmów różniących się właściwościami. Sam programista wybiera strategię odśmiecania najlepiej dopasowaną do profilu jego aplikacji.

Materiał przeznaczony tylko do własnego użytku. Publikacja na innych stronach wyłącznie za zgodą autora.
kontakt@lukasz-socha.pl