Proxy

Proxy to wzorzec pozwalający stworzyć obiekt zastępczy w miejsce innego obiektu. Proxy, zwane też pełnomocnikiem, nadzoruje dostęp do pierwotnego obiektu, pozwalając na wykonanie jakiejś czynności przed lub po przekazaniu do niego żądania.

To jeden z prostszych wzorców. Do naszego pośrednika przekazujemy docelową klasę, a sama klasa proxy implementuje ten sam interfejs co przekazany obiekt.

Dzięki temu Proxy może wykonać coś przed, czy po uruchomieniu właściwego kodu. Może to być w szczególności przydatne, jeśli określonej klasy, raczej nie powinno się używać - bo na przykład może być szkodliwa, może negatywnie wpływać na system, albo być trudna do używania.

Proxy może też pomóc w optymalizacji systemu. Można je śmiało wykorzystać do wdrożenia cache czy innych zadań, które sprawią, że nie będziesz musiał wykonywać kosztownych operacji, nadal wywołując ten sam kod.

Powiedzmy, że mamy klasę, która dostarcza nam danych z API, jednak chcemy mieć pewność, że wykonując taką samą operację, pobierzemy dane tylko raz.

Mamy dwie klasy które dostarczają nam informację o aktualnym kursie walut. Jedną pobierającą standardowe waluty, oraz taką która pozwala pobierać nam cypto.

```
interface CurrencyCalculator
{
    public function get(string $currency): float;
}

class FiatCurrencyCalculator implements CurrencyCalculator
{
    public function get(string $currency): float
    {
        echo "Właśnie pobraliśmy PLN w ".strtoupper($currency
```

```
        return match($currency) {
            'usd' => 3.8,
            'eur' => 4.3,
        };
    }
}

class CryptoCurrencyCalculator implements CurrencyCalculator
{
    public function get(string $currency): float
    {
        echo "Właśnie pobraliśmy PLN w ".strtoupper($currency
        return match($currency) {
            'btc' => 200000,
            'eth' => 5000,
        };
    }
}
```

Szczęśliwie, spełniają one ten sam interfejs. Oczywiście w tym momencie dane są bezpośrednio w klasie, ale możemy sobie wyobrazić, że dane te byłyby pobierane z zewnętrznego API, co powodowało by koszty.

Stwórzmy więc naszą klasę proxy

```
class CalculateCurrency implements CurrencyCalculator
{
    private array $cache = [];

    public function __construct(private CurrencyCalculator $c
    {
    }

    public function get(string $currency): float
    {
        if (!isset($this->cache[$currency])) {
            $result = $this->currencyValue->get($currency);
            $this->cache[$currency] = $result;
        }
```

```
        return $this->cache[$currency];
    }
}
```

Zauważ, że klasa ta implementuje `CurrencyCalculator` tak jak wszystkie klasy dla których jest Proxy. Jeśli dla określonej klasy pobierzemy już waluty, to nie będziemy tego już robić.

Tak więc nasza implementacja będzie wyglądała następująco

```
function calculate(CurrencyCalculator $currencyValue, string $
{
    $calculator = new CalculateCurrency($currencyValue);
    echo $calculator->get($currency)."zł \n";
    echo "Teraz już mamy tą wartość, więc nie pobieramy jej p
    echo $calculator->get($currency)."zł \n";
}

calculate(new FiatCurrencyCalculator(), 'usd');
calculate(new FiatCurrencyCalculator(), 'eur');
echo "\n-----\n";
calculate(new CryptoCurrencyCalculator(), 'btc');
calculate(new CryptoCurrencyCalculator(), 'eth');
```

Wady i zalety

Zalety:

- Można sterować obiektem usługi bez wiedzy klientów.
- Można zarządzać cyklem życia obiektu usługi, gdy klientów to nie interesuje.
- Pełnomocnik działa nawet wtedy, gdy obiekt udostępniający usługę nie jest jeszcze gotowy lub dostępny.
- Zasada otwarte/zamknięte. Można wprowadzać nowych pełnomocników do aplikacji bez modyfikowania usług lub klientów.

Wady:

- Kod może ulec skomplikowaniu, ponieważ trzeba wprowadzić wiele nowych klas.
- Odpowiedzi ze strony usługi mogą ulec opóźnieniu.