Clone / Prototype

Ten film będzie szybki. Dlaczego? Bo ten zły język, który przez tak wielu niedoświadczonych programistów, jest tak bardzo krytykowany... Tak - mam na myśli PHPa. Ma tą funkcjonalność zbudowaną natywnie i to od dawna.

Powiedzmy że mamy dwa obiekty w różnych stanach i chcemy rozważyć, jak zachowają się w przypadku określonych zmian. W tym wypadku przydało by się mieć dokładnie dwa takie obiekty, które nie będą się zmieniać i tutaj cała na biało wchodzi funkcja `clone`.

Używa się jej wtedy, gdy chcesz utworzyć nowy obiekt na podstawie istniejącego obiektu, ale zachować niezależność między nimi. Obiekty w PHP przekazywane są przez referencję, czego mogłeś doświadczyć w poprzednich lekcjach. Jeśli więc wpływamy na obiekt, to zmienia on swój stan. W przypadku, gdy obiekt zawiera referencje do innych obiektów lub zasobów, bez użycia `clone`, oba obiekty będą wskazywać na te same referencje, co może prowadzić do nieoczekiwanych efektów ubocznych. `clone` zapewnia, że otrzymujesz nowy obiekt z takimi samymi wartościami, ale niezależny od oryginalnego obiektu.

Zastosowania `clone` obejmują:

Kopiowanie obiektów: Kiedy chcesz utworzyć kopię obiektu, ale nie chcesz, aby zmiany dokonane na jednym obiekcie wpływały na drugi.

```
class MyClass {
    public string $data;
}

$obj1 = new MyClass();
$obj1->data = "Hello";

// Tworzymy kopię obiektu $obj1
$obj2 = clone $obj1;
$obj3 = $obj1;

echo $obj1->data; // Output: Hello
echo $obj2->data; // Output: Hello
echo $obj3->data; // Output: Hello
```

```
$obj2->data = "World";

echo $obj1->data; // Output: Hello
echo $obj2->data; // Output: World
echo $obj3->data; // Output: Hello
```

Zabezpieczenie przed zmianami: Kiedy potrzebujesz przekazać obiekt innym częściom kodu, ale nie chcesz, aby którykolwiek z nich mógł zmienić stan pierwotnego obiektu.

```
class ImmutableClass {
    public string $data;

    public function __construct($data) {
        $this->data = $data;
    }
}

function short(ImmutableClass $obj): string
{
    $obj->data = substr($obj->data, 0, 3);
    return $obj->data;
}

$obj1 = new ImmutableClass("Original");

echo short(clone $obj1); // Output: Ori
echo $obj1->data; // Output: Original - no changes thanks t
o clone
```

Używanie `clone` należy dobrze przemyśleć, ponieważ niektóre obiekty mogą zawierać referencje do zasobów lub obiektów, których kopiowanie nie jest pożądane lub nawet niemożliwe. W takich przypadkach może być konieczne zaimplementowanie własnej metody kopiowania, która uwzględnia specyficzne dla danego obiektu operacje.

Jeśli jednak chcemy, bez żadnych problemów możemy zarządzać tym, co i w jaki sposób jest klonowane za pośrednictwem magicznej metody `__clone()`. Metoda ta jest automatycznie wywoływana, gdy obiekt jest klonowany za pomocą operatora `clone`.

Jeśli więc jakiś parametrów nie chcemy przekazywać, możemy skorzystać z tej metody by przekazać tylko to co nas interesuje.

```
class MyClass {
    public $data = 'default';

    public function __clone() {
        $this->data = 'cloned';
    }
}

$obj1 = new MyClass();
$obj2 = clone $obj1;

echo $obj1->data; // Output: default
echo $obj2->data; // Output: cloned
```

Jeśli natomiast chcemy wykonać “głęboki clone”, to tutaj również metoda __clone może nam pomóc.

```
class MyClass {
    public SomeOtherClass $data;

    public function __clone() {
        // Przykład głębokiej kopii, kopiujemy również obiekt
        $this->data = clone $this->data;
    }
}

class SomeOtherClass {
    public $data;
}

$obj1 = new MyClass();
```

```
$obj1->data = new SomeOtherClass();
$obj1->data->data = "Hello";

$obj2 = clone $obj1;
$obj3 = $obj1;
$obj1->data->data = "World";

echo $obj1->data->data; // Output: World
echo $obj2->data->data; // Output: Hello
echo $obj3->data->data; // Output: World
```

Myślę że już dobrze widzisz, do czego można użyć funkcji clone. Sądzę też że rozumiesz czym jest mutowalność i niemutowalność. To taki mały bonusik.

Czyli co, ten PHP jednak nie taki wcale zły i niektóre jego narzędzia są naprawdę dobre!

Wady i zalety

zalety:

- Możesz klonować obiekty bez konieczności sprzęgania ze szczegółami ich konkretnych klas.
- Możesz pozbyć się wielokrotnie powtarzanego kodu inicjalizacyjnego na rzecz klonowania prefabrykowanych prototypów.
- Dużo wygodniejsze produkowanie złożonych obiektów.
- Podejście to stanowi alternatywę do dziedziczenia w przypadku gdy mamy do czynienia z wcześniej zdefiniowanymi konfiguracjami złożonych obiektów.

wady:

• Klonowanie złożonych obiektów, które mają odniesienia cykliczne, może być trudne.