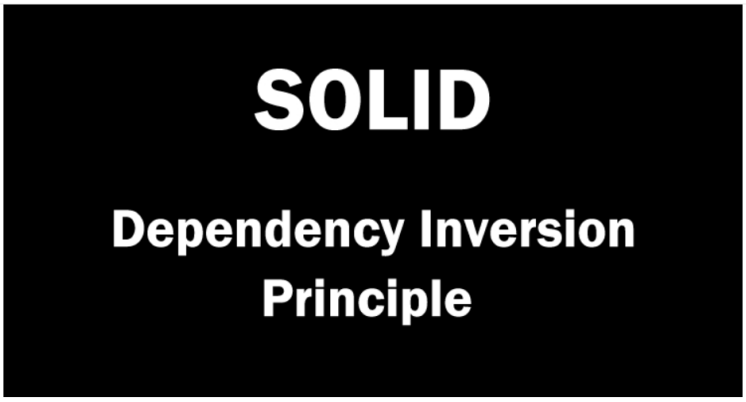


Dependency Inversion Principle in PHP (in Arabic)

Published on September 14, 2021

Osama Mustafa
PHP Back-End Developer
4 articles ✓ Following

مبدأ Dependency Inversion Principle هو المبدأ رقم (5) في الـ SOLID Principles و هي مجموعة من المبادئ في الـ Object Oriented Design و هم كالآتي:

1. Single Responsibility Principle
2. Open Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. **Dependency Inversion Principle**

دي المبادئ اللي اتشرحت قبل كده

- Liskov Substitution Principle | **https://bit.ly/2XeCebf**
- Interface Segregation Principle | **https://bit.ly/3no1573**
- Open Closed Principle | **https://bit.ly/3EbMnpC**

خلال هذه المقالة هنستخدم الاختصار (DIP) عوضاً عن المصطلح الكامل (**D**ependency **I**nversion **P**rinciple)

قبل ما نقول تعريف المبدأ، تعالى نشوف ايه المشكلة اللي المبدأ ده اتعمل علشان يحلها، دلوقتي أنا عندي system عبارة عن لمبة (Lamp) ومفتاح لتشغيل الكهرباء (Button)، اللمبة طبعاً متوصلة على المفتاح، المفتاح ده عنده وظيفتين و هما تشغيل اللمبة (Switch On) أو أنه يطفيها (Switch Off)

ده معناه بديهياً أن لو المفتاح في وضعية التشغيل (On) بالتالي اللمبة هتكون شغالة والنور هيكون موجود؛ أما لو المفتاح في وضعية (Off) فاللمبة هتكون مطفية والاوضة هتكون ضلمة

طيب تعالى نشوف Lamp Class وايه الميثودز الموجودة فيها في *Image 1*

Image 1

```php
class Lamp {

    //TRUE = on, FALSE = Off
    protected bool $currentState = FALSE;

    public function turnOn() {
        $this->currentState = TRUE;
    }

    public function turnOff() {
        $this->currentState = FALSE;
    }

    public function getState() {
        return $this->currentState;
    }

    public function getStateString(): string {
        if ($this->currentState) {
            return 'On';
        } else {
            return 'Off';
        }
    }
}
```


إحنا قلنا من شوية إن اللمبة والمفتاح متوصلين مع بعض، طيب تعالى نشوف الـ Button Class في *Image 2*

Image 2

```php
class Button {
    protected Lamp $lamp;

    public function __construct(Lamp $l) {
        $this->lamp = $l;
    }

    public function On() {
        $this->lamp->turnOn();
    }

    public function Off() {
        $this->lamp->turnOff();
    }
}

$l = new Lamp();
$b = new Button($l);
echo $l->getStateString(); //Off

$b->On();
echo $l->getStateString(); //On
```

في صورة *Image 2* هتلاحظ إن Button Class بيستخدم الـ Lamp Class، أو بمعنى تاني المفتاح الكهربائي معتمد على اللمبة، لأننا بعتنا object من اللمبة في الـ Constructor بتاع Button Class، وده غلط وبيعتبر انتهاك لمبدأ DIP لأني ممكن في أي وقت بعد كده، احتاج إني اوصل المفتاح على أي جهاز تاني غير اللمبة وليكن جهاز كمبيوتر أو شاشة تلفزيون، كده أنا مش هقدر استخدم المفتاح إلا ومعاه اللمبة

طب ايه تعريف مبدأ DIP ؟

High level classes should not depend upon (A)
low level classes, both should depend on
abstractions

Abstractions should not depend on details, (B)
details should depend on abstractions

محتاجين دلوقتي نعرف مين هيا الـ Classes اللي تعتبر High Level عشان نقدر نطبق المبدأ بشكل صحيح، لو عندي Class اسمها (B) بتستخدم Class تانية اسمها (A) ، أو بطريقة تانية ممكن نقول إن Class B معتمدة على Class A بالتالي التقسيم هيكون كالأتي:

Class A => Low Level

Class B => High Level

طب هيا ايه المشكلة في الكود الموجود فوق ؟ واللي مخلياه بينتهك مبدأ DIP

1- أولاً المفتاح الكهربي معتمد بشكل أساسي على اللمبة

2- مقدرش استخدم المفتاح في أي مكان إلا مع استخدام اللمبة

طيب تعالى نحل المشكلة باتباع مبدأ DIP

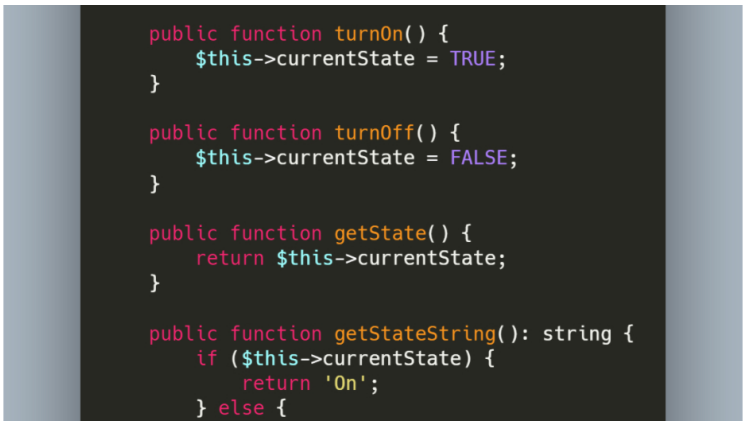
المبدأ بتاعنا متقسم لكلمتين وهما (Dependency) ومعناها "الاعتماد" وكلمة (Inversion) ومعناها "عكس"، فالمبدأ بيقولك باختصار إن المفتاح اللي هو يعتبر (High Level Modules) مينفعش يعتمد على اللمبة اللي هيا تعتبر (Low Level Module) ، ولكن الاتنين لازم يعتمدوا على الـ Abstraction ويُقصد بيه الـ Interface، بحيث إني أقدر استخدم المفتاح في أي وقت مع أي جهاز تاني ... كمان التفاصيل الموجودة في الكود تكون معتمدة على الـ Abstraction وليس العكس

حل المشكلة إني أعمل Interface وليكن مثلاً (DeviceInterface) واللمبة هتعمل implements للـ Interface ده وهتعمل override على الـ methods الموجودة فيه، زي المثال الموجود في صورة *Image 3*

Image 3

```php
interface DeviceInterface {
    public function turnOn();
    public function turnOff();
}

class Lamp implements DeviceInterface {
    //TRUE = on, FALSE = Off
    protected bool $currentState = FALSE;
```



```php
    public function turnOn() {
        $this->currentState = TRUE;
    }

    public function turnOff() {
        $this->currentState = FALSE;
    }

    public function getState() {
        return $this->currentState;
    }

    public function getStateString(): string {
        if ($this->currentState) {
            return 'On';
        } else {
```