Published in Dev Genius

.com software Follow
Aug 9, 2022 · 2 min read · · Listen

Save

Clean code tricks in PHP everyone should follow

Improve your code drastically with these few simple tricks. Today I would like to share my coding techniques with you.

Photo by Ben Griffiths on Unsplash

I have a few techniques to share to improve your coding skills and make your code more friendly for people to read and digest.

I will write each technique before and after the change. Let us jump straight into the business.

Keep code indentation as small as possible

Before

```php
<?php

class DiscountApplier
{
    private DiscountCalculator $calculator;

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

    public function apply(Basket $basket): void
    {
        if ($basket->hasItems()) {
            $discount = 0.0;

            foreach ($basket->getItems() as $basketItem) {
                if ($basketItem->isTaxable()) {
                    if ($basketItem->getPrice() > 0) {
                        $discount += $this->calculator->getDiscount($basketItem);
                    }
                }
            }

            $basket->applyDiscount($discount);
        }
    }
}
```

cc-indentation-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

class DiscountApplier
{
    private DiscountCalculator $calculator;

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

    public function apply(Basket $basket): void
    {
        if (!$basket->hasItems()) {
            return;
        }

        $discount = 0.0;

        foreach ($this->getDiscountableItems($basket) as $basketItem) {
            $discount += $this->calculator->getDiscount($basketItem);
        }

        $basket->applyDiscount($discount);
    }

    /**
     * @return iterable<BasketItem>
     */
    private function getDiscountableItems(Basket $basket): iterable
    {
        foreach ($basket->getItems() as $basketItem) {
            if (!$basketItem->isTaxable() || $basketItem->getPrice() === 0) {
                continue;
            }

            yield $basketItem;
        }
    }
}
```

cc-indentation-after.php hosted with ❤ by GitHub view raw

Remove unnecessary condition clauses

Before

```php
<?php

function mapOrderStatusToLabel(Order $order): string
{
    if ($order->isComplete()) {
        return 'Completed';
    } elseif ($order->isPending()) {
        return 'In transport';
    } else {
        return 'New';
    }
}
```

cc-condition-clauses-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

function mapOrderStatusToLabel(Order $order): string
{
    if ($order->isComplete()) {
        return 'Completed';
    }

    if ($order->isPending()) {
        return 'In transport';
    }

    return 'New';
}
```

cc-condition-clauses-after.php hosted with ❤ by **GitHub** **view raw**

Invert conditions to reduce code indentations

Before

```php
<?php

function act(bool $success)
{
    if ($success) {
        // towering block of code
        // towering block of code
        // towering block of code
        // towering block of code
        // towering block of code
        // towering block of code
        // towering block of code
    } else {
        // one liner
    }
}
```

cc-invert-conditions-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

function act(bool $success)
{
    if (!$success) {
        // one liner
        return;
    }

    // towering block of code
    // towering block of code
    // towering block of code
    // towering block of code
    // towering block of code
    // towering block of code
    // towering block of code
}
```

cc-invert-conditions-after.php hosted with ❤ by **GitHub** **view raw**

Prefer switches over if-conditions when possible

Before

```php
<?php

function mapOrderStatusToLabel(Order $order): string
{
    $label = 'unknown';

    if ('complete' === $order->getStatus()) {
        $label = 'Completed';
    }

    if ('pending' === $order->getStatus()) {
        $label = 'In transport';
    }

    if ('new' === $order->getStatus()) {
        $label = 'New';
    }

    return $label';
}
```

cc-switches-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

function mapOrderStatusToLabel(Order $order): string
{
    switch ($order->getStatus()) {
        case 'complete':
            return 'Completed';

        case 'pending':
        case 'in_transport':
            return 'In transport';

        case 'new':
            return 'New';

        default:
            return 'unknown';
    }
}
```

cc-switches-after.php hosted with ❤ by **GitHub** **view raw**

Reformat complex equations for brevity

Before

```php
<?php

function getTotalWithTax(Payment $payment, int $taxPercent): float
{
    // is the order correct at all?

    return (float) (int) $payment->getTotal() * (1 + $taxPercent / 100);
}
```

cc-parenthesis-before.php hosted with ❤ by GitHub view raw

After

```php
<?php

function getTotalWithTax(Payment $payment, int $taxPercent): float
{
    // descriptive variables with intermediary values for easier debugging
    $paymentAmount = (int) $payment->getAmount();
    $taxCoefficient = 1 + ($taxPercent / 100);

    $totalWithTax = $paymentAmount * $taxCoefficient;

    return (float) $totalWithTax;
}
```

cc-parenthesis-after.php hosted with ❤ by GitHub view raw

Rearrange conditions in if-statements

Before

```php
<?php

if ($httpApi->hasAccess($user) && $user->isActive() && $isSuccess) {
    // all conditions must be met in order for this condition to execute
    // in case of 'and' we should keep the heaviest conditions last
    // and the fastest to execute in front
}
```

cc-conditions-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

if ($isSuccess && $user->isActive() && $httpApi->hasAccess($user)) {
    // if $isSuccess is false
    // other conditions will not be evaluated
}
```

cc-conditions-after.php hosted with ❤ by **GitHub** **view raw**

Prefer named methods over flags

Before

```php
<?php

class UserRepository
{
    public function getUsers(?bool $includingDeleted = false): array
    {
    }
}
```

cc-flags-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

class UserRepository
{
    public function getUsers(): array
    {
    }

    public function getUsersIncludingDeleted(): array
    {
    }
}
```

cc-flags-after.php hosted with ❤ by GitHub view raw

Use array destructuring

Before

```php
<?php

$workers = [
  ['id' => 1, 'name' => 'John Doe'],
  ['id' => 3, 'name' => 'Jane Doe'],
  ['id' => 4, 'name' => 'Monica Trullo'],
  ['id' => 5, 'name' => 'Jonathan Bank'],
];

foreach ($workers as $worker) {
  $workerId = $worker['id'];
  $workerName = $worker['name'];

  // …
}
```

cc-array-destruct-before.php hosted with ❤ by GitHub view raw

After

```php
<?php

$workers = [
  ['id' => 1, 'name' => 'John Doe'],
  ['id' => 3, 'name' => 'Jane Doe'],
  ['id' => 4, 'name' => 'Monica Trullo'],
  ['id' => 5, 'name' => 'Jonathan Bank'],
];

foreach ($workers as ['id' => $workerId, 'name' => $workerName]) {
  // …
}
```

cc-array-destruct-after.php hosted with ❤ by **GitHub** **view raw**

Document output structures with annotations

Before

```php
<?php

/**
 * @return array
 */
function basketToViewModel(Basket $basket): array
{
    $items = \array_map(function (BasketItem $item): array {
        return [
            'id' => $item->getProductId(),
            'name' => $item->getProductName(),
        ];
    }, $basket->getItems());

    return [
        'total' => $basket->getTotal(),
        'items' => $items,
    ];
}
```

cc-output-before.php hosted with ❤ by **GitHub** **view raw**

After

```php
<?php

/**
 * @return array{total: float, items: array<array{id: int, name: string}>}
 */
function basketToViewModel(Basket $basket): array
{
    // function body skipped for brevity

    // output annotations especially useful if using tools like PsalmPHP
    // PHPStorm is increasing its support for generics too

    // see how amazing validation can you get
    // https://psalm.dev/r/4a32324404
}
```

cc-output-after.php hosted with ❤ by **GitHub** **view raw**

Document output primitives with annotations

Before

```php
<?php

class User {
    public const STATUS_NEW = 'new';
    public const STATUS_ACTIVE = 'active';
    public const STATUS_BLOCKED = 'blocked';

    private string $status = self::STATUS_NEW;
    private int $age;

    public function getStatus(): string
    {
        return $this->status;
    }

    public function getAge(): int
    {
        return $this->age;
    }

    public function getClassName(): string
    {
        return get_class($this);
    }
}
```

cc-output-prim-before.php hosted with ❤ by **GitHub** **view raw**

After

```
        * @return self::STATUS_*
        */
       public function getStatus(): string
       {
           return $this->status;
       }

       /**
        * @return positive-int
        */
       public function getAge(): int
       {
           return $this->age;
       }

       /**
        * @return class-string<User>
        */
       public function getClassName(): string
       {
           return get_class($this);
       }
   }
```

cc-output-prim-after.php hosted with ❤ by **GitHub** **view raw**

I would also like to remind you about the following things when writing any code.

Remember that you're writing the code for your future self. The code may seem obvious at the moment. Let it stay like that next month:

- prefer descriptive variable names `$index` over `$i`
- prefer descriptive method names `getProductTotalForDisplay` over `getTotal`
- prefer descriptive constant names `const CACHE_TTL_WEEK = 7 * 86400` over `const TTL = 604800`

- use any static analysis tool, I recommend Psalm

PHP Php Developers Php Development Clean Code Laravel