Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Статические методы перечислений »
« Типизированные перечисления

- Руководство по PHP
- Справочник языка
- Перечисления

Методы перечислений

Перечисления (как чистые, так и типизированные) могут содержать методы и могут реализовывать интерфейсы. Если перечисление реализует интерфейс, то любая проверка типа этого интерфейса также примет и все варианты этого перечисления.

```
<?php

interface Colorful
{
public function color(): string;
}

enum Suit implements Colorful
{
case Hearts;
case Diamonds;
case Clubs;
case Spades;

// Выполняет контракт интерфейса.
public function color(): string
{
return match($this) {
Suit::Hearts, Suit::Diamonds => 'Красный',
Suit::Clubs, Suit::Spades => 'Чёрный'
};
}

// Не часть интерфейса; хорошо.
public function shape(): string
{
return "Rectangle";
}
}

function paint(Colorful $c)
{
/* ... */
}

paint(Suit::Clubs); // Работает

print Suit::Diamonds->shape(); // выведет "Rectangle"
?>
```

В этом примере каждый из четырёх экземпляров `Suit` имеет два метода: `color()` и `shape()`. В вызывающем коде и при проверке типов экземпляры перечисления ведут себя точно так же, как и любой другой экземпляр объекта.

В типизированных перечислениях объявление интерфейса идёт после объявления типа перечисления.

```
<?php

interface Colorful
{
public function color(): string;
}

enum Suit: string implements Colorful
{
case Hearts = 'H';
```

```
case Diamonds = 'D';
case Clubs = 'C';
case Spades = 'S';

// Выполняет интерфейсный контракт.
public function color(): string
{
return match($this) {
Suit::Hearts, Suit::Diamonds => 'Красный',
Suit::Clubs, Suit::Spades => 'Чёрный'
};
}
}
?>
```

Переменная `$this` определена внутри метода и ссылается на экземпляр варианта.

Сложность методов в перечислениях не ограничена, но на практике методы перечислений чаще возвращают статическое значение или результат обработки переменной `$this` выражением match, чтобы результаты обработки отдельных экземпляров перечисления отличались.

Обратите внимание, в этом примере более хорошей практикой построения данных было бы — определить тип перечисления `SuitColor` со значениями Red и Black и возвращать их вместо строковых литералов. Однако это усложнило бы пример.

Иерархия в примере логически похожа на следующую структуру классов (хотя это не настоящий исполняемый код):

```
<?php

interface Colorful
{
public function color(): string;
}

final class Suit implements UnitEnum, Colorful
{
public const Hearts = new self('Hearts');
public const Diamonds = new self('Diamonds');
public const Clubs = new self('Clubs');
public const Spades = new self('Spades');

private function __construct(public readonly string $name) {}

public function color(): string
{
return match($this) {
Suit::Hearts, Suit::Diamonds => 'Красный',
Suit::Clubs, Suit::Spades => 'Чёрный'
};
}

public function shape(): string
{
return "Прямоугольник";
}

public static function cases(): array
{
// Недопустимый метод, поскольку определение метода cases() в перечислениях вручную запрещено.
// Смотрите также раздел "Список значений".
}
}
?>
```

В перечислениях разрешено объявлять общедоступные, закрытые и защищённые методы, хотя на практике закрытые и защищённые методы эквивалентны, поскольку наследование не разрешено.

User Contributed Notes 1 note

0
iloveyii at yahoo dot com ¶
6 months ago

```
Just to complete the function shape in the above neat example:

<?php
interface Colorful
{
public function color(): string;
}

enum Suit implements Colorful
{
case Hearts;
case Diamonds;
case Clubs;
case Spades;

// Fulfills the interface contract.
public function color(): string
{
return match($this) {
Suit::Hearts, Suit::Diamonds => 'Red',
Suit::Clubs, Suit::Spades => 'Black',
};
}

// Not part of an interface; that's fine.
public function shape(): string
{
return match($this) {
Suit::Hearts => '❤',
Suit::Diamonds => '♦',
Suit::Clubs => '♣',
Suit::Spades => ' ♠'
};

}
}

echo Suit::Diamonds->shape();
echo PHP_EOL;
echo Suit::Clubs->shape();
```