Интерфейсы

Отличия и сходства интерфейсов от классов

- Нельзя создать «экземпляр интерфейса».
- Интерфейсы, как и классы, могут находиться в пространстве имён.
- Интерфейсы, как и классы, можно загружать через механизм автозагрузки. Функции автозагрузки будет передано полное имя интерфейса (с пространством имён).
- В каждом интерфейсе есть предопределенная константа ThisInterface::class, содержащая его полное имя
- Интерфейс, как и класс, может участвовать справа в операторе instanceof
- Интерфейс, как и класс, может быть указан в качестве типа в тайп-хинтинге (указание типа аргумента либо возвращаемого значения функции)

Тонкости реализации интерфейсов

Важный аспект, который отличает реализацию интерфейса от наследования от другого класса — это возможность реализовать в одном классе несколько интерфейсов сразу.

Во-первых действительно, наследование класса от интерфейса называется реализацией.

Смысл в том, что вы не просто получаете в наследство методы и константы, но обязаны реализовать те методы, которые заданы сигнатурами, наполнить их кодом:

```
interface IntSumInterface
{
  public function sum(int $x, int $y): int;
}

interface IntMultInterface
{
  public function mult(int $x, int $y): int;
}

class Math
  implements IntSumInterface,
IntMultInterface
{
  public function sum(int $x, int $y): int
  {
    return $x + $y;
  }

  public function mult(int $x, int $y): int
  {
    return $x * $y;
  }
}
```

Наследование

Если ли в PHP множественное наследование?

Интерфейс может наследоваться от нескольких других интерфейсов.

```
interface First
{
  public function foo(int $x);
}

interface Second
{
  public function bar(string $s);
}

interface Third
  extends First, Second
{
  public function baz(array $a);
}
```

Правила решения конфликтов сигнатур методов при множественном наследовании точно такие же, как мы уже видели выше:

— либо сигнатуры совпадают полностью

— либо сигнатура метода интерфейса, упомянутого в списке предков первым, должна быть совместима с сигнатурой из второго предка (да, порядок упоминания имеет значение, но это очень редкий кейс, просто не принимайте его никогда во внимание)

Наследование интерфейсов

Интерфейсы могут наследоваться друг от друга:

```
interface First
{
  public const PI = 3.14159;
  public function foo(int $x);
}

interface Second
  extends First
{
  public const E = 2.71828;
  public function bar(string $s);
}

assert(3.14159 === First::PI);
assert(true === method_exists(First::class, 'foo'));

assert(3.14159 === Second::PI);
assert(2.71828 === Second::E);
assert(true === method_exists(Second::class, 'foo'));
assert(true === method_exists(Second::class, 'bar'));
```

Интерфейс-наследник получает от интерфейса-предка в наследство все определенные в предке методы и константы.

В интерфейсе-наследнике можно переопределить метод из родительского интерфейса. Но только при условии, что либо его сигнатура будет в точности совпадать с сигнатурой родительского, либо будет совместима (см. предыдущий раздел):

```
interface First
{
  public function foo(int $x);
}

interface Second
  extends First
{//...}
```

- // Так можно, но бессмысленно
  public function foo(int $x);
- // Так нельзя, фатальная ошибка Declaration must be compatible
  public function foo(int $x, int $y);
- // Так можно, потому что эта сигнатура совместима с родительской - мы просто добавили необязательный аргумент
  public function foo(int $x, int $y = 0);
- // Так тоже можно, все аргументы после "..." являются необязательными
  public function foo(int $x, ...$args);
- // И так тоже можно
  public function foo(int $x, ...$args): int;

Основная тема 2

Интерфейсы

Определение

Интерфейсы впервые появились в **PHP 5**

Интерфейсы объектов позволяют создавать код, который указывает, какие методы должен реализовать класс, без необходимости определять, как именно они должны быть реализованы.

Интерфейсы разделяют пространство имён с классами и трейтами, поэтому они не могут называться одинаково.

Сигнатуры

Совместимость сигнатур методов

Для дальнейшего изучения интерфейсов нам с вами нужно узнать о важнейшем понятии, которое незаслуженно обойдено вниманием в мануале по PHP: о **понятии «совместимости сигнатур»**.

Сигнатура — это описание функции (метода), включающее в себя:

- **Модификатор доступа**
- **Имя функции (метода)**
- **Список аргументов, где для каждого аргумента указано**
  - Тип
  - Имя
  - Значение по умолчанию
  - либо оператор «три точки»
- **Тип возвращаемого значения**

Пример

```
function ();
public function foo($arg = null);
protected function sum(int $x, int $y, ...$args):
int;
```

Предположим, что у нас есть две функции, A и B.

Сигнатура функции B считается совместимой с A (порядок важен, отношение несимметрично!) в строгом смысле, если:

- **Они полностью совпадают**
- **В добавляет к A аргументы по умолчанию**
  - **A:**
    ```
    function foo($x);
    ```
  - **совместимые B:**
    ```
    function foo($x, $y = null);
    function foo($x, ...$args);
    ```
- **В сужает область значений A**
  - **A:**
    ```
    function foo(int $x);
    ```
  - **совместимые B:**
    ```
    // B A допускался возврат любых значений, в B эта область сужена только до целых чисел
    function foo(int $x): int;
    ```

Содержание интерфейса

Чего не может содержать интерфейс?

Больше ничего не может. Кроме заголовков публичных методов и публичных констант.

Нельзя включать в интерфейс:

- **Любые свойства**
- **Непубличные методы**
- **Методы с реализацией**
- **Непубличные константы**

Что может содержать интерфейс?

Очевидно, что публичные методы, причем без реализации: сразу после заголовка (сигнатуры) метода следует закончить его точкой с запятой:

```
interface SomeInterface
{
  public function foo();
  public static function bar(Baz $baz);
}
```

Чуть менее очевиден (хотя и описан в мануале) тот факт, что интерфейс может содержать константы (разумеется, только публичные!):

```
interface SomeInterface
{
  public const STATUSES = [
    'OK'    => 0,
    'ERROR' => 1,
  ];
}

if (SomeInterface::STATUSES['OK'] === $status) {
  // ...
}
```

Почему же константы в интерфейсах не получили широкого распространения в промышленном коде, хотя и используются иногда?

Причина в том, что их невозможно переопределить в интерфейсе-наследнике или в классе, реализующем данный интерфейс. Константы интерфейсов — самые константные константы в мире :)