Интерфейсы в ООП на PHP

В предыдущем уроке мы с вами изучили абстрактные классы.

Как вы уже знаете, абстрактные классы представляют собой набор методов для своих потомков. Часть этих методов может быть реализована в самом классе, а часть методов может быть объявлена абстрактными и требовать реализации в дочерних классах.

Представим себе ситуацию, когда ваш абстрактный класс представляет собой только набор **абстрактных публичных** методов, не добавляя методы с реализацией.

Фактически ваш родительский класс описывает *интерфейс* потомков, то есть набор их **публичных** методов, обязательных для реализации.

Зачем нам такое нужно: чтобы при программировании совершать меньше ошибок - описав все необходимые методы в классе-родителе, мы можем быть уверенны в том, что все потомки их действительно реализуют.

Когда это поможет: пусть мы создадим наш класс-родитель и несколько потомков в нему. Если потом через некоторое время, например, через месяц, мы решим создать еще одного потомка, наверняка мы уже забудем детали нашего кода и вполне можем забыть написать реализацию какого-нибудь метода в новом потомке. Однако сам PHP не позволит потерять метод - и просто выдаст ошибку.

То же самое касается другого программиста, работающего с вашим проектом. Пусть код класса-родителя писали вы, а затем ваш коллега решил создать еще одного потомка. У вашего коллеги также не получится потерять парочку методов.

Есть, однако, проблема: фактически мы сделали наш класс-родитель для того, чтобы писать в нем абстрактные публичные методы, но мы сами или наш коллега имеем возможность случайно добавить в этот класс НЕ публичный метод или НЕ абстрактный.

Пусть мы хотим физически запретить делать в родителе иные методы, кроме абстрактных публичных.

В PHP для этого вместо абстрактных классов можно использовать **интерфейсы**.

Интерфейсы представляют собой классы, у которых все методы являются публичными и не имеющими реализации. Код методов должны реализовывать классы-потомки интерфейсов.

Интерфейсы объявляются так же, как и обычные классы, но используя ключевое слово **interface** вместо слова **class**.

Для наследования от интерфейсов используется немного другая терминология: говорят, что классы не наследуют от интерфейсов, а *реализуют* их. Соответственно вместо слова **extends** следует использовать ключевое слово **implements** (переводится как *реализует*).

Замечание: нельзя создать объект интерфейса.

Еще замечание: все методы интерфейса не должны иметь реализации.

Еще замечание: все методы интерфейса должны быть объявлены как **public**.

Кроме того, у интерфейса могут быть только методы, но не свойства.

Нельзя также сделать интерфейс и класс с одним и тем же названием.

Попробуем на практике

Давайте попробуем на практике. Решим задачу на фигуры из предыдущего урока, но уже используя интерфейсы, а не абстрактные классы.

Итак, теперь у нас дан интерфейс **Figure**:

```php
<?php
  interface Figure
  {
    public function getSquare();
    public function getPerimeter();

```

```
    }
?>
```

Обратите внимание на то, что ключевое слово **abstract** в описании метода писать не нужно.

Давайте напишем класс **Quadrate**, который будет реализовывать методы этого интерфейса:

```php
<?php
  class Quadrate implements Figure
  {
    private $a;

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

    public function getSquare()
    {
      return $this->a * $this->a;
    }

    public function getPerimeter()
    {
      return 4 * $this->a;
    }
  }
?>
```

Как это работает: если забыть реализовать какой-нибудь метод, описанный в интерфейсе, PHP выдаст нам фатальную ошибку.

Давайте реализуем также класс **Rectangle**:

```php
<?php
  class Rectangle implements Figure
  {
    private $a;
    private $b;

    public function __construct($a, $b)
    {
      $this->a = $a;
      $this->b = $b;
    }

    public function getSquare()
    {
      return $this->a * $this->b;
    }

    public function getPerimeter()
    {
      return 2 * ($this->a + $this->b);
    }
  }
?>
```

Задача 36.1

Сделайте класс **Disk** (круг), реализующий интерфейс **Figure**.

Замечание

Как уже было написано выше, не может быть интерфейса и класса с одинаковым названием. Это создает некоторые проблемы с придумыванием названий.

Например, мы хотим сделать класс **User**, реализующий интерфейс **User**. Как мы видим, у нас конфликт имен. Для его разрешения, нужно или класс назвать по-другому, или интерфейс. Общепринято в таком случае название интерфейса начать с маленькой буквы **i**, чтобы показать, что это интерфейс, а не класс. То есть в нашем случае мы сделаем интерфейс **iUser**, а реализовывать его будет класс **User**. Такой подход мы иногда будем применять в следующих уроках.