Несколько интерфейсов

В PHP нет множественного наследования - каждый класс может иметь только одного родителя.

С интерфейсами дело, однако, обстоит по другому: каждый класс может реализовывать любое количество интерфейсов.

Для этого имена интерфейсов нужно перечислить через запятую после ключевого слова **implements**.

В этом проявляется еще одно отличие интерфейсов от абстрактных классов - можно реализовывать много интерфейсов, но унаследовать несколько абстрактных классов нельзя.

Давайте попробуем на практике. Пусть кроме интерфейса **Figure** у нас также есть интерфейс **Tetragon** (четырехугольник). Методы этого интерфейса будут реализовывать классы **Quadrate** (квадрат) и **Rectangle** (прямоугольник), так как у них 4 стороны, но не класс **Disk** (круг).

Пусть интерфейс **Tetragon** описывает геттеры для всех четырех сторон четырехугольника:

```php
<?php
  interface Tetragon
  {
    public function getA();
    public function getB();
    public function getC();
    public function getD();
  }
?>
```

Пусть также у нас есть интерфейс **Figure**, который мы уже делали ранее:

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

Сделаем так, чтобы класс **Quadrate** реализовывал два интерфейса: и **Figure**, и **Tetragon**.

Для этого перечислим оба интерфейса через запятую после ключевого слова **implements**:

```php
<?php
  class Quadrate implements Figure, Tetragon
  {
    // тут будет реализация
  }
?>
```

Доработаем теперь наш класс **Quadrate**, чтобы он реализовывал интерфейс **Tetragon**. Понятно, что наш квадрат является вырожденным случаем четырехугольника, ведь у квадрата все стороны равны.

Поэтому все новые методы будут возвращать одно и тоже - ширину квадрата:

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

    public function __construct($a)
    {
```

```
    $this->a = $a;
  }

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

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

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

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

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

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

Очевидно, что в прямоугольнике уже не все стороны одинаковы, а только противоположные. В этом случае новые методы станут немного отличаться.

Ну, и в какой-нибудь трапеции вообще все 4 стороны будут разные.

Однако, не имеет значения, что за фигуру мы будем рассматривать - важно, что все эти фигуры будут иметь описанные методы (пусть некоторые фигуры и вырожденные) и работать однотипно.

Задача 42.1

Сделайте так, чтобы класс **Rectangle** также реализовывал два интерфейса: и **Figure**, и **Tetragon**.

Задача 42.2

Сделайте интерфейс **Circle** (круг) с методами **getRadius** (получить радиус) и **getDiameter** (получить диаметр).

Задача 42.3

Сделайте так, чтобы класс **Disk** также реализовывал два интерфейса: и **Figure**, и **Circle**.