Применение интерфейсов

Итак, мы уже выяснили, что интерфейсы хороший способ контролировать то, что реализованы все необходимые методы класса.

Давайте рассмотрим еще один, более практический, пример.

Пусть у нас есть класс **FiguresCollection**, который будет хранить в себе массив объектов-фигур:

```php
<?php
  class FiguresCollection
  {
    private $figures = []; // массив для фигур
  }
?>
```

Реализуем в нашем классе метод **addFigure** для добавления объектов в коллекцию:

```php
<?php
  class FiguresCollection
  {
    private $figures = [];

    // Параметром передается объект с фигурой:
    public function addFigure($figure)
    {
      $this->figures[] = $figure;
    }
  }
?>
```

Очевидно, что мы рассчитываем на то, что параметром метода **addFigure** будет передаваться объект с фигурой. Однако за этим нет никакого контроля!

Давайте используем подсказку для типов и явно укажем тип объектов как **Figure**:

```php
<?php
  class FiguresCollection
  {
    private $figures = [];

    public function addFigure(Figure $figure)
    {
      $this->figures[] = $figure;
    }
  }
?>
```

Давайте разберемся с тем, что мы сделали.

Если бы **Figure** был реально существующим классом то в параметр метода мы смогли бы передать объекты этого класса, а также и его наследников.

У нас, однако, **Figure** - это интерфейс. В таком случае подсказка обозначает то, что параметром метода могут быт переданы только объекты класса, реализующих интерфейс **Figure**.

Давайте попробуем создать объект класса **FiguresCollection** и добавить в него фигуры:

```php
<?php
  $figuresCollection = new FiguresCollection;
```

```
3
4     // Добавим парочку квадратов:
5     $figuresCollection->add(new Quadrate(2));
6     $figuresCollection->add(new Quadrate(3));
7
8     // Добавим парочку прямоугольников:
9     $figuresCollection->add(new Rectangle(2,
        3));
10    $ figuresCollection->add(new Rectangle(3,
        4));
11  ?>
```

Попытка добавить объект какого-либо другого класса приведет к ошибке:

```
1   <?php
2     $figuresCollection = new FiguresCollection;
3
4     class Test {}; // какой-то другой класс
5     $figuresCollection->add(new Test); // выдаст оши
        бку
6   ?>
```

Что на практике дает нам такой контроль: так как все фигуры, добавленные в коллекцию, реализуют интерфейс **Figure**, мы можем быть уверены, что у каждой из них будет метод **getSquare** и метод **getPerimeter**.

Возможно в дальнейшем кроме квадрата и прямоугольника появится, например, еще и треугольник. В этом случае и у треугольника также будут методы **getSquare** и **getPerimeter**.

На практике это дает нам следующее: мы можем в классе **FiguresCollection** сделать, к примеру, метод **getTotalSquare**, находящий полную площадь фигур коллекции.

При этом в методе **getTotalSquare** мы будем перебирать циклом массив фигур и у каждой фигуры вызывать метод **getSquare**.

Так как каждая фигура реализует интерфейс **Figure**, мы можем быть на 100% уверены в том, что у каждой фигуры будет этот метод **getSquare**.

Итак, вот реализация метода:

```
1   <?php
2     class FiguresCollection
3     {
4       private $figures = [];
5
6       public function addFigure(Figure $figure)
7       {
8         $this->figures[] = $figure;
9       }
10
11      // Найдем полную площадь:
12      public function getTotalSquare()
13      {
14        $sum = 0;
15
16        foreach ($this->figures as $figure) {
17          $sum += $figure->getSquare(); // используем мет
              од getSquare
18        }
19
20        return $sum;
21      }
22    }
23  ?>
```

Задача 37.1

Не подсматривая в мой код реализуйте такой же класс **FiguresCollection**.

Задача 37.2

Добавьте в класс **FiguresCollection** метод **getTotalPerimeter** для нахождения суммарного периметра всех фигур.