Date
25/08/2021

tiate the class and pass in an array of shapes, and display the output at the bottom of the page.

```
$shapes = array(
    new Circle(2),
    new Square(5),
    new Square(6)
);
$areas = new AreaCalculator($shapes);
echo $areas->output();
```

The problem with the output method is that the AreaCalculator handles the logic to output the data. Therefore, what if the user wanted to output the data as json or something else?

All of that logic would be handled by the AreaCalculator class, this is what SRP against; the AreaCalculator class should only sum the areas of provided shapes, it should not care whether the user wants json or HTML.
So, to fix this you can create an SumCalculatorOutputter class and use this to handle whatever logic you need to handle how the sum areas of all provided shapes are displayed.
The SumCalculatorOutputter class would work like this:

```
$shapes = array(
   new Circle(2),
   new Square(5),
   new Square(6)
);
```

Date
25/05/2017

```
$areas = new AreaCalculator($shapes);
$output = new SumCalculatorOutputter($areas);
echo $output→JSON();
echo $output → HAML();
echo $output → HTML();
echo $output → JADE();
```

Now, whatever logic you need to output the data to the user it's now handled by the SumCalculatorOutputter class.

2. Open/Closed principle: Objects or entities should be open for extension, but closed for modification.
This simply means that a class should be easily extendable without modifying the class itself.
Let's take a look at the AreaCalculator class, especially it's sum method.

```
public function sum()
{
   foreach($this→shapes as $shape) {
     if(is_a($shape, 'Square')) {
        $area[] = pow($shape→length, 2);
     } else if(is_a($shape, 'Circle')) {
        $area[] = pi() * pow($shape→radius, 2);
     }
   }
   return array_sum($area);
}
```

Date
25/05/2017

If we wanted the sum method to be able to sum the areas of more shapes, we would have to add more if/else blocks and that goes against the open/closed principle.
A way we can make this sum method better is to remove the logic to calculate the area of each shape out of the sum method and attach it to the shape's class.

```
class Square
{
   public $length;
   public function __construct($length)
   {
      $this->length = $length;
   }
   public function area()
   {
      return pow($this->length, 2);
   }
}
```

The same thing should be done for the circle class, an area method should be added. Now, to calculate the sum of any shape provided should be as simple as:

```
public function sum()
{
   foreach($this->shapes as $shape) {
      $area[] = $shape->area();
   }
   return array_sum($area);
}
```

Date
25/05/2017

Now we can create another shape class and pass it in when calculating the sum without breaking our code. However, now another problem arises, how do we know that the object passed into the AreaCalculator is actually a shape or if the shape has a method named area?
Coding to an interface is an integral part of S.O.L.I.D., a quick example is we create an interface, that every shape implements:

```
interface ShapeInterface
{
    public function area();
}
class Circle implements ShapeInterface
{
    public $radius;
    public function __construct($radius)
    {
        $this->radius = $radius;
    }
    public function area()
    {
        return pi() * pow($this->radius, 2);
    }
}
```

In our AreaCalculator sum method we can check if the shapes provided are actually instances of the ShapeInterface, otherwise we throw an exception:

Date
25/05/2017

```
public function sum()
{
  foreach($this->shapes as $shape){
    if(is_a($shape, 'ShapeInterface')){
      $area[] = $shape->area();
      continue;
    }
    throw new AreaCalculatorInvalidShapeException;
  }
  return array_sum($area);
}
```

3. Liskov substitution principle: Every subclass/derived class should be substitutable for their base/parent class.

Example:

```
class A
{
   public function fire() {}
}
class B extends A
{
   public function fire() {}
}
                                          B
function doSomething(A $obj)
{
   //do something with it
}
```

LSP says that if we start use of B instead of