Clase 5: Diseño de componentes

Rodrigo Arturo Saffie Kattan

Pontificia Universidad Católica de Chile

rasaffie@ing.puc.cl

16 de agosto de 2016

Contenidos

Repaso Clase Anterior

Principios del diseño detallado:

- Abstracción
- Ocultamiento
- Cohesión
- Acoplamiento

Diseño de Componentes

¿Qué es un componente?

"A component is a modular building block for computer software." [Pressman, 2009]

"... a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces." [Object Managed Group, 2003]

Diseño de Componentes

¿Qué es un componente?

Depende del punto de vista:

- Vista orientada a objetos
- Vista tradicional
- Vista orientada a procesos

Diseño de Componentes

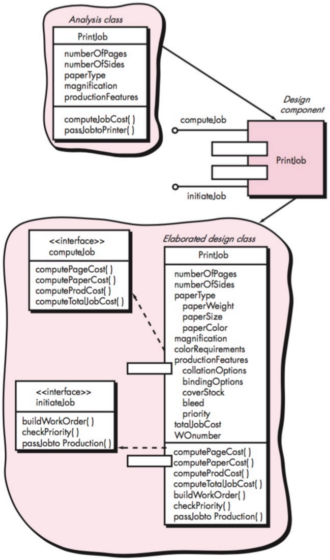
Vista orientada a objetos

Es un conjunto de clases que colaboran:

- Incluyen atributos y operaciones relevantes
- Definen interfaces para la comunicación

Diseño de Componentes

Vista orientada a objetos



Diseño de Componentes

Vista tradicional

Un *modulo* es un componente funcional del sistema:

- Tiene lógica del proceso
- Datos y estructuras para ejecutar la lógica
- Una interfaz para ser invocado

Diseño de Componentes

Vista tradicional

Puede tener uno de estos roles:

- Coordinar la invocación de otros componentes
- Resolver un problema de la lógica del sistema
- Soportar el procesamiento necesario para el problema

Diseño de Componentes

Vista orientada a procesos

Componentes que resuelven necesidades recurrentes:

- Reutilizables
- Especializados y descritos completamente
- Patrones de diseño

Diseño de Componentes

- Diseñar los componentes reduce el nivel de abstracción de la solución
- Es el paso previo a la **Construcción** de software

Principios del diseño de Componentes

Principios S.O.L.I.D.

- Propuestos por Robert C. Martin (cerca del 2000)
- Ayudan a desarrollar software fácil de mantener y extender

Principios del diseño de Componentes

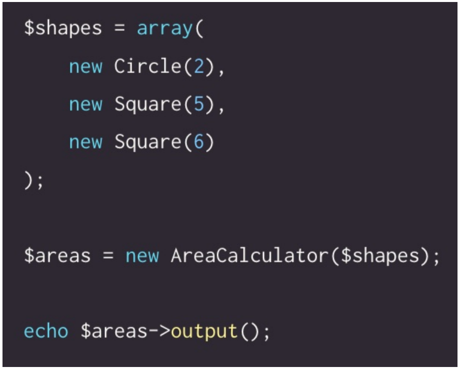
Principios S.O.L.I.D.

- S – Single-responsiblity principle
- O – Open-closed principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency Inversion principle

Principios del diseño de Componentes

Single-responsiblity principle

"A class should have one and only one reason to change, meaning that a class should have only one job."

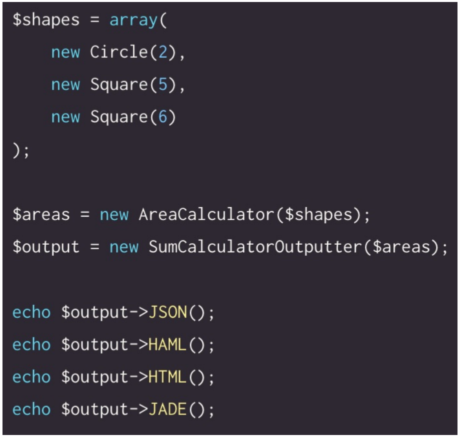


```
$shapes = array(
    new Circle(2),
    new Square(5),
    new Square(6)
);

$areas = new AreaCalculator($shapes);

echo $areas->output();
```

Principios del diseño de Componentes

Single-responsiblity principle

"A class should have one and only one reason to change, meaning that a class should have only one job."

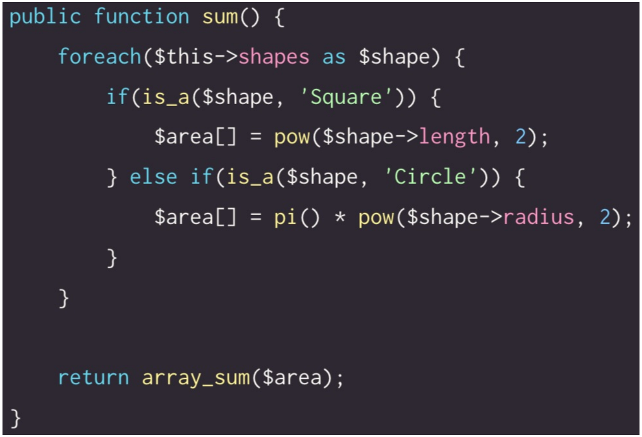


```
$shapes = array(
    new Circle(2),
    new Square(5),
    new Square(6)
);

$areas = new AreaCalculator($shapes);
$output = new SumCalculatorOutputter($areas);

echo $output->JSON();
echo $output->HAML();
echo $output->HTML();
echo $output->JADE();
```

Principios del diseño de Componentes

Open-closed principle

"Objects or entities should be open for extension, but closed for modification."

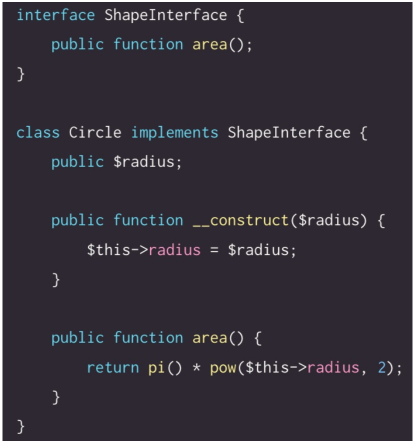


```
public function sum() {
    foreach($this->shapes as $shape) {
        if(is_a($shape, 'Square')) {
            $area[] = pow($shape->length, 2);
        } else if(is_a($shape, 'Circle')) {
            $area[] = pi() * pow($shape->radius, 2);
        }
    }

    return array_sum($area);
}
```

Principios del diseño de Componentes

Open-closed principle

"Objects or entities should be open for extension, but closed for modification."

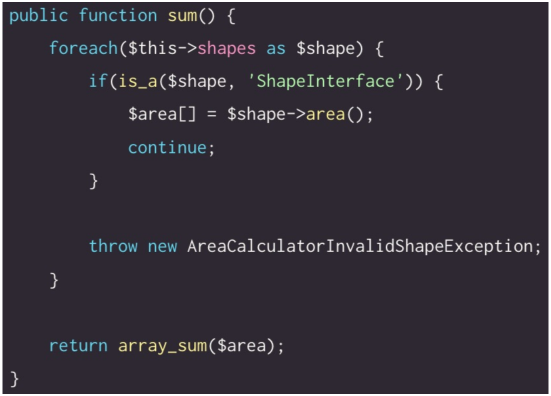


```php
interface ShapeInterface {
    public function area();
}

class Circle implements ShapeInterface {
    public $radius;

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

    public function area() {
        return pi() * pow($this->radius, 2);
    }
}
```

Principios del diseño de Componentes

Open-closed principle

"Objects or entities should be open for extension, but closed for modification."



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

        throw new AreaCalculatorInvalidShapeException;
    }

    return array_sum($area);
}
```

Principios del diseño de Componentes

Liskov substitution principle

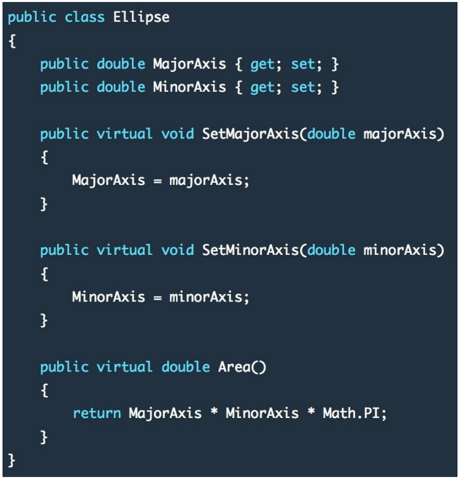
"Subclasses should be substitutable for their base classes."

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Principios del diseño de Componentes

Liskov substitution principle

"Subclasses should be substitutable for their base classes."

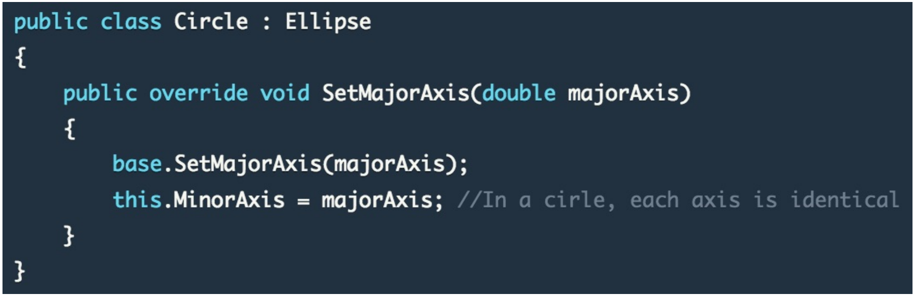


```
public class Ellipse
{
    public double MajorAxis { get; set; }
    public double MinorAxis { get; set; }

    public virtual void SetMajorAxis(double majorAxis)
    {
        MajorAxis = majorAxis;
    }

    public virtual void SetMinorAxis(double minorAxis)
    {
        MinorAxis = minorAxis;
    }

    public virtual double Area()
    {
        return MajorAxis * MinorAxis * Math.PI;
    }
}
```

Principios del diseño de Componentes

Liskov substitution principle

"Subclasses should be substitutable for their base classes."

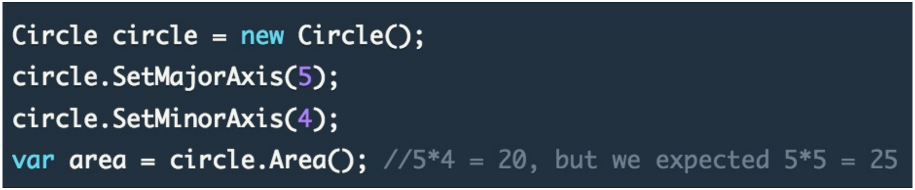


```
public class Circle : Ellipse
{
    public override void SetMajorAxis(double majorAxis)
    {
        base.SetMajorAxis(majorAxis);
        this.MinorAxis = majorAxis; //In a cirle, each axis is identical
    }
}
```

Principios del diseño de Componentes

Liskov substitution principle

"Subclasses should be substitutable for their base classes."

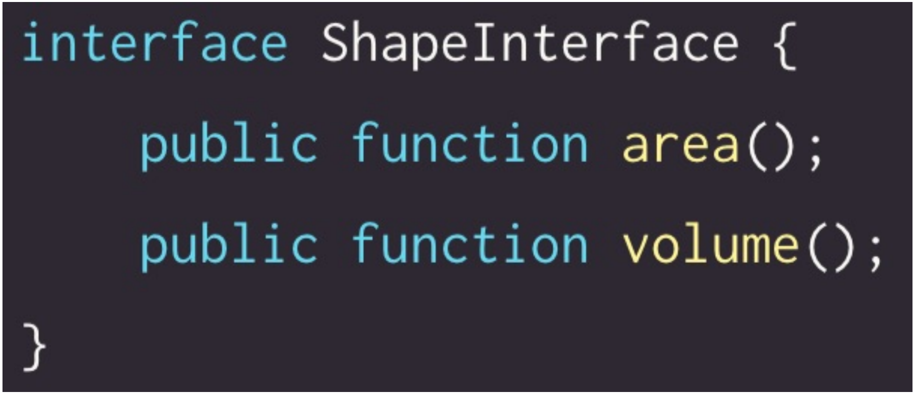


```
Circle circle = new Circle();
circle.SetMajorAxis(5);
circle.SetMinorAxis(4);
var area = circle.Area(); //5*4 = 20, but we expected 5*5 = 25
```

Principios del diseño de Componentes

Interface segregation principle

"Many client-specific interfaces are better than one general purpose interface."

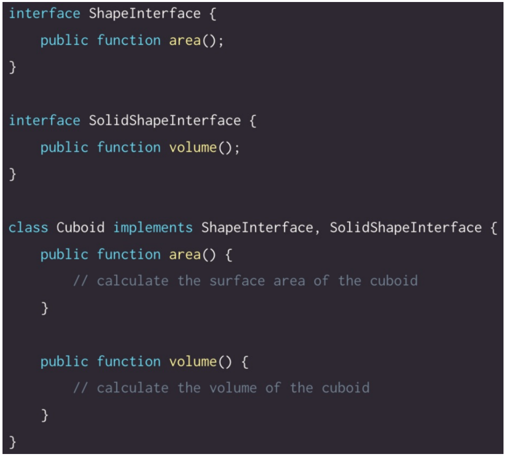


```
interface ShapeInterface {
    public function area();
    public function volume();
}
```

Principios del diseño de Componentes

Interface segregation principle

"Many client-specific interfaces are better than one general purpose interface."

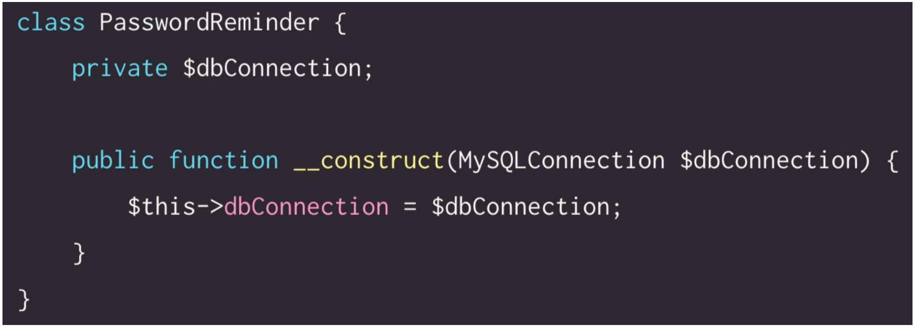


```
interface ShapeInterface {
    public function area();
}

interface SolidShapeInterface {
    public function volume();
}

class Cuboid implements ShapeInterface, SolidShapeInterface {
    public function area() {
        // calculate the surface area of the cuboid
    }

    public function volume() {
        // calculate the volume of the cuboid
    }
}
```

Principios del diseño de Componentes

Dependency Inversion Principle

"Depend on abstractions. Do not depend on concretions."

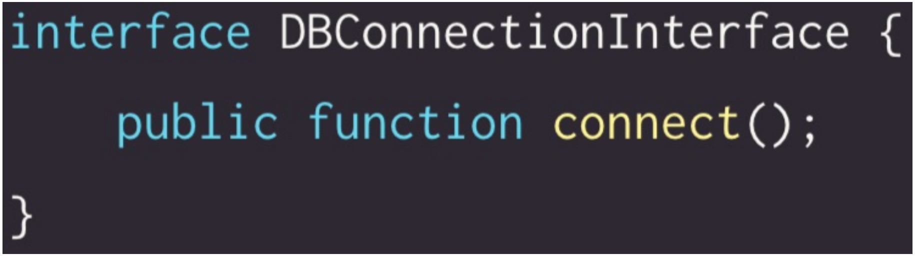


```
class PasswordReminder {
    private $dbConnection;

    public function __construct(MySQLConnection $dbConnection) {
        $this->dbConnection = $dbConnection;
    }
}
```

Principios del diseño de Componentes

Dependency Inversion Principle

"Depend on abstractions. Do not depend on concretions."

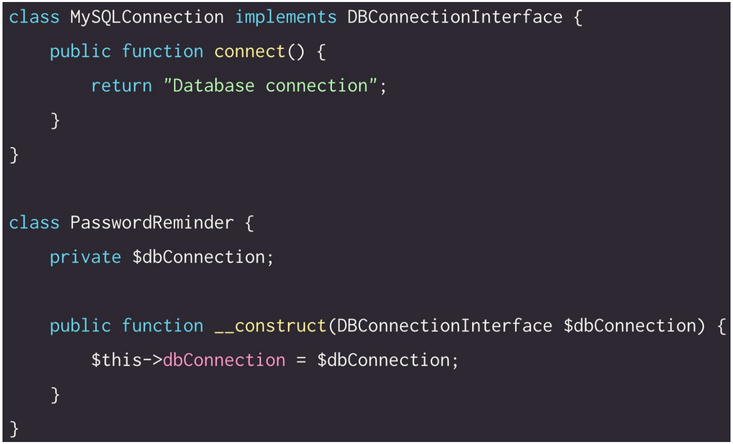


```
interface DBConnectionInterface {
    public function connect();
}
```

Principios del diseño de Componentes

Dependency Inversion Principle

"Depend on abstractions. Do not depend on concretions."



```
class MySQLConnection implements DBConnectionInterface {
    public function connect() {
        return "Database connection";
    }
}

class PasswordReminder {
    private $dbConnection;

    public function __construct(DBConnectionInterface $dbConnection) {
        $this->dbConnection = $dbConnection;
    }
}
```

Principios del diseño Componentes

Bajo Acomplamiento, Alta Cohesión

Principios del diseño de Componentes

Charla sobre S.O.L.I.D.
https://www.youtube.com/watch?v=TAVn7s-kO9o

Referencias

Pressman, R. S. (2009)
Software Engineering: A Practitioner's Approach
7th ed., *McGraw-Hill Education*

Object Management Group (2003)
http://www.omg.org/

Oloruntoba, S. (2015)
https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Jones, M. (2015)
https://www.exceptionnotfound.net/simply-solid-the-liskov-substitution-principle/

Fin