
Make Your Project SOLID

PHP Benelux 2013

Tobias Schlitt (@tobySen)

2013-01-26

About me

- ▶ Degree in computer science
- ▶ Professional PHP since 2000
- ▶ Open source enthusiast
- ▶ Passion for
 - ▶ Software Design
 - ▶ Automated Testing

Co-founder of



Helping people to create high quality web applications.

<http://qafoo.com>

- ▶ Expert consulting
- ▶ Individual training

Get a training on object oriented design for your team!

Why Object Orientation?

So, why do you want object oriented code?

- ▶ Manage complexity
- ▶ Extensible
- ▶ Maintainable

S.O.L.I.D

- ▶ 5 essential principles of object oriented design
- ▶ Introduced by Robert C. Martin (Uncle Bob)
 - ▶ not the inventor of the principles
- ▶ Have proven to lead to better code
- ▶ Scientific background (partly)

A weather loader component

- ▶ Fetch weather for a city
- ▶ Relevant data:
 - ▶ Condition
 - ▶ Temperature
 - ▶ Wind
- ▶ Be service-agnostic
 - ▶ Weather service come and go
 - ▶ Data licenses may change
- ▶ Log service failures
- ▶ Make it possible to add service fallbacks later

Single Responsibility Principle

“There should never be more than one reason for a class to change.”

The Issue

```
1 <?php
2
3 class GoogleWeatherService
4 {
5     public function getWeatherForLocation( Location $location )
6     {
7         $xml      = $this->getData( $location );
8         $weather  = $this->extractWeather( $xml );
9         return $weather;
10    }
11
12    protected function extractWeather( $xml )
13    {
14        $weather = new Weather();
15        $weather->conditions = $this->parseConditions( $xml );
16        // ...
17        $weather->windSpeed = $this->convertMilesToKilometer(
18            $this->parseWindSpeed( $xml )
19        );
20        return $weather;
21    }
22
23    /* ... */
24 }
```


The Fix

```
1 <?php
2
3 class GoogleWeatherService
4 {
5     public function __construct(
6         HttpClient $client , GoogleDataParser $parser )
7     { /* ... */ }
8
9     public function getWeatherForLocation( Location $location )
10    {
11        $xml = $this->client->get( sprintf(
12            'http://.../? city=%s',
13            $location->city
14        ) );
15        return $this->parser->parseWeather( $xml );
16    }
17 }
```

Single Responsibility Principle

- ▶ One responsibility per class
- ▶ Separation of concerns
- ▶ Responsibilities hard to detect

Open/Close Principle

“Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”

The Wrong Way

```
3 class WeatherLoader
4 {
5     public function __construct( $service )
6         { /* ... */ }
7
8     public function getWeatherForLocation( Struct\Location $location )
9     {
10         // ...
11         switch( get_class( $this->service ) )
12             {
13             case 'GoogleWeatherService':
14                 return $this->service->getWeather( $location );
15
16             case 'WetterComWeatherService':
17                 return $this->service->retrieveWeather(
18                     $location->city, $location->country
19                 );
20             // ...
21         }
22     }
23 }
```

The Right Way

```
3 class WeatherLoader
4 {
5     public function __construct( WeatherService $service )
6     { /* ... */ }
7
8     public function getWeatherForLocation( Struct\Location $location )
9     {
10         // ...
11         return $this->service->getWeatherForLocation( $location );
12     }
13 }
```

Open/Close Principle

- ▶ Changes introduce errors
 - ▶ Especially cascading changes
- ▶ Ideally: Write once, change never!
- ▶ Extend software only by new code
 - ▶ New interface implementations
 - ▶ Inheritance
 - ▶ Aggregation

Liskov Substitution Principle

“Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.”

A Simple Class

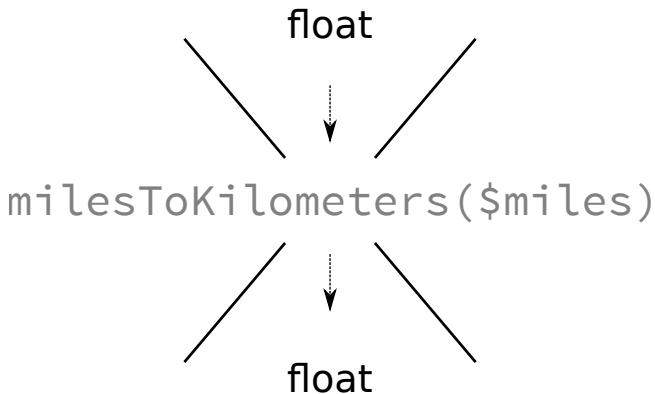
```
1 <?php
2
3 class DistanceConverter
4 {
5     const FACTOR = 0.6214;
6
7     public function milesToKilometers( $miles )
8     {
9         return $miles / self::FACTOR;
10    }
11 }
```


Getting into Trouble

```
1 <?php
2
3 class FormattingDistanceConverter extends DistanceConverter
4 {
5     public function milesToKilometers( $miles )
6     {
7         if ( $miles < 0 )
8         {
9             throw new InvalidArgumentException();
10        }
11        return sprintf(
12            '%01.2f km', parent::milesToKilometers( $miles )
13        );
14    }
15 }
```

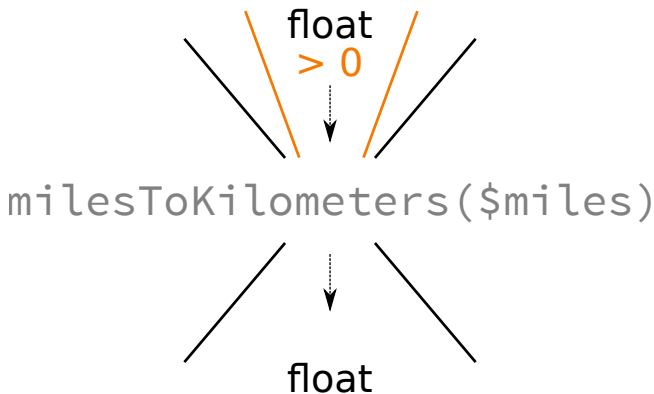
Liskov Substitution Principle

- ▶ Be less strict on input
- ▶ Be more strict on output



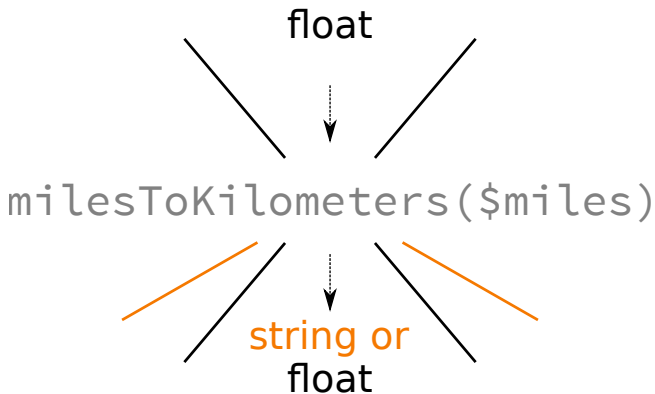
Liskov Substitution Principle

- ▶ Be less strict on input
- ▶ Be more strict on output



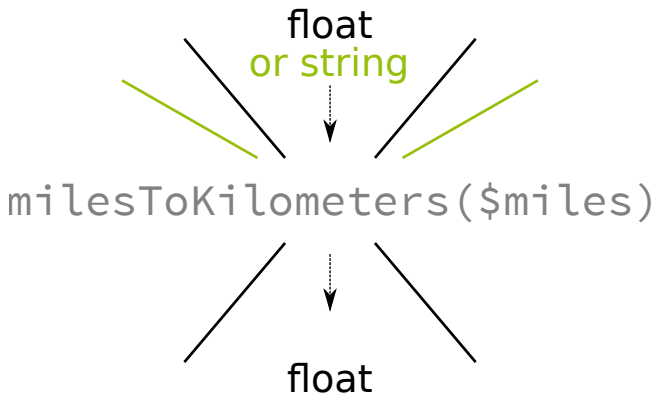
Liskov Substitution Principle

- ▶ Be less strict on input
- ▶ Be more strict on output



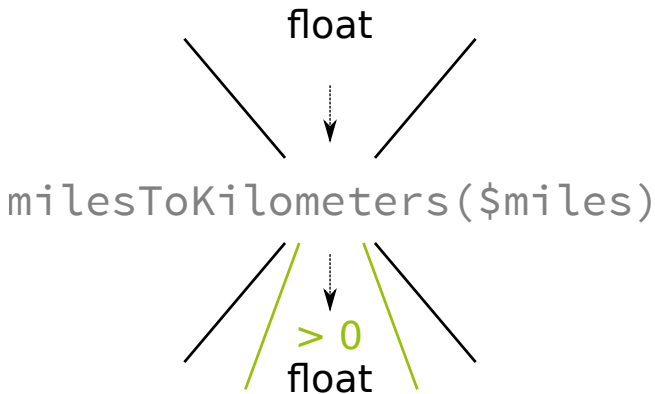
Liskov Substitution Principle

- ▶ Be less strict on input
- ▶ Be more strict on output



Liskov Substitution Principle

- ▶ Be less strict on input
- ▶ Be more strict on output



Liskov Substitution Principle

- ▶ Do not change contracts by inheritance
- ▶ Methods must work as expected in derived classes
- ▶ Users must not distinguish between super- and subclass
- ▶ Subtype polymorphism

Dependency Inversion Principle

“A. High-level modules should not depend on low level modules. Both should depend on abstractions.”

“B. Abstractions should not depend upon details. Details should depend upon abstractions.”

The Issue

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function __construct(
6         GoogleWeatherService $weatherService , FileLogger $logger )
7     {
8         $this->weatherService = $weatherService ;
9         $this->logger          = $logger ;
10    }
11    public function getWeatherForLocation( Location $location )
12    {
13        // ...
14        $this->logger->log( 'Some_log_message.' );
15        // ...
16        $this->logger->writeToFile ();
17    }
18 }
```

Doing it Right

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function __construct(
6         WeatherService $weatherService , Logger $logger )
7     {
8         $this->weatherService = $weatherService ;
9         $this->logger          = $logger ;
10    }
11    public function getWeatherForLocation( Location $location )
12    {
13        // ...
14        $this->logger->log( 'Some_log_message.' );
15        // ...
16    }
17 }
```

Dependency Inversion Principle

- ▶ Use abstraction to encapsulate low level modules
- ▶ Abstractions are the APIs
- ▶ Abstractions hide implementation details
- ▶ Depend on interfaces, not realizations
- ▶ Define interfaces from a usage point of view
- ▶ Finding abstractions is not easy
- ▶ Dependency Injection, anyone?

Interface Segregation Principle

“Clients should not be forced to depend upon interfaces that they do not use.”

Suboptimal

```
3 class Loader
4 {
5     public function __construct( WeatherService $weatherService, Logger $logger )
6     { /* ... */ }
7
8     public function getWeatherForLocation( Location $location )
9     { /* ... */ }
10 }

3 abstract class WeatherService
4 {
5     abstract public function getWeatherForLocation( Location $location );
6
7     abstract public function getForecastForLocation( Location $location, $weekDay );
8 }
```

The Fix

```
1 interface LocationWeatherProvider
2 {
3     function getWeatherForLocation( Location $location );
4 }

1 abstract class WeatherService implements LocationWeatherProvider
2 {
3     abstract public function getWeatherForLocation( Location $location );
4
5     abstract public function getForecastForLocation( Location $location , $weekDay );
6 }

1 class Loader
2 {
3     public function __construct( LocationWeatherProvider $provider , Logger $logger )
4     { /* ... */ }
5
6     public function getWeatherForLocation( Location $location )
7     { /* ... */ }
8 }
```

Interface Segregation Principle

- ▶ Avoid not needed dependencies
- ▶ Design interfaces from a usage point of view
- ▶ Do not let unnecessary functionality float in

Motivationals

Find motivational posters "SOLID in pictures" here:

<http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

Thanks for Listening

Rate this talk: <https://joind.in/7797>

Stay in touch

- ▶ Tobias Schlitt
- ▶ toby@qafoo.com
- ▶ [@tobySen](#) / [@qafoo](#)

Rent a web quality expert:
<http://qafoo.com>