ZooRoyal IT

Mockery

Sebastian Knott

10. April 2018

Überblick

- Was sind Doubles
- Zweck von Doubles
- Mockery
- Arten von Doubles
  - Dummy
  - Stub
  - Mock
  - Spy
- Advanced Usage
  - Mock vs. Spy
  - Partial Mock
- Code Dojo
  - Das Dojo
  - Code Kata
  - Das Tennis Kata

Double
Begriffsklärung

- Doubles sind Objekte
- Ersetzen Klassen ohne Änderungen am Quellcode
- Weniger komplex als das Original
- Ermöglichen…
  - Beobachten von Code
  - Kontrolle über Programmfluss

Double
Zweck von Doubles

Doubles spielen eine wichtige Rolle bei Unit-Tests.

- Abhängigkeiten des Testsubjekts werden durch Doubles ersetzt
- Subjekt wird vollständig isoliert (keine Nebeneffekte)
- Abhängigkeiten müssen nicht aufwendig erzeugt werden
- Interaktion des Subjekts mit den Doubles kann geprüft werden

```
2  protected function setUp()
3  {
4    $container = Mockery::mock(ContainerInterface::class);
5
6    $subject = new MyClass($container);
7  }
```

Mockery
WTF is Mockery?

> Mockery ist ein PHP-Mock-Framework für PHPUnit, PHPSpec oder einem anderen Testframework. Sein Hauptziel ist es, ein Double-Framework mit einer prägnanten API zu bieten, die in der Lage ist, alle möglichen Objektoperationen und -interaktionen unter Verwendung einer für Menschen lesbaren domänenspezifischen Sprache (DSL) klar zu definieren. Entworfen als eine Alternative zur PHPUnit phpunit-mock-objects-Bibliothek , ist Mockery leicht mit PHPUnit zu integrieren und kann zusammen mit phpunit-mock-Objekten ohne, dass die Welt endet.
>
> – Mockery.io

Dummy

Dummies sind leere Implementierungen einer Signatur einer Klasse.

Dummy Code

```
2  class Container {
3    public function __construct(DateTime $dateTime){}
4    public function foo($targetFilePath){}
5  }
```

Test Code

```
2  public function testArticle($targetFilePath){
3    $container = new Container(new DateTime());
4    $subject   = new MyClass($container)
5    $subject->call();
6  }
```

Dummy

- Aufwendig zu erzeugen
- Unflexibel
- Keine erweiterte Funktionalität
- Nur für einfachste Fälle praktikabel

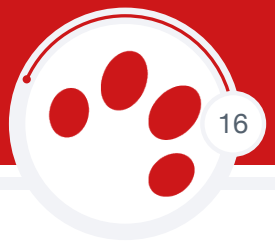
Stub

Stubs sind dynamisch erzeugte Objekte, vom Typ der Originalklasse.

```
2  protected function setUp()
3  {
4    $container = Mockery::mock(Container::class);
5    $subject   = new MyClass($container);
6  }
```

- Sehr einfach zu erzeugen
- Keine erweiterte Funktionalität
- “Quick and Dirty“-Doubles.

Mock



Mocks sind ebenfalls dynamisch erzeugte Objekte, vom Typ der Originalklasse. Mocks bieten erweiterte Funktionalität zur Simulation einer Klasse.

```
2  protected function setUp()
3  {
4    $container = Mockery::mock(Container::class);
5    $container->shouldReceive('foo')->once()
6      ->with(stringValue)->andReturn(true);
7    $subject   = new MyClass($container);
8  }
```

- Einfach zu erzeugen
- Programmflusskontrolle
- Programmflussanalyse
- Konfiguration in komplexen Fällen umständlich

Spy

Spies sind dynamisch erzeugte Objekte, vom Typ der Originalklasse. Sie agieren als Proxy zwischen Test und Originalklasse. Sie leiten Methodenaufrufe an das Original weiter und zeichnen die Interaktion zur späteren Analyse auf.

```
2  protected function setUp()
3  {
4    $container= Mockery::spy(Container::class)->makePartial();
5    $subject  = new MyClass($container);
6    $subject->call();
7    $container->shouldHaveReceived('foo');
8  }
```

- Sehr einfach zu erzeugen
- Nachvollziehen des Programmflusses
- Bricht Isolation des Subjects

Mock vs. Spy


```php
$mock = Mockery::mock('MyClass');
$spy  = Mockery::spy('MyClass')->makePartial();

$mock->shouldReceive('foo')->andReturn(42);

$mockResult = $mock->foo();
$spyResult  = $spy->foo();

$spy->shouldHaveReceived()->foo();

var_dump($mockResult); // int(42)
var_dump($spyResult); // original class result
```

Partial Mocks

Test

```
2  $mock = Mockery::mock('MyClass[foo,blub]',
3    array($arg1, $arg2));
4
5  $mock->shouldReceive('foo')->andReturn(42);
6
7  $mockResult = $mock->bar();
```

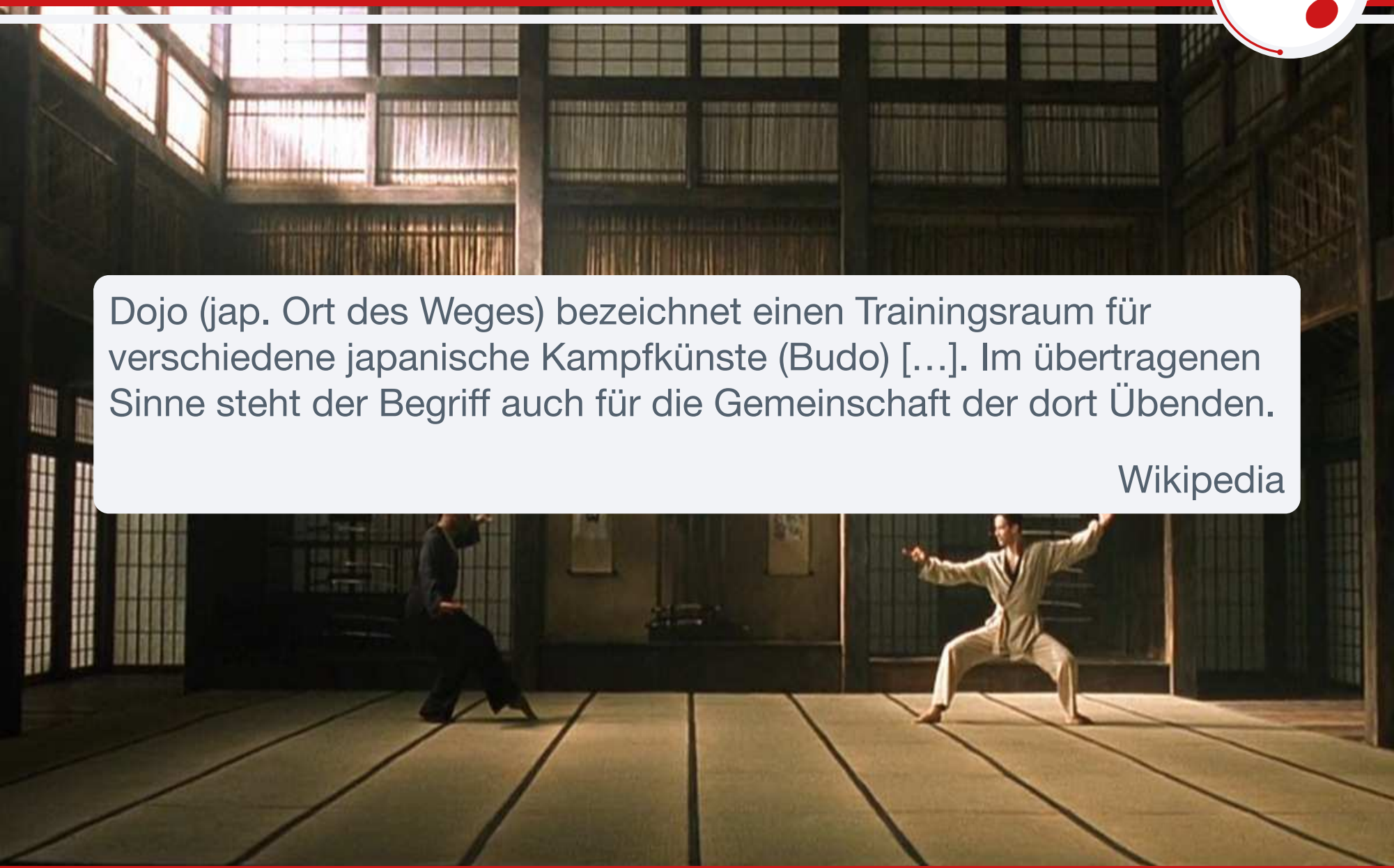
Subject

```
2  class MyClass() {
3    public function foo() {/* complex code */}
4    public function bar() {
5      foo();
6    }
7  }
```

Danke für eure Aufmerksamkeit!

Code Dojo

Was ist ein Dojo?



Dojo (jap. Ort des Weges) bezeichnet einen Trainingsraum für verschiedene japanische Kampfkünste (Budo) […]. Im übertragenen Sinne steht der Begriff auch für die Gemeinschaft der dort Übenden.

Wikipedia

Code Dojo
Was ist ein Dojo?

Die Gemeinschaft führt gemeinsam Übungen – so genannte Katas – durch um ihr Wissen in der entsprechenden Disziplin zu vervollkommnen.

Code Kata

Was ist ein Code Kata?

- Kleine, unabhängige, fokussierte, in sich geschlossene Übung
- Übt die Ausführung und Herangehensweise
- Bietet Raum für gemeinsames Lernen
- Lösung der Aufgabe erklärtes Nicht-Ziel

Fokus

- Pair Programming + TDD = TDD-Game
- Keine PHPMD Regel darf gebrochen werden
- Absichtsvolles Testen
  - Test wird zuallererst dem Pair-Partner erklärt, dann programmiert
  - Auswahl des Zwecks (use case, Erwartete Eingaben …)
  - Auswahl der Kategorie (Black-, White-box)
  - Gegebenenfalls Auswahl der Äquivalenzklasse.

Anforderung

- Eine Klasse muss die folgenden public Methoden haben
  - addPointToPlayer(PlayerName:string):null - Soll aufgerufen werden wenn ein Spieler einen Punkt erzielt. Wirft eine Exception, wenn das Spiel vorbei ist
  - getCurrentScore():string - Gibt eine Übersicht über alle gespielten Sätze, den aktuellen Punktestand und ob ein Spieler gewonnen hat aus. Es gelten die Tennisregeln mit 2 Gewinnsätzen.
- Das Konzept des Tennisspieler muss in einer einzelnen Klasse `Player` implemtiert werden. Daten den Spieler betreffend müssen in `Player` gekapselt sein.
- Test-Subjects müssen vollständig isoliert sein