PHPUNIT MOCKERY HAMCREST

ZooRoyal IT

Sebastian Knott

8/31/2018

SEBASTIAN KNOTT

Entwickler bei Zooroyal

Interner Workshoper

Software Architekt

TDD-Apologist

PHP seit ~~Ewigkeiten~~ 2006


ZooRoyal
UMFRAGE
Wie sieht's denn aus ...
... mit Unit Tests ...
... mit PHPUnit ...
... und Mockery ...
... im Alltag?

WARUM MOCKERY

ZooRoyal

UNIT TEST SIND ...

... praktisch um Bugs zu finden.

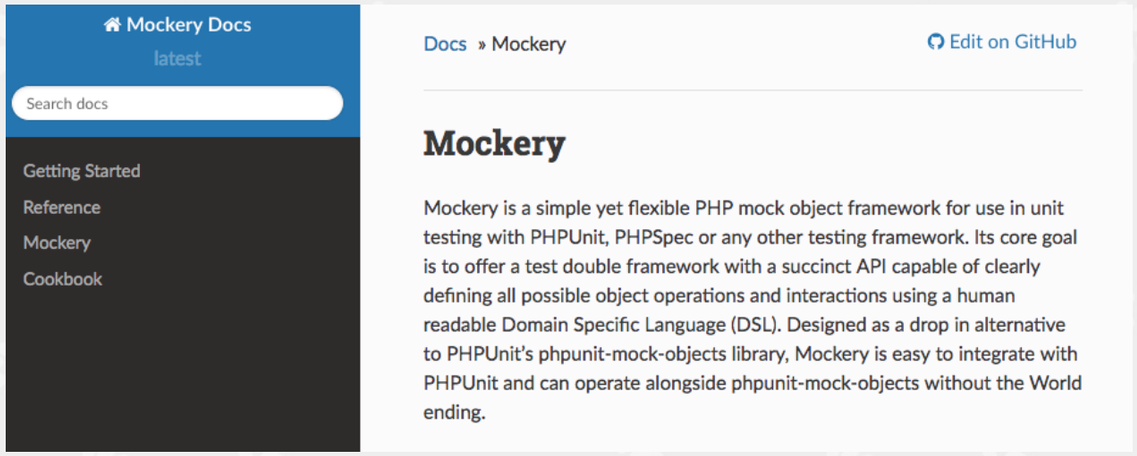
... gut fürs Geschäft.

... voll im Trend.

... zeitaufwendig.

ZooRoyal

WIE HILFT UNS MOCKERY?



— docs.mockery.io

- PHP-Mock-Framework
- Prägnanten API
- Menschen lesbaren domänen-spezifischen Sprache (DSL)
- Alternative zu PHPUnits phpunit-mock-objects-Bibliothek

BEISPIEL GEFÄLLIG?

TESTEN EINES FLUENT INTERFACES ...

```php
namespace Zooroyal\MeetUpExample;

class Example
{
    public function __construct(FileReader $fileReader)
    {
        $this->fileReader = $fileReader;
    }

    public function useFluentInterface()
    {
        return $this->fileReader->setX()
            ->setX()
            ->setX()
            ->setY()
            ->readLine();
    }
}
```

ZooRoyal

… MIT PHPUNIT-MOCKS

```php
    public function useFluentInterface()
    {
        $expectation = 'blub';

        $mockedFileReader = $this->createMock(FileReader::class);
        $mockedFileReader->expects($this->exactly(3))
            ->method('setX')->willReturnSelf();
        $mockedFileReader->expects($this->once())
            ->method('setY')->willReturnSelf();
        $mockedFileReader->expects($this->once())
            ->method('readLine')->willReturn($expectation);

        $subject = new Example($mockedFileReader);
        $result = $subject->useFluentInterface();

        self::assertSame($expectation, $result);
    }
```

… MIT MOCKERY-MOCKS

```
    public function useFluentInterface()
    {
        $expectation = 'blub';

        $mockedFileReader = Mockery::mock(FileReader::class);
        $mockedFileReader
            ->shouldReceive('setX->setX->setX->setY->readLine')
            ->andReturn($expectation);

        $subject = new Example($mockedFileReader);
        $result = $subject->useFluentInterface();

        self::assertSame($expectation, $result);
    }
```

WAS KANN MOCKERY?

DOUBLES

- Doubles sind Objekte
- Ersetzen Klassen
- Weniger komplex als Original
- Interaktion beobachten
- Kontrolle über Programmfluss

DOUBLES IN TESTS

Doubles spielen eine wichtige Rolle bei Unit-Tests.

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

```
$container = Mockery::mock(ContainerInterface::class);

$subject = new MyClass($container);
```

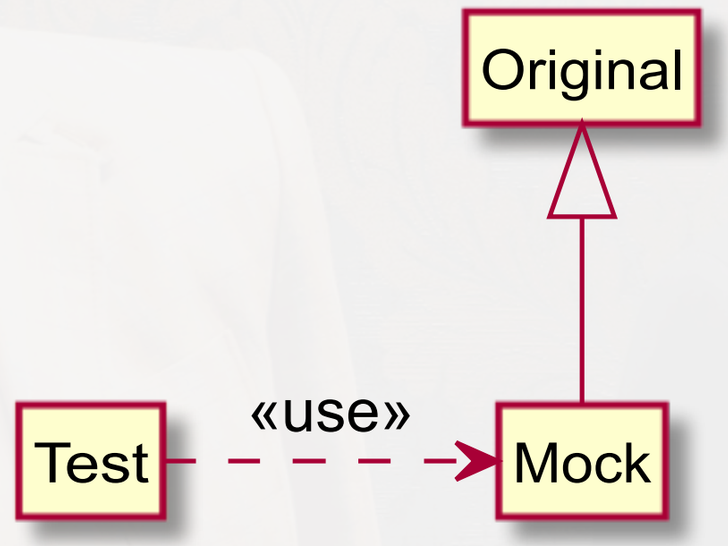
Es gibt zwei relevante Arten von Doubles in Mockery

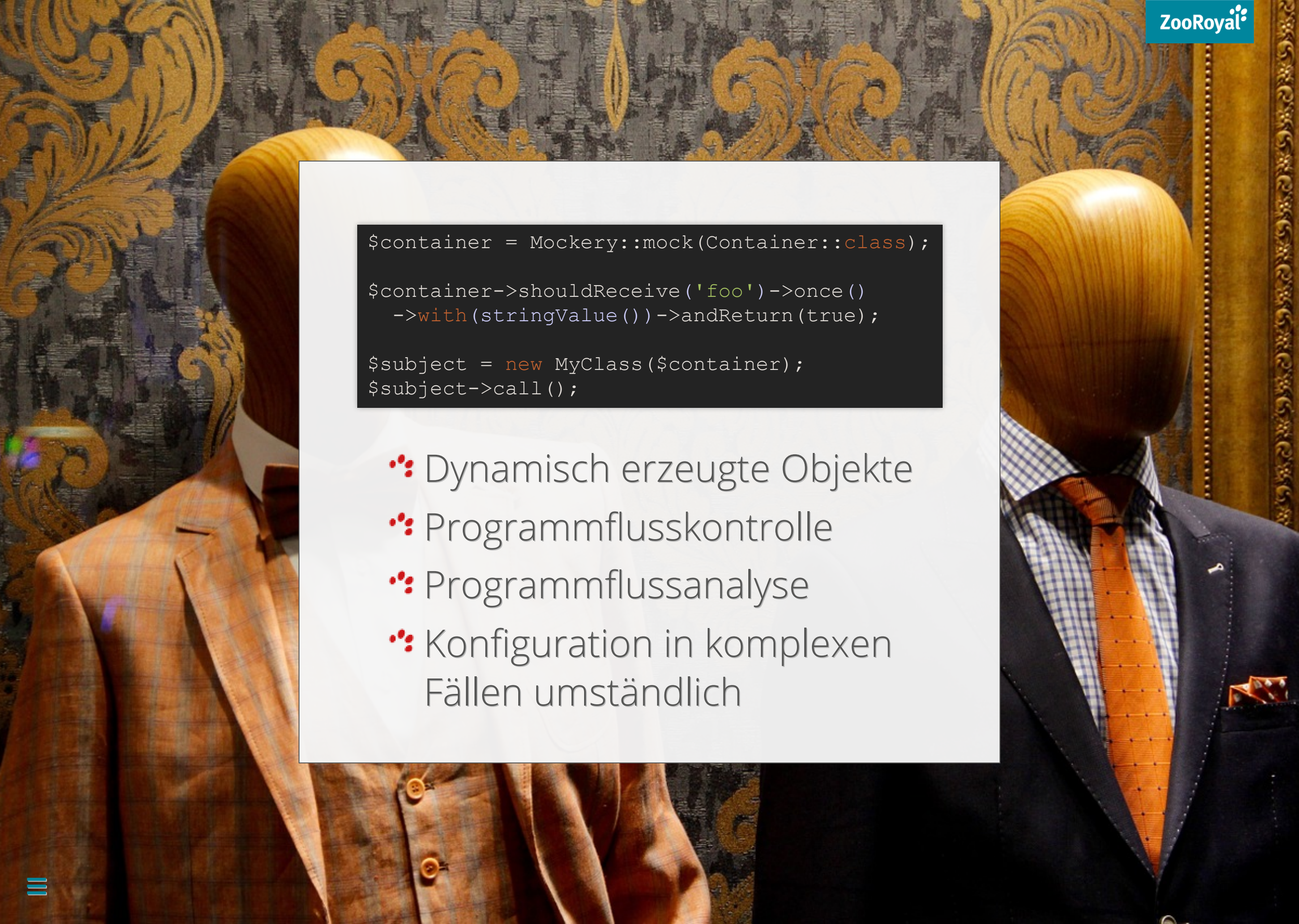
MOCKS

SPIES

MOCKS

Mocks bieten erweiterte Funktionalität zur Simulation einer Klasse.



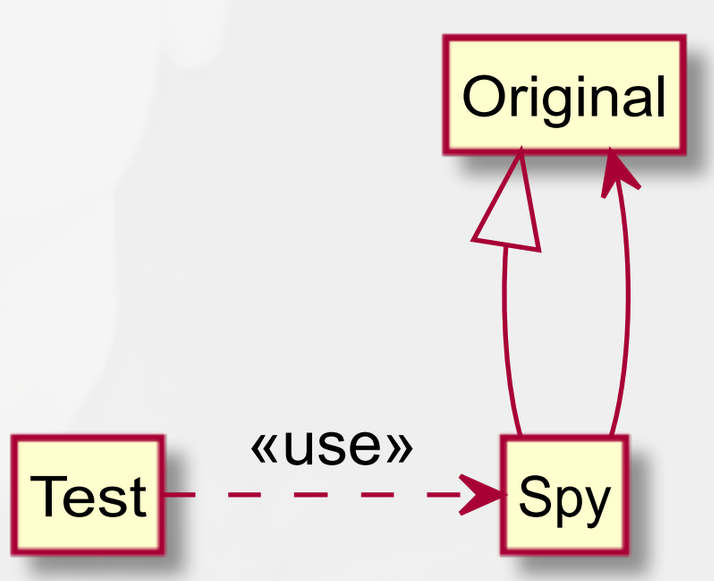


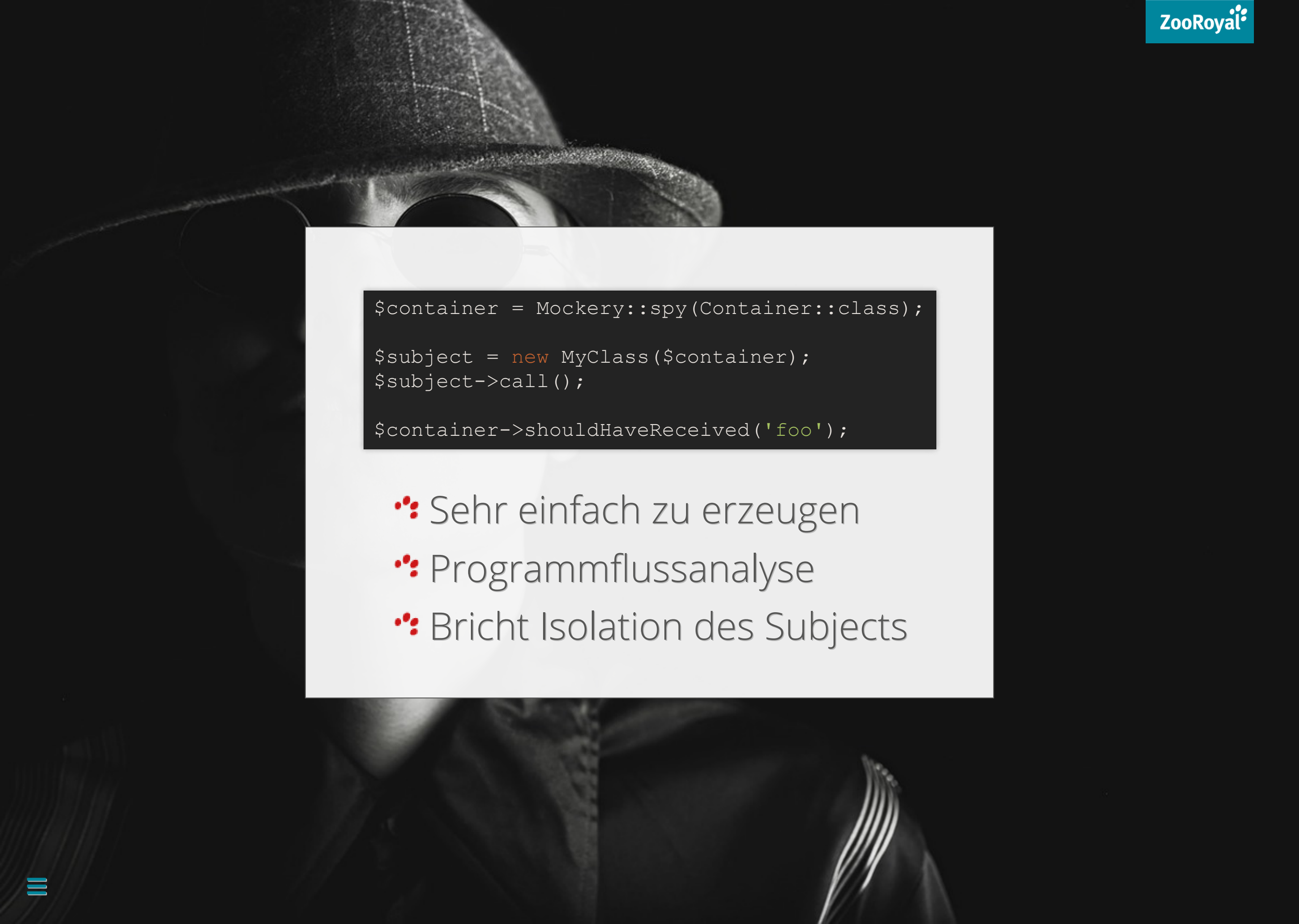
```
$container = Mockery::mock(Container::class);

$container->shouldReceive('foo')->once()
  ->with(stringValue())->andReturn(true);

$subject = new MyClass($container);
$subject->call();
```

- Dynamisch erzeugte Objekte
- Programmflusskontrolle
- Programmflussanalyse
- Konfiguration in komplexen Fällen umständlich

SPIES

Spies leiten Methodenaufrufe an das Original weiter und zeichnen die Interaktion zur späteren Analyse auf.





```
$container = Mockery::spy(Container::class);

$subject = new MyClass($container);
$subject->call();

$container->shouldHaveReceived('foo');
```

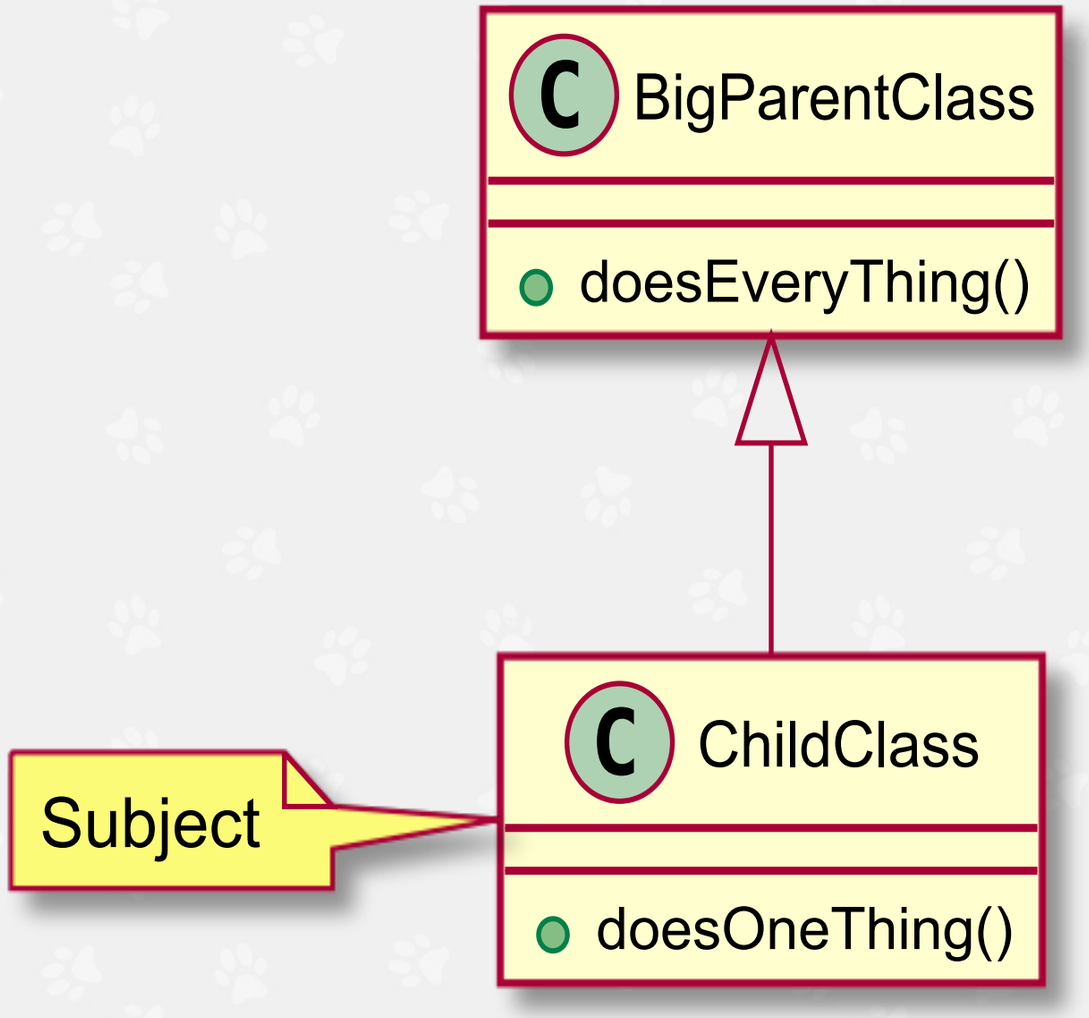
- Sehr einfach zu erzeugen
- Programmflussanalyse
- Bricht Isolation des Subjects

ZooRoyal

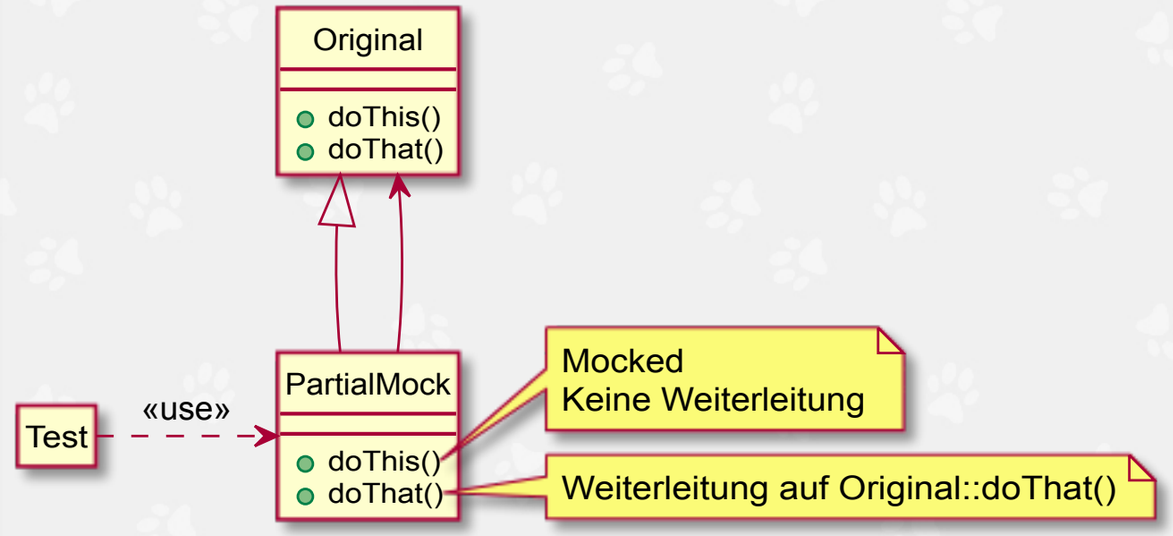
BIG PARENT CLASS

```php
class BigParentClass
{
    public function doesEverything()
    {
        // sets up database connections
        // writes to log files
    }
}
```

```php
class ChildClass extends BigParentClass
{
    public function doesOneThing()
    {
        // but calls on BigParentClass methods
        $result = $this->doesEverything();
        // does something with $result
        return $result;
    }
}
```



PARTIAL MOCKS



- Bestimmte Methoden werden gemockt
- Andere werden an das Original geleitet

ZooRoyal

TEST IN MOCKERY

```
$childClass = Mockery::mock('ChildClass')->makePartial();

$childClass->shouldReceive('doesEverything')
    ->andReturn('some result from parent');

$childClass->doesOneThing(); // string("some result from parent");
```

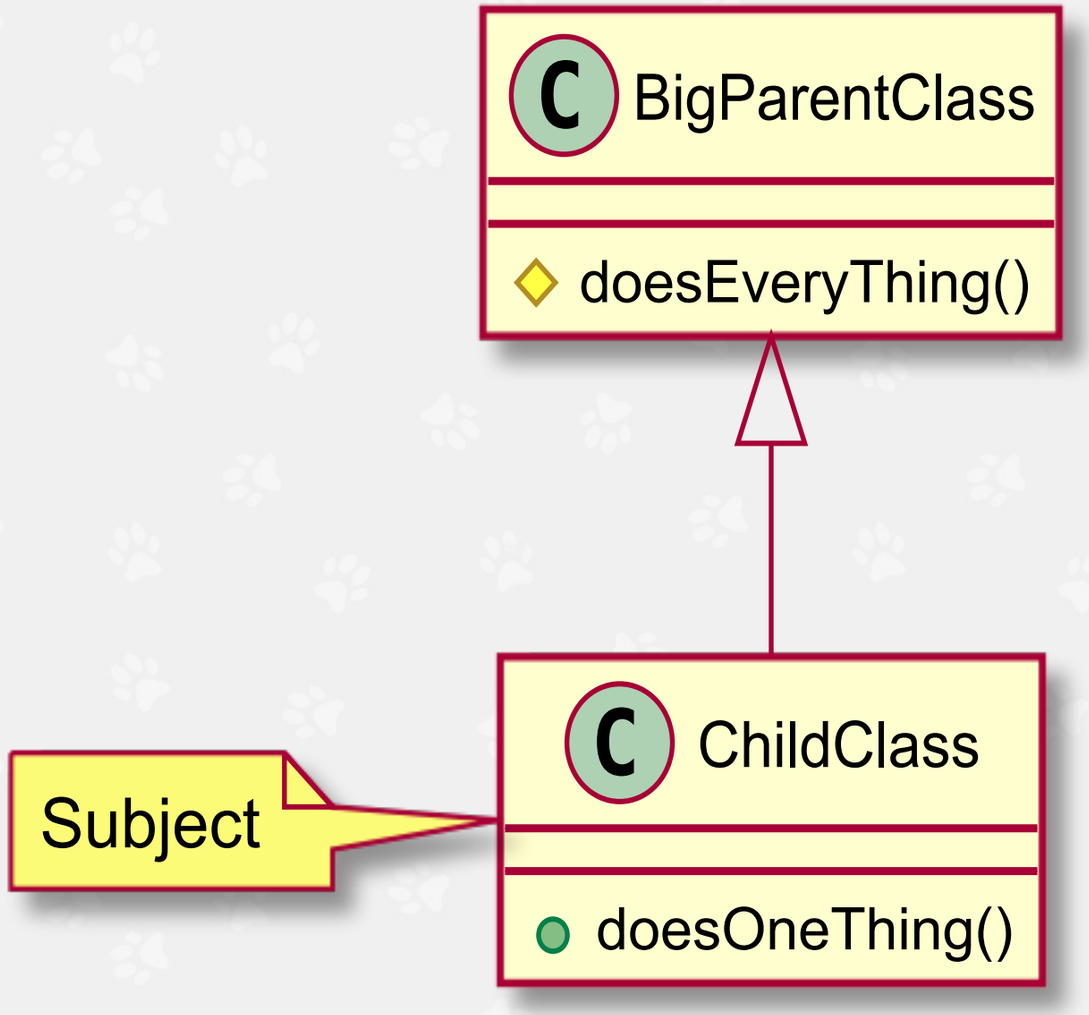
ZooRoyal

BIG PARENT CLASS

Protected Edition

```php
class BigParentClass
{
    protected function doesEverything()
    {
        // sets up database connections
        // writes to log files
    }
}
```

```php
class ChildClass extends BigParentClass
{
    public function doesOneThing()
    {
        // but calls on BigParentClass methods
        $result = $this->doesEverything();
        // does something with $result
        return $result;
    }
}
```



MOCKING PROTECTED METHODS

Mit Mockery ist es möglich `protected function` zu mocken.

```php
class MyClass
{
  protected function foo(){}
}
```

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

$mock->shouldReceive('foo');
```

TEST IN MOCKERY

```
        $childClass = Mockery::mock('ChildClass')->makePartial()
            ->shouldAllowMockingProtectedMethods();

        $childClass->shouldReceive('doesEverything')
            ->andReturn('some result from parent');

        $childClass->doesOneThing(); // string("some result from parent");
```

THE DREADED NEW OPERATOR

```php
class Service
{
    function callExternalService($param)
    {
        $externalService = new Service\External();
        $externalService->sendSomething($param);
        return $externalService->getSomething();
    }
}
```

MOCKING HARD DEPENDENCIES

Mockery ermöglicht es das Autoloading von Klassen zu überschreiben.
Damit sind auch `new`-Operatoren kein Problem mehr.

```php
class Blub {
  public function foo()
  {
    $bar = new Bar();
    $bar->baz();
  }
}
```

```php
$mock = m::mock('overload:' . Bar::class);

$mock->shouldReceive('baz');
```

TEST

```php
<?php
namespace Zooroyal\MeetUpExample\Tests\Snippets;

use Mockery as m;
use PHPUnit\Framework\TestCase;
use Zooroyal\MeetUpExample\Snippets\Service;

class ServiceTest extends TestCase
{
    /**
     * @runTestsInSeparateProcesses
     * @preserveGlobalState disabled
     */
    public function testCallingExternalService()
    {
        $param = 'Testing';

        $externalMock = m::mock('overload:' . External::class);
        $externalMock->shouldReceive('sendSomething')->once()
            ->with($param);
        $externalMock->shouldReceive('getSomething')->once()
            ->andReturn('Tested!');

        $service = new Service();

        $result = $service->callExternalService($param);

        $this->assertSame('Tested!', $result);
    }
}
```

ZooRoyal

COMPLICATED CALLBACKS

```php
class Archivist
{
    /** @var Storage */
    private $storage;

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

    public function copyAndStore($object)
    {
        $copy = $this->copyObject($object);
        $this->storage->add($copy);
    }
    // ...
}
```

ZooRoyal

HAMCREST

Matchers that can be combined to create flexible expressions of intent.
– hamcrest.org

```
MatcherAssert::assertThat(
  ['asd', 'qwe'],
  both(hasItem('asd'))
  ->andAlso(everyItem(is(stringValue()))))
);
```

Core	Logical	Object	Number
anything	allOf	equalTo	closeTo
describedAs	anyOf	anInstanceOf	greaterThan
is	not	nullValue	
	both	sameInstance	

$subject is both anInstanceOf $type andAlso not sameInstance $object

```
is(
  both(
    anInstanceOf($type)
  )->andAlso(
    not(
      sameInstance($object)
    )
  )
);
```

ZooRoyal

TEST

```php
class ArchivistTest extends TestCase
{
    public function copyAndStoreSendCopyToStorage()
    {
        $mockedItem = new stdClass();
        $mockedStorage = Mockery::mock(Storage::class);

        $mockedStorage->shouldReceive('add')->with(
            is(
                both(anInstanceOf(Storage::class))
                ->andAlso(not(sameInstance($mockedItem)))
            )
        );

        $subject = new Archivist(Storage::class);
        $subject->copyAndStore($mockedItem);
    }
}
```