Mocking With Mockery

Ben Ramsey
Midwest PHP Conference
5 March 2016

HI, I'M BEN.

I'm a web craftsman, author, and speaker. I build a platform for professional photographers at ShootProof. I enjoy APIs, open source software, organizing user groups, good beer, and spending time with my family. Nashville, TN is my home.

- *Zend PHP Certification Study Guide*
- Nashville PHP & Atlanta PHP user groups
- array_column()
- ramsey/uuid
- league/oauth2-client

ShootProof

We're hiring.

Introduction to Mocking

JA JA JA JA JA JA JA JA JA JA JA JA JA JA

What is a mock object?

- Mock objects are a form of **test double**
- Test doubles are “any kind of pretend object used in place of a real object for testing purposes” (Martin Fowler)
- Mocks differ from other test doubles (like stubs) in that they are programmed with expectations about the calls they should receive
- Mocks are used in unit tests to replace behaviors of objects, services, etc. that are external to the current unit being tested but need to be called by it

Mockery vs. PHPUnit

- Mockery provides a better user experience for working with mock objects, through an easy-to-use API
- Mockery provides abilities to mock things that PHPUnit can’t, like static methods and hard dependencies
- Mockery may be used together with PHPUnit or with any other testing framework

Getting Started With Mockery

Installing Mockery

```
composer require mockery/mockery

composer require phpunit/phpunit
```

GETTING STARTED

```php
namespace Ramsey\Talks;

class Temperature
{
    public function __construct($service)
    {
        $this->_service = $service;
    }

    public function average()
    {
        $total = 0;
        for ($i = 0; $i < 3; $i++) {
            $total += $this->_service->readTemp();
        }
        return $total / 3;
    }
}
```

GETTING STARTED

```php
namespace Ramsey\Talks;

class Service
{
    public function readTemp()
    {
        // Communicate with an external service and return
        // the current temperature.
    }
}
```

GETTING STARTED

```php
$service = new \Ramsey\Talks\Service($params);
$temperature = new \Ramsey\Talks\Temperature($service);

echo $temperature->average();
```

GETTING STARTED

```php
namespace Ramsey\Talks\Test;

class TemperatureTest extends \PHPUnit_Framework_TestCase
{
    public function tearDown()
    {
        \Mockery::close();
    }

    public function testGetsAverageTemperature()
    {
        $service = \Mockery::mock('servicemock');
        $service->shouldReceive('readTemp')
            ->times(3)
            ->andReturn(10, 12, 14);

        $temperature = new \Ramsey\Talks\Temperature($service);

        $this->assertEquals(12, $temperature->average());
    }
}
```

Review

- A mock replaces an object that is expected to make certain calls
- `\Mockery::mock('servicemock')` creates a \Mockery\Mock object and is the loosest form of mock object
- Be sure to provide a `tearDown()` method in your tests that calls `\Mockery::close()`, to avoid problems

Mock Object Basics

MOCK OBJECT BASICS

```php
$mock = \Mockery::mock(['foo' => 1, 'bar' => 2]);

$this->assertEquals(1, $mock->foo());
$this->assertEquals(2, $mock->bar());
```

MOCK OBJECT BASICS

```php
namespace Ramsey\Talks;

class Temperature
{
    public function __construct(Service $service)
    {
        $this->_service = $service;
    }

    public function average()
    {
        $total = 0;
        for ($i = 0; $i < 3; $i++) {
            $total += $this->_service->readTemp();
        }
        return $total / 3;
    }
}
```

MOCK OBJECT BASICS

```php
$service = \Mockery::mock('Ramsey\\Talks\\Service');

$service = \Mockery::mock('Ramsey\\Talks\\AbstractService');

$service = \Mockery::mock('Ramsey\\Talks\\ServiceInterface');

$service = \Mockery::mock(
    'Ramsey\\Talks\\ServiceInterface, Countable, RecursiveIterator'
);
```

MOCK OBJECT BASICS

```php
$mock = \Mockery::mock('classname', [
    'methodOne' => 'some return value',
    'methodTwo' => 'another return value',
    'methodThree' => 'yet another return value',
]);

$this->assertEquals('some return value', $mock->methodOne());
$this->assertEquals('another return value', $mock->methodTwo());
$this->assertEquals('yet another return value', $mock->methodThree());
```

Review

- Mockery allows you to define a named or unnamed mock object, naming all its methods and return values
- Mock objects can be type-hinted using a class, abstract class, or interface
- By default, any method called that is not defined will result in a BadMethodCallException; to return `null` instead, use the `shouldIgnoreMissing()` behavior modifier

Mock Expectations

MOCK EXPECTATIONS

```php
$service = \Mockery::mock('Ramsey\\Talks\\Service');
$service->shouldReceive('readTemp')
    ->times(3)
    ->andReturn(10, 12, 14);
```

We could have defined it like this:

```php
$service = \Mockery::mock('Ramsey\\Talks\\Service', [
    'readTemp' => 10
]);
```

But then we couldn’t test the expectation that it should be called three times.

MOCK EXPECTATIONS

```php
namespace Ramsey\Talks;

class Temperature
{
    public function __construct($service)
    {
        $this->_service = $service;
    }

    public function average()
    {
        $total = 0;
        for ($i = 0; $i < 3; $i++) {
            $total += $this->_service->readTemp();
        }
        return $total / 3;
    }
}
```

MOCK EXPECTATIONS

```
$service = \Mockery::mock('Ramsey\\Talks\\Service');
$service->shouldReceive('readTemp')
    ->times(3)
    ->andReturn(10, 12, 14);
```

MOCK EXPECTATIONS

```php
$mock = \Mockery::mock('Foo');

$mock->shouldReceive('methodCall')
    ->with('method', 'arg', 'values')
    ->andReturn(true);
```

MOCK EXPECTATIONS

```php
$mock->shouldReceive('methodCall')
    ->with('different', 'arg', 'values')
    ->andReturn(false);
```

MOCK EXPECTATIONS

```
$mock->shouldReceive('methodCall')
    ->withNoArgs()
    ->andReturn(123);
```

MOCK EXPECTATIONS

```php
$this->assertFalse($mock->methodCall('different', 'arg', 'values'));
$this->assertTrue($mock->methodCall('method', 'arg', 'values'));
$this->assertEquals(123, $mock->methodCall());
```

MOCK EXPECTATIONS

```
$user = \Mockery::mock('User');

$user->shouldReceive('getFriendById')
    ->andReturnUsing(function ($id) {
        // Do some special handling with the arguments here.
        // For example:
        $friendStub = file_get_contents("tests/stubs/friend{$id}.json");
        return json_decode($friendStub);
    });

$friend = $user->getFriendById(1);

$this->assertEquals('Jane Doe', $friend->name);
```

MOCK EXPECTATIONS

```php
/**
 * @expectedException RuntimeException
 * @expectedExceptionMessage An error occurred
 */
public function testServiceThrowsException()
{
    $service = \Mockery::mock('Ramsey\\Talks\\Service');
    $service->shouldReceive('readTemp')
        ->andThrow('RuntimeException', 'An error occurred');

    $temperature = new \Ramsey\Talks\Temperature($service);
    $average = $temperature->average();
}
```

Review

- Expectations on a mocked method affect its behavior depending on inputs and number of times called
- We covered `times()`, `with()`, `withNoArgs()`, `andReturn()`, `andReturnUsing()`, and `andThrow()`, but Mockery provides many more options

Partial Mocks

PARTIAL MOCKS

```php
$service = \Mockery::mock('Ramsey\\Talks\\Service[readTemp]');

$service->shouldReceive('readTemp')
    ->times(3)
    ->andReturn(10, 12, 14);

$temperature = new \Ramsey\Talks\Temperature($service);

$this->assertEquals(12, $temperature->average());
```

PARTIAL MOCKS

```
$service = \Mockery::mock('Ramsey\\Talks\\Service[readTemp]', [
    $constructorArg1,
    $constructorArg2,
]);
```

Mocking Final Classes

MOCKING FINAL CLASSES

```php
$staticUuid = 'dd39edd7-bb9c-414d-a7a0-78bd41edb4fb';

$uuid = \Mockery::mock('Ramsey\\Talks\\Uuid');
$uuid->shouldReceive('uuid4')
    ->andReturn($staticUuid);

$this->assertEquals($staticUuid, $uuid->uuid4());
```

MOCKING FINAL CLASSES

```
1) Ramsey\Talks\Test\UserTest::testUuid

Mockery\Exception: The class \Ramsey\Talks\Uuid is marked final
and its methods cannot be replaced. Classes marked final can be
passed in to \Mockery::mock() as instantiated objects to create a
partial mock, but only if the mock is not subject to type hinting
checks.
```

MOCKING FINAL CLASSES

```
$staticUuid = 'dd39edd7-bb9c-414d-a7a0-78bd41edb4fb';

$uuidInstance = new \Ramsey\Talks\Uuid();

$uuid = \Mockery::mock($uuidInstance);
$uuid->shouldReceive('uuid4')
    ->andReturn($staticUuid);

$this->assertEquals($staticUuid, $uuid->uuid4());
```

This is referred to as a “proxied partial” mock.

Mocking Public Properties

MOCKING PUBLIC PROPERTIES

```php
$mock = \Mockery::mock('Foo');
$mock->publicProperty = 123;

$this->assertEquals(123, $mock->publicProperty);
```

MOCKING PUBLIC PROPERTIES

```php
$mock = \Mockery::mock('Foo');
$mock->shouldReceive('methodCall')
    ->andSet('publicProperty', 123)
    ->andReturn(true);

$this->assertTrue($mock->methodCall());
$this->assertEquals(123, $mock->publicProperty);
```

Mocking Fluent Interfaces

MOCKING FLUENT INTERFACES

```php
namespace Ramsey\Talks;

class Bar
{
    public function getSomething(Foo $foo)
    {
        $result = $foo->bar()->baz()->qux()->quux();

        return "Now, we're {$result}";
    }
}
```

MOCKING FLUENT INTERFACES

```php
$mock = \Mockery::mock('Ramsey\\Talks\\Foo');
$mock->shouldReceive('bar->baz->qux->quux')
    ->andReturn('done!');

$bar = new \Ramsey\Talks\Bar;

$this->assertEquals("Now, we're done!", $bar->getSomething($mock));
```

Mocking Static Methods

MOCKING STATIC METHODS

```php
namespace Ramsey\Talks;

class User
{
    public $addressId;

    public function getAddress()
    {
        return Address::getById($this->addressId);
    }
}
```

MOCKING STATIC METHODS

```
/**
 * @runInSeparateProcess
 * @preserveGlobalState disabled
 */
public function testGetAddress()
{
    $address = \Mockery::mock('alias:Ramsey\\Talks\\Address');
    $address->shouldReceive('getById')
        ->andReturn(new \Ramsey\Talks\Address());

    $user = new \Ramsey\Talks\User();

    $this->assertInstanceOf(
        'Ramsey\\Talks\\Address',
        $user->getAddress()
    );
}
```

MOCKING STATIC METHODS

```
1) Ramsey\Talks\Test\UserTest::testGetAddress

Mockery\Exception\RuntimeException: Could not load mock
Ramsey\Talks\Address, class already exists
```

MOCKING STATIC METHODS

```
/**
 * @runInSeparateProcess
 * @preserveGlobalState disabled
 */
public function testGetAddress()
{
    $address = \Mockery::mock('alias:Ramsey\\Talks\\Address');
    $address->shouldReceive('getById')
        ->andReturn(new \Ramsey\Talks\Address());

    $user = new \Ramsey\Talks\User();

    $this->assertInstanceOf(
        'Ramsey\\Talks\\Address',
        $user->getAddress()
    );
}
```

Mocking Hard Dependencies

MOCKING HARD DEPENDENCIES

```php
namespace Ramsey\Talks;

class User
{
    public $addressId;

    public function getAddress()
    {
        return new Address($this->addressId);
    }
}
```

MOCKING HARD DEPENDENCIES

```php
/**
 * @runInSeparateProcess
 * @preserveGlobalState disabled
 */
public function testGetAddress()
{
    $address = \Mockery::mock('overload:Ramsey\\Talks\\Address');

    $user = new \Ramsey\Talks\User();
    $user->addressId = 123;

    $this->assertInstanceOf(
        'Ramsey\\Talks\\Address',
        $user->getAddress()
    );
}
```

Wrapping Up

WRAPPING UP

```php
$foo = \Mockery::mock('Ramsey\\Talks\\Foo');

/* ... */

if ($foo instanceof \Mockery\MockInterface) {
    /* ... */
}
```

Review

- Mock objects are used to replace real objects in tests
- Mockery lets us create *dumb* mocks, mocks inherited from classes and interfaces, partial mocks, and aliases
- We saw how to use proxied partial mocks to mock *final* classes and methods
- We mocked public properties and fluent interfaces
- We created an aliased mock to mock a static method and an overloaded mock to instantiate instance mocks with the *new* keyword

THANK YOU.
ANY QUESTIONS?

If you want to talk more, feel free to contact me.

- joind.in/talk/e85df
- benramsey.com
- @ramsey
- github.com/ramsey
- ben@benramsey.com

This presentation was created using Keynote. The text is set in Chunk Five and Helvetica Neue. The source code is set in Menlo. The iconography is provided by Font Awesome.

Ramsey, Ben. “Mocking With Mockery.” Midwest PHP Conference. Hilton Minneapolis, Minneapolis. 5 Mar. 2016. Conference presentation.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

PHOTO CREDITS

1. “Mockingbird Silhouette” by Jim Mullhaupt
2. Untitled by Eli White
3. “Northern Mockingbird” by Kelly Colgan Azar
4. “Burla” by Daniel Lobo
5. “Mockingbird” by Neal Simpson
6. “Tropical Mockingbird” by hjhipster
7. “Mockingbird” by Henry T. McLin
8. “Northern Mockingbird (Cape May Point SP)” by Brian Henderson
9. “Northern Mockingbird” by Sandy/Chuck Harris
10. “Northern Mockingbird (Mimus polyglottos)” by Nicole Beaulac
11. “mockingbird in spring” by Melinda Shelton
12. “Mockingbird” by magpie_drain
13. “Mockingbird” by Kerry Lannert
14. “Northern Mockingbird” by Kelly Colgan Azar