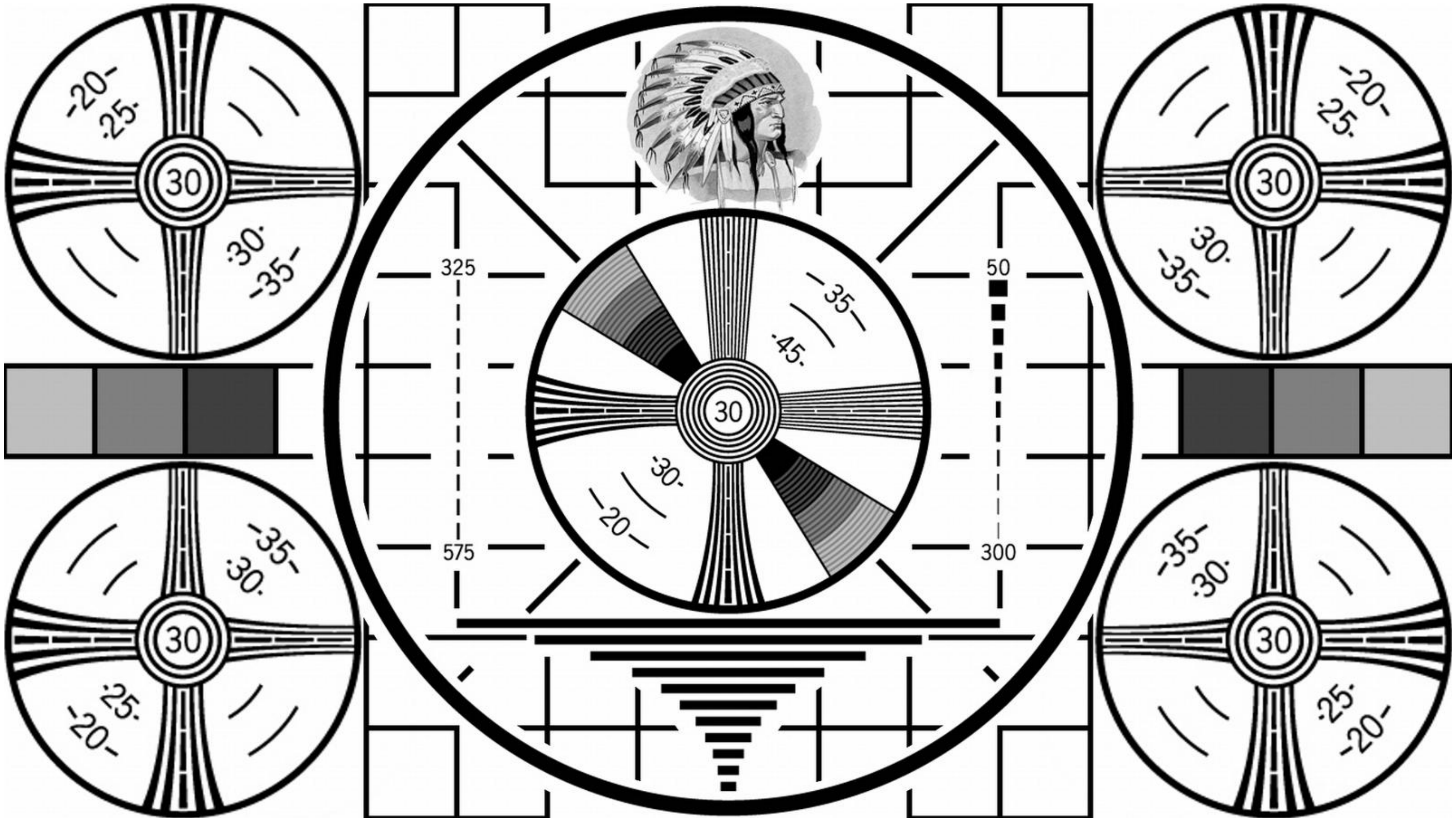
Unit Tests: Using PHPUnit to Test Your Code



With Your Host
Juan Treminio

- http://jtreminio.com
- http://github.com/jtreminio
- @juantreminio
- #phpc
- I love writing tests
- I like to work from home
- I sometimes write things for my website
- My first presentation!!!

- Moderator of /r/php

You Already Test

- Setting up temporary code
  - Write code then execute
- Hitting F5
  - Abuse F5 to see changes
- Deleting temporary code
  - Delete test code
  - Have to write it again

Why Test with PHPUnit?

- Automate testing
  - Make machine do the work
- Many times faster than you
  - Run 3,000 tests in under a minute
- Uncover bugs
  - Previously unidentified paths
  - “What happens if I do this?”
- Change in behavior
  - Test was passing, now failing. Red light!
- Teamwork
  - Bob may not know your code!
- Projects require tests
  - Can’t contribute without tests

Installing PHPUnit

- Don’t use PEAR
  - Old version
  - No autocomplete
  - Keeping multiple devs in sync
- Use Composer
  - Easy!
  - Fast!

```
composer.json
{
    "require": {
        "EHER/PHPUnit": "1.6"
    },
    "minimum-stability": "dev"
}
```

Your First (Useless) Test

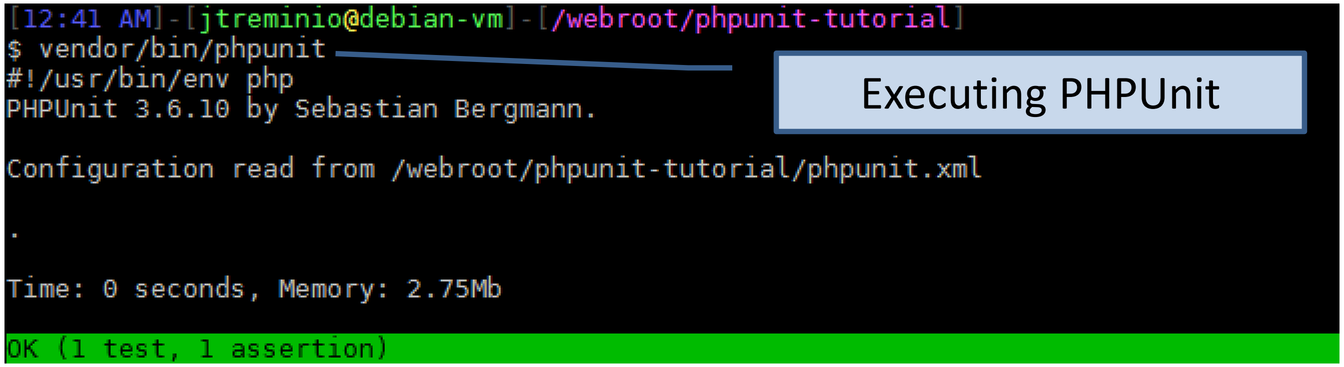
```
<?php

// tests/DumbTest.php

class DumbTest extends \PHPUnit_Framework_TestCase
{
    public function testWhatADumbTest()
    {
        $this->assertTrue(true);
    }
}
```

Tests must be called {Class}Test.php

Class name should be the same as filename.

Extends PHPUnit_Framework_TestCase

Must have the word "test" in front of method name



```
[12:41 AM]-[jtreminio@debian-vm]-[/webroot/phpunit-tutorial]
$ vendor/bin/phpunit
#!/usr/bin/env php
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /webroot/phpunit-tutorial/phpunit.xml

.

Time: 0 seconds, Memory: 2.75Mb

OK (1 test, 1 assertion)
```

Executing PHPUnit

Results of test suite run

Breaking Down a Method for Testing

```php
<?php

class Payment
{
    const API_ID = 123456;
    const TRANS_KEY = 'TRANSACTION KEY';

    public function processPayment(array $paymentDetails)
    {
        $transaction = new AuthorizeNetAIM(API_ID, TRANS_KEY);
        $transaction->amount = $paymentDetails['amount'];
        $transaction->card_num = $paymentDetails['card_num'];
        $transaction->exp_date = $paymentDetails['exp_date'];

        $response = $transaction->authorizeAndCaptu

        if ($response->approved) {
            return $this->savePayment($response->transaction_id);
        } else {
            throw new \Exception($response->error_message);
        }
    }
}
```

Expecting an array to be passed in

Using **new**

Calls method in outside class

Interacts with result

Calls method inside class

Throws Exception

Dependency Injection

- Don’t use **new**
- Pass in dependencies in method parameters
- Learn yourself some DI [1]

```
// Bad method
public function processPayment(array $paymentDetails)
{
    $transaction = new AuthorizeNetAIM(API_ID, TRANS_KEY);
    // …


// Good method
public function processPayment(
    array $paymentDetails,
    AuthorizeNetAIM $transaction
){
    // …
```

[1] http://fabien.potencier.org/article/11/what-is-dependency-injection

Updated Payment Class

```php
<?php

class Payment
{
    public function processPayment(
        array $paymentDetails,
        AuthorizeNetAIM $transaction
    ){
        $transaction->amount = $paymentDetails['amount'];
        $transaction->card_num = $paymentDetails['card_num'];
        $transaction->exp_date = $paymentDetails['exp_date'];

        $response = $transaction->authorizeAndCapture();

        if ($response->approved) {
            return $this->savePayment($response->transaction_id);
        } else {
            throw new \Exception($response->error_message);
        }
    }
}
```

Introducing Mocks and Stubs

- Mocks
  - Mimic the original method closely
  - Execute actual code
  - Give you some control
- Stubs
  - Methods are completely overwritten
  - Allow complete control

Both are used for outside dependencies we don’t want to our test to have to deal with.

How to Mock an Object

- Create separate files
  - Lots of work
  - Lots of files to keep track of
- Use getMock()
  - Too many optional parameters!
  - `public function getMock($originalClassName, $methods = array(), array $arguments = array(), $mockClassName = '', $callOriginalConstructor = TRUE, $callOriginalClone = TRUE, $callAutoload = TRUE)`
- Use getMockBuilder() !
  - Uses chained methods
  - Much easier to work with
- Mockery [1]
  - Once you master getMockBuilder() it is no longer necessary

[1] https://github.com/padraic/mockery

->getMockBuilder()

- Create a basic mock
  - Creates a mocked object of the AuthorizeNetAIM class

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMock();
```

Mocked method created at runtime

->getMockBuilder()->setMethods() 1/4

setMethods() has 4 possible outcomes

- Don’t call setMethods()
  - All methods in mocked object are stubs
  - Return **null**
  - Methods easily overridable

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMock();
```

Passes is_a() checks!

->getMockBuilder()->setMethods() 2/4

setMethods() has 4 possible outcomes

- Pass an empty array
  - Same as if not calling setMethods()
  - All methods in mocked object are stubs
  - Return `null`
  - Methods easily overridable

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->setMethods(array())
                ->getMock();
```

->getMockBuilder()->setMethods() 3/4

setMethods() has 4 possible outcomes

- Pass `null`
  - All methods in mocked object are mocks
  - Run actual code in method
  - Not overridable

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->setMethods(null)
                ->getMock();
```

->getMockBuilder()->setMethods() 4/4

setMethods() has 4 possible outcomes

- Pass an array with method names
  - Methods identified are stubs
    - Return null
    - Easily overridable
  - Methods *not* identified are mocks
    - Actual code is ran
    - Unable to override

```php
$payment = $this->getMockBuilder('Payment')
    ->setMethods(
        array('authorizeAndCapture',)
    )
    ->getMock();
```

Other getMockBuilder() helpers

- disableOriginalConstructor()
  - Returns a mock with the class __construct() overriden

```php
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->disableOriginalConstructor()
                ->getMock();
```

- setConstructorArgs()
  - Passes arguments to the __construct()

```php
$payment = $this->getMockBuilder('AuthorizeNetAIM ')
                ->setConstructorArgs(array(API_LOGIN_ID, TRANSACTION_KEY))
                ->getMock();
```

- getMockForAbstractClass()
  - Returns a mocked object created from abstract class

```php
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMockForAbstractClass();
```

Using Stubbed Methods 1/3

`->expects()`

- `$this->once()`
- `$this->any()`
- `$this->never()`
- `$this->exactly(10)`
- `$this->onConsecutiveCalls()`

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMock();

$payment->expects($this->once())
        ->method('authorizeAndCapture');
```

Using Stubbed Methods 2/3

```
->method('name')
->will($this->returnValue('value'))
```

Overriding stub method means specifying what it returns.

- Doesn’t run any code
- Expected call count
- Can return anything

```
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMock();

$payment->expects($this->once())
        ->method('authorizeAndCapture')
        ->will($this->returnValue(array('baz' => 'boo')));
```

Using Stubbed Methods 3/3

A stubbed method can return a mock object!

```php
$payment = $this->getMockBuilder('AuthorizeNetAIM')
                ->getMock();

$invoice = $this->getMockBuilder('Invoice')
                ->getMock();

$payment->expects($this->once())
        ->method('getInvoice')
        ->will($this->returnValue($invoice));
```

Assertions

- Define what you expect to happen
- Assertions check statement is true
- 36 assertions as of PHPUnit 3.6

```php
$foo = true;
$this->assertTrue($foo);

$foo = false;
$this->assertFalse($foo);

$foo = 'bar';
$this->assertEquals(
    'bar',
    $foo
);

$arr = array('baz' => 'boo');
$this->assertArrayHasKey(
    'baz',
    $arr
);
```

Run a Complete Test 1/2

Payment.php

```php
<?php

namespace phpunitTests;

class Payment
{
    const API_ID = 123456;
    const TRANS_KEY = 'TRANSACTION KEY';

    public function processPayment(
        array $paymentDetails,
        \phpunitTests\AuthorizeNetAIM $transaction
    ) {
        $transaction->amount = $paymentDetails['amount'];
        $transaction->card_num = $paymentDetails['card_num'];
        $transaction->exp_date = $paymentDetails['exp_date'];

        $response = $transaction->authorizeAndCapture();

        if ($response->approved) {
            return $this->savePayment($response->transaction_id);
        } else {
            throw new \Exception($response->error_message);
        }
    }

    protected function savePayment()
    {
        return true;
    }
}
```

Mock AuthorizeNetAIM object

Mock authorize object (stdClass)

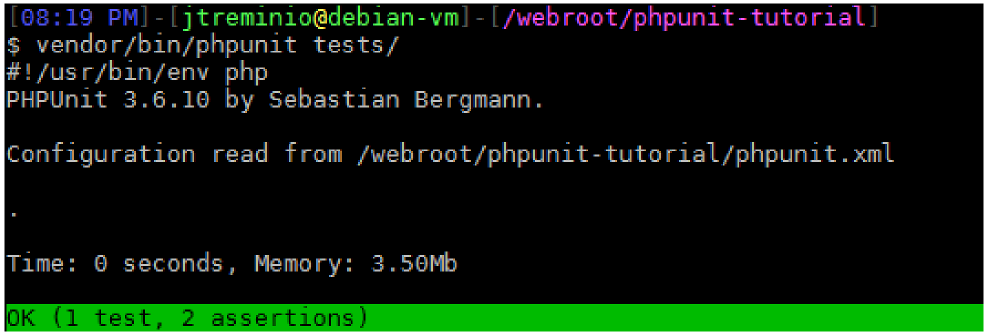
PaymentTest.php

```php
<?php

class PaymentTest extends \PHPUnit_Framework_TestCase
{
    public function testProcessPaymentReturnTrueOnApprovedResponse()
    {
        $authorizeNetAIM = $this
            ->getMockBuilder('\phpunitTests\AuthorizeNetAIM')
            ->getMock();

        $authorizeNetResponse = new \stdClass();
        $authorizeNetResponse->approved = true;
        $authorizeNetResponse->transaction_id = 12345;

        $authorizeNetAIM->expects($this->once())
            ->method('authorizeAndCapture')
            ->will($this->returnValue($authorizeNetResponse));

        $arrayDetails = array(
            'amount'   => 123,
            'card_num' => '1234567812345678',
            'exp_date' => '04/07',
        );

        $payment = new \phpunitTests\Payment();

        $this->assertTrue(
            $payment->processPayment(
                $arrayDetails,
                $authorizeNetAIM
            )
        );
    }
}
```

Return object

Instantiate our class to be tested

Our assertion



```
[08:19 PM]-[jtreminio@debian-vm]-[/webroot/phpunit-tutorial]
$ vendor/bin/phpunit tests/
#!/usr/bin/env php
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /webroot/phpunit-tutorial/phpunit.xml

.

Time: 0 seconds, Memory: 3.50Mb

OK (1 test, 2 assertions)
```

Run a Complete Test 2/2

Payment.php

```php
<?php

namespace phpunitTests;

class Payment
{
    const API_ID = 123456;
    const TRANS_KEY = 'TRANSACTION KEY';

    public function processPayment(
        array $paymentDetails,
        \phpunitTests\AuthorizeNetAIM $transaction
    ) {
        $transaction->amount = $paymentDetails['amount'];
        $transaction->card_num = $paymentDetails['card_num'];
        $transaction->exp_date = $paymentDetails['exp_date'];

        $response = $transaction->authorizeAndCapture();

        if ($response->approved) {
            return $this->savePayment($response->transaction_id);
        } else {
            throw new \phpunitTests\PaymentException(
                $response->error_message
            );
        }
    }

    protected function savePayment()
    {
        return true;
    }
}
```

Set expected Exception Cannot be \Exception()!

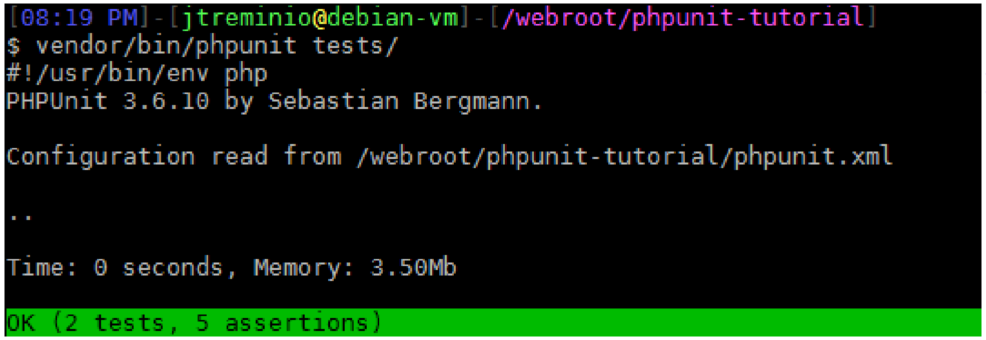
Exception thrown

PaymentTest.php

```php
public function testProcessPaymentThrowsExceptionOnUnapproved()
{
    $exceptionMessage = 'Grats on failing lol';

    $this->setExpectedException(
        '\phpunitTests\PaymentException',
        $expectedExceptionMessage
    );

    $authorizeNetAIM = $this
        ->getMockBuilder('\phpunitTests\AuthorizeNetAIM')
        ->disableOriginalConstructor()
        ->setConstructorArgs(
            array(
                \phpunitTests\Payment::API_ID,
                \phpunitTests\Payment::TRANS_KEY
            )
        )
        ->setMethods(array('authorizeAndCapture'))
        ->getMock();

    $authorizeNetResponse = new \stdClass();
    $authorizeNetResponse->approved = false;
    $authorizeNetResponse->error_message = $exceptionMessage;

    $authorizeNetAIM->expects($this->once())
        ->method('authorizeAndCapture')
        ->will($this->returnValue($authorizeNetResponse));

    $arrayDetails = array(
        'amount'   => 123,
        'card_num' => '1234567812345678',
        'exp_date' => '04/07',
    );

    $payment = new \phpunitTests\Payment();

    $payment->processPayment($arrayDetails, $authorizeNetAIM);
}
```

Force else{} to run in code

No assertion. Was already defined.



```
[08:19 PM]-[jtreminio@debian-vm]-[/webroot/phpunit-tutorial]
$ vendor/bin/phpunit tests/
#!/usr/bin/env php
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /webroot/phpunit-tutorial/phpunit.xml

..

Time: 0 seconds, Memory: 3.50Mb

OK (2 tests, 5 assertions)
```

Mocking Object Being Tested

```php
public function testProcessPaymentThrowsExceptionOnUnapproved()
{
    $exceptionMessage = 'Grats on failing lol';

    $this->setExpectedException(
        '\phpunitTests\PaymentException',
        $expectedExceptionMessage
    );

    $authorizeNetAIM = $this
        ->getMockBuilder('\phpunitTests\AuthorizeNetAIM')
        ->disableOriginalConstructor()
        ->setConstructorArgs(
            array(
                \phpunitTests\Payment::API_ID,
                \phpunitTests\Payment::TRANS_KEY
            )
        )
        ->setMethods(array('authorizeAndCapture'))
        ->getMock();

    $authorizeNetResponse = new \stdClass();
    $authorizeNetResponse->approved = false;
    $authorizeNetResponse->error_message = $exceptionMessage;

    $authorizeNetAIM->expects($this->once())
        ->method('authorizeAndCapture')
        ->will($this->returnValue($authorizeNetResponse));

    $arrayDetails = array(
        'amount'   => 123,
        'card_num' => '1234567812345678',
        'exp_date' => '04/07',
    );

    $payment = $this
        ->getMockBuilder('\phpunitTests\Payment')
        ->setMethods(array('hash'))
        ->getMock();

    $payment->processPayment($arrayDetails, $authorizeNetAIM);
}
```

Stub one method

Statics are Evil… Or Are They?

- Statics are convenient
- Statics are quick to use

- Statics are now easy to mock*
  - *Only if both caller and callee are in same class

- Statics create dependencies within your code
- Static properties keep values
  - PHPUnit has a “backupStaticAttributes” flag

Mocking Static Methods

Original Code

```php
<?php
class Foo
{
    public static function doSomething()
    {
        return static::helper();
    }

    public static function helper()
    {
        return 'foo';
    }
}
```

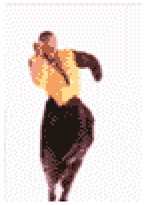
Static method call within Foo class

Test Code

```php
<?php
class FooTest extends PHPUnit_Framework_TestCase
{
    public function testDoSomething()
    {
        $class = $this->getMockClass(
                /* name of class to mock */
            'Foo',
                /* list of methods to mock */
            array('helper')
        );

        $class::staticExpects($this->any())
              ->method('helper')
              ->will($this->returnValue('bar'));

        $this->assertEquals(
            'bar',
            $class::doSomething()
        );
    }
}
```

Taken directly from Sebastion Bergmann's Website
http://sebastian-bergmann.de/archives/883-Stubbing-and-Mocking-Static-Methods.html

Can’t Mock This



- Can’t mock static calls to outside classes!

```php
<?php
class Foo
{
    public static function doSomething()
    {
        return PaymentException::helper();
    }

    public static function helpe
    {
        return 'foo';
    }
}
```

When to Use Statics?

- Same class
- Non-complicated operations
- Never

Annotations

- @covers
  - Tells what method is being tested
  - Great for coverage reports
- @group
  - Separate tests into named groups
  - Don’t run full test suite
- @test
  - May as well!
- @dataProvider
  - Run single test with different input
- Many more!

@test

```php
<?php

class PaymentTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @test
     */
    public function processPaymentReturnTrueOnApprovedResponse()
    {
        // ...
    }

    /**
     * @test
     */
    public function processPaymentThrowsExceptionOnUnapproved()
    {
        // ...
    }
}
```

@group

```php
<?php

class PaymentTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @test
     * @group me
     */
    public function processPaymentReturnTrueOnApprovedResponse()
    {
        // ...
    }

    /**
     * @test
     * @group exceptions
     */
    public function processPaymentThrowsExceptionOnUnapproved()
    {
        // ...
    }
}
```

@covers

```php
<?php

class PaymentTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @test
     * @covers \phpunitTests\Payment::processPayment
     * @group me
     */
    public function processPaymentReturnTrueOnApprovedResponse()
    {
        // ...
    }

    /**
     * @test
     * @covers \phpunitTests\Payment::processPayment
     * @group exceptions
     */
    public function processPaymentThrowsExceptionOnUnapproved()
    {
        // ...
    }
}
```

@dataProvider 1/2

Original Code

```php
<?php

namespace phpunitTests;

class Sluggify
{
    public function sluggify(
        $string,
        $delimiter = '-',
        $maxLength = 96
    ){
        $clean = iconv('UTF-8', 'ASCII//TRANSLIT', $string);
        $clean = preg_replace("%[^-/+|\w ]%", '', $clean);
        $clean = strtolower(
                trim(substr($clean, 0, $maxLength), '-'));
        $clean =
            preg_replace("/[\/_|+ -]+/", $delimiter, $clean);

        return $clean;
    }
}
```

Same overall code, different input

http://cubiq.org/the-perfect-php-clean-url-generator

Test Code

```php
<?php

class SluggifyTest extends \PHPUnit_Framework_TestCase
{
    public function sluggifyReturnsCorrectStringTestOne()
    {
        $sluggify = new \phpunitTests\Sluggify();

        $rawString = "Perch頬'erba 戅erde?"."'";
        $expectedString = 'perche-lerba-e-verde';

        $this->assertEquals(
            $expectedString,
            $sluggify->sluggify($rawString)
        );
    }

    public function sluggifyReturnsCorrectStringTestTwo()
    {
        $sluggify = new \phpunitTests\Sluggify();

        $rawString = "Peux-tu m'aider s'il te pla□".",";
        $expectedString = 'peux-tu-maider-sil-te-plait';

        $this->assertEquals(
            $expectedString,
            $sluggify->sluggify($rawString)
        );
    }

    public function sluggifyReturnsCorrectStringTestThree()
    {
        $sluggify = new \phpunitTests\Sluggify();

        $rawString = "T驃 efter nu  fn vi f dig bort";
        $expectedString = 'tank-efter-nu-forrn-vi-foser-dig-bort';

        $this->assertEquals(
            $expectedString,
            $sluggify->sluggify($rawString)
        );
    }
}
```

@dataProvider 2/2

Original Code

```php
<?php

namespace phpunitTests;

class Sluggify
{
    public function sluggify(
        $string,
        $delimiter = '-',
        $maxLength = 96
    ){
        $clean = iconv('UTF-8', 'ASCII//TRANSLIT', $string);
        $clean = preg_replace("%[^-/+|\w ]%", '', $clean);
        $clean = strtolower(
                    trim(substr($clean, 0, $maxLength), '-'));
        $clean =
            preg_replace("/[\/_|+ -]+/", $delimiter, $clean);

        return $clean;
    }
}
```

Test Code

```php
<?php

class SluggifyTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @test
     * @dataProvider providerSluggifyReturnsSluggifiedString
     */
    public function sluggifyReturnsSluggifiedString(
        $rawString, $expectedResult
    ){
        $sluggify = new \phpunitTests\Sluggify();

        $this->assertEquals(
            $expectedResult,
            $sluggify->sluggify($rawString)
        );
    }

    /**
     * Provider for sluggifyReturnsSluggifiedString
     */
    public function providerSluggifyReturnsSluggifiedString()
    {
        return array(
            array(
                "Perch頻'erba 齇erde?"."'",
                'perche-lerba-e-verde',
            ),
            array(
                "Peux-tu m'aider s'il te pla□".",",
                'peux-tu-maider-sil-te-plait',
            ),
            array(
                "T驃 efter nu  fn vi f dig bort",
                'tank-efter-nu-forrn-vi-foser-dig-bort',
            ),
        );
    }
}
```

setUp() && tearDown()

- setUp()
  - Runs code before *each* test method
  - Set up class variables

- tearDown()
  - Runs code after *each* test method
  - Useful for database interactions

setUpBeforeClass()

```php
<?php

class TestBase extends \PHPUnit_Framework_TestCase
{
    static $runOncePerSuite = false;

    public static function setUpBeforeClass()
    {
        if (!self::$runOncePerSuite) {
            /**
             * Requires table yumiliciousTests to exist.
             * Drops all data from this table and clones yumilicious into it
             */
            exec(
                'mysqldump -u root --no-data --add-drop-table yumiliciousTests | ' .
                'grep ^DROP | ' .
                'mysql -u root yumiliciousTests && ' .
                'mysqldump -u root yumilicious | ' .
                'mysql -u root yumiliciousTests'
            );

            self::$runOncePerSuite = true;
        }
    }
}
```

Extending PHPUnit

```php
<?php

/**
 * Some useful methods to make testing with PHPUnit faster and more fun
 */
abstract class TestBase extends \PHPUnit_Framework_TestCase
{

    /**
     * Set protected/private attribute of object
     *
     * @param object &$object       Object containing attribute
     * @param string $attributeName Attribute name to change
     * @param string $value         Value to set attribute to
     *
     * @return null
     */
    public function setAttribute(&$object, $attributeName, $value)
    {
        $class = is_object($object) ? get_class($object) : $object;

        $reflection = new \ReflectionProperty($class, $attributeName);
        $reflection->setAccessible(true);
        $reflection->setValue($object, $value);
    }

    /**
     * Call protected/private method of a class.
     *
     * @param object &$object    Instantiated object that we will run method on.
     * @param string $methodName Method name to call
     * @param array  $parameters Array of parameters to pass into method.
     *
     * @return mixed Method return.
     */
    public function invokeMethod(&$object, $methodName, array $parameters = array())
    {
        $reflection = new \ReflectionClass(get_class($object));
        $method = $reflection->getMethod($methodName);
        $method->setAccessible(true);

        return $method->invokeArgs($object, $parameters);
    }

}
```

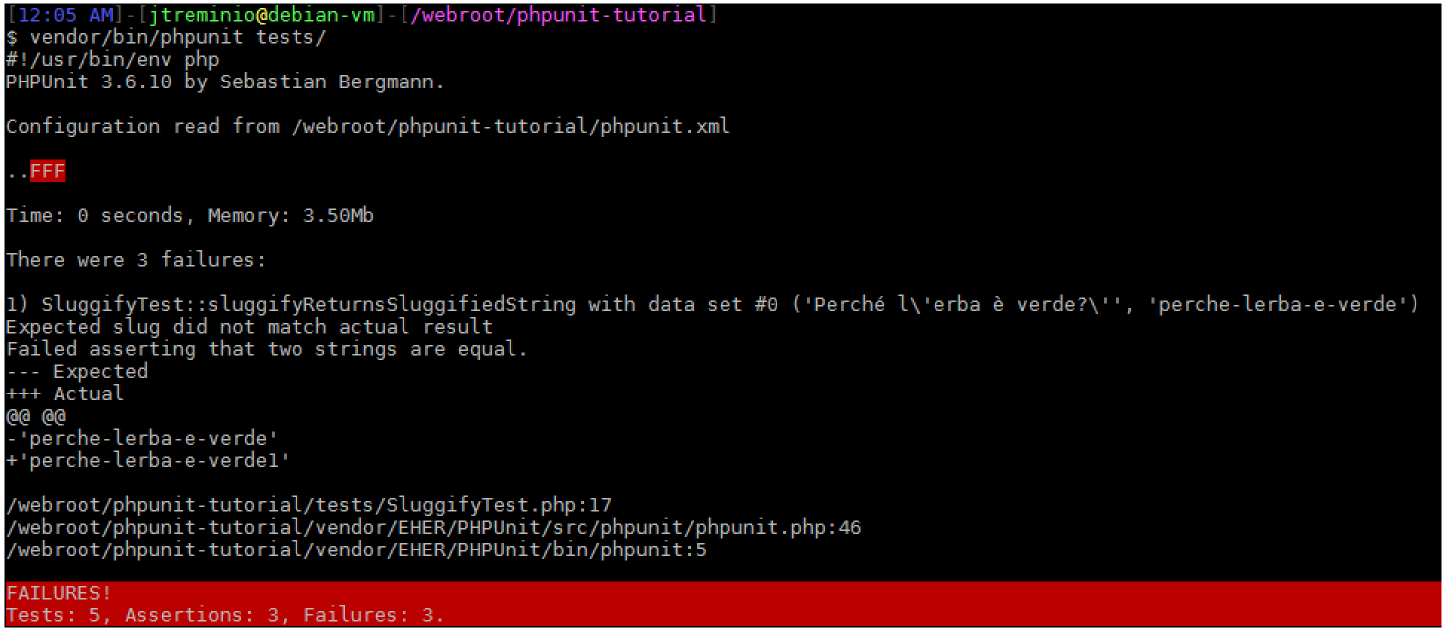
XML Config File

phpunit.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<phpunit backupGlobals="false"
    backupStaticAttributes="true"
    colors="true"
    convertErrorsToExceptions="true"
    convertNoticesToExceptions="true"
    convertWarningsToExceptions="true"
    processIsolation="false"
    stopOnFailure="false"
    stopOnError="false"
    stopOnIncomplete="false"
    stopOnSkipped="false"
    syntaxCheck="false"
    bootstrap="index.php">
    <testsuites>
        <testsuite name="Application Test Suite">
            <directory>./tests/</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

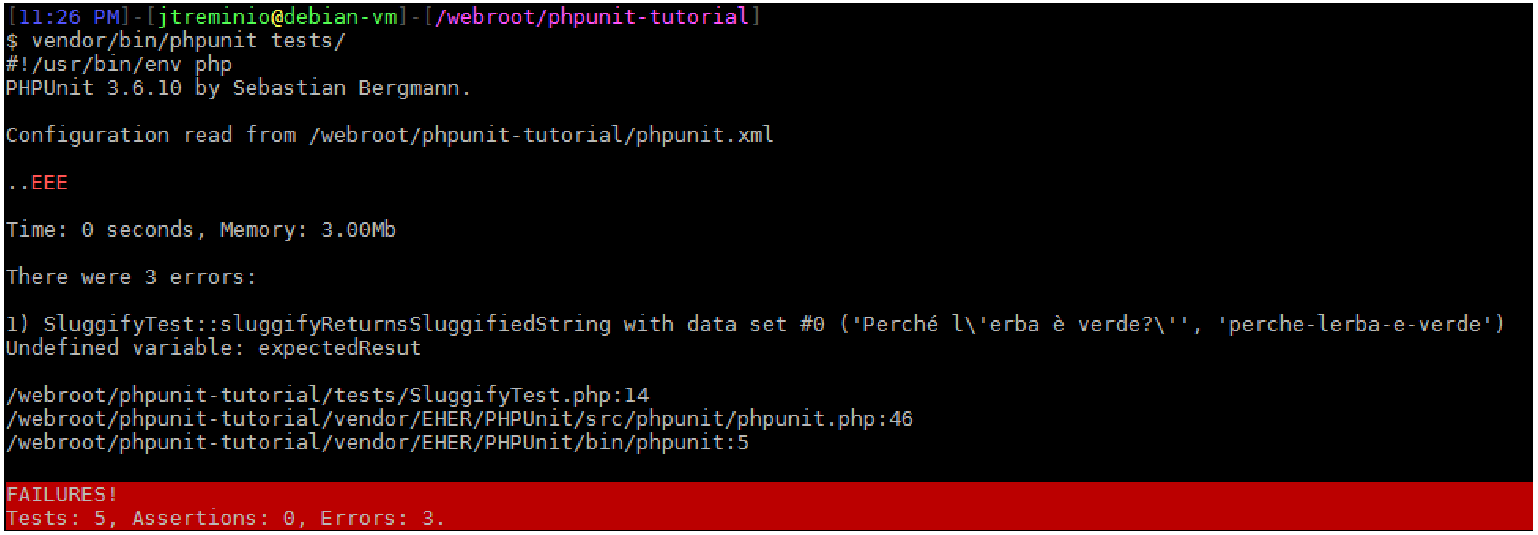
Errors and Failures

- Failures



```
[12:05 AM]-[jtreminio@debian-vm]-[/webroot/phpunit-tutorial]
$ vendor/bin/phpunit tests/
#!/usr/bin/env php
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /webroot/phpunit-tutorial/phpunit.xml

..FFF

Time: 0 seconds, Memory: 3.50Mb

There were 3 failures:

1) SluggifyTest::sluggifyReturnsSluggifiedString with data set #0 ('Perché l\'erba è verde?\'', 'perche-lerba-e-verde')
Expected slug did not match actual result
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'perche-lerba-e-verde'
+'perche-lerba-e-verde1'

/webroot/phpunit-tutorial/tests/SluggifyTest.php:17
/webroot/phpunit-tutorial/vendor/EHER/PHPUnit/src/phpunit/phpunit.php:46
/webroot/phpunit-tutorial/vendor/EHER/PHPUnit/bin/phpunit:5

FAILURES!
Tests: 5, Assertions: 3, Failures: 3.
```

- Errors



```
[11:26 PM]-[jtreminio@debian-vm]-[/webroot/phpunit-tutorial]
$ vendor/bin/phpunit tests/
#!/usr/bin/env php
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /webroot/phpunit-tutorial/phpunit.xml

..EEE

Time: 0 seconds, Memory: 3.00Mb

There were 3 errors:

1) SluggifyTest::sluggifyReturnsSluggifiedString with data set #0 ('Perché l\'erba è verde?\'', 'perche-lerba-e-verde')
Undefined variable: expectedResut

/webroot/phpunit-tutorial/tests/SluggifyTest.php:14
/webroot/phpunit-tutorial/vendor/EHER/PHPUnit/src/phpunit/phpunit.php:46
/webroot/phpunit-tutorial/vendor/EHER/PHPUnit/bin/phpunit:5

FAILURES!
Tests: 5, Assertions: 0, Errors: 3.
```

Mocking Native PHP Functions

- DON’T USE RUNKIT!
  - Allows redefining PHP functions at runtime
- Wrap functions in class methods
  - Allows for easy mocking and stubbing
- Why mock native PHP functions?
  - Mostly shouldn’t
  - cURL, crypt

Classes Should Remind Ignorant

- Should not know they are being tested
- Never change original files with test-only code
- Creating wrappers for mocks is OK

No ifs or Loops in Tests

- Tests should remain simple
- Consider using @dataProvider
- Consider splitting out the test
- Consider refactoring original class

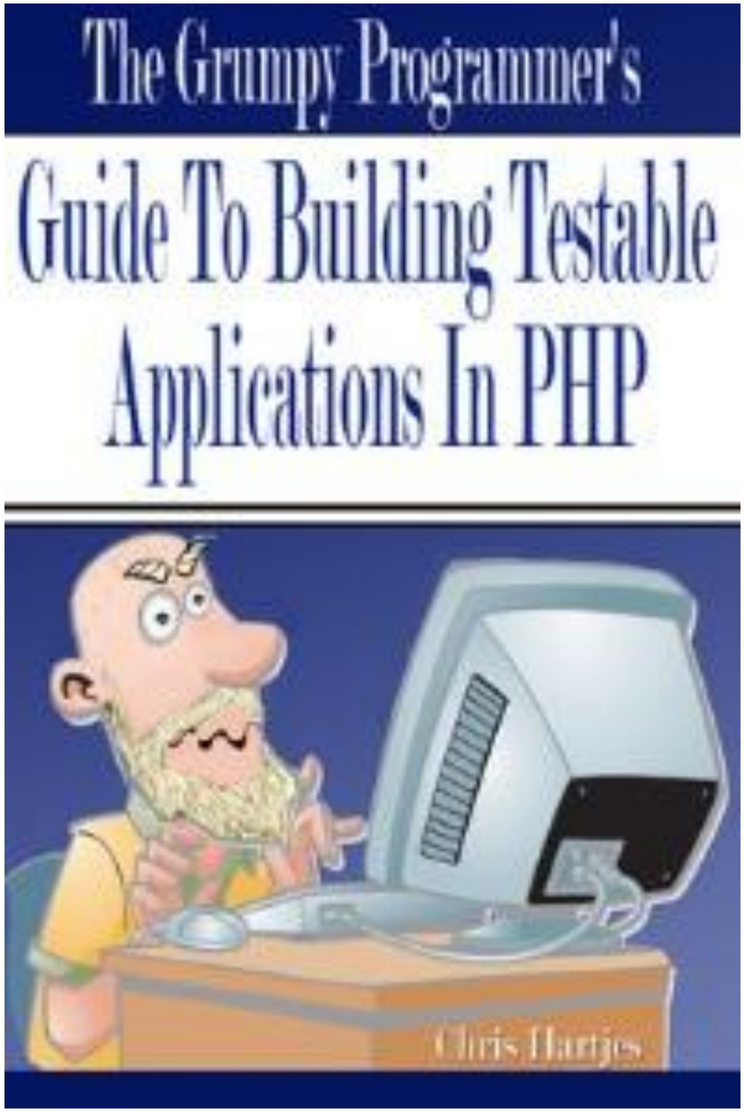
Few Assertions!

- As few assertions as possible per method

- Max one master assertion

Further Reading

- Upcoming Series
  - http://www.jtreminio.com
  - Multi-part
  - Much greater detail
- Chris Hartjes'
  - *The Grumpy Programmer's Guide To Building Testable PHP Applications*



php
TRY AND STOP US