Prev (installation.html) Next (textui.html)

Chapter 2. Writing Tests for PHPUnit

Example 2.1 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.examples.StackTest.php) shows how we can write tests using PHPUnit that exercise PHP's array operations. The example introduces the basic conventions and steps for writing tests with PHPUnit:

1. The tests for a class `Class` go into a class `ClassTest`.
2. `ClassTest` inherits (most of the time) from `PHPUnit\Framework\TestCase`.
3. The tests are public methods that are named `test*`.

   Alternatively, you can use the `@test` annotation in a method's docblock to mark it as a test method.
4. Inside the test methods, assertion methods such as `assertEquals()` (see Appendix A (appendixes.assertions.html)) are used to assert that an actual value matches an expected value.

Example 2.1: Testing array operations with PHPUnit

```
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    public function testPushAndPop()
    {
        $stack = [];
        $this->assertEquals(0, count($stack));

        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertEquals(1, count($stack));

        $this->assertEquals('foo', array_pop($stack));
        $this->assertEquals(0, count($stack));
    }
}
?>
```

Whenever you are tempted to type something into a `print` statement or a debugger expression, write it as a test instead.

--Martin Fowler

Test Dependencies

Unit Tests are primarily written as a good practice to help developers identify and fix bugs, to refactor code and to serve as documentation for a unit of software under test. To achieve these benefits, unit tests ideally should cover all the possible paths in a program. One unit test usually covers one specific path in one function or method. However a test method is not necessary an encapsulated, independent entity. Often there are implicit dependencies between test methods, hidden in the implementation scenario of a test.

--Adrian Kuhn et. al.

PHPUnit supports the declaration of explicit dependencies between test methods. Such dependencies do not define the order in which the test methods are to be executed but they allow the returning of an instance of the test fixture by a producer and passing it to the dependent consumers.

- A producer is a test method that yields its unit under test as return value.
- A consumer is a test method that depends on one or more producers and their return values.

Example 2.2 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.examples.StackTest2.php) shows how to use the `@depends` annotation to express dependencies between test methods.

Example 2.2: Using the `@depends` annotation to express dependencies

```php
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    public function testEmpty()
    {
        $stack = [];
        $this->assertEmpty($stack);

        return $stack;
    }

    /**
     * @depends testEmpty
     */
    public function testPush(array $stack)
    {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);

        return $stack;
    }

    /**
     * @depends testPush
     */
    public function testPop(array $stack)
    {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
?>
```

In the example above, the first test, `testEmpty()` , creates a new array and asserts that it is empty. The test then returns the fixture as its result. The second test, `testPush()` , depends on `testEmpty()` and is passed the result of that depended-upon test as its argument. Finally, `testPop()` depends upon `testPush()` .

Note

The return value yielded by a producer is passed "as-is" to its consumers by default. This means that when a producer returns an object a reference to that object is passed to the consumers. When a copy should be used instead of a reference then `@depends clone` should be used instead of `@depends`.

To quickly localize defects, we want our attention to be focussed on relevant failing tests. This is why PHPUnit skips the execution of a test when a depended-upon test has failed. This improves defect localization by exploiting the dependencies between tests as shown in Example 2.3 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.examples.DependencyFailureTest.php).

Example 2.3: Exploiting the dependencies between tests

```php
<?php
use PHPUnit\Framework\TestCase;

class DependencyFailureTest extends TestCase
{
    public function testOne()
    {
        $this->assertTrue(false);
    }

    /**
     * @depends testOne
     */
    public function testTwo()
    {
    }
}
?>
```

```
phpunit --verbose DependencyFailureTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

FS

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) DependencyFailureTest::testOne
Failed asserting that false is true.

/home/sb/DependencyFailureTest.php:6

There was 1 skipped test:

1) DependencyFailureTest::testTwo
This test depends on "DependencyFailureTest::testOne" to pass.


FAILURES!
Tests: 1, Assertions: 1, Failures: 1, Skipped: 1.
```

A test may have more than one `@depends` annotation. PHPUnit does not change the order in which tests are executed, you have to ensure that the dependencies of a test can actually be met before the test is run.

A test that has more than one `@depends` annotation will get a fixture from the first producer as the first argument, a fixture from the second producer as the second argument, and so on. See Example 2.4 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.examples.MultipleDependencies.php)

Example 2.4: Test with multiple dependencies

```php
<?php
use PHPUnit\Framework\TestCase;

class MultipleDependenciesTest extends TestCase
{
    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

    public function testProducerSecond()
    {
        $this->assertTrue(true);
        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     */
    public function testConsumer()
    {
        $this->assertEquals(
            ['first', 'second'],
            func_get_args()
        );
    }
}
?>
```

```
phpunit --verbose MultipleDependenciesTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

...

Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 3 assertions)
```

Data Providers

A test method can accept arbitrary arguments. These arguments are to be provided by a data provider method (`additionProvider()` in Example 2.5 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.data-providers.examples.DataTest.php)). The data provider method to be used is specified using the `@dataProvider` annotation.

A data provider method must be `public` and either return an array of arrays or an object that implements the `Iterator` interface and yields an array for each iteration step. For each array that is part of the collection the test method will be called with the contents of the array as its arguments.

Example 2.5: Using a data provider that returns an array of arrays

```
<?php
use PHPUnit\Framework\TestCase;

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return [
            [0, 0, 0],
            [0, 1, 1],
            [1, 0, 1],
            [1, 1, 3]
        ];
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 (1, 1, 3)
Failed asserting that 2 matches expected 3.

/home/sb/DataTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

When using a large number of datasets it's useful to name each one with string key instead of default numeric. Output will be more verbose as it'll contain that name of a dataset that breaks a test.

Example 2.6: Using a data provider with named datasets

```php
<?php
use PHPUnit\Framework\TestCase;

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return [
            'adding zeros'  => [0, 0, 0],
            'zero plus one' => [0, 1, 1],
            'one plus zero' => [1, 0, 1],
            'one plus one'  => [1, 1, 3]
        ];
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set "one plus one" (1, 1, 3)
Failed asserting that 2 matches expected 3.

/home/sb/DataTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

Example 2.7: Using a data provider that returns an Iterator object

```php
<?php
use PHPUnit\Framework\TestCase;

require 'CsvFileIterator.php';

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return new CsvFileIterator('data.csv');
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 ('1', '1', '3')
Failed asserting that 2 matches expected '3'.

/home/sb/DataTest.php:11

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

Example 2.8: The CsvFileIterator class

```php
<?php
use PHPUnit\Framework\TestCase;

class CsvFileIterator implements Iterator {
    protected $file;
    protected $key = 0;
    protected $current;

    public function __construct($file) {
        $this->file = fopen($file, 'r');
    }

    public function __destruct() {
        fclose($this->file);
    }

    public function rewind() {
        rewind($this->file);
        $this->current = fgetcsv($this->file);
        $this->key = 0;
    }

    public function valid() {
        return !feof($this->file);
    }

    public function key() {
        return $this->key;
    }

    public function current() {
        return $this->current;
    }

    public function next() {
        $this->current = fgetcsv($this->file);
        $this->key++;
    }
}
?>
```

When a test receives input from both a `@dataProvider` method and from one or more tests it `@depends` on, the arguments from the data provider will come before the ones from depended-upon tests. The arguments from depended-upon tests will be the same for each data set. See Example 2.9 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.data-providers.examples.DependencyAndDataProviderCombo.php)

Example 2.9: Combination of @depends and @dataProvider in same test

```
<?php
use PHPUnit\Framework\TestCase;

class DependencyAndDataProviderComboTest extends TestCase
{
    public function provider()
    {
        return [['provider1'], ['provider2']];
    }

    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

    public function testProducerSecond()
    {
        $this->assertTrue(true);
        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     * @dataProvider provider
     */
    public function testConsumer()
    {
        $this->assertEquals(
            ['provider1', 'first', 'second'],
            func_get_args()
        );
    }
}
?>
```

```
phpunit --verbose DependencyAndDataProviderComboTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 3.50Mb

There was 1 failure:

1) DependencyAndDataProviderComboTest::testConsumer with data set #1 ('provi
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
Array (
-    0 => 'provider1'
+    0 => 'provider2'
1 => 'first'
2 => 'second'
)

/home/sb/DependencyAndDataProviderComboTest.php:31

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

Note

When a test depends on a test that uses data providers, the depending test will be executed when the test it depends upon is successful for at least one data set. The result of a test that uses data providers cannot be injected into a depending test.

Note

All data providers are executed before both the call to the `setUpBeforeClass` static method and the first call to the `setUp` method. Because of that you can't access any variables you create there from within a data provider. This is required in order for PHPUnit to be able to compute the total number of tests.

Testing Exceptions

Example 2.10 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.exceptions.examples.ExceptionTest.php) shows how to use the `expectException()` method to test whether an exception is thrown by the code under test.

Example 2.10: Using the expectException() method

```php
<?php
use PHPUnit\Framework\TestCase;

class ExceptionTest extends TestCase
{
    public function testException()
    {
        $this->expectException(InvalidArgumentException::class);
    }
}
?>
```

```
phpunit ExceptionTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

In addition to the `expectException()` method the `expectExceptionCode()`, `expectExceptionMessage()`, and `expectExceptionMessageRegExp()` methods exist to set up expectations for exceptions raised by the code under test.

Alternatively, you can use the `@expectedException`, `@expectedExceptionCode`, `@expectedExceptionMessage`, and `@expectedExceptionMessageRegExp` annotations to set up expectations for exceptions raised by the code under test. Example 2.11 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.exceptions.examples.ExceptionTest2.php) shows an example.

Example 2.11: Using the @expectedException annotation

```
<?php
use PHPUnit\Framework\TestCase;

class ExceptionTest extends TestCase
{
    /**
     * @expectedException InvalidArgumentException
     */
    public function testException()
    {
    }
}
?>
```

```
phpunit ExceptionTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

Testing PHP Errors

By default, PHPUnit converts PHP errors, warnings, and notices that are triggered during the execution of a test to an exception. Using these exceptions, you can, for instance, expect a test to trigger a PHP error as shown in Example 2.12 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.exceptions.examples.ErrorTest.php).

Note

PHP's `error_reporting` runtime configuration can limit which errors PHPUnit will convert to exceptions. If you are having issues with this feature, be sure PHP is not configured to suppress the type of errors you're testing.

Example 2.12: Expecting a PHP error using @expectedException

```
<?php
use PHPUnit\Framework\TestCase;

class ExpectedErrorTest extends TestCase
{
    /**
     * @expectedException PHPUnit_Framework_Error
     */
    public function testFailingInclude()
    {
        include 'not_existing_file.php';
    }
}
?>
```

```
phpunit -d error_reporting=2 ExpectedErrorTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

.

Time: 0 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

`PHPUnit_Framework_Error_Notice` and `PHPUnit_Framework_Error_Warning` represent PHP notices and warnings, respectively.

Note

You should be as specific as possible when testing exceptions. Testing for classes that are too generic might lead to undesirable side-effects. Accordingly, testing for the `Exception` class with `@expectedException` or `setExpectedException()` is no longer permitted.

When testing that relies on php functions that trigger errors like `fopen` it can sometimes be useful to use error suppression while testing. This allows you to check the return values by suppressing notices that would lead to a phpunit `PHPUnit_Framework_Error_Notice`.

Example 2.13: Testing return values of code that uses PHP Errors

```
<?php
use PHPUnit\Framework\TestCase;

class ErrorSuppressionTest extends TestCase
{
    public function testFileWriting() {
        $writer = new FileWriter;
        $this->assertFalse(@$writer->write('/is-not-writeable/file', 'stuff'
    }
}
class FileWriter
{
    public function write($file, $content) {
        $file = fopen($file, 'w');
        if($file == false) {
            return false;
        }
        // ...
    }
}

?>
```

```
phpunit ErrorSuppressionTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

Without the error suppression the test would fail reporting `fopen(/is-not-writeable/file): failed to open stream: No such file or directory`.

Testing Output

Sometimes you want to assert that the execution of a method, for instance, generates an expected output (via `echo` or `print`, for example). The `PHPUnit\Framework\TestCase` class uses PHP's Output Buffering (http://www.php.net/manual/en/ref.outcontrol.php) feature to provide the functionality that is necessary for this.

Example 2.14 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.output.examples.OutputTest.php) shows how to use the `expectOutputString()` method to set the expected output. If this expected output is not generated, the test will be counted as a failure.

Example 2.14: Testing the output of a function or method

```php
<?php
use PHPUnit\Framework\TestCase;

class OutputTest extends TestCase
{
    public function testExpectFooActualFoo()
    {
        $this->expectOutputString('foo');
        print 'foo';
    }

    public function testExpectBarActualBaz()
    {
        $this->expectOutputString('bar');
        print 'baz';
    }
}
?>
```

```
phpunit OutputTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) OutputTest::testExpectBarActualBaz
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'


FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

Table 2.1 (writing-tests-for-phpunit.html#writing-tests-for-phpunit.output.tables.api) shows the methods provided for testing output

Table 2.1. Methods for testing output

Method	Meaning
`void expectOutputRegex(string $regularExpression)`	Set up the expectation that the output matches a `$regularExpression`.
`void expectOutputString(string $expectedString)`	Set up the expectation that the output is equal to an `$expectedString`.
`bool setOutputCallback(callable $callback)`	Sets up a callback that is used to, for instance, normalize the actual output.
`string getActualOutput()`	Get the actual output.

Note

A test that emits output will fail in strict mode.

Error output

Whenever a test fails PHPUnit tries its best to provide you with as much context as possible that can help to identify the problem.

Example 2.15: Error output generated when an array comparison fails

```
<?php
use PHPUnit\Framework\TestCase;

class ArrayDiffTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [1, 2,  3, 4, 5, 6],
            [1, 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit ArrayDiffTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArrayDiffTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
     0 => 1
     1 => 2
-    2 => 3
+    2 => 33
     3 => 4
     4 => 5
     5 => 6
 )

/home/sb/ArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

In this example only one of the array values differs and the other values are shown to provide context on where the error occurred.

When the generated output would be long to read PHPUnit will split it up and provide a few lines of context around every difference.

Example 2.16: Error output when an array comparison of an long array fails

```
<?php
use PHPUnit\Framework\TestCase;

class LongArrayDiffTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2,  3, 4, 5, 6],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit LongArrayDiffTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) LongArrayDiffTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
     13 => 2
-    14 => 3
+    14 => 33
     15 => 4
     16 => 5
     17 => 6
 )


/home/sb/LongArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

Edge cases

When a comparison fails PHPUnit creates textual representations of the input values and compares those. Due to that implementation a diff might show more problems than actually exist.

This only happens when using assertEquals or other 'weak' comparison functions on arrays or objects.

Example 2.17: Edge case in the diff generation when using weak comparison

```
<?php
use PHPUnit\Framework\TestCase;

class ArrayWeakComparisonTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [1, 2, 3, 4, 5, 6],
            ['1', 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit ArrayWeakComparisonTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArrayWeakComparisonTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
-    0 => 1
+    0 => '1'
     1 => 2
-    2 => 3
+    2 => 33
     3 => 4
     4 => 5
     5 => 6
 )


/home/sb/ArrayWeakComparisonTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

In this example the difference in the first index between `1` and `'1'` is reported even though assertEquals considers the values as a match.

Prev (installation.html) Next (textui.html)