Cheatsheet for test-driven development with PHPUnit

Oliver Klee | typo3-coding@oliverklee.de | @oliklee
https://github.com/oliverklee/tdd-reader

Version 2.0.2, April 30, 2017, for TYPO3 PHP >= 5.6 and CMS 7.6

License

[1] https://github.com/oliverklee/tdd-reader
[2] http://creativecommons.org/licenses/by-sa/4.0/

Contents

1 File and class naming

1.1 File names

Production code file name	Test file name
Classes/Domain/Model/Shoe.php	Tests/Unit/Domain/Model/ShoeTest.php
Classes/Service/BaristaService.php	Tests/Unit/Service/BaristaServiceTest.php

1.2 Class names

Production code class name	Test class name
Shoes\Shop\Domain\Model\Shoe	Shoes\Shop\Tests\Unit\Domain\Model\ShoeTest
Shoes\Shop\Service\BaristaService	Shoes\Shop\Tests\Unit\Service\BaristaServiceTest

2 Test class structure

2.1 Extbase extensions

There's an example project (the tea example) for this on GitHub:
`https://github.com/oliverklee/ext_tea`

```php
namespace OliverKlee\Shop\Tests\Unit\Domain\Model;

use OliverKlee\Shop\Domain\Model\Article;

class ArticleTest extends \TYPO3\CMS\Core\Tests\UnitTestCase {
    /**
     * @var Article;
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new Article;
        $this->subject->initializeObject();
    }

    /**
     * @test
     */
    public function getNameInitiallyReturnsEmptyString()
    {
        self::assertSame('', $this->subject->getName());
    }

    /**
     * @test
     */
    public function setNameSetsName()
    {
        $name = 'foo bar';

        $this->subject->setName($name);

        self::assertSame($name, $this->subject->getName());
    }

    // ...
}
```

2.2 Non-extbase extensions

```
class AttachmentTest extends \Tx_Phpunit_TestCase {
    /**
     * @var \Tx_Oelib_Attachment
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new \Tx_Oelib_Attachment();
    }

    /**
     * @test
     */
    public function getFileNameInitiallyReturnsAnEmptyString()
    {
        self::assertSame('', $this->subject->getFileName());
    }

    /**
     * @test
     */
    public function getFileNameWithFileNameSetReturnsFileName()
    {
        $fileName = 'test.txt';

        $this->subject->setFileName($fileName);

        self::assertSame($fileName, $this->subject->getFileName());
    }

    /**
     * @test
     * @expectedException \InvalidArgumentException
     */
    public function setFileNameWithEmptyFileNameThrowsException()
    {
        $this->subject->setFileName('');
    }

    // ...
}
```

2.3 Non-TYPO3 PHP projects with Composer

There is an empty starter project for this on GitHub:
`https://github.com/oliverklee/tdd-seed`

2.3.1 composer.json

This setup installs PHPUnit and vfsStream **for PHP up to 5.6**:

```
{
    "require-dev": {
        "phpunit/phpunit": "^5.7.19",
        "mikey179/vfsStream": "^1.6.4"
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "psr-4": {
            "..."
        }
    }
}
```

This setup installs PHPUnit and vfsStream **for PHP 6**:

```
{
    "require-dev": {
        "phpunit/phpunit": "^6.0.13",
        "mikey179/vfsStream": "^1.6.4"
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "psr-4": {
            "..."
        }
    }
}
```

2.3.2 Test case

```
1  namespace OliverKlee\Books\Tests\Unit\Domain\Model;
2
3  use OliverKlee\Books\Domain\Model;
4
5  class BookTest extends \PHPUnit_Framework_TestCase {
6      /**
7       * @var Book
8       */
9      protected $subject = null;
10
11     protected function setUp()
12     {
13         $this->subject = new Book();
14     }
15
16     /**
17      * @test
18      */
19     public function getTitleInitiallyReturnsEmptyString()
20     {
21         self::assertSame('', $this->subject->getTitle());
22     }
23
24     /**
25      * @test
26      */
27     public function setTitleSetsTitle()
28     {
29         $title = 'foo bar';
30
31         $this->subject->setTitle($title);
32
33         self::assertSame('foo bar', $this->subject->getTitle());
34     }
35 }
```

3 Testing for Exceptions

3.1 Test for the Exception class only

```
/**
 * @test
 * @expectedException InvalidArgumentException
 */
public function createBreadWithNegativeSizeThrowsException()
{
    $this->subject->createBread(-1);
}
```

3.2 Test for the exception class, message and the code

```
/**
 * @test
 * @expectedException \InvalidArgumentException
 * @expectedExceptionMessage size must be > 0.
 * @expectedExceptionCode 1323700434
 */
public function createBreadWithNegativeSizeThrowsException()
{
    $this->subject->createBread(-1);
}
```

4 Testing abstract classes

4.1 Using the PHPUnit mock builder

This will create an instance of the abstract class with all abstract methods mocked.

```php
namespace OliverKlee\Coffee\Tests\Unit\Domain\Model;

use OliverKlee\Coffee\Domain\Model\AbstractBeverage;

class AbstractBeverageTest {
    /**
     * @var AbstractBeverage|\PHPUnit_Framework_MockObject_MockObject
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = $this->getMockForAbstractClass(
            AbstractBeverage::class
        );
    }
```

4.2 Creating a concrete subclass

This is recommended if you need to provide your subclass with some additional or specific behavior.

In `Tests/Unit/Domain/Model/Fixtures/`, create a subclass of the abstract class:

```php
namespace OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures;

class TestingBeverage extends \OliverKlee\Coffee\Domain\Model\AbstractBeverage {
    // ...
}
```

Then you can use and instantiate the concrete subclass in your unit tests:

```php
use OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures\TestingBeverage;

class AbstractBeverageTest {
    /**
     * @var TestingBeverage
     *
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new TestingBeverage();
    }
```

5 Using the testing framework of the PHPUnit TYPO3 extension

```php
class DataMapperTest extends \Tx_Phpunit_TestCase {
    /**
     * @var \Tx_Phpunit_Framework
     */
    protected $testingFramework = null;

    protected $subject = null;

    protected function setUp()
    {
        $this->testingFramework = new \Tx_Phpunit_Framework('tx_oelib');

        $this->subject = new ...;
    }

    protected function tearDown()
    {
        $this->testingFramework->cleanUp();
    }

    /**
     * @test
     */
    public function findWithUidOfExistingRecordReturnsModelDataFromDatabase()
    {
        $title = 'foo';
        $uid = $this->testingFramework->createRecord(
            'tx_oelib_test', ['title' => $title]
        );

        self::assertSame($title, $this->subject->find($uid)->getTitle());
    }
```

5.1 Executable examples

The functional tests for the FileUtility class in the tea example show what tests with vfsStream can look like.

6 Using mock file systems with vfsStream

6.1 Setting it all up

```
use org\bovigo\vfs\vfsStream;
use org\bovigo\vfs\vfsStreamDirectory;

/**
 * @var \org\bovigo\vfs\vfsStreamFile
 */
protected $moreStuff;

protected function setUp()
{
    // This is the same as ::register and ::setRoot.
    $this->root = vfsStream::setup('home');
    $this->targetFilePath = vfsStream::url('home/target.txt');

    $this->subject = new ...
}
```

6.2 Using the files

```
/**
 * @test
 */
public function concatenateWithOneEmptySourceFileCreatesEmptyTargetFile()
{
    // This is one way to create a file with contents, using PHP's file functions.
    $sourceFileName = vfsStream::url('home/source.txt');
    // Just calling vfsStream::url does not create the file yet.
    // We need to write into it to create it.
    file_put_contents($sourceFileName, '');

    $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);

    self::assertSame('', file_get_contents($this->targetFilePath));
}

/**
 * @test
 */
public function concatenateWithOneFileCopiesContentsFromSourceFileToTargetFile()
{
    // This is vfsStream's way of creating a file with contents.
    $contents = 'Hello world!';
    $sourceFileName = vfsStream::url('home/source.txt');
    vfsStream::newFile('source.txt')->at($this->root)->setContent($contents);

    $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);

    self::assertSame($contents, file_get_contents($this->targetFilePath));
}
```

7 PHPUnit assertions

This list is current for PHPUnit 5.7.x.

```
assertArrayHasKey()
assertClassHasAttribute()
assertArraySubset()
assertClassHasStaticAttribute()
assertContains()
assertContainsOnly()
assertContainsOnlyInstancesOf()
assertCount()
assertDirectoryExists()
assertDirectoryIsReadable()
assertDirectoryIsWritable()
assertEmpty()
assertEqualXMLStructure()
assertEquals()
assertFalse()
assertFileEquals()
assertFileExists()
assertFileIsReadable()
assertFileIsWritable()
assertGreaterThan()
assertGreaterThanOrEqual()
assertInfinite()
assertInstanceOf()
assertInternalType()
assertIsReadable()
assertIsWritable()
assertJsonFileEqualsJsonFile()
assertJsonStringEqualsJsonFile()
assertJsonStringEqualsJsonString()
assertLessThan()
assertLessThanOrEqual()
assertNan()
assertNull()
assertObjectHasAttribute()
assertRegExp()
assertStringMatchesFormat()
assertStringMatchesFormatFile()
assertSame()
assertStringEndsWith()
assertStringEqualsFile()
assertStringStartsWith()
assertThat()
assertTrue()
assertXmlFileEqualsXmlFile()
assertXmlStringEqualsXmlFile()
assertXmlStringEqualsXmlString()
```