

# Cheatsheet for test-driven development with PHPUnit (with and without TYPO3)

Oliver Klee | [typo3-coding@oliverklee.de](mailto:typo3-coding@oliverklee.de) | @oliklee  
<https://github.com/oliverklee/tdd-reader>

Version 2.0.2, April 20, 2017, for TYPO3 CMS 6.2

## License

This handout is licensed under a *Creative Commons* license, in this case under an *Attribution-ShareAlike 4.0 (CC BY-SA 4.0)*. This means that you can use, edit and distribute this handout (even commercially) under the following conditions:

**Attribution.** You need to give credit to the author (me) by listing my name (Oliver Klee). If you also list the source<sup>1</sup>, that would be nice. And if you want to make me happy, please drop me an e-mail if you use this document.

**ShareAlike.** If you edit or change this document or use it as a basis for some other document, you must use the same license for the resulting document.

**Name the license.** If you distribute this document, you'll need to mention or enclose the license.

You can find a more comprehensive version of this license online.<sup>2</sup>

---

<sup>1</sup><https://github.com/oliverklee/tdd-reader>

<sup>2</sup><http://creativecommons.org/licenses/by-sa/4.0/>

# Contents

<b>1</b>	<b>File and class naming</b>	<b>3</b>
1.1	File names . . . . .	3
1.2	Class names . . . . .	3
<b>2</b>	<b>Test class structure</b>	<b>4</b>
2.1	Extbase extensions . . . . .	4
2.2	Non-extbase extensions . . . . .	4
2.3	Non-TYPO3 PHP projects with Composer . . . . .	5
2.3.1	composer.json . . . . .	5
2.3.2	Test case . . . . .	6
<b>3</b>	<b>Testing for Exceptions</b>	<b>8</b>
3.1	Test for the Exception class only . . . . .	8
3.2	Test for the exception class, message and the code . . . . .	8
<b>4</b>	<b>Testing abstract classes</b>	<b>9</b>
4.1	Using the PHPUnit mock builder . . . . .	9
4.2	Creating a concrete subclass . . . . .	9
<b>5</b>	<b>Using the testing framework of the PHPUnit TYPO3 extension</b>	<b>10</b>
5.1	Executable examples . . . . .	10
<b>6</b>	<b>Using mock file systems with vfsStream</b>	<b>11</b>
6.1	Setting it all up . . . . .	11
6.2	Using the files . . . . .	11
<b>7</b>	<b>PHPUnit assertions</b>	<b>12</b>

# 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
OliverKlee\Shop\Domain\Model\Shoe	OliverKlee\Shop\Tests\Unit\Domain\Model\ShoeTest
OliverKlee\Shop\Service\BaristaService	OliverKlee\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](https://github.com/oliverklee/ext_tea)

```
1 namespace OliverKlee\Shop\Tests\Unit\Domain\Model;
2
3 use OliverKlee\Shop\Domain\Model\Article;
4
5 class ArticleTest extends \TYPO3\CMS\Core\Tests\UnitTestCase {
6     /**
7      * @var Article;
8      */
9     protected $subject = null;
10
11     protected function setUp()
12     {
13         $this->subject = new Article;
14         $this->subject->initializeObject();
15     }
16
17     /**
18      * @test
19      */
20     public function getNameInitiallyReturnsEmptyString()
21     {
22         self::assertSame('', $this->subject->getName());
23     }
24
25     /**
26      * @test
27      */
28     public function setNameSetsName()
29     {
30         $name = 'foo bar';
31
32         $this->subject->setName($name);
33
34         self::assertSame($name, $this->subject->getName());
35     }
36
37     // ...
38 }
```

### 2.2 Non-extbase extensions

```
1 class AttachmentTest extends \Tx_Phpunit_TestCase {
2     /**
3      * @var \Tx_Delib_Attachment
4      */
5 }
```

```

5     protected $subject = null;
6
7     protected function setUp()
8     {
9         $this->subject = new \Tx_Oelib_Attachment();
10    }
11
12    /**
13     * @test
14     */
15    public function getFileNameInitiallyReturnsAnEmptyString()
16    {
17        self::assertSame('', $this->subject->getFileName());
18    }
19
20    /**
21     * @test
22     */
23    public function getFileNameWithFileNameSetReturnsFileName()
24    {
25        $fileName = 'test.txt';
26
27        $this->subject->setFileName($fileName);
28
29        self::assertSame($fileName, $this->subject->getFileName());
30    }
31
32    /**
33     * @test
34     * @expectedException \InvalidArgumentException
35     */
36    public function setFileNameWithEmptyFileNameThrowsException()
37    {
38        $this->subject->setFileName('');
39    }
40
41    // ...
42 }

```

## 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:**

```

1 {
2     "require-dev": {
3         "phpunit/phpunit": "^5.7.19",
4         "mikey179/vfsStream": "^1.6.4"

```

```

5     },
6     "autoload": {
7         "psr-4": {
8             "...":
9         }
10    },
11    "autoload-dev": {
12        "psr-4": {
13            "...":
14        }
15    }
16 }

```

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

```

1 {
2     "require-dev": {
3         "phpunit/phpunit": "^6.0.13",
4         "mikey179/vfsStream": "^1.6.4"
5     },
6     "autoload": {
7         "psr-4": {
8             "...":
9         }
10    },
11    "autoload-dev": {
12        "psr-4": {
13            "...":
14        }
15    }
16 }

```

### 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

```
1  /**
2   * @test
3   * @expectedException InvalidArgumentException
4   */
5  public function createBreadWithNegativeSizeThrowsException()
6  {
7      $this->subject->createBread(-1);
8  }
```

### 3.2 Test for the exception class, message and the code

```
1  /**
2   * @test
3   * @expectedException \InvalidArgumentException
4   * @expectedExceptionMessage size must be > 0.
5   * @expectedExceptionCode 1323700434
6   */
7  public function createBreadWithNegativeSizeThrowsException()
8  {
9      $this->subject->createBread(-1);
10 }
```



## 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.

```
1 namespace OliverKlee\Coffee\Tests\Unit\Domain\Model;
2
3 use OliverKlee\Coffee\Domain\Model\AbstractBeverage;
4
5 class AbstractBeverageTest {
6     /**
7      * @var AbstractBeverage|\PHPUnit_Framework_MockObject_MockObject
8      */
9     protected $subject = null;
10
11     protected function setUp()
12     {
13         $this->subject = $this->getMockForAbstractClass(
14             AbstractBeverage::class
15         );
16     }
17 }
```

### 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:

```
1 namespace OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures;
2
3 class TestingBeverage extends \OliverKlee\Coffee\Domain\Model\AbstractBeverage {
4     // ...
5 }
```

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

```
1 use OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures\TestingBeverage;
2
3 class AbstractBeverageTest {
4     /**
5      * @var TestingBeverage
6      */
7     protected $subject = null;
8
9     protected function setUp()
10    {
11        $this->subject = new TestingBeverage();
12    }
13 }
```

## 5 Using the testing framework of the PHPUnit TYPO3 extension

```
1 class DataMapperTest extends \Tx_Phpunit_TestCase {
2     /**
3      * @var \Tx_Phpunit_Framework
4      */
5     protected $testingFramework = null;
6
7     protected $subject = null;
8
9     protected function setUp()
10    {
11        $this->testingFramework = new \Tx_Phpunit_Framework('tx_oelib');
12
13        $this->subject = new ...;
14    }
15
16    protected function tearDown()
17    {
18        $this->testingFramework->cleanUp();
19    }
20
21    /**
22     * @test
23     */
24    public function findWithUidOfExistingRecordReturnsModelDataFromDatabase()
25    {
26        $title = 'foo';
27        $uid = $this->testingFramework->createRecord(
28            'tx_oelib_test', ['title' => $title]
29        );
30
31        self::assertSame($title, $this->subject->find($uid)->getTitle());
32    }
33 }
```

### 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

```
1 use org\bovigo\vfs\vfsStream;
2 use org\bovigo\vfs\vfsStreamDirectory;
3
4 /**
5  * @var \org\bovigo\vfs\vfsStreamFile
6  */
7 protected $moreStuff;
8
9 protected function setUp()
10 {
11     // This is the same as ::register and ::setRoot.
12     $this->root = vfsStream::setup('home');
13     $this->targetFilePath = vfsStream::url('home/target.txt');
14
15     $this->subject = new ...
16 }
```

### 6.2 Using the files

```
1 /**
2  * @test
3  */
4 public function concatenateWithOneEmptySourceFileCreatesEmptyTargetFile()
5 {
6     // This is one way to create a file with contents, using PHP's file functions.
7     $sourceFileName = vfsStream::url('home/source.txt');
8     // Just calling vfsStream::url does not create the file yet.
9     // We need to write into it to create it.
10    file_put_contents($sourceFileName, '');
11
12    $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);
13
14    self::assertSame('', file_get_contents($this->targetFilePath));
15 }
16
17 /**
18  * @test
19  */
20 public function concatenateWithOneFileCopiesContentsFromSourceFileToTargetFile()
21 {
22     // This is vfsStream's way of creating a file with contents.
23     $contents = 'Hello world!';
24     $sourceFileName = vfsStream::url('home/source.txt');
25     vfsStream::newFile('source.txt')->at($this->root->setContent($contents);
26
27     $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);
28
29     self::assertSame($contents, file_get_contents($this->targetFilePath));
30 }
```

## 7 PHPUnit assertions

This list is current for PHPUnit 4.8.x.

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