

# Cheatsheet for test-driven development with TYPO3 CMS

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

Version 2.0.1, May 7, 2016, 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>3</b>
2.1	Extbase extensions . . . . .	3
2.2	Non-extbase extensions . . . . .	5
2.3	Non-TYPO3 PHP projects with Composer . . . . .	6
2.3.1	composer.json . . . . .	6
2.3.2	Test case . . . . .	7
<b>3</b>	<b>Testing for Exceptions</b>	<b>7</b>
3.1	Test for the Exception class only . . . . .	7
3.2	Test for the exception class, message and the code . . . . .	8
<b>4</b>	<b>Testing abstract classes</b>	<b>8</b>
4.1	Using the PHPUnit mock builder . . . . .	8
4.2	Creating a concrete subclass . . . . .	8
<b>5</b>	<b>Using the testing framework of the PHPUnit TYPO3 extension</b>	<b>9</b>
5.1	Executable examples . . . . .	10
<b>6</b>	<b>Using mock file systems with vfsStream</b>	<b>10</b>
6.1	Setting it all up . . . . .	10
6.2	Using the files . . . . .	11
<b>7</b>	<b>PHPUnit assertions</b>	<b>11</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
pi1/class.tx_frubble_pi1.php	Tests/Unit/pi1/pi1Test.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
tx_frubble_pi1	tx_frubble_Tests_Unit_pi1_pi1Test

# 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(
23             '',
24             $this->subject->getName()
25         );
26     }
27
28     /**
29      * @test
30      */
31     public function setNameSetsName()
32     {
33         $this->subject->setName('foo bar');
34
35         self::assertSame(
36             'foo bar',
37             $this->subject->getName()
38         );
39     }
40
41     // ...
42 }

```

## 2.2 Non-extbase extensions

```
1 // You need to require_once the class to test only if your extension
2 // does not make use of ext_autoload.php.
3 // require_once t3lib_extMgm::extPath('oelib') . 'Classes/Attachment.php';
4
5 class Tx_Oelib_Tests_Unit_AttachmentTest extends \Tx_Phunit_TestCase {
6     /**
7      * @var \Tx_Oelib_Attachment
8      */
9     protected $subject = null;
10
11     protected function setUp()
12     {
13         $this->subject = new \Tx_Oelib_Attachment();
14     }
15
16     /**
17      * @test
18      */
19     public function getFileNameInitiallyReturnsAnEmptyString()
20     {
21         self::assertSame(
22             '',
23             $this->subject->getFileName()
24         );
25     }
26
27     /**
28      * @test
29      */
30     public function getFileNameWithFileNameSetReturnsFileName()
31     {
32         $this->subject->setFileName('test.txt');
33
34         self::assertSame(
35             'test.txt',
36             $this->subject->getFileName()
37         );
38     }
39
40     /**
41      * @test
42      * @expectedException \InvalidArgumentException
43      */
44     public function setFileNameWithEmptyFileNameThrowsException()
45     {
46         $this->subject->setFileName('');
47     }
48
49     // ...
50 }
```

## 2.3 Non-TYPO3 PHP projects with Composer

### 2.3.1 composer.json

This setup installs PHPUnit and vfsStream:

```
1 {  
2     "require-dev": {  
3         "phpunit/phpunit": "*",  
4         "mikey179/vfsStream": "*"   
5     },  
6     "autoload": {  
7         "psr-4": {  
8             "..."  
9         }  
10    }  
11 }
```

### 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(
22             '',
23             $this->subject->getTitle()
24         );
25     }
26
27     /**
28      * @test
29      */
30     public function setTitleSetsTitle()
31     {
32         $this->subject->setTitle('foo bar');
33
34         self::assertSame(
35             'foo bar',
36             $this->subject->getTitle()
37         );
38     }
39 }
```

## 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 Tx_Coffee_Domain_Model_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              'OliverKlee\Coffee\Domain\Model\AbstractBeverage'
15          );
16      }
```

### 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 Tx_Coffee_Domain_Model_AbstractBeverageTest {
4     /**
5      * @var TestingBeverage
6      *
7      protected $subject = null;
8
9     protected function setUp()
10    {
11        $this->subject = new TestingBeverage();
12    }

```

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

```

1 class tx_oelib_DataMapperTest extends \Tx_Phunit_TestCase {
2     /**
3      * @var \Tx_Phunit_Framework
4      */
5     protected $testingFramework = null;
6
7     protected $subject = null;
8
9     protected function setUp()
10    {
11        $this->testingFramework = new Tx_Phunit_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        $uid = $this->testingFramework->createRecord(
27            'tx_oelib_test', array('title' => 'foo')
28        );
29
30        self::assertSame(
31            'foo',
32            $this->subject->find($uid)->getTitle()
33        );
34    }

```

## 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, array($sourceFileName));
13
14     self::assertSame(
15         '',
16         file_get_contents($this->targetFilePath)
17     );
18 }
19
20 /**
21 * @test
22 */
23 public function concatenateWithOneFileCopiesContentsFromSourceFileToTargetFile()
24 {
25     // This is vfsStream's way of creating a file with contents.
26     $contents = 'Hello world!';
27     $sourceFileName = vfsStream::url('home/source.txt');
28     vfsStream::newFile('source.txt')->at($this->root)->setContent($contents);
29
30     $this->subject->concatenate($this->targetFilePath, array($sourceFileName));
31
32     self::assertSame(
33         $contents,
34         file_get_contents($this->targetFilePath)
35     );
36 }
```

## 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()
```