Introduction to Unit Tests & Test Driven Development

Ján “mice” MÁC

About me

- Web developer since 2005
- Enterprise & SMB experience

- Former full-stack, focusing on back-end lately:
  - PHP, Python, SQL
  - HTML, CSS, JS

- Contacts:
  - Web: http://mice.sk
  - Twitter: @MicE_sk
  - Facebook: mice.sk
  - StackOverflow: mice
  - GitHub: mice-sk
  - LinkedIn: Ján Mác

Agenda

- Introduction
  - What are “tests”? Why test?
  - Caveats
  - Code example
- Concepts
  - Terminology
  - Types of tests
  - Test life cycle
- Test Driven Development
  - Live demo
- Best practices: Do’s & Don’ts
- Resources
- Q&A

Before we begin...

… does any of you have any practical experience with unit tests?

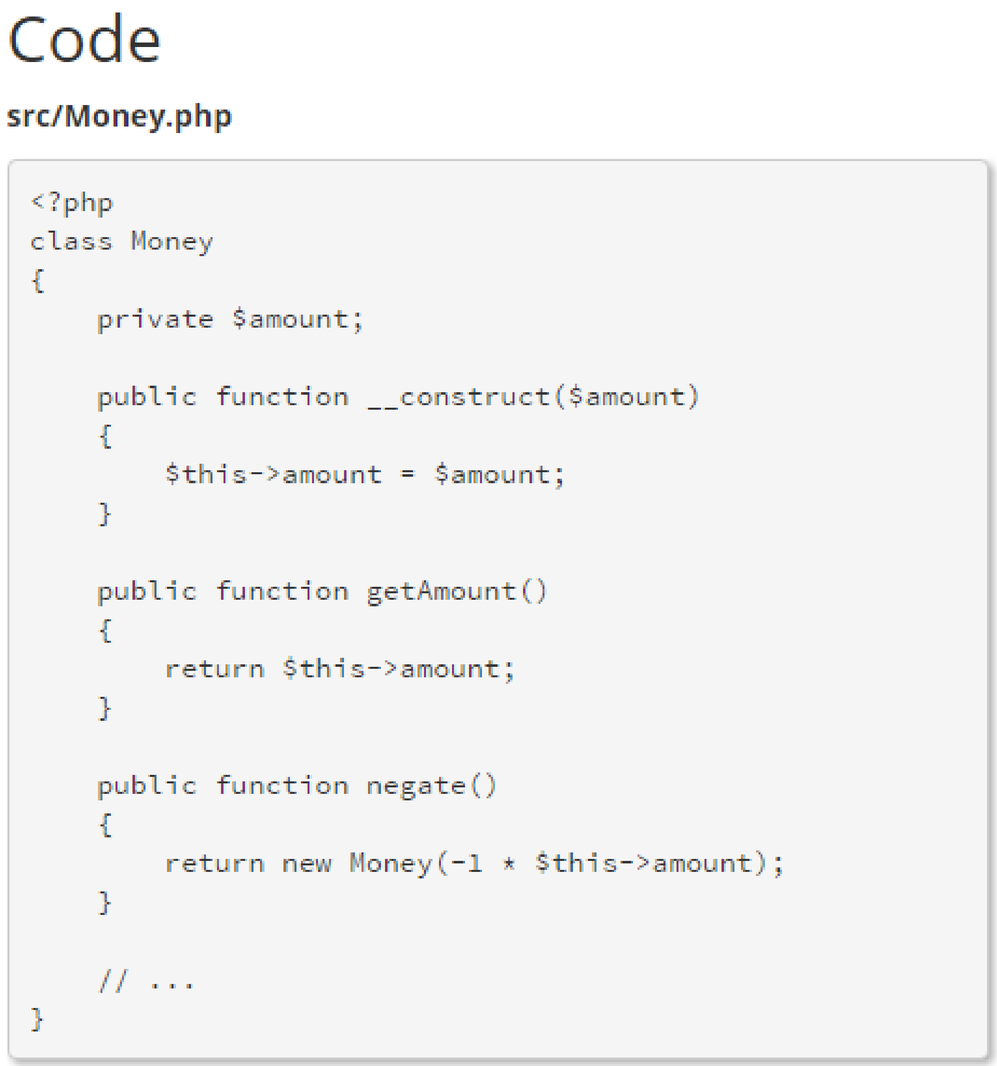
Introduction

- What are “tests”?
  - The same thing you do when you test your code while writing it, but:
    - Automated
    - Reproducible
    - Shared execution & maintenance by the whole team
- Why test?
  - Identify bugs when writing new code
  - Identify regressions when refactoring existing code
  - Enhance design of your code
  - Tests document functioning state of your code base

Caveats

- Investment:
  - Initial development of tests
  - Subsequent maintenance of tests
- Return of investment during project life cycle:
  - Less bugs & regressions = more time to write new features
  - Safety net for refactoring
- Not a dogma – consider postponing or completely skipping tests when:
  - Writing throw-away code that gets executed just once
  - Experimenting
  - Having insufficient understanding of your goal

Code example



Code

src/Money.php

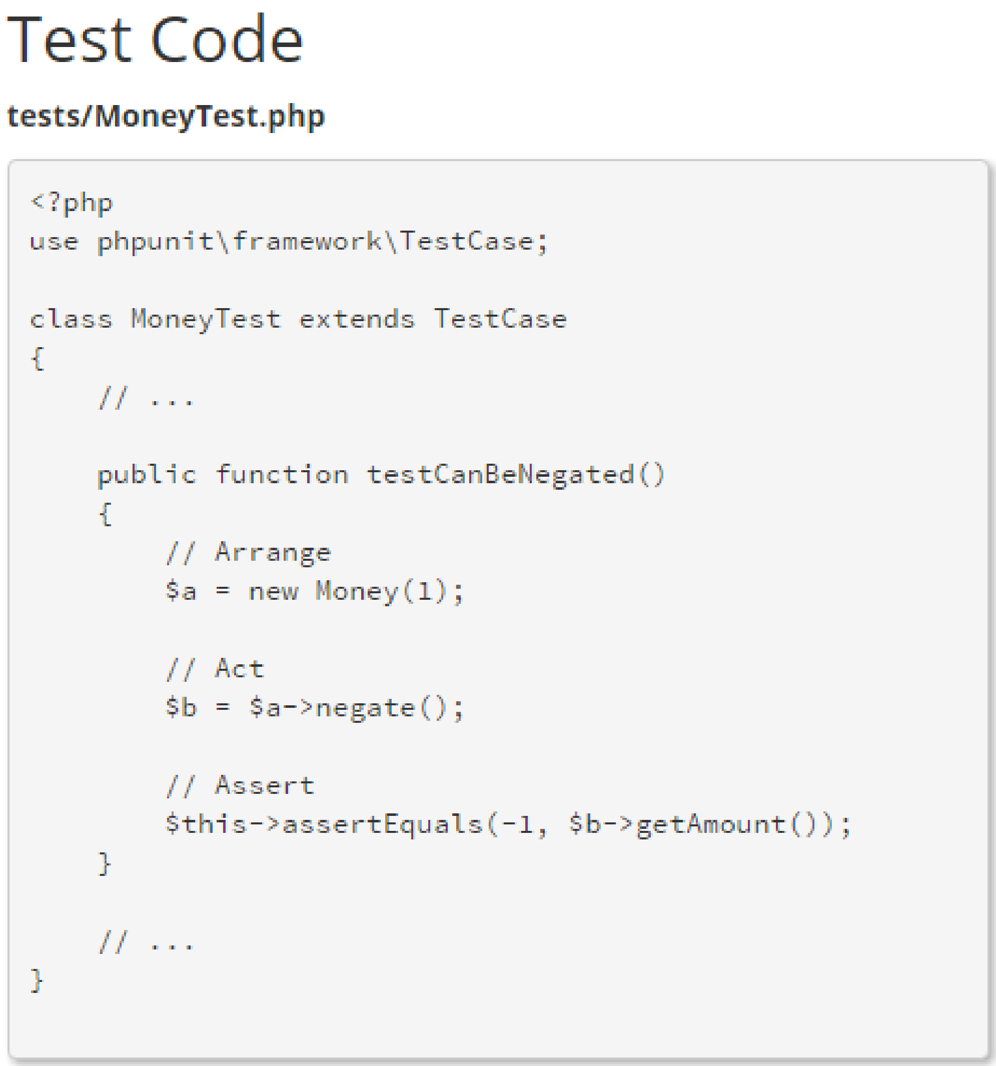
```php
<?php
class Money
{
    private $amount;

    public function __construct($amount)
    {
        $this->amount = $amount;
    }

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

    public function negate()
    {
        return new Money(-1 * $this->amount);
    }

    // ...
}
```



Test Code

tests/MoneyTest.php

```php
<?php
use phpunit\framework\TestCase;

class MoneyTest extends TestCase
{
    // ...

    public function testCanBeNegated()
    {
        // Arrange
        $a = new Money(1);

        // Act
        $b = $a->negate();

        // Assert
        $this->assertEquals(-1, $b->getAmount());
    }

    // ...
}
```

Source: phpunit.de

Terminology

- System under test = application that is being tested
- Unit = function or class method that is being tested
- Assert = comparison of actual vs. expected state

- Test doubles
  - Dummy = objects not used in tests, but required for tests to work
  - Stubs = “stupid” doubles (record call counts, but not whether they’re correct)
  - Mocks = “smarter” doubles with expectations (method calls, arguments, responses)
  - Fake = objects with working implementations, but not suitable for production
    - e.g. in memory database, child class for testing public methods of abstract class

- Fixtures = fixed data set against which tests are executed
- Test suite = sequence of tests, grouped by context or test type
- Code coverage = percentage of your code base that is covered by your test suites

Types of tests

- Unit tests:
  - The smallest amount of code you can test
- Integration tests
  - Test integration of two or more closely related components
- System tests
  - Test integration of all components, i.e. the system as a whole
- Other types of tests (out of scope of this talk):
  - Usability tests
  - Load tests, Stress tests, etc

Test life cycle

1. Setup – prepare resources for test
2. Test – execute the code under test
3. Assert – verify expectations
4. Tear down – clean up resources

Test Driven Development

- Practice of incrementally writing tests before actual code implementation
  - Define/amend interface
  - Write failing test
  - Write smallest amount of code that passes the test
  - Repeat

- Live demo

Do's & ...

- Make sure your tests are isolated and don’t influence each other.
- Test only one thing in one test case.
  - Consider a single assert per test case (but this is not practical in all scenarios).
- Use a naming convention for test cases.
  - E.g. CalculatorTest::testDiv_divisionByZero_throwsException
    1. Class and method under test
    2. State under test
    3. Expected outcome
- Received a bug report? Add a test case for it, kill the bug forever.
- Optimize & refactor your tests when needed. Tests are “code” too.

... Don'ts

- Slow tests are useless tests.
  - If unavoidable, have at least 1-2 test suites that cover core functionality that can be executed quickly during daily development.
  - Delegate the rest to your continuous integration server.
- Don’t mix types of tests - keep them in separate test suites that can be executed independently.
- Don’t test private and protected class methods. Test public ones only.
- Don’t test code with existing tests (external libs, etc). Be wary of test overlaps.
- Don’t test external dependencies
  - File system, database, etc
  - Slow execution, risk of data loss
  - Use stubs & mocks instead. Test APIs against specification (interfaces, actions & expected results).

Resources

- This presentation:
  - presentation, code examples: https://github.com/mice-sk/talks
  - a video re-recording may eventually appear at: http://mice.sk
- Other resources:
  - PHPUnit: Getting Started
  - Roy Osherove: Definition of a Unit Test
  - StackOverflow: Fakes, Stubs & Mocks

Questions
&
Answers

Thank you

Disclaimer