PHPUnit

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!

Intro

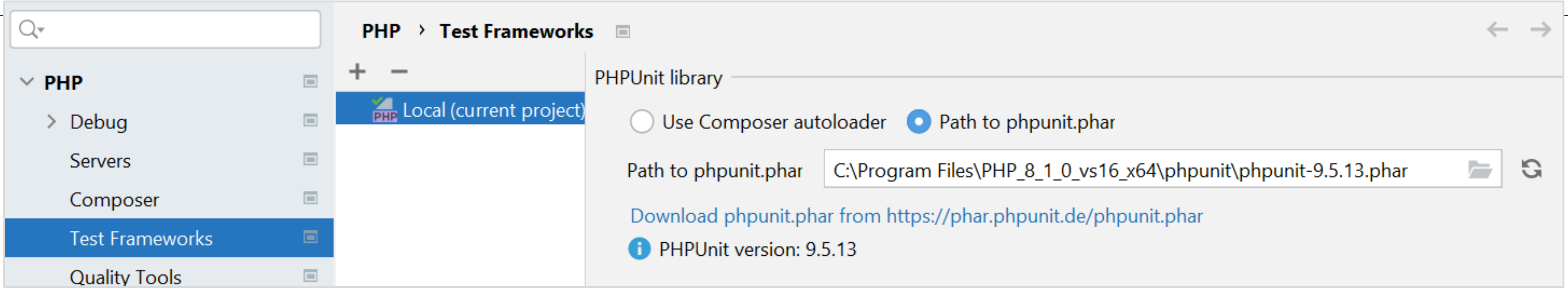
PHPUnit is a programmer-oriented testing framework for PHP. It is an instance of the xUnit architecture for unit testing frameworks.

Disclaimer! This course is not about automated testing (and unit testing in particular), so here we'll see just the main idea and usage samples.

Setup and configuration

The easiest way to start PHPUnit is to integrate it in PhpStorm, so:

1. Go to https://phpunit.de.
2. Download the latest version (phar file).
3. In PhpStorm press Ctrl+Alt+S.
4. Add local PHPUnit phar:



PHPUnit library

( ) Use Composer autoloader [✓ Path to phpunit.phar]

Path to phpunit.phar: C:\Program Files\PHP_8_1_0_vs16_x64\phpunit\phpunit-9.5.13.phar

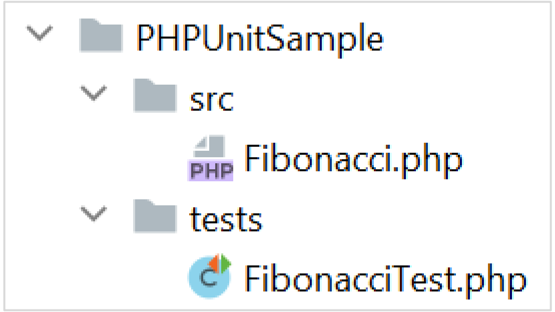
Download phpunit.phar from https://phar.phpunit.de/phpunit.phar

PHPUnit version: 9.5.13

Project structure

Usually two directories are required:

- src – for the code itself;
- tests – for tests.



```
PHPUnitSample
  src
    Fibonacci.php
  tests
    FibonacciTest.php
```

And see the sample in a couple of seconds…

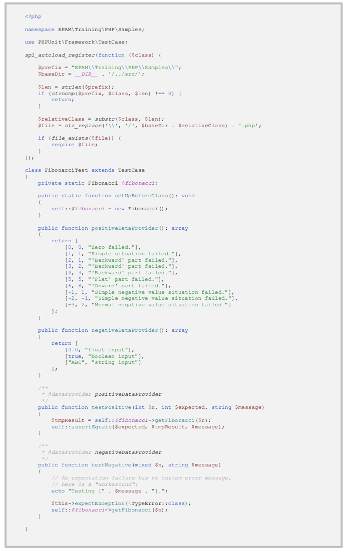
The code under test, just a simple class

Yes, PHPUnit is capable of much more, but the most common case is to test classes, methods, interactions and so on. So, here’s our class:

```php
<?php

namespace EPAM\Training\PHP\Samples;

use PHPUnit\Util\Test;

class Fibonacci
{

   // Yes, this code has errors :)

   public function getFibonacci(int $n): int
   {
      if ($n == 0) {
         return 0;
      } else
         if ($n == 1) {
            return 1;
         } else {
            return ($this->getFibonacci($n - 1) + $this->getFibonacci($n - 2));
         }
   }

}
```

The test code

Then we have to create a class with tests. For simplicity and convenience we'll use autoloading here:



```php
<?php

namespace EPAM\Training\PHP\Samples;

use PHPUnit\Framework\TestCase;

spl_autoload_register(function ($class) {

    $prefix = "EPAM\\Training\\PHP\\Samples\\";
    $baseDir = __DIR__ . '/../src/';

    $len = strlen($prefix);
    if (strncmp($prefix, $class, $len) !== 0) {
        return;
    }

    $relativeClass = substr($class, $len);
    $file = str_replace('\\', '/', $baseDir . $relativeClass) . '.php';

    if (file_exists($file)) {
        require $file;
    }
});

class FibonacciTest extends TestCase
{
    private static Fibonacci $fibonacci;

    public static function setUpBeforeClass(): void
    {
        self::$fibonacci = new Fibonacci();
    }

    public function positiveDataProvider(): array
    {
        return [
            [0, 0, "Zero failed."],
            [1, 1, "Simple situation failed."],
            [2, 1, "'Backward' part failed."],
            [3, 2, "'Backward' part failed."],
            [4, 3, "'Backward' part failed."],
            [5, 5, "'Flat' part failed."],
            [6, 8, "'Onward' part failed."],
            [-1, 1, "Simple negative value situation failed."],
            [-2, -1, "Simple negative value situation failed."],
            [-3, 2, "Normal negative value situation failed."]
        ];
    }

    public function negativeDataProvider(): array
    {
        return [
            [0.0, "float input"],
            [true, "boolean input"],
            ["ABC", "string input"]
        ];
    }

    /**
     * @dataProvider positiveDataProvider
     */
    public function testPositive(int $n, int $expected, string $message)
    {
        $tmpResult = self::$fibonacci->getFibonacci($n);
        self::assertEquals($expected, $tmpResult, $message);
    }

    /**
     * @dataProvider negativeDataProvider
     */
    public function testNegative(mixed $n, string $message)
    {
        // As expectation failure has no custom error message,
        // here is a "workaround":
        echo "Testing [" . $message . "].";

        $this->expectException(\TypeError::class);
        self::$fibonacci->getFibonacci($n);
    }

}
```

Let's see some important parts in details…

The test code: using data providers

Data providers are methods that provide data for multiple test runs.

```php
public function positiveDataProvider(): array
{
    return [
        [0, 0, "Zero failed."],
        [1, 1, "Simple situation failed."],
        [2, 1, "'Backward' part failed."],
        [3, 2, "'Backward' part failed."],
        [4, 3, "'Backward' part failed."],
        [5, 5, "'Flat' part failed."],
        [6, 8, "'Onward' part failed."],
        [-1, 1, "Simple negative value situation failed."],
        [-2, -1, "Simple negative value situation failed."],
        [-3, 2, "Normal negative value situation failed."]
    ];
}
```

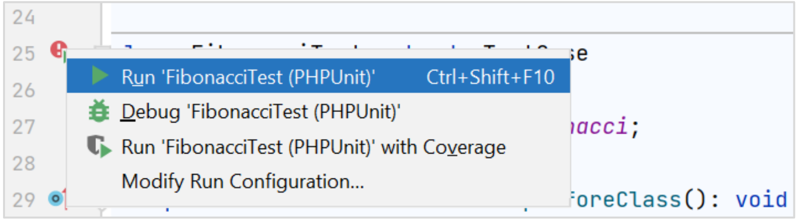
You may either "hardcode" data here, or use external sources (databases, files). Generally, each line of this array is a set of parameters for one test run.

```php
/**
 * @dataProvider positiveDataProvider
 */
public function testPositive(int $n, int $expected, string $message)
{
    $tmpResult = self::$fibonacci->getFibonacci($n);
    self::assertEquals($expected, $tmpResult, $message);
}
```

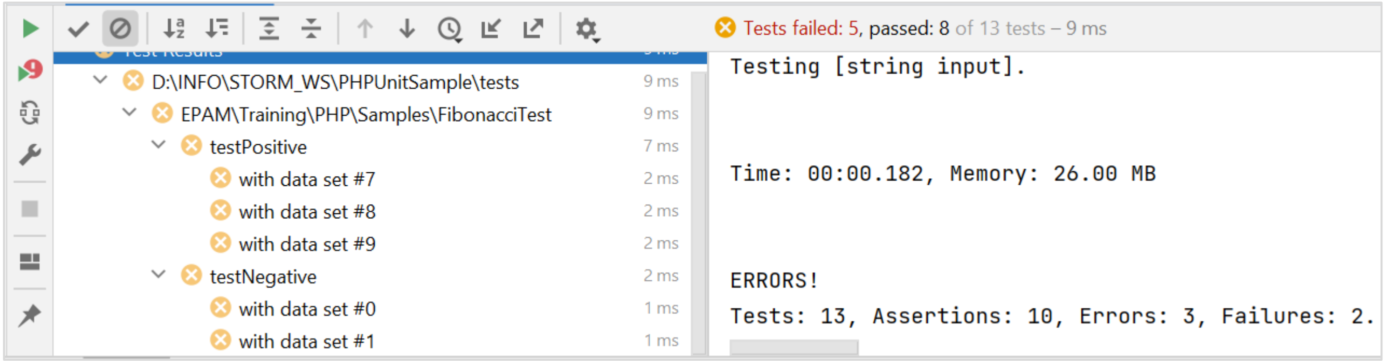
This test will be run 10 times.

Run tests

Now we only have to “run” the class with tests and see the result:


Run 'FibonacciTest (PHPUnit)' Ctrl+Shift+F10
Debug 'FibonacciTest (PHPUnit)'
Run 'FibonacciTest (PHPUnit)' with Coverage
Modify Run Configuration...

If our code has no errors, all test would have passed successfully.


Tests failed: 5, passed: 8 of 13 tests – 9 ms

```
D:\INFO\STORM_WS\PHPUnitSample\tests              9 ms
  EPAM\Training\PHP\Samples\FibonacciTest         9 ms
    testPositive                                  7 ms
      with data set #7                            2 ms
      with data set #8                            2 ms
      with data set #9                            2 ms
    testNegative                                  2 ms
      with data set #0                            1 ms
      with data set #1                            1 ms
```

```
Testing [string input].


Time: 00:00.182, Memory: 26.00 MB


ERRORS!
Tests: 13, Assertions: 10, Errors: 3, Failures: 2.
```

So, what’s the idea?

Unit tests…

- should always pass.
- should be separated from the application code.
- should be independent and simple.
- may be created by developers.
- may be created before code under test creation.

So, what’s the idea?

The idea is to create enough different tests that will analyze your code behavior in most likely (all possible?) conditions and situations.

Don’t wait till something happens to your code in production environment. Make it happen now with unit tests and be sure that your code behaves as expected.

So, what’s the idea?

Unit tests…

- improve architecture quality.
- encourage to write simple classes / methods.
- encourage changes in the code.
- simplify modules integration.
- help code documenting.
- minimize dependencies.
- are based on unified principles for many programming languages.

So, what’s the idea?

Unit tests test…

- methods (even void ones).
- classes.
- classes interaction.
- **complex** getters/setters.
- constructors.
- exceptions / errors.
- external dependencies.

And finally…

1. Read the documentation: https://phpunit.readthedocs.io
2. Read a good book: https://www.manning.com/books/unit-testing
3. Start using unit tests and see how your code improves.

PHPUnit

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!