Testing in PHP

Limit the Scope

- Several testing philosophies and tools are available
  - Unit testing, test-driven development - phpunit
  - Behavioral-driven development, specification by example - behat
  - Selenium, headless browsers, etc. available
  - XDebug extension needed for code-coverage reports
- Today we are doing unit testing with phpunit
- Documentation - https://phpunit.de/

The Plan

- Create an empty CakePHP project using composer
- Verify phpunit is installed
- Implement the Singleton design pattern
- Unit-test the implementation

The Singleton Design Pattern

What is a Singleton?

- The Singleton pattern ensures that one and only one instance of an object is available
- Often misused as a way of capturing global state - a huge source of errors and testing difficulty
- Widely used, especially inside Framework core code, for specific use cases

Singleton Use Cases

- Modeling an external resource such as a database table. A table may have many rows, but it's still a single table (or view or whatever) and we can represent it as such
- External connections such as for MySQL, RabbitMQ, or Elasticsearch
- Caching or memoization, sometimes with one Singleton per lookup key
- Immutable service with all configuration set up when object is constructed, and no saved state or state changes thereafter

Tricky to Test

- The Singleton holds on to its state from test to test to test - that's its purpose
- But each test assumes that we are starting "clean"
- I therefore build a "reset()" into every Singleton class to support unit testing of the class

The Fun Part

- We'll use new language knowledge in practice:
  - Private
  - Static
  - Docblock annotations
  - Self
  - Final
  - New
  - Default value for optional parameter

Create New Project

Create Project

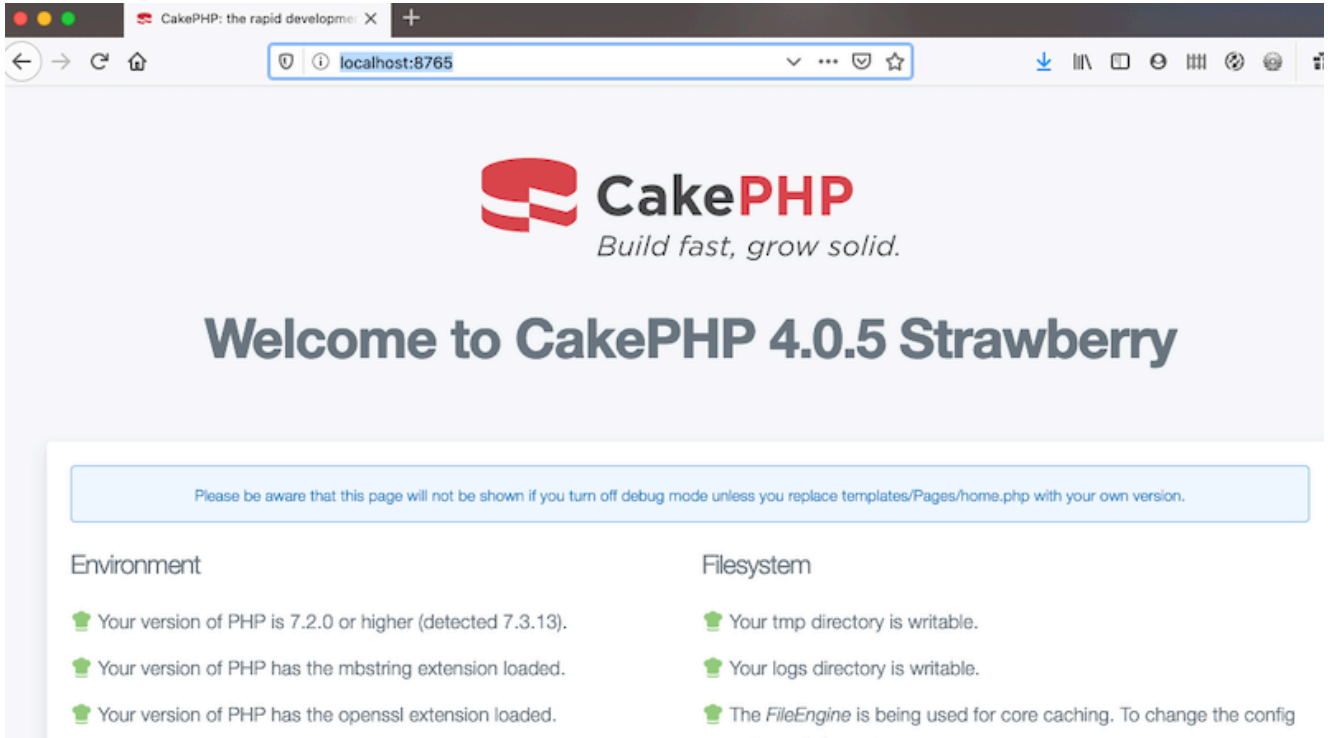
Installation instructions - https://book.cakephp.org/4/en/installation.html

composer create-project --prefer-dist cakephp/app:4.* design-pattern
cd design-pattern/
bin/cake server

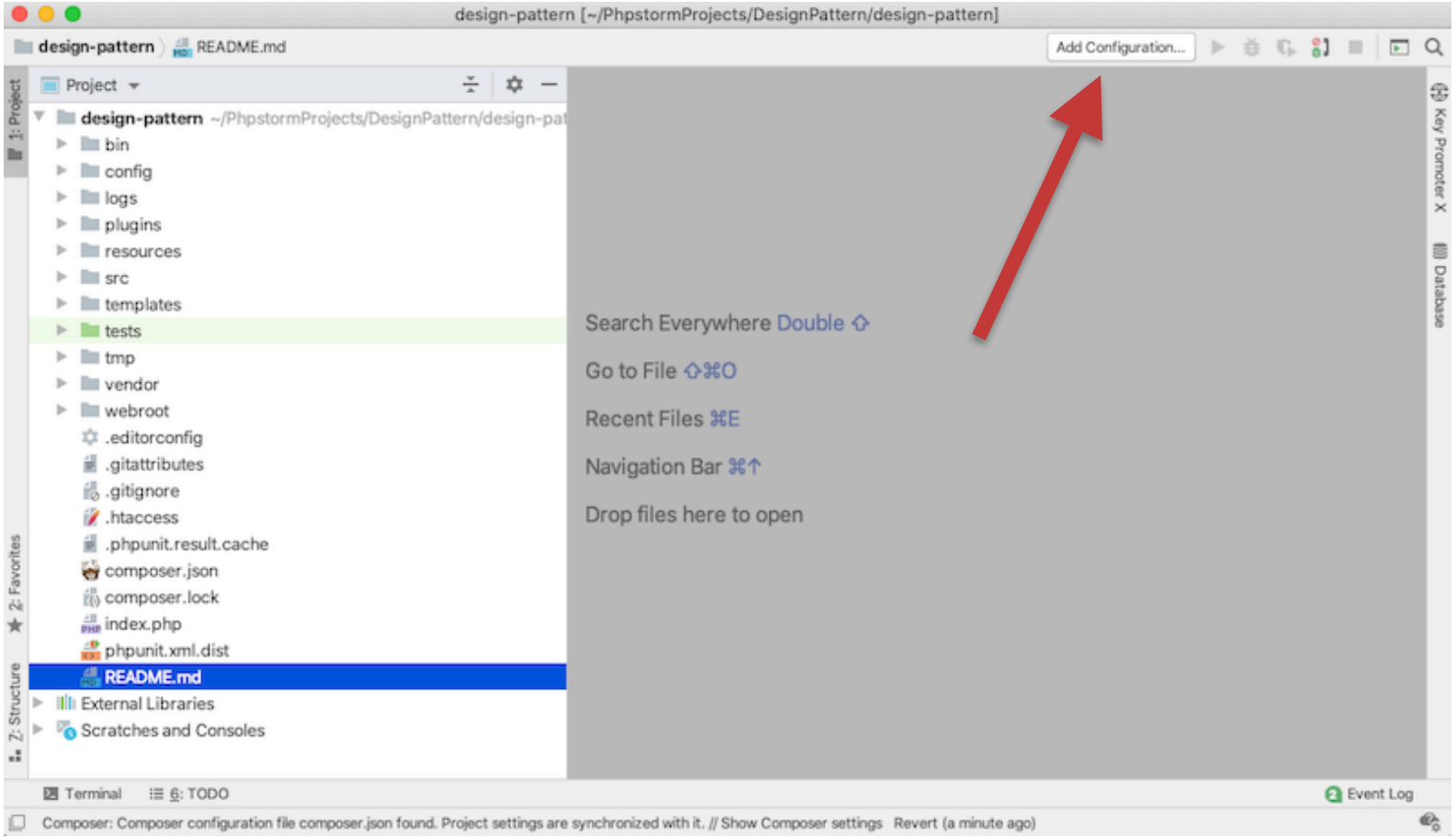
```
Edwards-MacBook-Pro:DesignPattern edwardbarnard$ composer create-project --prefer-dist cakephp/app:4.* design-pattern
Creating a "cakephp/app:4.*" project at "./design-pattern"
Installing cakephp/app (4.0.3)
  - Installing cakephp/app (4.0.3): Loading from cache
Created project in /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 83 installs, 0 updates, 0 removals
  - Installing cakephp/plugin-installer (1.2.0): Loading from cache
  - Installing m1/env (2.2.0): Loading from cache
  - Installing josegonzalez/dotenv (3.2.0): Loading from cache
  - Installing psr/simple-cache (1.0.1): Loading from cache
  - Installing psr/log (1.1.3): Loading from cache
```

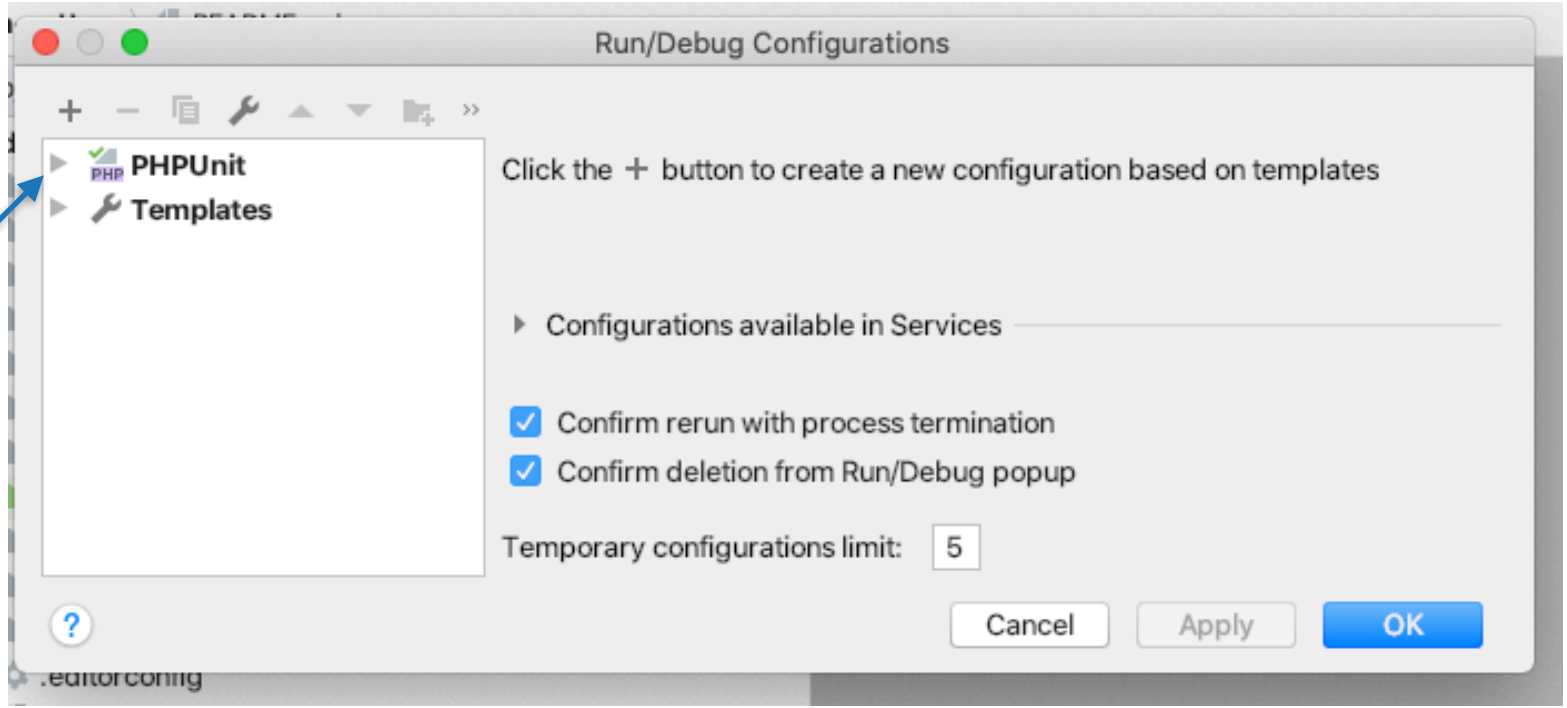
View Splash Page

- http://localhost:8765/ - comes from running bin/cake server



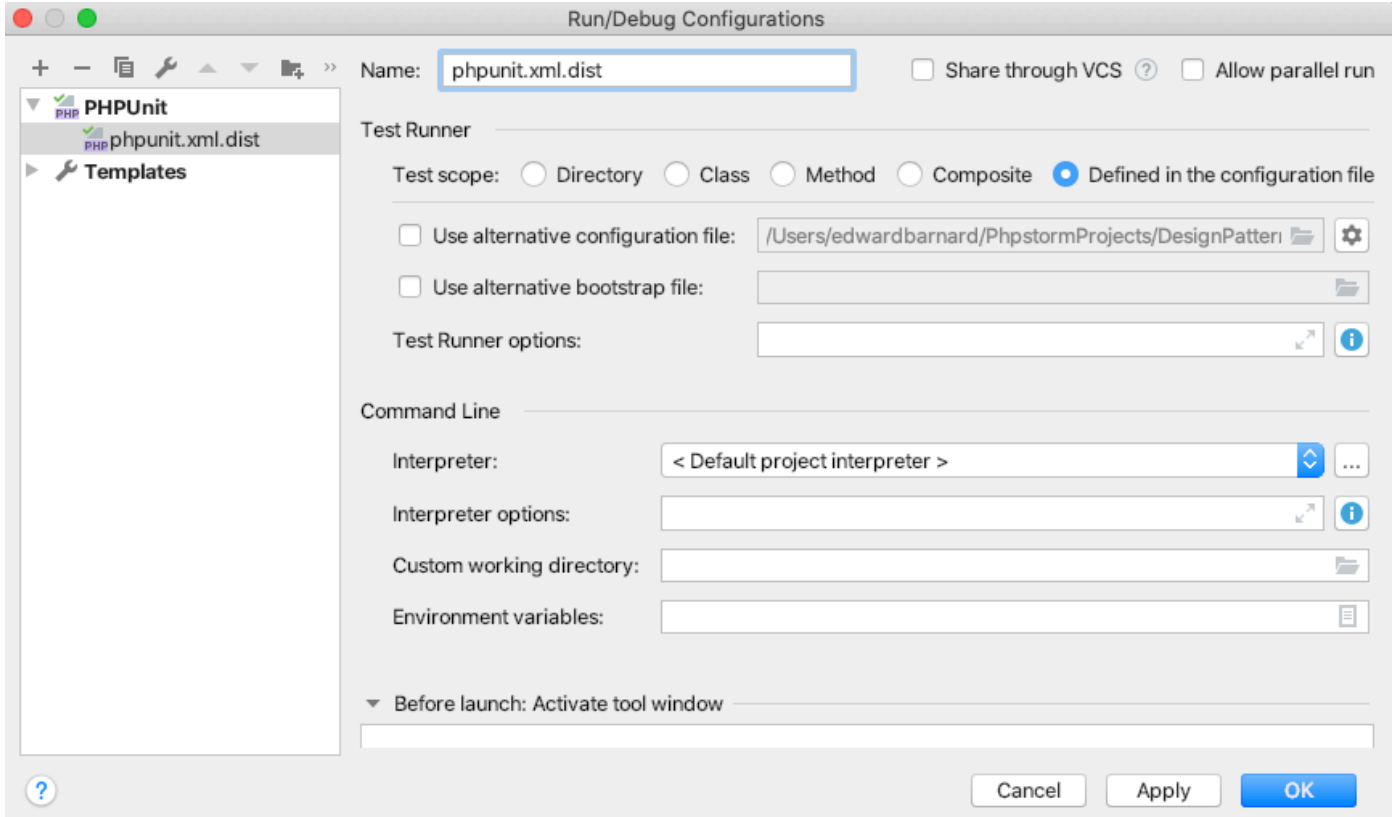
PhpStorm Project From Existing Files




Run/Debug Configurations
PHPUnit
Templates
Click the + button to create a new configuration based on templates
Configurations available in Services
Confirm rerun with process termination
Confirm deletion from Run/Debug popup
Temporary configurations limit:
5
Cancel
Apply
OK
.editorconfig

Set Configuration

- Use PhpStorm online instructions, may need to point to your installed PHP



Run PHPUnit

Click to run PHPUnit

Write Unit Tests

```php
<?php
declare(strict_types=1);

namespace App\DesignPattern;

class Singleton
{
    /** @var Singleton */
    private static $instance;
    /** @var array */
    private $data;

    final private function __construct(array $properties = [])
    {
        $this->data = $properties;
    }

    final public static function instance(array $properties = []): self
    {
        if (!self::$instance) {
            self::$instance = new self($properties);
        }

        return self::$instance;
    }

```

```
26
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
35
36      public static function reset(): void
37      {
38          self::$instance = null;
39      }
40  }
```

First Test - Make it fail

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\DesignPattern;

use PHPUnit\Framework\TestCase;

class SingletonTest extends TestCase
{
    public function testShouldFail(): void
    {
        self::fail(__FUNCTION__);
    }
}
```

First Test Fails

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\DesignPattern;

use PHPUnit\Framework\TestCase;

class SingletonTest extends TestCase
{
    public function testShouldFail(): void
    {
        self::fail(__FUNCTION__);
    }
}
```

Run: phpunit.xml.dist

Tests failed: 1, passed: 10 of 11 tests – 115 ms

```
Testing started at 5:52 PM ...
/usr/local/opt/php@7.3/bin/php /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/pl
PHPUnit 8.5.3 by Sebastian Bergmann and contributors.


testShouldFail
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/DesignPattern/SingletonTest.php:12



Time: 176 ms, Memory: 14.00 MB

FAILURES!
Tests: 11, Assertions: 26, Failures: 1.
Process finished with exit code 1
```

Writing Failing Test (assertNotSame)

```
9   class SingletonTest extends TestCase
10  {
11      public function testInstanceReturnsSame(): void
12      {
13          $one = Singleton::instance();
14          $two = Singleton::instance();
15          self::assertNotSame($one, $two, message: 'Instances should be same');
16      }
17  }
```

It fails

- With the failing test, we are verifying that the test is in fact being run (stuff happens - often)
- We are verifying that it is able to detect a failing condition

```
Instances should be same
Failed asserting that two variables don't reference the same object.
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/te



Time: 182 ms, Memory: 14.00 MB

FAILURES!
Tests: 11, Assertions: 26, Failures: 1.
Process finished with exit code 1
```

Passing Test (assertSame)

```php
class SingletonTest extends TestCase
{
    public function testInstanceReturnsSame(): void
    {
        $one = Singleton::instance();
        $two = Singleton::instance();
        self::assertSame($one, $two, message: 'Instances should be same');
    }
}
```

```
Tests passed: 11 of 11 tests – 107 ms
Testing started at 6:00 PM ...
/usr/local/opt/php@7.3/bin/php /Users/e
PHPUnit 8.5.3 by Sebastian Bergmann and

Time: 163 ms, Memory: 14.00 MB

OK (11 tests, 26 assertions)
Process finished with exit code 0
```

Testing reset()

```
18    final public static function instance(array $properties = []): self
19    {
20        if (!self::$instance) {
21            self::$instance = new self($properties);
22        }
23
24        return self::$instance;
25    }
26
27    public function get(string $field){...}
35
36    public static function reset(): void
37    {
38        self::$instance = null;
39    }
```

Testing reset()

```php
11  public function testInstanceReturnsSame(): void
12  {
13      $one = Singleton::instance();
14      $two = Singleton::instance();
15      self::assertSame($one, $two, message: 'Instances should be same');
16  }
17
18  public function testResetChangesInstance(): void
19  {
20      $one = Singleton::instance();
21
22      Singleton::reset();
23
24      $two = Singleton::instance();
25      self::assertSame($one, $two, message: 'This should fail');
26  }
```

Tests failed: 1, passed: 11 of 12 tests – 109 ms

```
Testing started at 6:07 PM ...
/usr/local/opt/php@7.3/bin/php /Users/ed
PHPUnit 8.5.3 by Sebastian Bergmann and


This should fail
Failed asserting that two variables refe
 /Users/edwardbarnard/PhpstormProjects/D



Time: 179 ms, Memory: 14.00 MB

FAILURES!
Tests: 12, Assertions: 27, Failures: 1.
```

Testing reset()

```php
18    public function testResetChangesInstance(): void
19    {
20        $one = Singleton::instance();
21
22        Singleton::reset();
23
24        $two = Singleton::instance();
25        self::assertNotSame($one, $two, message: 'Reset should change instance');
26    }
```

```
Tests passed: 12 of 12 tests – 103 ms

Testing started at 6:10 PM ...
/usr/local/opt/php@7.3/bin/php /Users/
PHPUnit 8.5.3 by Sebastian Bergmann an



Time: 161 ms, Memory: 14.00 MB

OK (12 tests, 27 assertions)
Process finished with exit code 0
```

Test Suite Setup

```
9   class SingletonTest extends TestCase
10  {
11      protected function setUp(): void
12      {
13          Singleton::reset();
14      }
15
16      public function testInstanceReturnsSame(): void
17      {
18          $one = Singleton::instance();
19          $two = Singleton::instance();
20          self::assertSame($one, $two,  message: 'Instances
```

```
» ✔ Tests passed: 12 of 12 tests – 98 ms

Testing started at 6:13 PM ...
/usr/local/opt/php@7.3/bin/php /Users/
PHPUnit 8.5.3 by Sebastian Bergmann an

Time: 153 ms, Memory: 14.00 MB

OK (12 tests, 27 assertions)
Process finished with exit code 0
```

Tests run the same - but now
setUp() executed before each test

How to test get() ?

- Test no properties
- Test one property
- Test two properties
- Test wrong property
- Test second instance

```
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
```

Test No Properties

```
9   class SingletonTest extends TestCase
10  {
11      private const PROPERTIES = ['a' => 3, 'b' => 'extract'];
12
```

```
35      public function testNoProperties(): void
36      {
37          $target = Singleton::instance();
38
39          $target->get('bogus');
40
41          static::fail('Should throw exception');
42      }
```

```
InvalidArgumentException : No field bogus
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/src/DesignPattern/Singleton.php:30
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/DesignPattern/SingletonTest.php:39
```

Test No Properties

```
27    public function get(string $field)
28    {
29        if (!array_key_exists($field, $this->data)) {
30            throw new \InvalidArgumentException( message: "No field $field");
31        }
32
33        return $this->data[$field];
34    }
```

```
InvalidArgumentException : No field bogus
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/src/DesignPattern/Singleton.php:30
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/DesignPattern/SingletonTest.php:39
```

Tell PHPUnit to EXPECT the Exception to Happen

```
6   use App\DesignPattern\Singleton;
7   use InvalidArgumentException;
8   use PHPUnit\Framework\TestCase;
```

```
36      public function testNoProperties(): void
37      {
38          $this->expectException(InvalidArgumentException::class);
39          $target = Singleton::instance();
40
41          $target->get('bogus');
42
43          static::fail('Should throw exception');
44      }
```

```
» ✔ Tests passed: 13 of 13 tests – 167 ms
Testing started at 6:45 PM ...
/usr/local/opt/php@7.3/bin/php /Users
PHPUnit 8.5.3 by Sebastian Bergmann a



Time: 266 ms, Memory: 14.00 MB

OK (13 tests, 28 assertions)
Process finished with exit code 0
```

How to test get() ?

- ~~Test no properties~~
- Test one property
- Test two properties
- Test wrong property
- Test second instance

```
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
```

Test One Property

```
9   class SingletonTest extends TestCase
10  {
11      private const PROPERTIES = ['a' => 3, 'b' => 'extract'];
12
```

```
46      public function testOneProperty(): void
47      {
48          $target = Singleton::instance( properties: self::PROPERTIES);
49
50          $actual = $target->get('a');
51
52          self::assertSame(self::PROPERTIES['a'], $actual);
53      }
```

```
Tests passed: 13 of 13 tests – 167 ms
Testing started at 6:45 PM ...
/usr/local/opt/php@7.3/bin/php /Users
PHPUnit 8.5.3 by Sebastian Bergmann a

Time: 266 ms, Memory: 14.00 MB

OK (13 tests, 28 assertions)
Process finished with exit code 0
```

```
Tests passed: 14 of 14 tests – 170 ms
Testing started at 6:52 PM ...
/usr/local/opt/php@7.3/bin/php /User
PHPUnit 8.5.3 by Sebastian Bergmann

Time: 273 ms, Memory: 14.00 MB

OK (14 tests, 29 assertions)
Process finished with exit code 0
```

Red - Green - Refactor

1. Create a failing test. Test runner shows **red**
2. Write enough code to make the test pass. Test runner shows **green**
3. **Refactor** production code and/or tests to better express design intent

- Continue rapid red-green-refactor cycle as you develop code
- Only refactor when tests are running green (as they are now)

Observation - leading to refactoring opportunity

- We are in the midst of zero, one, two, parameters sequence of tests
- The test setup
  `$target = Singleton::instance(self::PROPERTIES);`
  is likely the same for each test in the sequence
- One improvement is to move tests with the same setup into the same class - when the setup needs change, start a new class
- Create SingletonPropertyTest class
- 14 tests should still pass (run green)

Extracted Test Class

Tests passed: 14 of 14 tests – 102 ms

```
Testing started at 7:39 PM ...
/usr/local/opt/php@7.3/bin/php /Us
PHPUnit 8.5.3 by Sebastian Bergman



Time: 163 ms, Memory: 14.00 MB

OK (14 tests, 29 assertions)
Process finished with exit code 0
```

SingletonTest.php | SingletonPropertyTest.php

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\DesignPattern;

use App\DesignPattern\Singleton;
use PHPUnit\Framework\TestCase;

class SingletonPropertyTest extends TestCase
{
    private const PROPERTIES = ['a' => 3, 'b' => 'extract'];

    /** @var Singleton */
    private $target;

    protected function setUp(): void
    {
        Singleton::reset();
        $this->target = Singleton::instance( properties: self::PROPERTIES);
    }

    public function testOneProperty(): void
    {
        $actual = $this->target->get('a');

        self::assertSame(self::PROPERTIES['a'], $actual);
    }
}
```

How to test get() ?

- ~~Test no properties~~
- ~~Test one property~~
- Test two properties
- Test wrong property
- Test second instance

```
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
```

Test Two Properties

```php
public function testTwoProperties(): void
{
    $one = $this->target->get('a');
    $two = $this->target->get('b');

    self::assertSame(self::PROPERTIES['a'], $one);
    self::assertSame(self::PROPERTIES['b'], $two);
}
```

```
Time: 161 ms, Memory: 14.00 MB

OK (15 tests, 31 assertions)
Process finished with exit code 0
```

How to test get() ?

- ~~Test no properties~~
- ~~Test one property~~
- ~~Test two properties~~
- Test wrong property
- Test second instance

```
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
```

Test Wrong Property (expect exception)

```php
39    public function testWrongProperty(): void
40    {
41        $this->expectException(InvalidArgumentException::class);
42
43        $this->target->get('bogus');
44
45        self::fail('Should see exception');
46    }
```

```
Time: 156 ms, Memory: 14.00 MB

OK (16 tests, 32 assertions)
Process finished with exit code 0
```

How to test get() ?

- ~~Test no properties~~
- ~~Test one property~~
- ~~Test two properties~~
- ~~Test wrong property~~
- Test second instance

```
27      public function get(string $field)
28      {
29          if (!array_key_exists($field, $this->data)) {
30              throw new \InvalidArgumentException( message: "No field $field");
31          }
32
33          return $this->data[$field];
34      }
```

Test Second Instance Returns Same Content

```php
48    public function testSecondInstance(): void
49    {
50        $instance = Singleton::instance();
51
52        $actual = $instance->get('a');
53
54        self::assertSame(self::PROPERTIES['a'], $actual);
55    }
```

```
Time: 159 ms, Memory: 14.00 MB

OK (17 tests, 33 assertions)
Process finished with exit code 0
```

Data Provider

- PHPUnit's data provider provides a way of feeding several different use cases to the same unit test
- The data provider can be a Generator
- But we'll use arrays rather than Generator
- Example: Test list of properties

Example Data Provider

```
Time: 163 ms, Memory: 14.00 MB

OK (19 tests, 35 assertions)
Process finished with exit code
```

```php
    public function dataFields(): array
    {
        return [
            ['a'],
            ['b'],
        ];
    }

    /**
     * @param string $field
     * @return void
     * @dataProvider dataFields
     */
    public function testProvider(string $field): void
    {
        self::assertSame(self::PROPERTIES[$field], $this->target->get($field));
    }
```

Easy! When does it get difficult?

- If you're trying to add unit tests to an existing (legacy) code base, it won't go well
- That's because the code was never designed to be testable - it was written without unit tests in mind
- Outside dependencies are the other killer
  - Database
  - Third-party services or APIs
  - Framework request/response structures
  - Etc

The hard stuff is out of scope! But tell me more!

- PHPUnit - phpunit.de and https://phpunit.readthedocs.io/en/9.0/index.html
- Dependency-mocking library Mockery - http://docs.mockery.io/en/latest/
- Laracasts and Symfonycasts
- Testing documentation for your framework of choice

Conclusion - Why unit test?

- The tests are like a vice, holding the rest of your code in place as you refactor
- If your tests are green before you refactor, and return to green after you refactor, you probably didn't break anything in the process - assuming you have complete enough test coverage
- When you build tests as you code, your production code becomes FAR more clean than otherwise