Resumen rápido

* EL nombre de las clases tiene que terminar por test.
* Tiene que extender de la clase PHPUnit\_Framework\_TestCase.

class pruebaTest extends \PHPUnit\_Framework\_TestCase

Las funciones tienen que que empezar por test

Public function testPrueba

Ejecutar test

vendor/bin/phpunit

# Métodos auxiliares.

Métodos que nos ayudan a trabajar más cómodos.

### **Fixtures**

Cuando se escribe un test se necesita empezar en un estado conocido. Este estado es la fixture del test.

Se suele hacer en el **seup(){…},** se ejecuta antes de que empiecen los tests. Sirve para inicializar objetos que utilizaremos una y otra vez a lo largo del tests.

Esta función puede coger datos de diferentes fuentes de datos como archivos csv.

return new csvFileIterator(‘’);

**teardow(){…}**

Se ejecuta al finalizar los tests y se utiliza para desmontar cualquier estructura realizada en los tests (destructor).

**@ dataProvider (Proveedor de datos).**

Se crea una función con los datos que se quieren probar y después de se añade en los comentarios el dataProvider.

Con los @dataProviders puedes ejecutar varias veces un test con distintos argumentos.

\* @dataProvider <Nombre de la función a llamar>

Public function testX($variable) {…}

public function provideUsersSiSoportados(){

return [

'Correcto 5' => ['Carlo'],

'Correcto 10' => ['SamuelCast'],

'Correcto 6' => ['scasdo'],

'Correcto 9' => ['SamuelCas']

];

}

/\*\*

\* **@dataProvider provideUsersSiSoportados**

\*/

public function test\_User($nombre){

$user = new User();

$reultado = $user->setUsuario($nombre);

$this->assertTrue($reultado instanceof UserInterface);

}

**@ depends (Dependencias).**

Los @depends se suelen usar para proveer de argumentos a otros tests. Si falla el test del cual dependen los demás, estos no se ejecutan.

\* @depends <Nombre de la función de la que depende>

# Aseveraciones

assertions ($expected, );

assertArrayHasKey, assertClassHasAttribute, 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.

Intentar utilizar siempre la mejor asación posible, estos no ayudara a comprender mejor cuando falla un test.

Que un array contenga cierto valor

assertTrue(ii\_array(1, array(1,2,3))); MAL

assertContains(1, , array(1,2,3)); BIEN

De cuantos elementos se compone un array

assertEquals(1, count(array(1)))); MAL

assertCount(3, array(1,2,3)); BIEN

Para saber si un objeto es de un tipo de instancia

assertTrue($myObjeto instanceof MyClase); MAL

assertInstanceOf($myObjeto, ‘MyClase’); BIEN

# Ejemplos

public $test;  
public function setUp(){  
 $this->test = new Calculator;  
}  
public function provider(){  
  
 return array(  
 array(20,15,5),  
 array(45,20,25)  
 );  
  
}  
public function dataSuma(){  
 return array(  
 array(75,array(10,15,20,30)),  
 array(113,array(15,10,25,47,16)),  
 array(19,array(10,-25,30,4))  
 );  
}

/\*\*  
 \* @dataProvider dataSuma  
 \*/  
public function testSuma($result, $arreglo){  
 $this->assertEquals($result, $this->test->suma($arreglo));  
}

/\*\*  
 \* @dataProvider provider  
 \*/  
public function testSumaSimple($result, $val1, $val2){  
  
 $this->assertEquals($result,

$this->test->sumaSimple($val1,$val2));  
  
}

/\*\*  
 \* assertArrayHasKey Verifica si existe un key con el

\* nombre indicado.  
 \*/  
public function testFailure(){  
  
 $this->assertArrayHasKey('hola',  
 array(  
 'hola'=> 'texto',  
 'otro' => 'mas Texto')  
 );  
}

/\*\*  
 \* assertClassHasAttribute() Verifica si existe un atributo en

\* la clase assertClassHasAttribute(string $attributeName,

\* string $className)  
 \*/  
public function testClassAttribute(){  
  
 $this->assertClassHasAttribute('operacion', 'Calculator');  
}

/\*\*  
 \* assertClassHasStaticAttribute verifica si existe un

\* atributo 'static' dentro de la clase  
 \*/  
public function testClassAttributeStatic(){  
 $this->assertClassHasStaticAttribute('saldo',

'Calculator');  
}

/\*\*  
 \* assertContains verifica si existe el valor en un array  
 \* assertContains(value, array)  
 \*/  
public function testContains(){  
 $this->assertContains(10, array(5,10,24,65));  
 $this->assertContains('hola', array('como',

'estas',

'tu',

'hola',

'quien'));  
}

/\*\*  
 \* assertNotContains verifica si no existe el valor en un

\* array  
 \*/  
public function testNoContains(){  
 $this->assertNotContains(12, array(5,10,24,65));  
}

/\*\*  
 \* assertContainsOnly verifica que solo exista el tipo de dato   
 \* definido  
 \*/  
public function testContainsOnly(){  
 $this->assertContainsOnly('string', array('1','2','3'));  
 $this->assertContainsOnly('int',array(1,2,3,4));  
}

/\*\*  
 \* assertNotContainsOnly verifica que no solo exista el tipo

\* de dato definido.  
 \*/  
public function testNotContainsOnly(){  
  
 $this->assertNotContainsOnly('string',

array(1,'2','3',4));  
}

/\*\*  
 \* assertCount verifica el número de registros del array del

\* número definido  
 \*/  
public function testCount(){  
 $this->assertCount(3,array(1,2,3));  
}

/\*\*  
 \* assertEmpty verifica si el array esta vacio  
 \*/  
public function testEmpty(){  
 $this->assertEmpty(array());  
}

/\*\*  
 \* assertEquals verifica que sea igual el resultado con el

\* valor definido  
 \*/  
public function testEquals(){  
 $this->assertEquals(10, 5+5);  
  
 //ejemplo con DOMDocument  
 $expected = new DOMDocument;  
 $expected->loadXML('<div><label>Hola</label></div>');  
 $actual = new DOMDocument;  
 $actual->loadXML('<div><label>Hola</label></div>');  
 $this->assertEquals($expected,$actual);  
}  
  
/\*\*  
 \* assertNotEquals verifica que no sea igual el resultado con

\* el valor definido  
 \*/  
public function testNotEquals(){  
 $this->assertNotEquals(20, (10+15));  
}

/\*\*  
 \* assertFileEquals verifica que sean iguales los archivos  
 \*/  
public function testFileEquals(){  
 $this->assertFileEquals

('/var/www/phpunit/Calculator.php',

'/var/www/phpunit/Calculator.php');  
}

/\*\*  
 \* assertFileExists verifica si existe el archivo  
 \*/  
public function testFileExists(){  
 $this->assertFileExists(

'/var/www/phpunit/Calculator.php');  
}

/\*\*  
 \* assertFalse verifica que sea falso el resultado  
 \*/  
public function testFalse(){  
 $this->assertFalse(FALSE, $message = 'es verdadero :(');  
}

**Covertura de los test**

phpunit -c ./ --coverage-html ./tests/cobertura/

phpunit --coverage-html ./web/coverage ./tests/AppBundle/

Si analizamos el comando, a phpunit le hemos pasado --coverage-html como opción, el primer parámetro es la ruta donde queremos que creé la interfaz, y el segundo parámetro sería la parte de nuestra aplicación que vamos a testear, en este caso crearemos el directorio coverage dentro de la parte pública de symfony, para poder acceder desde el navegador.

<http://www.mercenariophp.com/PHPUnit-skeleton-generator-netbeans/>

<logging>  
 <log type="coverage-html" target="../../build/phpUnit"  
 lowUpperBound="35" highLowerBound="70"/>  
 <log type="coverage-clover" target="../../build/phpunit-coverage.xml" />  
 <log type="junit" target="../../../build/phpunit.xml" />  
 </logging>