Table des matières

Installation

Sur une distribution Ubuntu/Debian, il faut réaliser les actions suivantes :

```
$ sudo pear config-set auto_discover 1
```

```
$ sudo pear install phpunit/PHPUnit
```

```
$ sudo pear install phpunit/PHPUnit_SkeletonGenerator
```

Remarque : il vous préalablement « passer » sur un compte « administrateur » en faisant **`su iris`** (ou `sudo -s`).

Liens

La documentation en français est disponible en ligne à partir de ce lien :

http://phpunit.de/manual/3.7/fr/

On peut aussi conseiller la lecture de ces différents articles :

- http://jp-grossglauser.developpez.com/tutoriels/langages/php/phpunit/
- http://www.unixgarden.com/index.php/gnu-linux-magazine-hs/phpunit-tests-unitaires-pour-php

Objectif

Ce document décrit une **mise en œuvre simple d'une procédure de tests unitaires en utilisant le *framework* PhpUnit**.

Conventions

Nous utiliserons les conventions suivantes pour les noms de fichiers :

- les classes seront suffixées par **`*.class.php`**
- les classes de test seront suffixées par **`Test*.class.php`**

Exemple n°1 : La classe CompteBancaire

Cet exemple illustre une démarche pour écrire des tests unitaires **après** avoir écrit du code source pour une application, ici la classe `CompteBancaire`.

Cette classe est fournie à titre d'exemple dans le manuel (exemple 12.3) de PhpUnit :

http://phpunit.de/manual/3.7/fr/test-driven-development.html#test-driven-development.bankaccount-example.examples.BankAccount2.php

On va préalablement créer deux répertoires afin de séparer le code source de l'application (`src/`) du code source des tests unitaires (`tests/`) :

```
$ mkdir /var/www/compte-bancaire/src
$ mkdir /var/www/compte-bancaire/tests
```

Puis :

```
$ cd /var/www/compte-bancaire/src
```

On crée le fichier (vide) pour la classe CompteBancaire :

```
$ touch CompteBancaire.class.php
```

On édite ensuite le fichier :

```
<?php
class CompteBancaireException extends Exception { }

class CompteBancaire
{
    protected $balance = 0;

    public function getBalance()
    {
        return $this->balance;
    }

    protected function setBalance($balance)
    {
        if ($balance >= 0)
        {
            $this->balance = $balance;
        }
        else
        {
            throw new CompteBancaireException;
        }
    }

    public function deposerArgent($balance)
    {
        $this->setBalance($this->getBalance() + $balance);
        return $this->getBalance();
    }

    public function retirerArgent($balance)
    {
        $this->setBalance($this->getBalance() - $balance);
        return $this->getBalance();
    }
}
?>
```

Maintenant on se place dans le répertoire de test :

```
$ cd /var/www/compte-bancaire/tests
```

PhpUnit fournit un moyen de **générer un squelette de classe de cas de test** :

http://phpunit.de/manual/3.7/fr/skeleton-generator.html#skeleton-generator.test

L'usage de cette commande est le suivant :

```
$ phpunit-skelgen --test -- Class [Class.php] [ClassTest]
[ClassTest.php]

--test : Generate ClassTest [in ClassTest.php] based on Class [in
Class.php]
```

Donc pour générer notre classe de test, on fera :

```
$ phpunit-skelgen --test -- CompteBancaire
../src/CompteBancaire.class.php TestCompteBancaire
./TestCompteBancaire.class.php
```

On peut donc déjà « tester notre classe » (même si on n'a encore implémenté aucun test !) avec la commande suivante :

```
$ phpunit --bootstrap ../src/CompteBancaire.class.php --verbose
TestCompteBancaire TestCompteBancaire.class.php
PHPUnit 3.7.28 by Sebastian Bergmann.

III

Time: 24 ms, Memory: 4.25Mb

There were 3 incomplete tests:

1) TestCompteBancaire::testGetBalance
This test has not been implemented yet.

/var/www/compte-bancaire/tests/TestCompteBancaire.class.php:38

2) TestCompteBancaire::testDeposerArgent
This test has not been implemented yet.

/var/www/compte-bancaire/tests/TestCompteBancaire.class.php:50

3) TestCompteBancaire::testRetirerArgent
This test has not been implemented yet.

/var/www/compte-bancaire/tests/TestCompteBancaire.class.php:62
```

```
OK, but incomplete or skipped tests!
Tests: 3, Assertions: 0, Incomplete: 3.
```

Remarque : l'option `--bootstrap ../src/CompteBancaire.class.php` évite d'ajouter la ligne suivante dans `TestCompteBancaire.class.php`

```
require_once('src/CompteBancaire.class.php');
```

Pour la suite, on ajoutera cette ligne dans le fichier de la classe de test.

On complète maintenant le squelette généré et on obtient :

```
<?php

require_once('src/CompteBancaire.class.php');

/**
 * Generated by PHPUnit_SkeletonGenerator 1.2.1 on 2013-12-18 at
10:44:24.
 */
class TestCompteBancaire extends PHPUnit_Framework_TestCase
{
    /**
     * @var CompteBancaire
     */
    protected $object;

    /**
     * Sets up the fixture, for example, opens a network connection.
     * This method is called before a test is executed.
     */
    protected function setUp()
    {
        $this->object = new CompteBancaire;
    }

    /**
     * Tears down the fixture, for example, closes a network
connection.
     * This method is called after a test is executed.
     */
    protected function tearDown()
    {
    }

    /**
     * @covers CompteBancaire::getBalance
     * @todo   Implement testGetBalance().
     */
```

```
    public function testGetBalance()
    {
        $this->assertEquals(0, $this->object->getBalance());
    }

    /**
     * @covers CompteBancaire::deposerArgent
     * @todo   Implement testDeposerArgent().
     */
    public function testDeposerArgent()
    {
      /* classe valide */
      $argent = $this->object->getBalance();
      $this->object->deposerArgent($argent);
      $this->assertEquals(0, $this->object->getBalance());

      /* classe invalide */
      try
      {
         $this->object->retirerArgent(1);
      }
      catch (CompteBancaireException $e)
      {
         $this->assertEquals(0, $this->object->getBalance());
      }
    }

    /**
     * @covers CompteBancaire::retirerArgent
     * @todo   Implement testRetirerArgent().
     */
    public function testRetirerArgent()
    {
      /* classe invalide */
      try
      {
         $this->object->deposerArgent(-1);
      }
      catch (CompteBancaireException $e)
      {
         $this->assertEquals(0, $this->object->getBalance());
      }

      /* classe valide */
      $this->object->deposerArgent(1);
      $this->assertEquals(1, $this->object->getBalance());
    }

    /**
      * @covers CompteBancaire::getBalance
```

```
     * @covers CompteBancaire::deposerArgent
     * @covers CompteBancaire::retirerArgent
     */

    public function testDeposerRetirerArgent()
    {
      $this->assertEquals(0, $this->object->getBalance());
      $this->object->deposerArgent(1);
      $this->assertEquals(1, $this->object->getBalance());
      $this->object->retirerArgent(1);
      $this->assertEquals(0, $this->object->getBalance());
    }
}
```

On se place à la racine :

```
$ cd /var/www/compte-bancaire/
```

Et on relance le test :

```
$ phpunit --verbose TestCompteBancaire
tests/TestCompteBancaire.class.php

....

Time: 4 ms, Memory: 4.50Mb

OK (4 tests, 8 assertions)
```

Remarque : Un indicateur de résultat est fourni pour chaque méthode de test exécutée

. -> Le test passe.
F -> Le test a échoué (Failure).
E -> Le test a généré une erreur PHP.
S -> Le test est ignoré (Skipped).
I -> Le test est marqué comme incomplet (Incomplete).

Exemple n°2 : La classe Fichier

Cet exemple illustre une démarche pour écrire des tests unitaires **avant** d'avoir écrit du code source pour l'application, ici la classe `Fichier` que l'on ajoute à notre projet.

Cette classe doit posséder deux attributs :

- `nomFichier` : le nom de fichier (de 1 à 32 caractères alphanumériques, trait d'union et sous-tiret compris) sous forme d'une chaîne de caractères
- `extension` : l'extension de fichier (de 1 à 3 caractères alphanumériques) sous forme d'une chaîne de caractères

Et les accesseurs suivants :

- `setNomFichier($nomFichier)` : permet d'affecter un nom de fichier et retourne vrai si celui-ci est valide
- `setExtension($extension)` : permet d'affecter une extension de fichier et retourne vrai si celle-ci est valide
- `getNomFichier()` : retourne le nom de fichier
- `getExtension()` : retourne l'extension de fichier

Elle devra aussi posséder un constructeur par défaut et un constructeur acceptant deux paramètres : un nom et une extension de fichier.

On crée le fichier (vide) pour la classe de test `TestFichier` :

```
$ touch tests/TestFichier.class.php
```

On édite ensuite le fichier :

```
<?php

require_once('src/Fichier.class.php');

class TestFichier extends PHPUnit_Framework_TestCase
{
    protected $file;

    protected function setUp()
    {
        $this->file = new Fichier;
    }

    protected function tearDown() {}

    public function testSetNomFichier()
    {
      /* classes valides */
```

```
      $this->assertTrue($this->file->setNomFichier('newfile'));
      $this->assertTrue($this->file->setNomFichier('newfile1'));
      $this->assertTrue($this->file->setNomFichier('new_file'));
      $this->assertTrue($this->file->setNomFichier('new-file'));

      /* classes invalides */
      $this->assertFalse($this->file->setNomFichier(null));
      $this->assertFalse($this->file-
>setNomFichier('ThisFileNameIsVeryLongTooLongReallyTooLong'));
      $this->assertFalse($this->file-
>setNomFichier('NewFileWith.ext'));
      $this->assertFalse($this->file->setNomFichier('àfile'));
      $this->assertFalse($this->file->setNomFichier(1));
    }

    public function testSetExtension()
    {
      /* classes valides */
      $this->assertTrue($this->file->setExtension('txt'));
      $this->assertTrue($this->file->setExtension('csv'));

      /* classes invalides */
      $this->assertFalse($this->file->setExtension(null));
      $this->assertFalse($this->file->setExtension('é'));
      $this->assertFalse($this->file->setExtension('abcdef'));
      $this->assertFalse($this->file->setExtension(1));
    }

    public function testGetNomFichier()
    {
      /* classe valide */
      $resultatAttendu = 'newfile';

      $this->assertTrue($this->file->setNomFichier('newfile'));
      $this->assertEquals($resultatAttendu, $this->file-
>getNomFichier());
    }

    public function testGetExtension()
    {
      /* classe valide */
      $resultatAttendu = 'txt';

      $this->assertTrue($this->file->setExtension('txt'));
      $this->assertEquals($resultatAttendu, $this->file-
>getExtension());
    }
}

?>
```

On va maintenant utiliser **le générateur de squelette pour créer le fichier de classe `Fichier` à tester** :

```
$ phpunit-skelgen --class -- TestFichier tests/TestFichier.class.php Fichier src/Fichier.class.php
```

On peut déjà exécuter le test :

```
$ phpunit --verbose TestFichier tests/TestFichier.class.php
PHPUnit 3.7.28 by Sebastian Bergmann.

EEEE

Time: 2 ms, Memory: 4.25Mb

There were 4 errors:

1) TestFichier::testSetNomFichier
RuntimeException: Not yet implemented.

/var/www/compte-bancaire/src/Fichier.class.php:40
/var/www/compte-bancaire/tests/TestFichier.class.php:21

2) TestFichier::testSetExtension
RuntimeException: Not yet implemented.

/var/www/compte-bancaire/src/Fichier.class.php:31
/var/www/compte-bancaire/tests/TestFichier.class.php:37

3) TestFichier::testGetNomFichier
RuntimeException: Not yet implemented.

/var/www/compte-bancaire/src/Fichier.class.php:40
/var/www/compte-bancaire/tests/TestFichier.class.php:52

4) TestFichier::testGetExtension
RuntimeException: Not yet implemented.

/var/www/compte-bancaire/src/Fichier.class.php:31
/var/www/compte-bancaire/tests/TestFichier.class.php:61

FAILURES!
Tests: 4, Assertions: 0, Errors: 4.
```

Évidemment le test échoue car nous avons écrit aucune ligne de code dans notre classe `Fichier` !

On édite ensuite le fichier `Fichier.class.php` :

```
<?php

class Fichier
{
   protected $nomFichier;
   protected $extension;

   public function __construct($nomFichier = null, $extension = null)
   {
      if (!is_null($nomFichier))
      {
         $this->setnomFichier($nomFichier);
      }

      if (!is_null($extension))
      {
         $this->setExtension($extension);
      }
   }

    /**
     * Affecte un nom de fichier (de 1 à 32 caractères
alphanumériques, trait d'union et sous-tiret compris).
     *
     * @param string $nomFichier
     * @return boolean
     */
   public function setNomFichier($nomFichier)
   {
      if (!is_string($nomFichier))
      {
         return false;
      }

      $this->nomFichier = $nomFichier;

      return true;
   }

    /**
     * Affecte une extension de fichier (de 1 à 3 caractères
alphanumériques).
     *
     * @param string $extension
     * @return boolean
     */
   public function setExtension($extension)
   {
```

```
      if (!is_string($extension))
      {
         return false;
      }

      $this->extension = $extension;

      return true;
   }

   /**
     * Retourne le nom de fichier.
     *
     * @return string
     */
   public function getNomFichier()
   {
      return $this->nomFichier;
   }

   /**
     * Retourne l'extension du fichier
     *
     * @return string
     */
   public function getExtension()
   {
      return $this->extension;
   }
}

?>
```

On lance le test :

```
$ phpunit --verbose TestFichier tests/TestFichier.class.php
PHPUnit 3.7.28 by Sebastian Bergmann.

FF..

Time: 4 ms, Memory: 4.75Mb

There were 2 failures:

1) TestFichier::testSetNomFichier
Failed asserting that true is false.

/var/www/compte-bancaire/tests/TestFichier.class.php:28

2) TestFichier::testSetExtension
```

```
Failed asserting that true is false.

/var/www/compte-bancaire/tests/TestFichier.class.php:42

FAILURES!
Tests: 4, Assertions: 14, Failures: 2.
```

Le test détecte **deux erreurs** (sur les 14 assertions testées) dans les méthodes `testSetNomFichier` et `testSetExtension` :

- ligne 28 : `$this->assertFalse($this->file->setNomFichier('ThisFileNameIsVeryLongTooLongReallyTooLong'));` car le nom de fichier est trop long !
- Ligne 42 : `$this->assertFalse($this->file->setExtension('é'));` car 'é' est un caractère invalide pour une extension de fichier

On va donc apporter une correction (en utilisant une **expression régulière** pour vérifier la validité du nom et de l'extension passés en paramètre) aux méthodes `testSetNomFichier` et `testSetExtension` :

```
    /**
     * Affecte un nom de fichier (de 1 à 32 caractères
alphanumériques, trait d'union et sous-tiret compris).
     *
     * @param string $nomFichier
     * @return boolean
     */
   public function setNomFichier($nomFichier)
   {
      if (!is_string($nomFichier))
      {
         return false;
      }

      if (!preg_match('/^[a-z0-9-_]{1,32}$/i',$nomFichier))
      {
         return false;
      }

      $this->nomFichier = $nomFichier;

      return true;
   }

    /**
     * Affecte une extension de fichier (de 1 à 3 caractères
alphanumériques).
     *
     * @param string $extension
     * @return boolean
```

```
     */
   public function setExtension($extension)
   {
      if (!is_string($extension))
      {
         return false;
      }

      if (!preg_match('/^[a-z0-9]{1,3}$/i',$extension))
      {
         return false;
      }

      $this->extension = $extension;

      return true;
   }
```

On relance le test :

```
$ phpunit --verbose TestFichier tests/TestFichier.class.php

PHPUnit 3.7.28 by Sebastian Bergmann.

....

Time: 4 ms, Memory: 4.50Mb

OK (4 tests, 19 assertions)
```

Maintenant, aucune erreur n'a été détectée sur les 19 assertions testées !

Exemple n°3 : Organiser les tests

Le manuel fournit un chapitre sur les différentes façons d'organiser les tests :

http://phpunit.de/manual/3.7/fr/organizing-tests.html

Méthode n°1 :
Comme nous avons pris soin de composer notre suite de tests (cad tous les fichiers sources des cas de test) dans un répertoire `tests`, PHPUnit peut automatiquement trouver et exécuter les tests en parcourant récursivement ce répertoire.

Remarque : par défaut, PHPUnit va chercher les fichiers `*Test.php`. Notre convention est différente !

Dans notre cas, nous devons exécuter la commande suivante pour s'adapter au comportement de PHPUnit :

```
$ phpunit --verbose --test-suffix class.php tests

PHPUnit 3.7.28 by Sebastian Bergmann.

........

Time: 5 ms, Memory: 4.75Mb

OK (8 tests, 27 assertions)
```

Méthode n°2 :
On peut aussi composer une suite de tests en utilisant un fichier de configuration XML.

Pour cela, nous allons créer le fichier de configuration **`compte-bancaire.xml`** :

```
<phpunit>
  <testsuites>
    <testsuite name="Compte Bancaire">
      <file>tests/TestCompteBancaire.class.php</file>
      <file>tests/TestFichier.class.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

Pour lancer l'ensemble des tests, on exécutera la commande suivante :

```
$ phpunit --configuration compte-bancaire.xml

PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /var/www/compte-bancaire/compte-bancaire.xml
```

```
........

Time: 5 ms, Memory: 5.25Mb

OK (8 tests, 27 assertions)
```

Nous n'irons pas plus loin et pour conclure :

LISEZ LE MANUEL !

http://phpunit.de/manual/3.7/fr/