Hébergement & Cybersécurité

TP - Tests unitaires avec PHP

IUT de Lannion - BUT3 MMI

Objectifs :

— Installer et comprendre un nouvel environnement en PHP.
— Écrire et comprendre vos premiers tests unitaires avec PHPUnit.
— Analyser un mini cahier des charges, coder l'application en PHP et écrire les tests correspondants.

Prérequis :

— disposer de PHP installé sur votre machine.
— disposer d'un IDE compatible PHP (comme VSCode ou PhpStorm).
— disposer d'un compte Gitlab (ou GitHub).
— avoir installé Composer, le gestionnaire de dépendances PHP.

Pour une introduction à Git : `https://www.cristal.univ-lille.fr/TPGIT/`
Pour une introduction à PHPUnit : `https://phpunit.de/getting-started/phpunit-10.html`

1 Création du projet en PHP et premier commit

Dans votre IDE, créez un nouveau projet en PHP. Pour initialiser votre projet, utilisez Composer pour ajouter PHPUnit en tant que dépendance de développement avec la commande suivante :

```
composer require --dev phpunit/phpunit
```

Créez une classe Money.php. Maintenant que vous avez un début de classe, il est temps (et oui déjà) de faire votre premier commit. Attention néanmoins à ne pas commiter tout le répertoire ! Il contient de nombreux fichiers de configuration liés à votre IDE, qui sont donc propres à votre machine. Vous ne souhaitez pas imposer ces fichiers à vos collègues, ni récupérer les leurs lorsque vous collaborerez à plusieurs.

A l'aide de votre IDE, ajoutez un nouveau fichier à la racine de votre répertoire de travail intitulé .gitignore dans lequel vous allez préciser les fichiers et dossiers que vous ne souhaitez pas commiter (par exemple, le répertoire vendor où Composer installe les dépendances).

Configurez un système de versionnage via votre IDE.

2 Amélioration de Money et exécution des tests avec PHPUnit

Etoffons la classe Money, qui représente un montant entier ($amount) dans une devise donnée ($currency). Ajoutez un constructeur ainsi qu'une méthode __toString().

Créez un fichier index.php, où vous allez instancier la classe Money avec un montant de 42 EUR, puis afficher le résultat avec echo.
Exemple :

```php
<?php
require( 'Money.php');

$money = new Money(42, 'EUR');
echo $money;
```

Vous pourrez constater le résultat dans la sortie de la console de l'IDE ou en exécutant la commande :

```
php index.php
```

Le répertoire vendor et les fichiers de configuration n'ont pas vocation à être mis en ligne dans votre dépôt. Mettez à jour votre fichier .gitignore si nécessaire. Vous pouvez ensuite à nouveau commiter le reste de votre travail.

3 Mon premier test

Créez un répertoire tests dans le répertoire racine de votre projet.

Dans le répertoire tests, ajoutez un fichier MoneyTest.php et intégrez une méthode de test testAddEuros() pour vérifier qu'il est possible d'additionner deux instances de la classe Money ayant la même devise, comme suit :

```php
<?php
use PHPUnit\Framework\TestCase;
require_once ('Money.php');
```

```php
class MoneyTest extends TestCase
{
  public function testAddEuros()
  {
    $m1 = new Money(20, 'EUR');
    $m2 = new Money(10, 'EUR');
    $expected = new Money(30, 'EUR');
    $this->assertEquals($expected, $m1->add($m2));
  }
}
```

La méthode `$this->assertEquals()` de PHPUnit est utilisée ici pour vérifier que les deux instances sont identiques, validant ainsi le test. Assurez-vous que votre méthode `add()` retourne une nouvelle instance de `Money` avec le montant mis à jour.

Voici la méthode `add()` à ajouter dans la classe `Money` :

```php
<?php
public function add(Money $that)
{
  if ($this->currency !== $that->currency) {
    throw new Exception("Devises différentes");
  }
  return new Money($this->amount + $that->amount, $this->currency);
}
```

Vous pouvez maintenant exécuter le test avec la commande suivante :

```
vendor/bin/phpunit --testdox tests
```

Pour que PHPUnit puisse comparer les instances de la classe Money, il est nécessaire de redéfinir la méthode `equals()` dans la classe Money pour que deux instances ayant le même montant et la même devise soient considérées comme égales :

```php
<?php
public function equals(Money $that): bool
{
  return $this->amount === $that->amount && $this->currency ===
  ↪  $that->currency;
}
```

Votre premier test devrait maintenant s'exécuter correctement dans l'IDE et via PHPUnit. Vous pouvez également effectuer un commit et mettre à jour votre dépôt en ligne avec un `git push`.

Après avoir effectué votre premier test, vous pouvez consulter un rapport des tests dans un fichier HTML en configurant PHPUnit pour générer un rapport sous forme HTML dans un répertoire de sortie.

4 Est-ce que Gitlab peut confirmer que mon test passe bien de son côté aussi ?

(Premier pas vers l'intégration continue)

Ajoutez le fichier `.gitlab-ci.yml` à la racine de votre répertoire de travail. Ce fichier indiquera à GitLab quel comportement adopter pour exécuter les tests unitaires. Voici un exemple de fichier `.gitlab-ci.yml` configuré pour PHP et PHPUnit :

```
stages:
- test

phpunit_test:
image: php:latest
stage: test
script:
- apt-get update && apt-get install -y git zip unzip
- curl -sS https://getcomposer.org/installer | php
- php composer.phar install
- vendor/bin/phpunit --testdox tests
```

Commitez ce nouveau fichier et poussez-le dans votre dépôt GitLab. Ensuite, allez dans le menu CI/CD sur votre dépôt GitLab pour suivre l'exécution de votre pipeline CI/CD : GitLab installera les dépendances, exécutera les tests, et affichera les résultats.

Variante GitHub

Créez un dossier `.github/workflows` dans le répertoire racine de votre projet. Ajoutez un fichier `php.yml` pour configurer GitHub Actions et exécuter les tests avec PHPUnit :

```
name: PHP PHPUnit Test

on: [push, pull_request]

jobs:
phpunit:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- name: Set up PHP
uses: shivammathur/setup-php@v2
with:
php-version: '8.0'
- name: Install dependencies
run: composer install
- name: Run PHPUnit tests
run: vendor/bin/phpunit --testdox tests
```

Une fois le fichier commité, accédez à l'onglet « Actions » sur votre dépôt GitHub pour visualiser le déroulement des tests.

5 Premiers pas vers la couverture de code

Vous avez précédemment écrit un test pour vérifier l'addition de deux instances de la classe Money. Vous pouvez maintenant vérifier la couverture de code de vos tests.

Installez Xdebug, un outil compatible avec PHPUnit pour analyser la couverture de code, et activez-le dans la configuration PHP. Lancez les tests avec :

```
vendor/bin/phpunit --coverage-html coverage-report
```

Cela générera un rapport de couverture en HTML dans un répertoire `coverage-report`. Vous pourrez ouvrir le fichier `index.html` dans votre navigateur pour voir les résultats détaillés.

En améliorant vos tests, vous augmenterez la couverture de votre code. Vous pouvez explorer d'autres outils pour automatiser l'analyse de la couverture de code, comme Codecov.

6 En bonus...

Au-delà de l'addition, quelles autres opérations pourriez-vous imaginer effectuer à l'aide de cette classe Money ? Comment les tester ? Testez !

Ajoutez une nouvelle classe `MoneyBag` qui prend en charge plusieurs devises et permet la normalisation du `MoneyBag` (une seule occurrence de chaque devise). Définissez des méthodes pour ajouter et retirer des instances de `Money` et testez-les. Ensuite, vérifiez la couverture de vos tests.

Pour des transactions dans des devises différentes, vous devrez gérer les taux de change au sein d'un `MoneyBag`.

Enfin, envisagez un système `Wallet` pour stocker des moyens de paiement (carte de débit, crédit, chèque, etc.) associés à une devise. Implémentez des méthodes pour transférer de l'argent entre ces moyens de paiement et un `MoneyBag` en testant divers comportements.

7 Gestion des comptes bancaires après les monnaies

Après avoir géré des monnaies, passez à la gestion des comptes bancaires en vous basant sur le cahier des charges ci-dessous.

7.1 Cahier des charges

Il s'agit de définir une classe en PHP pour modéliser des comptes bancaires. Les besoins sont les suivants :

1. Un compte bancaire est identifié par un numéro de compte. Ce numéro de compte est un entier positif permettant de désigner et distinguer sans ambiguïté possible chaque compte géré par l'établissement bancaire. Chaque compte possède donc un numéro unique. Ce numéro est attribué par la banque à l'ouverture du compte et ne peut être modifié par la suite.
2. Un compte est associé à une personne titulaire du compte, cette personne étant décrite par son nom, son prénom et son adresse. Une fois le compte créé, le titulaire du compte ne peut plus être modifié. Une même personne peut être titulaire de plusieurs comptes.
3. La somme d'argent disponible sur un compte est exprimée en Euros. Cette somme est désignée sous le terme de solde du compte. Ce solde est un nombre décimal qui peut être positif, nul ou négatif.
4. Le solde d'un compte peut être éventuellement (et temporairement) être négatif. Dans ce cas, on dit que le compte est à découvert. Le découvert d'un compte est nul si le solde du compte est positif ou nul, il est égal à la valeur absolue du solde si ce dernier est négatif.
5. En aucun cas le solde d'un compte ne peut être inférieur à une valeur fixée pour ce compte. Cette valeur est définie comme étant - (moins) le découvert maximal autorisé pour ce compte.
6. Créditer un compte consiste à ajouter un montant positif au solde du compte.
7. Débiter un compte consiste à retirer un montant positif au solde du compte. Le solde résultant ne doit en aucun cas être inférieur au découvert maximal autorisé pour ce compte.
8. Lors d'une opération de retrait, un compte ne peut être débité d'un montant supérieur à une valeur désignée sous le terme de débit maximal autorisé.
9. Effectuer un virement consiste à débiter un compte au profit d'un autre compte qui sera crédité du montant du débit.
10. Lors de la création d'un compte seul l'identité du titulaire du compte est indispensable. En l'absence de dépôt initial le solde est fixé à 0.

7.2 Travail demandé

Définissez et testez une classe (Compte) permettant de créer et d'utiliser des comptes bancaires.

— **Question 1** : Spécifiez la classe Compte.

— **Question 2** : Écrivez un test PHPUnit pour la classe Compte.

— **Question 3** : Implémentez la classe Compte et vérifiez les tests au fur et à mesure.

— **Question 4** : Vérifiez la couverture de vos tests.

— **Question 5** : Testez des opérations pour différents comptes.

1. Créer un compte pour Eric Martin avec un solde de 5 000 €.
2. Créer un compte pour Wulfric Tartampion avec un solde de – 20 €.
3. Créer un compte pour Eric Nopon avec un solde de 600 €.
4. Créer un compte pour Berlioz Le Du avec un découvert autorisé de – 3 000 €.
5. Créer un compte pour Adélaïde de Bonnefamille avec un solde de 350 000 €.
6. Virer 1 500 € du compte d'Eric Martin vers le compte d'Eric Nopon.
7. Créditer 50 € sur le compte d'Eric Nopon.
8. Débiter – 30 € du compte d'Adélaïde de Bonnefamille.
9. Débiter 400 € du compte d'Eric Martin.
10. Créditer – 4 000 € sur le compte d'Eric Nopon.
11. Vérifier si les 3 comptes créés ci-dessus sont à découvert ou non (return bool).
12. Fixer le découvert maximum du compte de Wulfric Tartampion à – 2 000 €.
13. Vérifier quel est le montant maximum pouvant être retiré en une seule opération du compte d'Adélaïde de Bonnefamille (return double).
14. Virer 1 200 € du compte d'Eric Nopon vers le compte d'Adélaïde de Bonnefamille.
15. Modifier le débit maximal autorisé d'Eric Nopon à 1 500 €.
16. Virer 1 500 € du compte d'Eric Nopon vers le compte d'Adélaïde de Bonnefamille.
17. Vérifier quel est le montant maximum pouvant être retiré en une seule opération du compte d'Eric Nopon (return double).
18. Créer un compte pour Marie Bambelle avec un débit maximal autorisé de – 4000 €.
19. Créer un compte pour Marie de Bonnefamille.
20. Mettre à jour le nom de Marie de Bonnefamille en Marie de Bonnefamille O'Malley suite à son mariage, ainsi que le nom d'Eric Nopon en Eric Nopon Le Du suite à son mariage.
21. Modifier l'adresse d'Adélaïde de Bonnefamille pour indiquer « Manoir de Bonnefamille, 75000 Paris ».

— **Question 6** : Quelles sont les limites de cette modélisation (plutôt simpliste) d'un système bancaire ? Quels besoins exprimés dans la partie .1 étaient plutôt discutables, et pourquoi ?

Remarque : toutes les situations décrites avec un numéro pair dans la Question 5 doivent générer une exception que vous devrez gérer.