Lycée Léonard de Vinci *Section Technicien Supérieur en Informatique SIO Option SLAM*

CONTRÔLE DES CONNAISSANCES N° 1 Vendredi 17 septembre 2021
DURÉE : 1h

Note sur 30 : Note sur 20 :

***Partie I** : Test Unitaire : Questions à Choix Multiples (**8 points**) – (Rédacteur Yacine)*

Q1 Une méthode de test unitaire : (**2 points**)

- [ ] traite plusieurs type de problèmes
- [x] traite de préférence un seul type de problème
- [ ] traite généralement plusieurs types de problème
- [ ] je ne sais pas

Une méthode de test unitaire traite de préférence un seul type de problème indépendamment des autres tests.

Q2 Avec les frameworks de type xUnit, une méthode de test unitaire est : (**2 points**)

- [x] une méthode d'une classe dédiée aux tests
- [ ] une méthode d'une classe métier
- [ ] une méthode « static »
- [x] une méthode d'instance
- [ ] une méthode de type fonction (retournant un résultat)
- [ ] je ne sais pas

C'est précisément le but du test unitaire. Une méthode d'instance s'utilise sur l'instance d'une classe, appelée aussi objet. L'instance d'une classe c'est quand tu fais un new. Pour l'appeler il faudra faire objet.maMethode(), elle peut bien sur prendre des paramètres et retourner quelque chose et peut modifier l'**état** de l'objet.

Q3 Une classe de test est généralement : (**2 points**)

- [x] liée à un projet
- [x] placée dans un package de même nom que la classe testée
- [ ] placée en dehors de tout package
- [(x)] constituée de plus de lignes de code que la classe qu'elle teste

Une classe de test est généralement liée a un projet et généralement, on groupe dans une classe les tests ayant la même classe comme point d'entrée et on nomme la classe de test à partir du nom de la

classe testée préfixé ou suffixé par Test en la plaçant dans le même package. Par exemple, pour tester la classe DateFormatter , on créera une classe TestDateFormatter ou DateFormatterTest.

Q4 Un test unitaire qui passe avec succès : (**2 points**)

- [ ] est une garantie que la méthode testée n'est pas boguée
- [x] est une garantie que la méthode testée répond correctement aux spécifications du scénario de test
- [ ] est une garantie que la méthode testée fonctionne correctement
- [ ] est une garantie que la méthode testée tient ses engagements vis à vis de ses spécifications
- [ ] est une garantie que la méthode testée n'entraînera pas de dysfonctionnement à l'exécution

Un test unitaire qui passe avec succès est une garantie que la méthode testée répond correctement aux spécifications du scénario de ce test.

Partie 2 Compréhension et conception de TU (22 points) (Rédacteur Gabriel)

Soit le test unitaire suivant :

```
public function testColor()
  {
    $card = new Card('As', 'Trefle');
    $this->assertEquals('Trefle', $card->getColor());
    $card = new Card('As', 'Pique');
    $this->assertEquals('Pique', $card->getColor());
  }
```

à l'exécution, nous recevons le message suivant :

```
1) App\Tests\Core\CardTest::testColor
Failed asserting that two strings are equal.
--- Expected  =>  'Pique '
+++ Actual    =>  'Trefle'
```

Q5 => Expliquer le problème rencontré (**3 points**)

Au deuxième assert, le test attend 'Pique' mais reçoit 'Trefle'

.

.

.

.

.

Q6 => Donner (une simple hypothèse) ce qui pourrait être à l'origine de ce bug. (**3 points**)

Le problème pourrait venir directement de getColor() qui pourrait juste renvoyer 'Trefle' tel que :

```
public getColor(){
     return 'Trefle';
}
```

La méthode Card::compare s'appuie sur le tableau suivant pour définir une relation d'ordre de 2 cartes de même nom mais de couleurs différentes.

// du plus fort au moins fort

```
const ORDER_COLORS=['trefle' => 4, 'carreau' => 3, 'pique' => 2, 'coeur' => 1 ];
```

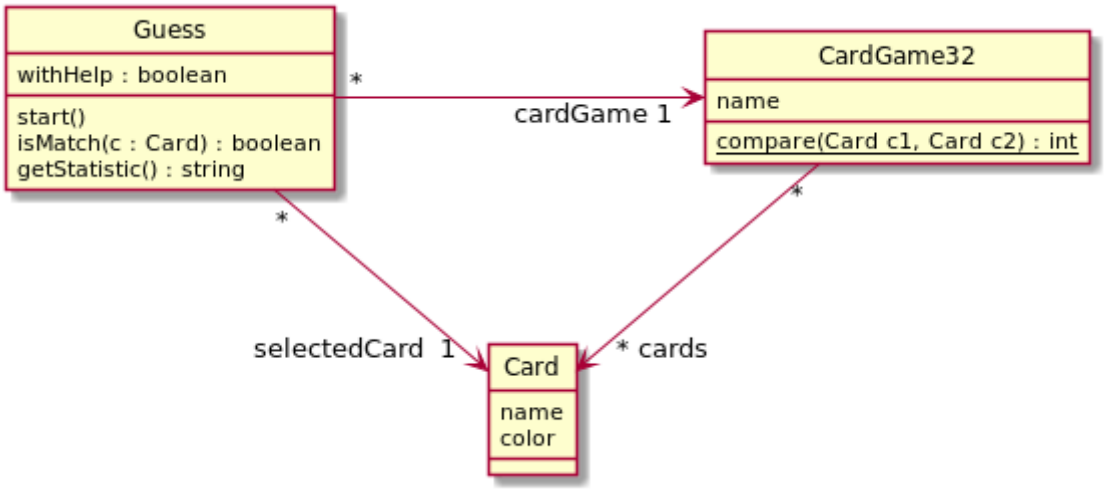
Q7 Déterminer (en le justifiant) la valeur qui doit être mise dans la variable **$expected** du test suivant (ligne 105) : (**4 points**)

```
101 :  public function testCmpSameNameNoSameColor()
102 :  {
103 :    $card1 = new Card('valet', 'trefle');
104 :    $card2 = new Card('valet', 'pique');
105 :    $expected = ?????;  <=== à vous de déterminer la valeur attendue
106 :    $this->assertEquals($expected, Card::compare($card1,$card2));
107 :  }
```

votre réponse => 1 (conforme aux SPECS de Card::compare)

pourquoi ? => car d'après ORDER_COLORS, la valeur d'ordre associée à trefle est supérieure à celle de pique (donc trefle > pique)

À partir de cette analyse :



Votre collègue a conçu le constructeur de CardGame32 suivant :

```
1 : class CardGame32 {
2 :
  [...]
 100 : public function __construct(array $cards) {
 101 :   $this->name = 'Jeu de 32 cartes';
 102 :   if (count($cards) === 0) {
 103 :     $this->cards = self::factoryCardGame32()->getCards();
 104 :   } else {
 105 :     $this->cards = $cards;
 106 :   }
 107 :}
108 :}
```

Remarque : la fonction PHP *count* retourne le nombre d'éléments du tableau qu'elle a reçu en argument.

Q8 Expliquer la logique qui a amené votre collègue à coder ce constructeur. (**4 points**)

Il faut qu'un jeu de carte ait au minimum une carte. Si on en donne pas à la création, le code créer un jeu de 32 cartes, si on le crée avec au moins une carte alors le jeu se crée avec les cartes reçues en paramètre

Q9 Expliquer le rôle de l'instruction en ligne 105 (**2 points**)

La valeur du paramètre $cards est affecté à l'attribut $cards de l'objet.

Q10 Concevoir 2 méthodes de test différentes qui vérifient le bon comportement du constructeur de CardGame32 défini par votre collègue (**6 points**)

```
public function testConstruct1()
{
      $cardGame = new CardGame32();
      $this -> assertEquals(32,count($cardGame->getCard()));
}
public function testConstruct2()
{
      $cardGame=new CardGame32(new Card('As','Coeur'),new Card('Roi','Coeur'));
      $this -> assertEquals(2,count($cardGame->getCard()));
}
```