Doublure d'objets pour la réalisation des tests - EasyMock

Source : le cours de Michel Doudoux - https://www.jmdoudoux.fr/java/dej/chap-objets-mock.htm -

L'usage de doublure d'objets dans les tests

Dans une application, les classes ont généralement des dépendances entre elles. Ceci est particulièrement vrai dans les applications développées en couches (présentation, service, métier, accès aux données, ...).

Il est très important que les tests unitaires ne concernent que le code de la méthode en cours de test. Autrement, il est difficile de trouver un bug qui peut être dans un objet dépendant. Donc, lors de l'exécution d'un test unitaire, il faut tester la plus petite unité de code possible, une **méthode** et uniquement le code de la méthode. Cependant, les méthodes font généralement appel à des méthodes d'un ou plusieurs autres objets. Le but n'est pas de tester les méthodes appelées qui feront eux-mêmes l'objet de tests unitaires : le test unitaire doit concerner uniquement la méthode et ne pas tester les dépendances.

L'utilisation des doublures permet aux tests unitaires de se concentrer sur les tests du code de la méthode qui correspond au System Under Test (**SUT**) sans avoir à se préoccuper des dépendances.

Le but d'une doublure est de remplacer un objet en forçant les valeurs de retour de ses méthodes selon certains paramètres. Ainsi l'invocation d'une méthode d'une doublure garantit d'avoir les valeurs attendues selon les paramètres fournis.

En « Programmation Orienté Objet », il existe plusieurs types de doublures permettant de simuler le comportement d'un autre objet :

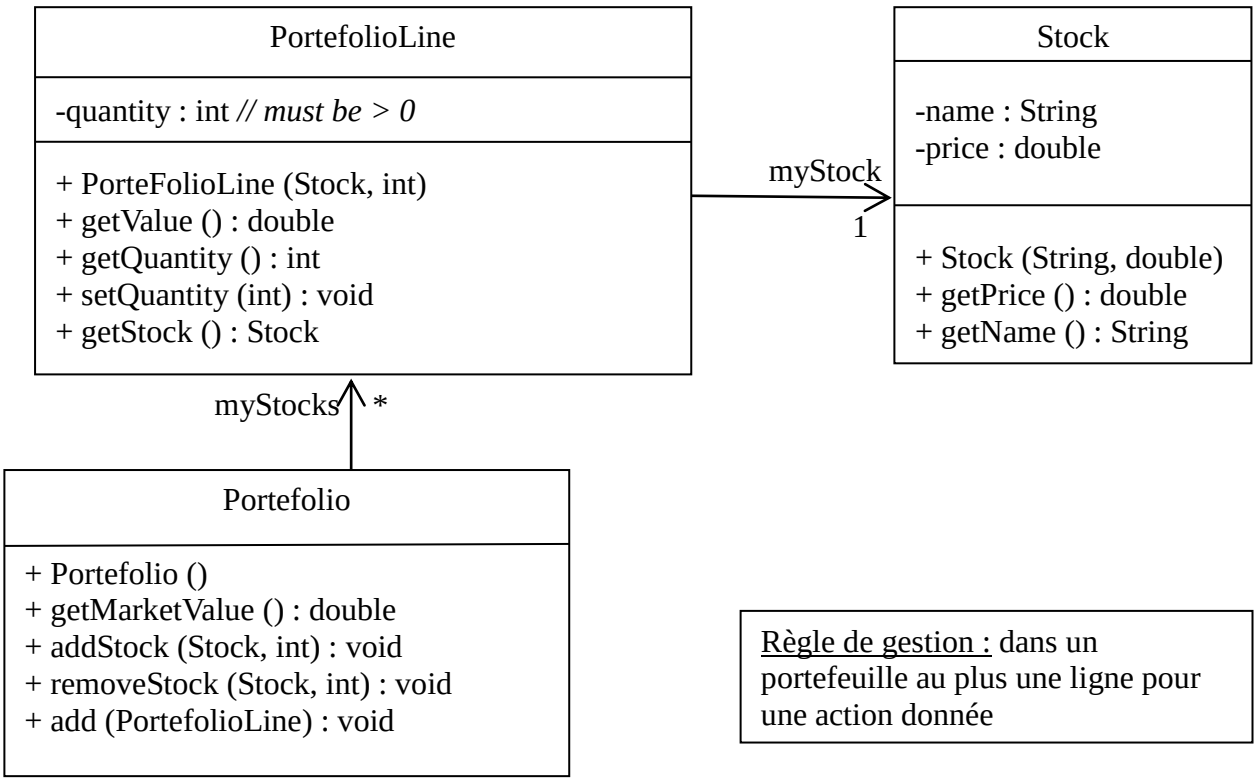
- Fantôme (dummy) : objets "vides" qui n'ont pas de fonctionnalité implémentée.
- bouchon (stub) : classes qui renvoient une valeur codée en dur à l'invocation d'une méthode
- simulateur (fake) : classes qui sont une implémentation partielle et qui, par exemple, renvoient toujours les mêmes réponses quels que soient les paramètres fournis.
- espion (spy) : classe qui enregistre les appels à ses méthodes avec leur paramètre.
- simulacre (mock) : classes qui agissent comme un bouchon et un espion.

Le vocabulaire lié à ces types d'objets est assez confus dans la langue anglaise donc il l'est aussi dans la langue française où l'on tente de le traduire.

Les doublures ont pour rôle de simuler le comportement d'un objet permettant ainsi de réaliser les tests de l'objet de façon isolée et répétable. Mais aussi de vérifier les invocations aux objets.
Cette double fonctionnalité permet dans un test unitaire de faire des tests sur les appels aux méthodes des autres objets (comportement), en sus des tests sur les états.

Portefeuille d'actions –Tests exhaustifs en utilisant l'API EasyMock

Diagramme de classes - Le code est sur le moodle.



Exemple de test de la méthode **`getValue()`** de la classe **`PortefolioLine`** à l'aide d'une doublure de la classe **`Stock`**. La méthode **`getValue()`** retourne la valeur de cette ligne d'un portefeuille d'action.
Info : La méthode **`reset()`** permet de supprimer tous les comportements définis d'une doublure.

```
public class PortefolioLineTest {
	Stock stock1;

	@Before
	public void setUp() throws Exception {
		stock1 = EasyMock.createMock(Stock.class);}

	@Test
	public void testGetValue() {
		getValueTest(1);   getValueTest(100);    }

	private void getValueTest(int qt) throws IllegalArgumentException {
		EasyMock.reset(stock1);                                          //1
		PortefolioLine line = new PortefolioLine(stock1, qt);

		EasyMock.expect(stock1.getPrice()).andReturn(50.00);             //2
		EasyMock.replay(stock1);                                         //3

		double value = line.getValue();

		Assert.assertEquals(50.0*qt, value,  0.0);

		EasyMock.verify(stock1);                                         //4
   }
}
```

Question 0 :

1. Exécutez les tests « PortefolioLineTest », observez, notez et expliquez le résultat dans les cas suivants :
   - **a.** la ligne notée //1 est en commentaire. ;
   - **b.** la ligne notée //2 est en commentaire ;
   - **c.** la ligne notée //3 est en commentaire ;
   - **d.** la ligne notée //4 est en commentaire ;
   - **e.** aucune ligne n'est en commentaire.

Le constructeur **`PorteFolioLine`** lève une exception **`IllegalArgumentException`** si la quantité d'action est un entier négatif ou nul. Voici les tests du constructeur de PortFolioLine :

```
@Test
public void PorteFolioConstructeurTestPostive() {
              new PortefolioLine(stock1,2); new PortefolioLine(stock1,200);  }

@Test(expected=IllegalArgumentException.class)
public void PorteFolioConstructeurTestZero() {  new PortefolioLine(stock1,0);     }

@Test(expected=IllegalArgumentException.class)
public void PortFolioConstructeurTestNegatif() { new PortefolioLine(stock1,-20); }

@Test(expected=IllegalArgumentException.class)
public void PorteFolioConstructeurTestNull() { new PortefolioLine(null,20); }
```

`setQuantity(x)` de la classe **`PortefolioLine`** lève l'exception **`IllegalArgumentException`** si **`x <= 0.`**

Question 1 : Testez de manière exhaustive la méthode **`SetQuantity()`** de la classe **`PortefolioLine`**.

« getMarketValue » d'un portefeuille d'actions est la somme des lignes.

Question 2 : Utilisez des doublures pour tester que la méthode **`getMarketValue()`** de la classe **`Portefolio`** appelle bien la méthode **`getValue()`** de toutes les lignes incluent dans le portefeuille (tester avec un portefeuille ayant plusieurs lignes, une ligne, zéro ligne, …).

La méthode **`addStock(Stock s, int x)`** de la classe **`Portefolio`** ajoute **`x`** exemplaires de l'action **`s`** au portefeuille ; la méthode **`removeStock(Stock s, int x)`** retire **`x`** exemplaires de l'action **`s`** au portefeuille.

- Une exception est levée si **`x`** est un entier strictement négatif, ou si **`s`** est l'objet null.
- Si **`x`** est égale à 0 alors la méthode ne fait rien.
  Lla méthode **`removeStock(Stock s, int x)`** lève une exception de type **`IllegalArgumentException`** lorsque le portefeuille contient moins de **`x`** exemplaires de l'action **`s`**.
- La méthode **`removeStock(Stock s, int x)`** retire la ligne correspondant à l'action **`s`** du portefeuille, si le portefeuille contenait exactement **`x`** exemplaires de l'action **`s`**.

Questions 3 :

1. Listez les cas d'usage de la méthode **`addStock(Stock s, int x)`** de la classe **`Portefolio.`**
2. Utilisez une doublure ou des doublures de **`PortefolioLine`** pour tester de manière

exhaustive la méthode **`addStock(Stock s, int x)`** de la classe **`Portefolio`**.

3. Utilisez des doublures pour tester que la méthode **`addStock()`** de la classe **`Portefolio`** appelle bien la méthode **`getStock()`** de toutes les lignes jusqu'au moment au la ligne concernée est trouvée dans le portefeuille (tester avec un portefeuille ayant plusieurs lignes, une ligne, zéro ligne, …).
4. Listez les cas d'usage de la méthode **`removeStock(Stock s, int x)`** de la classe **`Portefolio`**.
5. Utilisez une doublure ou des doublures de **`PortefolioLine`** pour tester de manière exhaustive la méthode **`removeStock(Stock s, int x)`** de la classe **`Portefolio`**.