Devoir de spécification et validation du logiciel

Université Lille1 — Master 1 informatique, S2
UFR IEEA — partie Test SVL -1h30- 2014-2015 — FIL

Durée totale : **1h30**. Tous documents autorisés. Nombre de pages : **1** + 1 annexe.

Toutes les classes de test seront écrites en Python3 (inutile de donner les clauses `import`). Tous les tests seront unitaires et en isolation. Les mocks nécessaires seront décrits avec mockito. Si le cas d'étude vous semble sous-spécifié (ce qui fait partie de l'exercice), c'est à vous de faire un choix et de l'indiquer dans les tests. Si vous utilisez une classe fournie par Python et que vous avez oublié le nom exact des méthodes fournies par cette classe, vous pouvez utiliser un nom de votre cru, en précisant en une phrase le comportement de cette méthode. Les tests et le code doivent être soignés et **très clairs** (y compris l'écriture).

Vous devez répondre aux questions ci-dessous en simulant sur le papier une démarche type Test Driven Development : votre réponse doit donc être incrémentale. Pour chaque incrément sur votre copie, vous devez fournir :

1. un cas de test
2. le code le plus simple qui permet de faire passer ce test, en maintenant la couverture de code à 100%.

À chaque nouveau cas de test, votre code doit faire passer le nouveau cas de test, mais aussi tous les cas de tests précédents.

1. Vous devez ré-écrire à chaque fois l'intégralité du code, c'est à dire l'intégralité de la classe testée (inutile de donner les clauses `import`). Si une partie du code bien identifiée ne change pas d'un test à l'autre, par exemple le constructeur, vous pouvez vous contenter d'indiquer **clairement** que cette partie est présente sans changement.
2. Vous devez préciser en français si vous jugez nécessaire de modifier les tests précédents, et si c'est le cas indiquer en quelques mots les modifications à apporter. Il n'est pas nécessaire de ré-écrire les tests, ni de les annoter, une phrase suffit. En fin de copie vous pouvez indiquer par une simple phrase un refactoring qu'il vous semblerait judicieux d'effectuer.

Vous pouvez introduire dans la classe testée les accesseurs et constructeurs qui vous semblent nécessaires à l'écriture des tests, mais il n'est pas demandé de les tester.

Exercice 1 : Panier électronique

On s'intéresse à un panier électronique, du type de ceux utilisés lors d'achats sur un site web. Le panier mémorise les articles en cours d'achat en leur associant la quantité achetée. On aura par exemple un panier contenant 2 articles, l'un en quantité 2 et l'autre en quantité 1. Il n'est pas possible d'ajouter un article déjà présent dans le panier. Lors de l'ajout d'une quantité donnée x d'un nouvel article, le panier interroge un stock pour savoir si le stock contient bien cet article en quantité suffisante. Si tel est le cas, l'ajout est effectif : le panier contient désormais l'article avec la quantité x. Si tel n'est pas le cas, l'ajout échoue. En aucun cas l'ajout au panier ne diminue le stock.

Q 1.1 : Quels scénarios vous semblent judicieux pour tester/documenter la fonctionnalité d'ajout d'un article au panier ? □

Q 1.2 : Tester et coder la fonctionnalité d'ajout d'un article au panier, de la manière qui vous semble la plus judicieuse, et en suivant les instructions ci-dessus. Les `doctest` ne sont pas demandées. □

Q 1.3 : On veut faire évoluer le panier en lui associant des favoris qui mémorisent les articles qui ont transité par le panier. Expliquez en français comment vous procédez pour mettre à jour :
- votre base de tests, en donnant en Python l'oracle à utiliser
- votre code.

□

A La table associative de Python3 : `dict`

Un dictionnaire est une collection d'associations (clé, valeur) comme les `HashMap` de Java. Les clés doivent être d'un type hachable.
NB : les mocks de mockito sont hachables.

```
>>> d = dict()
>>> d
{}
>>> d["cle1"] = 2
>>> "cle1" in d
True
>>> d["cle1"]
2
>>> d
{'cle1': 2}
>>> "cle2" in d
False
>>> "cle2" not in d
True
>>> d["cle2"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'cle2'
>>> d["cle1"] = 50
>>> d
{'cle1': 50}
```