Devoir de spécification et validation du logiciel

Université Lille1 — Master 1 informatique, S2
UFR IEEA — partie Test SVL -1h30- 2016-2017 — FIL

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

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). Le code des exceptions et les `doctest` ne sont pas demandés.

Vous devez répondre aux questions ci-dessous en simulant sur le papier une démarche type Test Driven Development : votre réponse doit 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 : Caisse automatique avec pesée

Dans une grande enseigne, on peut payer ses achats à une caisse automatique. Pour chaque article à payer, le client de l'enseigne doit commencer par passer le code barre de l'article devant un détecteur, ce qui affiche à l'écran les caractéristiques de l'article, puis déposer l'article avec les autres dans un bac-balance. Les articles restent dans la balance jusqu'au paiement effectif. La caisse refuse de prendre en compte tout article pour lequel le poid délivré par la balance ne correspond pas au poid approximatif théorique de l'article.

On va réaliser une version très simplifiée de la classe qui contrôle les enchaînements des interactions de l'utilisateur avec la caisse, avec 2 fonctionnalités qui correspondent du point de vue du client à : 1) passer un article devant le détecteur et 2) poser un article dans le bac-balance :

1. quand le client passe l'article devant le détecteur, le contrôleur reçoit une référence unique issue du code barre, qui identifie l'article. ll interroge le système d'information pour récupérer ses caractéristiques, qui sont affichées à l'écran (erreur si la référence est inconnue). Le contrôleur attend ensuite la pesée de l'article, en refusant tout nouveau passage devant le détecteur ;
2. quand le client pose l'article dans la balance, celle-ci indique au contrôleur son poid. Le contrôleur interroge le système d'information pour vérifier que ce poid est cohérent (erreur si ce n'est pas le cas) et met fin à l'attente de pesée. Une erreur est aussi produite si le contrôleur n'attendait pas de pesée.

Q 1.1 : Donner les scénarios qui vous semblent judicieux pour tester/documenter le contrôleur de la caisse pour les 2 fonctionnalités de passage et pesée d'un article décrites ci-dessus. **Soigner la rédaction** : on doit identifier clairement dans un scénario le setUp et l'oracle qui en découlent. □

Q 1.2 : Tester et coder les 2 fonctionnalités du contrôleur de caisse, en vous basant de manière claire sur les scénarios de la question précédente, et en respectant les consignes du début de sujet. □