27/02/2017

AJ séance 5

Objectifs :

→ Etre capable d'écrire des tests unitaires avec les JUnit 4.
→ Comprendre et savoir comment organiser les tests unitaires.

Thèmes abordés :

- Annotations
- JUnits

Récupérez la solution des séances 2 et 3 dans le dossier seance3 sur claco. Créez un nouveau projet et configurez-le avec Eclipse afin qu'il contienne les 2 projets que vous avez récupérés dans son `Build Path`. Créez un package `tests` dans le nouveau projet.

Pour créer une classe de tests Junit pour la classe `ClasseAtester`, il faut respecter la marche à suivre suivante dans Eclipse :

1. Sélectionnez la classe `ClasseAtester`, faites un clic droit click sur New et demandez la création d'un JUnit test case.
2. Changez le répertoire source et indiquez celui du projet de la semaine (Browse)
3. Indiquez le package `tests` (Browser).
4. Vérifiez que le nom de la classe créée est bien `ClasseAtesterTest` et que la classe testée est `ClasseAtester` (Class under test).
5. Cochez `setUp`() (Which method …)
6. Appuyez sur Next
7. Sélectionnez les méthodes de la classe `classeAtester` pour lesquelles vous voulez écrire des tests.
8. Acceptez d'inclure la librairie JUnit 4 au projet.

Dans un premier temps, nous testons les objets du domaine ensuite nous effectuerons des tests sur le use cas controleur.

Tests de la classe Objet

Préparation du test

Afin de réaliser les tests, vous allez créer des attributs qui seront instanciés dans la méthode annotée avec `@Before` : créez d'abord trois utilisateurs. À quoi cela sert-il ? (réponse : voir la théorie ☺)

Ensuite, créez 3 objets mis en vente par le premier utilisateur. Le premier objet n'a pas encore d'enchère, le deuxième possède trois enchères faites par les deux autres utilisateurs tandis que le troisième objet sera déjà vendu au deuxième utilisateur. Pour créer ces objets, vous devez utiliser directement les classes du domaine sans passer par le use case. Pour les tests qui suivent, évitez le plus possible de recréer de nouveaux objets en utilisant ceux qui sont créés dans la méthode annotée @Before.

N'oubliez pas que chaque test doit se trouver dans une méthode annotée `@Test` ; nous vous recommandons de numéroter les tests de façon concordante aux numérotations proposées dans cet énoncé. Par exemple les tests du constructeur de la classe `Objet` sont `testObjet1` ; `testObjet2`, `testObjet3`, …

Test du constructeur

1. Vérifier que le constructeur lance une `NullPointerException` si la description passée en paramètre est `null`.
2. Vérifier que le constructeur lance une `IllegalArgumentException` si la description passée en paramètre est une chaîne de caractères constituée uniquement de blanc.
3. Vérifier que le constructeur lance une `IllegalArgumentException` si l'utilisateur passé en paramètre est `null`.

4. Vérifiez que les numéros d'objets sont bien attribués de façon séquentielle.

Exécutez fréquemment vos tests avec Run as JUnit test.

Test des getters

1. Vérifier que la description correspond bien à celle passée en paramètre au constructeur.
2. Vérifier que le vendeur correspond bien à celui passé en paramètre au constructeur.
3. Vérifier que le getter du vendeur renvoie un clone du vendeur.

Test des enchères

1. Vérifier qu'une enchère est automatiquement ajoutée à l'objet pour lequel elle est créée.
2. Vérifier qu'on ne peut pas créer pour un objet une enchère antérieure à la dernière enchère.
3. Vérifier qu'on ne peut pas créer pour un objet une enchère au même moment que la dernière enchère.
4. Vérifier qu'on ne peut pas créer pour un objet une enchère d'un montant inférieur à celui de la dernière enchère.
5. Vérifier qu'on ne peut pas créer pour un objet une enchère d'un montant égal à celui de la dernière enchère.
6. Vérifier qu'on ne peut plus créer d'enchère pour un objet déjà vendu.
7. Vérifier que la méthode `meilleureEnchere` renvoie null s'il n'y a pas encore d'enchère.
8. Vérifier que la méthode `meilleureEnchere` renvoie la bonne enchère.
9. Vérifier que la méthode `prixDeVente` renvoie bien 0 si l'objet n'a pas encore d'enchère.
10. Vérifier que la méthode `prixDeVente` renvoie bien 0 si l'objet a des enchères mais qu'il n'a pas encore été vendu.
11. Vérifier que la méthode `prixDeVente` renvoie bien le bon prix quand l'objet a été vendu.
12. Vérifier que la méthode `encheres` renvoie la bonne liste d'enchères.
13. Vérifier que la liste renvoyée par la méthode enchère n'est pas modifiable.

Test de la méthode clone

1. Vérifier que la méthode renvoie bien un objet qui est égal à l'objet cloné mais pas le même !
2. Vérifier que la description, le vendeur et les enchères du clone correspondent bien à ceux de l'original.
3. Construire un clone de l'objet sans enchère et ajoutez-lui une nouvelle enchère. Vérifier que la nouvelle enchère ne se trouve pas dans l'objet cloné et bien dans le clone.

Test de la méthode hashCode

1. Vérifier que deux objets égaux ont le même `hashCode`.

Tests de la classe Enchere

Test de la méthode getEncherisseur

1. Vérifier que l'enchérisseur correspond bien à celui passé en paramètre
2. Vérifier que la méthode `getEncherisseur` renvoie un clone de l'enchérisseur.

Test de la méthode equals

1. Vérifier que deux enchères faites au même moment par le même enchérisseur sont égales.
2. Vérifiez que deux enchères faites à des moments différents par le même enchérisseur sont différentes.
3. Vérifiez que deux enchères faites au même moment par des enchérisseurs différents sont différentes.

Test de la méthode hashCode

Vérifier que deux objets égaux ont le même `hashCode`.

Pour lancer l'exécution de plusieurs classes de tests en même temps, il suffit de créer une JUnit Test Suite et sélectionner les classes de tests à exécuter en une fois.

Tests de la classe GestionEncheres

Test de la méthode mettreEnVente

1. Vérifier que la méthode `mettreEnVente` lance une `UtilisateurInexistantException` si l'utilisateur n'est pas inscrit.
2. Vérifier que la méthode `mettreEnVente` renvoie un clone de l'objet.
3. Vérifier que l'objet mis en vente est bien dans la liste des objets en vente.

Test de la méthode encherir

1. Vérifier que la méthode `encherir` lance une `UtilisateurInexistantException` si l'utilisateur n'est pas inscrit.
2. Vérifier que la méthode `encherir` lance une `ObjetInexistantException` si l'objet n'est pas en vente.
3. Vérifier que la méthode renvoie `null` s'il existe déjà une enchère pour l'utilisateur au même moment mais pour un autre objet.
4. Vérifier qu'il n'y a pas eu d'enchère ajoutée à l'objet s'il existe déjà une enchère pour l'utilisateur au même moment mais pour un autre objet.
5. Vérifier que la méthode renvoie `null` si l'enchère n'a pas pu être créée (par exemple, si le montant est inférieur au montant de la dernière enchère pour cet objet).
6. Vérifier que l'enchère a bien été ajoutée à la liste des enchères de l'objet.
7. Vérifier que l'enchère a bien été ajoutée à la liste des enchères du jour de l'enchère.
8. Vérifier que la méthode `encherir` lance une `ObjetInexistantException` si l'objet a déjà été vendu.

Test de la méthode accepter

1. Vérifier que la méthode `accepter` lance une `EnchereInexistanteException` s'il n'y a pas encore d'enchère pour l'objet.
2. Vérifiez que la méthode renvoie `true` si tout se passe normalement.
3. Vérifier que l'objet a bien été ajouté dans les objets achetés par celui qui a fait la meilleure enchère.
4. Vérifier que l'objet est bien enlevé de la liste des objets en vente.
5. Vérifier que l'objet a été ajouté à l'ensemble des objets vendus.

Si vous avez encore le temps, testez la classe `Utilisateur` ainsi que les autres méthodes de la classe `GestionEncheres`.