R4.02 - Qualité de développement
TP4

Semestre 4
2022/2023

Le code pour ce TP est disponible sur Gitlab :

`https://gitlabinfo.iutmontp.univ-montp2.fr/r402/tp4/`

La classe `MockitoDemoBasics` illustre l'utilisation du framework Mockito pour créer des doublures. La classe `MockitoDemoInjection` illustre son utilisation pour injecter ces doublures dans des objets à tester (vous aurez besoin de ces fonctionnalités uniquement pour le dernier exercice).

Application à tester

Dans ces exercices, nous allons tester les classes du package `yahtzee`. Le Yahtzee [1] est un jeu de dés semblable au poker. À chaque tour, un joueur lance cinq dés pour tenter de former une figure parmi 13 possibles (petite suite, carré, full...). Un joueur a au maximum trois lancers de dés par tour. Pour le deuxième et troisième lancer, il peut choisir de relancer tous ses dés ou seulement quelques-uns. À la fin du tour, le joueur doit enregistrer sa combinaison comme une des figures possibles. Cette figure est alors rayée, et ne pourra pas être utilisée par la suite. La partie s'arrête après 13 tours, lorsque plus aucune figure n'est disponible sur les feuilles des joueurs.

La classe `Turn` modélise un tour de jeu, la classe `Dice` modélise un dé, et l'interface `PlayerI` correspond aux joueurs. La classe abstraite `Combination` modélise les figures, et chacune de ses sous-classes représente un type de figure particulier. Pour l'instant, seules deux figures sont implémentées : `FullHouseCombination` et `SixesCombination`.

Défauts

Vos tests peuvent révéler des défaillances. La documentation des classes et les règles du jeu forment la spécification de l'application. Dans un document,

1. `https://fr.wikipedia.org/wiki/Yahtzee`

pour chaque défaillance observée :

1. indiquez le test qui illustre la défaillance ;
2. trouvez le défaut qui provoque la défaillance ;
3. proposez un correctif (mais ne l'implémentez pas ! Les classes à tester ne doivent *pas* être modifiées).

1 Créer et utiliser une doublure

Dans ce premier exercice, nous voulons tester la classe `Turn`. La création d'une instance de `Turn` requiert un objet qui implémente l'interface `PlayerI`. À ce stade du développement, aucune classe satisfaisant cette interface n'a été implémentée, il faut donc utiliser Mockito pour créer un simulacre qui implémente l'interface. Utilisez ce simulacre pour créer une instance de `Turn`, et réalisez les tests suivants :

1. testez tous les cas d'utilisation de la classe `Turn` qui peuvent mener à une exception `IllegalStateException`, et vérifiez que l'exception est bien lancée [2] ;
2. testez un scénario de tour normal (sans exception) qui se conclut par un appel à la méthode `end` avec l'argument `FULL_HOUSE`. Testez que la méthode `addCombinationToScoreSheet` du simulacre est bien appelée, et que son argument est du type attendu.

2 Programmer une doublure

Dans ce deuxième exercice, nous voulons tester la création d'instance de `Combination`, qui dépend des valeurs affichée par les dés. Puisque le comportement de la classe `Dice` est aléatoire, il est difficile de l'utiliser pour les tests. À la place, nous allons utiliser des doublures de cette classe.

1. créez un simulacre à partir de la classe `Dice` et programmez-le pour retourner une valeur de votre choix ;
2. créez et programmez cinq simulacres à partir de la classe `Dice`, puis utilisez-les pour créer des instances des classes `FullHouseCombination` et `SixesCombination` ;
3. vérifiez que la valeur de `getScore` retournée par ces instances est conforme à la spécification.

2. pour l'un de ces cas, vous devrez programmer le simulacre afin que sa méthode `addCombinationToScoreSheet` réagisse comme si la combinaison avait été déjà été ajoutée à la feuille de score. Voir la documentation de l'interface `PlayerI` qui décrit la réaction attendue.

3 Injecter des doublures

Nous allons à présent utiliser l'injection de dépendance pour remplacer l'attribut `dice` d'une instance de `Turn`, afin de surveiller son comportement et contrôler son contenu.

1. créez un espion de la classe `Vector<Dice>` et injectez-le dans une instance de `Turn`. Attention, ce vecteur de dés injecté est vide. Pour lancer un test sur l'instance de `Turn`, vous devrez au préalable ajouter cinq instances de `Dice` dans le vecteur ;
2. établissez un cas de test qui simule un tour de jeu et fait appel aux différentes méthodes publique de la class `Turn`. À la fin de ce scénario, vérifiez que l'espion a été utilisé comme prévu : sa méthode `add` doit avoir été appelée cinq fois, et ses méthodes `set` et `remove` ne doivent pas avoir été appelées.

Dans une autre série de tests, vous allez créer des espions de la classe `Dice` et les utiliser pour tester les méthodes `selectForRoll` et `unselectForRoll` de la classe `Turn`.

3. créez des espions de la classe `Dice` et ajoutez-les au vecteur de dés injecté (au lieu des dés normaux) ;
4. établissez des cas de tests qui simulent des tours de jeu, en utilisant les méthodes `selectForRoll` et `unselectForRoll` pour déterminer quels dés sont lancés à chaque fois ;
5. vérifiez que les dés sont lancés conformément aux attentes, en contrôlant le nombre d'appels à la méthode `roll` sur les dés espions.