M1 Informatique - Ingénierie Logicielle
TD-TP No 1 - séance 4 ou 5 (selon avancement)

Tests

1 Première classe de test simple pour `OrderedDictionary`

Mettre en place une première classe de test simple pour la classe `OrderedDictionary`.
JUnit est installé en standard dans Eclipse.

1.1 Création de la classe de test

Dans votre projet :

- clic droit → new JUnit test case
- Choisissez bien JUnit 5 (jupiter) et pas JUnit 4 et encore moins 3
- Nommez votre classe de test. Cochez la case permettant de générer la méthode `setUp`. Ne sélectionnez aucune autre option.

1.2 Environnement de test

Dans la classe test nouvellement créée, déclarez un attribut de type `OrderedDictionary`. Créez-le dans la méthode `setUp` qui a été générée. Vous noterez que cette méthode est annotée par l'annotation `@Before`, c'est donc une méthode de préambule qui sera appelée avant chaque méthode de test.

1.3 Une première méthode de test

Ajoutez à votre classe une première méthode de test `testAddOneElementToEmptyDico()`. N'oubliez pas d'ajouter l'annotation `@Test` à cette méthode, afin qu'elle soit bien considérée par `JUnit` comme une méthode de test. Dans le corps de cette méthode, écrivez le code permettant d'ajoutez le couple (clef-valeur) de votre choix. Puis, vérifiez que tout s'est bien passé :

- Vérifiez que la taille du dictionnaire est bien 1 (par une assertion du type `assertEquals(1, dico.size()`)
- Vérifiez que l'élément ajouté existe bien dans votre dictionnaire (par une assertion du type `assertTrue(dico.co`
- Vérifiez que vous pouvez bien retrouver l'élément ajouté dans votre dictionnaire.

Exécutez votre test (`run as JUnit application`).

1.4 Fin de la classe de test pour OrderedDictionary

Complétez le test de la classe `OrderedDictionary` en ajoutant autant de méthodes de test qu'il vous semble nécessaire. Corrigez les erreurs que vous détecterez au fur et à mesure.

2 Test des autres dictionnaires

Testez les 2 autres types de dictionnaires. Vous vous rendrez compte que de nombreuses méthodes de test sont identiques à celles de la classe de test pour `OrderedDictionary`. Il pourra être intéressant de créer une super-classe encapsulant ces tests.
Finalement, créez une suite de test permettant d'enchaîner l'exécution des tests des 3 classes de test.

3 Couverture de code

Quand vous pensez avoir terminé vos tests, ré-exécutez-les, cette fois avec `run as Coverage Application`.

`Coverage` est un outil provenant du plugin Eclipse `ECLEmma`, qui permet d'analyser le code couvert par les tests, c'est-à-dire de déterminer quelles parties du code sont exécutées lors de l'éxécution des tests, et quelles parties ne le sont pas. On obtient une mesure de la couverture de code. Attention, avoir 100% de couverture de code ne signifie pas nécessairement d'avoir correctement testé votre code. Si `Coverage` n'est pas disponible sur la version d'Eclipse que vous utilisez, installez le plug-in `ECLEmma` depuis le marketplace d'Eclipse.

Observez les parties de votre code couvertes et non couvertes. (Re-)Testez en conséquence.

4 Test d'une autre version de dictionnaire

Les tests croisés sont une pratique industrielle courante. Utilisez maintenant vos tests pour tester une autre version de dictionnaires que la vôtre, par exemple celle disponible sur la page suivante :
`http://www.lirmm.fr/~dony/enseig/IL/TP-Dictionnaires-Atester`.

Question 1. Cette version est définie dans un package `withBug` ; vous pouvez la chargez en même temps que la votre. Elle contient donc des bugs, saurez vous trouver lesquels ?

5 Utilisation de PowerMock et Mockito pour les dictionnaires

Si vous le souhaitez vous pouvez faire des tests plus sophistiqués avec les packages sus-nommés.

Si nécessaire Téléchargez à cette adresse : `https://github.com/jayway/powermock/wiki/Downloads` le fichier :
`powermock-mockito2-junit-1.7.1.zip`

Placez les jar contenus dans le zip dans votre buildpath (dans le projet Eclipse dans lequel vous allez travailler).

Application aux dictionnaires

Question 2. Certaines situations étaient difficiles à faire apparaitre lors du test des dictionnaires (comme par exemple le conflit de hachage et la position d'un tel conflit en fin de tableau pour tester la recherche circulaire). Reprenez avec PowerMock et Mockito vos tests sur les dictionnaires de manière à utiliser les mocks pour faire apparaitre ces situations artificiellement. Vous serez certainement amenés pour cela à refactoriser une partie de votre code pour le rendre plus facilement testable.