Test unitaire en Java

Exercice 1 *Mise en jambe : JUnit, testabilité et exhaustivité*

On cherche à tester la classe SUT (System Under Test). On donne la classe TestFooBar. La spécification des méthodes est donnée en commentaires dans le code.

Question 1. Imaginons que l'on exécute avec JUnit la classe TestFooBar. Quelles seraient les méthodes appelées et dans quel ordre ?

Question 2. On exécute maintenant réellement la classe TestFoo avec JUnit.

a- Etudiez les verdicts émis par la méthode testLouche. Expliquez. Qu'en déduisez-vous ?

b- Etudiez les méthodes testFooInitParDefaut (de 1 à 4), ainsi que les verdicts qu'elles émettent. Qu'en déduisez-vous ?

Question 3. On s'intéresse au test de la méthode bar.

a- Que pensez-vous de la méthode de testBar ?

b- Essayez de concevoir une méthode de test plus pertinente. Quel problème rencontrez-vous ? Que pensez-vous de la testabilité de la classe SUT ?

Question 4. On s'intéresse maintenant à la classe TestParametreFoo.

a- Imaginons que l'on exécute cette classe avec JUnit. Quelles méthodes seraient exécutées ?

b- A votre avis, quel est l'objectif de test de cette classe ?

c- Créez 3 autres classes de test paramétrés selon le même principe en variant légèrement l'objectif de test.

d- Avez-vous ainsi réalisé un test exhaustif de la méthode foo ?

Question 5. Ecrivez dans une nouvelle classe de test de quoi tester la méthode foobar.

Question 6. Créez maintenant une suite de test qui exécute tous les tests écrits. Exécutez-là.

Question 7. Corrigez l'erreur dans la classe SUT, ré-exéutez vos tests et vérifiez qu'ils ne détectent plus de faute, à part testLouche.

Exercice 2 *Test d'un système de construction de parcours touristiques*

Le syndicat d'initiative d'une municipalité met en place des parcours de visite. Les parcours ne sont pas des visites guidées mais des propositions pour les touristes afin de leur faciliter la visite de la ville. On met en place le logiciel permettant de gérer ces parcours. Ce logiciel doit permettre au personnel du syndicat d'initiatives de créer de nouveaux parcours (ce qui nécessite d'en définir les étapes et les itinéraires entre les étapes à partir d'un ensemble d'étapes et d'itinéraires connus, mais qui peut aussi impliquer de créer de nouvelles étapes et de nouveaux itinéraires entre étapes), de parcourir les parcours existants, et d'exporter en format pdf un parcours. Des éléments de modélisation vous sont fournis sous forme du diagramme de classes partiel de la figure 1. On y trouve les notions de parcours, qui se compose d'étapes et de tronçons, un tronçon représentant l'itinéraire entre une étape de départ et une étape d'arrivée. Une étape a un nom et une durée de visite qui se compte en minutes. Chaque étape a un type qui représente la nature de l'étape. Le syndicat d'initiative a défini quatre "natures" possibles : musée, visite de monument, visite de jardin, ou lieu d'intérêt sans visite. Un tronçon est décrit par les noms des rues qui le composent, et un temps de trajet moyen, en minutes.

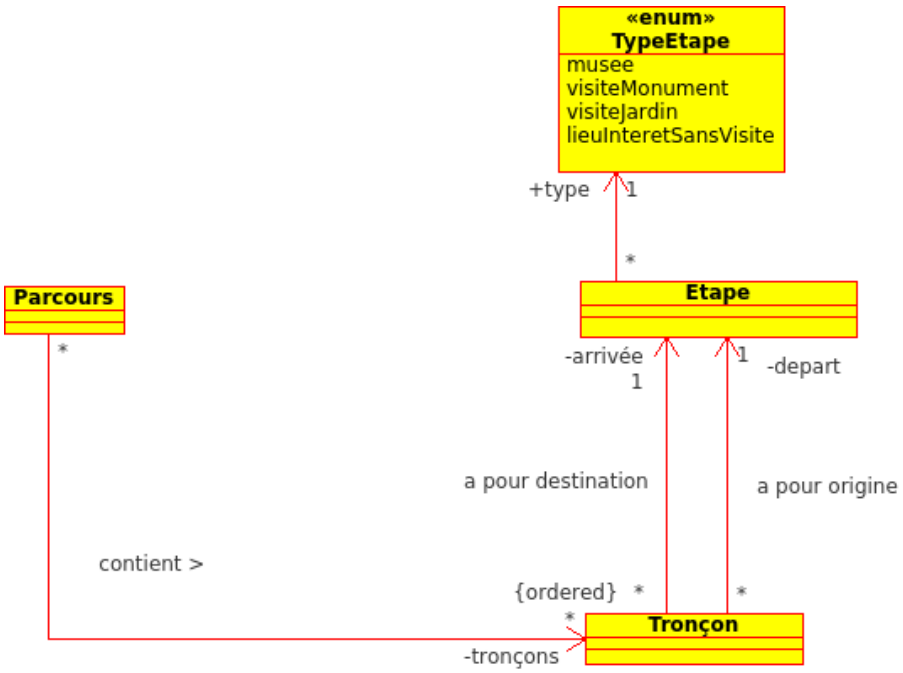

FIGURE 1 – Diagramme de classes partiel

On travaille ici sur une réalisation très partielle de ce système, en Java.

Question 8. Sur machine. Testez l'implémentation du système qui vous est donnée. Corrigez les erreurs au fur et à mesure que vous les trouvez.

Exercice 3 *Test d'une hiérarchie de classes*

Question 9. Sur machine. Réécrivez en JUnit vos tests pour l'exemple des fichiers musicaux et des playlists. Vous devrez gérer correctement les méthodes de type @Before et @After vis à vis des fichiers créés ou modifiés lors des tests.