TD3 – Testabilité – Dépendance – Doublure de tests (Jean-Marie Mottu)

Première partie – Cas d'étude : classe Chien

Nous travaillons sur le cas d'étude d'une classe Chien que vous avez déjà aperçue en POO mais qui a été redéveloppée pour ce TD. Vous allez progressivement vous rendre compte de problèmes de testabilité, de la difficulté de gérer des dépendances quand on teste un système, de la nécessité d'implémenter des doublures de test.

De nouveau, on considère une approche fonctionnelle en n'exploitant que la spécification pour concevoir les tests. Ici considérons toutes les informations ci-dessous :

- la première ligne de la javadoc spécifie les « exigences »
- les @ de la javadoc complété par la signature des méthodes spécifient le domaine d'entrée.

```
1   package but1.iut.r203.chenil
2
3   import ...
5
6   class Chien (nomParam : String, raceParam : String){
7       val nom = nomParam
8       val race = raceParam
9       private var dateNaissance : LocalDate? = null
10          set(value) {...}
13
14      /**
15       * Affecte une date de naissance au chien
16       * @param anneeNaissance: Int
17       * @param moisNaissance: Int
18       * @param jourNaissance: Int
19       * @throws IllegalArgumentException : quand les paramètres sont incorrectes
20       */
21      fun setDateNaissance(anneeNaissance: Int, moisNaissance: Int, jourNaissance: Int) {...}
24
25      /**
26       * Calcule l'age en mois du chien
27       * @return age : Long
28       */
29      fun ageMois(): Long {...}
35
36  }
```

Le but est d'implémenter ces quelques cas de test :

CT1 : 10 mois entre le (2021, 2, 28) et le (2022, 1, 1)
CT2 : 0 mois entre le (2021, 12, 31) et le (2022, 1, 1)
CT3 : 1 mois entre le (2021, 12, 1) et le (2022, 1, 1)
CT4 : 12 mois entre le (2021, 2, 15) et le (2022, 2, 28)
CT5 : 12 mois entre le (2021, 2, 15) et le (2022, 2, 15)
CT6 : 11 mois entre le (2021, 2, 15) et le (2022, 2, 1)

La première date de chaque cas de test est la date de naissance du chien. La deuxième date est la supposée date du jour.
On remarque donc que :

- les 3 premiers cas de test font varier la date de naissance. La date du jour est fixe mais ce n'est pas la date réelle d'aujourd'hui.
- les 3 derniers cas de test font varier la date du jour. La date de naissance est fixe.

Récupérer le code dans IntelliJ depuis ce projet gitlab :
https://univ-nantes.io/iut.info1.qd1.automatisationtests/butinfo1-qd1-td3

Exercice 1 :

Question 3.1 : Commencez par lancer le Main du package Chenil :
Quel défaut constatez-vous ?
Pouvez-vous modifier `println(ch1)` pour afficher la date de naissance du chien ?

Question 3.2 : Quel est ce problème de testabilité ?

Question 3.3 : Une fois ce problème identifié, le testeur demande au développeur s'il peut améliorer définitivement le design du logiciel pour en améliorer la testabilité. Faites-le : en bas de la classe Chien « Generate… > toString() ». Relancez et observer que le problème est résolu.

Exercice 2 :

Question 3.4 : Toujours dans le Main du package Chenil : on y entre bien les dates de naissance de 2 chiens correspondant au CT1 et CT4. Si on implémentait les cas de test correspondant, pourrait-on paramétrer la date du jour choisie dans les cas de test ?

Question 3.5 : Quel est ce problème de testabilité ?

Exercice 3 :
Notre objectif est de permettre de substituer cette dépendance par une doublure de test dont on aura la maitrise (en lui faisant renvoyer les dates qui nous nécessaire aux tests)
Mettons en place en trois temps un mécanisme « d'injection de dépendance ».

Question 3.6 : Déportons la dépendance dans une classe. On maitrisera cette classe contrairement à la librairie (qui est définitivement indépendante à notre volonté).

Cette classe est déjà dans les sources que vous avez clonées, avec une interface. Changez la ligne de code 30 pour utiliser ces interfaces et classe.

Question 3.7 : Créons une première doublure de cette classe. Faites un stub : une autre classe implémentant l'interface DateChenil qui renvoie une réponse prédéfinie (2022, 1, 1)

Dans la méthode ageMois, n'instanciez plus DateChenilSysteme mais DateChenilStub. Lancez le test, obtenez-vous le comportement attendu ?

Par contre, avez-vous respecté la bonne pratique de l'exercice 1 ?

Question 3.8 : Ajoutez un mécanisme d'injection de dépendance, cf CM.

Exercice 4 :
Question 3.9 : Implémentez les 3 premiers cas de test dans une classe de test.

Pour ces 3 premiers cas de test, la date du jour est donc maîtrisée, mais on ne la double qu'une fois. Les 3 cas de test suivants ont besoin d'une date du jour différente à chaque fois : cela serait laborieux avec trois stubs.

Exercice 5 :

Question 3.10 : Implémentez les trois derniers cas de test dans une classe de test avec des mocks.

Deuxième partie – Cas d'étude : Opérations Dépendantes

On reprend notre cas d'étude des opérations. On considère à nouveau la division et la factorielle, mais cette fois elles vont chercher leurs opérandes dans la console : l'utilisateur (le testeur !) doit les taper à chaque fois.

Exercice 6 :

Question 3.11 : Est-ce une bonne pratique de test ?

Question 3.12 : Reprenez les tests de ces 2 méthodes conçus au TD2 et réimplémenter une classe de test qui va donc demander en console les valeurs.

Ce n'est tellement pas une bonne pratique, que les tests n'attendent pas qu'on écrive en console pour avancer et échouer :

https://intellij-support.jetbrains.com/hc/en-us/community/posts/115000556544-Why-can-t-I-input-anything-from-console-when-i-run-unit-test-with-JUNIT

Exercice 7 :

Question 3.13 : Mettez en place un mécanisme complet d'injection de dépendance, comme fait dans la première partie.

Question 3.14 : Adapter les cas de tests pour utiliser des mocks qui renverront les valeurs nécessaires aux tests plutôt que de les lire en console.

Troisième partie – Subsidiaire

Exercice 8 :

La méthode `setDateNaissance` ne gère pas elle-même ses levées d'exception, mais le sous-traite à la classe LocalDate. Pourtant c'est demandé dans la spécification :

```
@throws IllegalArgumentException : quand les paramètres sont incorrectes
```

Question 3.15 : Concevez et implémentez les tests fonctionnels de cette méthode. On n'ira pas avant 1900 (ce n'est déjà plus l'âge d'aucun humain, alors d'un chien…)

Question 3.16 : Corrigez le code en conséquence.