TD1 – Test fonctionnel – Dépendance – Doublure de tests - Diagnostic (Jean-Marie Mottu)

Cas d'étude : réveil AlarmClock

Nous travaillons sur le cas d'étude d'une classe qui implémente un réveil. Vous allez concevoir des tests fonctionnels par analyse partitionnelle. Puis vous allez implémenter les tests en vous rendant compte de problèmes de testabilité nécessitant d'améliorer l'implémentation pour effectuer les tests avec l'aide de doublures de test.

On considère une approche fonctionnelle en n'exploitant que la spécification pour concevoir les tests. Ici considérons toutes les informations synthétisées dans l'interface de ce système :

https://gitlab.univ-nantes.fr/iut.info2.qd3.conceptiontests/butinfo2-qd3-td1-students/-/blob/main/src/main/kotlin/fr/nantes/univ/clock/AlarmClock.kt

Exercice 1.1. Découverte du cas d'étude

Question 1.1.1. Faites une lecture rapide de l'interface (10 minutes)

- les premières lignes de la javadoc spécifie les « exigences »
- certains @xxx de la javadoc complété par la signature des méthodes spécifient le domaine d'entrée.

Première partie – Conception de tests fonctionnels par Analyse Partitionnelle

Pour effectuer le test fonctionnel par analyse partitionnelle, on réalise plusieurs étapes :

Etape préliminaire pour la classe : considérant une classe (et pas juste une fonction comme l'an passé), il faut d'abord identifier les états valides du système. En effet, en entrée et en sortie de chaque test, il faut que le système soit dans un état valide.

Etapes, méthode par méthode :

1. Identifiez les **variables** qui forment chaque Donnée de Test, ainsi que les variables qui seront contrôlées par l'Oracle.
2. Réalisez pour chacune des variables formant les Données de Test une **analyse partitionnelle**, afin d'en déduire des classes d'équivalences.
    a. identifiez le **type** de la variable
    b. identifiez la **plage** de la variable
        i. Des intervalles **nominaux**
        ii. La/les plages de valeurs du fonctionnement **exceptionnel**
    c. identifiez des partitions **fonctionnelles**
3. Établissez une **table de décision** décrivant le comportement attendu.
4. Déduisez un ensemble fini de **Cas de Test**.

Exercice 1.2. Conception des tests fonctionnels d'AlarmClock par analyse partitionnelle

Question 1.2.1. Quels sont les états valides d'une instance d'AlarmClock (10 minutes)

Question 1.2.2. Concevez les tests fonctionnels du constructeur (20 minutes)

Il faut davantage d'information de spécification que ce qu'il y a dans l'interface, le reste est dans la factory :
https://gitlab.univ-nantes.fr/iut.info2.qd3.conceptiontests/butinfo2-qd3-td1-students/-/blob/main/src/main/kotlin/fr/nantes/univ/clock/AlarmClockFactory.kt

Question 1.2.3. Concevez les tests fonctionnels de la fonction selectRing() (20 minutes)

Question 1.2.4. Concevez les tests fonctionnels de la fonction checkTimeAndRing () (20 minutes)

Deuxième partie – Implémentation de tests sans dépendances

Exercice 1.3. Implémentation des tests fonctionnels du constructeur

(les cours de QDev1-R2.03 sont dans une section du cours madoc R402, le TD1 peut être utile)

Question 1.3.1. Récupérez le code dans IntelliJ en clonant ce projet gitlab : https://gitlab.univ-nantes.fr/iut.info2.qd3.conceptiontests/butinfo2-qd3-td1-students (5 minutes)

Question 1.3.2. Nous allons tester l'implémentation de l'interface faite par la classe BasicAlarmClock. Créez sa classe de test : (2 minutes)

Attention à la procédure sous IntelliJ :

1. sur le nom du projet, faire « New », « Directory », « src/test/kotlin »
2. dans la classe sous test « BasicAlarmClock », clic-droit sur le nom de la classe (ligne 8 du code), « Generate… », « Test… », sélectionner les méthodes qu'on va tester (checkTimeAndRing, le constructeur implicite en kotlin n'est malheureusement pas listé), dans le source root test.(il faut peut-être décocher « Show… » pour la voir)

Question 1.3.3. Implémentez le test nominal du constructeur. (8 minutes)

Question 1.3.4. Implémentez les tests exceptionnels du constructeur avec des tests paramétriques. (15 minutes)

(cf le CM2 et le TD1 de R2.03)

Troisième partie – Implémentation de tests avec dépendances

Exercice 1.4. Testabilité

Question 1.4.1. Quel problème de testabilité a-t-on pour implémenter les tests de checkTimeAndRing () ? (10 minutes)

(cf CM3 de R2.03)

Question 1.4.2. Améliorez l'implémentation de la classe pour résoudre ce problème. (10 minutes)

(cf les slides 17-18 du CM3 de R2.03)

Exercice 1.5. Implémentation des tests fonctionnels de checkTimeAndRing ()

Question 1.5.1. Implémentez le test sans dépendance (10 minutes)

Question 1.5.2. Implémentez les tests avec dépendance en utilisant Mockk (10 minutes)

(cf la section Mockk du CM3 de R2.03)

Quatrième partie – diagnostic

Exercice 1.6. Diagnostic de checkTimeAndRing () (10 minutes)

Question 1.6.1. Quels tests échouent ?

Question 1.6.2. Diagnostiquez pourquoi et corrigez la faute.

(cf le CM4 de R2.03)

Cinquième partie – subsidiaire

Exercice 1.7. Implémentation des tests de selectRing()

Question 1.7.1. Implémentez les tests.

Question 1.7.2. Diagnostiquez le test qui échoue et corrigez la faute.