TD1 – Test unitaire en JUnit (Jean-Marie Mottu)

Première partie – Tutoriel pour la création et le lancement de classe de test JUnit sous IntelliJ.

Dans la continuité du cours de POO, nous allons tester des programmes codés en kotlin en utilisant l'IDE Intellij. M. Lanoix a préparé tout un tutoriel que je vous encourage à relire :
https://gitlab.univ-nantes.fr/iut.info1.dev.objets/dev.objets.tutoriel.intellij.idea

Concernant ce TD1, deux pages de ce tutoriel vont être particulièrement utiles.

Question 1.1 : suivez le tutoriel suivant

https://gitlab.univ-nantes.fr/iut.info1.dev.objets/dev.objets.tutoriel.intellij.idea/-/blob/main/tuto/git.md

pour cloner le dépôt git suivant :

https://univ-nantes.io/iut.info1.qd1.automatisationtests/butinfo1-qd1-td1

Cela va vous créer un dossier `main` mais pas de dossier `test`, ce qui est nécessaire :
Sur `src`, faites un clic droit, `new Directory`, `test/kotlin`.

Remarque : vous n'aurez pas le droit de pusher sur ce dépôt, ce qui est ennuyeux pour versionner et sauvegarder votre travail. Vous pourriez faire un fork du projet, cf cours R2.10 – GPO2)

Nous travaillons sur le cas d'étude d'une classe d'Operations pour faire des calculs sur des tableaux de nombres et des couples de nombre :

```
package but1.iut.r203

/**
 * @author mottu-jm (inspired from rpouiller's java code)
 */
class Operations {

    private val COURT = 0
    private val LONG = 1
    private val TRESLONG = 2

    var cf: ConsoleFactorielle = ConsoleFactorielle()

    /**
     * additionner une liste de int
     * @param nombres : int[] tableau d'entiers à additionner
     * @return somme : int
     */
    fun additionner(nombres: Array<Int>): Int {
…    }

    /**
     * Soustraire successivement des int (requiert au moins 2 nombres)
     * @param nombres : int[] tableau d'entiers à soustraire
     * @return resultat des soustractions
     */
    fun soustraire(nombres: Array<Int>): Int {
…    }

    /**
     * Multiplier une liste de int
     * @param nombres : int[] tableau d'entiers à multiplier
     * @return produit : int
     */
    fun multiplier(nombres: Array<Int>): Int {
…    }


    /**
     * Affiche le resultat de l'operation de differentes manieres
     * (de COURT à TRESLONG pour le parametre de l'afficheur)
     * @param resultat : nombre à afficher
     * @param operation : opération effectuée
     * @param afficheur : constante déterminant la longueur de l'affichage
     */
    private fun afficherResultat(resultat: Int, operation: String, afficheur: Int) {
…    }

}
```

Question 1.2 : Programmez le premier cas de test suivant en créant une classe de test en vous inspirant de cette page de tutoriel :

https://gitlab.univ-nantes.fr/iut.info1.dev.objets/dev.objets.tutoriel.intellij.idea/-/blob/main/tuto/test.md

CT(« addition de 1 et 2 », {[1,2]}, {3})

Remarque : vous aurez peut-être le message « No tasks available » en essayant de lancer le test (et il ne se passera rien). Dans ce cas : Settings > Build, Execution, Deployment > Build Tools > Gradle et changer Run tests using: de Gradle (Default) par IntelliJ IDEA.

Question 1.3 : Quels éléments composent le précédent cas de test. Quel autre élément peut être nécessaire dans un cas de test.

Deuxième partie – Programmation des suites de cas de test

Question 1.4 : Programmez ces huit tests dans 8 fonctions séparées :

CT1(« addition de 5 et 5 », {[5,5]}, {10}) // implémenté par testAdditionner1( )

CT2(« addition de 5 et 5 et 5 », {[5,5,5]}, {15}) // implémenté par testAdditionner2( )

CT3(« addition de 1 et 2 et 3 », {[1,2,3]}, {6}) //etc.

CT4(« addition de 2 et 1 et 3 », {[2,1,3]}, {6})

CT5(« addition de 3 et 2 et 1 », {[3,2,1]}, {6})

CT6(« addition de 0 », {[0]}, {0})

CT7(« addition de -4 et 4 », {[-4,4]}, {0})

CT8(« addition de -2 et 0 et 2 », {[-2,0,2]}, {0})

Question 1.5 : Programmez ces huit tests dans 8 fonctions séparées :

CT1(« soustraction de 3 et 3 », {[3,3]}, {0})

CT2(« soustraction de 3 et 3 et 3 », {[3,3,3]}, {-3})

CT3(« soustraction de 1 et 2 et 3 », {[1,2,3]}, {-4})

CT4(« soustraction de 2 et 1 et 3 », {[2,1,3]}, {-2})

CT5(« soustraction de 3 et 2 et 1 », {[3,2,1]}, {0})

CT6(« soustraction de 0 », {[0]}, {0})

CT7(« soustraction de -5 et 5 », {[-5,5]}, {-10})

CT8(« soustraction de -2 et 0 et 2 », {[-2,0,2]}, {-4})

Question 1.6 : Les tests sont-ils des succès ? Dans ce cas, en temps normal on demanderait au développeur de corriger le bug. Faites le vous-même.

Troisième partie – Programmation de tests paramétriques

Dans cette partie, on crée des tests paramétriques cf. **cette partie du cours vue en TD**.

Question 1.7 : Programmez avec un test paramétrique ces huit tests :

CT1(« multiplication de 3 et 3 », {[3,3]}, {9})

CT2(« multiplication de 3 et 3 et 3 », {[3,3,3]}, {27})

CT3(« multiplication de 1 et 2 et 3 », {[1,2,3]}, {6})

CT4(« multiplication de 2 et 1 et 3 », {[2,1,3]}, {6})

CT5(« multiplication de 3 et 2 et 1 », {[3,2,1]}, {6})

CT6(« multiplication de 0 », {[0]}, {0})

CT7(« multiplication de -5 et 5 », {[-5,5]}, {-25})

CT8(« multiplication de -2 et 0 et 2 », {[-2,0,2]}, {0})

Question 1.8 : Les tests sont-ils des succès ? Dans ce cas, en temps normal on demanderait au développeur de corriger le bug. Faites-le vous-même.

Quatrième partie – Réflexions sur la qualité des tests

Question 1.10 :

Les suites de test de `additionner`, `soustraire`, `multiplier` ont 3 cas de tests intéressants, dont les données de test sont [1,2,3], [2, 1, 3], [3,2,1].

On remarque que pour multiplier et additionner, les 3 cas de test ont le même oracle, ces cas de tests sont quand même intéressants car ils testent une propriété mathématique, la _____ ?

Question 1.12 :

La méthode soustraire, par contre, a les mêmes données de test mais 3 oracles différents. Partage-t-elle la même propriété mathématique ?

Question 1.13 :

Comment tester l'associativité de l'addition et la multiplication ? Implémentez cela avec les cas de tests utilisant [1,2,3].

Sixième partie – Réflexions sur le test aux extrêmes

Question 1.14 : Pour constater des problèmes aux extrêmes, programmez ces cas de test :

CT30(« multiplication de 2^29 et 2 », {[2^29,2]}, {2^30})

CT31(« multiplication de 2^29 et 4 », {[2^29,2^2]}, {2^31})

CT32(« multiplication de 2^29 et 8 », {[2^29,2^3]}, {2^32})

CT33(« multiplication de 2^29 et 16 », {[2^29,2^4]}, {2^33})