Des tests unitaires dans Android Studio

Jérôme Tanghe

Mercredi 16 novembre

Ce TP a pour objectif de présenter l'utilisation des tests unitaires dans Android Studio. Pour rappel, les tests unitaires permettent de vérifier que votre application répond bien aux spécifications techniques en fournissant un ensemble de règles auxquelles votre code doit se conformer.

Ils se distinguent de l'analyseur de code qui vérifie la structure, la fiabilité et l'efficacité de votre code (par exemple, l'utilisation de méthodes dépréciées, l'usage d'espaces superflus dans un fichier XML, etc.), et qui se lance automatiquement lorsque vous compilez votre projet.

Dans cet IDE, et comme dans toute application écrite en Java, nous utiliserons JUnit dans sa version 4. Les tests unitaires seront stockés dans le répertoire `src/test/java` du projet.

Clonez le dépôt suivant pour commencer :

`https://www.github.com/Deuchnord/android-junit`

Il s'agit d'un petit projet très basique, contenant une classe `Operation` fournissant quatre méthodes calculant, pour tous nombres flottants n_1 et n_2, les opération $n_1 + n_2$, $n_1 - n_2$, $n_1 \times n_2$ et $\frac{n_1}{n_2}$.

1 Préparation de l'environnement de test

Avant de commencer, vérifiez que vous avez bien JUnit inscrit dans les dépendances de votre projet. Ouvrez votre `build.gradle` et vérifiez que vous avez bien la ligne suivante dans la section `dependencies` :

```
testCompile 'junit:junit:4.12'
```

2 Notre premier test unitaire

Ouvrez maintenant la classe `OperationTest` dans le répertoire `com.tanghe.test`. Vous y trouverez cinq méthodes de tests unitaires, pour chaque méthode de la classe `Operation`. Lisez et vérifiez que vous comprenez le code.

Nous allons maintenant lancer les tests unitaires. Vous pouvez faire cela de trois façons différentes :

- en cliquant droit sur une méthode et en choisissant **Run** (cela permet de ne lancer qu'une méthode de test unitaire) ;
- en cliquant droit sur une classe de tests unitaires et en choisissant **Run** (cela lance les tests unitaires de cette classe) ;
- en cliquant droit sur le répertoire `com.tanghe.test` (cela permet de lancer l'ensemble des tests unitaires du projet).

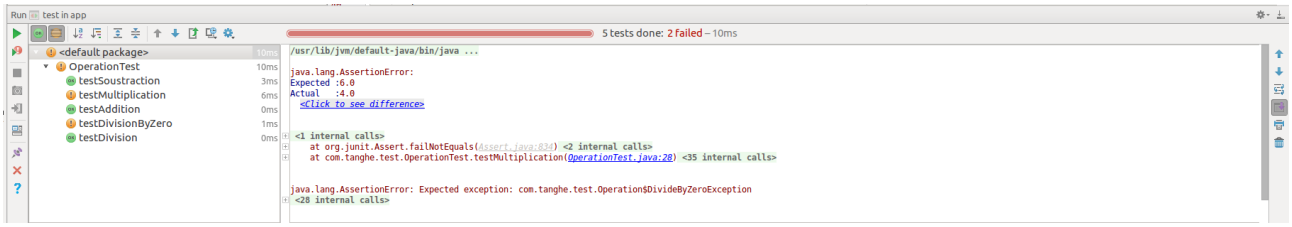

FIGURE 1 – Les tests unitaires ne passent pas.

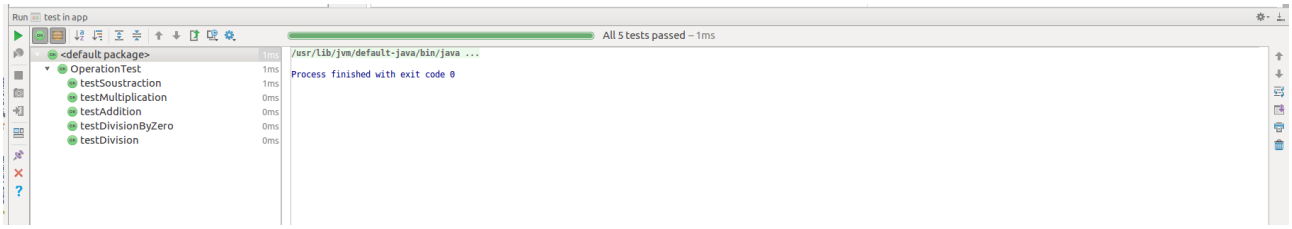

FIGURE 2 – Les tests unitaires passent.

Comme vous pouvez le voir, deux tests unitaires échouent (figure 1). Vous pouvez voir le détail de chaque test échoué en cliquant sur le nom de la méthode dans la liste de gauche.

Regardons ce qui n'a pas été :

— `testMultiplication` : la méthode a retourné 4, mais on attendait 6.

— `testDivisionByZero` : la méthode n'a pas levé d'exception.

Retournons sur la classe `Operation`. Voici comment corriger les problèmes :

— Dans la méthode `multiplication`, remplacez un des `n1` par `n2`.

— Dans la méthode `division`, décommentez les lignes commentées.

Il ne reste plus qu'à relancer les tests unitaires pour vérifier que les problèmes ont bien été corrigés. Normalement, tout devrait passer maintenant (figure 2) !

3 À vous !

Écrivez les tests unitaires des méthodes suivantes dans la classe `OperationTest` :

— `double carre(double n)` qui retourne `n` élevé au carré ;

— `double racineCarree(double n)` qui retourne la racine carrée [1] de `n`.
Si `n` est un nombre négatif, cette méthode devra lever une `NegativeNumberException` (déjà implémentée dans la classe `Operation`).

Puis implémentez ces méthodes à la classe `Operation`.

4 Références

— **Building Effective Unit Tests**
`https://developer.android.com/training/testing/unit-testing/index.html`

— **Automating User Interface Tests**
On n'en a pas parlé, mais c'est toujours cool de savoir que ça existe ☺
`https://developer.android.com/training/testing/ui-testing/index.html`

1. La racine carrée d'un nombre peut être calculée à l'aide de la méthode `Math.sqrt(n)`.