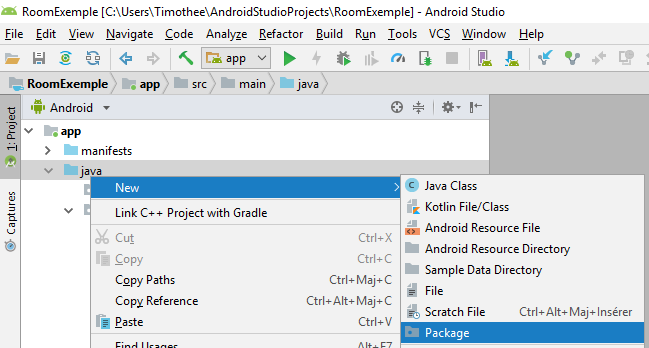
**Contexte 1ère partie** : vous êtes en stage dans une boulangerie où vous êtes chargés de développer une calculatrice enregistreuse pour remplacer la caisse enregistreuse dont la maintenance est onéreuse. La boulangère, méfiante, vous demande de tester soigneusement votre application avant d'essayer le pilote

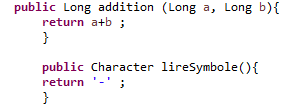
**Exercice n°1 : test d’un programme calculatrice**

Etape 1: créer un nouveau package Java, dans le répertoire app\java. Appelez le par exemple calculatrice

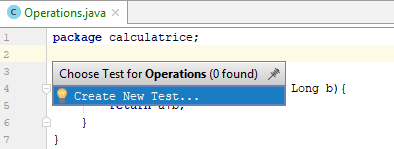


Etape 2: créez une classe nommée Operations, en se plaçant dans le package **calculatrice** que vous venez de créer.

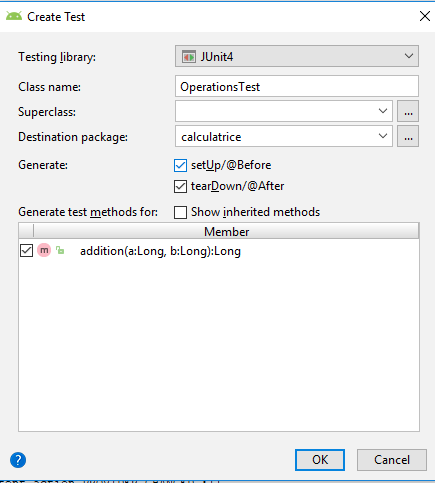
Etape 3: ajoutez les 2 méthodes suivantes à la classe Operations :



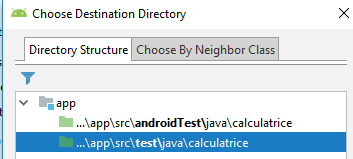
Etape 4: une fois le code entré, mettez vous n’importe où dans le code et tapez : Ctrl+Shift+T



Etape 5 : ajoutez un setup et teardown



Etape 6 : conservez le choix par défaut (test unitaire)



Remarque : le répertoire **test** stocke les tests unitaires tandis que le répertoire **androitTest** stocke les tests unitaires ***instrumentés*** c’est-à-dire sur le device.

Etape 6: Faire un clic droit dans le fichier OperationsTest.java et sélectionnez **Run OperationsTest with coverage**

**Question 1**: quel est le résultat du test ? Pourquoi ?

Etape 7: on ajoute le code suivant à la classe OperationsTest.java

Une variable d'instance:

**public** Operations op;

**Question 2**: où faut il ajouter cette déclaration ? qu'est ce que op ?

Etape 10: on ajoute le code suivant à la classe OperationsTest.java

--> dans la méthode setUp()

op = **new** Operations() ;

**Question 3**: à quoi sert cette instruction ?

Etape 8: on remplace la partie @Test par le code suivant

@Test

**public** **void** addition() **throws** Exception {

*assertEquals*((Character)'+', op.lireSymbole());

}

Etape 9: Faire un clic droit dans le fichier OperationsTest.java et sélectionnez de nouveau Run OperationsTest with coverage

**Question 4**:

* quel est le résultat du test ? Pourquoi ?
* comment corriger le problème ? Effectuez la correction.

Etape 10: dans le fichier OperationsTest.java, on ajoute un nouveau test

@Test

**public** **void** testCalculer() **throws** Exception {

*assertEquals*(**new** Long(4),

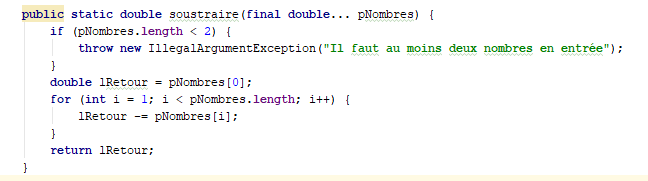
op.addition(**new** Long(1), **new** Long(3)));

}

**Question 5**:

* Où placer ce test ? Exécutez les tests et commenter les résultats. Que font ces tests ?
* Comment Junit sait-il qu'il faut exécuter ces méthodes (testCalculer() et addition() ) ?

Etape 11:nous allons ajouter dans la classe Operations une méthode permettant d'effectuer des soustractions à partir de 2 ou plusieurs nombres



**Point de cours**:

- le terme **static** pour la méthode soustraire signifie que c'est une méthode de classe

- le terme **final** dans le paramètre de la méthode *soustraire* signifie que la valeur du paramètre qui suit ne peut pas être modifié par la suite

- le terme ... dans le paramètre de la méthode *soustraire* signifie que le paramètre pNombres est un tableau de 1 à N éléments

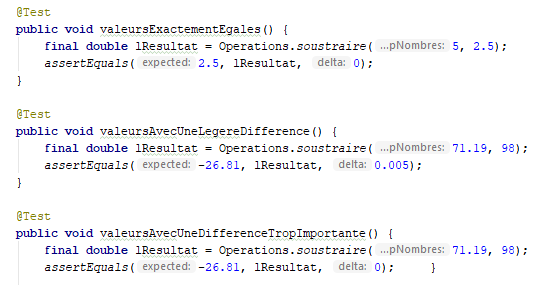
**Question 6**:

- que fait cette méthode ? que retourne t'elle ?

**Rappel**: comme la méthode soustraire est une méthode de classe (static), elle va s'appliquer à la classe et non un objet, de la forme *Operations.soustraire* (à compléter).

Etape 12:nous allons créer dans des tests unitaires permettant de mesurer la précision de cette calculatrice pour s'assurer qu'elle ne fait pas perdre des centimes à la boulangère.

Ajoutez 3 tests à la classe OperationsTest



**Question 7**:

* quel est le résultat des tests ? que font ces tests ?
* ajoutez un test qui soustrait 4 valeurs: 5 euros, la valeur d'un billet, 0.65, 1.10, 0.15 les valeurs d'une baguette, d'une baguette tradition et d'un bonbon.
* A t'on une précision exacte ? Que faut-il faire ? Qu'en concluez vous ?