TP 1 – Les Tests Unitaires avec JUnit 5

Exercice 1:

Soit une classe **Operation** qui permet d'ajouter la valeur 5 à un entier n.

1. Ecrire la classe **Operation** ayant une méthode *addToFive(int n)* qui retourne la valeur *n+5*
2. Ecrire une classe **OperationTest** qui teste la méthode *addToFive(int n)* avec la valeur 0, -20 et 10
3. Exécuter les tests et vérifier que les tests ont passé avec succès. Si un des tests a échoué corriger votre classe **Operation**.
4. Ecrire une méthode vide nommée *divide()* dans la classe **Operation**, qui permet de diviser deux entiers passés en paramètre.
5. Ecrire les Tests Unitaires correcpondant à la méthode *divide(int a, int b)* et exécuter ces tests (tester avec des entiers positifs, négatifs et 0).
6. Modifier la méthode *divide(int a, int b)* pour que tous vos tests passent.
7. Remodifier la méthode *divide(int a,int b)* de façon à lancer une exception dans le cas ou b est égal à 0. Vous pouvez utiliser la classe d'exception ***ArithmeticException*** déjà définie (ou bien vous définissez votre propre exception).
8. Ajouter une méthode de test qui permet de tester le cas de la division par zéro (vérifier s'il y a une exception qui a été levée).
9. Lancer votre classe de Test avec l'option de couverture de code (test with coverage) et vérifier si tous votre code est testé.

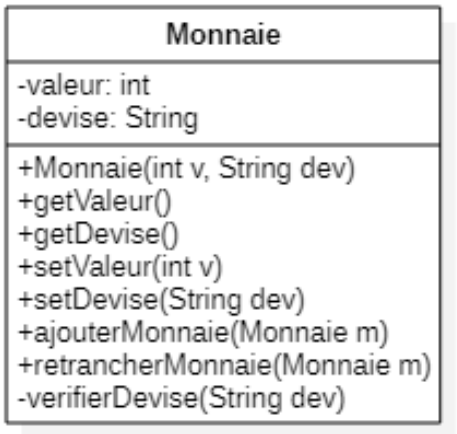
Exerice 2: TDD

Soit la classe **UitilitaireChaine**, qui contient des traitements sur une chaîne de caractère. Cette classe peut nous informer sur le nombre d'occurrence d'un caractère dans une chaîne. Elle permet de vérifier aussi que la chaine est palindrôme.

1. Ecrire la classe **UtilitaireChaine** ayant un constructeur qui initialise la chaîne.
2. Ecrire les méthodes (vide) *nbOccurence()* et *estPalindrome()*
3. Ecrire les Tests unitaires pour les méthodes *nbOccurence()* et *estPalindrome()*
4. Exécuter les tests développés et modifier les méthodes pour que vos tests passent.

Exercie 3:

Soit la classe **Monnaie** suivante:



1. Ecrire les exceptions **NegativeMonnaieException** et **DeviseInvalidException**.
2. Il n'est pas possible d'avoir une monnaie dont la valeur est négative. Et pour pouvoir additionner deux monnaies il faut qu'ils soient de la même devise. Dans ces cas les exceptions adéquates seront levées.
3. Ecrire la classe TestMonnaie qui contient les méthodes de test unitaire pour vérifier les méthodes *ajouterMonnaie()* et *retrancherMonnaie()*.

4. Mettre à jour la classe Monnaie pour faire passer tous vos tests.

Exercice 4: les tests paramétrés

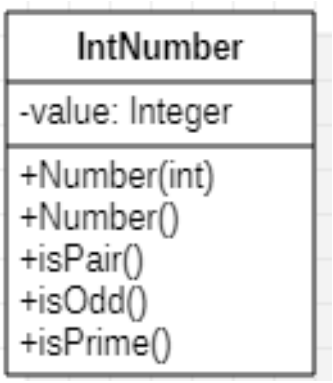
@ParameterizedTest permet d'exécuter plusieurs fois le test avec des valeurs différentes. Cette annotation est utilisé avec une source de données qui contient les différentes valeurs de test.

Reprendre l'exercice n°2 et modifier les tests unitaires de façon à donner les chaînes de caractère à tester dans une source de données.

- Pour le test de la méthode estPalindrome(), on utilisera la source de données ***@ValueSource***.
- Pour le test de la méthode nbOccurrences(char), on utilisera la source de données ***@CsvSource*** ou ***@CsvFileSource***

Exercice 5: Les tests dynamiques (@TestFactory)

Soit la classe suivante:



Ecrire la classe **Number** et développer les tests nécessaires pour valider votre classe.