Faculté des Sciences
Aix*Marseille Université

Département Informatique et Interactions
Année 2020-21

Master 1 Informatique
Fiabilité Logicielle

TP Tests Unitaires avec `JUnit`

Ce TP introduit à l'utilisation de l'environnement `JUnit` de tests unitaires. `JUnit` est un environnement de tests unitaires pour Java et l'IDE Eclipse contient un plugin `JUnit` dans sa distribution (sinon charger le plugin). La version `JUnit` actuelle est la 5, mais la 4 est encore utilisée. La version 5 est une évolution majeure qui offre de nouvelles constructions. Le site pour `JUnit` 5 `https://junit.org/junit5`.

1 Ecriture de test

Sous Eclipse, on peut créer une classe de type `JUnit` test en spécifiant les constructions voulues et la classe à tester. Ensuite, il faut écrire les tests de cette classe. Un test est une méthode précédée de l'annotation `JUnit` *@Test*. Un test utilisera les assertions de la classe *Assert* qui seront utilisées pour vérifier des égalités (de valeurs, objets, ...) et feront échouer le test si elles ne sont pas vraies.

```
@Test
public void testAdditionZero(){
double expected = 1.0;
Essai e = Essai(1.0);
e.ajouter(0.0);
double val = e.getVal();
assertTrue("Test 0 neutre", expected == val);
}
```

D'autres annotations `JUnit` permettent de gérer les initialisations et cas particuliers. Elles sont de la forme `@mot-clé`.

- `@BeforeAll` et `@AfterAll` permettent d'exécuter des instructions avant et après l'exécution de la suite de tests (test fixture).
- `@BeforeEach` permet de définir des initialisations à faire avant chaque test (typiquement définir un objet qui sera utilisé par tous les tests) et `@AfterEach` est similaire mais est effectué après chaque test.
- `@Disabled` permet de ne pas effectuer le test qui suit.

Les annotations `@...` et assertions `assert...` sont à importer (voir les fichiers exemples et la documentation).

2 Utilisation de `JUnit`

Lire la documentation ! Vous utiliserez l'IDE `eclipse` pour ce TP et tous les TP suivants en `Java`.

2.1 Prise en main

Cette première partie va vous faire voir ou revoir un usage basique de `JUnit` .

1. Créer un projet `tptestunitaires` et le paramétrer pour pouvoir utiliser `junit5`.
2. Récupérer `EssaiTest` pour voir un début d'implémentation d'une classe de test et regarder sa structure.
3. Dans un package `essai`, écrire une classe `Essai` ayant un attribut `val` de type `double`, les méthodes `getVal()` et `setVal(double )` et `void ajouter(double v)` qui ajoute v à val et le constructeur `Essai(double)`.
4. Exécuter la classe de test fournie et effectuer ce qui est signalé en commentaire. Compléter la classe de test `EssaiTest` avec une initialisation pour des objets `essai1, essai2` et des tests pour les méthodes `getVal()` et `setVal(double )` et le constructeur. Exécuter les tests.
5. En rajoutant des instructions d'impression, vérifier que `@BeforeAll` et `@AfterAll` sont effectuées avant et après chaque test. Idem pour `@BeforeEach` et `@AfterEach`.
6. Ajouter dans `Essai` une méthode `double inverserVal()` qui renvoie `1/val` et lève une exception de type `IllegalArgumentException` pour $val == 0.0$. Ecrire les tests pour cette nouvelle méthode.
7. Ajouter une méthode `double infinite()` qui ne termine pas dans la classe `Essai` puis activer le test sur `infinite`. Que se passe-t-il ? Modifier le test pour qu'il échoue si `infinite` ne termine pas en 100ms (utiliser l'assertion `assertTimeoutPreemtively` (voir la partie `Timeout` de la documentation). Vérifier que l'exécution des test termine.

Avertissement : `JUnit` 5 permet de programmer une gestion de test très évoluée (voir la documentation) ce qui signifie qu'il est possible d'introduire des erreurs dans les suites de test.

3 Travail à faire

Les deux applications à tester sont celles du premier TP.

3.1 Application triangle

Vous allez transcrire en Java le code C de la fonction `typeTriangle` . Cette fonction va devenir une méthode sans argument (ceux-ci étant les attributs) de la classe `Triangle`. Cette classe sera dans un package `triangle` du répertoire `src`.

1. Ecrire la classe Java `Triangle` avec 3 attributs privés pour les cotés, les `getters` et `setters` des attributs et la méthode `int typeTriangle()` en reprenant votre code C du dernier TP.
2. Ecrire la suite de test `JUnit testTypeTriangle`. Consigne : un test ne vérifie qu'une seule assertion à la fois. Cette suite de test sera dans un package `triangle test` du répertoire `test`.
3. **cette question est à traiter après la partie recherche dans un tableau trié**. Pour rajouter `readData` la spécification est précisée : si le fichier n'existe pas, une exception est renvoyée et on pourra supposer que seuls les fichiers textes sont traités. Le fichier ne doit contenir que 3 lignes correspondant chacune à une valeur de type double. Ecrire la suite de tests `testReadData` pour la méthode `readData`.

Pour les entrées/sorties en Java, de nombreux sites expliquent avec des exemples
`http://thecodersbreakfast.net/index.php?post/2012/01/15/java-io-explique-simplement`

3.2 Application recherche dans un tableau trié

Dans cette partie vous pourrez utiliser des constructions comme les timeout, test de levée d'exception, tests paramétrés, ... Les sources de l'application seront dans un package `chercher` de `src` et les tests dans un package `testchercher` de `test`.

1. Ecrire une classe Java `Chercher` qui contient la méthode `chercherElt(int, int [] tab)` qui effectue une recherche par dichotomie en $O(log_2(n))$ dans le tableau trié `tab` de taille n en renvoyant l'indice minimal de l'élément cherché s'il est présent, -1 s'il est absent. Le tableau `tab` est supposé être trié et on ne vérifiera pas cette propriété. Ecrire la classe de test correspondante.
2. Récupérer sous AMETICE le fichier `chercher.java` avec 5 méthodes différentes de recherche dans un tableau d'entiers triés par ordre croissant `chercher1,2,3,4,5(int x, int[] tab)`. Reprendre votre classe de test et l'adapter pour tester chacune des fonctions. Vous écrirez une classe de tests pour chacune des fonctions.

3.3 Rendus demandés

1. Le projet `tptestunitaires` organisé en deux répertoires `src` et `test`.
2. l'application et les tests pour le triangle avec les packages `triangle` et `testtriangle` avec la classe `Triangle` et les classes `JUnit` de tests pour `typeTriangle` et `ReadData` (chaque classe contiendra des commentaires permettant de comprendre ce qui est fait ou testé).
3. l'application et les tests pour l'application `chercher` avec les packages `chercher` et `testchercher` avec la classe `Chercher` qui contiendra votre méthode `chercherElt` et les méthodes `chercher1,2,...,5` et les classes `JUnit` de tests (chaque classe contiendra des commentaires permettant de comprendre ce qui est fait ou testé).
4. Le compte rendu donnera le résultat de vos tests avec une analyse des résultats et des constructions `JUnit` utilisées.