Génie Logiciel Avancé
TP nº 2 : Test avec JUnit 5

Le but de cette séance de travaux pratiques est de vous familiariser avec l'écriture de tests unitaires en Java à l'aide de la bibliothèque JUnit dans sa version 5. On en profitera également pour introduire Gradle, un outil de construction logicielle (*build*) en train de s'imposer dans l'écosystème de la *Java Virtual Machine* (JVM).

Prérequis. Assurez-vous d'avoir installé sur votre machine le *Java Development Kit* en version 14+ et l'outil Gradle en version 6.5+. La dépendance à JUnit sera gérée automatiquement par Gradle. Cette séance de travaux pratiques a été conçue sous Unix ; vous pouvez, à vos risques et périls, essayer de la suivre sous Windows, les environnements Java étant en théorie portables. Le projet a été testé avec les environnements Eclipse et IntelliJ.

Matériel. Les exercices supposent que vous modifiez du code existant. Celui-ci vous est fourni sur la page Moodle du cours, située à l'adresse ci-dessous.

`https://moodle.u-paris.fr/course/view.php?id=10699`

La lecture des documentations officielles de Gradle et JUnit 5 est obligatoire, en particulier la section *Writing Tests* de cette dernière.

Exercice 1 – *Fibonacci*

Le code de cet exercice contient trois implémentations d'une classe calculant le $n$ème nombre de la suite de Fibonacci : deux implémentations récursives naïves, l'une séquentielle et l'autre parallèle, ainsi qu'une implémentation itérative d'une bien meilleure complexité calculatoire.

1. Téléchargez l'archive `tp2.zip` sur la page Moodle du cours, et décompressez là. Prenez le temps de parcourir la hiérarchie des répertoires obtenue ainsi que de lire le contenu des fichiers d'extension `.java` et `.gradle` pour commencer à vous familiariser avec Gradle.

2. Comment compiler le code source que vous avez obtenu ? Référez-vous à la documentation de Gradle, et vérifiez que cette tâche fonctionne correctement.

3. Écrivez un fichier `code/src/test/java/tp2/ex1/TestFib.java` de sorte à tester que le 15ème nombre de la suite de Fibonacci est 377, et ce pour les trois implémentations disponibles. Si l'une des implémentations s'avère défectueuse, corrigez son bug.
   Indications : vous devez définir une classe `TestFib` comprenant une méthode annotée avec `@Test` ; vous devez importer les classes encapsulant le calcul de Fibonacci via une directive `import` ; vous pouvez utiliser la méthode statique `assertEquals` pour comparer les valeurs.

4. En plus de tester des valeurs concrètes, on peut chercher à vérifier que nos trois implémentations de la suite Fibonacci renvoient le même résultat pour un entier n fixé mais arbitraire. On peut pour cela utiliser un test *paramétré*, déclaré via l'annotation `@ParameterizedTest`. Utilisez un test paramétré et l'annotation `@ValueSource`, pour tester que les résultats sont les mêmes pour $n \in \{0, 12, 15, 20, 30\}$.

5. Ajoutez au code de l'exercice précédent une classe `PrecomputedFib` qui précalcule les 30 premiers entiers de la séquence de Fibonacci, par exemple en complétant le code ci-dessous.

```
public class PrecomputedFib {
  // Le nombre d'éléments de la suite de Fibonacci à précalculer.
  final int N = 30;
  // Le tableau contenant les éléments précalculés.
  int elements[] = new int[N];

  public PrecomputedFib() {
    // À compléter.
  }
  public int fib(int n) {
    // À compléter.
  }
}
```

   Ajoutez des tests unitaires pour cette classe aux méthodes de test définies à l'exercice précédent. Vous pouvez créer une instance de la classe dans chaque méthode.

6. Créer une instance de la classe à chaque appel de chaque méthode de test a l'inconvénient d'être possiblement coûteux en temps et en espace. Faites en sorte de ne créer qu'une unique instance de la classe `PrecomputedFib` pour toutes les méthodes de test. *Indication* : lire la section *Test Instance Lifecycle* de la documentation de JUnit 5.

Exercice 2 – *Fibonacci et exceptions*

1. Testez les classes fournies ainsi que celle obtenue à l'exercice précédent en leur fournissant des entiers strictement négatifs. Que constatez-vous ?

2. Modifiez les méthodes de ces classes pour lever une exception spécialisée que vous aurez préalablement définie.

3. Écrivez un jeu de tests qui vérifie que toutes les implémentation lèvent votre exception en cas de nombres négatifs. Quelle méthode JUnit utiliser ?

Exercice 3 – *File à double-entrée*

1. Lisez le fichier `BoundedQueue.java` fourni dans `src/tp2/ex2`. Implémentez l'interface `BoundedQueue` via une classe `ArrayBoundedQueue` utilisant un tableau et deux compteurs.

2. Écrivez des tests unitaires pour l'interface `BoundedQueue`, et utilisez les pour tester la classe `ArrayBoundedQueue`.

3. Le fichier `settings.gradle` qui vous a été fourni fait appel au plugin JaCoCo pour calculer automatiquement la couverture de code de vos tests. Faites en sorte que vos tests atteignent 85% de couverture.