TP TDD : Développement dirigé par les tests et utilisation de JUnit

Attention : à l'issue de ce TP, chaque binôme doit envoyer par mail à son encadrant de TP un certificat (un par binôme) qui prouve que vous avez terminé le TP. Ce certificat vous sera donné par le "client" qui est à exécuter depuis votre session sur une machine de l'ISTIC. Si vous travaillez sur votre machine personnelle, pensez à générer le certificat depuis une machine de l'ISTIC.

1 Préambule

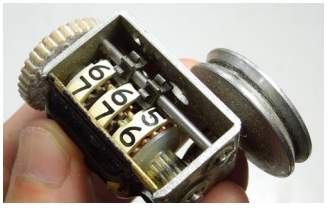
Dans ScalaIDE, importez le projet `/share/l2ie/gen/TPtdd/TPtdd.zip` en utilisant :
File>Import>Existing Projects into Workspace>Select archive file.

2 Contenu du projet `TP_TDD`

Le projet comporte deux répertoires `src` (le source) et `tests` (les tests).

- Dans le répertoire `src`, vous trouverez deux packages :
  - ⋆ `compteur` contient le trait `Compteur` ainsi qu'une implémentation à compléter `CompteurImpl` ;
  - ⋆ `client` contient une application de test automatique : l'objet `Client` testera votre implantation `CompteurImpl` et vous dira si elle satisfait votre client par rapport aux fonctionnalités attendues.
- Dans le répertoire `tests`, vous trouverez un package `compteur` qui contient une classe de test JUnit nommée `TestCompteur` dans laquelle vous pourrez placer vos tests sur `CompteurImpl`.

3 Principe des compteurs



On souhaite représenter des compteurs basés sur les même principes que les compteurs mécaniques composés de roues. Cependant, contrairement au compteurs usuels, on ne supposera pas que les roues ont toutes le même nombre de valeurs possibles. Un compteur pourra être initialisé avec une liste de valeurs maximales pour chaque roue : une liste non vide de valeurs supérieures ou égales à 0. Comme dans les compteurs mécaniques, lorsqu'une roue aura fait un tour complet, elle fera avancer d'un cran la roue située à sa gauche (si elle existe). Le trait `Compteur` comprend les opérations suivantes :

- `init(l: List[Int])` qui initialise le compteur comme décrit ci-dessus ;
- `courant: List[Int]` qui retourne la liste de valeurs courante du compteur ;
- `suivant: Unit` qui incrémente le compteur. Si on a atteint le maximum, le compteur est bloqué sur la dernière valeur comptée ;
- `suivantPossible: Boolean` qui dit s'il est encore possible d'incrémenter le compteur ou s'il a atteint la valeur maximum ;
- `def solPossibles: Int` qui donne le nombre total de valeurs que le compteur peut prendre tel qu'il est initialisé ;
- `def solRestantes: Int` qui donne le nombre d'incréments que peut encore réaliser le compteur.

Exemple : Imaginons que l'on ait un compteur `c` implantant le trait `Compteur`. Voici, une séquence d'opérations effectuée dans l'interprète Scala et le résultat obtenu.

```
> c.init(List(1,2))
> c.solPossibles
res8: Int = 6
> c.solRestantes
res9: Int = 5
> c.courant
res10: List[Int] = List(0, 0)
> c.suivant
> c.solRestantes
res12: Int = 4
> c.courant
res13: List[Int] = List(0, 1)
> c.suivant
> c.courant
res15: List[Int] = List(0, 2)
> c.suivant
> c.courant
res17: List[Int] = List(1, 0)
> c.suivant
> c.courant
res19: List[Int] = List(1, 1)
> c.solRestantes
res20: Int = 1
> c.suivantPossible
res21: Boolean = true
> c.suivant
> c.courant
res23: List[Int] = List(1, 2)
> c.solRestantes
res24: Int = 0
> c.suivantPossible
res25: Boolean = false
> c.suivant
> c.courant
res27: List[Int] = List(1, 2)
> c.solRestantes
res28: Int = 0
```

Remarque : On supposera qu'un compteur, s'il n'est pas initialisé avec `init`, ne comporte qu'une roue avec une seule valeur (0).

4 Objectif du TP et marche à suivre

L'objectif du TP est de réussir à développer une classe `CompteurImpl` qui passe avec succès tous les tests effectués par l'application de test `Client`. **Remarque :** même si l'initialisation du compteur et l'opération `suivant` manipulent des listes, la représentation interne des roues est libre. En particulier, vous pouvez utiliser un tableau d'entier pour représenter l'état courant des roues. Si vous voulez avoir une chance de passer 100% des tests en 2h, nous vous conseillons d'utiliser une approche incrémentale et dirigée par les tests. Commencez par des compteurs à une seule roue et ajouter des fonctionnalités (et des roues) au fur et à mesure :

Pour ajouter une fonctionnalité X :

1. Ecrire le(s) test(s) montrant que X fonctionne (dans la classe `TestCompteur` du répertoire `tests`)
2. Lancer JUnit et **vérifier que le nouveau test échoue** (X n'est pas encore développée). Ceci permet de vérifier que le test est valide !
3. Ecrire le code nécessaire pour passer le test (et pas plus)
4. Lancer JUnit et **vérifier que le test passe**, sinon retourner en 3.
5. Si nécessaire, remaniez le code pour le simplifier (principe KISS : "Keep It Simple Stupid")

Lorsque vous pensez avoir fini, lancez l'application `Client` elle vous dira à quelle distance de la bonne solution vous vous situez. `Client` vous donnera un **certificat** si vous passez tous les tests. Le certificat est à envoyer à votre encadrant de TP avec votre numéro d'étudiant.