JUnit et introspection

Exercice 1 : La classe `Lanceur`

L'architecture de JUnit conduit à écrire un grand nombre de petites classes, une par test. Ceci peut décourager le programmeur d'écrire des programmes de test. Aussi, JUnit permet de définir plusieurs tests dans la même classe. La convention [1] est de considérer que chaque méthode dont le nom commence par « test » correspond à un test élémentaire. Une méthode `setUp` (`preparer` dans notre cas) initialise les attributs correspondant aux données utilisées dans les programmes de test, `tearDown` (`nettoyer` dans notre cas) est exécutée à la fin du test.

Le programme qui contient les deux tests initiaux de Monnaie sont donnés par la classe MonnaieTest suivante.

```
1  /** Classe regroupant les tests unitaires de la classe Monnaie.  */
2  public class MonnaieTest {
3
4          protected Monnaie m1;
5          protected Monnaie m2;
6
7          public void preparer() {
8                  this.m1 = new Monnaie(5, "euro");
9                  this.m2 = new Monnaie(7, "euro");
10         }
11
12         public void testerAjouter() throws DeviseInvalideException {
13                 m1.ajouter(m2);
14                 Assert.assertTrue(m1.getValeur() == 12);
15         }
16
17         public void testerRetrancher() throws DeviseInvalideException {
18                 m1.retrancher(m2);
19                 Assert.assertTrue(m1.getValeur() == -2);
20         }
21
22 }
```

Pour lancer les tests, on donne alors le nom de la classe contenant les tests à un programme (dans notre cas la classe Lanceur) qui est chargé d'identifier toutes les méthodes de test, de construire une suite de tests, de lancer les tests et, enfin, d'afficher les résultats.

Bien entendu, pour que ceci fonctionne, la classe `Lanceur` doit pouvoir découvrir dans la classe donnée en argument toutes les méthodes de test, la méthode `preparer` et la méthode `nettoyer`. Pour résoudre ce problème, Java permet de faire de l'introspection, c'est-à-dire examiner l'état d'un programme. On utilisera en particulier :

1. Ceci était vrai jusqu'à la version 3 de JUnit. Depuis la version 4 ce sont les annotations de Java qui sont utilisées.

— la méthode `forName` définie dans la classe `Class`,
— les méthodes `getMethod` et `getMethods` de `Class`,
— la méthode `startsWith` de la classe `String`,
— la méthode `getModifiers()` de la classe `Method`,
— la classe `Modifier`,
— la méthode `newInstance` de `Class`,
— la méthode `invoke` de la classe `Method`.

Le principe est de construire une suite de tests pour chaque nom de classe donné sur la ligne de commande. Chaque suite de tests contient autant de tests élémentaires que de méthodes d'instance de la classe dont le nom commence par « test », qui ne prennent pas de paramètres et qui sont publiques. Ces tests élémentaires sont en fait des instances d'une classe `TestAvecIntrospection`, spécialisation de `TestElementaire` qui :

— propose un constructeur prenant en paramètres au moins une instance de la classe de test (de type `Object`) et la méthode de test (de type `Method`),
— redéfinit les méthodes `preparer`, `nettoyer` et `tester` pour exécuter respectivement les méthodes `preparer`, `nettoyer` et la méthode de test (celle passée en paramètre du constructeur) définies dans la classe de test.

1.1 Expliquer pourquoi utiliser `getMethod` et `getMethods` plutôt que `getDeclaredMethod` et `getDeclaredMethods`.

1.2 Compléter la classe `Lanceur` et écrire la classe `TestAvecIntrospection`.

1.3 Exécuter le lanceur sur les classes de test fournies :

1. `MonnaieTest`,
2. `MonnaieTest2`,
3. `CasLimitesTest`.

Exercice 2 : Tester le lanceur

Écrire une classe de test `LanceurTest` qui permet de tester le `Lanceur` grâce aux classes de test déjà écrites... et à de nouvelles à proposer.