

JUnit et introspection

Exercice 1 : La classe Lanceur

JUnit permet de définir plusieurs tests dans la même classe. La convention¹ est de considérer que chaque méthode d'instance dont le nom commence par « test » correspond à un test élémentaire. Une méthode `setUp` (préparer 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. Toutes ces méthodes doivent être publiques et sans paramètre.

Le programme qui contient les deux tests initiaux de Monnaie correspond à la classe `MonnaieTest` suivante.

```
/** Classe regroupant les tests unitaires de la classe Monnaie. */
public class MonnaieTest {

    protected Monnaie m1;
    protected Monnaie m2;

    public void preparer() {
        this.m1 = new Monnaie(5, "euro");
        this.m2 = new Monnaie(7, "euro");
    }

    public void testerAjouter() throws DeviseInvalideException {
        m1.ajouter(m2);
        Assert.assertTrue(m1.getValeur() == 12);
    }

    public void testerRetrancher() throws DeviseInvalideException {
        m1.retrancher(m2);
        Assert.assertTrue(m1.getValeur() == -2);
    }

}
```

Pour lancer les tests, on donne alors le nom des classes contenant les tests à un programme (dans notre cas la classe `LanceurIndependant`) 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.

1. Ceci était vrai jusqu'à la version 3 de JUnit. La version 4 de JUnit utilise les annotations.

Bien entendu, pour que ceci fonctionne, la classe `LanceurIndependant` 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 :

- 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`² de `Class`,
- la méthode `invoke` de la classe `Method`.

Théoriquement, il faudrait s'appuyer sur l'architecture vue en TD et donc construire une suite de tests contenant tous les tests définis dans une classe et une suite générale contenant toutes ces suites de tests. Comme l'objectif de ce TP est l'introspection, nous n'allons pas réutiliser l'architecture conçue en TD et nous contenter de faire tous les traitements dans la classe `LanceurIndependant`. Vous connaissez maintenant l'explication du nom de cette classe : elle est indépendante de l'architecture JUnit vue en TD.

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

1.2. La classe de test peut ne pas définir la méthode `nettoyer`. Que faut-il faire dans ce cas ?

Même question pour la méthode `preparer`.

1.3. Compléter la classe `LanceurIndependant`. Il est conseillé d'avancer petit à petit en compilant et en exécutant régulièrement et en vérifiant que vous obtenez bien les résultats attendus grâce à des traces placées dans le code.

1.4. Exécuter le lanceur sur les classes de test fournies :

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

1.5. En JUnit 3, une classe de test doit hériter de `TestCase` (`TestElementaire` pour nous). Expliquer pourquoi ce choix a été fait.

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.

2. Cette méthode est maintenant obsolète. Il faut donc récupérer le constructeur par défaut et exécuter sa méthode `newInstance`.