Exercices test en isolation / des interactions

Master1 info S2
TP SVL – 2017-2018

Les tests seront écrits en JUnit4. Tous les tests seront des test en isolation.

Exercice 1 : Des tests comme documentation

Cet exercice concerne le test d'un framework de test appelé MyJUnit, évidemment inspiré de JUnit mais très simplifié. Seuls les tests sont demandés. Le but de l'exercice est de produire une suite de test suffisante pour avoir confiance dans le code testé, mais aussi suffisamment **concise** et **claire** pour pouvoir servir de documentation pour les fonctionnalités des classes testées.

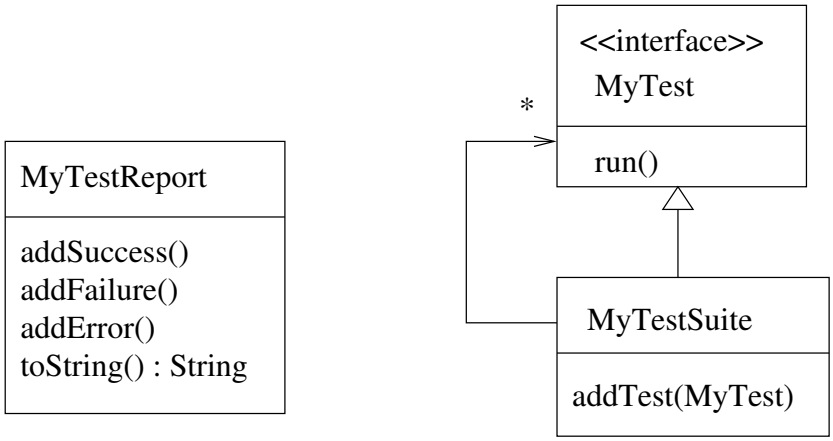

FIGURE 1 – Diagramme UML du paquetage `myjunit`

1.1 : Test de `MyTestReport`

La classe `MyTestReport`[1] collecte grâce aux méthodes `addSuccess`, `addFailure` et `addError` les verdicts (*success*, *failure*, *error*) des tests qui sont exécutés. La méthode `toString` produit une chaîne décrivant les verdicts dans l'ordre dans lequel ils auront été collectés, avec les conventions suivantes :

- *failure* est représenté par `F` ;
- *success* est représenté par `.` (un point) ;
- *error* est représenté par `E`.

Par exemple, dans le cas de deux *success* et d'une *error* collectés successivement, la chaîne produite sera `"..E"`.

Q 1.1 : Donner une classe de test JUnit qui teste la méthode `toString` de la classe `MyTestReport`, et documente les fonctionnalités de cette classe. □

1.2 : Test de la classe `MyTestSuite`

La classe `MyTestSuite` représente une collection de tests de type `MyTest`. On lui ajoute un nouveau test par la méthode `addTest(MyTest)`. La méthode `run` de `MyTestSuite` est chargée d'appeler la méthode `run` de chacun des tests qui la constituent, peu importe dans quel ordre. Si la suite de test ne contient aucun test, rien ne se passe.

Q 1. 2 : Donner une classe de test JUnit qui teste la méthode `run` de `MyTestSuite` et illustre les fonctionnements de `MyTestSuite`. □

1. Cette classe est très simplifiée par rapport à son homologue JUnit, qui associe notamment chaque verdict à un objet `Test`.

Exercice 2 : Processus

On s'intéresse à un container qui permet de contrôler des processus. Les processus ont l'interface suivante :

```
public interface Processus {
   public void demarrer();
   public void arreter();
   public boolean estEnMarche();
}
```

et le contrôleur a l'interface suivante :

```
public class ControleurProcessus {
   public void ajouterProcessus(Processus p);
   public void arreterTous();
   public int compterLesProcEnMarche();
}
```

Q 2.1 : Écrire une implémentation et les tests pour la méthode `compterLesProcEnMarche`. □

Q 2.2 : Que pensez-vous du test suivant ? □

```
public void testAbsurde() {
   ControleurProcessus controleur = new ControleurProcessus();
   Processus proc1 = mock(Processus.class, "proc1");
   Processus proc2 = mock(Processus.class, "proc2");
   Processus proc3 = mock(Processus.class, "proc3");
   controleur.ajouterProcessus(proc1);
   controleur.ajouterProcessus(proc2);
   controleur.ajouterProcessus(proc3);
   when(proc1.estEnMarche()).thenReturn(false);
   when(proc2.estEnMarche()).thenReturn(false);
   when(proc3.estEnMarche()).thenReturn(false);
   controleur.arreterTous();
   assertTrue(! proc1.estEnMarche());
   assertTrue(! proc2.estEnMarche());
   assertTrue(! proc3.estEnMarche());

}
```