

TP03 : AMELIORATION DES TESTS

JUNIT 3.8

Objectif du TP :

- Gérer un SetUp() pour les tests
- Gérer un tearDown() pour libérer la mémoire.
- Gérer des suites de Tests avec le Framework Junit.

OPTIMISER LES TESTS : 2 NOUVELLES METHODES

1) Problème soulevé :

💣 Lors des tests de la classe Cours, nous sommes obligés de faire des « new » pour chaque méthode testée afin de reproduire la situation que l'on veut tester :

```
public void testAjouterParticipant() throws Exception {

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

    ArrayList<Participant> listPart; /* INUTILE : new ArrayList<Participant>(); */

    p = new Participant("TAURAND", "Pierre", sdf.parse("10/10/1990"), (byte) 49);
    c = new Cours("Ski debutant", sdf.parse("15/12/2015"), 13, (byte) 10);

    etc...
}

public void testSupprimerParticipant() throws Exception {

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");


    Participant p1, p2 = null;
    Cours c = null;
    Date d1= new Date(9466812) ; /*Thu Jan 01 1970 03:37:46 GMT+0100 (Paris, Madrid)*/
    Date d2= new Date(9466812) ;
    try {
        d1 = sdf.parse("15/12/2015");
        d2 = sdf.parse("10/10/1990");
    } catch (ParseException e) {
        e.printStackTrace();
    }

    c = new Cours("Ski debutant",d1, 13, (byte) 10);
    p1 = new Participant("TAURAND", "Pierre", d2, (byte) 49);
    p2 = new Participant("TAURAND", "Sarah", d2, (byte) 49);

    etc...
}
```

2) Solution : Méthode setUp

JUnit nous propose d'utiliser la méthode `setUp` afin de configurer la situation dans laquelle on veut se trouver avant de lancer chaque Test.

 Il est important de se souvenir que la méthode `setUp()` est **invoquée systématiquement avant l'appel de chaque méthode de tests**. Sa mise en œuvre n'est donc requise que si toutes les méthodes de tests ont besoin de créer une instance d'un même type ou d'exécuter un même traitement.



exemple :

```
public class PersonneTest extends TestCase {  
  
    private Personne personne;  
  
    public PersonneTest(String name) {  
        super(name);  
    }  
  
    protected void setUp() throws Exception {  
        super.setUp();  
        personne = new Personne("nom1", "prenom1");  
    }  
}
```

Ici on redéfinit la méthode `setUp()` et on crée les différentes instances nécessaires à mes tests (les différentes méthodes que je vais écrire)



Créer la méthode `setUp()` afin d'alléger vos méthodes de la classe `CoursTest`

3) La Méthode tearDown



exemple :

```
protected void tearDown() throws Exception {  
    super.tearDown();  
    personne = null;  
}
```

Dans le même esprit, la méthode `tearDown` permet de remettre la mémoire à une situation initiale après chaque appel des méthodes de Tests.

Ici, on met la variable `personne` à `null` donc plus d'objet en lien avec cette variable. Donc le garbage collector va se charger de détruire l'objet qui n'est plus référencé.

Cela libère les ressources mémoire : très utile si l'on se connecte à une BDD dans le `setUp()` !



Créer la méthode `tearDown()` afin de remettre les variables dans une situation « vierge ».

TESTS EN CASCADE : LES SUITES DE TEST

1) Problème soulevé :

Lors des tests Fonctionnels (de recette interne) nous voulons vérifier l'intégralité d'une fonctionnalité (celle-ci est une suite de tests unitaires)

Comment faire pour enchaîner les tests unitaires (TestCase) ?

2) Solution : Classe TestSuite

JUnit nous propose d'utiliser la classe de type TestSuite et d'appeler la méthode addTest() pour chaque classe à ajouter.



exemple :

```
import junit.framework.*;
```

```
public class ExecuterPlusieursTests {
```

```
    public static Test suite() {  
        TestSuite suite = new TestSuite("Tous les tests pour cette fonctionnalité");  
        suite.addTestSuite(MaClasseTest.class);
```

OU

```
        suite.addTestSuite(new ClasseTest (« methodeprecise »));
```

```
        return suite;
```

```
    }
```

```
}
```

3) Application : Test de la fonctionnalité « Ajouter Et Rechercher Un Participant à un cours »

Nous voulons appeler :

- le test du nom du participant
- puis l'ajout de ce participant dans le cours
- enfin la recherche de ce participant par son nom de famille



Créer la classe qui va gérer les suites de Tests.