Laboratoire GEN *No II*

JUNIT

HES-SO - *Christophe Greppin/ Eric Lefrançois*

1 Introduction

Dans la programmation, le test unitaire permet de s'assurer du fonctionnement correct d'une partie d'un programme. C'est pour le programmeur un moyen efficace de tester, indépendamment du programme complet, un bout de code. Ainsi, cela permet de contrôler que ce dernier correspond aux spécifications et qu'il fonctionne en toutes circonstances. JUnit est une librairie pour le langage de programmation Java qui permet d'effectuer ces tests. De plus, elle est très simple à utiliser et elle est intégrée par défaut dans les environnements de développement Eclipse & Together (basé Eclipse). L'un ou l'autre peut être utilisé pour ce laboratoire.

2 Exemple de classe à tester

Nous allons utiliser une classe d'exemple en Java qui fournit diverses méthodes. Elle permettra de tester la majorité des tests unitaires utiles. Cette classe se veut la plus simple possible afin de ne pas perdre de temps sur la compréhension du code.

> ***Note POO** : cet exemple utilise des méthodes d'instance (`factoriel`, `min`, etc..). Du stricte point de vue méthodologie POO, il serait préférable de déclarer ces dernières en tant que méthodes de classe. Du point de vue JUnit, cela n'a aucune importance, on peut tester des méthodes de classe comme on peut tester des méthodes d'instance.*

```
public class Exemple {

        private int result;
        private String nom;
        private int valeur;
        private Object object1 = new Object();
        private Object object2 = new Object();

        public Exemple(){}

        public void incVal() {valeur++;}
        public int getVal() {return valeur;}
```

```
	public int getResult(){
		return result;
	}

	public String getNom(){
		return nom;
	}

	public Object getObject1(){
		return object1;
	}

	public Object getObject2(){
		return object2;
	}

	public void factoriel(int n){
		int prov;
		prov = 1;
		for(int i = 1; i <= n; i++)
			prov = prov*i;
		result = prov;
	}

	public void min(int a, int b){
		if(a>b)
			result = b;
		else
			result = a;
	}

	public void plusGrand(){
		nom = "Toto";
	}

	public void plusPetit(){
		nom = "Jean";
	}

	public Object renvoiNull(){
		return null;
	}
}
```

3 *Utilisation de JUnit*

3.1 CREATION DE LA CLASSE DE TEST

Afin de créer un test sur une classe existante, il faut tout simplement créer une nouvelle classe qui doit hériter de la classe « `TestCase` ». Par convention, il faut nommer cette nouvelle classe du

même nom que celle qui est testée (dans notre cas : « `Exemple` »), suivi de « `Test` ». Cet héritage est utilisé dans le cas où nous utilisons la version 3 de JUnit. Cela se présente comme ci-dessous :

```
import junit.framework.TestCase;

public class ExempleTest extends TestCase {…}
```

Si vous utilisez la version 4 de JUnit. La classe de test se présenterait comme ci-dessous. L'héritage n'est plus utile.

```
public class ExempleTest {…}
```

3.2 CREATION DES METHODES DE TEST UNITAIRE

Les différentes méthodes de tests doivent commencer par le mot « `test` » (en minuscule), suivi du nom que l'on désire. Cette règle n'est plus obligatoire pour la version 4 de JUnit. Par contre le tag « `@Test` » doit être présent en dessus de la méthode. Dans tous les cas, la méthode doit être déclarée public et ne doit rien renvoyer (`void`). Voici un exemple permettant de tester la méthode `factoriel` de la classe « `Exemple` ». Elle est compatible avec les versions 3 et 4 de JUnit.

```
public class ExempleTest etc..

	private Exemple exemple = new Exemple() ;

	@Test
	public void testFactoriel() {
		exemple.factoriel(4);
		assertEquals(exemple.getResult(),24);	assertEquals
							⇨ voir ci-dessous

	}
}
```

3.3 LA CLASSE ASSERT

Comme vous pouvez le voir, la méthode « `assertEquals` » est utilisée afin de créer le test unitaire de comparaison des 2 nombres dans cet exemple. Il existe diverses méthodes dans la classe «`junit.framework.Assert`» qui vont être décrites ci-dessous.

Pour plus de détail, voir la documentation :

http://www.loria.fr/~cirstea/TEACHING/TESTS/javadoc/junit/framework/package-summary.html

- `assertEquals`

  Permet de tester si deux types primitifs sont égaux (`boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short`). L'égalité de deux objets peut être testée également (attention, ce n'est pas un test sur la référence). Pour les `double` et les `float`, il est possible de spécifier un delta, pour lequel le test d'égalité passera quand même.

- `assertFalse` et `assertTrue`

  Permet de tester une condition booléenne.

- `assertNotNull` et `assertNull`

  Permet de tester si une référence est (non) nulle.

- `assertNotSame` et `assertSame`

  Permet de tester si deux « `Object` » (Java) se réfèrent ou non au même objet.

- `fail`

  Permet de faire échouer sans condition. En cas d'utilisation de `fail`, il est conseillé de faire figurer un message indiquant pourquoi le test à échouer.

3.4 METHODES DE DEBUT ET DE FIN

Une grande partie du code permettant la réalisation d'un test unitaire sert à établir les conditions d'exécution du test. Il se peut qu'au sein d'une même classe de `TestCase`, les différentes méthodes de tests aient besoin d'une initialisation commune. (Par exemple l'ouverture d'un fichier). Dans la même optique, elles ont peut-être également besoin d'une procédure de fin commune (Par exemple la fermeture du fichier).

Le framework JUnit fournit ces deux méthodes sous les noms de « `setUp()` » et « `tearDown()` ». Par défaut, elles sont lancées automatiquement au début et à la fin de chaque test unitaire. C'est donc libre à vous de les implémenter selon vos besoins. Dans la version 4 de JUnit il faut rajouter le tag « `@Before` » devant la méthode « `setUp()` » et « `@After` » devant celle du « `tearDown()` » comme dans l'exemple qui suit :

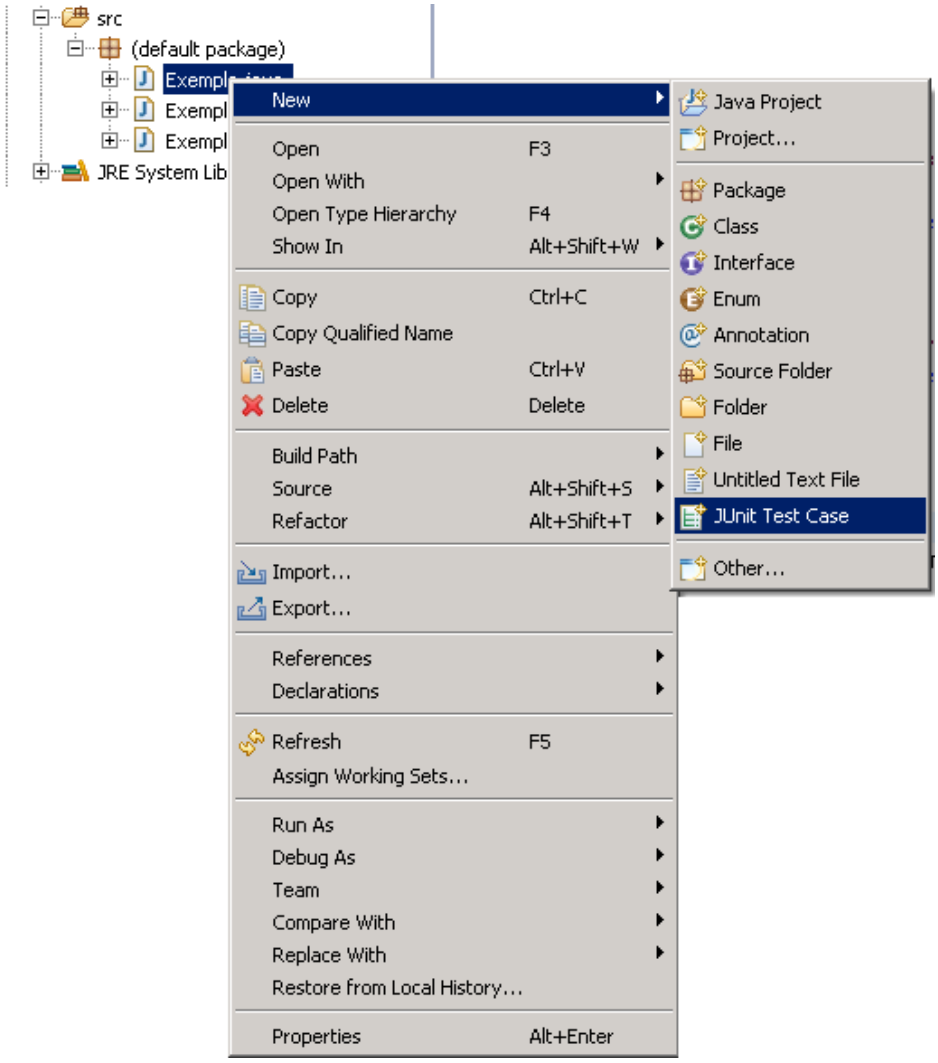
```
private static int numero = 0;

@Before
public void setUp() throws Exception {
	numero++;
	System.out.println("Le test numéro "+numero+" a commencé");
}

@After
public void tearDown() throws Exception {
	System.out.println("Le test numéro "+numero+" est terminé");
}
```

☺ Notons qu'avec la version 4 de JUnit, les méthodes de début et de fin n'ont plus besoin de s'appeler `setUp` ou `tearDown`.

4 Mise en place sous Eclipse

4.1 Generer la classe de test sous Eclipse

La création de la classe de test unitaire est très simplifiée sous Eclipse. Il vous faut tout d'abord sélectionner la classe qui va être testée puis – au moyen d'un clic droit -, choisir de créer un nouveau JUnit Test Case. La création est également possible depuis le menu Fichier → Nouveau.



Dans la fenêtre suivant, choisissez le nom de la classe de test. Par défaut, le nom correspond à votre classe testée suivit du mot « `Test` ». Vous pouvez également choisir la version de JUnit à utiliser.

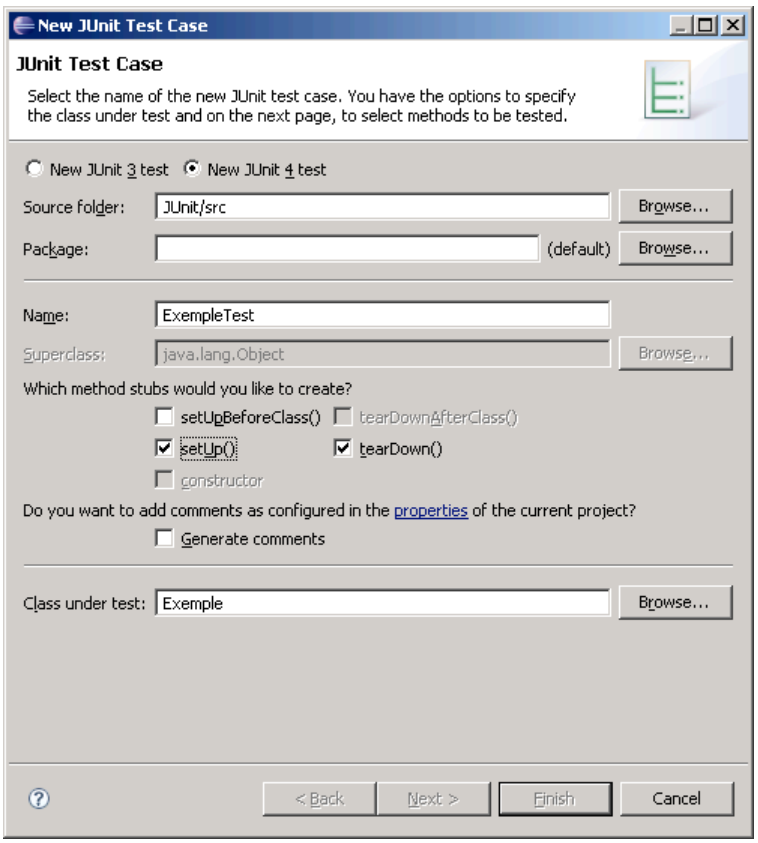
Méthodes **`setUp()`** et **`tearDown()`**

⇨Dans la partie du bas, vous pouvez ajouter les méthodes « `setUp()` » et « `tearDown()` » vues précédemment.

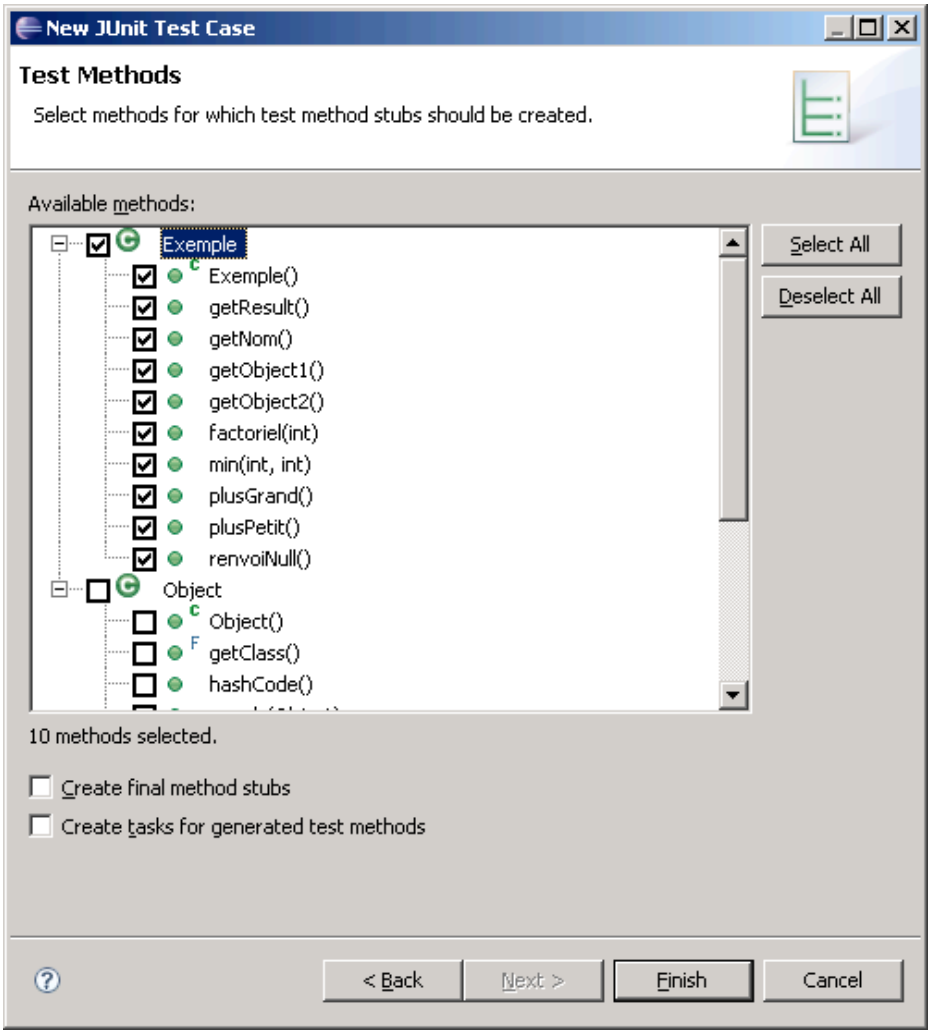
Méthodes **setUpBeforeClass()** & **tearDownAfterClass()**

⇨ Plutôt que d'encadrer chaque méthode de test, Il existe encore 2 méthodes qui permettent d'encadrer **toute la série** de tests. Elles seront exécutées au début et à la fin de la série de tests et nommées respectivement «**setUpBeforeClass()**» et «**tearDownAfterClass()** ».

Dans la version 4 de JUnit, il est possible de précéder une méthode par le symbole `@Beforeclass` (resp. `@Afterclass`) pour la spécifier en tant que méthode d'encadrement.



Une fois les paramètres choisis, allez sur la fenêtre suivante (Next). Vous pouvez alors choisir les méthodes de votre classe principale qui vont être testées. Il est bien sur possible d'en rajouter par la suite. Cette option permet uniquement de générer le code des méthodes de tests automatiquement. Il vous reste alors à les implémenter. Une fois votre choix effectué, appuyez sur le bouton terminer (Finish).



4.2 Classe ExempleTest

Cette classe permet de donner un aperçu des différents tests unitaires décrits plus haut. Compétez la classe générée par l'environnement JUnit et testez le résultat.

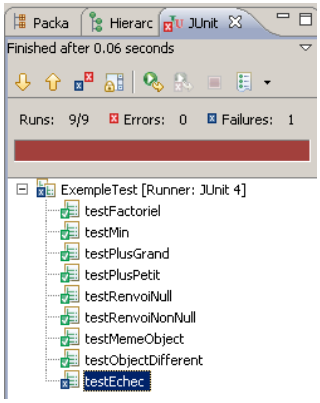
```
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
```

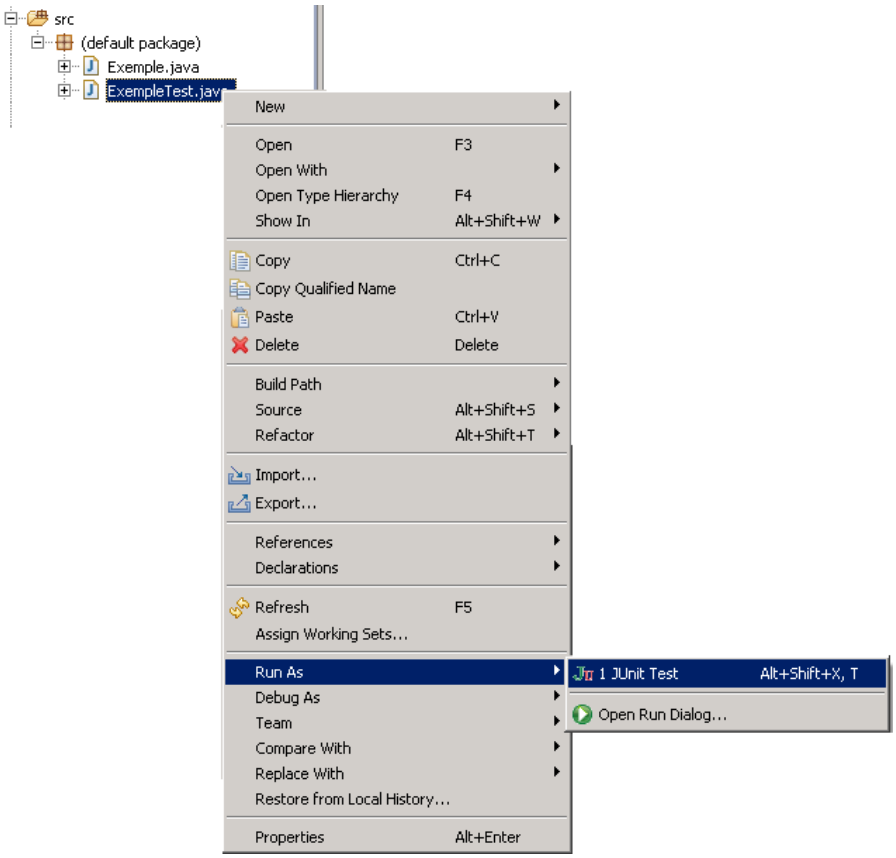
```
import org.junit.BeforeClass;
import org.junit.Test;

public class ExempleTest {

	private static int numero = 0;
	private Exemple exemple = new Exemple();
	private static boolean erreur;

	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
		erreur = true;
	}

	@Before
	public void setUp() throws Exception {
		numero++;
		System.out.println("Le test numéro "+numero+" a commencé");
	}

	@After
	public void tearDown() throws Exception {
		System.out.println("Le test numéro "+numero+" est terminé");
		System.out.println("Etat de la valeur: "+ exemple.getVal());
		exemple.incVal();
	}

	@Test
	public void testFactoriel() {
		exemple.factoriel(4);
		assertEquals(exemple.getResult(),24);
	}

	@Test
	public void testMin() {
		exemple.min(5, 6);
		assert(exemple.getResult()==5);
	}

	@Test
	public void testPlusGrand() {
		exemple.plusGrand();
		assertTrue(exemple.getNom().equals("Toto"));
	}

	@Test
	public void testPlusPetit() {
		exemple.plusPetit();
		assertFalse(exemple.getNom().equals("Toto"));
	}

	@Test
	public void testRenvoiNull() {
		assertNull(exemple.renvoiNull());
	}

	@Test
	public void testRenvoiNonNull() {
```

```
            assertNotNull(exemple.getObject1());
      }

      @Test
      public void testMemeObject() {
            assertSame(exemple.getObject1(),exemple.getObject1());
      }

      @Test
      public void testObjectDifferent() {
            assertNotSame(exemple.getObject1(),exemple.getObject2());
      }

      @Test
      public void testEchec() {
            if(erreur)
                  fail("Echec");
      }

}
```

4.3 EXECUTION DES TESTS

Afin de lancer l'exécution des différents tests unitaires, il suffit de sélectionner la classe de test puis de faire un click droit par dessus. Choisissez ensuite dans le menu l'option Run as → JUnit Test. Votre programme de tests est alors exécuté. Vous observez alors la fenêtre JUnit qui vous donne l'état d'exécution des différents tests.





➔Essayez !

Et vous constarez (manipulation de la variable « `valeur` » de l'objet « `exemple` »)..

Les tests JUNIT sont stateless! (sans état)

A chaque test une nouvelle instance de la classe Test est créée!

==> L'état des variables d'instance éventuelles déclarées dans la classe de Test n'est donc pas sauvegardé d'un test à l'autre !!

==> Les informations générales à tous les tests doivent être enregistrés dans des variables de classe

En conséquence:

Les tests sont tous indépendants les uns des autres..

==> peuvent s'exécuter dans n'importe quel ordre

==> peuvent être exécutés en parallèle!!

5 But du laboratoire

En annexe de la donnée, vous trouverez les classes « Personne.java » et « Annuaire.java ». La première classe permet de représenter de manière très simple une personne. La personne est caractérisée par un nom, un prénom, une année de naissance et une adresse e-mail. Diverses méthodes existent afin de modifier ou récupérer ces informations. La deuxième classe regroupe un ensemble de personnes. Elle permet l'ajout et la suppression d'une personne à l'annuaire. Elle fournit également les méthodes permettant de récupérer les données relatives à une personne. L'adresse e-mail permet de différencier les différentes personnes étant donné qu'elle est unique.

Nous vous demandons de créer une ou deux classes de tests unitaires sur ces deux classes. Vous devez utiliser au moins une fois chacune des méthodes de la classe « `Assert` » vu plus haut ainsi que les méthodes de début et de fin de classe et de test (« `setUp()` » et « `tearDown()` ») .