Utilisation de JUnit4 de Java sous Eclipse

Utilisation de la bibliothèque Java JUnit pour le test unitaire :

a) Introduction :

JUnit est framework de test open source pour les programmeurs Java. Le programmeur Java peut créer des cas de test et tester son propre code. C'est l'un des Frameworks de tests unitaires.Sous Eclipse, il y'a trois versions : 3, 4 et 5.Pour effectuer des tests unitaires, nous devons créer des cas de test. Le cas de test unitaire est un code qui garantit que la logique du programme fonctionne comme prévu.

Le package org.junit contient de nombreuses interfaces et classes pour les tests junit tels que Assert, Test, Before, After etc.

Types de tests unitaires : Il existe deux façons d'effectuer des tests unitaires :

1) Test manuel Si vous exécutez les cas de test manuellement sans aucune prise en charge d'outil, on parle de test manuel. Cela prend du temps et est moins fiable.

2)Tests automatisés Si vous exécutez les cas de test par support d'outil, cela s'appelle des tests automatisés. C'est rapide et plus fiable.

Annotations pour les tests Junit :

Le framework Junit 4.x est basé sur des annotations, voyons donc les annotations qui peuvent être utilisées lors de l'écriture des cas de test.

- L'annotation **@Test** spécifie que la méthode est la méthode de test.
- L'annotation **@Test(timeout=1000)** spécifie que la méthode échouera si elle prend plus de 1000 millisecondes (1 seconde).
- L'annotation **@BeforeClass** spécifie que la méthode ne sera invoquée qu'une seule fois, avant de démarrer tous les tests.
- L'annotation **@Before** spécifie que la méthode sera invoquée avant chaque test.
- **L**'annotation **@After** spécifie que la méthode sera invoquée après chaque test.
- L'annotation **@AfterClass** spécifie que la méthode ne sera invoquée qu'une seule fois, après avoir terminé tous les tests.

Assertion de classe :

La classe org.junit.Assert fournit des méthodes pour affirmer la logique du programme.
Méthodes de la classe Assert Les méthodes courantes de la classe Assert sont les suivantes :

- void **assertEqual**s (booléen attendu, booléen réel) : vérifie que deux primitives/objets sont égaux. Il est surchargé.
- void **assertTrue**(condition booléenne) : vérifie qu'une condition est vraie.
- void **assertFalse**(condition booléenne) : vérifie qu'une condition est fausse.
- void **assertNull**(Object obj) : vérifie que l'objet est nul.
- void **assertNotNull**(Object obj) : vérifie que l'objet n'est pas nul.

Fichiers jar requis :

Vous devez charger les fichiers **junit4.jar** et **hamcrest-core.jar**.

b) Quels sont les avantages de Junit4 par rapport à Junit3 :

- Annotations Java 5 pour la configuration et le démontage (@before et @after) au lieu de setUp() et tearDown().
- On n'a plus besoin d'étendre **TestCase**.
- L'annotation @Test remplace la convention de dénomination testSomeMethod().
- Importations statiques pour les assertions.
- Les théories Junit, qui permettent de séparer les ensembles de données du test lui-même.

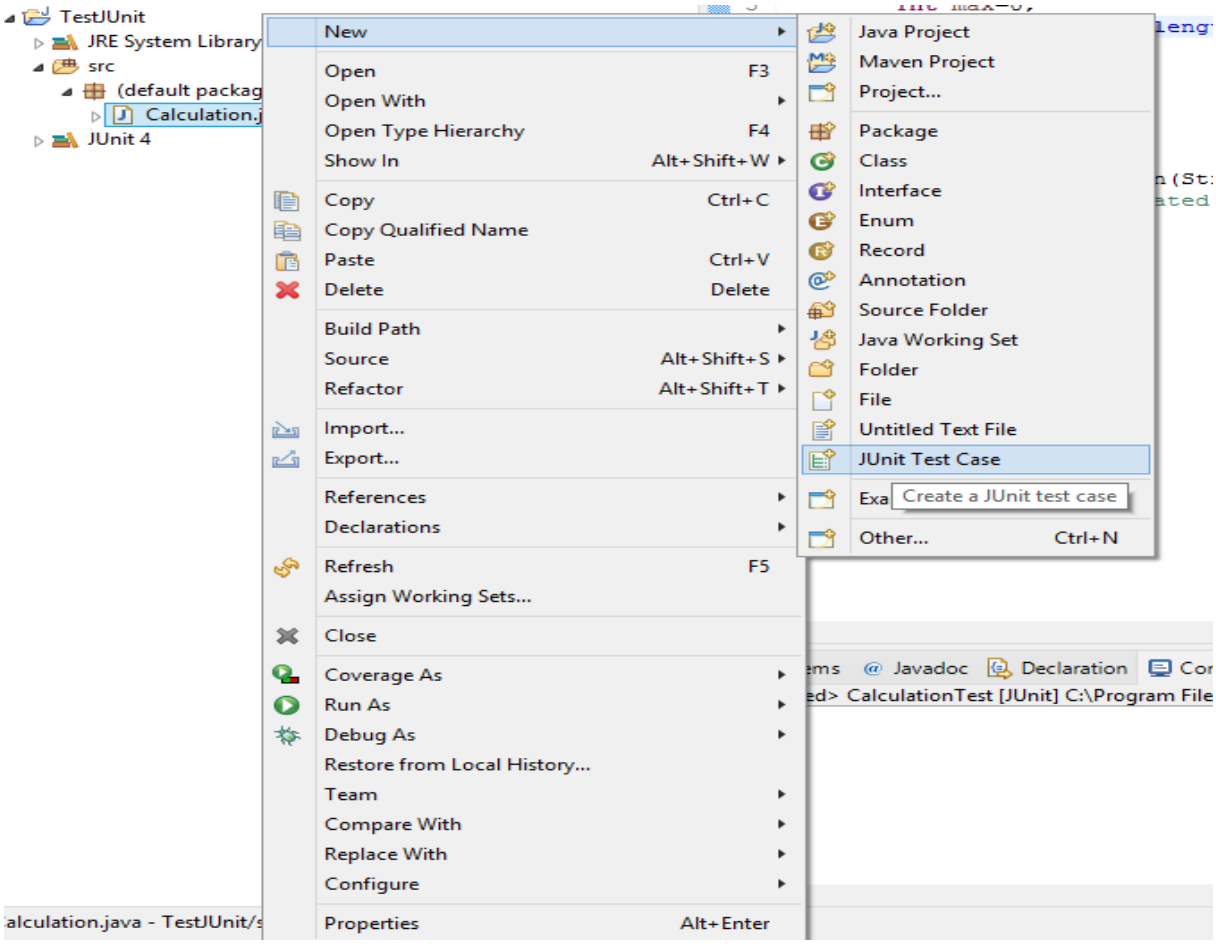
c) Exemple sur l'utilisation de JUnit4 :

1) On doit avoir une classe à tester : cette classe contient des méthodes. A titre d'exemple on peut tester la classe suivante « `Calculation` » qui contient une seule méthode «`findMax`» qui permet de trouver le max dans un tableau :

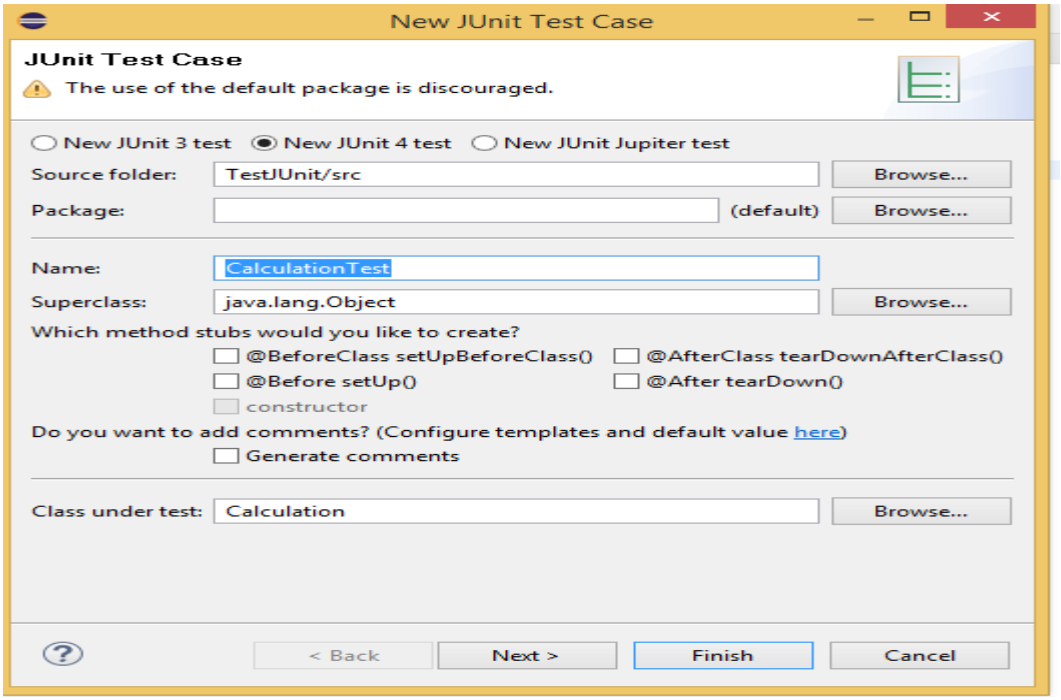
```
Public class Calculation {
public static int findMax(int arr[]){
int max=0;
for(inti=1;i<arr.length;i++){
if(max<arr[i])
max=arr[i];
}
Return max;
    }

      }
```

2) Pour tester la classe précédente, on doit définir une classe de test à part par exemple on appelle cette classe « `ClalculationTest`» qui va utiliser le concept d'assertion afin de tester les méthodes de classe précédente. Dans la classe `ClalculationTest`on met une méthode dite « `testFindMax` » qui va tester la méthode « `findMax` » de la classe « `Calculation` ». La méthode « `testFindMax` » contient par exemple deux assertions qui permettent de voir est ce que la méthode « `testFindMax` » fonctionne correctement ou pas.

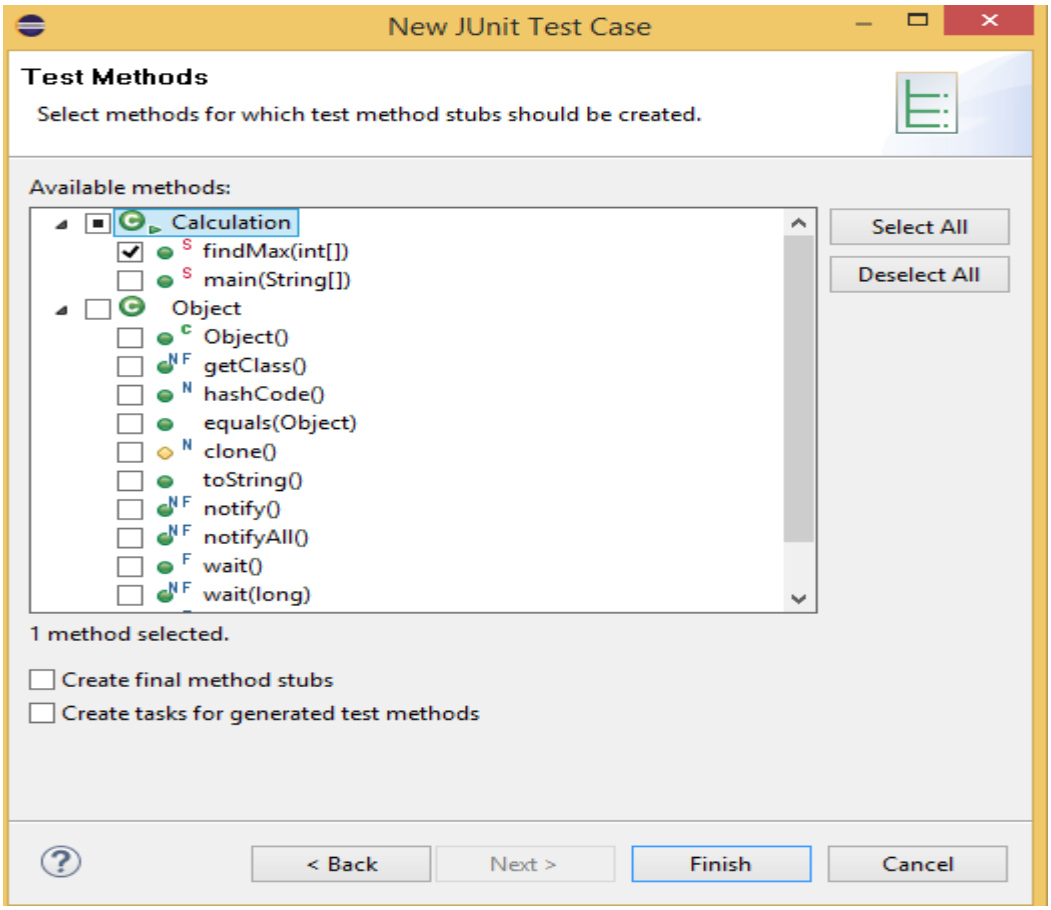
   Directement depuis IDE Eclipse :



Ensuite :



Choisir les méthodes à tester :



Vous allez avoir ce code :

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

import org.junit.Test;

public class CalculationTest {

	@Test
	public void testFindMax() {
		fail("Not yet implemented");
	}

}
```

Ajouter les tests souhaités

```
Public class ClalculationTest{

@Test
Public void testFindMax(){
   assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));
   assertEquals(-1,Calculation.findMax(new int[]{-12,-1,-3,-4,-2}));
   }
}
```

Par exemple :

La première assertion :

`assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));`

confirme que si la méthode findMax est correcte elle doit retourner la valeur 4 quand le tableau donné est [1,3,4,2].

La 2eme assertion :

`assertEquals(-1,Calculation.findMax(new int[]{-12,-1,-3,-4,-2}));`

confirme que si la méthode findMax est correcte elle doit retourner la valeur -1 quand le tableau donné est [-12,-1,-3,-4,-2].

3) Maintenant, les deux classes seront mises dans le package:

```
package pack;
```

et dans ce cas les codes complets des classes seront comme suit :

Classe 1 :

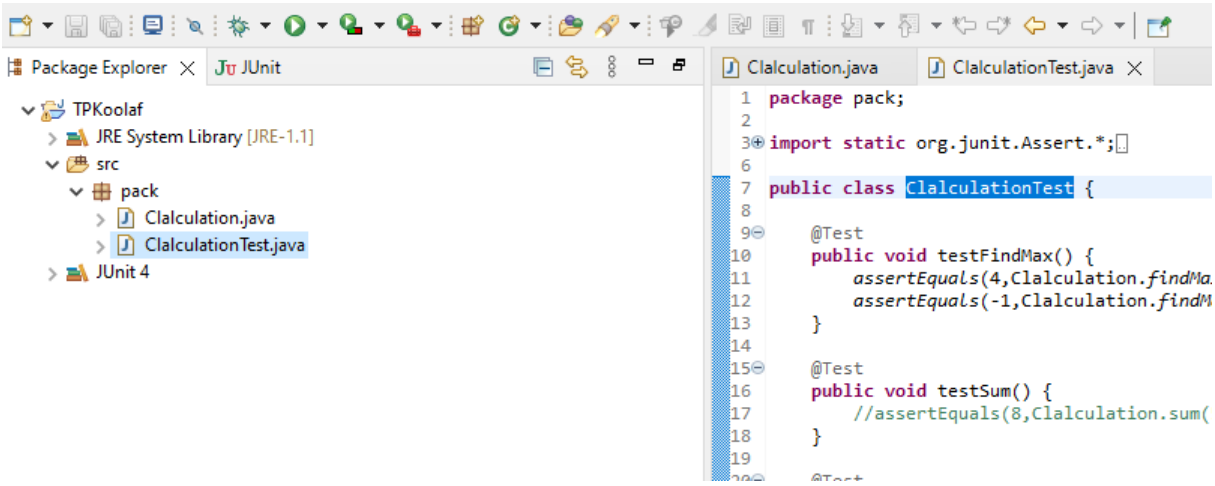
```
package pack;

public class Calculation {
public static int findMax(intarr[]){
int max=0;
for(inti=1;i<arr.length;i++){
if(max<arr[i])
max=arr[i];
}
return max;
    }
// public static void main(String arg[]) {
//    System.out.println("Good");
// }

}
```

Classe 2 :

```
package pack;

import static org.junit.Assert.*;
import com.javatpoint.logic.*;
import org.junit.Test;

public class ClalculationTest{
```

```
        @Test
        Public void testFindMax(){
        assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));
        assertEquals(-1,Calculation.findMax(new int[]{-12,-1,-3,-4,-2}));
        }
}
```

4) Bien sûr dans ce cas, la structure du projet Java sera comme suit :



Dans notre projet, on a utilisé la version JUnit4 comme il est clair dans la structure du projet.

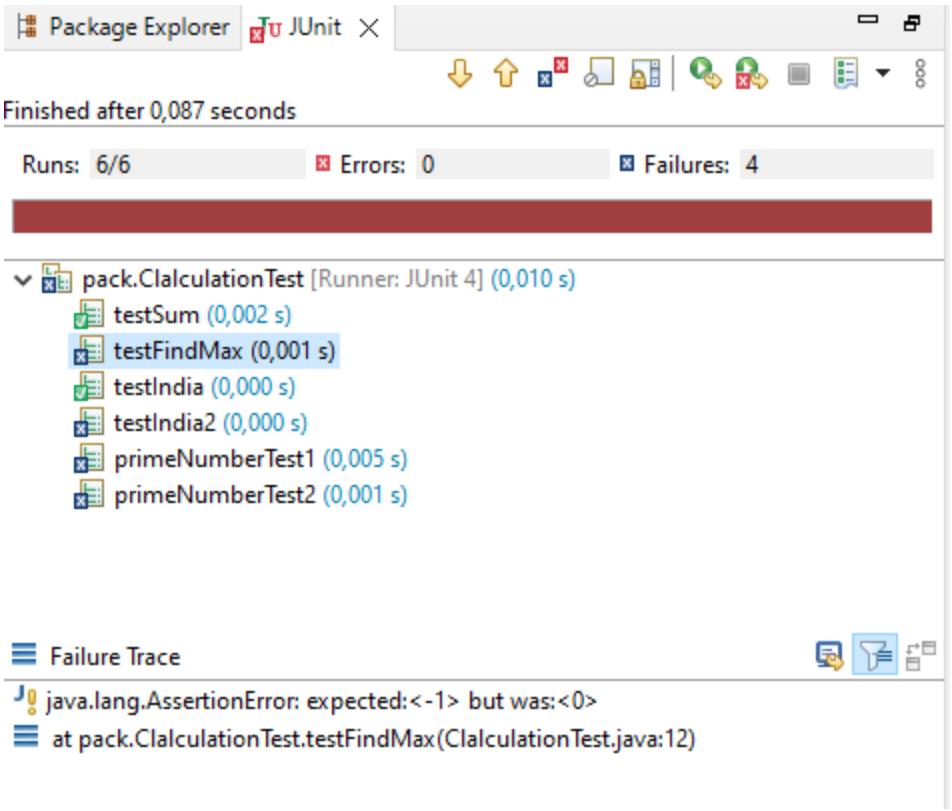
5) Maintenant vient la phase de test unitaire automatique réalisé par la librairie JUnit4 :

Il suffit d'exécuter la classe de test : « `ClalculationTest`»

Pour faire cette exécution, on doit faire :

right click on `ClalculationTest`class -> Run As -> 1Junit Test.

6) Une fois la classe est exécutée, on va avoir les résultats suivants :



Comme on voit sur ce résultat, il y'a une erreur détectée dans la deuxième assertion. On voit dans l'onglet Failure Trace, que normalement la valeur attendue c'est (-1) expected value <-1> mais la valeur retournée par la méthode était 0 (but was : <0>).
Explication : Cette erreur est normale car la méthode testée était incorrecte. Si on revoit le code de nouveau de notre méthode :

```
public static int findMax(intarr[]){
int max=0;
for(inti=1;i<arr.length;i++){
if(max<arr[i])
max=arr[i];
}
return max;
    }
```

On voit que la méthode commence par mettre 0 dans max. Donc, si toutes les valeurs du tableau sont négatives comme notre cas dans la deuxième assertion : {-12,-1,-3,-4,-2}donc le max va rester 0 et ne va pas changer d'où il y'a une erreur dans le code de cette méthode

7) Corriger l'erreur dans cette méthode :
Après qu'on a trouvé cette erreur, on peut facilement corriger le code de nouveau pour qu'il soit comme suit :

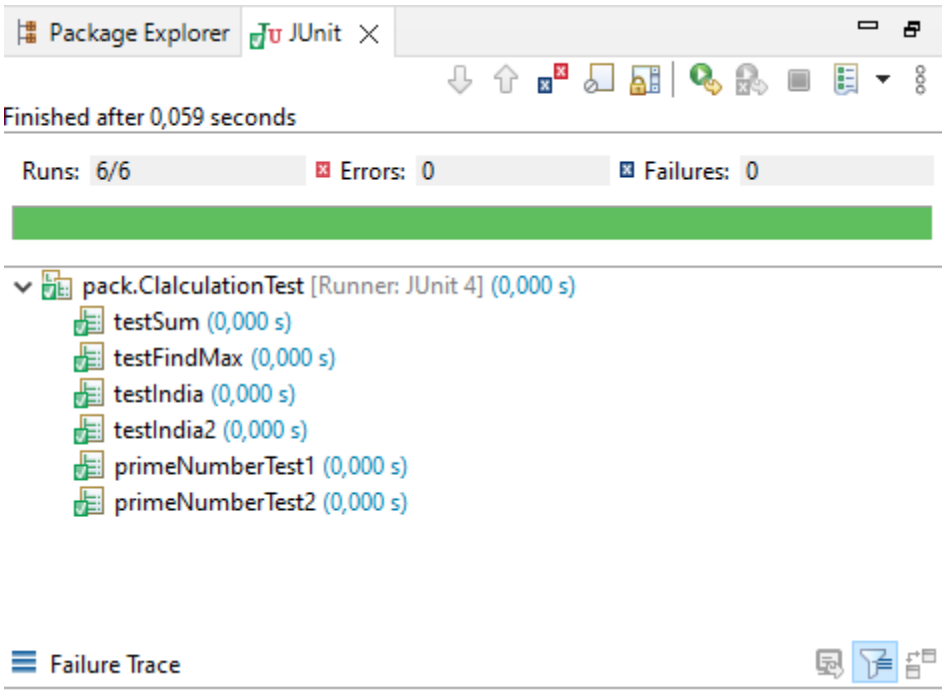
```
1. package pack;
2. public class ClalculationTest{
```

```
3.      public static int findMax(int arr[]){
4.        int max=arr[0];//arr[0] instead of 0
5.        for(int i=1;i<arr.length;i++){
6.           if(max<arr[i])
7.              max=arr[i];
8.        }
9.        return max;
10.    }
11. }
```

Maintenant, ce code si on le test de nouveau on oura pas des erreurs

8) Test de nouveau :
right click on `ClalculationTest` class -> Run As -> 1Junit Test.

On aura cette fois, le résultat suivant :



9) Fin du test avec JUnit.