Junit

Flexitime – Outil de gestion des emplois du temps

Projet de Génie Logiciel – 3ème année - Ingénieurs 2000

Auteur :	**Adrien Bouvet**	**Destinataire(s) :**	**FlexiTeam**
Date de création :	**22/11/2004**	**Référence :**	**JUnit-2004-11-25v1.0.doc**

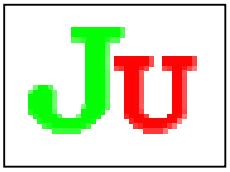
HISTORIQUE

Date	Version	Modifié par	Motifs
2004/11/22	0.1	Adrien BOUVET	Creation
2004/11/23	0.2	Adrien BOUVET	Intégration de la charte graphique
2004/11/25	0.3	Adrien BOUVET	Partie Installation + utilisation + exemple + zone JUnit
2004/11/25	1.0	Adrien BOUVET	Préambule + conclusion

CONTACTS

Société	Nom	Fonction
FlexiTime	Adrien BOUVET	Responsable Qualité + Web

PREAMBULE



JUnit est un plug-in d'éclipse permettant de réaliser des tests unitaires sur les classes d'un projet JAVA.

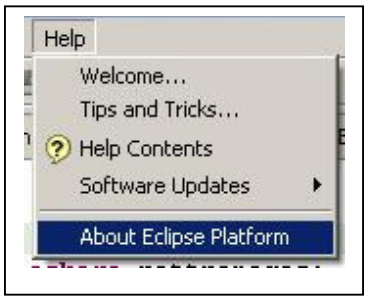
JUnit s'utilise facilement, le plus difficile étant de définir des tests « clés » et révélateurs de problèmes susceptibles d'exister dans des méthodes.

Voyons comment s'installe et s'utilise JUnit, notamment au travers d'exemples simples.

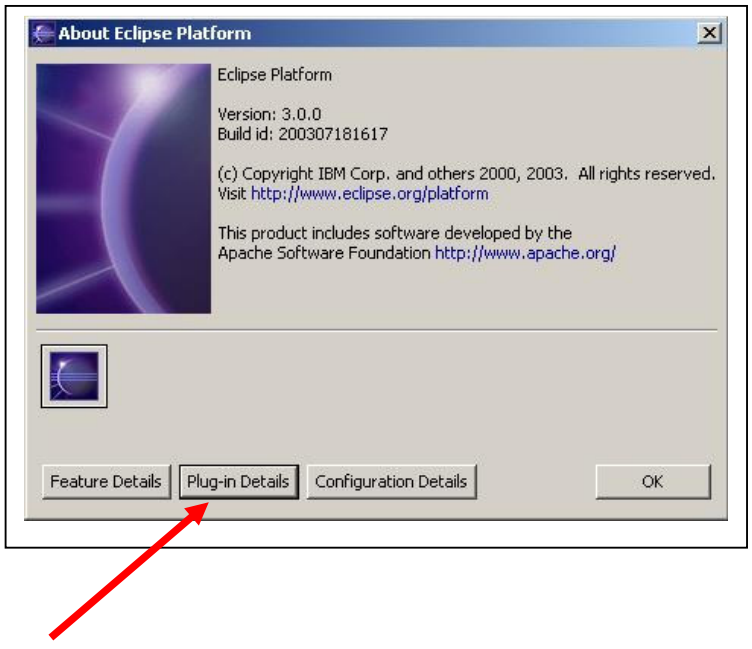
TABLE DES MATIERES

1 Installer JUnit sur Eclipse

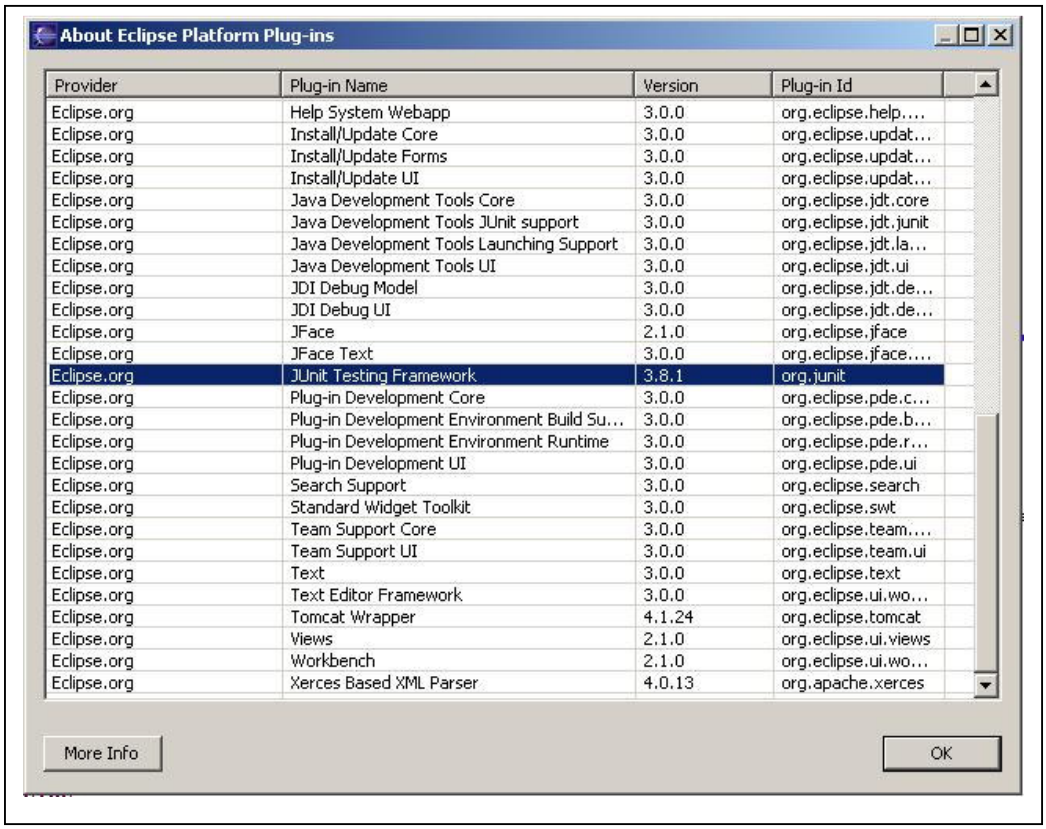
Par défaut, avec Eclipse 3.0.0 (et supérieur je pense), junit est intégré dans la liste des plug-ins disponibles. Pour le vérifier il faut se rendre dans le menu Help puis About Eclipse Platform.



Dans la fenêtre qui s'ouvre il faut cliquer sur le bouton *Plug-in Details* afin d'afficher la liste des plug-ins intégrés dans Eclipse.



Dans la liste qui apparaît dans la nouvelle fenêtre, il faut vérifier l'existence d'une ligne concernant Junit, telle que sur le screenshot suivant.



On remarque ici, que le plug-in *JUnit* est en version 3.8.1, c'est la dernière version stable de *JUnit* disponible sur le web (au 24/11/2004).

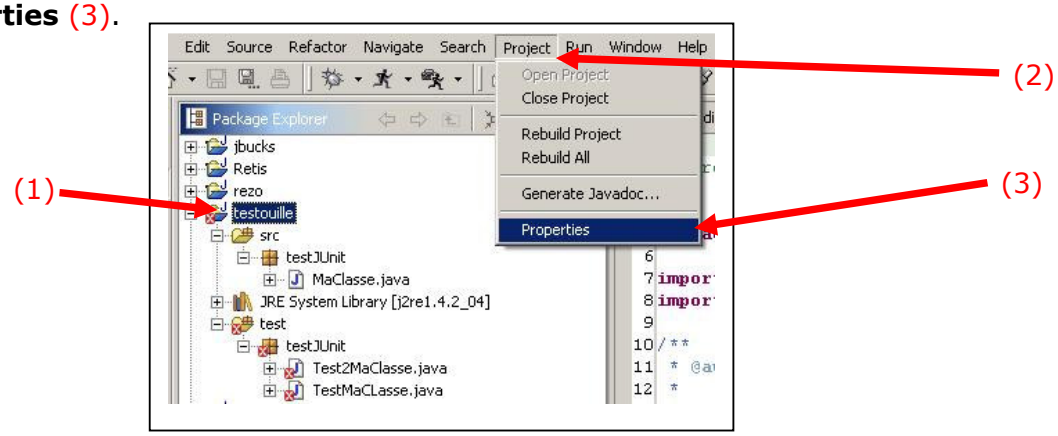
Si cette ligne n'est pas visible, il faut ajouter ce plug-in à la main. Pour cela, il faut télécharger *JUnit* sur le site web www.junit.org.

Puis il faut ajouter le dossier *JUnit* dans le repertoire plug-ins d'Eclipse (exemple : *D:\eclipse\plugins*). Après avoir relancer Eclipse la ligne concernant *JUnit* devrait être visible dans la liste des plug-ins.

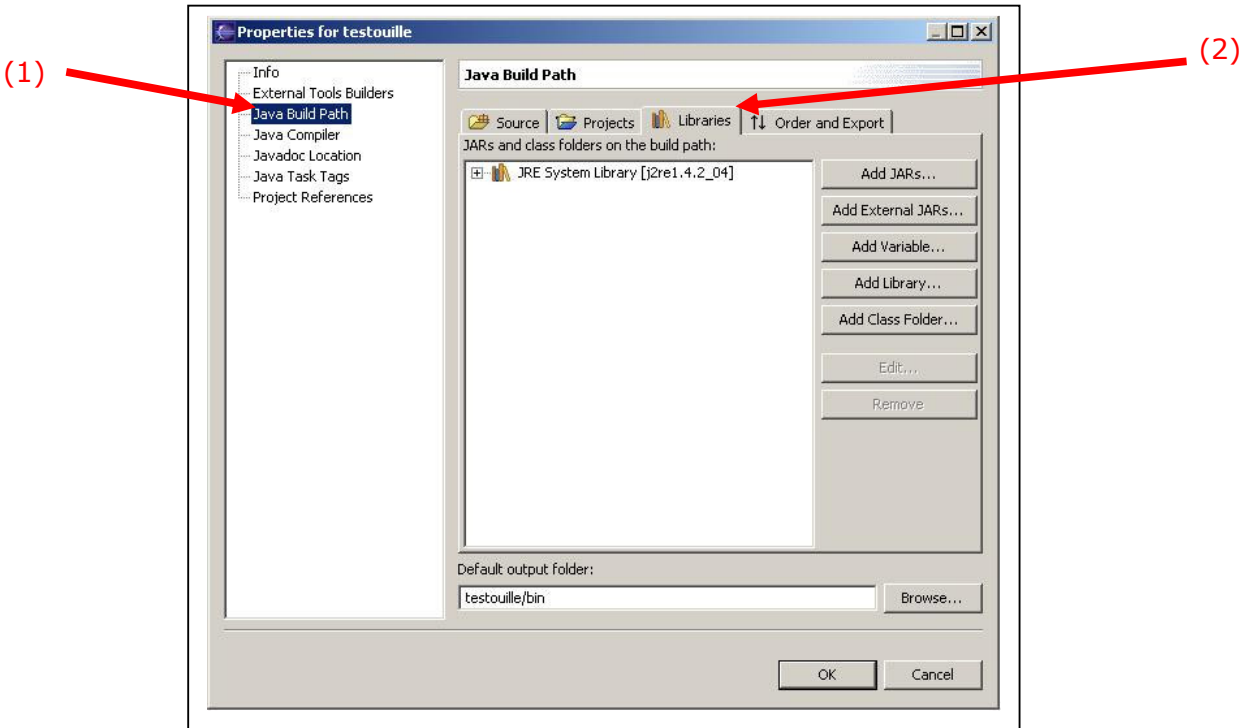
2 Comment se servir de JUnit ?

Avant toute chose, on va séparer les classes de notre applicatin des classes de test en créant deux dossiers (*source folder*), l'un nommé *src* et l'autre *test*. C'est ce dernier qui va contenir toutes les classes de test utilisant JUnit.

Il faut tout d'abord importer les paquetages *JUnit* dans notre *Projet Eclipse*. Pour cela il faut sélectionner le projet (1) , puis cliquer sur le menu **Project** (2), puis **Properties** (3).

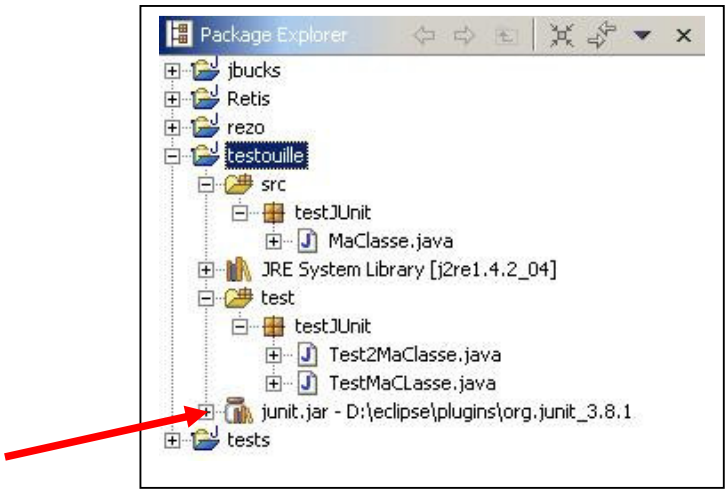


Dans la fenêtre qui s'affiche, il faut se rendre dans la partie *Java Build Path (1)*, puis dans l'onglet *Libraries (2)*. Cet onglet contient l'ensemble des librairies importées pour notre projet, on remarque que par défaut il y a le Java Runtime Environnement (JRE) de présent.



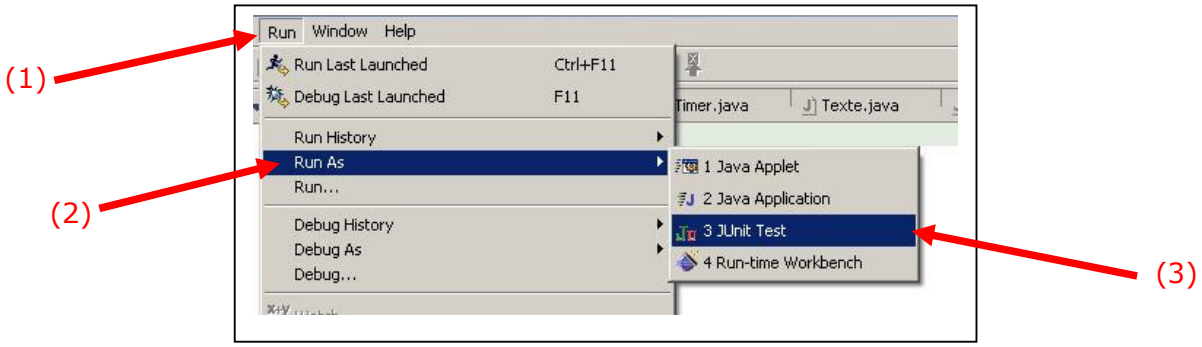
A partir de cette fenêtre on va ajouter la librairie JUnit. Il faut cliquer sur le bouton *Add External JARs*. Dans la nouvelle fenêtre il faut parcourir le disque dur afin de retrouver *junit.jar*.

Une fois ajouté, on voit dans notre projet la bibliothèque JUnit.



Ensuite il faut créer une classe de test qui étend *TestCase* et qui va contenir les tests à effectuer.

Pour exécuter cette classe de test, il faut se rendre dans **Run (1)** -> **Run As... (2)** -> **3 JUnit Test (3)**.



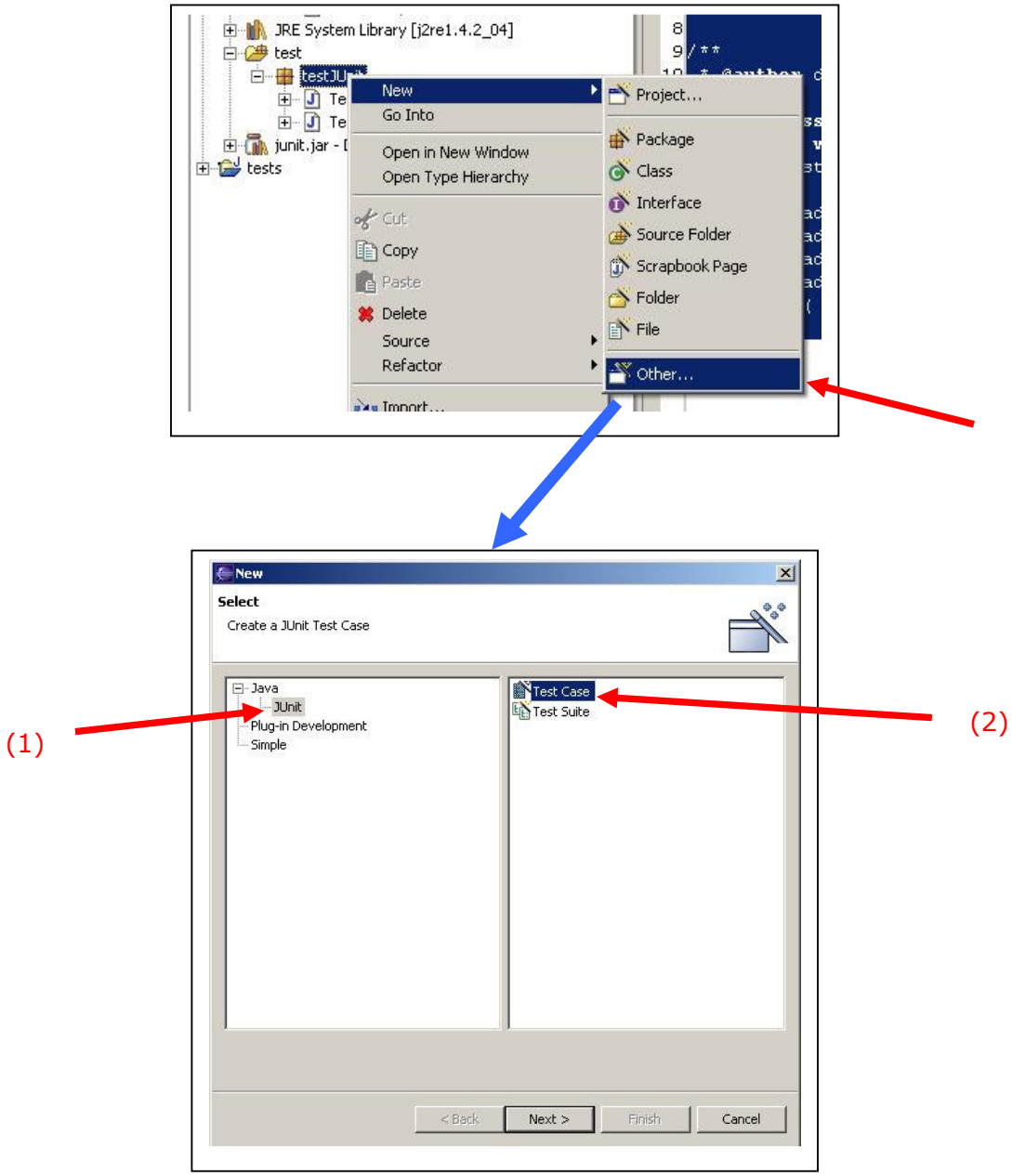
3 Exemples de tests possible

3.1 Test basique

Nous allons illustrer JUnit par un exemple basique. Pour cela on cré une classe *MaClasse.java* qui sera notre classe à tester. Voici le contenu de cette classe :

```
/*
 * Created on 24 nov. 2004
 */
package testJUnit;

import java.util.*;

/**
 * @author diam
 */
public class MaClasse {
	public static void main(String[] args) {
		List l = new ArrayList();
		int res =0;

		l.add(0, new Integer(10));
		l.add(1, new Integer(20));
		l.add(2, new Integer(30));
		l.add(3, new Integer(40));
		res = calc(l);
		System.out.println("Resultat1 = "+res);
	}

	static int calc(List l) {
		return ( ((Integer)l.get(0)).intValue() +
((Integer)l.get(2)).intValue() );
	}
}
```

Puis dans le dossier *test* on va créer une classe *TestMaClasse.java* qui va tester la méthode d'addition *calc()*. La classe de test n'est pas une classe java classique, pour la créer il faut donc aller dans *New -> Other…* .



Dans cette fenêtre il faut choisir *Java -> JUnit* (1) puis Test Case (2).

Voici un exemple de classe de test :

```
/*
 * Created on 24 nov. 2004
 */
package testJUnit;

import java.util.*;
import junit.framework.TestCase;

/**
 * @author diam
 */
public class TestMaCLasse extends TestCase {
	public void testCalc() {
		List l = new ArrayList();

		l.add(0, new Integer(10));
		l.add(1, new Integer(20));
		l.add(2, new Integer(30));
		l.add(3, new Integer(40));
		if( MaClasse.calc(l) != 40 ) fail();  //(1)
	}
}
```

Cette classe sert à tester la méthode *calc()* de la classe MAClasse.java. On écrit donc une méthode *testCalc()* dans laquelle on va appeler la méthode *calc()*.

Le principe de nommage est : ***void test<nom_classe_a_tester>()***.

Dans la méthode *testCalc()*, on recré les variables puis on effectue un test avec *if()* dans lequel on appelle la méthode *calc()* et on compare son résultat à celui que l'on est sensé obtenir.

Si le résultat est différent de celui attendu, la méthode *fail()* est appelée (1). Elle notifie à JUnit que le test a échoué.

3.2 Test avancé

Le plus souvent, une classe contient plusieurs méthodes, il est donc intéressant de pouvoir tester chacune de ses méthodes une par une mais dans une unique classe de test (au lieu d'une classe de test par méthode).

Voici la classe à tester :

```
/*
 * Created on 24 nov. 2004
 */
package testJUnit;

import java.util.*;

/**
 * @author diam
 */
public class MaClasse {
	public static void main(String[] args) {
		List l = new ArrayList();
		int res =0;

		l.add(0, new Integer(10));
		l.add(1, new Integer(20));
		l.add(2, new Integer(30));
		l.add(3, new Integer(40));
		res = calc(l);
		System.out.println("Resultat1 = "+res);
		res = calc2(l);
		System.out.println("Resultat2 = "+res);
	}

	static int calc(List l) {
		return ( ((Integer)l.get(0)).intValue() + ((Integer)l.get(2)).intValue() );
	}

	static int calc2(List l) {
		return ( ((Integer)l.get(1)).intValue() + ((Integer)l.get(3)).intValue() );
	}
}
```

Cette classe dispose de deux méthodes *calc()* et *calc2()*. Chacune effectuant une action différente.

Voici la classe de test :

```
/*
 * Created on 25 nov. 2004
 */
package testJUnit;

import java.util.*;
import junit.framework.TestCase;

/**
 * @author Administrateur
 */
public class Test2MaClasse extends TestCase {

	static private List l = new ArrayList();
	static {
		l.add(0, new Integer(10));
		l.add(1, new Integer(20));
		l.add(2, new Integer(30));
		l.add(3, new Integer(40));
	}

	public void testCalc() {
		System.out.println("test de comment");
		if( MaClasse.calc(l) != 40  ) fail("boulettttttttt");
	}

	public void testCalc2() {
		if( MaClasse.calc2(l) !=  60 ) fail("boulettttttttt2222");
	}
}
```

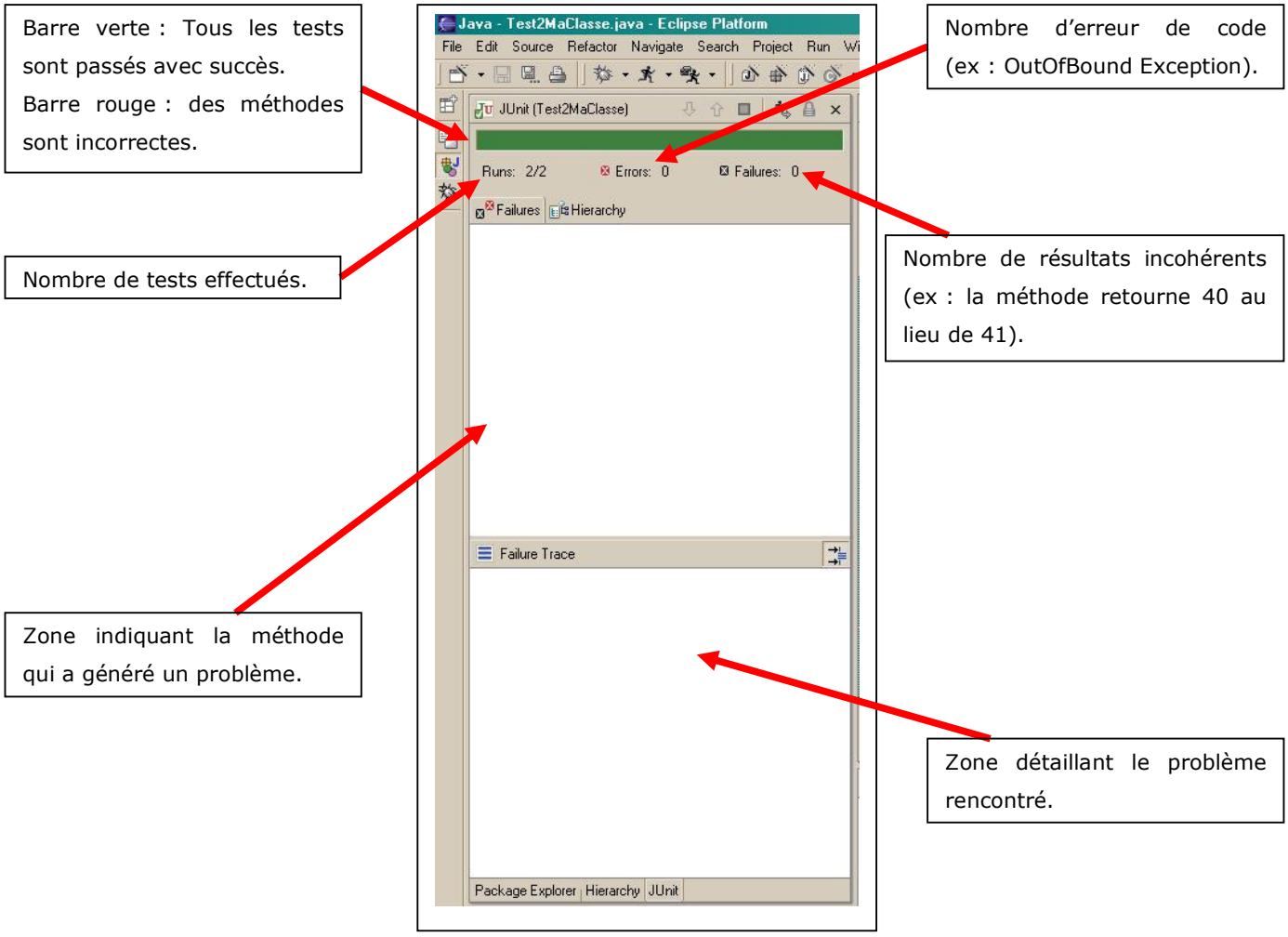
Les deux méthodes de *MaClasse.java* sont testées dans cette classe. Pour cela deux méthodes *testCalc()* et *testCalc2()* sont créées selon la convention de nommage. La structure (*List*) qui est utilisée dans les deux méthodes est déclarée de manière statique en début de classe.

Il est aussi possible de placer des affichages (*System.out.println(String)*) dans les méthodes de test, ces affichages seront visibles dans la console d'Eclipse.

Enfin, la méthode *fail()* peut prendre un *String* en paramètre. Celui-ci sera affiché dans la zone JUnit (voir la partie suivante).

4 La zone d'affichage JUnit

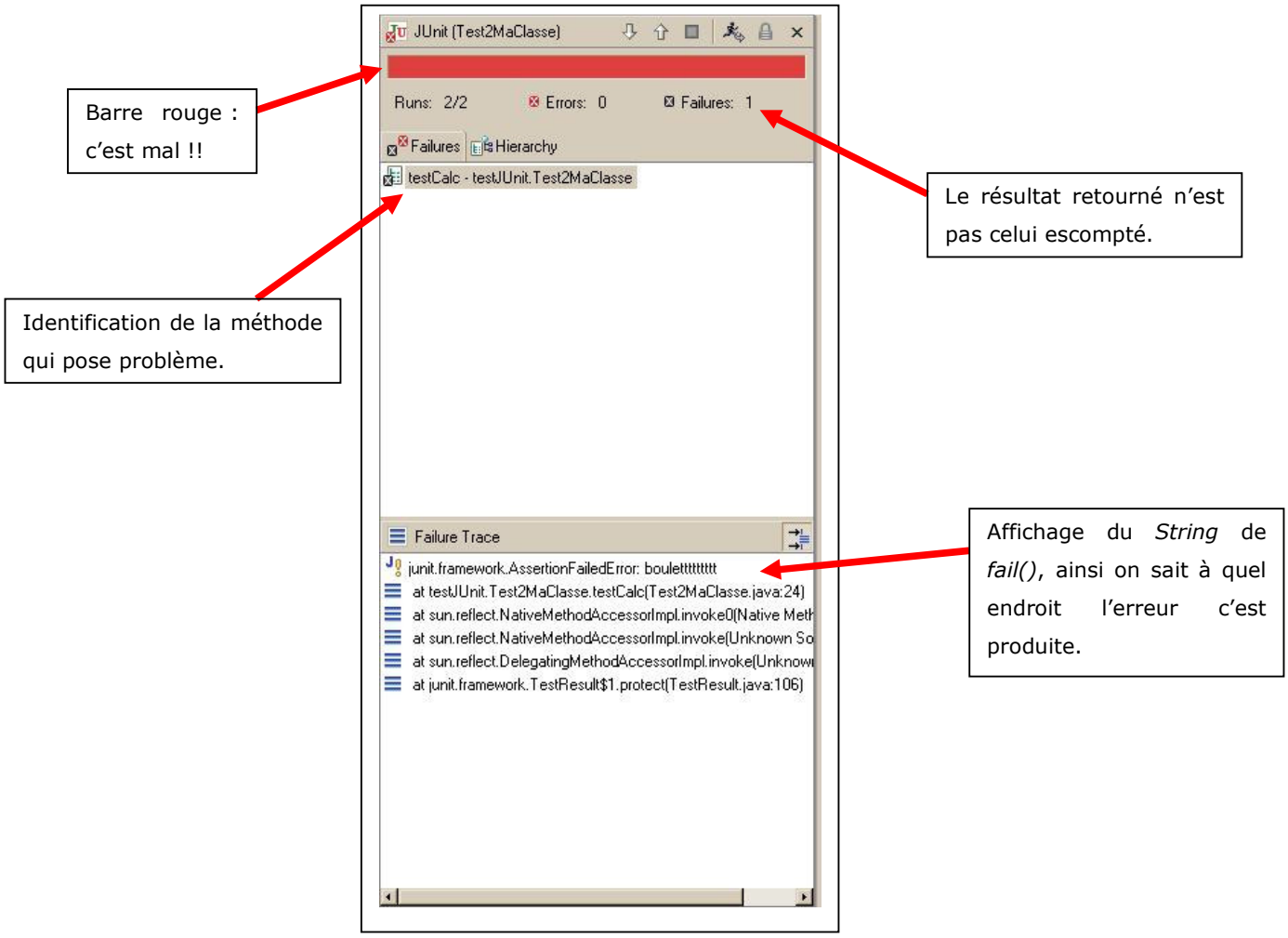
Une fois la classe de test exécutée, une zone d'affichage spécifique à JUnit s'ouvre. Voici à quoi elle ressemble :



Si par exemple on remplace *testCalc()* dans notre classe de test par la méthode suivante :

```
public void testCalc() {
        System.out.println("test de comment");
        if( MaClasse.calc(1) != 39  ) fail("boulettttttttt");
}
```

Cette méthode s'attend à avoir un résultat de 39 au lieu de 40. Voici l'affichage de JUnit après avoir exécuter la méthode de test :



De plus, le message *Test de comment* s'affiche dans la console d'Eclipse.

CONCLUSION

JUnit est uniquement utile pour tester les opérations standards (additions, etc…) des classes métiers.

L'utilisation de JUnit n'est pas compliqué, le plus difficile étant de correctement prédéfinir les tests.

Citations trouvées sur le net :

« L'une des premières questions qui se pose lorsque l'on met ces tests en pratique est : "Que doit-on tester ?"

*Selon le mantra XP, on teste "**Everything that could possibly break**". Chacun pourra trouver une interprétation personnelle de cette formule dans différents contextes, mais nous allons tenter ici de tracer quelques pistes.*

Tout d'abord, il est important de noter que l'on ne cherche pas systématiquement à tester unitairement chacune des méthodes de chacune des classes du système. Par exemple, on ne testera pas individuellement des accesseurs ou des méthodes d'une ou deux lignes qui consistent en de simples appels à des outils tiers éprouvés.

Ce que l'on cherche à tester avant tout, ce sont les services rendus par une classe. Ainsi, si certaines méthodes seront suffisamment complexes pour mériter des tests spécifiques, de nombreux tests auront pour but de mettre en oeuvre plusieurs méthodes d'une même classe pour vérifier leur cohérence d'ensemble. »

« le fait de s'attaquer à des petits problèmes les uns après les autres, et de poser le problème avant de partir sur une solution, permet de faciliter et de canaliser la réflexion - et ainsi de gagner du temps »

« Toutes ces raisons font des tests unitaires l'une des pratiques principales d'Extreme Programming.» ouahhh on fait de l'Extreme Programming ☺ !!!