

Tests unitaires automatisés avec JUnit4




par Régis **POUILLER** ([Home](#))

Date de publication : 24 avril 2009

Dernière mise à jour : 24 mai 2009

Cet article a pour objectif de présenter ce qu'apporte JUnit4 pour les tests automatisés (annotations, nouvelles assertions, suppositions, tests paramétrés).

I - INTRODUCTION


 *Pour une compréhension aisée de cet article, il est conseillé d'avoir quelques connaissances en Java, JUnit (de version précédente) et Eclipse.*

JUnit est un framework de tests unitaires pour Java. Ce framework appartient à l'ensemble des frameworks de tests **xUnit**. Ce framework a été écrit par **Kent Beck** (en autre créateur de l'**Extreme Programming** et du framework de tests **SUnit** pour Smalltalk, qui a inspiré tous les autres xUnit) et **Erich Gamma** (entre autre un des quatre auteurs de **Design Patterns: Elements of Reusable Object-Oriented Software** et concepteur de Java Development Tools, la partie d'**Eclipse** supportant Java).

Depuis Java 5, d'importantes évolutions ont été apportées à JUnit. Les dernières versions de JUnit ont donc changé de numéro majeur de version pour passer de JUnit 3.x à JUnit 4.x.

Cet article va présenter les particularités que JUnit4 (et plus particulièrement la version 4.5) apporte par rapport à JUnit3 :

- les annotations pour les tests
- les nouvelles assertions
- les suppositions
- les tests paramétrés
- les annotations pour les suites de tests

 *Lors de mes recherches, j'ai remarqué que parfois les imports statiques étaient considérés comme une évolution de JUnit4. Il s'agit cependant d'une évolution entièrement liée à Java5. J'ai tendance à ne pas les utiliser dans l'article, non par conviction anti-"imports statiques" ;-), mais parce que j'ai pensé que ça serait plus lisible dans l'article. Il y a cependant quelques exemples avec des imports statiques.*

Pour la réalisation de cet article les versions des outils sont :

- **Java Runtime Environment : 5.0 Update 17**
- **Eclipse : developpement Java 3.4 Ganymede SR2**
- **JUnit : 4.5**
- **JMock : 2.5.1**

II - AUTRES RESSOURCES SUR DEVELOPPEZ.COM

FAQ Eclipse (Comment utiliser JUnit avec Eclipse ?)

FAQ NetBeans (Comment utiliser JUnit avec NetBeans ?)

Tests unitaires par Sébastien MERIC

Conception de tests unitaires avec JUnit par Romain Guy

Test unitaire avec Spring par Johnny Beuve

Article complet: JUnit Anti-patterns sur ady's blog

Chapitre 11 (JUnit et Eclipse) de Développons en Java avec Eclipse par Jean-Michel DOUDOUX

Chapitre 67 (Les frameworks de tests) de Développons en Java avec Eclipse par Jean-Michel DOUDOUX

III - LES ANNOTATIONS POUR LES TESTS

Les annotations sont en Java une nouveauté de Java 5. Les annotations de JUnit apportent des fonctionnalités en plus (initialisation globale à l'ensemble des méthodes de tests d'un cas de test) et de la souplesse. Il n'est plus nécessaire que le cas de tests soit une classe héritant de **TestCase**. Les annotations indiquent le comportement du cas de tests. Toutes les annotations, ci-après, sont apparues à partir de la version 4.0 de JUnit.

III-A - Test simple

La javadoc de cette annotation est disponible [ici](#).

Pour déterminer les méthodes de tests, JUnit3 recherche toutes celles commençant par "test" dans les classes héritant de **TestCase**. JUnit4, quant à lui, recherche les méthodes avec l'annotation **@Test** dans n'importe quelle classe.

Operations.java

```
package com.developpez.rpouiller.testsjunit4;

public class Operations {

    public static long additionner(final long...pNombres) {
        long lRetour = 0;
        for(final long lNombre : pNombres) {
            lRetour += lNombre;
        }
        return lRetour;
    }

    // Cette méthode ne fonctionne pas correctement
    // Les tests vont le vérifier
    public static long multiplier(final long...pNombres) {
        long lRetour = 0;
        for(final long lNombre : pNombres) {
            lRetour *= lNombre;
        }
        return lRetour;
    }
}
```

TestSimple.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.Assert;
import org.junit.Test;

public class TestSimple {

    @Test
    public void additionAvecDeuxNombres() {
        final long lAddition = Operations.additionner(10, 20);
        Assert.assertEquals(30, lAddition);
    }

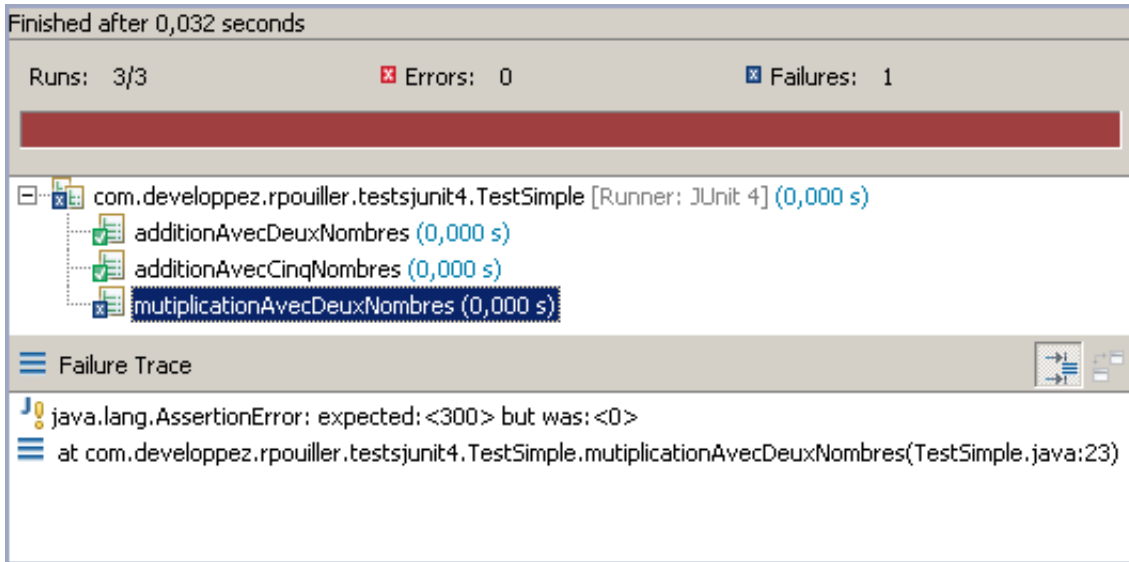
    @Test
    public void additionAvecCinqNombres() {
        final long lAddition = Operations.additionner(256, 512, 1024, 2048, 4096);
        Assert.assertEquals(7936, lAddition);
    }

    @Test
    public void multiplicationAvecDeuxNombres() {
        final long lMultiplication = Operations.multiplier(10, 20);
        Assert.assertEquals(300, lMultiplication);
    }
}
```

TestSimple.java

```
public void testMutiplicationAvecCinqNombres() {
    final long lMultiplication = Operations.additionner(256, 512, 1024, 2048, 4096);
    Assert.assertEquals(1125899906842624L, lMultiplication);
}
```

Comme le montre la capture d'écran ci-dessous, seules les méthodes de tests avec l'annotation ont été exécutées.



III-B - Test devant déclencher un Throwable

La javadoc de cette annotation est disponible [ici](#).

Cette annotation permet de vérifier qu'un Throwable a bien été déclenché. L'annotation reçoit la classe du Throwable attendu en paramètre.

Operations.java

```
package com.developpez.rpouiller.testsjunit4;

public class Operations {

    public static long diviser(final long...pNombres) {
        if(pNombres.length < 2) {
            throw new IllegalArgumentException(
                "Il faut au moins deux nombres en entrée");
        }
        long lRetour = pNombres[0];
        for(int i=1; i<pNombres.length; i++) {
            lRetour /= pNombres[i];
        }
        return lRetour;
    }
}
```

TestThrowable.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.Test;

public class TestThrowable {

    @Test(expected=IllegalArgumentException.class)
```

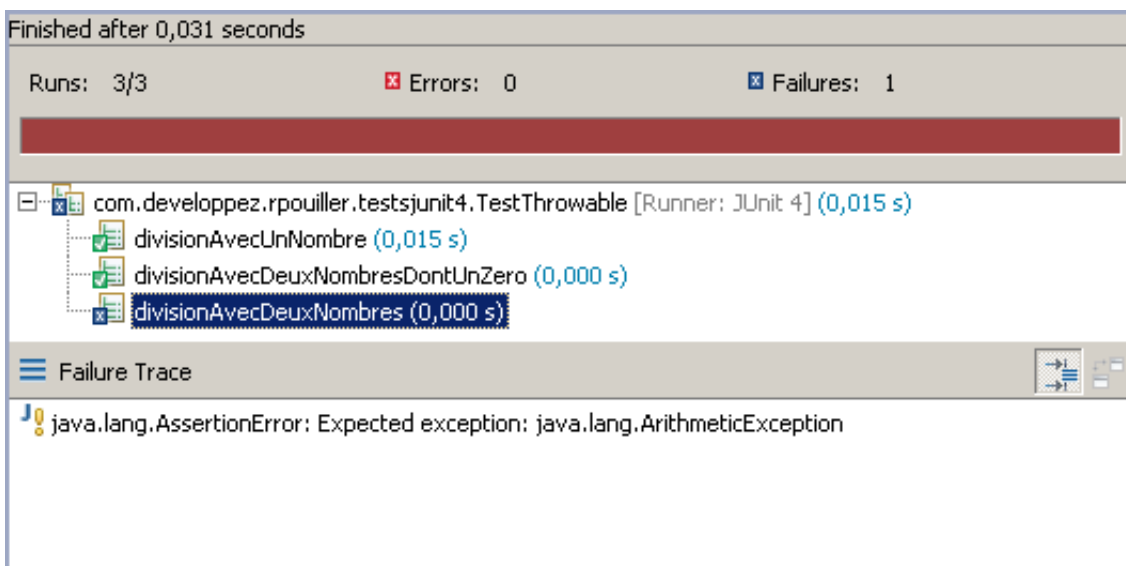
TestThrowable.java

```
public void divisionAvecUnNombre() {
    Operations.diviser(1);
}

@Test(expected=ArithmeticException.class)
public void divisionAvecDeuxNombresDontUnZero() {
    Operations.diviser(10, 0);
}

@Test(expected=ArithmeticException.class)
public void divisionAvecDeuxNombres() {
    Operations.diviser(10, 5);
}
}
```

Comme le montre la capture d'écran ci-dessous, les deux premières méthodes de test provoquent des Exceptions (la première à cause du contrôle en début de la méthode diviser, la deuxième à cause d'une division par zéro). La dernière ne déclenche pas d'erreur, le test est donc considéré comme échoué.



III-C - Test d'une durée limitée

La javadoc de cette annotation est disponible [ici](#).

Cette annotation permet de vérifier qu'un test ne dépasse pas une durée. Au delà de cette durée, le test passe en erreur. La durée en millisecondes est passée en paramètre à l'annotation.

TestDureeLimitee.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.Test;

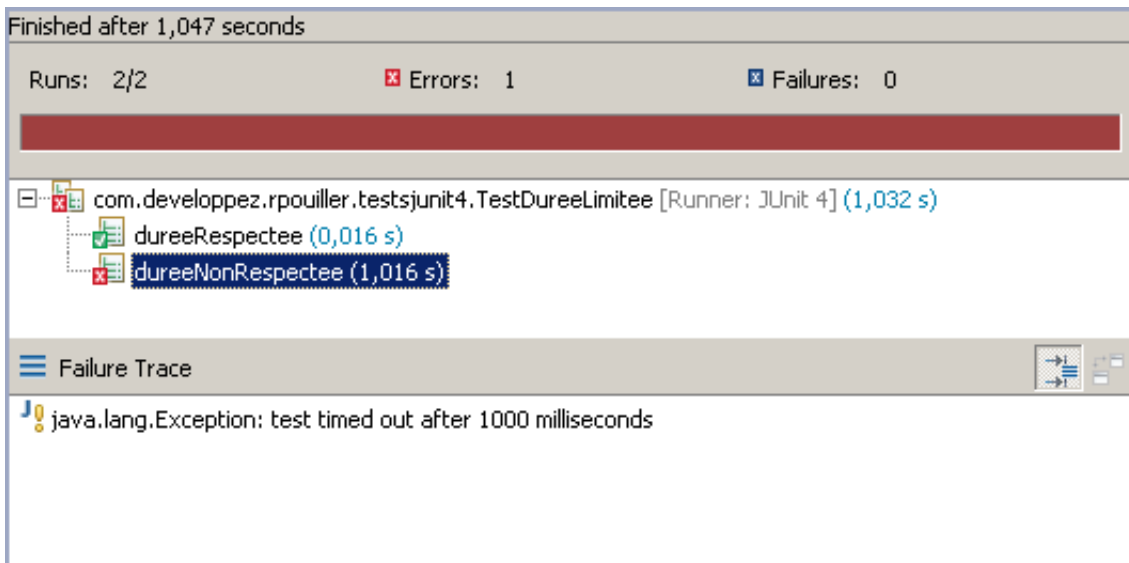
public class TestDureeLimitee {

    @Test(timeout=1000)
    public void dureeRespectee() {

    }

    @Test(timeout=1000)
    public void dureeNonRespectee() throws InterruptedException {
        Thread.sleep(10000);
    }
}
```

Comme le montre la capture d'écran ci-dessous, lorsque la durée d'un test dépasse la durée indiquée, le test passe en erreur.



III-D - Indisponibilité d'un test

La javadoc de cette annotation est disponible [ici](#).

Cette annotation permet de pas passer un test.

```
TestIndisponibilite.java
package com.developpez.rpouiller.testsjunit4;

import org.junit.Assert;
import org.junit.Ignore;
import org.junit.Test;

public class TestIndisponibilite {

    @Test
    public void nonIgnore1() {

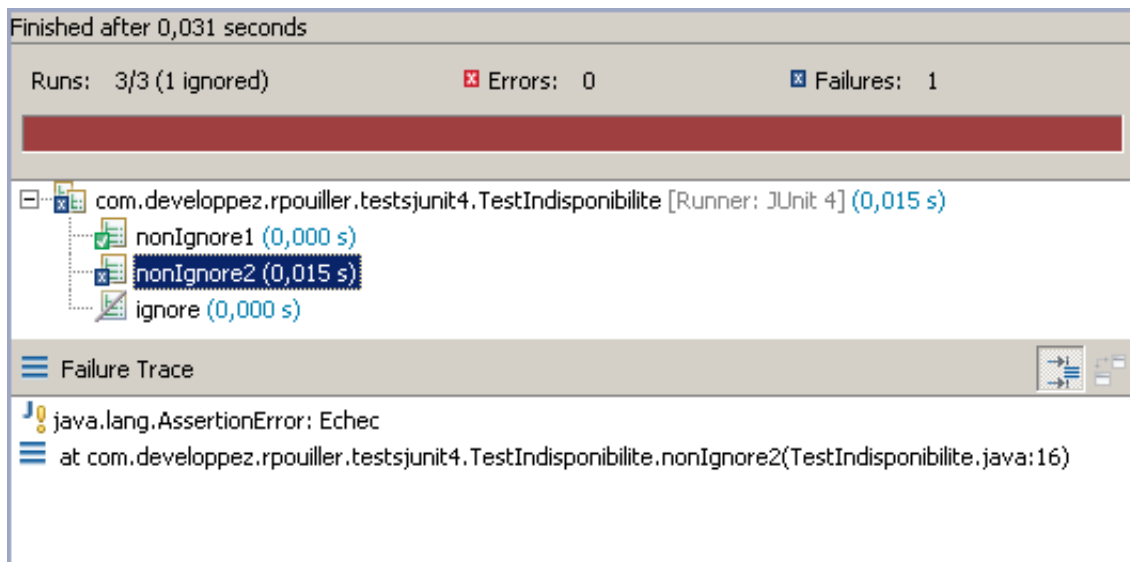
    }

    @Test
    public void nonIgnore2() {
        Assert.fail("Echec");
    }

    @Ignore
    @Test
    public void ignore() {
        Assert.fail("Echec ignoré");
    }

}
```

Comme le montre la capture d'écran ci-dessous, le premier test est passé avec succès, le deuxième a échoué et le troisième a été ignoré.



III-E - Préparation avant et démontage après chaque test d'un cas de test

La javadoc de ces annotations est disponible [ici](#) et [ici](#).

Ces annotations permettent d'indiquer une méthode qui sera exécutée avant chaque test et une méthode qui sera exécutée après chaque test.

TestAvantApres.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestAvantApres {

    @Before
    public void avantTest() {
        System.out.println("-----");
        System.out.println("Avant Test");
    }

    @After
    public void apresTest() {
        System.out.println("Après Test");
        System.out.println("-----");
    }

    @Test
    public void premierTest() {
        System.out.println("Premier Test");
    }

    @Test
    public void deuxiemeTest() {
        System.out.println("Deuxième Test");
    }

    @Test
    public void troisiemeTest() {
        System.out.println("Troisième Test");
    }
}
```

Comme le montre la trace ci-dessous, les méthodes **avantTest** et **apresTest** sont exécutées avant et après les méthodes de test.

```

-----
Avant Test
Premier Test
Après Test
-----

-----
Avant Test
Deuxième Test
Après Test
-----

-----
Avant Test
Troisième Test
Après Test
-----

```

III-F - Préparation avant et démontage après l'ensemble de tests d'un cas de tests

La javadoc de ces annotations est disponible à l'url : [ici](#) et [ici](#).

Ces annotations permettent d'indiquer une méthode qui sera exécutée avant l'ensemble des tests d'un cas de tests et une méthode qui sera exécutée après l'ensemble des tests d'un cas de tests.

TestAvantApresEnsemble.java

```

package com.developpez.rpouiller.testsjunit4;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestAvantApresEnsemble {

    @BeforeClass
    public static void avantTests() {
        System.out.println("-----");
        System.out.println("Avant Tests");
        System.out.println("-----");
    }

    @AfterClass
    public static void apresTests() {
        System.out.println("-----");
        System.out.println("Après Tests");
        System.out.println("-----");
    }

    @Test
    public void premierTest() {
        System.out.println("Premier Test");
    }

    @Test
    public void deuxiemeTest() {
        System.out.println("Deuxième Test");
    }

    @Test
    public void troisiemeTest() {
        System.out.println("Troisième Test");
    }
}

```


Comme le montre la trace ci-dessous, les méthodes **avantTests** et **apresTests** sont exécutées avant et après l'ensemble des méthodes de test.

```
-----  
Avant Tests  
-----  
Premier Test  
Deuxième Test  
Troisième Test  
-----  
Après Tests  
-----
```

IV - LES NOUVELLES ASSERTIONS

IV-A - Assertions d'égalité pour les tableaux

Ces assertions comprennent des assertions d'égalité pour les tableaux de :

- **"byte"** dont la javadoc est disponible [ici](#) et [ici](#).
- **"char"** dont la javadoc est disponible [ici](#) et [ici](#).
- **"short"** dont la javadoc est disponible [ici](#) et [ici](#).
- **"int"** dont la javadoc est disponible [ici](#) et [ici](#).
- **"long"** dont la javadoc est disponible [ici](#) et [ici](#).
- **"Object"** dont la javadoc est disponible [ici](#) et [ici](#).

Ces assertions sont présentes dans JUnit depuis la version 4.3.1.

Operations.java

```
package com.developpez.rpouiller.testsjunit4;  
  
import java.util.Arrays;  
  
public class Operations {  
  
    // Cette méthode vérifie que les longueurs passées en paramètre  
    // sont celles d'un triangle rectangle  
    public static Boolean[] pythagore(final long[]...pLongueurs) {  
        final Boolean[] lRetours = new Boolean[pLongueurs.length];  
  
        for(int i=0;i<pLongueurs.length;i++) {  
            final long[] lLongueurs = pLongueurs[i];  
            if(lLongueurs.length != 3) {  
                throw new IllegalArgumentException(  
                    "Les blocs de longueurs doivent être de 3 éléments");  
            }  
  
            final long[] lCopieLongueurs = lLongueurs.clone();  
            Arrays.sort(lCopieLongueurs);  
  
            final long lLongueur1 = lCopieLongueurs[0] * lCopieLongueurs[0];  
            final long lLongueur2 = lCopieLongueurs[1] * lCopieLongueurs[1];  
            final long lLongueur3 = lCopieLongueurs[2] * lCopieLongueurs[2];  
  
            if(lLongueur1 + lLongueur2 == lLongueur3) {  
                lRetours[i] = true;  
            }  
            else {  
                lRetours[i] = false;  
            }  
        }  
  
        return lRetours;  
    }  
}
```

Operations.java

```
}
```

TestTableau.java

```
package com.developpez.rpouiller.testsjunit4;

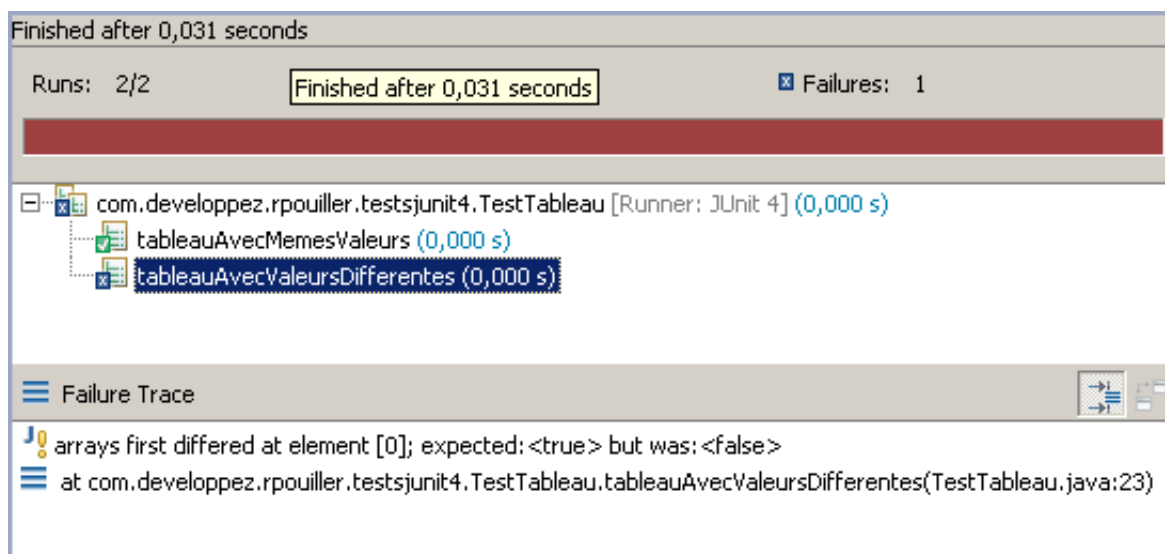
import org.junit.Assert;
import org.junit.Test;

public class TestTableau {

    @Test
    public void tableauAvecMemesValeurs() {
        final Boolean[] lResutat = Operations.pythagore(
            new long[]{3, 4, 5},
            new long[]{6, 10, 8}
        );
        Assert.assertArrayEquals(new Boolean[]{true, true}, lResutat);
    }

    @Test
    public void tableauAvecValeursDifferentes() {
        final Boolean[] lResutat = Operations.pythagore(
            new long[]{3, 4, 6},
            new long[]{6, 11, 8}
        );
        Assert.assertArrayEquals(new Boolean[]{true, true}, lResutat);
    }
}
```

Comme le montre la capture d'écran ci-dessous, l'assertion vérifie que les éléments des tableaux sont égaux.



IV-B - Assertion d'égalité pour les "double" avec un delta maximal entre les deux valeurs à comparer

La javadoc de cette assertion est disponible [ici](#) et [ici](#).

Cette assertion est présente dans JUnit depuis la version 4.4. L'assertion d'égalité pour les doubles sans delta est passé en déprécié à partir de la version 4.5 de JUnit.

Operations.java

```
package com.developpez.rpouiller.testsjunit4;

public class Operations {
```

Operations.java

```
public static double soustraire(final double...pNombres) {
    if(pNombres.length < 2) {
        throw new IllegalArgumentException(
            "Il faut au moins deux nombres en entrée");
    }
    double lRetour = pNombres[0];
    for(int i=1;i<pNombres.length;i++) {
        lRetour -= pNombres[i];
    }
    return lRetour;
}
```

TestDoubleAvecDelta.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.Assert;
import org.junit.Test;

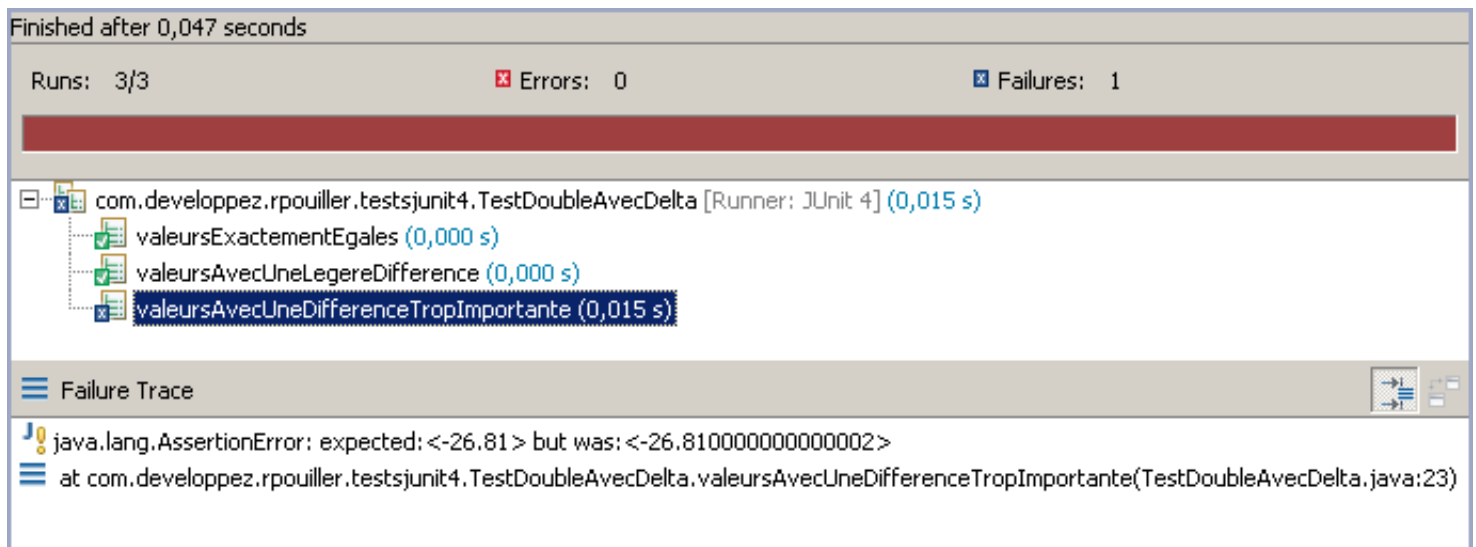
public class TestDoubleAvecDelta {

    @Test
    public void valeursExactementEgales() {
        final double lResultat = Operations.soustraire(5, 2.5);
        Assert.assertEquals(2.5, lResultat, 0);
    }

    @Test
    public void valeursAvecUneLegereDifference() {
        final double lResultat = Operations.soustraire(71.19, 98);
        Assert.assertEquals(-26.81, lResultat, 0.005);
    }

    @Test
    public void valeursAvecUneDifferenceTropImportante() {
        final double lResultat = Operations.soustraire(71.19, 98);
        Assert.assertEquals(-26.81, lResultat, 0);
    }
}
```

Comme le montre la capture d'écran ci-dessous, le paramètre delta permet de considérer que l'assertion est juste même s'il y a une différence entre les valeurs, tant que cette différence est inférieure au delta.



Finished after 0,047 seconds

Runs: 3/3 ✖ Errors: 0 ✖ Failures: 1

com.developpez.rpouiller.testsjunit4.TestDoubleAvecDelta [Runner: JUnit 4] (0,015 s)

- ✓ valeursExactementEgales (0,000 s)
- ✓ valeursAvecUneLegereDifference (0,000 s)
- ✖ valeursAvecUneDifferenceTropImportante (0,015 s)

Failure Trace

java.lang.AssertionError: expected: <-26.81> but was: <-26.810000000000002>
 at com.developpez.rpouiller.testsjunit4.TestDoubleAvecDelta.valeursAvecUneDifferenceTropImportante(TestDoubleAvecDelta.java:23)

IV-C - Assertion sur une condition définie par contrat

La javadoc de cette assertion est disponible [ici](#) et [ici](#).

Cette assertion est présente dans JUnit depuis la version 4.4.

Pour que l'assertion soit juste, il faut que le contrat soit vérifié.

IV-C-1 - Contrat "pareil" : IsSame

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que l'instance passée en premier paramètre de l'assertion et celle passée en paramètre du contrat sont les mêmes.

TestContratPareil.java

```
package com.developpez.rpouiller.testsjunit4;

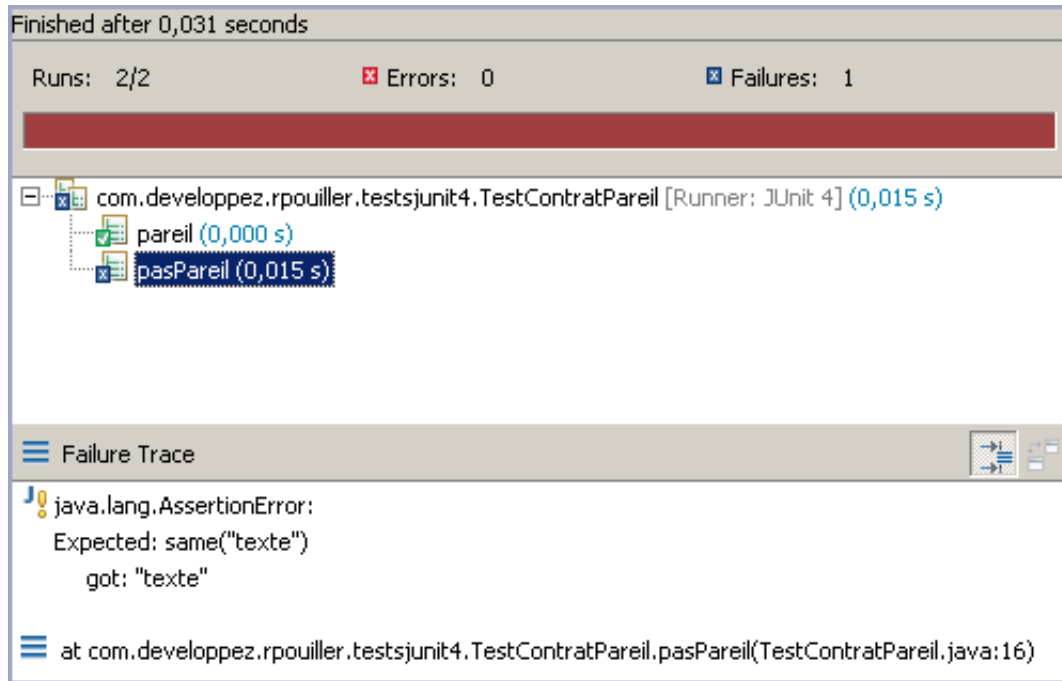
import org.hamcrest.core.IsSame;
import org.junit.Assert;
import org.junit.Test;

public class TestContratPareil {

    @Test
    public void pareil() {
        // Remarque : c'est la même instance à cause du cache
        Assert.assertThat("texte", IsSame.sameInstance("texte"));
    }

    @Test
    public void pasPareil() {
        // Remarque : ce n'est pas la même instance parce que new String() force une nouvelle instance.
        Assert.assertThat(new String("texte"), IsSame.sameInstance("texte"));
    }
}
```

Comme le montre la capture d'écran ci-dessous, l'assertion échoue si les instances ne sont pas les mêmes.



IV-C-2 - Contrat "égalité" : IsEqual

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que l'instance passée en premier paramètre de l'assertion et celle passée en paramètre du contrat sont égales.

```

TestContratEgal.java
package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.IsEqual;
import org.junit.Assert;
import org.junit.Test;

public class TestContratEgal {

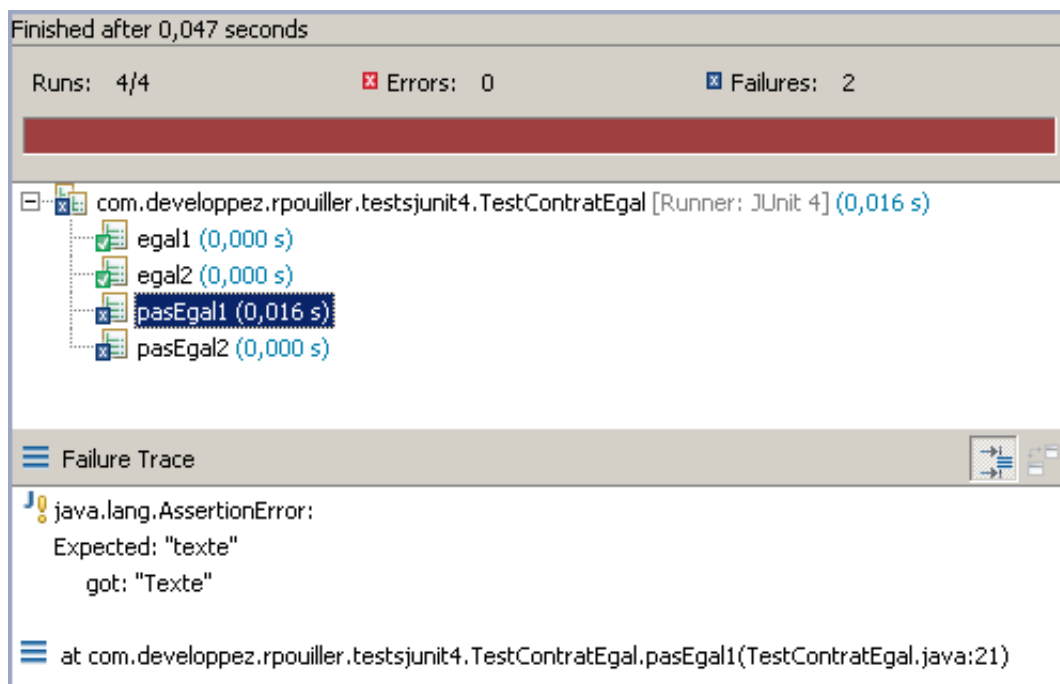
    @Test
    public void egal1() {
        Assert.assertThat("texte", IsEqual.equalTo("texte"));
    }

    @Test
    public void egal2() {
        Assert.assertThat(new String("texte"), IsEqual.equalTo("texte"));
    }

    @Test
    public void pasEgal1() {
        Assert.assertThat("Texte", IsEqual.equalTo("texte"));
    }

    @Test
    public void pasEgal2() {
        Assert.assertThat(new String("Texte"), IsEqual.equalTo("texte"));
    }
}
  
```

Comme le montre la capture d'écran ci-dessous, l'assertion échoue si les instances ne sont pas égales.



IV-C-3 - Contrat "classe de l'instance" : IsInstanceOf

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que l'instance passée en premier paramètre de l'assertion est de la classe passée en paramètre du contrat.

```

TestContratClasseInstance.java
package com.developpez.rpouiller.testsjunit4;

import java.io.Serializable;

import org.hamcrest.core.IsInstanceOf;
import org.junit.Assert;
import org.junit.Test;

public class TestContratClasseInstance {

    @Test
    public void classeDeString() {
        Assert.assertThat("texte", IsInstanceOf.instanceOf(String.class));
    }

    @Test
    public void classeDeSerializable() {
        Assert.assertThat("texte", IsInstanceOf.instanceOf(Serializable.class));
    }

    @Test
    public void classeDeCharSequence() {
        Assert.assertThat("texte", IsInstanceOf.instanceOf(CharSequence.class));
    }

    @Test
    public void classeDeInteger() {
        Assert.assertThat("texte", IsInstanceOf.instanceOf(Integer.class));
    }
}
  
```

Comme le montre la capture d'écran ci-dessous, l'assertion échoue si l'instance n'est pas de la bonne classe.

Finished after 0,031 seconds

Runs: 4/4 ✖ Errors: 0 ✖ Failures: 1

com.developpez.rpouiller.testsjunit4.TestContratClasseInstance [Runner: JUnit 4] (0,015 s)

- ✓ classeDeString (0,000 s)
- ✓ classeDeSerializable (0,000 s)
- ✓ classeDeCharSequence (0,000 s)
- ✖ classeDeInteger (0,015 s)

Failure Trace

java.lang.AssertionError:
 Expected: an instance of java.lang.Integer
 got: "texte"

at com.developpez.rpouiller.testsjunit4.TestContratClasseInstance.classeDeInteger(TestContratClasseInstance.java:28)

IV-C-4 - Contrat "valeur null" : IsNull

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que la référence passée en premier paramètre de l'assertion est null ou n'est pas null.

```
TestContratValeurNull.java
package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.IsNull;
import org.junit.Assert;
import org.junit.Test;

public class TestContratValeurNull {

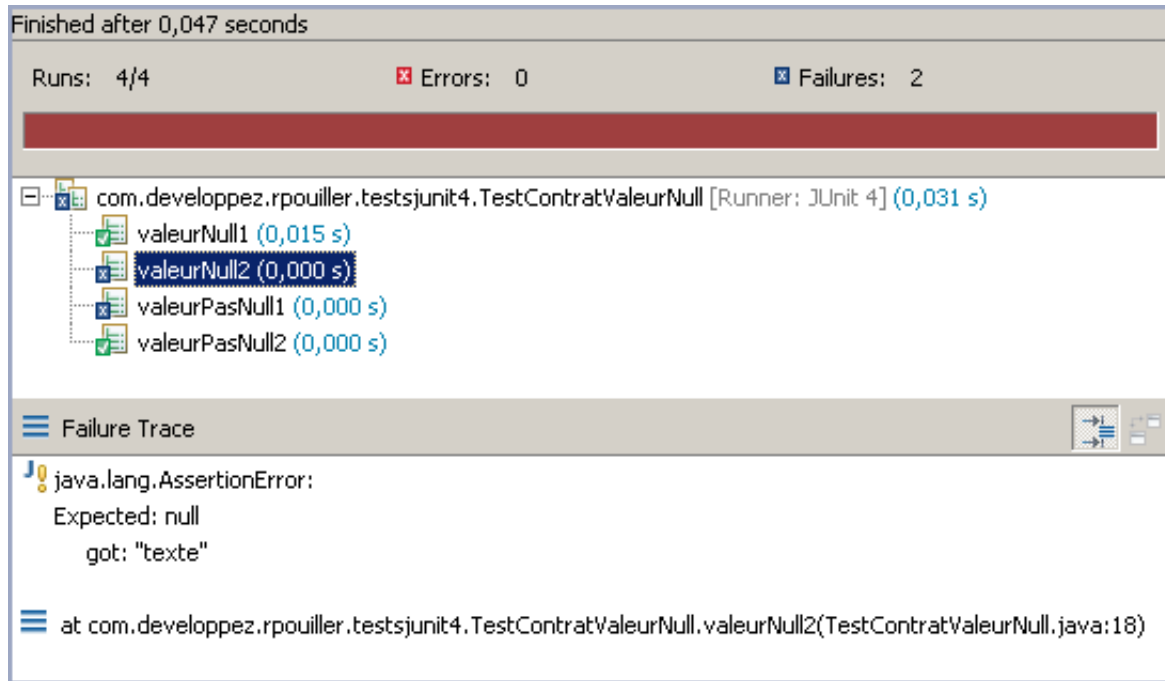
    @Test
    public void valeurNull1() {
        final String lTexte = null;
        Assert.assertThat(lTexte, IsNull.nullValue());
    }

    @Test
    public void valeurNull2() {
        final String lTexte = "texte";
        Assert.assertThat(lTexte, IsNull.nullValue());
    }

    @Test
    public void valeurPasNull1() {
        final String lTexte = null;
        Assert.assertThat(lTexte, IsNull.notNullValue());
    }

    @Test
    public void valeurPasNull2() {
        final String lTexte = "texte";
        Assert.assertThat(lTexte, IsNull.notNullValue());
    }
}
```

Comme le montre la capture d'écran ci-dessous, l'assertion échoue si la référence ne correspond pas au contrat.



IV-C-5 - Contrat "être" : Is

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que l'instance passée en premier paramètre correspond à ce qui est passé (classe, valeur, autre contrat) en paramètre du contrat. Ce contrat n'apporte pas de nouveau type de contrat :

- si on passe une classe, cela correspond à un contrat "classe de l'instance"
- si on passe une valeur, cela correspond à un contrat "égalité"
- si on passe un autre contrat, cela correspond à cet autre contrat

TestContratEtre.java

```

package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.Is;
import org.hamcrest.core.IsSame;
import org.junit.Assert;
import org.junit.Test;

public class TestContratEtre {

    @Test
    public void classeDeString() {
        Assert.assertThat("texte", Is.is(String.class));
    }

    @Test
    public void classeDeInteger() {
        Assert.assertThat("texte", Is.is(Integer.class));
    }

    @Test
    public void egal() {
        Assert.assertThat("texte", Is.is("texte"));
    }

    @Test
    public void pasEgal() {
        Assert.assertThat("Texte", Is.is("texte"));
    }
}

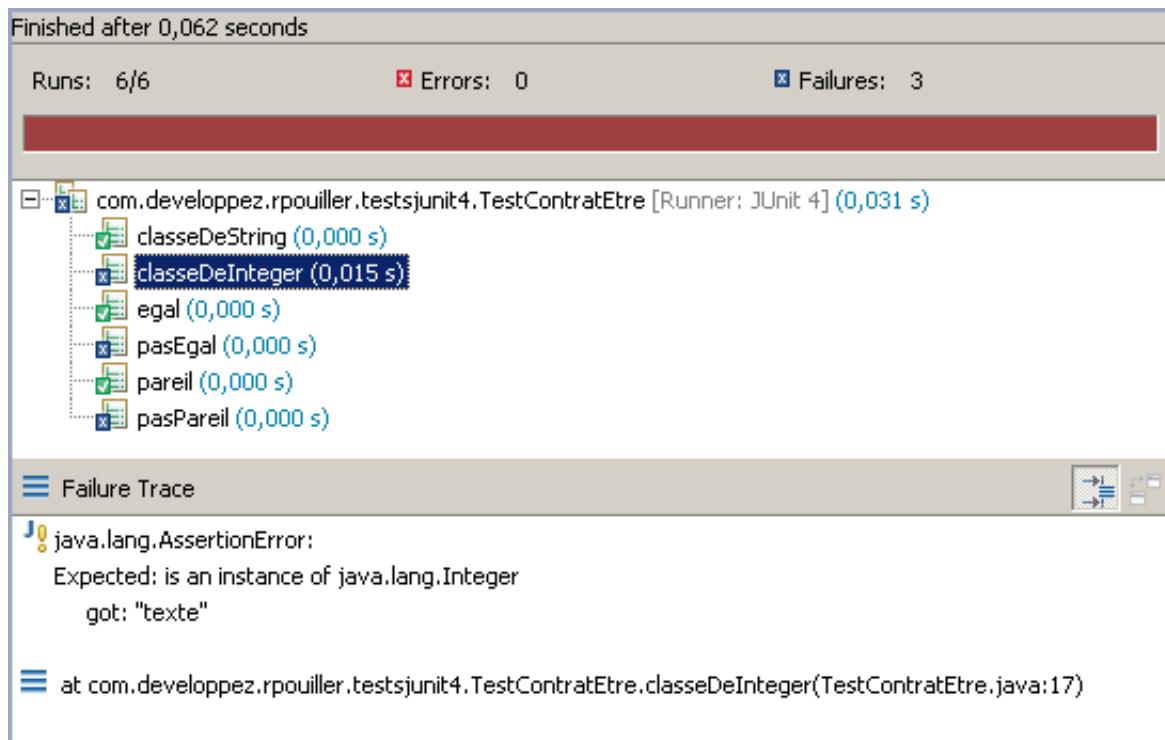
```


TestContratEtre.java

```
@Test
public void pareil() {
    Assert.assertThat("texte", Is.is(IsSame.sameInstance("texte")));
}

@Test
public void pasPareil() {
    Assert.assertThat(new String("texte"), Is.is(IsSame.sameInstance("texte")));
}
}
```

Comme le montre la capture d'écran ci-dessous, l'assertion réussit si l'instance est de la classe passée en paramètre du contrat, si elle est égale à la valeur passée en paramètre du contrat ou si le contrat passé en paramètre du contrat "être" est respecté.



IV-C-6 - Contrat "ne pas être" : IsNot

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que l'instance passée en premier paramètre ne correspond pas à ce qui est passé (valeur ou autre contrat) en paramètre du contrat. Ce contrat n'apporte pas de nouveau type de contrat :

- si on passe une valeur, cela correspond à un contrat "égalité"
- si on passe un autre contrat, cela correspond à cet autre contrat

TestContratNePasEtre.java

```
package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.IsNot;
import org.hamcrest.core.IsSame;
import org.junit.Assert;
import org.junit.Test;

public class TestContratNePasEtre {
```

TestContratNePasEtre.java

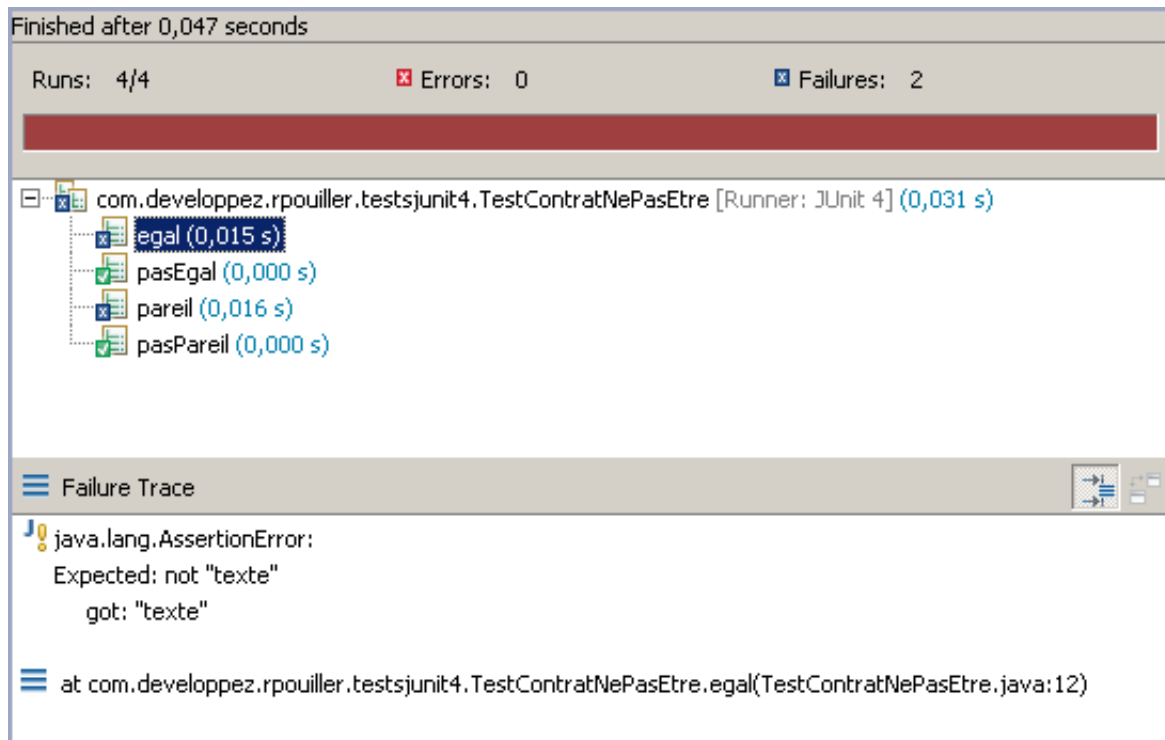
```
@Test
public void egal() {
    Assert.assertThat("texte", IsNot.not("texte"));
}

@Test
public void pasEgal() {
    Assert.assertThat("Texte", IsNot.not("texte"));
}

@Test
public void pareil() {
    Assert.assertThat("texte", IsNot.not(IsSame.sameInstance("texte")));
}

@Test
public void pasPareil() {
    Assert.assertThat(new String("texte"), IsNot.not(IsSame.sameInstance("texte")));
}
}
```

Comme le montre la capture d'écran ci-dessous, l'assertion réussit si l'instance n'est pas égale à la valeur passée en paramètre du contrat ou si le contrat passé en paramètre du contrat "ne pas être" n'est pas respecté.



IV-C-7 - Contrat "tous" : AllOf

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie que tous les contrats passés en paramètre du contrat sont respectés.

TestContratTous.java

```
package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.AllOf;
import org.hamcrest.core.IsEqual;
import org.hamcrest.core.IsInstanceOf;
import org.hamcrest.core.IsNot;
```

TestContratTous.java

```
import org.junit.Assert;
import org.junit.Test;

public class TestContratTous {

    @Test
    public void classeDeStringEtEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsInstanceOf.instanceOf(String.class), IsEqual.equalTo("texte")
        ));
    }

    @Test
    public void classeDeIntegerEtEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsInstanceOf.instanceOf(Integer.class), IsEqual.equalTo("texte")
        ));
    }

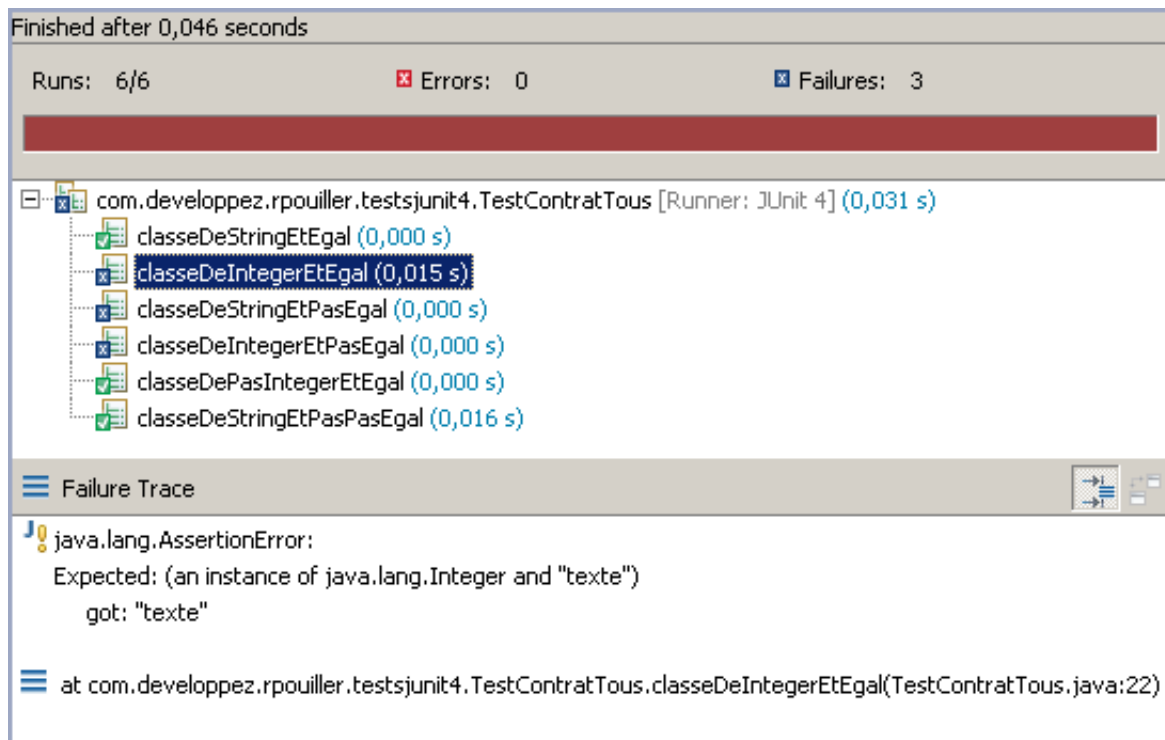
    @Test
    public void classeDeStringEtPasEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsInstanceOf.instanceOf(String.class), IsEqual.equalTo("Texte")
        ));
    }

    @Test
    public void classeDeIntegerEtPasEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsInstanceOf.instanceOf(Integer.class), IsEqual.equalTo("Texte")
        ));
    }

    @Test
    public void classeDePasIntegerEtEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsNot.not(IsInstanceOf.instanceOf(Integer.class)), IsEqual.equalTo("texte")
        ));
    }

    @Test
    public void classeDeStringEtPasPasEgal() {
        Assert.assertThat("texte", AllOf.allOf(
            IsInstanceOf.instanceOf(String.class), IsNot.not(IsEqual.equalTo("Texte"))
        ));
    }
}
```

Comme le montre la capture d'écran ci-dessous, les assertions où les deux sous-contrats sont respectés réussissent.



IV-C-8 - Contrat "au moins un" : AnyOf

La javadoc de ce contrat est disponible [ici](#).

Le contrat vérifie qu'au moins un des contrats passés en paramètre du contrat est respecté.

TestContratAuMoinsUn.java

```
package com.developpez.rpouiller.testsjunit4;

import org.hamcrest.core.AnyOf;
import org.hamcrest.core.IsEqual;
import org.hamcrest.core.IsInstanceOf;
import org.junit.Assert;
import org.junit.Test;

public class TestContratAuMoinsUn {

    @Test
    public void classeDeStringEtEgal() {
        Assert.assertThat("texte", AnyOf.anyOf(
            IsInstanceOf.instanceOf(String.class), IsEqual.equalTo("texte")
        ));
    }

    @Test
    public void classeDeIntegerEtEgal() {
        Assert.assertThat("texte", AnyOf.anyOf(
            IsInstanceOf.instanceOf(Integer.class), IsEqual.equalTo("texte")
        ));
    }

    @Test
    public void classeDeStringEtPasEgal() {
        Assert.assertThat("texte", AnyOf.anyOf(
            IsInstanceOf.instanceOf(String.class), IsEqual.equalTo("Texte")
        ));
    }

    @Test
    public void classeDeIntegerEtPasEgal() {
```

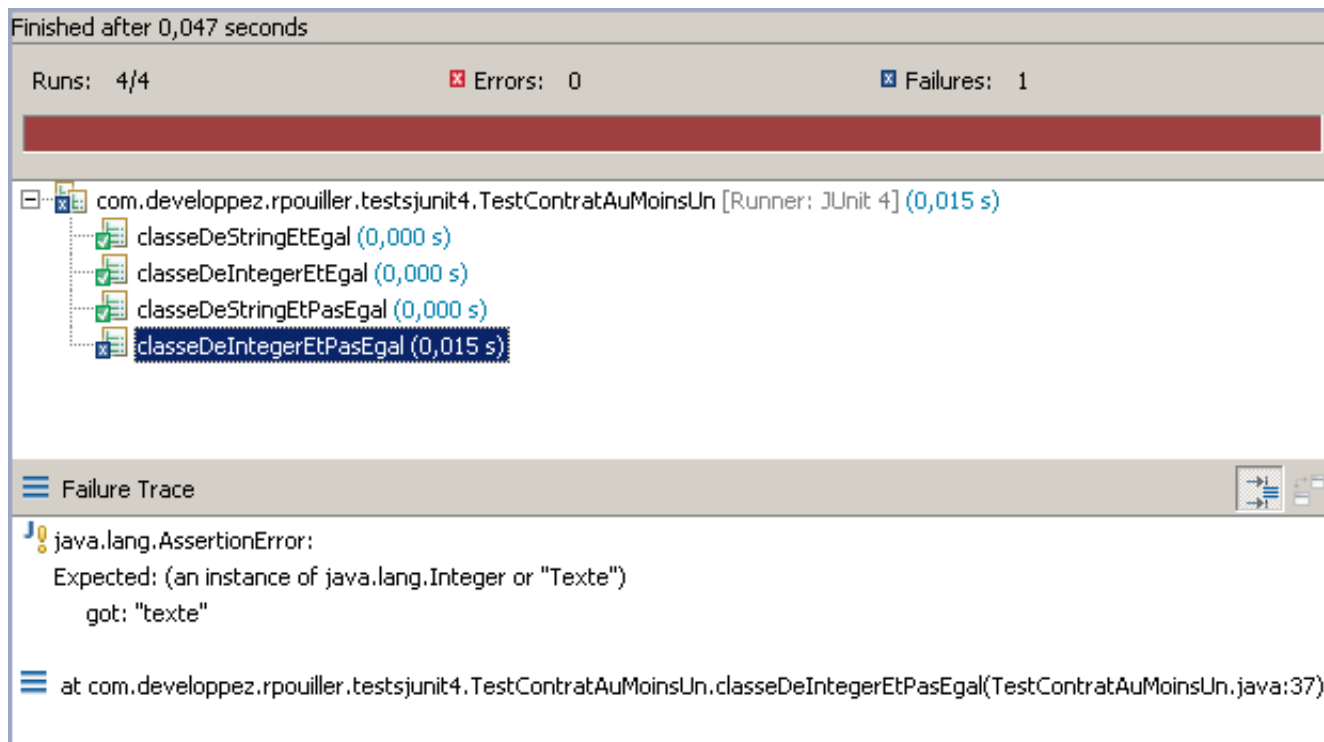
TestContratAuMoinsUn.java

```

    Assert.assertThat("texte", AnyOf.anyOf(
        IsInstanceOf.instanceOf(Integer.class), IsEqual.equalTo("Texte")
    ));
  }
}

```

Comme le montre la capture d'écran ci-dessous, les assertions où au moins un sous-contrat est respecté réussissent.



IV-C-9 - Pour aller plus loin : les contrats hors JUnit

Les assertions par contrat sont désormais intégrées à JUnit, mais sont à l'origine développées par **JMock**.

La librairie originale comprend des contrats supplémentaires. La javadoc de JMock est disponible [ici](#).

Ces contrats sont séparés en 7 groupes :

- core (contrats intégrés dans JUnit et déjà vus dans les chapitres précédents) : contrats de base sur les objets et les valeurs, ainsi que les contrats composés.
- beans : contrats sur les beans Java et leurs valeurs.
- collection : contrats sur les tableaux et les collections.
- number : contrats de comparaison numérique.
- object : contrats de contrôle des objets et des classes.
- text : contrats de comparaison de textes.
- xml : contrats sur les documents XML.

Pour pouvoir les utiliser, il faut rajouter les jar : **"jmock-2.5.1.jar"**, **"hamcrest-core-1.1.jar"** et **"hamcrest-library-1.1.jar"**.

IV-C-9-a - Les contrats beans

Le contrats "HasProperty" vérifie qu'un bean possède bien les getter et setter correspondants à une propriété. La javadoc de ce contrat est disponible [ici](#).

Le contrat "HasPropertyWithValue" vérifie que la valeur d'une propriété d'un bean respecte un contrat passé en paramètre. La javadoc de ce contrat est disponible [ici](#).

BeanSansPropriete.java

```
package com.developpez.rpouiller.testsjunit4;

public class BeanSansPropriete {

}
```

BeanAvecPropriete.java

```
package com.developpez.rpouiller.testsjunit4;

public class BeanAvecPropriete {
    private Integer prop;
    public void setProp(Integer pProp){prop = pProp;}
    public Integer getProp(){return prop;}
}
```

TestContratBean.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.equalTo;
import static org.hamcrest.Matchers.hasProperty;
import static org.junit.Assert.assertThat;

import org.junit.Test;

public class TestContratBean {

    @Test
    public void beanSansPropriete() {
        assertThat(new BeanSansPropriete(), hasProperty("prop"));
    }

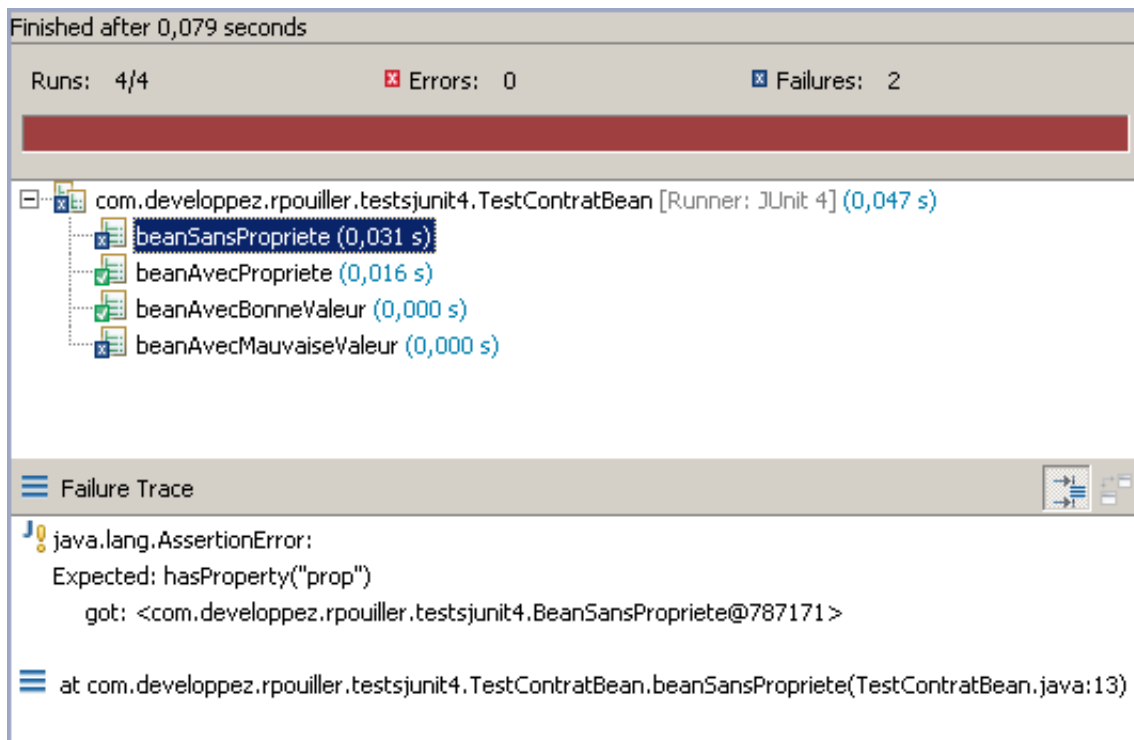
    @Test
    public void beanAvecPropriete() {
        assertThat(new BeanAvecPropriete(), hasProperty("prop"));
    }

    @Test
    public void beanAvecBonneValeur() {
        final BeanAvecPropriete lBean = new BeanAvecPropriete();
        lBean.setProp(10);
        assertThat(lBean, hasProperty("prop", equalTo(10)));
    }

    @Test
    public void beanAvecMauvaiseValeur() {
        final BeanAvecPropriete lBean = new BeanAvecPropriete();
        assertThat(lBean, hasProperty("prop", equalTo(10)));
    }

}
```

Comme le montre la capture d'écran ci-dessous, le contrat valide la présence d'une propriété et le respect d'un contrat par la valeur d'une propriété.



IV-C-9-b - Les contrats collection

Le contrat "IsArray" vérifie que les éléments d'un tableau respecte un tableau de contrats (un contrat par élément). La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsArrayContaining" vérifie qu'un tableau contient au moins un élément qui respecte un contrat passé en paramètre ou qui est égal à une valeur passée en paramètre. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsCollectionContaining" vérifie qu'une Collection contient des éléments qui respectent un ou des contrats passés en paramètres ou qui sont égaux à une valeur ou des valeurs passées en paramètres. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsIn" vérifie qu'un objet est contenu dans une Collection, un tableau ou parmi les paramètres du contrat. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsMapContaining" vérifie qu'une Map contient une entrée, une clé ou valeur qui respecte un contrat ou qui sont égaux à une valeur passée en paramètre. La javadoc de ce contrat est disponible [ici](#).

TestContratCollectionIsArray.java

```

package com.developpez.rpouiller.testsjunit4;

import static org.junit.Assert.assertThat;
import static org.hamcrest.collection.IsArray.array;
import static org.hamcrest.Matchers.is;

import org.junit.Test;

public class TestContratCollectionIsArray {

    @Test
    public void elementsEgaux() {
        assertThat(new Integer[]{1, 2}, array(is(1), is(2)));
    }

    @Test
    public void elementsDifférents() {

```

TestContratCollectionIsArray.java

```

    assertThat(new Integer[]{1, 2}, array(is(1), is(3)));
}

```

Comme le montre la capture d'écran ci-dessous, le contrat valide le respect par chaque élément du tableau d'un contrat.

Finished after 0,047 seconds

Runs: 2/2 ✖ Errors: 0 ✖ Failures: 1

com.developpez.rpouiller.testsjunit4.TestContratCollectionIsArray [Runner: JUnit 4] (0,032 s)

- elementsEgaux (0,016 s) ✔
- elementsDifférents (0,016 s) ✖

Failure Trace

```

java.lang.AssertionError:
  Expected: [is <1>, is <3>]
    got: [<1>, <2>]

at com.developpez.rpouiller.testsjunit4.TestContratCollectionIsArray.elementsDifférents(TestContratCollectionIsArray.java:18)

```

TestContratCollectionIsArrayContaining.java

```

package com.developpez.rpouiller.testsjunit4;

import static org.junit.Assert.assertThat;
import static org.hamcrest.Matchers.hasItemInArray;
import static org.hamcrest.Matchers.nullValue;

import org.junit.Test;

public class TestContratCollectionIsArrayContaining {

    @Test
    public void contientElementContrat() {
        assertThat(new Long[]{1L, null}, hasItemInArray(nullValue()));
    }

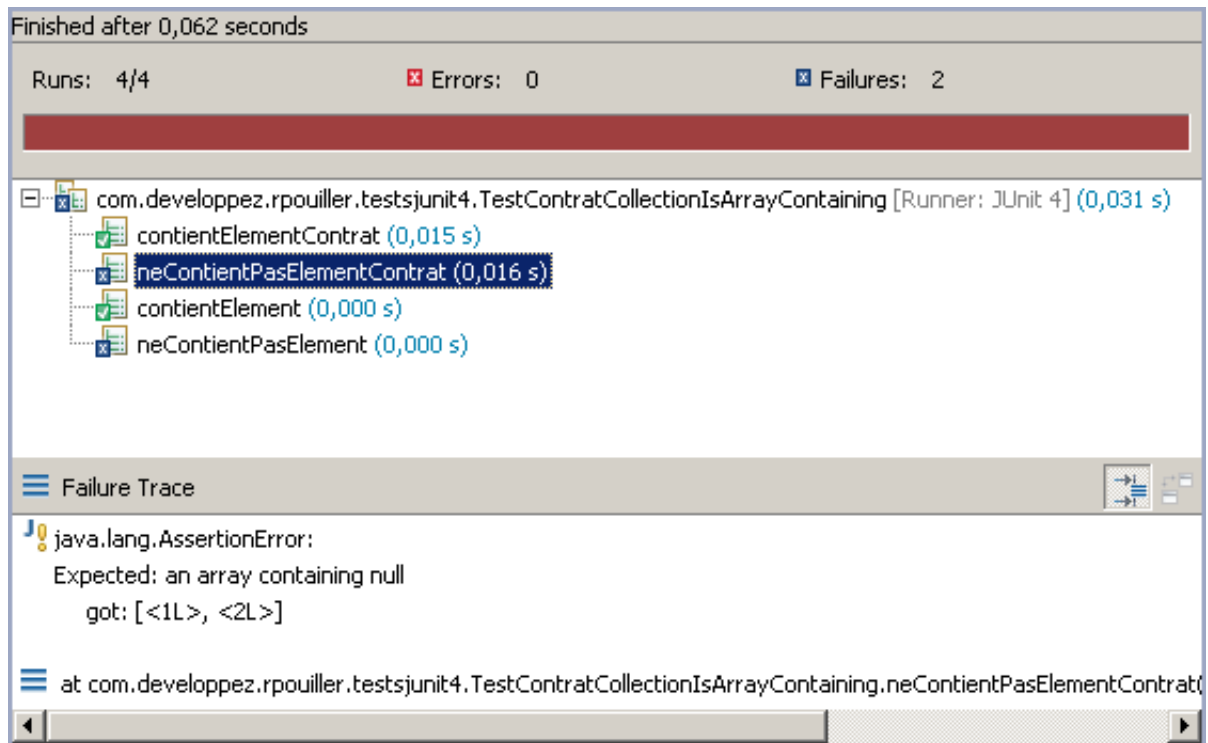
    @Test
    public void neContientPasElementContrat() {
        assertThat(new Long[]{1L, 2L}, hasItemInArray(nullValue()));
    }

    @Test
    public void contientElement() {
        assertThat(new Long[]{1L, 2L}, hasItemInArray(1L));
    }

    @Test
    public void neContientPasElement() {
        assertThat(new Long[]{1L, 2L}, hasItemInArray(3L));
    }
}

```


Comme le montre la capture d'écran ci-dessous, le contrat valide que le tableau possède au moins un élément avec la valeur ou respectant le contrat passé en paramètre.



TestContratCollectionIsCollectionContaining.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.hasItem;
import static org.hamcrest.Matchers.hasItems;
import static org.hamcrest.Matchers.is;
import static org.junit.Assert.assertThat;

import java.util.Arrays;

import org.junit.Test;

public class TestContratCollectionIsCollectionContaining {

    @Test
    public void contientElementContrat() {
        assertThat(Arrays.asList(1L, 2L), hasItem(is(1L)));
    }

    @Test
    public void neContientPasElementContrat() {
        assertThat(Arrays.asList(3L, 2L), hasItem(is(1L)));
    }

    @Test
    public void contientElementsContrat() {
        assertThat(Arrays.asList(1L, 2L), hasItems(is(1L), is(2L)));
    }

    @Test
    public void neContientPasElementsContrat() {
        assertThat(Arrays.asList(1L, null), hasItems(is(1L), is(2L)));
    }

    @Test
    public void contientElement() {
        assertThat(Arrays.asList(1L, 2L), hasItem(1L));
    }
}
```

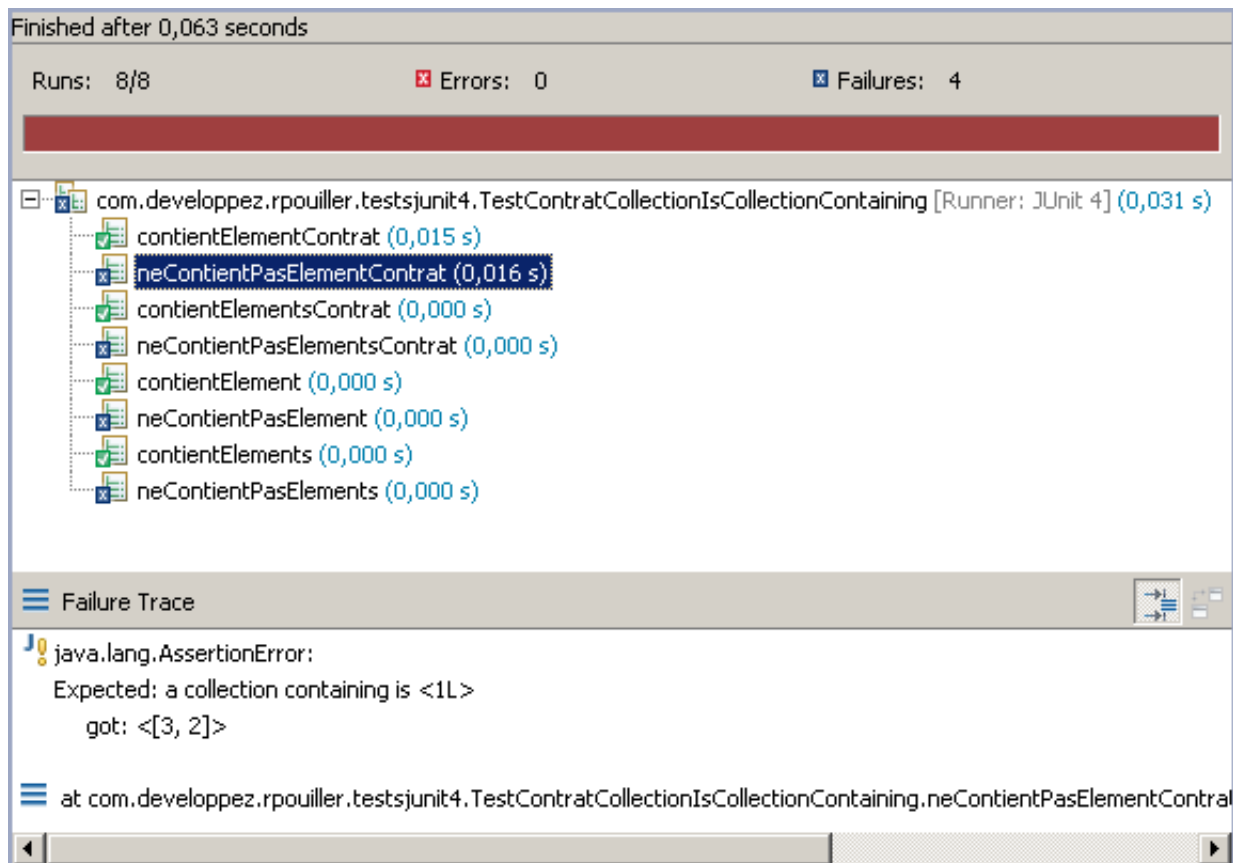
TestContratCollectionIsCollectionContaining.java

```
@Test
public void neContientPasElement() {
    assertThat(Arrays.asList(1L, 2L), hasItem(3L));
}

@Test
public void contientElements() {
    assertThat(Arrays.asList(1L, 2L), hasItems(1L, 2L));
}

@Test
public void neContientPasElements() {
    assertThat(Arrays.asList(1L, 2L), hasItems(1L, 2L, 3L));
}
}
```

Comme le montre la capture d'écran ci-dessous, les contrats valident que la collection possède au moins un élément ou des éléments avec la/les valeur(s) ou respectant le(s) contrat(s) passé(s) en paramètre(s).



TestContratCollectionIsIn.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.isIn;
import static org.hamcrest.Matchers.isOneOf;
import static org.junit.Assert.assertThat;

import java.util.Arrays;

import org.junit.Test;

public class TestContratCollectionIsIn {

    @Test
    public void presentDansCollection() {
```

TestContratCollectionIsIn.java

```

    assertThat("a", isIn(Arrays.asList("a", "z", "e", "r", "t", "y")));
}

@Test
public void absentDeCollection() {
    assertThat("A", isIn(Arrays.asList("a", "z", "e", "r", "t", "y")));
}

@Test
public void presentDansTableau() {
    assertThat("a", isIn(new String[]{"a", "z", "e", "r", "t", "y"}));
}

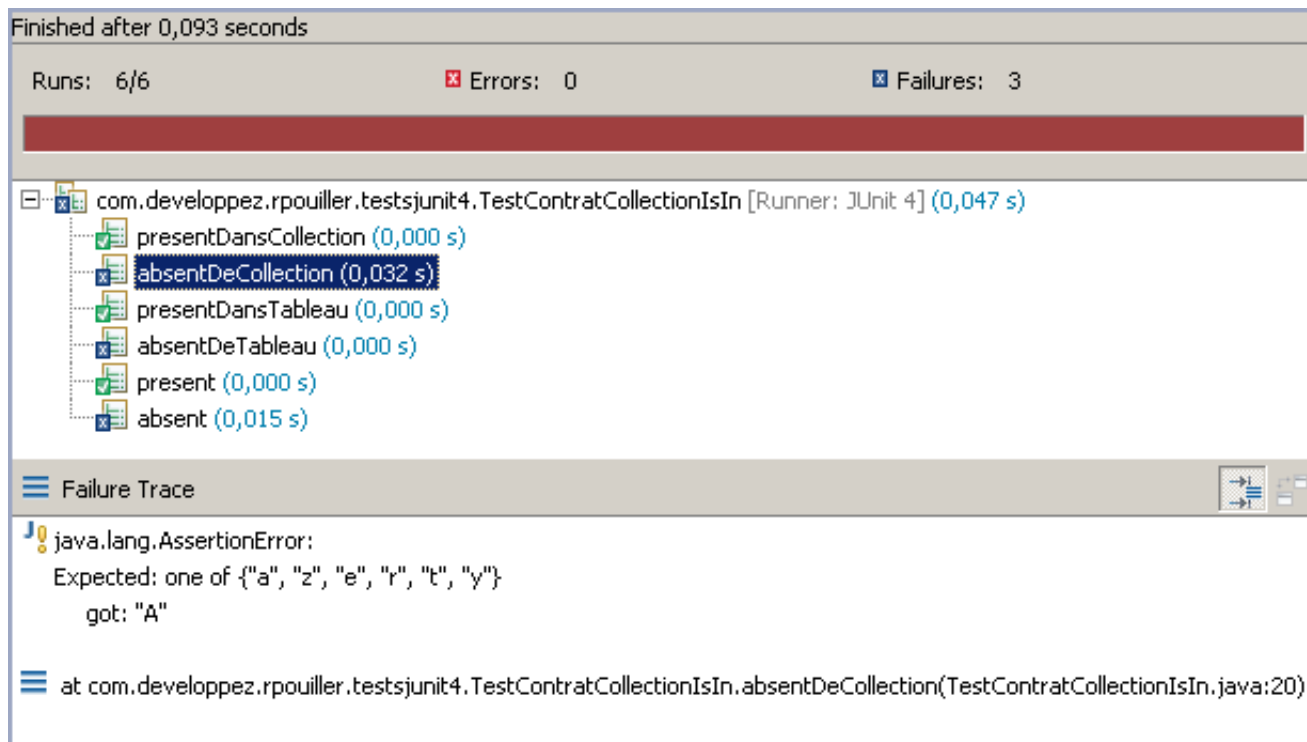
@Test
public void absentDeTableau() {
    assertThat("A", isIn(new String[]{"a", "z", "e", "r", "t", "y"}));
}

@Test
public void present() {
    assertThat("a", isOneOf("a", "z", "e", "r", "t", "y"));
}

@Test
public void absent() {
    assertThat("A", isOneOf("a", "z", "e", "r", "t", "y"));
}
}

```

Comme le montre la capture d'écran ci-dessous, les contrats valident que la valeur appartient à un ensemble.



Finished after 0,093 seconds

Runs: 6/6 Errors: 0 Failures: 3

com.developpez.rpouiller.testsjunit4.TestContratCollectionIsIn [Runner: JUnit 4] (0,047 s)

- presentDansCollection (0,000 s)
- absentDeCollection (0,032 s)
- presentDansTableau (0,000 s)
- absentDeTableau (0,000 s)
- present (0,000 s)
- absent (0,015 s)

Failure Trace

```

java.lang.AssertionError:
Expected: one of {"a", "z", "e", "r", "t", "y"}
got: "A"

at com.developpez.rpouiller.testsjunit4.TestContratCollectionIsIn.absentDeCollection(TestContratCollectionIsIn.java:20)

```

TestContratCollectionIsMapContaining.java

```

package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.hasEntry;
import static org.hamcrest.Matchers.is;
import static org.hamcrest.Matchers.not;
import static org.junit.Assert.assertThat;

import java.util.HashMap;

```

TestContratCollectionIsMapContaining.java

```
import java.util.Map;

import org.hamcrest.collection.IsMapContaining;
import org.junit.Before;
import org.junit.Test;

public class TestContratCollectionIsMapContaining {

    private Map<String, Long> hashMap;

    @Before
    public void initialise() {
        hashMap = new HashMap<String, Long>();
        hashMap.put("cle1", 1L);
        hashMap.put("cle2", 2L);
    }

    @Test
    public void contientEntree() {
        assertThat(hashMap, hasEntry("cle1", 1L));
    }

    @Test
    public void neContientPasEntree() {
        assertThat(hashMap, hasEntry("cle1", 2L));
    }

    @Test
    public void contientEntreeContrat() {
        assertThat(hashMap, hasEntry(is("cle1"), not(2L)));
    }

    @Test
    public void neContientPasEntreeContrat() {
        assertThat(hashMap, hasEntry(is("cle1"), not(1L)));
    }

    @Test
    public void contientCle() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasKey("cle1"));
    }

    @Test
    public void neContientPasCleEntree() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasKey("cle3"));
    }

    @Test
    public void contientCleContrat() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasKey(is("cle1")));
    }

    @Test
    public void neContientPasCleContrat() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasKey(is("cle3")));
    }

    @Test
    public void contientValeur() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasValue(1L));
    }

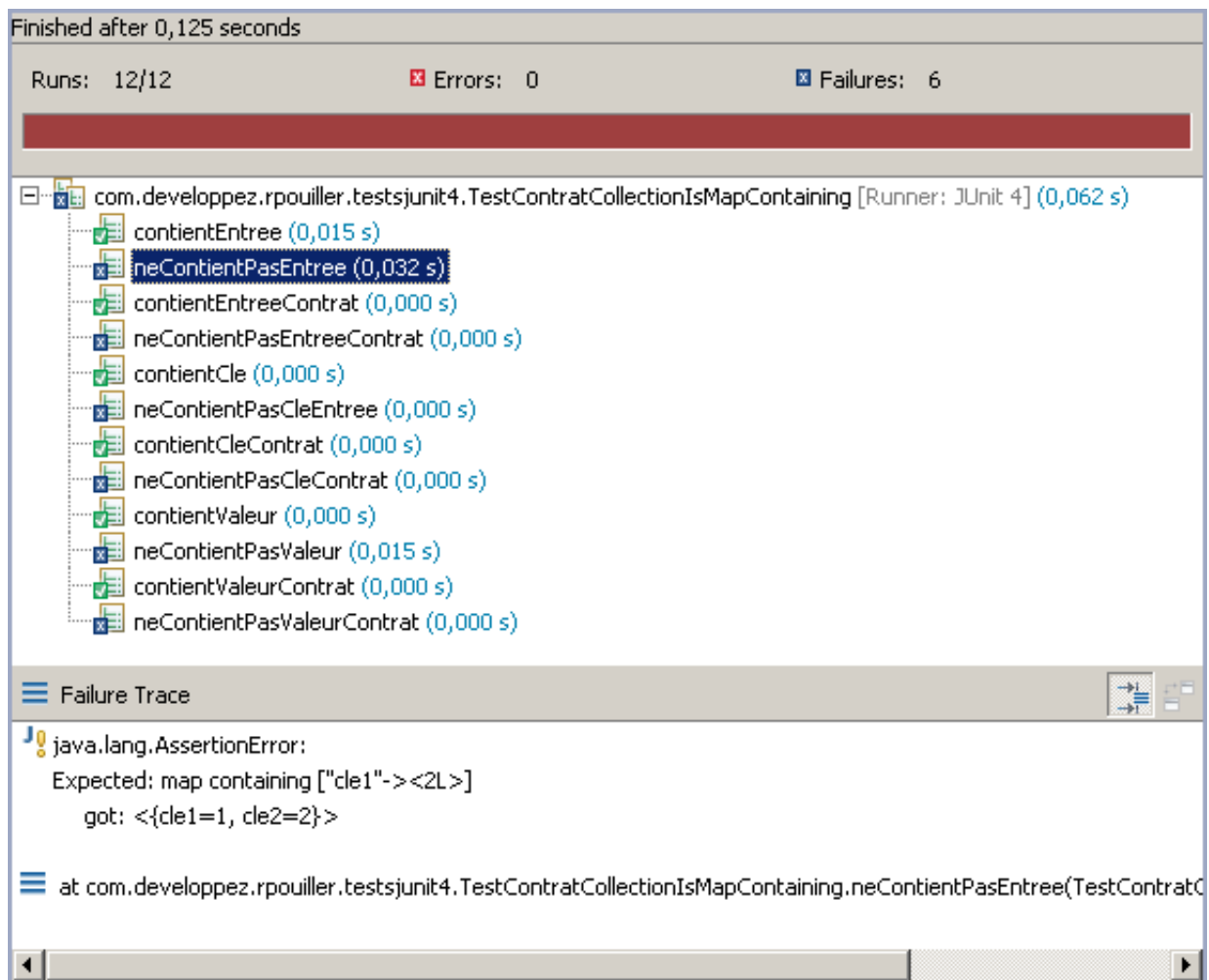
    @Test
    public void neContientPasValeur() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasValue(3L));
    }

    @Test
    public void contientValeurContrat() {
        assertThat(hashMap, IsMapContaining.<String, Long>hasValue(is(1L)));
    }
}
```

TestContratCollectionIsMapContaining.java

```
@Test
public void neContientPasValeurContrat() {
    assertThat(hashMap, IsMapContaining.<String, Long>hasValue(is(3L)));
}
```

Comme le montre la capture d'écran ci-dessous, les contrats valident qu'un objet Map possède une entrée, une clé ou une valeur ou respectant un contrat passé en paramètre.



IV-C-9-c - Les contrats number

Le contrat "IsCloseTo" vérifie qu'un nombre passé en paramètre est suffisamment proche d'un autre nombre. La javadoc de ce contrat est disponible [ici](#).

Le contrat "OrderingComparisons" vérifie qu'un nombre passé en paramètre est supérieur, supérieur ou égal, inférieur, inférieur ou égal à une autre valeur. La javadoc de ce contrat est disponible [ici](#).

TestContratNumber.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.closeTo;
import static org.hamcrest.Matchers.greaterThan;
import static org.hamcrest.Matchers.greaterThanOrEqualTo;
import static org.hamcrest.Matchers.lessThan;
import static org.hamcrest.Matchers.lessThanOrEqualTo;
```

TestContratNumber.java

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

import org.junit.Test;

public class TestContratNumber {

    @Test
    public void proche() {
        assertThat(20.5, closeTo(20, 0.5));
    }

    @Test
    public void pasProche() {
        assertThat(20.5, closeTo(20, 0.1));
    }

    @Test
    public void plusGrand() {
        assertThat(21, greaterThan(20));
    }

    @Test
    public void pasPlusGrand() {
        assertThat(20, greaterThan(20));
    }

    @Test
    public void plusGrandOuEgal() {
        assertThat(20, greaterThanOrEqualTo(20));
    }

    @Test
    public void pasPlusGrandOuEgal() {
        assertThat(19, greaterThanOrEqualTo(20));
    }

    @Test
    public void plusPetit() {
        assertThat(19, lessThan(20));
    }

    @Test
    public void pasPlusPetit() {
        assertThat(20, lessThan(20));
    }

    @Test
    public void plusPetitOuEgal() {
        assertThat(20, lessThanOrEqualTo(20));
    }

    @Test
    public void pasPlusPetitOuEgal() {
        assertThat(21, lessThanOrEqualTo(20));
    }
}
```

Comme le montre la capture d'écran ci-dessous, les contrats valident la comparaison entre les deux nombres.

Finished after 0,094 seconds

Runs: 10/10 ✖ Errors: 0 ✖ Failures: 5

com.developpez.rpouiller.testsjunit4.TestContratNumber [Runner: JUnit 4] (0,031 s)

- ✓ proche (0,000 s)
- ✖ pasProche (0,015 s)
- ✓ plusGrand (0,016 s)
- ✖ pasPlusGrand (0,000 s)
- ✓ plusGrandOuEgal (0,000 s)
- ✖ pasPlusGrandOuEgal (0,000 s)
- ✓ plusPetit (0,000 s)
- ✖ pasPlusPetit (0,000 s)
- ✓ plusPetitOuEgal (0,000 s)
- ✖ pasPlusPetitOuEgal (0,000 s)

Failure Trace

java.lang.AssertionError:
Expected: a numeric value within <0.1> of <20.0>
got: <20.5>

at com.developpez.rpouiller.testsjunit4.TestContratNumber.pasProche(TestContratNumber.java:19)

IV-C-9-d - Les contrats object

Le contrat "HasToString" vérifie que la valeur retournée par le toString() de l'objet passé en paramètre respecte un contrat. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsCompatibleType" vérifie que la classe passée en paramètre est compatible avec celle passée en paramètre du contrat. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsEventFrom" vérifie l'origine (et la classe) d'un évènement. La javadoc de ce contrat est disponible [ici](#).

TestContratObject.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.hasToString;
import static org.hamcrest.Matchers.is;
import static org.hamcrest.Matchers.typeCompatibleWith;
import static org.junit.Assert.assertThat;

import org.junit.Test;

public class TestContratObject {

    @Test
    public void toStringOK() {
        assertThat(new Long(30), hasToString(is("30")));
    }

    @Test
    public void toStringNOK() {
        assertThat(new Long(30), hasToString(is(" 30")));
    }

    @Test
    public void compatible() {
```

TestContratObject.java

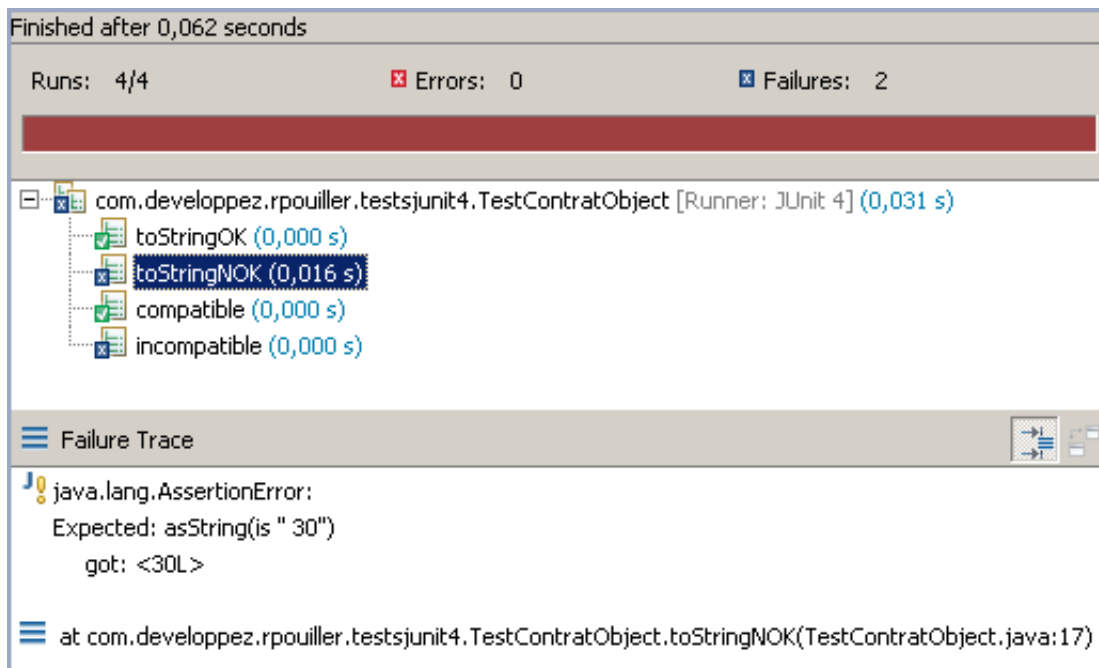
```

    assertThat(Long.class, typeCompatibleWith(Number.class));
  }

  @Test
  public void incompatible() {
    assertThat(Long.class, typeCompatibleWith(String.class));
  }
}

```

Comme le montre la capture d'écran ci-dessous, les contrats valident la valeur retournée par le toString() d'un objet et la compatibilité entre deux classes.



TestContratObjectIsEventFrom.java

```

package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.eventFrom;
import static org.junit.Assert.assertThat;

import java.beans.PropertyChangeEvent;
import javax.swing.event.ChangeEvent;

import org.junit.Before;
import org.junit.Test;

public class TestContratObjectIsEventFrom {

    private Object source;
    private ChangeEvent event;

    @Before
    public void initialise() {
        source = "source";
        event = new ChangeEvent(source);
    }

    @Test
    public void envenementOrigine() {
        assertThat(event, eventFrom(source));
    }

    @Test

```

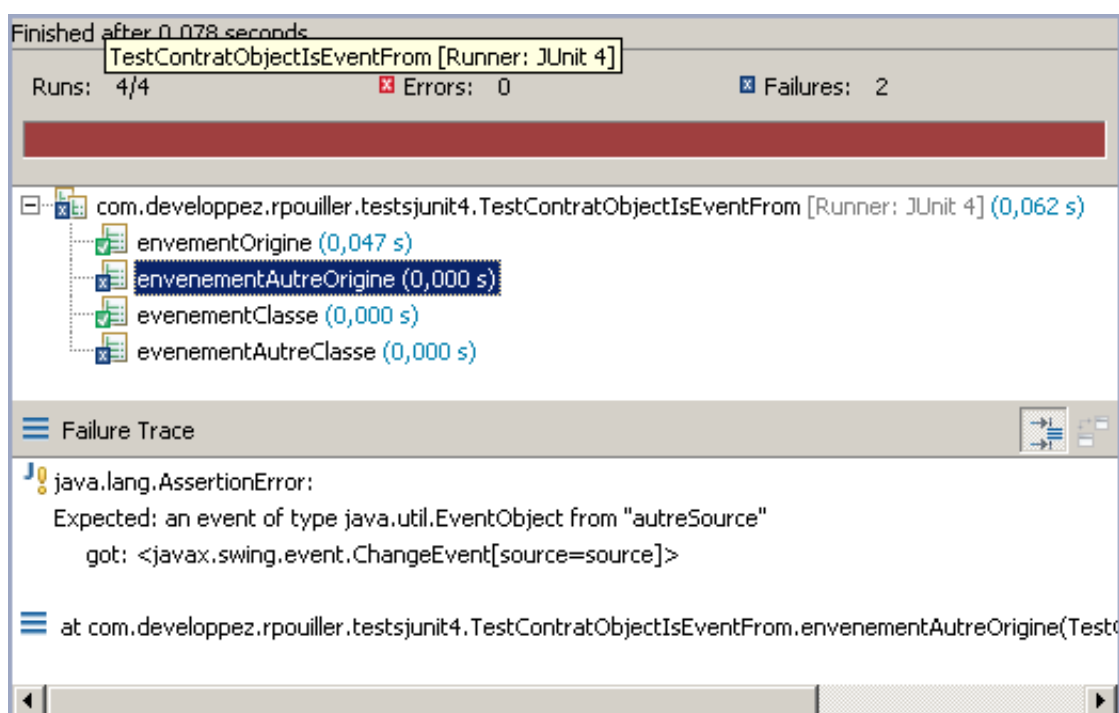

TestContratObjectIsEventFrom.java

```
public void evenementAutreOrigine() {
    assertThat(event, eventFrom("autreSource"));
}

@Test
public void evenementClasse() {
    assertThat(event, eventFrom(ChangeEvent.class, source));
}

@Test
public void evenementAutreClasse() {
    assertThat(event, eventFrom(PropertyChangeEvent.class, source));
}
}
```

Comme le montre la capture d'écran ci-dessous, le contrat valide l'origine d'un évènement et sa classe.



IV-C-9-e - Les contrats text

Le contrat "IsEqualIgnoringCase" vérifie l'égalité de deux chaînes de caractères sans tenir compte de la casse. La javadoc de ce contrat est disponible [ici](#).

Le contrat "IsEqualIgnoringWhiteSpace" vérifie l'égalité de deux chaînes de caractères sans tenir compte des espaces. La javadoc de ce contrat est disponible [ici](#).

Le contrat "StringContains" vérifie que la chaîne de caractères passée en paramètre contient la chaîne de caractère passée en paramètre du contrat. La javadoc de ce contrat est disponible [ici](#).

Le contrat "StringEndsWith" vérifie que la chaîne de caractère passée en paramètre finit par la chaîne de caractère passée en paramètre du contrat. La javadoc de ce contrat est disponible [ici](#).

Le contrat "StringStartsWith" vérifie que la chaîne de caractère passée en paramètre commence par la chaîne de caractères passée en paramètre du contrat. La javadoc de ce contrat est disponible [ici](#).

TestContratText.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.containsString;
import static org.hamcrest.Matchers.endsWith;
import static org.hamcrest.Matchers.equalToIgnoringCase;
import static org.hamcrest.Matchers.equalToIgnoringWhiteSpace;
import static org.hamcrest.Matchers.startsWith;
import static org.junit.Assert.assertThat;

import org.junit.Test;

public class TestContratText {

    @Test
    public void egalCasse() {
        assertThat("azerty", equalToIgnoringCase("Azerty"));
    }

    @Test
    public void pasEgalCasse() {
        assertThat("azerty", equalToIgnoringCase("Qwerty"));
    }

    @Test
    public void egalEspace() {
        assertThat("a\nz e r t y", equalToIgnoringWhiteSpace("a z e r \n t y"));
    }

    @Test
    public void pasEgalEspace() {
        assertThat("a z e r t y", equalToIgnoringWhiteSpace("q w e r t y"));
    }

    @Test
    public void contient() {
        assertThat("azerty", containsString("zert"));
    }

    @Test
    public void neContientPas() {
        assertThat("azerty", containsString("wert"));
    }

    @Test
    public void commencePar() {
        assertThat("azerty", startsWith("azer"));
    }

    @Test
    public void neCommencePasPar() {
        assertThat("azerty", startsWith("qwer"));
    }

    @Test
    public void finitPar() {
        assertThat("azerty", endsWith("erty"));
    }

    @Test
    public void neFinitPasPar() {
        assertThat("azerty", endsWith("erti"));
    }
}
```

Comme le montre la capture d'écran ci-dessous, les contrats valident les relations entre les deux chaînes de caractères.


Finished after 0,078 seconds

Runs: 10/10 ✖ Errors: 0 ✖ Failures: 5

com.developpez.rpouiller.testsjunit4.TestContratText [Runner: JUnit 4] (0,031 s)

- ✔ egalCasse (0,000 s)
- ✖ **pasEgalCasse (0,016 s)**
- ✔ egalEspace (0,000 s)
- ✖ pasEgalEspace (0,015 s)
- ✔ contient (0,000 s)
- ✖ neContientPas (0,000 s)
- ✔ commencePar (0,000 s)
- ✖ neCommencePasPar (0,000 s)
- ✔ finitPar (0,000 s)
- ✖ neFinitPasPar (0,000 s)

Failure Trace

 java.lang.AssertionError:
 Expected: eqIgnoringCase("Qwerty")
 got: "azerty"
 at com.developpez.rpouiller.testsjunit4.TestContratText.pasEgalCasse(TestContratText.java:19)

IV-C-9-f - Les contrats xml

Le contrat "HasXPath" vérifie qu'un chemin existe et/ou respecte un contrat dans un document XML. La javadoc de ce contrat est disponible [ici](#).

TestContratXML.java

```

package com.developpez.rpouiller.testsjunit4;

import static org.hamcrest.Matchers.equalToIgnoringCase;
import static org.hamcrest.Matchers.hasXPath;
import static org.junit.Assert.assertThat;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.junit.Before;
import org.junit.Test;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class TestContratXML {

    private Document document;

    @Before
    public void initialise() throws ParserConfigurationException {
        final DocumentBuilderFactory lDocumentBuilderFactory = DocumentBuilderFactory.newInstance();
        final DocumentBuilder lDocumentBuilder = lDocumentBuilderFactory.newDocumentBuilder();

        document = lDocumentBuilder.newDocument();

        final Element lElement = document.createElement("racine");
        document.appendChild(lElement);
    }
  
```

TestContratXML.java

```

    final Element lElement1 = document.createElement("element1");
    lElement1.appendChild(document.createTextNode("texteElement1"));
    lElement.appendChild(lElement1);
    final Element lElement2 = document.createElement("element2");
    lElement2.appendChild(document.createTextNode("texteElement2"));
    lElement.appendChild(lElement2);
  }

  @Test
  public void avecChemin() {
    assertThat(document, hasXPath("/racine/element1"));
  }

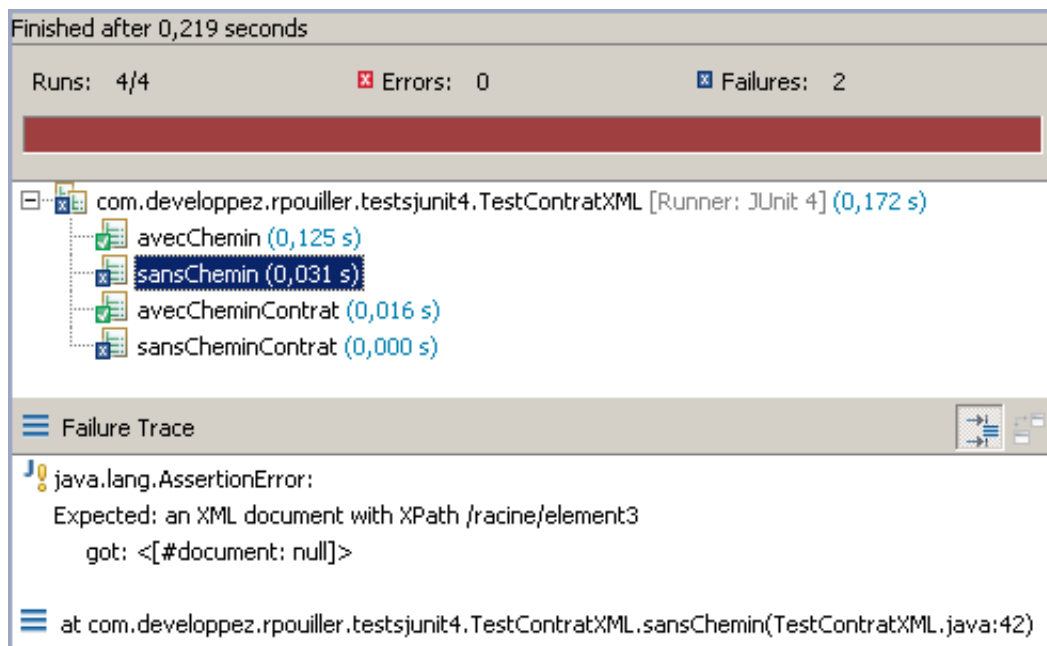
  @Test
  public void sansChemin() {
    assertThat(document, hasXPath("/racine/element3"));
  }

  @Test
  public void avecCheminContrat() {
    assertThat(document, hasXPath("/racine/element1", equalToIgnoringCase("texteelement1")));
  }

  @Test
  public void sansCheminContrat() {
    assertThat(document, hasXPath("/racine/element1", equalToIgnoringCase("texteelement2")));
  }
}

```

Comme le montre la capture d'écran ci-dessous, le contrat valide que le document XML possède un chemin et que ce chemin respecte un contrat.



V - LES SUPPOSITIONS

La javadoc des suppositions est disponible [ici](#).

Les suppositions sont présentes dans JUnit depuis la version 4.4.

Une supposition vérifie une condition. Si la condition n'est pas vérifiée, le test s'arrête mais ne passe pas en erreur ou en échec.

Les différentes suppositions sont :

- **assumeNoException(java.lang.Throwable)** : vérifie qu'une opération s'est déroulée sans lever de Throwable.
- **assumeNotNull(java.lang.Object...)** : vérifie qu'aucun des paramètres n'est à null.
- **assumeThat(T, org.hamcrest.Matcher)** : vérifie qu'une condition définie par contrat est respectée (voir "[Assertion sur une condition définie par contrat](#)").
- **assumeTrue(boolean)** : vérifie que le paramètre est vrai.

TestSupposition.java

```
package com.developpez.rpouiller.testsjunit4;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.junit.Assert;
import org.junit.Assume;
import org.junit.Test;

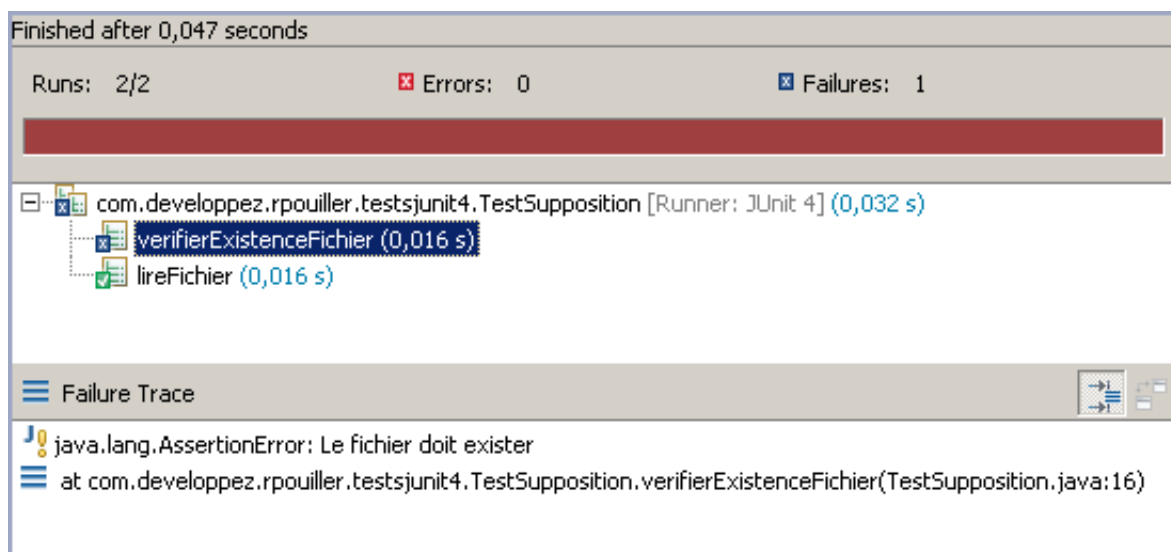
public class TestSupposition {

    @Test
    public void verifierExistenceFichier() {
        final File lFile = new File("fichier.txt");
        Assert.assertTrue("Le fichier doit exister", lFile.exists());
    }

    @Test
    public void lireFichier() throws IOException {
        final File lFile = new File("fichier.txt");
        Assume.assertTrue(lFile.exists());

        final FileInputStream lFileInputStream = new FileInputStream(lFile);
        final byte[] lBytes = new byte[16];
        lFileInputStream.read(lBytes);
        Assert.assertArrayEquals("Bonjour le monde".getBytes(), lBytes);
    }
}
```

Comme le montre la capture d'écran ci-dessous, si le fichier n'existe pas, seul le test vérifiant la présence du fichier passe en échec.



VI - LES TESTS PARAMETRES

La javadoc des tests paramétrés est disponible [ici](#).

Les tests paramétrés permettent d'exécuter plusieurs fois un cas de tests avec des valeurs différentes. Le cas de test doit être annoté avec **@RunWith** avec pour valeur **Parameterized**. Les paramètres sont indiqués par une méthode annotée avec **@Parameters** retournant une Collection. Le cas de tests paramétrés doit comporter un constructeur correspondant aux paramètres. Les tests paramétrés sont apparus avec la version 4.0 de JUnit.

Operations.java

```
package com.developpez.rpouiller.testsjunit4;

public class Operations {

    public static long additionner(final long...pNombres) {
        long lRetour = 0;
        for(final long lNombre : pNombres) {
            lRetour += lNombre;
        }
        return lRetour;
    }

    // Cette méthode ne fonctionne pas correctement
    // Les tests vont le vérifier
    public static long multiplier(final long...pNombres) {
        long lRetour = 0;
        for(final long lNombre : pNombres) {
            lRetour *= lNombre;
        }
        return lRetour;
    }
}
```

TestParametre.java

```
package com.developpez.rpouiller.testsjunit4;

import java.util.Arrays;
import java.util.List;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.class)
public class TestParametre {

    @Parameters
    public static List<Object[]> getParametres() {
        return Arrays.asList(new Object[][] {
            { 10L, 0L, new long[] { 10L, 0L } },
            { 30L, 300L, new long[] { 10L, 20L } },
            { 7936L, 1125899906842624L, new long[] { 256L, 512L, 1024L, 2048L, 4096L } },
        });
    }

    private long resultatAddition;
    private long resultatMultiplication;
    private long[] nombres;

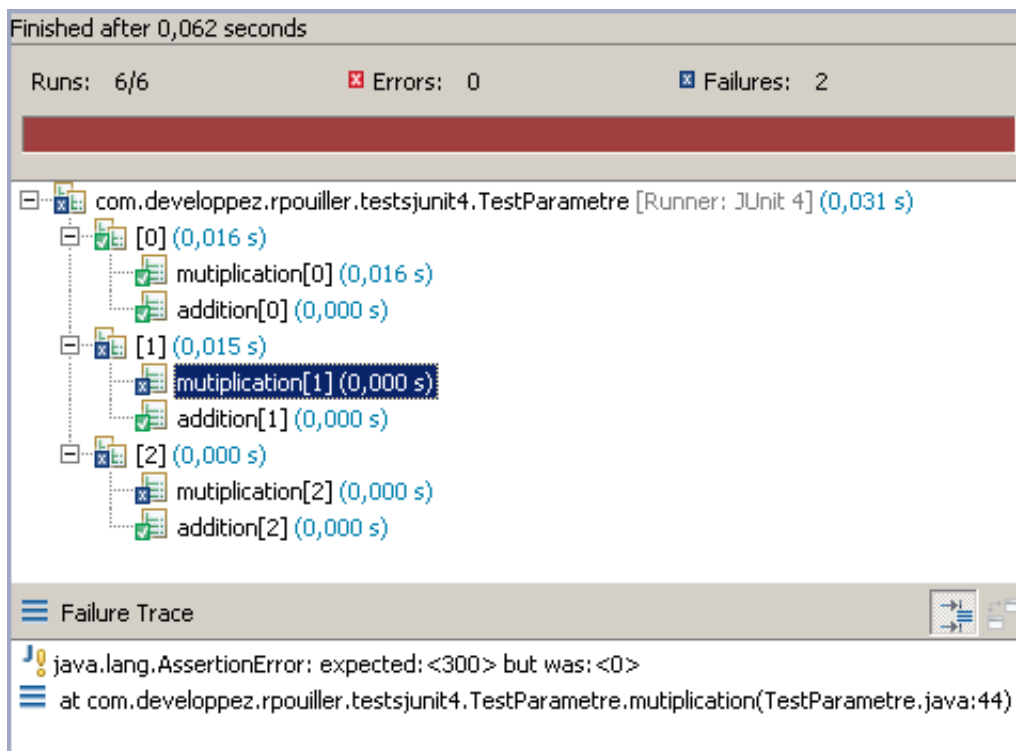
    public TestParametre(long pResultatAddition, long pResultatMultiplication, long[] pNombres) {
        resultatAddition = pResultatAddition;
        resultatMultiplication = pResultatMultiplication;
        nombres = pNombres;
    }
}
```

TestParametre.java

```
@Test
public void addition() {
    System.out.println("addition :      - attendu : " + resultatAddition +
        "\n                        - nombres : " + Arrays.toString(nombres));
    final long lAddition = Operations.additionner(nombres);
    Assert.assertEquals(resultatAddition, lAddition);
}

@Test
public void mutiplication() {
    System.out.println("mutiplication : - attendu : " + resultatMultiplication +
        "\n                        - nombres : " + Arrays.toString(nombres));
    final long lMultiplication = Operations.multiplier(nombres);
    Assert.assertEquals(resultatMultiplication, lMultiplication);
}
}
```

Comme le montre la capture d'écran et la trace ci-dessous, le cas de tests est exécuté plusieurs fois avec des paramètres différents.



```
mutiplication : - attendu : 0
                - nombres : [10, 0]
addition :      - attendu : 10
                - nombres : [10, 0]
mutiplication : - attendu : 300
                - nombres : [10, 20]
addition :      - attendu : 30
                - nombres : [10, 20]
mutiplication : - attendu : 1125899906842624
                - nombres : [256, 512, 1024, 2048, 4096]
addition :      - attendu : 7936
                - nombres : [256, 512, 1024, 2048, 4096]
```

VII - LES ANNOTATIONS POUR LES SUITES DE TESTS

La javadoc de l'annotation des suites de tests est disponible [ici](#).

Les suites des tests permettent d'exécuter plusieurs cas de tests dans la même exécution. La classe de suite doit être annotée avec **@RunWith** avec pour valeur **Suite**. Les cas de tests composants la suite sont indiqués par l'annotation **@SuiteClasses**.

TestSuitePartie1.java

```
package com.developpez.rpouiller.testsjunit4;

import org.junit.Test;

public class TestSuitePartie1 {

    @Test
    public void test1() {
    }

}
```

TestSuitePartie2.java

```
package com.developpez.rpouiller.testsjunit4;

import static org.junit.Assert.fail;

import org.junit.Test;

public class TestSuitePartie2 {

    @Test
    public void test2() {
        fail();
    }

}
```

TestSuite.java

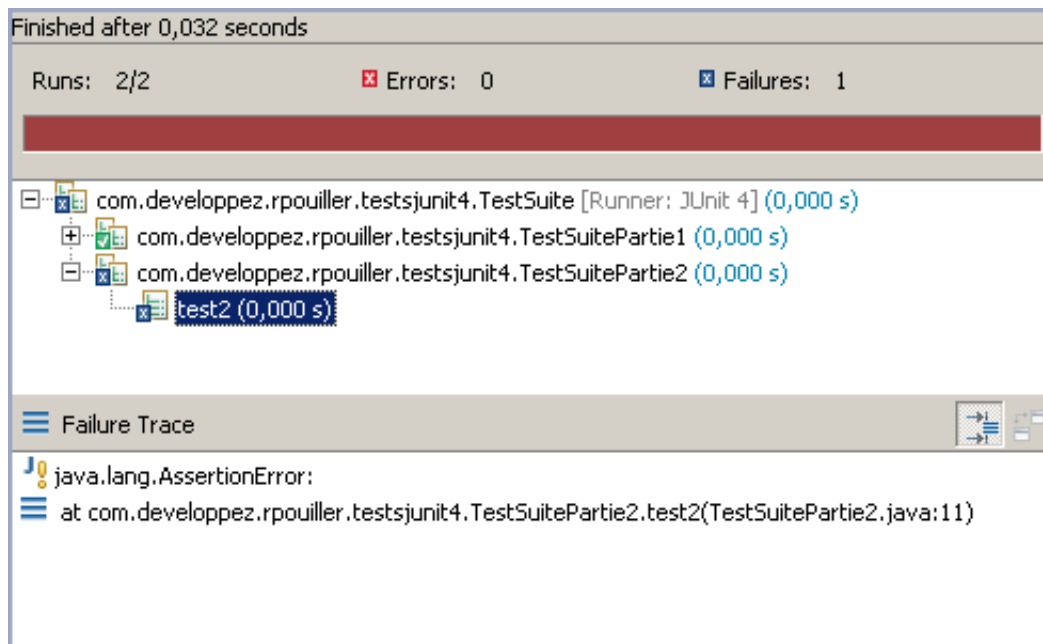
```
package com.developpez.rpouiller.testsjunit4;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({TestSuitePartie1.class, TestSuitePartie2.class})
public class TestSuite {

}
```

Comme le montre la capture d'écran, les différents cas de tests sont exécutés dans la même exécution.



VIII - REMERCIEMENTS

Je remercie sincèrement par ordre plus ou moins chronologique :

- www.developpez.com me permettant de publier cet article.
- [Nono40](#) pour ses outils.
- [Mikrob](#) pour ses encouragements.
- [djo.mos](#) pour ses suggestions pertinentes et ses encouragements.
- [romaintaz](#) pour sa relecture orthographique rigoureuse et ses encouragements.

IX - SOURCES

Sources ([Miroir](#))