JUNIT

Description de la séquence

Description du contenu : JUnit

Durée de travail estimée :

Auteur : Yannick.Parchemal@parisdescartes.fr
Objectifs : voir comment réaliser des tests unitaires avec JUnit

Connaissances acquises : les possibilités de base de JUnit
Compétences acquises : savoir utiliser Junit pour faire des tests unitaires
Pré requis : Connaissance de bases et pratique du langage Java

Test unitaires

Pouvoir tester facilement toutes les fonctions de son programme
Avoir un bilan synthétique des tests

→ Tests réalisés tout au long du développement à chaque build

→ Peuvent permettre la détection de problèmes de régression

En java : JUnit version 4.12 (10/2015)

1 méthode à tester → 1 ou plusieurs méthodes de test

Une méthode à tester …

```
public class Util1 {
    /**…*/
    public static String getStringFromChar(int nb, char c) {
        StringBuffer sb = new StringBuffer( );
        for (int i=0;i<nb;i++)
            sb.append(i);
        return sb.toString( );
    }
}
```

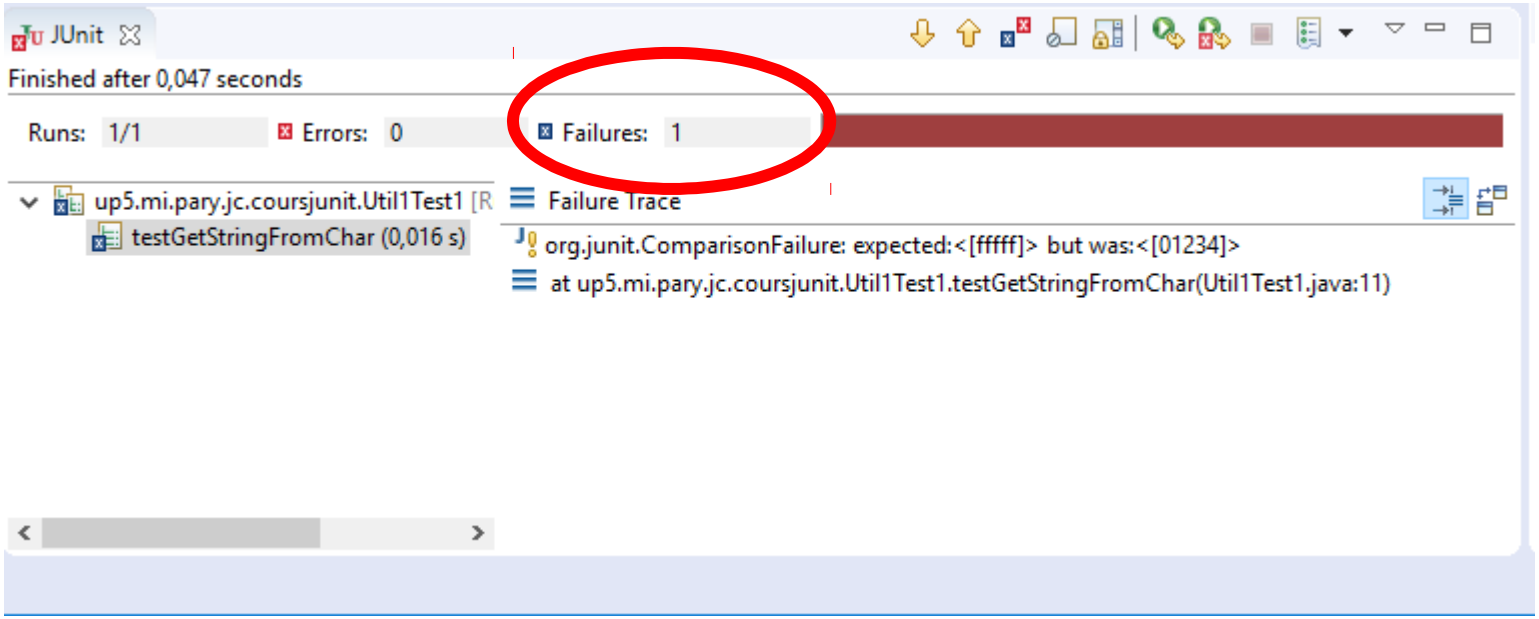
Une classe de test

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

public class Util1Test {

    @Test
    public void testGetStringFromChar() {
        Assert.assertEquals("fffff",Util1.getStringFromChar(5, 'f'));
    }

}
```

assertEquals lance une java.lang.assertionError
si les deux arguments ne sont pas equals

Lancement des tests unitaires

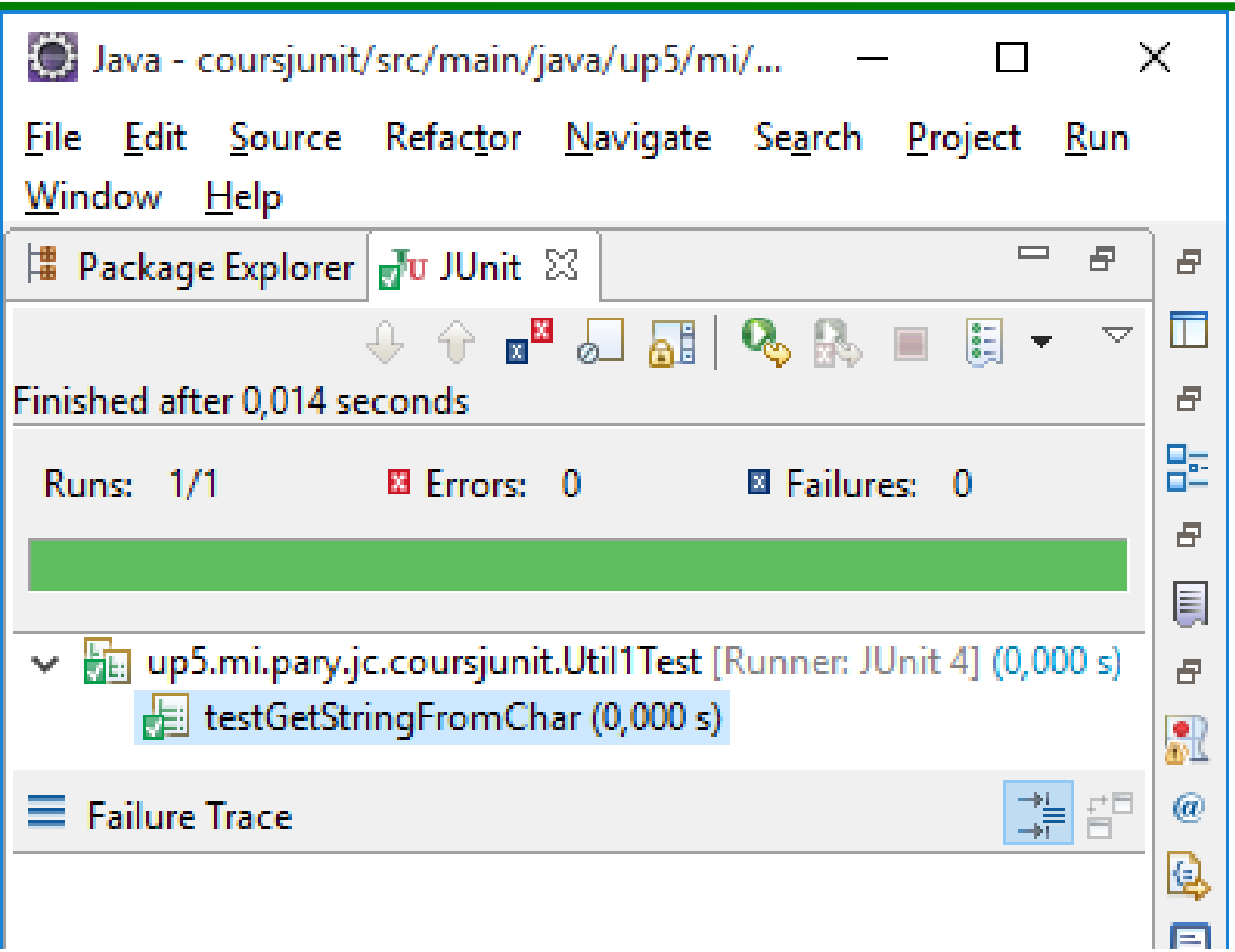
Eclipse : run as… JUnit Test



Correction de l'erreur

```
public class Util1 {
    /**…*/
    public static String getStringFromChar(int nb, char c) {
        StringBuffer sb = new StringBuffer( );
        for (int i=0;i<nb;i++)
            sb.append(c);   //  et pas   sb.append(i) !!
        return sb.toString( );
    }
}
```

Nouveau lancement des tests unitaires



import static

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

import org.junit.Test;

public class Util1Test {

    @Test
    public void testGetStringFromChar() {
        Assert.assertEquals("fffff",Util1.getStringFromChar(5, 'f'));
    }

}
```

import static est une facilité syntaxique pour éviter de mettre le nom de la classe avant le nom de la méthode statique

Test unitaires avec Junit 4

Une fonction de test par fonction à tester …

```
public class Util1Test2 {
    @Test
    public void testGetStringFromChar() {
        assertEquals("fffff",Util1.getStringFromChar(5, 'f'));
        assertEquals("",Util1.getStringFromChar(0, 'f'));
    }
```

Plusieurs fonctions de test par fonction à tester …

```
public class Util1Test3 {
    @Test
    public void testGetStringFromCharShouldReturnfffff() {
        assertEquals("fffff",Util1.getStringFromChar(5, 'f'));
    }
    @Test
    public void testGetStringFromCharShouldReturnEmptyString() {
        assertEquals("",Util1.getStringFromChar(0, 'f'));
    }}
```

Une autre fonction à tester …

```
public class Util2 {
    /**
     * retourne une chaine de longueur donnée dont tous les caractères  sont égaux à un caractère donné
     * @param nb le nombre de répétitions souhaité du caractère
     * @param c le caractère
     * @return une chaine composée de 'nb' fois le caractère 'c'
     * @throws IllegalArgumentException si le nombre de répétitions est négatif
     */
    public static String getStringFromChar(int nb, char c) {
        if (nb<0)
            throw new IllegalArgumentException("nombre positif attendu :"+nb);
        StringBuffer sb = new StringBuffer( );
        for (int i=0;i<nb;i++)
            sb.append(c);
        return sb.toString( );
    }

}
```

Test unitaires avec Junit 4

```
public class Util2Test1 {

    @Test
    public void testGetStringFromChar() {
        assertEquals("fffff",Util2.getStringFromChar(5, 'f'));
        assertEquals("",Util2.getStringFromChar(0, 'f'));
    }

    @Test(expected=IllegalArgumentException.class)
    public void testGetStringFromCharShouldThrowsException() {
        Util2.getStringFromChar(-5, 'f');
    }
}
```

Test unitaires avec Junit 4

```
enum Cote {GAUCHE,DROITE;}
public class UtilCadrage
{    public static String getStringFromChar(int nb, char c) {...}
 /** retourne une chaine formatée sur une taille donnée avec troncature ou complément avec des étoiles selon le cas
 * @param chaine la chaine à cadrer
 * @param taille la taille de la chaine resultante
 * @param cote GAUCHE ou DROITE
 * @return une chaine correspondant à la chaine donnée formatée sur le nombre de caractères souhaitée
 * @throws IllegalArgumentException si la taille de la zone souhaitée est négative
 */  public static String cadrer(String chaine,int taille,Cote cote){
        if (taille<0) throw new IllegalArgumentException("Taille négative : "+taille);
        if (chaine.length( )>=taille)
            return chaine.substring(0,taille);
        else {
            String etoiles = getStringFromChar(taille-chaine.length(),'*');
            return (cote==Cote.GAUCHE)?chaine+ etoiles : etoiles +chaine;
        }
    }}
```

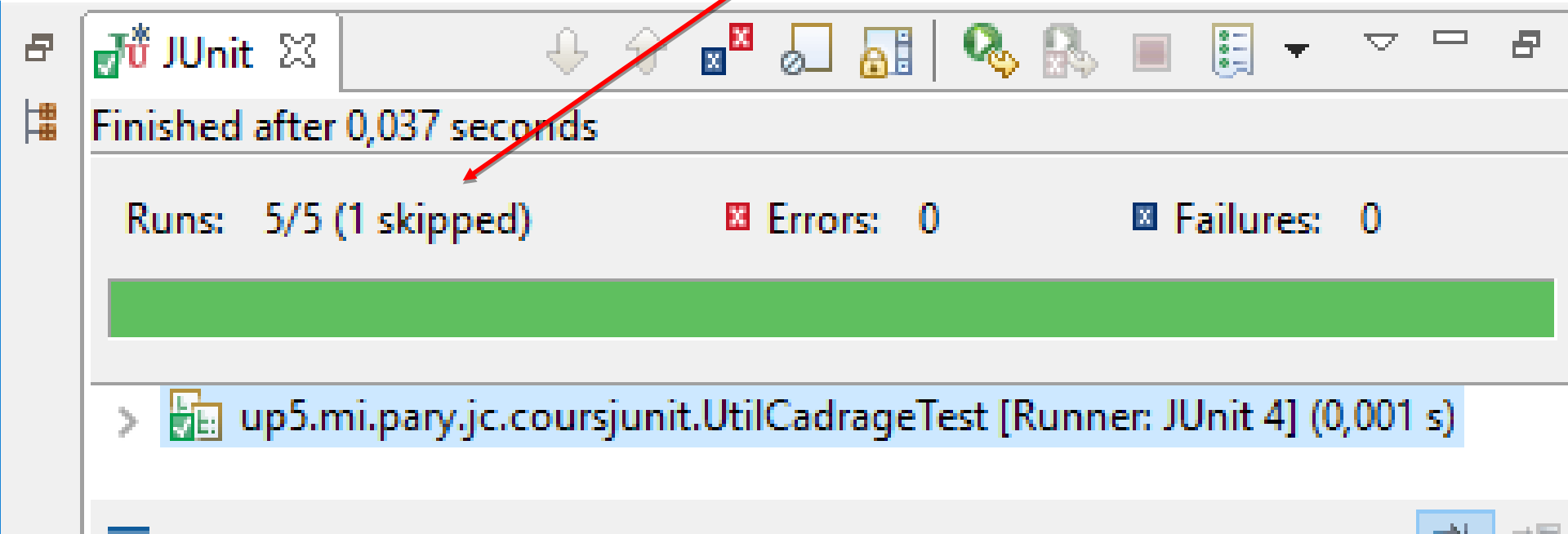
UtilCadrageTest

```
public class UtilCadrageTest {
    @Test
    public final void testCadrerEnCompletant() {
        assertEquals("Bon**",UtilCadrage.cadrer("Bon", 5, Cote.GAUCHE));
        assertEquals("**Bon",UtilCadrage.cadrer("Bon", 5, Cote.DROITE));
    }
    @Test
    public final void testCadrerEnTronquant() {
        assertEquals("Bon",UtilCadrage.cadrer("Bonjour", 3, Cote.GAUCHE));
        assertEquals("Bon",UtilCadrage.cadrer("Bonjour", 3, Cote.DROITE));
    }

    @Test(expected=IllegalArgumentException.class)
    public void testCadrerShouldThrowsException() {
        UtilCadrage.cadrer("Bonjour", -5, Cote.GAUCHE);
    }

    // mettre ici les tests de getStringFromChar
```

L'annotation @Ignore

```
@Ignore("ce test est ignoré à ce stade du développement")
@Test
public final void testAutreFonction() {
        …
}
```

Placé au dessus de l'annotation @Test, elle indique que ce test ne doit pas être effectuée actuellement



Une classe répertoire

```
/**
 * crée un répertoire vide
 */
public Repertoire( ){
}

/**
 * ajoute une nouvelle entrée à ce répertoire
 * @param nom le nom de la nouvelle entrée
 * @param tel le numéro de téléphone de la nouvelle entrée
 */
public void ajouterEntree(String nom,String tel) {
   … }

/** retourne le numéro de téléphone assoocié à un nom
 * @param nom le nom
 * throws RepertoireException  si aucun numéro de téléphone n'est associé à ce nom
 */
public String getTel(String nom) throws RepertoireException{
}
```

L'annotation @Before

```
public class RepertoireTest {
    private Repertoire rep=new Repertoire();;

    @Before
    public void initRep() {
        rep.ajouterEntree("Pierre", "(33)763646327");
        rep.ajouterEntree("Paul", "(33)663646122");
    }
    @Test
    public void testAjouterEntree() throws RepertoireException {
        rep.ajouterEntree("Julie", "(33)600553322");
        assertEquals("(33)600553322",rep.getTel("Julie"));
    }
    @Test
    public void testGetTel() throws RepertoireException {
        assertEquals("(33)763646327",rep.getTel("Pierre"));
    }
    @Test(expected= RepertoireException.class)
    public void testGetTelPasTrouve() throws RepertoireException{
        rep.getTel("Julie");
    }}
```

@Before @BeforeClass @After @AfterClass

Annote une fonction à appeler …

@Before	avant chaque test
@After	après chaque test
@BeforeClass	une fois avant le premier test
@AfterClass	une fois après le dernier test

Tests paramétrés

Objectif
Pouvoir faire des tests avec de nombreux jeux de valeurs

chaineInit	taille	cote	result
"Bon"	5	Cote.GAUCHE	"Bon**"
"Bon"	5	Cote.DROITE	"**Bon"
"Bonjour"	3	Cote.GAUCHE	"Bon"
"Bonjour"	3	Cote.DROITE	"Bon"

```
assertEquals(result,UtilCadrage.cadrer(chaineInit,taille,cote));
```

Tests paramétrés : (1)

```
// annotation pour indiquer que c'est une classe de tests paramétrés
@RunWith(Parameterized.class)

public class UtilCadrageTestParametre {
    private String chaineInit; private int taille; private Cote cote; private String result;
```

Une méthode de test avec paramètres

```
    @Test
    public final void testCadrer( ) {
        assertEquals(result,UtilCadrage.cadrer(chaineInit,taille,cote));
    }
```

… avec paramètres initialisés dans le constructeurs

```
    public UtilCadrageTestParametre(String chaineInit, int taille, Cote cote, String result) {
        this.chaineInit = chaineInit; this.taille = taille; this.cote = cote; this.result = result;
    }
```

Tests paramétrés : le principe

Méthode rendant une collection dont chaque élément est un tableau de valeurs pour un test

```
@Parameters

public static Collection<Object[]> gestTests(){
        List<Object[ ]> list = new ArrayList<>( );
        list.add(new Object[ ]{"Bon", 5, Cote.GAUCHE,"Bon**"});
        list.add(new Object[ ]{"Bon", 5, Cote.DROITE,"**Bon"});
        list.add(new Object[ ]{"Bonjour", 3, Cote.GAUCHE,"Bon"});
        list.add(new Object[ ]{"Bonjour", 3, Cote.DROITE,"Bon"});
        return list;
    }
}
```

Tests paramétrés : la classe en un slide

```
@RunWith(Parameterized.class)
public class UtilCadrageTestParametre {
    private String chaineInit; private int taille; private Cote cote; private String result;

    public UtilCadrageTestParametre(String chaineInit, int taille, Cote cote, String result) {
        this.chaineInit = chaineInit; this.taille = taille; this.cote = cote; this.result = result;
    }
    @Parameters
     public static Collection<Object[]> gestTests(){
        List<Object[ ]> list = new ArrayList<>( );
        list.add(new Object[ ]{"Bon", 5, Cote.GAUCHE,"Bon**"});
        list.add(new Object[ ]{"Bon", 5, Cote.DROITE,"**Bon"});
        list.add(new Object[ ]{"Bonjour", 3, Cote.GAUCHE,"Bon"});
        list.add(new Object[ ]{"Bonjour", 3, Cote.DROITE,"Bon"});
        return list;
     }
    @Test
    public final void testCadrer( ) {
        assertEquals(result,UtilCadrage.cadrer(chaineInit,taille,cote));
    }
}
```

La méthode fail

```
// code simplifié de org.junit.Assert

static public void assertEquals(Object expected, Object actual) {
      if (expected==null)
          return actual ==null;
      else if ( ! expected.equals(actual) )
          fail( );
       }

static public void fail( ) {
          throw new AssertionError(message);
   }
```

D'autres méthodes de la classe org.junit.Assert

assertTrue
assertFalse

assertEquals
assertNotEquals

assertArrayEquals

assertNull
assertNotNull

assertSame (operator ==)
assertNotSame (operator !=)

fail