

# Programmation Avancée et Application

## Tests Unitaires - JUnit

---

Jean-Guy Mailly

`jean-guy.mailly@u-paris.fr`

LIPADE - Université de Paris

<http://www.math-info.univ-paris5.fr/~jmailly/>

## 1. Introduction

## 2. Les bases de JUnit

Débuter avec JUnit

Différents types de tests

# Introduction

---

- Lorsqu'on a défini une méthode, avant de l'utiliser, on souhaite s'assurer de son fonctionnement
- Idée naive : on l'applique avec différents paramètres, et on affiche le résultat pour s'assurer que c'est correct

## Tester son code : Exemple

```
public class Test {  
    public static int maMethode(int x, int y){  
        // Du code ici...  
    }  
  
    public static void main(String [] args){  
        System.out.println("maMethode(0,0) _=_"  
                           + maMethode(0,0));  
        System.out.println("maMethode(0,1) _=_"  
                           + maMethode(0,1));  
        System.out.println("maMethode(1,0) _=_"  
                           + maMethode(1,0));  
        System.out.println("maMethode(5,10) _=_"  
                           + maMethode(5,10));  
        // ...  
    }  
}
```

- On connaît à l'avance le résultat attendu pour un test
  - on sait que `maMethode(0,0)` doit retourner 10
  - l'affichage donne `maMethode(0,0) = 0`
  - Erreur détectée
- Les exemples testés doivent être suffisamment représentatifs des situations que la méthode peut rencontrer
- Si les tests sont bien écrits, et que la méthode passe tous les tests avec succès, on considère que la méthode est correcte

- Lors de mises à jour du code, on s'assure que les tests qui étaient satisfaits par la version  $n$  sont toujours satisfaits par la version  $n + 1$ 
  - Pas de perte de fonctionnalités
  - Pas d'ajout de bugs

- Les tests sont complémentaires de la documentation du code
- Ils fournissent des exemples de comment utiliser les différentes méthodes



# Développement dirigé par les tests (TDD)

TDD : *Test driven development*

1. Phase de conception : déterminer quelles seront les classes, les méthodes, etc
  - On connaît les paramètres et le type de retour, mais le code des méthodes n'est pas encore écrit
2. Écriture des tests pour la méthode `maMethode`
3. Écriture de la méthode `maMethode`
4. Si certains tests ne sont pas passés avec succès :
  - Corriger la méthode jusqu'à ce que tous les tests passent
  - Sinon
    - Retour à l'étape 2 pour une nouvelle méthode

## Brève liste de frameworks

- Cppunit pour C++
- CUnit pour C
- OUnit pour OCaml
- OUnit pour Objective C
- PHPUnit, SimpleTest et Atoum pour PHP
- unittest et PyUnit pour Python
- ...

Dans ce cours : JUnit 5

**Attention** : si vous cherchez des exemples/tuto/... sur Internet, vous risquez de trouver des choses correspondant à d'anciennes versions de JUnit, qui ne sont plus forcément valables avec JUnit 5

**Documentation officielle :**

<https://junit.org/junit5/docs/current/user-guide/>

# Les bases de JUnit

---

## Un premier exemple

Une méthode qui crée un String à partir d'un caractère et de son nombre d'occurrences :

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

## Un premier exemple

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class UtilTest {

    @Test
    public void testGetStringFromChar(){
        assertEquals("ffffff",
            Util.getStringFromChar(5, 'f'));
    }
}
```

- Lors de l'exécution de ce test, la méthode `assertEquals` vérifie si son premier paramètre (le résultat attendu) est égal à son second paramètre (le résultat de la méthode implémentée)
  - Si c'est le cas, le test est un succès
  - Sinon, une `org.opentest4j.AssertionFailedError` est lancée, le test a échoué

# Exécution du test dans Eclipse

The screenshot displays the Eclipse IDE interface during a JUnit test execution. On the left, the Package Explorer shows the project structure with 'UtilTest' selected. Below it, the Test Runner window indicates the test 'testGetStringFromChar()' failed. The main editor on the right shows the source code of 'UtilTest.java'.

Package Explorer: JUnit

Finished after 0,103 seconds

Runs: 1/1 Errors: 0 Failures: 1

UtilTest [Runner: JUnit 5] (0,003 s)

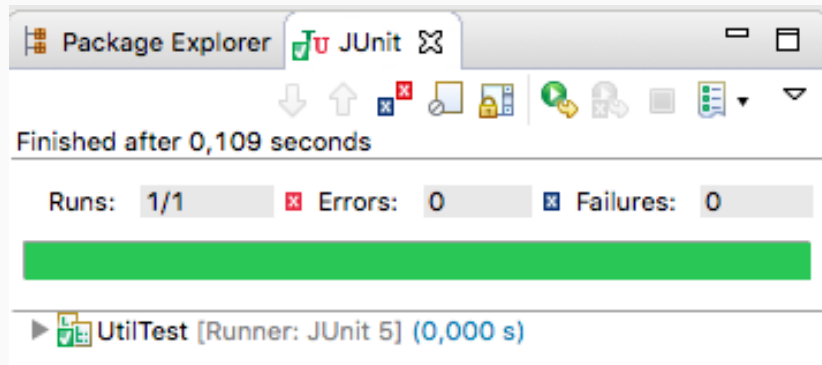
testGetStringFromChar() (0,003 s)

```
1 package tests_unitaires;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 class UtilTest {
8
9     @Test
10     void testGetStringFromChar() {
11         assertEquals("fffff", Util.getStringFromChar(5, 'f'));
12     }
13
14 }
```

## Un premier exemple

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

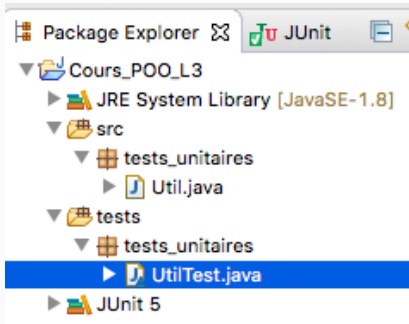
## Exécution du test dans Eclipse





# Description du fonctionnement de JUnit 5

- Un test unitaire : une méthode « spéciale » qui utilise l'API JUnit 5
- Les tests unitaires qui sont liés sont regroupés dans une même classe
- Une classe de tests correspond à une classe ou une méthode de `src`
- Les classes de tests sont regroupées dans répertoire `tests` situé au même niveau que le répertoire `src`
- Le répertoire `tests` est composé des mêmes packages que `src`
- Une méthode dans `src` → au moins un test



# Description d'un test

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class UtilTest {
    @Test
    public void testGetStringFromChar(){
        assertEquals("ffffff", Util.getStringFromChar(5, 'f'));
    }
}
```

- Un test est précédé par l'annotation @Test
- Chaque test fait appel à une (des) méthode(s) statique(s) de la classe org.junit.jupiter.api.Assertions
- chaque méthode assertXXXXXX vérifie si une certaine condition est vraie (le test est un succès) ou pas (le test échoue)

Documentation officielle :

[junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html](https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html)

D'après la documentation,

- *Assertions is a collection of utility methods that support asserting conditions in tests.*
- *Unless otherwise noted, a failed assertion will throw an `AssertionFailedError` or a subclass thereof.*

## Quelques méthodes usuelles : assertEquals


```
public static void assertEquals(Object expected ,  
                                Object actual)
```


Asserts that expected and actual are equal.

If both are **null**, they are considered equal.

### Exemple

```
assertEquals(" fffff", Util.getStringFromChar(5, 'f'));
```

 Failure Trace

 org.opentest4j.AssertionFailedError: expected: <fffff> but was: <01234>

## Quelques méthodes usuelles : `assertTrue` et `assertFalse`

```
public static void assertTrue(boolean condition)
```

Asserts that the supplied condition is **true**.

```
public static void assertFalse(boolean condition)
```


Asserts that the supplied condition is not **true**.

## Exemple d'utilisation de assertTrue

```
public class Util {  
    public static boolean isEven(int nb) {  
        return (nb%2) == 1;  
    }  
}
```

```
class UtilTest {  
    @Test  
    void testIsEven() {  
        assertTrue(Util.isEven(4));  
    }  
}
```

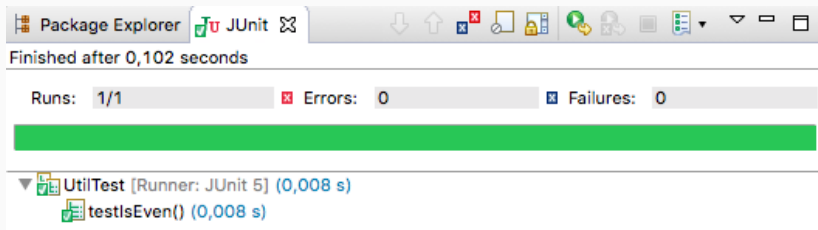
### Failure Trace

 org.opentest4j.AssertionFailedError: expected: <true> but was: <false>

## Exemple d'utilisation de assertTrue

On corrige la méthode isEven :

```
public class Util {  
    public static boolean isEven(int nb) {  
        return (nb%2) == 0;  
    }  
}
```



## Quelques méthodes usuelles : `assertThrows` et `assertDoesNotThrow`

```
public static <T extends Throwable> T assertThrows(  
    Class<T> expectedType, Executable executable)
```

Asserts that execution of the supplied executable **throws** an exception of the `expectedType` and returns the exception.

If no exception is thrown, or **if** an exception of a different type is thrown, **this** method will fail.

```
public static void assertDoesNotThrow(  
    Executable executable)
```

Asserts that execution of the supplied executable does not **throw** any kind of exception.



## Exemple d'utilisation de assertThrows

On modifie le code existant :

```
public class Util {  
    public static String getStringFromChar(int nb, char c)  
    {  
        if (nb < 0)  
            throw new IllegalArgumentException(  
                "Negative_value_" + nb + "_is_forbidden.");  
        StringBuffer sb = new StringBuffer();  
        for (int i = 0 ; i < nb ; i++)  
            sb.append(c);  
        return sb.toString();  
    }  
}
```

## Exemple d'utilisation de assertThrows

```
public class UtilTest {  
    @Test  
    void testNegativeValue() {  
        assertThrows(IllegalArgumentException.class,  
            () -> {  
                Util. getStringFromChar(-1, 'f');  
            });  
    }  
}
```

- Le test est un succès si une `IllegalArgumentException` est levée

## Quelques (autres) méthodes usuelles

```
public static void assertEquals(Object unexpected ,  
                                Object actual)
```

Asserts that the expected and actual are not equal.

```
public static void assertNull(Object actual)
```

Asserts that actual is **null**.

```
public static void assertNotNull(Object actual)
```

Asserts that actual is not **null**.

D'autres méthodes (et des surcharges de celles-ci) existent. Voir la documentation pour plus de détails.

# Méthode de tri de liste

```
package tests_unitaires;  
import java.util.List;  
  
public class ListUtil {  
    public static List<Integer> sort(  
        List<Integer> list){  
        // TO DO  
    }  
}
```

- Tout ce qu'on sait sur la méthode `sort`, c'est qu'elle prend en paramètre une `List<Integer>` et qu'elle retourne un objet du même type
- Avant de l'implémenter, on réfléchit à tous les cas représentatifs d'exécution de cette méthode
  - un cas représentatif = un test

## Premier test : tri d'une liste vide

*// Ne pas oublier les instructions package, import*

```
class ListUtilTest {  
    @Test  
    void testEmptyList() {  
        List<Integer> list = ListUtil.sort(  
            new ArrayList<Integer>());  
        assertTrue(list.isEmpty());  
    }  
}
```

- Premier cas particulier : si la liste initiale est vide, la liste retournée doit être vide aussi

## Deuxième test : tri d'une liste déjà triée

```
class ListUtilTest {  
    @Test  
    void testAlreadySortedList() {  
        List<Integer> list1 = new ArrayList<Integer>() ;  
        List<Integer> list2 = new ArrayList<Integer>() ;  
        list1.add(1); list1.add(3);  
        list1.add(5); list1.add(12);  
        list2.add(1); list2.add(3);  
        list2.add(5); list2.add(12);  
        assertEquals(list2 , ListUtil.sort(list1));  
    }  
}
```

- Deuxième cas particulier : si la liste initiale est déjà triée, la liste retournée est identique

## Troisième test : tri d'une liste « inversée »

```
class ListUtilTest {  
    @Test  
    void testReverseOrder() {  
        List<Integer> list1 = new ArrayList<Integer>() ;  
        List<Integer> list2 = new ArrayList<Integer>() ;  
        list1.add(12); list1.add(5);  
        list1.add(3); list1.add(1);  
        list2.add(1); list2.add(3);  
        list2.add(5); list2.add(12);  
        assertEquals(list2 , ListUtil.sort(list1 ));  
    }  
}
```

- Troisième cas particulier : si la liste initiale est triée dans l'ordre inverse, la liste retournée est l'inverse de la liste initiale

## Quelques cas représentatifs

On définit à présent quelques tests pour représenter des cas « normaux »

```
class ListUtilTest {  
    @Test  
    void testSomeList_01() {  
        List<Integer> list1 = new ArrayList<Integer>() ;  
        List<Integer> list2 = new ArrayList<Integer>() ;  
        list1.add(1); list1.add(5);  
        list1.add(12); list1.add(3);  
        list2.add(1); list2.add(3);  
        list2.add(5); list2.add(12);  
        assertEquals(list2 , ListUtil.sort(list1 ));  
    }  
}
```



## Quelques cas représentatifs

```
class ListUtilTest {  
    @Test  
    void testSomeList_02() {  
        List<Integer> list1 = new ArrayList<Integer>() ;  
        List<Integer> list2 = new ArrayList<Integer>() ;  
        list1.add(8); list1.add(6);  
        list1.add(2); list1.add(5);  
        list2.add(2); list2.add(5);  
        list2.add(6); list2.add(8);  
        assertEquals(list2 , ListUtil.sort(list1 ));  
    }  
}
```

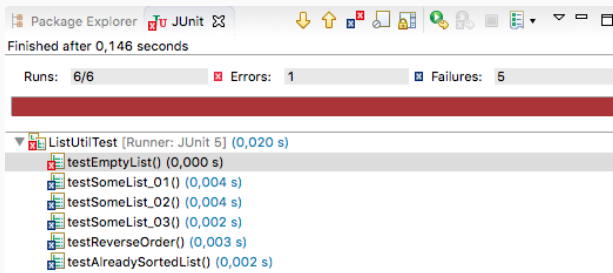
## Quelques cas représentatifs

```
class ListUtilTest {  
    @Test  
    void testSomeList_03() {  
        List<Integer> list1 = new ArrayList<Integer>() ;  
        List<Integer> list2 = new ArrayList<Integer>() ;  
        list1.add(8); list1.add(6);  
        list1.add(2); list1.add(5);  
        list1.add(15); list1.add(11);  
        list2.add(2); list2.add(5);  
        list2.add(6); list2.add(8);  
        list2.add(11); list2.add(15);  
        assertEquals(list2 , ListUtil.sort(list1 ));  
    }  
}
```

# Première exécution des tests

```
public class ListUtil {  
    public static List<Integer> sort(List<Integer> list){  
        return null ;  
    }  
}
```

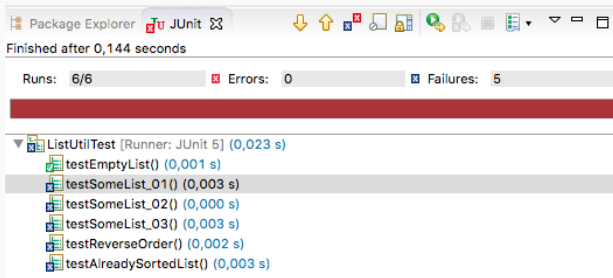
- On exécute les tests avec la méthode sort qui ne fait « rien »



# Première correction du code

```
public class ListUtil {  
    public static List<Integer> sort(List<Integer> list){  
        return new ArrayList<Integer>() ;  
    }  
}
```

- On exécute les tests avec la méthode sort qui retourne une liste vide

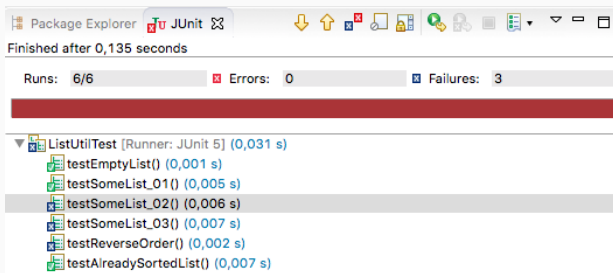


## Deuxième correction : code « presque » correct

```
public static List<Integer> sort(List<Integer> list){
    for(int i = 1 ; i < list.size() - 1 ; i++) {
        int min = i ;
        for(int j = i + 1 ; j < list.size() ; j++)
            if(list.get(j) < list.get(min)) min = j ;
        if(min != i) {
            int tmp = list.get(i);
            list.set(i, list.get(min));
            list.set(min, tmp);
        }
    }
    return list ;
}
```

## Deuxième correction : code « presque » correct

- La méthode est incorrecte : certains tests passent, mais pas tous

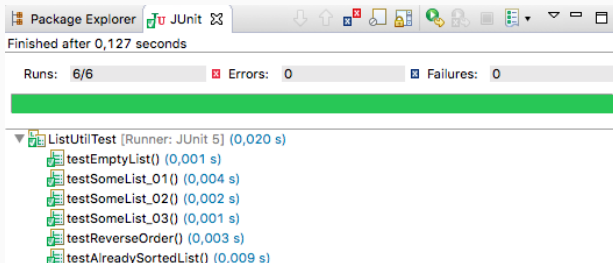


## Correction du code

```
public static List<Integer> sort(List<Integer> list){
    for(int i = 0 ; i < list.size() - 1 ; i++) {
        int min = i ;
        for(int j = i + 1 ; j < list.size() ; j++)
            if(list.get(j) < list.get(min)) min = j ;
        if(min != i) {
            int tmp = list.get(i);
            list.set(i, list.get(min));
            list.set(min, tmp);
        }
    }
    return list ;
}
```

# Correction du code

- La méthode est correcte : tous les tests passent





**Exécution de code avant ou  
après les tests**

---

- En plus des méthodes de tests, d'autres méthodes peuvent être définies dans les classes de tests
- On peut exiger que certaines méthodes soient exécutées avant ou après l'exécution de **l'ensemble des tests**
- On peut exiger que certaines méthodes soient exécutées avant ou après l'exécution de **chaque test**

# Éviter les redondances : méthodes d'initialisations

On parle des méthodes de *setup*

@BeforeAll

```
static void beforeAll() {  
    System.out.println("Before_all_test_methods");  
}
```

@BeforeEach

```
void beforeEach() {  
    System.out.println("Before_each_test_method");  
}
```

**Attention** : la méthode annotée @BeforeAll doit être static

# Éviter les redondances : méthodes de destruction

On parle des méthodes de *teardown*

@AfterEach

```
void afterEach() {  
    System.out.println("After_each_test_method");  
}
```

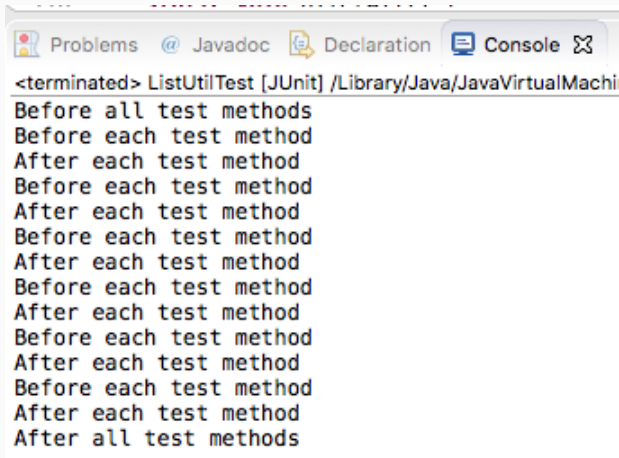
@AfterAll

```
static void afterAll() {  
    System.out.println("After_all_test_methods");  
}
```

**Attention** : la méthode annotée @AfterAll doit être **static**

## Effet des méthodes setup et teardown

- On ajoute les quatre méthodes vues précédemment à la classe ListUtilTest



```
<terminated> ListUtilTest [JUnit] /Library/Java/JavaVirtualMachi
Before all test methods
Before each test method
After each test method
Before each test method
After each test method
Before each test method
After each test method
Before each test method
After each test method
Before each test method
After each test method
Before each test method
After each test method
After all test methods
```

- Initialiser des objets qui vont être utilisés dans plusieurs tests
- Ouvrir et fermer des flux d'entrée/sortie (fichier, réseau, base de données, ...)
- Exécuter ou tuer un processus/thread
- ...

## Un exemple un peu plus concret : setUp de ListUtilTest

```
class ListUtilTest {  
    private List<Integer> list1;  
    private List<Integer> list2;  
    private List<Integer> list3;  
  
    @BeforeEach  
    void beforeEach() {  
        list1 = new ArrayList<Integer>();  
        list2 = new ArrayList<Integer>();  
        list3 = new ArrayList<Integer>();  
        list2.add(1);  
        list2.add(3);  
        list2.add(5);  
        list2.add(12);  
    }  
}
```

- Avant chaque test, on instancie les objets, et list2 est initialisée

## Un exemple un peu plus concret : setUp de ListUtilTest

- Les tests qui utilisent la même list2 sont plus concis. Par exemple :

```
@Test
void testAlreadySortedList() {
    list1.add(1);
    list1.add(3);
    list1.add(5);
    list1.add(12);
    assertEquals(list2, ListUtil.sort(list1));
}
```

- C'est similaire pour testReverseOrder et testSomeList\_01



## Un exemple un peu plus concret : setUp de ListUtilTest

- Les tests qui utilisaient une `list2` différente sont ré-écrits avec `list3`. Par exemple :

@Test

```
void testSomeList_02 () {  
    list1.add(8);  
    list1.add(6);  
    list1.add(2);  
    list1.add(5);  
    list3.add(2);  
    list3.add(5);  
    list3.add(6);  
    list3.add(8);  
    assertEquals(list3 , ListUtil.sort(list1));  
}
```

- C'est similaire pour `testSomeList_03`

# Tests paramétrés

---

# Principe des tests paramétrés

- Pour effectuer plusieurs tests similaires sur des valeurs différentes, deux options :
  - écrire à la main chacun de ces tests (`testSomeList_01`, `testSomeList_02`, `testSomeList_03`,...) ✖
  - écrire une seule fois un test, en lui indiquant de s'exécuter plusieurs fois en prenant ses paramètres dans un ensemble pré-défini ✔
- JUnit 5 fournit plusieurs façons de faire des tests paramétrés

## Liste de valeurs : Exemple des palindromes

- On suppose que `isPalindrome` est une méthode qui prend en paramètre un `String`, et retourne `true` si et seulement si le paramètre est un palindrome

```
@ParameterizedTest
```

```
@ValueSource(strings = {"bob", "", "b",  
                        "bonjourruojnob"})
```

```
void testPalindrome(String word) {  
    assertTrue(isPalindrome(word));  
}
```

```
@ParameterizedTest
```

```
@ValueSource(strings = {"bonjour", "toto", "descartes"})
```

```
void testNotPalindrome(String word) {  
    assertFalse(isPalindrome(word));  
}
```

# Liste de valeurs : Exemple des palindromes

- On implémente une méthode `isPalindrome` qui retourne toujours `false`

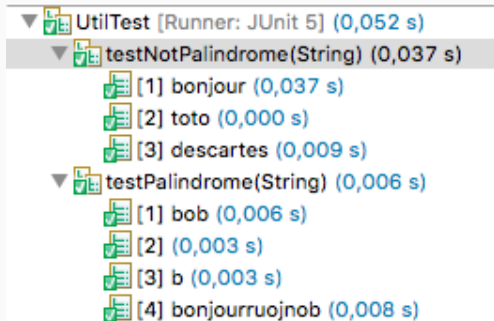
```
public static boolean isPalindrome(String word) {  
    return false;  
}
```

- Exécution des tests :

The screenshot displays a JUnit test runner window. At the top, a summary bar shows 'Runs: 7/7', 'Errors: 0', and 'Failures: 4'. Below this, a red progress bar is visible. The main area shows a tree view of test results. The root is 'UtilTest [Runner: JUnit 5] (0,055 s)'. It contains two test methods: 'testNotPalindrome(String) (0,032 s)' and 'testPalindrome(String) (0,008 s)'. 'testNotPalindrome' has three sub-items, all marked with green checkmarks: '[1] bonjour (0,032 s)', '[2] toto (0,006 s)', and '[3] descartes (0,008 s)'. 'testPalindrome' has four sub-items, all marked with blue 'X' icons indicating failure: '[1] bob (0,008 s)', '[2] (0,001 s)', '[3] b (0,003 s)', and '[4] bonjourruojnob (0,005 s)'.

## Liste de valeurs : Exemple des palindromes

```
public static boolean isPalindrome(String word) {  
    if(word.isEmpty()) return true ;  
    for (int i = 0; i <= word.length() / 2; i++) {  
        int j = word.length() - i - 1;  
        if (word.charAt(i) != word.charAt(j)) return false ;  
    }  
    return true ;  
}
```



## Liste de valeurs : Exemple des nombres pairs

- On reprend la méthode `isEven` vue précédemment

```
@ParameterizedTest
@ValueSource(ints = {0, 2, 4, 8, 12, 42, 666})
void testIsEven(int nb) {
    assertTrue(Util.isEven(nb));
}
```

```
@ParameterizedTest
@ValueSource(ints = {1, 3, 5, 7, 13, 55, 777})
void testIsNotEven(int nb) {
    assertFalse(Util.isEven(nb));
}
```

# Liste de valeurs : Exemple des nombres pairs

Runs: 14/14

Errors: 0

Failures: 0

▼ UtilTest [Runner: JUnit 5] (0,097 s)

▼ testIsNotEven(int) (0,051 s)

✓ [1] 1 (0,051 s)

✓ [2] 3 (0,007 s)

✓ [3] 5 (0,011 s)

✓ [4] 7 (0,003 s)

✓ [5] 13 (0,005 s)

✓ [6] 55 (0,006 s)

✓ [7] 777 (0,006 s)

▼ testIsEven(int) (0,006 s)

✓ [1] 0 (0,006 s)

✓ [2] 2 (0,002 s)

✓ [3] 4 (0,005 s)

✓ [4] 8 (0,006 s)

✓ [5] 12 (0,005 s)

✓ [6] 42 (0,004 s)

✓ [7] 666 (0,012 s)



# Fonctionnement détaillé des listes de valeurs

- @ValueSource permet de spécifier un tableau de valeurs
- Chaque appel du test paramétré utilise un seul paramètre
- @ValueSource accepte les types suivants :
  - short
  - byte
  - int
  - long
  - float
  - double
  - char
  - java.lang.String
  - java.lang.Class

# Utilisation de valeurs séparées par des virgules (CSV)

- Il est possible d'utiliser un tableau de `String` pour entrer les paramètres, tel que chaque `String` représente un ensemble de paramètres séparés par des virgules

```
@ParameterizedTest
```

```
@CsvSource({ "toto", "1", "titi", "12", "tutu", "42" })
```

```
void testWithCsvSource(String first, int second) {  
    // TO DO  
}
```

- Le test sera exécuté trois fois, avec
  1. `first = "toto"` et `second = 1`
  2. `first = "titi"` et `second = 12`
  3. `first = "tutu"` et `second = 42`
- Une conversion automatique est faite en fonction des types attendus (`String` et `int`)

# Utilisation de valeurs séparées par des virgules (CSV)

- Il est possible d'utiliser des String plus complexes grâce au symbole de quote simple '

```
@ParameterizedTest
```

```
@CsvSource({"toto", "1", "titi", "12", "'tutu', 'tata'", "42"})
```

```
void testWithCsvSource(String first, int second) {
```

```
    // TO DO
```

```
}
```

- Le test sera exécuté trois fois, avec
  1. first = "toto" et second = 1
  2. first = "titi" et second = 12
  3. first = "tutu, tata" et second = 42
- Grâce au symbole ', la partie 'tutu, tata' est convertie en une seule String

## Test paramétré avec CSV pour le tri de listes

- On commence par écrire une méthode qui « convertit » une `String` en liste d'entiers

```
private List<Integer> stringToList(String str){  
    List<Integer> list = new ArrayList<Integer>();  
    String[] tab = str.split(",");  
    for(String s : tab) {  
        list.add(Integer.parseInt(s));  
    }  
    return list ;  
}
```

## Test paramétré avec CSV pour le tri de listes

- On peut maintenant remplacer tous les tests de `ListUtilTest` par un seul test paramétré

```
@ParameterizedTest
```

```
@CsvSource({ "'',_''", "'1,3,5,12',_ '1,3,5,12'",  
    "'1,3,5,12',_ '12,5,3,1'", "'1,3,5,12',_ '1,5,12,3'",  
    "'2,5,6,8',_ '8,6,2,5'",  
    "'2,5,6,8,11,15',_ '8,6,2,5,15,11'"}))
```

```
void testSortingList(String expectedResult ,  
                    String input) {  
    assertEquals(stringToList(expectedResult),  
        ListUtil.sort(stringToList(input)));  
}
```

## Test paramétré avec fichier source CSV

- On peut remplacer la déclaration des paramètres CsvSource par la lecture d'un fichier CSV, grâce à CsvFileSource

```
@ParameterizedTest
@CsvFileSource(resources = "~/tri_listes.csv",
    numLinesToSkip = 1)

void testSortingList(String expectedResult,
                    String input) {
    assertEquals(stringToList(expectedResult),
        ListUtil.sort(stringToList(input)));
}
```

# Test paramétré avec fichier source CSV

- Le contenu du fichier `tri_listes.csv` est le suivant :

Expected\_Output Input

"", ""

"1,3,5,12", "1,3,5,12",

"1,3,5,12", "12,5,3,1",

"1,3,5,12", "1,5,12,3",

"2,5,6,8", "8,6,2,5",

"2,5,6,8,11,15", "8,6,2,5,15,11"

- `numLinesToSkip = 1` indique qu'on doit ignorer la première ligne
- Dans ce cas, c'est le symbole `"` qui est utilisé au lieu de `'`
- Le fichier doit être placé dans le classpath du projet :








## Test paramétré avec fichier source CSV

Runs: 6/6

✖ Errors: 0

✖ Failures: 0

- ▼  ListUtilTest [Runner: JUnit 5] (0,000 s)
  - ▼  testSortingList(String, String) (0,000 s)
    -  [1] , (0,000 s)
    -  [2] 1,3,5,12, 1,3,5,12 (0,000 s)
    -  [3] 1,3,5,12, 12,5,3,1 (0,000 s)
    -  [4] 1,3,5,12, 1,5,12,3 (0,017 s)
    -  [5] 2,5,6,8, 8,6,2,5 (0,004 s)
    -  [6] 2,5,6,8,11,15, 8,6,2,5,15,11 (0,018 s)



# Utilisation de CSV pour créer des instances d'objets

- On a utilisé dans l'exemple précédent les données CSV pour initialiser des listes
- On peut également instancier d'autres objets en utilisant les données CSV comme paramètres du constructeur

## Exemple : Gestion d'une collection de CD

- On définit un CD par le titre, l'artiste, l'année de sortie, et le genre
- Le titre et l'artiste sont des `String`, l'année un `int`, et le genre est une constante `Enum`

```
public class CD {  
    private String titre ;  
    private String artiste ;  
    private int annee ;  
    private Genre genre ;  
    public CD(String titre , String artiste ,  
               int annee, Genre genre) {  
        this.titre = titre ;  
        this.artiste = artiste ;  
        this.annee = annee ;  
        this.genre = genre ;  
    }  
}
```

## Exemple : Gestion d'une collection de CD

```
public enum Genre {  
    CLASSIQUE, ELECTRO, METAL, POP, RAP, ROCK;  
}
```

## Exemple : Gestion d'une collection de CD

```
class TestCD {
    @ParameterizedTest
    @CsvSource({
        "'A_Night_at_the_Opera ','Queen','ROCK','1975'",
        "'Thriller ','Michael_Jackson ','POP','1982'",
        "'Discovery ','Daft_Punk ','ELECTRO','2001'",
        "'Reise ','Reise ','Rammstein ','METAL','2004'"
    })
    void testWithArgumentsAccessor(ArgumentsAccessor
                                   arguments) {
        CD album = new CD(arguments.getString(0),
                           arguments.getString(1),
                           arguments.get(2, Genre.class),
                           arguments.getInteger(3));
        // Do assertions
    }
}
```