# TP TESTS UNITAIRES JUNIT

**Numéro de Référence**
**#UNIVPM001**

(Document de 9 pages)


**Résumé**

Rapport sur les Travaux Pratiques de l'unité d'enseignement Fiabilité Logicielle du Master 1 Informatique portant sur les tests unitaires sur les thèmes de vérification de type triangle et de recherche d'éléments dans un tableau. Ce TP fut réalisé à l'aide du langage Java



**Mots Clés**

Fiabilité Logicielle Test Unitaire JUnite



Université d'Aix-Marseille
Master 1 Informatique

171 avenue de Luminy

13009 Marseille

## SECTION DES REDACTEURS

| Nom | Prénom | Contribution |
|---|---|---|
| Gonzalez | Sébastien | readData, typeTriangle, chercherElt, tests, rapport |
| Rouabhia | Younes | readData, typeTriangle, chercherElt, tests, rapport |

## CONTACTS

| Nom | Prénom | Email | Fonction |
|---|---|---|---|
| Gonzalez | Sébastien | Sebastien.gonzalez@etu.univ-amu.fr | Développeur,testeur, rédacteur |
| Rouabhia | Younes | Younes.rouabhia@etu.univamu.fr | Développeur,testeur, rédacteur |

## HISTORIQUE DES MODIFICATIONS

| Modifications | Date | Version | Approbateur de la diffusion |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

# TABLE DES MATIERES

1.Introduction ..... 4

2.Fonction 1 ..... 5

3.Fonction 2 ..... 6

4.Fonction 3 ..... 7

5.Fonction 4 ..... 8

6.Fonction 5 ..... 9

# 1. INTRODUCTION

Vous trouverez dans ce rapport le détail de la réalisation du deuxième TP de Fiabilité Logicielle du Semestre 2 du Master 1 Informatique de l'Université d'Aix-Marseille.

Il s'agit ici de tester cinq fonctions que l'on nous a fournis. Ces fonctions doivent chercher un élément dans un tableau (fournis en paramètre).

Nous avons sept fonctions de tests :

- testValeurExistante : qui vérifie de façon générale
- testValeurIndice0 : qui test la valeur présente à l'indice 0 du tableau
- testValeurIndiceDernier : qui test la valeur présente au dernier indice du tableau
- testValeurIndiceMulti : qui test une valeur qui est présente plusieurs fois dans le tableau
- testValeurInconnue : qui test une valeur qui n'est pas présente dans le tableau
- testTableauVide : qui test une valeur dans un tableau vide
- testTableauNull : qui test une valeur dans un tableau non initialisé

## 2. FONCTION 1

```
public boolean chercher1(int x, int []  tab){
		/*recherche dichotomique 1*/
		int i,j,m;
		int n = tab.length;
		i = 0; j= n - 1;
		while (i <= j) {
			m = (i + j) / 2;
			if (tab[m] < x){
				i = m + 1;
			}
			if (tab[m] > x) {
				j = m - 1;
			}
			if (tab[m] == x){
				return true;
			}
			if (i > n-1 || j < 0) {
				return false;
			}
		}
		return false;
	}
```

Après avoir effectué l'ensemble des tests sur cette méthode nous constatons que l'ensemble de ces derniers sont réussis hormis lorsque le tableau n'est pas initialisé. En effet la ligne « int n=tab.length » renvoie une erreur puisque le tableau n'existe pas et donc l'attribue lenght non plus. Il aurait fallu un test pour le vérifier en début de méthode.

## 3. FONCTION 2

```
public boolean chercher2(int x, int [] tab){
    /*recherche dichotomique 2*/
    int i,j,m;
    boolean found;
    i=1;
    j= tab.length;
    m=0;
    found=false;
    while (!(i==j && !found)) {
        m=(i+j)/2;
        if (tab[m]<x){
            i=m+1;
        }    else {
            if(tab[x]==m) {
                found=true;
            }    else {
                j=m-1;
            }
        }
    }
    return found;
}
```

Cette fois ci la majeure partie des tests ne passe pas avec succès. Deux d'entre eux (testValeurMulti et testValeurIndice0) ne termine pas. Par conséquent il faut bien faire attention à ajouté une valeur de timeout à chaque test (dépassé la valeur de timeout le test s'arrête).

Les autres tests qui ne passent pas échouent à cause d'un dépassement de taille du tableau. En effet la fonction essaye d'accéder à une case du tableau supérieur à la taille de celui-ci. Il faudrait donc vérifier au sein de la méthode si l'indice est correct avant de chercher à accéder à l'élément du tableau.

Le seul test qui réussi sans problème est celui qui test une valeur non présente dans le tableau.

## 4. FONCTION 3

```
public boolean chercher3(int x, int [] tab){
    /*recherche dichotomique 3*/
    int i,j,m;
    i = 1;
    j = tab.length;
    m=0;
    while (i!=j) {
        m=(i+j)/2;
        if (tab[m]<=x) i=m;
        else j=m;
    }
    return (x==tab[m]);
}
```

Cette fonction ne passe aucun de nos tests. Le test « testValeurIndice0 » renvoie même une réponse erronée.

Le reste ne passe pas à cause du temps précédemment hormis le tableau null et le tableau vide qui renvoie respectivement une NullPointerException et un ArrayIndexOutOfBoundException puisque aucune vérification sur la validité du tableau n'est faite.

## 5.FONCTION 4

```
public boolean chercher4(int x, int [] tab){
    /*recherche dichotomique 4*/
    int i,j,m;
    i=1;
    j = tab.length;
    m=0;
    boolean trouve = false;
    while (i!=j) {
        m=(i+j)/2;
        if (tab[m]<=x) i=m;
        if (tab[m] == x) trouve = true;
        else j=m;
    }
    return trouve;
}
```

Nous constations que nous obtenons les mêmes résultats que pour la fonction précédente.

La seule différence entre ces deux méthodes est l'utilisation du boolean « trouve ». Nous pouvons donc en conclure que l'utilisation de celui-ci est inutile à l'algorithme.

# 6.FONCTION 5

```
public boolean chercher5(int x, int [] tab){
    /*recherche dichotomique 5*/
    int i,j,m;
    boolean trouve;
    i=0;
    j=tab.length - 1;
    m=0;
    trouve=false;
    while ((i<=j)&&(!trouve) ){
        m=(i+j)/2;
        if (x==tab[m]){
            trouve=true;
        }
        else{ if (x < tab[m]){
            j=m-1;
        } else {
            i=m+1;
        }
        }
    }
    return trouve;
}
```

Pour cette fonction nous obtenons cette fois-ci les mêmes résultats que la fonction 1.