



Derniere conclusion sur la partie Complex

La classe Complex du premier rendu se trouve dans le dossier des CR-TP2.

Dans le dernier rendu ont été déposés la classe **Complex** corrigée ainsi que la classe de test corrigée et remplie.

Lorsqu'on lance le fichier html résumant les tests:

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Complex		99 %		83 %	2	14	1	20	1	11	0	1
Total	1 of 124	99 %	1 of 6	83 %	2	14	1	20	1	11	0	1

on remarque assez bien que toutes les instructions ont été parcourues. La seule instruction qui fait qu'on est à 99% est:

```
public static void infinite()
{
    while (true);
}
```

Et notre test sur cette partie est conçu pour sortir une erreur. Donc c'est tout à fait normal.

Concernant les "Missed Branches", il s'agit d'une branche conditionnelle à ce niveau:

```
public boolean isZero() {
    return ((Math.abs(this.realPart) <= EPSILON) && (Math.abs(this.imaginaryPart) <= EPSILON));
}

public float getRealPart() {
```

Malheureusement mes tests n'ont réussi qu'à parcourir que 3 branches sur 4. J'ai cherché mais j'ai pas trouvé comment parcourir la dernière.

Du coup, au final j'ai un taux de couverture de 100% en instruction et 83% en branches (conditions).

Files à double extrémité:

La classe de test **ArrayDoubleEndedQueueTest** contient des tests de toutes les méthodes de la classe **ArrayDoubleEndedQueue**. Et permet de parcourir toutes les instructions et branches possibles à parcourir.

C'est ainsi que ces classes ont été créées:

- **testAddFirst():**

Ajoute plusieurs éléments en début de file à l'aide de `addFirst`.

Vérifie que le premier élément de la file correspond à la valeur attendue.

- **testAddLast():**

Ajoute plusieurs éléments en fin de file à l'aide de `addLast`.

Vérifie que le dernier élément de la file correspond à la valeur attendue.

- **testRemoveFirst():**

Ajoute deux éléments en fin de file.

Utilise `removeFirst` pour supprimer le premier élément.

Vérifie que la valeur supprimée correspond à celle attendue.

- **testRemoveLast():**

Ajoute plusieurs éléments en début et en fin de file.

Utilise `removeLast` pour supprimer le dernier élément.

Vérifie que la valeur supprimée correspond à celle attendue.

- **testGetFirst():**

Ajoute un élément en fin de file.

Utilise `getFirst` pour obtenir le premier élément.

Vérifie que la valeur obtenue correspond à celle attendue.

- **testGetLast():**

Ajoute un élément en fin de file.

Utilise `getLast` pour obtenir le dernier élément.

Vérifie que la valeur obtenue correspond à celle attendue.

- **testSize():**

Ajoute plusieurs éléments en fin de file.

Utilise size pour obtenir la taille de la file.

Vérifie que la taille correspond à celle attendue.



- **testContains():**

Ajoute plusieurs éléments en fin de file.















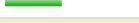

Utilise contains pour vérifier la présence ou l'absence d'éléments.

Vérifie que les résultats de contains sont conformes aux attentes, y compris pour le cas d'éléments null.

Voici le taux de couverture de la classe **ArrayDoubleEndedQueue**:
arraydoubleendedqueue

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
ArrayDoubleEndedQueue		86 %		72 %	6 20	6 45	0 9	0 1
Total	30 of 224	86 %	6 of 22	72 %	6 20	6 45	0 9	0 1

ArrayDoubleEndedQueue

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
removeFirst()		86 %		50 %	1 2	1 7	0 1
removeLast()		86 %		50 %	1 2	1 7	0 1
addFirst(Object)		78 %		50 %	1 2	1 5	0 1
addLast(Object)		78 %		50 %	1 2	1 5	0 1
getLast()		77 %		50 %	1 2	1 3	0 1
getFirst()		64 %		50 %	1 2	1 3	0 1
contains(Object)		100 %		100 %	0 6	0 8	0 1
ArrayDoubleEndedQueue(int)		100 %		n/a	0 1	0 6	0 1
size()		100 %		n/a	0 1	0 1	0 1
Total	30 of 224	86 %	6 of 22	72 %	6 20	6 45	0 9

Les couleurs **rouge** et **jaune** dans le fichier de test html sont tout à fait normales puisqu'il s'agit d'instruction qu'on parcourait en cas d'erreur. C'est la raison pour laquelle, **ne pas parcourir ces instructions ou branches est un choix volontaire.**

```
@Override
public void addFirst(Object o) {
    if(size == capacity) {
        throw new IllegalStateException("La queue est pleine");
    }
    else {
        elements[size] = o;
        size++;
    }
}
```

On est donc à un maximum de taux de couverture qui est de **86% en instructions parcourues** et de **72% en branches (conditions) parcourues** . Il est important de rappeler que les instructions et branches non parcourues sont des instructions qui mènent à des erreurs de tests. Dans le cas contraires on serait à **100% de taux de couverture globale**.