Emilien SANCHEZ 22105900
Robin Pujol 22104635
Le 12/02/2024, à Toulouse

Rapport de TP

ILU 4. Test

Répartition du travail :

Commun :
- Spécification des exigences
- Programmation

Robin :
- Phase de tests aléatoire
- Script .jar
- Tests de mutation

Emilien :
- Rapport du TP
- Couverture des exigences
- Tests fonctionnels

Choix du langage JAVA avec l'outil Eclipse pour la réalisation du programme et de sa phase de tests.

Lien du git : https://github.com/emilien3/Anagramme

Spécification des exigences du programme :

Le programme doit générer un anagramme d'une saisi de l'utilisateur.

L'anagramme doit être construit à partir des lettres du mot, qui sont présentées par ordre alphabétique.

La saisie du mot « facile » doit afficher « acefil ».

L'algorithme à utiliser doit être un tri par insertion.

Le mot doit être géré comme un tableau de caractères.

Une boucle de lecture ne doit lire qu'un caractère à la fois.

Chaque caractères doit être rangé dans un tableau, tant que la fin de ligne n'est pas atteinte.

Le tri ne doit utiliser qu'un seul tableau (c'est un tri sur place).

Tests coverage sur le mot mouton :

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Anagramme	14.1 %	104	634	738
test	0.0 %	0	631	631
modele	0.0 %	0	631	631
TriInsertionMutant.java	0.0 %	0	397	397
TriInsertionTest.java	0.0 %	0	168	168
MainTest.java	0.0 %	0	66	66
src	97.2 %	104	3	107
modele	97.2 %	104	3	107
TriInsertion.java	97.2 %	104	3	107
TriInsertion	97.2 %	104	3	107
anagrammeLu()	75.0 %	9	3	12
main(String[])	100.0 %	7	0	7
anagramme(String)	100.0 %	24	0	24
lecture()	100.0 %	15	0	15
retireAccent(String)	100.0 %	9	0	9
traitement(char[], int)	100.0 %	37	0	37

Analyse des résultats du coverage sur le mot mouton:

Il est normal de n'obtenir que 14,1% pour plusieurs raisons :
- Aucune des méthodes de tests ne sont utilisées, étant donné que c'est une execution de la classe TriInsertion.
- Nous pouvons observer un 75% dans la méthode anagrammeLu(), ce chiffre est dû au fait que l'on ne rentre pas dans le catch :

```
public void anagrammeLu() {
    String text;
    try {
        text = lecture();
        anagramme(text);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Les instructions permettant d'obtenir l'anagramme sont :

MainTest.java | test.txt | TriInsertion.java

```
1  package modele;
2
3  import java.io.BufferedReader;
7
8  public class TriInsertion {
9
10     public void traitement(char[] anagramme, int taille_anagramme) {
11         for (int i = 0; i < taille_anagramme; i++) {
12             char car = anagramme[i];
13             int j = i;
14             while ((j > 0) && (car < anagramme[j - 1])) {
15                 anagramme[j] = anagramme[j - 1];
16                 j--;
17             }
18             anagramme[j] = car;
19         }
20     }
21
22     public String lecture() throws IOException {
23         InputStreamReader isr = new InputStreamReader(System.in);
24         BufferedReader br = new BufferedReader(isr);
25         String text = br.readLine();
26         return text;
27     }
28
29     public void anagrammeLu() {
30         String text;
31         try {
32             text = lecture();
33             anagramme(text);
34         } catch (IOException e) {
35             e.printStackTrace();
36         }
37     }
38
39     public String retireAccent(String text) {
40         String res = Normalizer.normalize(text, Normalizer.Form.NFD).replaceAll("\\p{InCombiningDiacriticalMarks}+", "");
41         return res;
42     }
43
44     public String anagramme(String text) {
45         int taille_anagramme = text.length();
46         String textSansAccent = retireAccent(text);
47         String textSansMaj = textSansAccent.toLowerCase();
48         char[] anagramme = textSansMaj.toCharArray();
49         traitement(anagramme, taille_anagramme);
50         String res = new String(anagramme);
51         return res;
52
53     }
54
55     public static void main(String[] args) {
56         TriInsertion t = new TriInsertion();
57         t.anagrammeLu();
58     }
59
60 }
```

Script de test et phase de Test aléatoire

Nous avons commencé notre phase de tests avec des premiers tests aléatoire afin de nous assurer d'une première cohérence de nos résultats.

Nous avons ensuite passé notre phase de test aléatoire, fonctionnelle et de mutation sur le fichier test.txt afin de pouvoir automatiser la phase de test via un script .jar afin de pouvoir lancer les tests, sans devoir ouvrir Eclipse, via la commande suivante :

- java -jar testsAuto.jar pour pouvoir executer le .jar

Tests Fonctionnel:

- Classes et Tests :
    - Classes Valides: Entrées valides :
        - 1.1 : Mots contenant des lettres minuscules uniquement. *(decila utilisé)*
        - 1.2 : Mots contenant des lettres majuscules uniquement. *(DECILA utilisé)*
        - 1.3 : Mots contenant des lettres minuscules et majuscules. *(DeCiLa utilisé)*
    - Classes Invalides: Entrées invalides :
        - 1.1 : Chaîne vide. *(pas testé dans la phase de test automatique mais testé manuellement)*
        - 1.2 : Chaîne contenant des caractères non alphabétiques. *(!$%& utilisé)*
- Tests aux limites :
    - Limite 1 : Mot avec une seule lettre. *(a utilisé)*
    - Limite 2 : Mot avec le plus de lettres possibles. *(Integer.MAX_VALUE == 2 147 483 647 -> peu de chances d'atteindre un mot de cette taille, donc pas testé)*
    - Limite 3 : Mot avec toutes les lettres identiques. *(aaaa utilisé)*
    - Limite 4 : Mot avec toutes les lettres déjà ordonnées. *(abcd utilisé)*
- Extensions possibles :
    - Ajouter des tests pour des mots contenant des caractères spéciaux.
    - Tester des mots avec des lettres accentuées. *(Réalisé dans la phase de test aléatoire)*

Les extensions sont très facilement ajoutante du fait de l'utilisation d'un fichier .txt .

Tests de Mutation :

	Changement	Ligne (fichier TriInsertionMutant)	Zm!=Z	Zm=Z	Commentaire
M1	&& → ||	22	Mot = Chat Zm = null Z = acht	N'existe pas	Nous avons un mutant très facile à tuer (tous les mots).
M2	< → <=	42	N'existe pas	Mot = Chat Zm = acht Z = acht	Nous avons un mutant très résistant il survit à tous les mots.
M3	– → ++	64	Mot = Chat Zm = null Z = acht	Mot = a Zm = a Z = a	Nous avons un mutant très facile à tuer (presque tous les mots).
M4	j + 1 → j - 1	82,83	Mot = Chat Zm = null Z = acht	Mot = a Zm = a Z = a	Nous avons un mutant très facile à tuer (presque tous les mots).