Derramond Lylian 16/06/2023

Groupe 2A

Rapport d’analyse SAE 2.02

Table des matières

Outils d’évaluations :

Tests :

Pour vérifier le bon fonctionnement des algorithmes, j'ai effectué une série de tests visant à détecter les erreurs, valider les comportements attendus et vérifier que le code satisfait aux exigences spécifiées.

Test 1 (1 mot):

```
@Test
public void test1mot() {
    // Chaine à 1 mot
    assertEquals("Erreur de chaine à 1 mot", List.of("OK"), testExo.solution("OK",List.of('a', 'b', 'c', 'd', 'e')));
}
```

Test 2 :

```
@Test
public void testExemple() {
    // Chaine donnée en exemple
    assertEquals(List.of("fait", "Il", "aujourd", "aout", "beau", "hui", "comme", "en"), testExo.solution("Il fait beau aujourd'hui
}
```

Test 3 :

```
@Test
public void testSimple() {
    // Chaine simple avec ordre complet
    assertEquals(List.of("666","the", "the", "number", "of", "beast"), testExo.solution("666, the number of the beast", List.of('6',
}
```

Test 4 :

```
@Test
public void testVide() {
    // Chaine vide
    assertEquals(List.of(), testExo.solution("",List.of('6', 't', 'n', 'o', 'b')));
}
```

Test 5 :

```
@Test
public void testIdentique() {
    // Chaine avec des mots identiques
    assertEquals(List.of("demain","aujourd","hui","il", "il", "imagine", "imagine", "pleut", "pleut"), testExo.solution("imagine au
}
```

Test 6 :

```
@Test
public void testMinmaj() {
     // Chaine avec des mots majuscules et minuscules
    assertEquals(List.of("c", "Est", "En", "moment", "le", "aller", "A", "d", "piscine", "La", "somme", "nous", "juin"), testExo.sol
}
```

Test 7 :

```
@Test
public void testLong() {
     // Chaine avec des mots majuscules et minuscules
    assertEquals(List.of("adipisci", "adipisci", "aliquam", "amet", "animi", "architecto", "architecto", "aspernatur", "aspernatur",
}
```

Codacy :

Le site Codacy permet de mesurer la qualité du code, il note sur la Conformité aux conventions de codage les Erreurs et vulnérabilités, Performances et la Maintenabilité et la lisibilité.

En fonction du rang obtenu sur Codacy le

Temps d'exécution :

Pour le calcul du temps d'execution de chaque programme j'ai entouré l'appel de l'algorythme dans le main par ce code :

```
long startTime = System.nanoTime();
System.out.println(testExo.solution(texte,ordre));
long endTime = System.nanoTime();
long duration = (endTime - startTime) / 1000000;
System.out.println("Le programme s'est exécuté en " + duration + " millisecondes.");
```

Il indique donc pour chaque algorythme son temps d'éxéction.

En sachant que pour chaque test de temps j'utilise ces valeurs :

texte = "bonjour je suis lylian derramond et je suis content";

Algorythmes :

Efficacité:

Meilleur :

Algo 33

```java
public static List<String> solution(String str, List<Character> ordre) {
    Map<Character, List<String>> motParPremiereLettre = new HashMap<>();

    for(int i = 0; i < ordre.size(); i++) {
        motParPremiereLettre.put(ordre.get(i), new ArrayList<>());
    }
    StringBuilder motCourant = new StringBuilder();
    for (int i = 0; i < str.length(); i++) {
        char lettre = str.charAt(i);
        if (lettre == ' ') {
            if (motCourant.length() > 0 && motParPremiereLettre.containsKey(motCourant.charAt(0))) {
                motParPremiereLettre.get(motCourant.charAt(0)).add(motCourant.toString());
            }
            motCourant.setLength(0);
        } else {
            motCourant.append(lettre);
        }
    }

    List<String> motsTries = new ArrayList<>();
    for(int i = 0; i < ordre.size(); i++) {
        motsTries.addAll(motParPremiereLettre.get(ordre.get(i)));
    }

    return motsTries;
}
```

Note : 5

Passe seulement le test de la chaine vide

Codacy : B

Algo 67

```java
public static List<String> solution(String str, List<Character> ordre) {
    str = str.replaceAll( regex: "[^\\p{Alnum}]",  replacement: " ");

    List<String> mots = new ArrayList<>();
    StringBuilder mot = new StringBuilder();

    for (char c : str.toCharArray()) {
        if (Character.isLetterOrDigit(c)) {
            mot.append(c);
        } else if (mot.length() > 0) {
            mots.add(mot.toString());
            mot.setLength(0);
        }
    }

    if (mot.length() > 0) {
        mots.add(mot.toString());
    }

    List<String> resultat = new ArrayList<>();
    int[] ordreIndex = new int[256];
```

Note : 18

Passe tous les tests fournis initialements

Codacy : C

Temps d’éxécution : 14 ms

Pire :

Algo 9

```java
public static List<String> solution(String str, List<Character> ordre) {
    // Liste des mots classés dans l'ordre
    List<String> solution = new ArrayList<>();
    // On vérifie si le string rentré est vide, cela permet d'éviter de perdre du temps à parcourir toutes le
    if (!str.isEmpty()) {
        // Liste des mots séparés
        List<String> stringToWordList = new ArrayList<>();
        // Mot que l'on va append
        String current = "";
        // On parcours tout le string renseigné avec une for each
        for (char c : str.toCharArray()) {
            // Si le caractère courant est un espace ou un apostrophe on termine le mot et on l'ajoute à la l
            if (!isAlphanumeric(c)) {
                if(!current.isEmpty()){
                    stringToWordList.add(current);
                    current = "";
                }
            } else {
                current += c;
            }
        }
        // On ajoute le dernier mot à la liste
        stringToWordList.add(current);
        // On crée une nouvelle liste qui contient en premier lieu tous les mots non classés
        List<String> unsorted = new ArrayList<>();
```

Note : 18

Passe tous les tests fournis initialements.

Codacy : B

Temp d’éxécution : 51 ms

Algo 22 :

```java
public static List<String> solution(String texte, List<Character> ordre) {
    Map<Character, Integer> orderMap = new HashMap<>();
    int d = 0;
    for(int e=0; e<10000; e++) {
        d++;
    }
    // Création d'une hashMap pour stocker l'indice de chaque lettre dans l'ordre
    for (int i = 0; i < ordre.size(); i++) {
        orderMap.put(ordre.get(i), i);
    }

    // Liste permettant de stocker les mots ayant leur première lettre dans le dictionnaire
    List<String> words = new ArrayList<>();
    // Liste permettant de stocker les mots n'ayant pas leur première lettre dans le dictionnaire
    List<String> unknownWords = new ArrayList<>();

    StringBuilder currentWord = new StringBuilder();

    // Parcours du texte caractère par caractère
    for (int i = 0; i < texte.length(); i++) {
```

Note : 18

Passe les tests fournis (devient très long pour les grandes chaines à classer)

Codacy : C

Temps éxecution : 12 ms

Simplicité :

Meilleur :

Algo 7

```
new *
public static List<String> solution(String str, List<Character> ordre) {
    String[] split = str.split( regex: " ");
    List<String> result = new ArrayList<>();

    for (char c : ordre) {
        for (String word : split) {
            if (word.charAt(0) == c) {
                result.add(word);
                break;
            }
        }
    }

    return result;
}
```

Note : 2,5 (/2 car pas une classe Exercice)

Ne fonctionnne pas (ne passe aucun test).

Codacy : B

Algo 12

```java
public static List<String> solution(String str, List<Character> ordre) {
    List<String> listeTrier = new ArrayList<>();
    List<String> mots = new ArrayList<>();
    List<String> inconnus = new ArrayList<>();

    if (str.length() == 0) {
        return List.of();
    }

    mots = new ArrayList<>(Arrays.asList(str.split( regex: "[^a-zA-Z0-9]+")));
    inconnus.addAll(mots);

    for (char car : ordre) {
        for (String mot : mots) {
            if (car == mot.charAt(0)) {
                listeTrier.add(mot);
                inconnus.remove(mot);
            }
        }
    }

    listeTrier.addAll(inconnus);

    return listeTrier;
}
```

Note : 18

Passe tous les tests fournis initialements

Codacy : A

Temps d’éxécution : 11 ms

Algo 23

```java
public static String trierMots(String phrase, List<Character> alphabet) {
    // Créer une map pour associer chaque lettre à une liste de mots commençant par cette lettre
    Map<Character, List<String>> motsParLettre = new HashMap<>();
    for (char lettre : alphabet) {
        motsParLettre.put(lettre, new ArrayList<>());
    }

    // Diviser la phrase en mots
    String[] mots = phrase.toLowerCase().split( regex: "\\W+");

    // Parcourir chaque mot et l'ajouter à la liste correspondante dans la map
    for (String mot : mots) {
        if (!mot.isEmpty()) {
            char premiereLettre = mot.charAt(0);
            if (motsParLettre.containsKey(premiereLettre)) {
                motsParLettre.get(premiereLettre).add(mot);
            }
        }
    }

    // Créer une liste de mots triés en parcourant l'alphabet et concaténant les listes de mots
    StringBuilder phraseTrie = new StringBuilder();
    for (char lettre : alphabet) {
        List<String> motsDeLaLettre = motsParLettre.get(lettre);
        for (String mot : motsDeLaLettre) {
            phraseTrie.append(mot).append(" ");
```

Note : 4 (non respect de la nomemclature : trierMots)

Ne passe aucun test

Codacy : C

Pire :

Algo 20

```java
/**
 * La meilleure solution de l'exercice
 * @param ordre L'ordre des lettres
 * @param inputString Le texte à trier
 * @return Je sais pas, je suis pas un dev
 * @see <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">La meilleure solution</a>
 * @throws SkillIssueException
 * @throws YouAreNotAGoodProgrammerException
 */
public static List<String> solution(String ordre, List<Character> inputString) {
    String[] result = ordre.split( regex: "\\s+|'|’");

    List<String> newordre = new ArrayList<>();

    ArrayList<String>[] oldordre = new ArrayList[inputString.size()+1];
    for (int i = 0; i < oldordre.length; i++) {
        oldordre[i] = new ArrayList<>();
    }

    boolean amIAGoodProgrammer = false; // Normalement toujours vrai, je suis le meilleur ;)
    List<String> newinput = new ArrayList<>();
```

Hors concour (ne compile pas)

Algo 56

```java
public static List<String> solution(String t, List<Character> o) {
    List<String> m = new ArrayList<>();
    String[] mt = t.split( regex: " ");

    for (char l : o) {
        for (String w : mt) {
            if (w.charAt(0) == l) {
                m.add(w);
            }
        }
    }

    return m;
}
```

Note : 5

Ne passe aucun test

Codacy : B

Sobriété :

Meilleur :

Algo 19

```java
/**
 * Classe interne représentant un noeud dans un arbre binaire de recherche.
 */
7 usages  new *
public static class Arbre {
    4 usages
    private Arbre gauche;
    4 usages
    private Arbre droite;
    3 usages
    private String valeur;

    /**
     * Constructeur de la classe Arbre.
     * @param valeur la valeur associée au noeud
     */
    1 usage  new *
    public Arbre(String valeur) {
        this.valeur = valeur;
        gauche = null;
        droite = null;
    }
}
```

Note: 5 (/2 car pas une classe Exercice)

Ne passe pas tous les tests fournis initialement

Codacy : C

Algo 57

```java
public static List<String> solution(String str, List<Character> ordre) {
    List<String> mots = new ArrayList<>();
    StringBuilder motActuel = new StringBuilder();

    int length = str.length();

    for (int i = 0; i < length; i++) {
        char c = str.charAt(i);
        if (Character.isLetter(c)) {
            motActuel.append(c);
        } else {
            if (motActuel.length() > 0) {
                mots.add(motActuel.toString());
                motActuel.setLength(0);
            }
        }
    }

    if (motActuel.length() > 0) {
        mots.add(motActuel.toString());
    }

    mots.sort((a, b) -> {
        char premiereLettreA = Character.toLowerCase(a.charAt(0));
        char premiereLettreB = Character.toLowerCase(b.charAt(0));
```

Note : 5

Passe seulement les test chaine vide et 1 mot

Codacy : A

Pire :

Algo 5 :

```java
package exercice;

import java.util.ArrayList;

public class testExo {

    public static List<String> solution(String str, List<Character> ordre) {
        List<String> words = Arrays.asList(str.split("[^a-zA-Z0-9]"));
        List<String> sortedWords = new ArrayList<>();

        for (char c : ordre) {
            for (String word : words) {
                if (!sortedWords.contains(word) && word.charAt(0) == c) {
                    sortedWords.add(word);
                }
            }
        }

        for (String word : words) {

            if (!sortedWords.contains(word)) {
                sortedWords.add(word);
            }
        }

        return sortedWords;
    }
}
```

Note 10.

Passe seulement 2 tests sur les 7 (Exemple et 1 mot)

Temps d’execution : 12 ms

Codacy : rang A

Algo 14 :

```
3 usages
static ArrayList<String> tokens = new ArrayList<>();
7 usages
static ArrayList<String> listefinale = new ArrayList<>();

public static List<String> solution(String str, List<Character> ordre) {
    listefinale.clear();
    // Verifie que la chaine n'est pas vide
    if (str.isEmpty()) {
        List<String> list = listefinale;
        return list;
    }

    // Permet de separer la Sring en plusieur partie pour recuperer chaque mots
    // Les partie a separer sont délimité par des espace, des apostrophes, ect...
    StringTokenizer tokenizer = new StringTokenizer(str,  delim: " ;',;  ;'");

    // Premet de rajouter chaque mots dans une ArrayList
    while (tokenizer.hasMoreTokens()) {
        String token = tokenizer.nextToken();
        tokens.add(token);
    }
    // Verifie que la chaine ne contient pas qu'un seul mots
    if (tokens.size() == 1) {
        listefinale.add("" + str);
        return listefinale;
    }
```

Note : 10

Ne passe pas tous les tests founis initialements (Exemple et vide)

Codacy : B

Classement :

Efficacité:

1 er : 9-pire

2ème : 67-meilleur

3ème : 22-pire

4ème : 33-meilleur

Simplicité :

1 er : 12-meilleur

2ème : 56-pire

3ème : 23-meilleur

4ème : 7-meilleur

Sobriété :

1er : 5-pire (Codacy A)

2ème :14-pire(Codacy B)

3ème : 57-meilleur (Codacy A)

4ème : 19-meilleur (Codacy C)