

ANSIA (Animo_Nervoso:Studenti_Intensamente_distrAtti)

Versione 1.0

Data di rilascio: 21/01/2024

Integrazione e test di sistemi software
a.a. 2023-2024

Informatica e Tecnologie per la Produzione
del Software

Realizzato da:

Lorenzo Abatescianni ITPS - 759521
l.abatescianni@studenti.uniba.it

Maria Creanza ITPS - 758187
m.creanza6@studenti.uniba.it

Sommario

HOMEWORK 1	3
7 STEP	3
1. COMPRENDERE I REQUISITI (COSA DEVE FARE IL PROGRAMMA, INPUT E OUTPUT)	3
2. ESPLORA COSA FA IL PROGRAMMA PER VARI INPUT	3
3. ESPLORARE INPUT, OUTPUT ED IDENTIFICARE LE PARTIZIONI	5
4. CASI LIMITE	6
5. DEFINIRE I CASI DI TEST	7
6. AUTOMATIZZARE CASI DI TEST	8
7. AUMENTARE LA SUITE CON CREATIVITÀ ED ESPERIENZA	18
LEGGERE L'IMPLEMENTAZIONE	19
ESEGUIRE TEST CON TOOL DI CODE COVERAGE	19
HOMEWORK 2	21
SCELTA CODICE	21
PROPRIETÀ DELL'INPUT	21
PROPRIETÀ DELL'OUTPUT	21
LASCIARE CHE SIA IL FRAMEWORK A SCEGLIERE I TEST	22
STATISTICA: MOTIVO UTILIZZO ISTOGRAMMA	26
COVERAGE	27

HOMEWORK 1

7 STEP

1. COMPRENDERE I REQUISITI (COSA DEVE FARE IL PROGRAMMA, INPUT E OUTPUT).

- Creazione dell'hashMap delle Parole:

Il programma, tramite il metodo `createWordList`, deve prendere in input una stringa contenente il testo da analizzare (`testoRomanzo`).

Deve restituire una hashMap (`wordList`) contenente le parole del testo come chiavi e il numero di occorrenze di ciascuna parola come valore.

- Ordinamento per Numero di Occorrenze:

Deve fornire un metodo (`sortByOccurrences`) che prende l'hashMap delle parole (`wordList`) e restituisce una treeMap ordinata in base al numero di occorrenze delle parole, in ordine decrescente. Al suo interno e inoltre presente un metodo di comodo chiamato `sortedByValues` che implementa la logica attraverso la quale `sortByOccurrences` opera.

- Ordinamento Alfabetico con Gestione di Numeri e Caratteri Speciali:

Deve fornire un metodo (`sortByAlphabeticalOrder`) che prende l'hashMap delle parole (`wordList`) e restituisce una treeMap ordinata in ordine alfabetico (`alphabeticalOrder`). Le parole devono essere ordinate in modo che i numeri siano dopo le lettere, i caratteri speciali siano dopo i numeri, e i segni di punteggiatura siano dopo i caratteri speciali.

- Verifica dell'Esistenza di una Parola nell'hashMap:

Deve fornire un metodo (`checkWordExistence`) che dopo aver preso in input una stringa (`inputWord`) e la hashMap delle parole (`wordList`) restituisce `true` se la stringa è presente, altrimenti `false`.

2. ESPLORA COSA FA IL PROGRAMMA PER VARI INPUT.

- Per quanto riguarda il metodo `createWordList`, abbiamo creato quattro classi di test per verificare, in base ai vari input forniti nel testo, come il metodo risponde: Verifichiamo che il metodo restituisca delle parole con le loro occorrenze se l'input non prevede segni di punteggiatura o numeri:

```
//Verifico che il metodo createWordList restituisca la mappa delle parole con le rispettive occorrenze
@Test
@DisplayName("SoloTesto")
void createWordList_ShouldHandleWordsOnly() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Testo di esempio senza numeri o segni di punteggiatura nel testo";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals( expected: 2, result.get("testo"));
    assertEquals( expected: 2, result.get("di"));
    assertEquals( expected: 1, result.get("esempio"));
    assertEquals( expected: 1, result.get("senza"));
    assertEquals( expected: 1, result.get("numeri"));
    assertEquals( expected: 1, result.get("o"));
    assertEquals( expected: 1, result.get("segni"));
    assertEquals( expected: 1, result.get("punteggiatura"));
    assertEquals( expected: 1, result.get("nel"));
    assertEquals( expected: 1, result.get("testo"));
}

Tests passed: 1 of 1 test - 14ms

> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 757ms
3 actionable tasks: 1 executed, 2 up-to-date
12:17:05: Execution finished 'test --tests "OccorrenzeTest.createWordList_ShouldHandleWordsOnly"'
```

➔ Come si può notare, il test è andato a buon fine

- Verifichiamo che il metodo `sortByOccurrences` ordini la mappa in base ai criteri stabiliti:

```
@Test
@DisplayName("OrdinaMappaOccurrences")
public void testSortByOccurrences() {
    Occorrenze sorter = new Occorrenze();

    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", 3);
    wordList.put("banana", 2);
    wordList.put("orange", 5);
    wordList.put("grape", 1);

    TreeMap<String, Integer> sortedMap = sorter.sortByOccurrences(wordList);
    assertEquals( expected: "orange", sortedMap.keySet().toArray()[0]);
    assertEquals( expected: "apple", sortedMap.keySet().toArray()[1]);
    assertEquals( expected: "banana", sortedMap.keySet().toArray()[2]);
    assertEquals( expected: "grape", sortedMap.keySet().toArray()[3]);
}

✓ Tests passed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
```

- Verifichiamo che il metodo `sortByAlphabeticalOrder` ordini la mappa in base ai criteri stabiliti:

Verifico che l'ordinamento sia in ordine alfabetico per chiave

```
//Verifico che il metodo sortByAlphabeticalOrder restituisca la mappa ordinata in base all'ordinamento alfabetico della chiave
@Test
@DisplayName("OrdinaMappaAlfabeticoNormale")
public void testSortByAlphabeticalOrder() {
    // Crea un'istanza della classe che contiene i metodi di ordinamento
    Occorrenze sorter = new Occorrenze();

    // Caso di test con uno dei valori null
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", null);
    wordList.put("banana", 2);
    wordList.put("orange", null);
    wordList.put("grape", 3);

    // Chiama il metodo di ordinamento
    TreeMap<String, Integer> sortedMap = sorter.sortByAlphabeticalOrder(wordList);
    assertEquals( expected: "apple", sortedMap.keySet().toArray()[0]);
    assertEquals( expected: "banana", sortedMap.keySet().toArray()[1]);
    assertEquals( expected: "grape", sortedMap.keySet().toArray()[2]);
    assertEquals( expected: "orange", sortedMap.keySet().toArray()[3]);
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 15 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 710ms
```

- Verifichiamo che il metodo `checkWordExistence` restituisca un booleano secondo i criteri stabiliti:

Verifico che, se la parola è contenuta, restituisce true, altrimenti false

```
//Verifico che se inserisco una inputword che esiste nella hashmap,
// restituisce true, altrimenti false
@Test
@DisplayName("VerificaEsistenza")
public void testCheckWordExistence() {
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", 1);
    wordList.put("banana", 2);
    wordList.put("orange", 3);
    // Creare un oggetto della classe contenente il metodo checkWordExistence
    Occorrenze wordExistenceChecker = new Occorrenze();
    boolean result = wordExistenceChecker.checkWordExistence(wordList, inputWord: "apple");
    assertTrue(result);
    result = wordExistenceChecker.checkWordExistence(wordList, inputWord: "grape");
    assertFalse(result);
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 35 ms

> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine
which ones are important for your project.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command_line_interface.html#sec:command_line_warnings
BUILD SUCCESSFUL in 13s
3 actionable tasks: 2 executed, 1 up-to-date
15:55:52: Execution finished ':test --tests "OccorrenzeTest.testCheckWordExistence"
```

3. ESPLORARE INPUT, OUTPUT ED IDENTIFICARE LE PARTIZIONI

- *createWordList*
 - parametro: testoRomanzo
 - null || testoRomanzo.length() == 0
 - testoRomanzo.length() > 0
 - output atteso:
 - null
 - lista con numero di occorrenze
- *sortByAlphabeticalOrder*
 - parametro: wordList
 - null || wordList.size() == 0
 - wordList.size() > 0
 - output atteso:
 - null
 - lista ordinata in ordine alfabetico
- *sortedByValues*
 - parametro: map
 - null || map.size() == 0
 - map.size() > 0
 - output atteso:
 - null
 - lista ordinata per occorrenze

- *checkWordExistence*
 - parametro: wordList
 - null || wordList.size() == 0
 - wordList.size() > 0
 - parametro: inputWord
 - null || inputWord.length() == 0
 - inputWord.length() > 0
- *checkWordExistence* (combinazioni)

#	wordList	inputWord	Output attesi
C1	null wordList.size() == 0	null inputWord.length() == 0	false
C2	null wordList.size() == 0	inputWord.length() > 0	false
C3	wordList.size() > 0	null inputWord.length() == 0	false
C4	wordList.size() > 0	inputWord.length() > 0	Boolean che indica la presenza o meno della stringa

4. CASI LIMITE

- *createWordList*
 - testoRomanzo != null
 - null e l'out point e on point
 - "str" e un in point e off point
 - testoRomanzo.length() > 0
 - 0 e l'out point e on point
 - 1 e un in point e off point
- *sortByAlphabeticalOrder*
 - wordList != null
 - null e l'out point e on point
 - L'oggetto wordList inizializzato e un in point e off point
 - wordList.size() > 0
 - 0 e l'out point e on point
 - 1 e un in point e off point
- *sortedByValues*
 - map != null
 - null e l'out point e on point
 - L'oggetto map inizializzato e un in point e off point
 - map.size() > 0
 - 0 e l'out point e on point
 - 1 e un in point e off point
- *checkWordExistence*
 - wordList != null
 - null e l'out point e on point
 - L'oggetto wordList inizializzato e un in point e off point
 - wordList.size() > 0
 - 0 e l'out point e on point

- 1 e un in point e off point
- inputWord != null
 - null e l'out point e on point
 - "str" e un in point e off point
- inputWord.length() > 0
 - 0 e l'out point e on point
 - 1 e un in point e off point

5. DEFINIRE I CASI DI TEST

- Casi eccezionali
 - *createWordList*
 - T1: testoRomanzo == null || testoRomanzo.length() == 0
 - *sortByAlphabeticalOrder*
 - T2: wordList == null || wordList.size() == 0
 - *sortedByValues*
 - T3: map == null || map.size() == 0
- Test per i parametri non nulli o vuoti
 - *createWordList*
 - T4: testoRomanzo contiene segni di punteggiatura
 - T5: testoRomanzo contiene numeri
 - T6: testoRomanzo contiene segni di punteggiatura e numeri
 - *sortedByValues*
 - T7: la mappa contiene valori duplicati
 - T8: la mappa contiene valori nulli
 - T9: la mappa contiene valori duplicati e nulli
 - *sortByAlphabeticalOrder*
 - T10: le stringhe con solo numeri devono essere messe dopo le lettere
 - T11: le stringhe con solo caratteri speciali devono essere messe dopo i numeri
 - T12: le stringhe con solo segni di punteggiatura devono essere messe dopo i caratteri speciali
- Test sulle combinazioni
 - *checkWordExistence*
 - T13: la lista è nulla/vuota e la stringa è nulla/vuota
 - T14: la lista è nulla/vuota e la stringa non è nulla/vuota
 - T15: la lista è popolata e la stringa è nulla/vuota
 - T16: la lista è popolata e la stringa non è nulla/vuota

6. AUTOMATIZZARE CASI DI TEST

- *createWordList*
 - o T1

```
// T1
@Test
@DisplayName("TestoRomanzo nullo o di lunghezza 0")
public void testNoText() {
    String testoRomanzo = null;
    Occorrenze occorrenze = new Occorrenze();
    HashMap<String, Integer> wordList = occorrenze.createWordList(testoRomanzo);
    assertNull(wordList);
    testoRomanzo = "";
    wordList = occorrenze.createWordList(testoRomanzo);
    assertNull(wordList);
}

Tests failed: 1 of 1 test - 15ms

> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

java.lang.NullPointerException: Create breakpoint: Cannot invoke "String.split(String)" because "testoRomanzo" is null
    at org.example.Occorrenze.createWordList(Occorrenze.java:19)
>   at OccorrenzeTest.testNoText(OccorrenzeTest.java:260) <29 internal lines>
>   at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
```

→ Come si può notare, nel metodo non sono presenti controlli sulla presenza di parametri nulli o vuoti, di conseguenza bisogna modificare il codice:

```
public HashMap<String, Integer> createWordList(String testoRomanzo) {
    HashMap<String, Integer> wordList = new HashMap<>();
    if (testoRomanzo != null && !testoRomanzo.isEmpty()) {
        // Utilizza un'espressione regolare per dividere il testo in parole
        String[] words = testoRomanzo.split(regex: "\\b|\\p{Punct}|\\b");

        for (String word : words) {
            // Rimuovi eventuali spazi vuoti iniziali e finali
            word = word.trim();

            // Converti la parola in minuscolo
            word = word.toLowerCase();

            // Se la parola non è vuota, aggiorna la mappa delle parole
            if (!word.isEmpty()) {
                if (wordList.containsKey(word)) {
                    wordList.put(word, wordList.get(word) + 1);
                } else {
                    wordList.put(word, 1);
                }
            }
        }
    } else {
        wordList = null;
    }

    return wordList;
}
```

```
Tests passed: 1 of 1 test - 13ms

> Task :compileJava
Note: C:\Users\Utente\Documents\Università\III_Anno\Test\Occorrenze\src\main\java\org\example\Occorrenze.java use
Note: Recompile with -Xlint:unchecked for details.
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
```

→ Con l'aggiunta del controllo, il test funziona

- o T4
- Verifichiamo che il metodo restituisca delle parole con le loro occorrenze se l'input prevede segni di punteggiatura:



```
// T4
@Test
@DisplayName("ConPunteggiatura")
void createWordList_ShouldHandlePunctuation() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Testo di esempio, con segni di punteggiatura nel testo.";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals( expected: 2, result.get("testo"));
    assertEquals( expected: 2, result.get("di"));
    assertEquals( expected: 1, result.get("esempio"));
    assertEquals( expected: 1, result.get(","));
    assertEquals( expected: 1, result.get("con"));
    assertEquals( expected: 1, result.get("segni"));
    assertEquals( expected: 1, result.get("punteggiatura"));
    assertEquals( expected: 1, result.get("nel"));
    assertEquals( expected: 1, result.get("."));
}

Tests failed: 1 of 1 test - 17 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE

Expected :2
Actual    :1
<Click to see difference>
```

- E stato trovato un bug: il metodo considera le parole con segni di punteggiatura come se fossero diverse da quelle "staccate". Infatti, se modifichiamo il test inserendo spazi tra le parole e i segni di punteggiatura:

```
//Verifico che il metodo createWordList restituisca la mappa delle parole con le rispettive occorrenze, con la
//presenza di segni di punteggiatura
@Test
@DisplayName("ConPunteggiatura")
void createWordList_ShouldHandlePunctuation() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Testo di esempio , con segni di punteggiatura nel testo .";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals( expected: 2, result.get("testo"));
    assertEquals( expected: 2, result.get("di"));
    assertEquals( expected: 1, result.get("esempio"));
    assertEquals( expected: 1, result.get("con"));
    assertEquals( expected: 1, result.get("segni"));
    assertEquals( expected: 1, result.get("punteggiatura"));
    assertEquals( expected: 1, result.get("nel"));
}


```

Si ottiene:

```
Tests passed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 874ms
3 actionable tasks: 2 executed, 1 up-to-date
```

Quindi, non potendo inserire stringhe in questa maniera, la soluzione consiste in una piccola modifica della regex:

```
// Utilizza un'espressione regolare per dividere il testo in parole considerando la punteggiatura
String[] words = testoRomanzo.split( regex: "\\b|\\p{Punct}|\\b");
```

Riprovando quindi a effettuare il test:

```
//Verifico che il metodo createWordList restituisca la mappa delle parole con le rispettive occorrenze, con la
//presenza di segni di punteggiatura
@Test
@DisplayName("ConPunteggiatura")
void createWordList_ShouldHandlePunctuation() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Testo di esempio, con segni di punteggiatura nel testo.";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals( expected: 2, result.get("testo"));
    assertEquals( expected: 2, result.get("di"));
    assertEquals( expected: 1, result.get("esempio"));
    assertEquals( expected: 1, result.get(","));
    assertEquals( expected: 1, result.get("con"));
    assertEquals( expected: 1, result.get("segni"));
    assertEquals( expected: 1, result.get("punteggiatura"));
    assertEquals( expected: 1, result.get("nel"));
    assertEquals( expected: 1, result.get("."));
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 12ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 790ms
```

○ T5

Verifichiamo che il metodo restituisca delle parole con le loro occorrenze se l'input prevede numeri:

```
// T5
@Test
@DisplayName("ConNumeri")
void createWordList_ShouldHandleNumbers() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Testo di esempio con numeri come 123 e 456";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals( expected: 1, result.get("testo"));
    assertEquals( expected: 1, result.get("di"));
    assertEquals( expected: 1, result.get("esempio"));
    assertEquals( expected: 1, result.get("con"));
    assertEquals( expected: 1, result.get("numeri"));
    assertEquals( expected: 1, result.get("come"));
    assertEquals( expected: 1, result.get("123"));
    assertEquals( expected: 1, result.get("e"));
    assertEquals( expected: 1, result.get("456"));
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 34ms

Starting Gradle Daemon...
Gradle Daemon started in 1 s 671 ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from y

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 12s
```

- T6
Verifichiamo che il metodo restituisca delle parole con le loro occorrenze se l'input prevede numeri e segni di punteggiatura:

```
// T6
@Test
@DisplayName("ConPunteggiaturaenumeri")
void createWordList_ShouldHandleMixedInput() {
    Occorrenze occorrenzeAnalyzer = new Occorrenze();
    String testText = "Questo e un testo di esempio con segni di punteggiatura e numeri, come 123 e 456!";
    HashMap<String, Integer> result = occorrenzeAnalyzer.createWordList(testText);
    assertNotNull(result);
    assertEquals(expected: 1, result.get("questo"));
    assertEquals(expected: 3, result.get("e"));
    assertEquals(expected: 1, result.get("un"));
    assertEquals(expected: 1, result.get("testo"));
    assertEquals(expected: 2, result.get("di"));
    assertEquals(expected: 1, result.get("esempio"));
    assertEquals(expected: 1, result.get("con"));
    assertEquals(expected: 1, result.get("segni"));
    assertEquals(expected: 1, result.get("punteggiatura"));
    assertEquals(expected: 1, result.get("numeri"));
    assertEquals(expected: 1, result.get("come"));
    assertEquals(expected: 1, result.get("123"));
    assertEquals(expected: 1, result.get("456"));
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 14ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 3s
3 actionable tasks: 2 executed, 1 up-to-date
15:17:35: Execution finished ':test --tests "OccorrenzeTest.createWordList_ShouldHandleMixedInput"'.
```

- *sortByAlphabeticalOrder*

- T2

```
// T2
@Test
@DisplayName("wordList nullo o vuoto")
public void testNoList() {
    HashMap<String, Integer> wordList = null;
    Occorrenze occorrenze = new Occorrenze();
    TreeMap<String, Integer> sortedList = occorrenze.sortByAlphabeticalOrder(wordList);
    assertNull(sortedList);
    HashMap<String, Integer> Emptymap = new HashMap<>();
    sortedList = occorrenze.sortByAlphabeticalOrder(Emptymap);
    assertNull(sortedList);
}
```

```
✗ Tests failed: 1 of 1 test - 14ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

java.lang.NullPointerException: Create breakpoint : Cannot invoke "java.util.Map.size()" because "map" is null
at java.base/java.util.TreeMap.putAll(TreeMap.java:314)
```

- ➔ Anche in questo caso, non è presente il controllo, pertanto bisogna aggiungerlo:

```
public TreeMap<String, Integer> sortByAlphabeticalOrder(HashMap<String, Integer> wordList) {  
    // Controlla se wordList è null o vuota  
    if (wordList == null || wordList.isEmpty()) {  
        return null;  
    }  
  
    TreeMap<String, Integer> alphabeticalOrder = new TreeMap<>(new Comparator<String>() {  
        @Override  
        public int compare(String str1, String str2) {  
            // ...  
        }  
    });  
    // ...  
}
```

✓ Tests passed: 1 of 1 test - 12 ms

```
> Task :compileJava  
Note: C:\Users\Utente\Documents\Universita\III_Anno\Test\Occorrenze\src\main\java\org\example\Occorrenze.java use  
Note: Recompile with -Xlint:unchecked for details.  
> Task :processResources NO-SOURCE  
> Task :classes  
> Task :compileTestJava UP-TO-DATE
```

→ Con l'aggiunta del controllo, il test funziona

○ T10

Verifichiamo che gli elementi con chiave solo numerica siano inseriti dopo quelli con solo lettere

```
// T10  
@Test  
@DisplayName("OrdinaMappaAlfabeticoNumeri")  
public void testSortByAlphabeticalOrderWithNumbers() {  
    // Crea un'istanza della classe che contiene i metodi di ordinamento  
    Occorrenze sorter = new Occorrenze();  
  
    // Caso di test con uno dei valori null  
    HashMap<String, Integer> wordList = new HashMap<>();  
    wordList.put("apple", null);  
    wordList.put("321", 2);  
    wordList.put("orange", null);  
    wordList.put("grape", 3);  
  
    // Chiama il metodo di ordinamento  
    TreeMap<String, Integer> sortedMap = sorter.sortByAlphabeticalOrder(wordList);  
    assertEquals("expected: \"apple\"", sortedMap.keySet().toArray()[0]);  
    assertEquals("expected: \"grape\"", sortedMap.keySet().toArray()[1]);  
    assertEquals("expected: \"orange\"", sortedMap.keySet().toArray()[2]);  
    assertEquals("expected: \"321\"", sortedMap.keySet().toArray()[3]);  
}
```

✓ Tests passed: 1 of 1 test - 14 ms

```
> Task :compileJava UP-TO-DATE  
> Task :processResources NO-SOURCE  
> Task :classes UP-TO-DATE  
> Task :compileTestJava UP-TO-DATE  
> Task :processTestResources NO-SOURCE  
> Task :testClasses UP-TO-DATE  
> Task :test
```

○ T11

Verifichiamo che gli elementi con chiave costituita solo da caratteri speciali siano messi dopo quelli con solo numeri


```
// T11
@Test
@DisplayName("OrdinaMappaAlfabeticoCaratteriSpeciali")
public void testSortByAlphabeticalOrderWithSpecialChar() {
    // Crea un'istanza della classe che contiene i metodi di ordinamento
    Occorrenze sorter = new Occorrenze();

    // Caso di test con uno dei valori null
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", null);
    wordList.put("321", 2);
    wordList.put("orange", null);
    wordList.put("???", 3);

    // Chiama il metodo di ordinamento
    TreeMap<String, Integer> sortedMap = sorter.sortByAlphabeticalOrder(wordList);
    assertEquals( expected: "apple", sortedMap.keySet().toArray()[0]);
    assertEquals( expected: "orange", sortedMap.keySet().toArray()[1]);
    assertEquals( expected: "321", sortedMap.keySet().toArray()[2]);
    assertEquals( expected: "???", sortedMap.keySet().toArray()[3]);
}

✓ Tests passed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
```

○ T12

Verifico che gli elementi con chiave composta solo da segni di punteggiatura siano messi dopo quelli con solo numeri

```
// T12
@Test
@DisplayName("OrdinaMappaAlfabeticoPunteggiatura")
public void testSortByAlphabeticalOrderWithPunct() {
    // Crea un'istanza della classe che contiene i metodi di ordinamento
    Occorrenze sorter = new Occorrenze();

    // Caso di test con uno dei valori null
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", null);
    wordList.put("321", 2);
    wordList.put("???", null);
    wordList.put("???", 3);

    // Chiama il metodo di ordinamento
    TreeMap<String, Integer> sortedMap = sorter.sortByAlphabeticalOrder(wordList);
    assertEquals( expected: "apple", sortedMap.keySet().toArray()[0]);
    assertEquals( expected: "321", sortedMap.keySet().toArray()[1]);
    assertEquals( expected: "???", sortedMap.keySet().toArray()[2]);
    assertEquals( expected: "???", sortedMap.keySet().toArray()[3]);
}

✓ Tests passed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
```

- *sortedByValues*

o T3

```
// T3
@Test
@DisplayName("map nullo o vuoto")
public void testNoMap() {
    HashMap<String, Integer> map = null;
    Occorrenze occorrenze = new Occorrenze();
    TreeMap<String, Integer> sortedList = occorrenze.sortByOccurrences(map);
    assertNull(sortedList);
    HashMap<String, Integer> Emptymap = new HashMap<>();
    sortedList = occorrenze.sortByOccurrences(Emptymap);
    assertNull(sortedList);
}

Tests failed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

java.lang.NullPointerException: Create breakpoint: Cannot invoke "java.util.Map.size()" because "map" is null
```

→ Anche in questo caso, non è presente il controllo, pertanto bisogna aggiungerlo:

```
4 usages
public TreeMap<String, Integer> sortByOccurrences(HashMap<String, Integer> wordList) {
    // Controllo se wordList è nullo o vuota
    if (wordList == null || wordList.isEmpty()) {
        return null;
    } else {
        return (TreeMap<String, Integer>) sortedByValues(wordList);
    }
}

Tests passed: 1 of 1 test - 12 ms

> Task :compileJava
Note: C:\Users\Utente\Documents\Università\III_Anno\Test\Occorrenze\src\main\java\org\example\Occorrenze.java use
Note: Recompile with -Xlint:unchecked for details.
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
```

→ Con l'aggiunta del controllo, il test funziona

o T7

Verifichiamo che il metodo `sortByOccurrences` ordini la mappa in base ai criteri stabiliti. Se ci sono dei valori duplicati, l'ordinamento avviene sulla chiave in ordine decrescente

```
// T7
@Test
@DisplayName("OrdinaMappaDuplicati")
public void testSortByOccurrencesDuplicates() {
    Occorrenze sorter = new Occorrenze();

    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", 3);
    wordList.put("banana", 2);
    wordList.put("orange", 5);
    wordList.put("grape", 3);

    TreeMap<String, Integer> sortedMap = sorter.sortByOccurrences(wordList);
    assertEquals("expected: 'orange'", sortedMap.keySet().toArray()[0]);
    assertEquals("expected: 'grape'", sortedMap.keySet().toArray()[1]);
    assertEquals("expected: 'apple'", sortedMap.keySet().toArray()[2]);
    assertEquals("expected: 'banana'", sortedMap.keySet().toArray()[3]);
}
```

Si ottiene:

```

✓ Tests passed: 1 of 1 test - 13 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 685ms

```

○ T8

Se ci sono dei valori nulli, l'ordinamento avviene considerando i nulli come se fossero 0 (in questo caso il test restituiva failure in quanto l'elemento nullo non faceva più parte della mappa), quindi è stata apportata una modifica al metodo):

```

//Sostituisce null con 0
value1 = (value1 == null) ? (V) Integer.valueOf(0) : value1;
value2 = (value2 == null) ? (V) Integer.valueOf(0) : value2;

// T8
@Test
@DisplayName("OrdinaMappaZero")
public void testSortByOccurrencesZero() {
    // Crea un'istanza della classe che contiene i metodi di ordinamento
    Occorrenze sorter = new Occorrenze();

    // Caso di test con uno dei valori null
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", null);
    wordList.put("banana", 2);
    wordList.put("orange", 1);
    wordList.put("grape", 3);

    // Chiama il metodo di ordinamento
    TreeMap<String, Integer> sortedMap = sorter.sortByOccurrences(wordList);
    assertEquals("grape", sortedMap.keySet().toArray()[0]);
    assertEquals("banana", sortedMap.keySet().toArray()[1]);
    assertEquals("orange", sortedMap.keySet().toArray()[2]);
    assertEquals("apple", sortedMap.keySet().toArray()[3]);
}

```

Si ottiene:

```

✓ Tests passed: 1 of 1 test - 14 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 689ms

```

○ T9

In caso di entrambe le problematiche, il comportamento deve seguire entrambi i procedimenti:



```
// T9
@Test
@DisplayName("OrdinaMappaEntrambi")
public void testSortByOccurrencesBoth() {
    // Crea un'istanza della classe che contiene i metodi di ordinamento
    Occorrenze sorter = new Occorrenze();

    // Caso di test con uno dei valori null
    HashMap<String, Integer> wordList = new HashMap<>();
    wordList.put("apple", null);
    wordList.put("banana", 2);
    wordList.put("orange", null);
    wordList.put("grape", 3);

    // Chiama il metodo di ordinamento
    TreeMap<String, Integer> sortedMap = sorter.sortByOccurrences(wordList);
    assertEquals( expected: "grape", sortedMap.keySet().toArray()[0]);
    assertEquals( expected: "banana", sortedMap.keySet().toArray()[1]);
    assertEquals( expected: "orange", sortedMap.keySet().toArray()[2]);
    assertEquals( expected: "apple", sortedMap.keySet().toArray()[3]);
}
```

Si ottiene:

```
✓ Tests passed: 1 of 1 test - 14ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 824ms
```

- *checkWordExistence*

o T13

la lista è nulla/vuota e la stringa è nulla/vuota

```
// T13
@Test
@DisplayName("map nullo o vuoto e stringa nulla/vuota")
public void testNoMapString() {
    HashMap<String, Integer> map = null;
    String inputWord = null;
    Occorrenze occorrenze = new Occorrenze();
    boolean result = occorrenze.checkWordExistence(map, inputWord);
    assertFalse(result);
}
```

```
✗ Tests failed: 1 of 1 test - 14ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
```

- o Anche in questo caso non sono presenti controlli, bisogna modificare il codice:


```
public boolean checkWordExistence(HashMap<String, Integer> wordList, String inputWord) {
    if (wordList == null || wordList.isEmpty() || inputWord == null || inputWord.isEmpty()) {
        return false;
    }
    return wordList.containsKey(inputWord);
}
```

✓ Tests passed: 1 of 1 test – 11 ms

```
> Task :compileJava
Note: C:\Users\Utente\Documents\Università\III_Anno\Test\Occorrenze\src
Note: Recompile with -Xlint:unchecked for details.
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
```

- T14
la lista è nulla/vuota e la stringa non è nulla/vuota

```
// T14
@Test
@DisplayName("map nullo o vuoto e stringa non nulla/vuota")
public void testNoMapWithString() {
    HashMap<String, Integer> map = null;
    String inputWord = "str";
    Occorrenze occorrenze = new Occorrenze();
    boolean result = occorrenze.checkWordExistence(map, inputWord);
    assertFalse(result);
}
```

✓ Tests passed: 1 of 1 test – 12 ms

```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
```

- T15
la lista è popolata e la stringa è nulla/vuota

```
// T15
@Test
@DisplayName("map non nullo o vuoto e stringa nulla/vuota")
public void testMapNoString() {
    HashMap<String, Integer> map = new HashMap<>();
    map.put("apple", 3);
    String inputWord = null;
    Occorrenze occorrenze = new Occorrenze();
    boolean result = occorrenze.checkWordExistence(map, inputWord);
    assertFalse(result);
}
```

✓ Tests passed: 1 of 1 test – 12 ms

```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
```

- T16
la lista è popolata e la stringa non è nulla/vuota

```
// T16
@Test
@DisplayName("map non nullo o vuoto e stringa non nulla/vuota")
public void testMapWithString() {
    HashMap<String, Integer> map = new HashMap<>();
    map.put("apple", 3);
    String inputWord = "apple";
    Occorrenze occorrenze = new Occorrenze();
    boolean result = occorrenze.checkWordExistence(map, inputWord);
    assertTrue(result);
    inputWord = "str";
    result = occorrenze.checkWordExistence(map, inputWord);
    assertFalse(result);
}

✓ Tests passed: 1 of 1 test - 12 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
```

7. AUMENTARE LA SUITE CON CREATIVITÀ ED ESPERIENZA

○ T17

Il metodo prende una mappa di stringhe e interi wordList, una stringa inputWord, e restituisce un valore booleano che indica se la parola inputWord esiste nella mappa wordList. Il metodo testCheckWordExistence viene eseguito per ogni coppia di argomenti forniti dalla sorgente del metodo provideTestCasesCheckWordExistence. Gli argomenti sono coppie di mappe, stringhe e booleani che rappresentano casi di test diversi. Ad esempio, la prima coppia di argomenti verifica se la parola "banana" esiste nella mappa fornita e si aspetta che il risultato sia true. Gli altri casi coprono scenari come la mancanza di una parola nella mappa, l'input nullo o una mappa nulla. In breve, il test si assicura che il metodo checkWordExistence gestisca correttamente una serie di situazioni diverse e restituisca i risultati attesi per ciascuna di esse.

```
// T17
@ParameterizedTest
@MethodSource("provideTestCasesCheckWordExistence")
void testCheckWordExistence(HashMap<String, Integer> wordList, String inputWord, boolean expectedResult) {
    Occorrenze occorrenze = new Occorrenze();
    boolean result = occorrenze.checkWordExistence(wordList, inputWord);
    assertEquals(expectedResult, result);
}

1 usage
private static Stream<Arguments> provideTestCasesCheckWordExistence() {
    return Stream.of(
        Arguments.of(
            new HashMap<String, Integer>() {{
                put("apple", 2);
                put("banana", 3);
                put("cherry", 1);
            }}, "banana", true),
        Arguments.of(
            new HashMap<String, Integer>() {{
                put("apple", 2);
                put("banana", 3);
                put("cherry", 1);
            }}, "orange", false),
        Arguments.of(
            new HashMap<String, Integer>(), "grape", false),
        Arguments.of(
            null, "pear", false),
        Arguments.of(
            new HashMap<String, Integer>() {{
                put("dog", 5);
                put("cat", 2);
                put("fish", 7);
            }}, null, false),
        Arguments.of(
            new HashMap<String, Integer>() {{
                put("dog", 5);
                put("cat", 2);
                put("fish", 7);
            }}, "", false)
    );
}
```

```

✓ Tests passed: 7 of 7 tests - 34 ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

```

LEGGERE L'IMPLEMENTAZIONE

Dato che si tratta di un codice scritto da noi durante le lezioni di programmazione II, non è necessario sviluppare questo punto in quanto conosciamo l'implementazione.

ESEGUIRE TEST CON TOOL DI CODE COVERAGE

Abbiamo effettuato una analisi di code coverage con i test black-box effettuati nel primo homework e abbiamo notato che il codice risulta coperto al 98%, in quanto una riga di codice risulta parzialmente coperta. Ciò non vuol dire che è stato sicuramente testato tutto al meglio, ma può darci un'indicazione sulle parti che sono state interessate dai nostri test. Abbiamo provato ad aggiungere casi di test e a modificare le condizioni del metodo senza avere risultati; quindi, la line coverage per le righe rimane al 98%. Lo stesso vale per la branch coverage.

Element	Class, %	Method, %	Line, % ▾	Branch, %
✓ org.example	100% (2/2)	100% (8/8)	98% (55/56)	94% (51/54)
Occorrenze	100% (2/2)	100% (8/8)	98% (55/56)	94% (51/54)

```

88      // Numeri prima dei caratteri speciali
89      if (Character.isDigit(str1.charAt(0))) {
90          return -1;
91      } else if (Character.isDigit(str2.charAt(0))) {
92          return 1;
93      }
94      //Caratteri speciali dopo i numeri e prima dei segni di punteggiatura
95      if (!Character.isLetterOrDigit(str1.charAt(0))) {
96          return str2.compareToIgnoreCase(str1); // Inverti l'ordine
97      } else {
98          return str2.compareToIgnoreCase(str1); // Inverti l'ordine
99      }
100  }
101  });

```

```

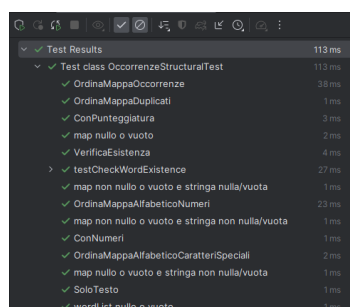
73      str1 = (str1 == null) ? "" : str1;
74      str2 = (str2 == null) ? "" : str2;

```

```

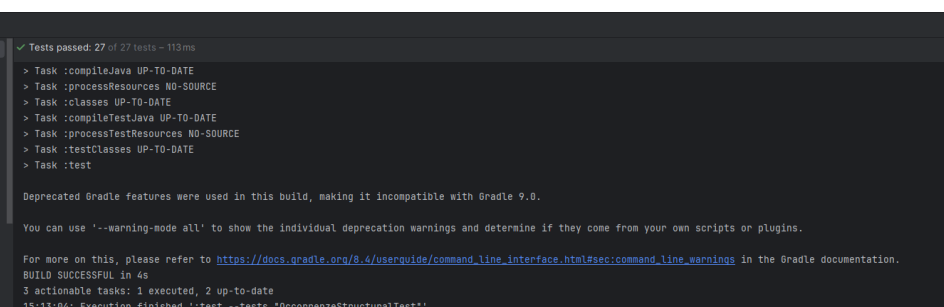
101      if (!Character.isLetterOrDigit(str1.charAt(0))) {
102          return str2.compareToIgnoreCase(str1);
103      } else {
104          return str2.compareToIgnoreCase(str1);
105      }
106  }
107  });

```



Test Results 113 ms

- ✓ Test class OccorrenzeStructuralTest 113 ms
 - ✓ OrdinaMappeOccorrenze 38 ms
 - ✓ OrdinaMappeDuplicati 1 ms
 - ✓ ConPunteggiatura 3 ms
 - ✓ map nullo o vuoto 2 ms
 - ✓ VerificaEsistenza 4 ms
 - ✓ testCheckWordExistence 27 ms
 - ✓ map non nullo o vuoto e stringa nulla/vuota 1 ms
 - ✓ OrdinaMappeAlfabeticoNumeri 23 ms
 - ✓ map non nullo o vuoto e stringa non nulla/vuota 1 ms
 - ✓ ConNumeri 1 ms
 - ✓ OrdinaMappeAlfabeticoCaratteriSpeciali 2 ms
 - ✓ map nullo o vuoto e stringa non nulla/vuota 1 ms
 - ✓ SoloTesto 1 ms
 - ✓ wordList nullo o vuoto 1 ms



Tests passed: 27 of 27 tests - 113 ms

```

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.4/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 4s
3 actionable tasks: 1 executed, 2 up-to-date
15:13:04: Execution finished '--tests "OccorrenzeStructuralTest"'.

```



OccorrenzeStructuralTest: 27 total, 27 passed

113 ms

[Collapse](#) | [Expand](#)

Test class OccorrenzeStructuralTest		113 ms
OrdinaMappaOccorrenze	passed	38 ms
OrdinaMappaDuplicati	passed	1 ms
ConPunteggiatura	passed	3 ms
map nullo o vuoto	passed	2 ms
VerificaEsistenza	passed	4 ms
testCheckWordExistence		27 ms
[1] wordList={banana=3, apple=2, cherry=1}, inputWord=banana, expectedResult=true	passed	21 ms
[2] wordList={banana=3, apple=2, cherry=1}, inputWord=orange, expectedResult=false	passed	1 ms
[3] wordList={}, inputWord=grape, expectedResult=false	passed	1 ms
[4] wordList=null, inputWord=pear, expectedResult=false	passed	1 ms
[5] wordList={cat=2, fish=7, dog=5}, inputWord=null, expectedResult=false	passed	1 ms
[6] wordList={cat=2, fish=7, dog=5}, inputWord=, expectedResult=false	passed	1 ms
[7] wordList={@@@=3, ###=2, \$\$\$=1}, inputWord=@@@, expectedResult=true	passed	1 ms
map non nullo o vuoto e stringa nulla/vuota	passed	1 ms
OrdinaMappaAlfabeticoNumeri	passed	23 ms
map non nullo o vuoto e stringa non nulla/vuota	passed	1 ms
ConNumeri	passed	1 ms
OrdinaMappaAlfabeticoCaratteriSpeciali	passed	2 ms
map nullo o vuoto e stringa non nulla/vuota	passed	1 ms
Solo Testo	passed	1 ms
wordList nullo o vuoto	passed	1 ms
TestoRomanzo nullo o di lunghezza 0	passed	1 ms
OrdinaMappaAlfabeticoNormale	passed	1 ms
OrdinaMappaEntrambi	passed	1 ms
OrdinaMappaZero	passed	1 ms
map nullo o vuoto e stringa nulla/vuota	passed	1 ms
ConPunteggiaturaeNumeri	passed	1 ms
OrdinaMappaAlfabeticoPunteggiatura	passed	1 ms

HOMework 2

SCELTA CODICE

Per il secondo homework abbiamo deciso di utilizzare la stessa classe in quanto riteniamo sia possibile effettuare property based testing sulla stessa.

PROPRIETÀ DELL'INPUT

- *createWordList*
 - o Verifichiamo che testoRomanzo sia nullo o meno
- *sortByOccurrences*
 - o Verifichiamo che wordList contenga valori duplicati o nulli oppure sia nulla o vuota, oppure non disponga di nessuna di queste caratteristiche.
- *sortByAlphabeticalOrder*
 - o Verifichiamo che wordList contenga chiavi numeriche, con segni di punteggiatura o con caratteri speciali, oppure sia nulla o vuota, oppure non disponga di nessuna di queste caratteristiche (solo lettere).
- *checkWordExistence*
 - o Verifichiamo che inputWord e wordList siano nulli o vuoti.

PROPRIETÀ DELL'OUTPUT

- *sortByAlphabeticalOrder*
 - o Dopo aver utilizzato il metodo, la TreeMap ottenuta deve avere elementi invariati rispetto a wordList.
 - o Viene collezionata la statistica "Null maps" se la TreeMap è nulla.
 - o Viene collezionata la statistica "Empty maps" se la TreeMap è vuota.
 - o Viene collezionata la statistica "Contains a numeric key" se ci sono chiavi numeriche
 - o Viene collezionata la statistica "Contains a punctuation signs key" se ci sono chiavi composte da segni di punteggiatura
 - o Viene collezionata la statistica "Contains a special characters key" se ci sono chiavi composte da caratteri speciali
- *sortByOccurrences*
 - o Dopo aver utilizzato il metodo, la TreeMap ottenuta deve avere elementi invariati rispetto a wordList.
 - o Viene collezionata la statistica "Null maps" se la TreeMap è nulla.
 - o Viene collezionata la statistica "Empty maps" se la TreeMap è vuota.
 - o Viene collezionata la statistica "Contains null values" o "Contains duplicate values" se wordList contiene valori nulli o duplicati.
- *checkWordExistence*
 - o Viene collezionata la statistica "Key found" o "Key not found" a seconda del risultato del metodo e "Empty inputWord" per la parola vuota, "Empty wordList" per la lista vuota, "Null inputWord" per la parola nulla, "Null wordList" per la lista nulla.
- *createWordList*
 - o In caso di testoRomanzo nullo o vuoto, il valore di ritorno deve essere nullo. Viene collezionata la statistica "testoRomanzo is null" o "testoRomanzo is not null" a seconda dei casi.

LASCIARE CHE SIA IL FRAMEWORK A SCEGLIERE I TEST

- *createWordList*

Questo test, denominato `createWordListPrPTest`, è un esempio finalizzato a valutare il comportamento del metodo `createWordList` quando viene fornito un input di tipo `String` rappresentante un romanzo o un testo letterario. Il test genera casualmente una stringa (`testoRomanzo`) composta da caratteri alfabetici, numerici, spazi bianchi e caratteri ASCII, con lunghezza variabile tra 0 e 200 caratteri. È anche possibile che la stringa sia nulla con una probabilità del 10%. Successivamente, il test chiama il metodo `createWordList` con la stringa generata come input e verifica il risultato. Se il risultato è nullo, viene registrata una statistica indicando che il testo del romanzo è nullo; in caso contrario, viene registrata una statistica indicando che il testo del romanzo non è nullo.

Questo test è progettato per coprire una gamma diversificata di scenari, testando la capacità del metodo `createWordList` di gestire input di testo romanzo di varie lunghezze e composizioni. La presenza di input nullo è inclusa per esplorare il comportamento del metodo in tali situazioni. La raccolta di statistiche è utilizzata per tracciare e analizzare il comportamento del metodo in base alle diverse condizioni di input.

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void createWordListPrPTest(
    /* We generate a list with 100 numbers, ranging from -1000 to 1000. Range chosen randomly. */
    @ForAll("arbitrarytestoRomanzo") String testoRomanzo
) {
    HashMap<String, Integer> result = occorrenze.createWordList(testoRomanzo);
    if (result == null) {
        Statistics.collect(Values.of("TestoRomanzo is null"));
    } else {
        Statistics.collect(Values.of("TestoRomanzo is not null"));
    }
}

no usages
@Provide
Arbitrary<String> arbitrarytestoRomanzo() {
    return Arbitraries.strings().alpha().numeric().whitespace().ascii().ofMinLength(0).ofMaxLength(200).injectNull( nullProbability: 0.1);
}

timestamp = 2024-01-21T10:54:29.879091400, [OccorrenzePropertyTest:createWordListPrPTest] (1000) statistics =
# | label | count |
-----|-----|-----|
0 | TestoRomanzo is not null | 837 | #####
1 | TestoRomanzo is null | 163 | #####
```

- *sortByOccurrences*

Il test `sortByOccurrencesPrPTest` è implementato per valutare il corretto funzionamento del metodo `sortByOccurrences`. Il test genera casualmente una mappa (`wordList`) contenente chiavi rappresentate da stringhe ("Key0", "Key1", ..., "Key99") e valori interi compresi tra 1 e 1000, con una probabilità del 10% di inserire valori nulli. Se la mappa è nulla, il risultato atteso è che anche la mappa ordinata (`sortedList`) sia nulla, e viene raccolta una statistica indicando la presenza di mappe nulle. Se la mappa non è nulla, il test ordina la mappa originale (`wordList`) e la mappa risultante (`sortedList`) in base alle chiavi e verifica che siano uguali. In caso di uguaglianza, viene raccolta una statistica indicando che non ci sono differenze tra le mappe ordinate e non ordinate. Successivamente, il test verifica la presenza di valori nulli o duplicati all'interno dei valori della mappa. Se vengono rilevati valori nulli, viene registrata una statistica corrispondente. Se vengono rilevati valori duplicati, viene registrata una statistica relativa a duplicati. In caso contrario, viene registrata una statistica indicando l'assenza di valori nulli o duplicati. Questo test è progettato per esplorare il comportamento del metodo `sortByOccurrences` in situazioni variegiate, considerando diversi scenari di input, inclusi casi con mappe nulle, mappe non nulle con valori casuali e la presenza di valori nulli o duplicati all'interno delle mappe. Le statistiche raccolte offrono un'analisi dettagliata del comportamento del metodo sotto diverse condizioni di input.



```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void sortByOccurrencesPrPTest(
    /* We generate a list with 100 numbers, ranging from -1000 to 1000. Range chosen randomly. */
    @ForAll("arbitrarywordList") HashMap<String, Integer> wordList
) {
    TreeMap<String, Integer> sortedList = occorrenze.sortByOccurrences(wordList);
    if (wordList == null) {
        // If wordList is null, sortedList should also be null
        assertNull(sortedList);
        Statistics.collect(...values: "Null maps");
    } else if (wordList.isEmpty()) {
        Statistics.collect(...values: "Empty maps");
    }

    if (sortedList != null && !sortedList.isEmpty()) {
        List<Map.Entry<String, Integer>> wordListEntries = wordList.entrySet().stream()
            .sorted(Map.Entry.comparingByKey())
            .collect(Collectors.toList());

        List<Map.Entry<String, Integer>> sortedListEntries = sortedList.entrySet().stream()
            .sorted(Map.Entry.comparingByKey())
            .collect(Collectors.toList());

        assertEquals(wordListEntries, sortedListEntries);
        boolean noNullsOrDuplicates = true;
        for (Integer value : wordList.values()) {
            if (value == null) {
                Statistics.collect(...values: "Contains null values");
                noNullsOrDuplicates = false;
                break;
            } else if (value.equals(wordList.values().iterator().next())) {
                Statistics.collect(...values: "Contains duplicate values");
                noNullsOrDuplicates = false;
            }
        }

        if (noNullsOrDuplicates) {
            Statistics.collect(...values: "No nulls or duplicates");
        }
    }
}

no usages
@Provide
Arbitrary<HashMap<String, Integer>> arbitrarywordList() {
    return Arbitraries.oneOf(
        // Generate a non-null map
        Arbitraries.integers().between(1, 1000).injectNull( nullProbability: 0.1)
            .filter(size -> size != null)
            .map(size -> {
                HashMap<String, Integer> wordList = new HashMap<>();
                for (int i = 0; i < size; i++) {
                    Integer randomValue = Arbitraries.integers().between(1, 1000).injectNull( nullProbability: 0.1).sample();
                    wordList.put("Key" + i, randomValue);
                }
                return wordList;
            }),
        // Generate an empty map
        Arbitraries.just(new HashMap<String, Integer>()),
        // Generate a null map
        Arbitraries.just( value: null)
    );
}

timestamp = 2024-01-21T12:59:19.312401900, [OccorrenzePropertyTest:sortByOccurrencesPrPTest] (1000) statistics =
# | Label | count |
-----|-----|-----|
0 | Contains duplicate values | 321 | #####
1 | Contains null values | 35 | #####
2 | Empty maps | 316 | #####
3 | Null maps | 328 | #####
```

- *checkWordExistence*

Il test `checkWordExistencePrPTest` mira a verificare la corretta implementazione del metodo `checkWordExistence`. Il test riceve due input generati casualmente:

- Una mappa (`wordList`) contenente chiavi rappresentate da stringhe alfabetiche, numeriche, spazi bianchi e caratteri ASCII, con valori interi fissati a 42. La mappa può anche essere nulla.
- Una stringa (`inputWord`) rappresentante una parola generata casualmente, composta da caratteri alfabetiche, numerici, spazi bianchi e caratteri ASCII, con una probabilità del 10% di essere nulla.

Successivamente, il test chiama il metodo `checkWordExistence` con la mappa e la stringa generate come input e verifica il risultato. Se il risultato è `true`, viene raccolta una statistica indicando che la chiave è stata trovata; se il risultato è `false`, viene raccolta una statistica indicando che la chiave non è stata trovata. Questo test è progettato per coprire una gamma diversificata di scenari, esplorando la capacità del metodo `checkWordExistence` di gestire mappe con chiavi variegata e la possibilità di trovare o non trovare una chiave specifica. L'inclusione di mappe nulle e la possibilità di una stringa di input nulla aggiungono ulteriori casi da considerare. Le statistiche raccolte offrono una traccia delle chiavi trovate o non trovate durante l'esecuzione del test, permettendo un'analisi del comportamento del metodo in base alle diverse condizioni di input.

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void checkWordExistencePrPTest(
    /* We generate a list with 100 numbers, ranging from -1000 to 1000. Range chosen randomly. */
    @ForAll("arbitrarywordListAlphabetical") HashMap<String, Integer> wordList,
    @ForAll("arbitrarytestoRomanzo") String inputWord
) {
    if (wordList == null) {
        Statistics.collect(...values: "Null wordList");
    } else if (wordList.isEmpty()) {
        Statistics.collect(...values: "Empty wordList");
    }
    if (inputWord == null) {
        Statistics.collect(...values: "Null inputWord");
    } else if (inputWord.isEmpty()) {
        Statistics.collect(...values: "Empty inputWord");
    }
    boolean result = occorrenze.checkWordExistence(wordList, inputWord);
    if (result) {
        Statistics.collect(...values: "Key found");
    } else {
        Statistics.collect(...values: "Key not found");
    }
}

timestamp = 2024-01-21T12:59:18.084418500, [OccorrenzePropertyTest:checkWordExistencePrPTest] (1801) statistics =
# | label | count |
-----|-----|-----|
0 | Empty inputWord | 37 | ##
1 | Empty wordList | 305 | #####
2 | Key found | 5 |
3 | Key not found | 995 | #####
4 | Null inputWord | 98 | #####
5 | Null wordList | 361 | #####
```

- *sortByAlphabeticalOrder*

Il test `sortByAlphabeticalOrderPrPTest` verifica il corretto ordinamento alfabetico di una mappa di stringhe. Il test riceve una mappa (`wordList`) generata casualmente contenente chiavi rappresentate da stringhe alfabetiche, numeriche, spazi bianchi e caratteri ASCII, con valori interi fissati a 42. La mappa può anche essere nulla. Successivamente, il test

chiama il metodo `sortByAlphabeticalOrder` per ordinare la mappa in base alle chiavi e ottiene una nuova mappa ordinata (`sortedList`).

Il test include diverse verifiche:

- i. Mappe Null o empty: Se la mappa originale (`wordList`) è nulla o vuota, ci si aspetta che anche la mappa ordinata (`sortedList`) sia nulla. In tal caso, viene registrata una statistica indicante la presenza di mappe nulle o vuote.
- ii. Confronto tra Mappe Ordinate e Non Ordinate: Se la mappa originale non è nulla, il test verifica che le mappe ordinate e non ordinate siano uguali dopo la conversione in liste ordinate.
- iii. Analisi delle Chiavi: Se la mappa originale non è nulla, il test effettua ulteriori controlli sulle chiavi della mappa. Verifica se le chiavi contengono caratteri speciali, sono numeriche o contengono segni di punteggiatura. In base a queste condizioni, vengono registrate statistiche aggiuntive.

Le statistiche raccolte offrono una panoramica dettagliata del comportamento del metodo `sortByAlphabeticalOrder` in diverse situazioni. In particolare, le statistiche forniscono informazioni sulla gestione di mappe nulle, sul corretto ordinamento alfabetico e su caratteristiche specifiche delle chiavi presenti nella mappa.

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void sortByAlphabeticalOrderPrpTest(
    /* We generate a list with 100 numbers, ranging from -1000 to 1000. Range chosen randomly. */
    @ForAll("arbitraryWordListAlphabetical") HashMap<String, Integer> wordList
) {
    TreeMap<String, Integer> sortedList = occorrenze.sortByAlphabeticalOrder(wordList);

    if (wordList == null) {
        // If wordList is null, sortedList should also be null
        assertNull(sortedList);
        Statistics.collect(...values: "Null maps");
    } else if (wordList.isEmpty()) {
        Statistics.collect(...values: "Empty maps");
    }

    if (sortedList != null && !sortedList.isEmpty()) {
        // Convert both maps to sorted lists before comparison
        List<Map.Entry<String, Integer>> wordListEntries = wordList.entrySet().stream()
            .sorted(Map.Entry.comparingByKey())
            .collect(Collectors.toList());

        List<Map.Entry<String, Integer>> sortedListEntries = sortedList.entrySet().stream()
            .sorted(Map.Entry.comparingByKey())
            .collect(Collectors.toList());

        assertEquals(wordListEntries, sortedListEntries);
        boolean notNormal = true;
        for (String key : wordList.keySet()) {
            try {
                // Attempt to parse the key as a numeric value
                Double.parseDouble(key);
            } catch (NumberFormatException e) {
                // Key is not a numeric value, collect statistics or perform other checks
                Statistics.collect(...values: "Contains a numeric key");
                notNormal = false;
            }
        }
    }
}
```

```

140         if (!key.matches( regex: "[a-zA-Z0-9]+")) {
141             // Key contains special characters, collect statistics or perform other checks
142             Statistics.collect( ...values: "Contains a special characters key");
143             notNormal = false;
144         }
145         if (Pattern.compile( regex: "\\p{Punct}").matcher(key).find()) {
146             // Key contains punctuation signs, collect statistics or perform other checks
147             Statistics.collect( ...values: "Contains a punctuation signs key");
148             notNormal = false;
149         }
150     }
151     if (notNormal) {
152         Statistics.collect( ...values: "Contains letters only");
153     }
154 }
155 }
156
@Provide
Arbitrary<HashMap<String, Integer>> arbitrarywordListAlphabetical() {
    return Arbitraries.oneOf(
        // Generate a non-null map
        Arbitraries.strings() StringArbitrary
            .alpha()
            .numeric()
            .whitespace()
            .ascii()
            .ofMinLength( 0)
            .ofMaxLength( 25)
            .list() ListArbitrary<String>
            .uniqueElements()
            .ofSize(25)
            .map(strings -> {
                HashMap<String, Integer> wordList = new HashMap<>();
                for (String str : strings) {
                    // Rimuovi i caratteri speciali e converti le lettere in minuscolo
                    str = str.toLowerCase();
                    // Assegna un valore intero casuale o costante, come necessario
                    wordList.put(str, 42);
                }
                return wordList;
            }),
        // Generate a null map
        Arbitraries.just( value: null),
        // Generate an empty map
        Arbitraries.just(new HashMap<String, Integer>())
    );
}

timestamp = 2024-01-21T12:59:19.817239900, [OccorrenzePropertyTest:sortByAlphabeticalOrderPrPTest] (22659) statis
# | label | count |
-----|-----|-----|
0 | Contains a numeric key | 8223 | #####
1 | Contains a punctuation signs key | 6050 | #####
2 | Contains a special characters key | 7720 | #####
3 | Empty maps | 335 | ###
4 | Null maps | 331 | ###

```

STATISTICA: MOTIVO UTILIZZO ISTOGRAMMA

Il motivo principale per utilizzare l'istogramma per le statistiche in questo contesto è che fornisce una rappresentazione visuale della distribuzione dei dati in intervalli discreti. In particolare, l'istogramma è utile quando si desidera analizzare la frequenza delle occorrenze dei dati in diverse categorie o bin.

In questo codice si sta lavorando con dati come la lunghezza del testo (testoRomanzo), le occorrenze delle parole (wordList), e le parole stesse (inputWord). Utilizzando l'istogramma come formato di rappresentazione delle statistiche, si ottiene una visione chiara e intuitiva di come queste variabili si distribuiscano in diverse categorie.

COVERAGE

Analizzando la coverage, abbiamo notato che la sezione non inclusa nello specification based testing non viene affrontata nemmeno con il property based testing, in quanto non siamo riusciti a identificare il problema alla base di questa mancata copertura.

Element	Class, %	Method, %	Line, %	Branch, %
org.example	100% (2/2)	100% (8/8)	98% (55/56)	94% (51/54)
Occorrenze	100% (2/2)	100% (8/8)	98% (55/56)	94% (51/54)

Test Results

2 sec 989 ms

Test class O

2 sec 989 ms

checkWor

1 sec 236 ms

sortByOccurre

925 ms

sortByAlphabe

703 ms

createWordList

125 ms

Tests passed: 4 of 4 tests - 2 sec 989 ms

```

Key5*=32,
  *Key17*=10,
  *Key4*=277,
  *Key18*=1,
  *Key3*=411,
  *Key19*=18,
  *Key9*=1,
  *Key8*=7,
  *Key7*=28
}
start: 11
end: 15

timestamp = 2024-01-21T10:58:23.875507900, generated =
wordList=

```

OccorrenzePropertyTest: 4 total, 4 passed

2.99 s

Collapse | Expand

Test class OccorrenzePropertyTest

2.99 s

checkWordExistencePrPTest

passed

1.24 s

sortByOccurrencesPrPTest

passed

925 ms

sortByAlphabeticalOrderPrPTest

passed

703 ms

createWordListPrPTest

passed

125 ms

Generated by IntelliJ IDEA on 24/01/2024, 10:59