Exercises

W05H01 - PinguPinguLos III - Die Rache der Suchmaschine

Points	Submission due date	Status	Difficulty	Categories
0 / 10	in 4 days	No graded result		Hausaufgabe

Tasks:

Suchmaschine - Aller guten Dinge sind 3

Wichtig!

- Die Tests sind wieder in Struktur und Verhalten aufgeteilt. Lies dir also erst die gesamte Aufgabenstellung gründlich durch und füge zuerst alle benötigten Klassen und Signaturen hinzu.
- In der gesamten Aufgabe ist das Benutzen der `java.util.Arrays` Klasse verboten. Im Paket `java.util` gibt es außerdem noch viele weitere Klassen mit ähnlichen Funktionalitäten, über die wir später im Semester noch sprechen werden. Verwende daher keine Klassen aus `java.util`.
  Abgaben, die gegen das Verbot verstoßen, werden mit 0 Punkten bewertet.

Das Ziel

Auch diese Woche soll die Suchmaschine erweitert werden. Wir wollen ein Maß für die Ähnlichkeit zweier Dokumente implementieren. Dazu nutzen wir das Verfahren aus der Vorlesung: Es sollen Arrays mit der Anzahl der vorkommenden Wörter der beiden Dokumente erstellt werden. Indem wir diese Arrays als Vektoren interpretieren, erhalten wir schlussendlich über das Skalarprodukt ein Indikator für die Ähnlichkeit der beiden Dokumente.

1. Die Klasse `WordCount`

Die Klasse WordCount wird dazu verwendet, die Häufigkeit spezifischer Wörter in einem Dokument zu speichern. Hierfür besitzt sie zwei private Attribute: `word` und `count`. Diese werden über den Konstruktor `WordCount(String, int)` der Klasse initialisiert, wobei word den Begriff und count die Häufigkeit repräsentiert. Sollte dabei ein negativer Wert für count übergeben werden, wird dieser automatisch auf 0 gesetzt.

Weil für bestimmte Zwecke `WordCount`-Objekte mit einem `count`-Wert von 0 erforderlich sind, gibt es zusätzlich einen zweiten Konstruktor `WordCount(String)`, der den Anfangswert von `count` auf 0 setzt.

Erstelle diese Klasse in `pgdp.searchengine.util` und implementiere zudem die öffentlichen Methoden `increment()`, welche den `count` eines `WordCount`-Objekts um genau 1 erhöht, sowie Getter für die beiden privaten Attribute.

◉ **Alle Konstruktoren, Methoden und Variablen in WordCount haben die korrekte Signatur** No results

◉ **Implementiere die Konstruktoren und Methoden von WordCount** No results

2. Die Klasse Document

Die Klasse `Document` soll nun um ein paar Methoden erweitert werden, die unsere neue Klasse `WordCount` benutzen und uns erlauben `Document`-Objekte zu analysieren und zu vergleichen.

◉ **Alle Methodensignaturen in Document sind korrekt** No results

2.1. Das WordCount Array eines `Document`-Objekts

Im ersten Schritt wollen wir den Inhalt des Dokuments in `WordCount`-Objekte herunterbrechen. Die Methode `getWordCountArray(): WordCount[]` soll den `content` des `Document`-Objekts in ein Array aus `WordCount`-Objekten umwandeln. Dazu soll die öffentliche Methode `getWordCountArray(): WordCount[]` zur Klasse `Document` hinzugefügt werden. Diese soll für jedes Wort im String `content` mithilfe der `WordCount`-Klasse die Häufigkeit des Wortes in einem Array abspeichern. Dieses Array soll am Ende zurückgegeben werden.

Achte darauf, dass alle hinzugefügten Wörter kleingeschrieben sind und kein neues `WordCount`-Objekt für ein Wort aufgrund eines Sonderzeichens erstellt wird. Dabei beschränken wir uns auf folgende Sonderzeichen: `.` (Punkt), `;` (Semikolon), `!` (Ausrufezeichen), `?` (Fragezeichen), `(` (runde, öffnende Klammer), `)` (runde, schließende Klammer), `*` (Sternchen) und `,` (Komma).

Beispiel: `Pinguin`, `Pinguin!`, `?Ping(uin!` sollen am Ende zum `WordCount`-Eintrag für `pinguin` zählen. Zwischen zwei Wörtern steht dabei immer mindestens ein Leerzeichen. Die Pinguine der PUM geben dir dabei folgenden Hinweis: Die Dokumentation der String Klasse hat viele praktische Methoden!

◉ **Implementiere die getWordCountArray-Methode in Document** No results

2.2. Zwei WordCount-Arrays abgleichen

Um am Ende das Skalarprodukt auf zwei WordCount-Arrays von zwei Dokumenten anwenden zu können, müssen wir sicherstellen, dass sie die gleichen Einträge an den gleichen Stellen enthalten. Um ein `WordCount`-Array mit den fehlenden Einträgen auffüllen zu können, soll die öffentliche Methode `static equalizeWordCountArray(WordCount[], WordCount[]): WordCount[]` in `Document` implementiert werden. Diese Methode soll für jedes Wort in Array `second` überprüfen, ob dieses Wort bereits in `first` vorhanden ist. Wenn ja, soll mit dem nächsten Wort aus `second` fortgefahren werden. Wenn nein, soll dieses Wort `first` hinzugefügt werden, aber mit `count` = 0, da das Wort nicht im Text vorhanden war. Am Ende enthält `first` damit alle Wörter von `second`. Die genaue Reihenfolge der `WordCount`-Objekte spielt dabei keine Rolle, solange die beschriebene Logik erfüllt ist.

◉ **Implementiere die equalizeWordCountArray-Methode in Document** No results

2.3. Sortieren der WordCount-Arrays

Jetzt haben zwei Arrays zwar die gleichen Einträge, aber nicht zwangsläufig in der gleichen Reihenfolge bzw. am gleichen Index. Darum benötigen wir noch eine öffentliche Methode `static sort(WordCount[]): void`, die das übergebene Array lexikographisch nach dem Attribut `word` von `WordCount` sortiert. Welchen Sortieralgorithmus du verwendest, ist dir überlassen. Du kannst den Bubblesort-Algorithmus der P-Aufgabe anpassen oder einen anderen Algorithmus verwenden. Wichtig ist, dass du keine fertige Sortier-Funktion der Java-Library nutzen darfst, sondern den Sortieralgorithmus selbst implementierst!

◉ **Implementiere die sort-Methode in Document** No results

2.4. Ähnlichkeitsmaß

Als letzten Schritt wird eine öffentliche Methode `static similarity(WordCount[], WordCount[]): double` benötigt, die für zwei `WordCount`-Arrays entsprechend der Vorlesung das Ähnlichkeitsmaß berechnet (siehe Folien Kapitel 4 und 5). Wir sehen die Arrays als Vektoren an und wenden das Skalarprodukt auf beide an. Dieses Teilergebnis soll noch durch das Produkt der beiden Arraylängen geteilt werden. Falls die beiden Arrays eine unterschiedliche Länge besitzen, soll `-1` zurückgegeben werden. Du darfst in dieser Methode davon ausgehen, dass die übergebenen arrays schon sortiert sind, es bedarf hier also keiner Sortierung.

Wir definieren das Ähnlichkeitsmaß wie folgt`

Sei n = `first.length`, m = `second.length` und `first[i]` bzw. `second[i]` der `count` des `i`-ten `WordCount` Objekts.

$$\text{result} = \begin{cases} -1 & \text{falls } n \neq m, \\ 0 & \text{falls } n * m = 0, \\ \frac{\sum_{i=0}^{n-1} \text{first}[i] \cdot \text{second}[i]}{n \cdot m} & \text{andernfalls.} \end{cases}$$

◉ **Implementiere die similarity-Methode in Document** No results

2.5. Alles zusammenführen

Wir haben nun 4 Hilfsmethoden implementiert, die jeweils einen Schritt zur Berechnung des Ähnlichkeitsmaßes zweier Dokumente übernehmen. Alle 4 Methoden nacheinander aufzurufen ist uns aber noch zu umständlich. Wir benötigen eine öffentliche Methode `computeSimilarity(Document): double`. Über diese erhalten wir die Ähnlichkeit zwischen dem Dokument, auf welchem wir die Methode aufrufen, und dem übergebenen Dokument als Rückgabewert. Rufe dafür die bis hierhin implementieren Methoden in der Reihenfolge, in der sie implementiert werden, mussten mit passenden Eingaben auf.

◉ **Implementiere die computeSimilarity-Methode in Document** No results

3. Testen (Optional)

In dieser Aufgaben sollst du Tests für deine Implementierung schreiben. Schreibe diese in die `SearchEngineTesting`-Klasse. Die Klasse `WordCount` muss nicht getestet werden.

Für die neu implementierten Methoden in `Document` sollen pro Methode zwei Tests geschrieben werden: ein Test für einen allgemeinen, regulären Fall und einen Test für einen beliebigen Sonderfall.

Es soll über jedem Test ein Kommentar existieren, warum ihr die Methode auf diese Art und Weise testet, also warum euer Test aussagekräftig ist, und bei den Sonderfällen noch zusätzlich, warum es sich bei den gewählten Testdaten um einen Sonderfall handelt. Die Kommentare dürfen wie immer auch auf Englisch sein.

(Die Tests werden nicht automatisch getestet, sie dienen dir aber als eigenständige Übungsmöglichkeit)

Hinweise

- Für die Verarbeitung von Strings in dieser Aufgabe können folgende Methoden nützlich sein: replace(), replaceAll(), split() und compareTo(). Diese müssen jedoch nicht verwendet werden.

FAQ

Sobald die gleichen Fragen öfters vorkommen wird die Frage + Antwort hier hinzugefügt. Unklarheiten in der Aufgabenstellung werden sehr wahrscheinlich direkt in der Aufgabenstellung angepasst.

Exercise details

Release date	Nov 23, 2025 17:05
Submission due date	Nov 30, 2025 17:00
Complaint possible	No