P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2017/

unique-chars

Anzahl verschiedener Zeichen

Woche 04 Aufgabe 4/4

Herausgabe: 2017-05-15
Abgabe: 2017-05-26

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `unique-chars`
Package `uniquechars`
Klassen

Main

`public static int uniqueChars(Map<String, Integer> cache, String input)`
 	
`public static List<Integer> allUniqueChars(`
`  Map<String, Integer> cache,`
`  List<String> input)`
 	
`public static List<Integer> allUniqueChars(List<String> input)`

Die Funktion `uniqueChars` berechnet, wie viele *unterschiedliche* Zeichen im String `input` vorkommen. Dabei benutzt sie einen *Cache* in Form einer `String`-zu-`Integer` Map. Das heißt, wenn das Ergebnis für einen bestimmten String schon in `cache` steht, soll dieses Ergebnis übernommen und nicht neu berechnet werden. (Zeichen die sich nur in der Groß-Kleinschreibung unterscheiden sind auch unterschiedlich).

Die Funktion `allUniqueChars` berechnet jeweils die Anzahl der unterschiedlichen Zeichen in den Strings der Liste `input` und gibt diese in einer Liste von `Integern` zurück. Das heißt, die Ergebnisliste hat die gleiche Länge wie `input` und an der Stelle i des Ergebnisses steht die Anzahl der unterschiedlichen Zeichen des Strings von `input.get(`i`)`.

Die Funktion `allUniqueChars` hat zwei überladenen Versionen. Die erste Version nimmt eine Map `cache` als Argument, die für alle Berechnungen verwendet werden soll, analog zum `cache` in `uniqueChars`. Die zweite Version soll intern einen frischen Cache erzeugen und dann diesen für die Berechnungen verwenden.

Diese Aufgabe soll gelöst werden, ohne unnötig Code zu duplizieren. Diese Anforderung fließt in die Bewertung der Code-Qualität ein.

Beispielaufruf 01:

```
Map<String, Integer> m = new HashMap<>();
uniqueChars(m, "Hhelloo");
```

ergibt `5`

Beispielaufruf 02:

```
Map<String, Integer> m = new HashMap<>();
m.put("hhelloo", 5);
uniqueChars(m, "hhelloo");
```

ergibt `5`

Beispielaufruf 03:

```
List<String> l = new ArrayList<String>();
l.add("hhelloo");
l.add("wwoorrld");
Map<String, Integer> m = new HashMap<String, Integer>();
allUniqueChars(m, l);
```

ergibt `[4, 5]` (als `List<Integer>`)

Hinweise

- Links zu Collections, Lists und Maps:
  - `https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html`
  - `https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html`
  - `https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html`
- Der Typ einer Collection wie `List<Integer>` trägt den Typ der enthaltenen Elemente in spitzen Klammern. Soll der Inhalt von primitiven Typ sein, wie `int`, muss man diesen allerdings „ausschreiben" (`Integer`). (Wen die Details dahinter interessieren: `https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html`)
- Tipp zum Testen: Collections lassen sich mit ihrer `toString`-Methode gut darstellen und ausdrucken.