



PSE Vorkurs Tag 5

Heute Abend UNO!

Feedback

Bitte füllt den Feedbackbogen aus, damit wir den Vorkurs für die nächsten Jahre verbessern können. Der Bogen ist anonym und dauert diesmal ein bisschen länger.

Ihr findet ihn unter: <https://forms.gle/HhisY91RZc8eDkPy5>

Musterlösungen

Tag 4 Lösungen

Präsentations-Cheatsheet

Map Funktionen:

Methode	Beschreibung
<code>put(k, v)</code>	Fügt einen neuen Schlüssel-Wert-Paar ein
<code>get(k)</code>	Gibt den Wert zu einem Schlüssel zurück
<code>remove(k)</code>	Entfernt einen Eintrag
<code>containsKey(k)</code>	Prüft, ob Schlüssel existiert
<code>keySet()</code>	Gibt alle Schlüssel zurück
<code>values()</code>	Gibt alle Werte zurück
<code>size()</code>	Gibt die Anzahl der Einträge zurück

Map-Syntax:

```
import java.util.HashMap;

HashMap<String, Integer> studis = new HashMap<>();
studis.put("Melanie", 1976370);
studis.put("Paul", 1249609);

System.out.println(studis.get("Melanie"));
```

Aufgabe 1: Maps

Maps verbinden ein Schlüsselement mit einem Wert. Diese können auch unterschiedliche Datentypen haben. Ein Telefonbuch könnte beispielsweise als Map dargestellt werden um Namen mit Telefonnummern zu verbinden:

```
Map<String, Long> telefonbuch = new HashMap<String, Long>();
//erstes Parameter ist der Schlüssel, zweites ist der Wert
telefonbuch.put("Paul Griller", 1374927394);
telefonbuch.put("Melanie Barbecue", 9257632984);
```

1. Schreibe eine Funktion die einen Dictionary entgegennimmt und alle Schlüssel und die dazugehörigen Werte ausgibt:

```
Paul Griller: 1374927394, Melanie Barbecue: 9257632984
```

2. Schreibe eine Funktion die einen String entgegennimmt und in einem Dictionary speichert, wie oft jeder Buchstabe darin vorkommt.

```
grimmige griller grillen glühend gern grobe grillgemüse-spieße.
```

wird mithilfe der Ausgabe-Funktion aus Aufgabe 1.1 zu:

```
g: 9, r: 7, i: 6, m: 3, e: 10, : 6, l: 7, n: 3, ü: 2, h: 1, d: 1, o: 1, b: 1,
s: 2, -: 1, p: 1, ß: 1, .: 1
```

Hinweis: Verwende die `split()` Funktion auf einem String um ihn in ein Zeichenarray umzuwandeln. Mehr dazu findest du hier: <https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String->.

3. Erweitere deine Funktion, sodass sie zusätzlich zum String einen Integer `length` entgegennimmt und statt einzelne Buchstaben, jetzt Buchstabenfolgen der Länge `length` zählt.

```
grimmige griller grillen glühend gern grobe grillgemüse-spieße.
```

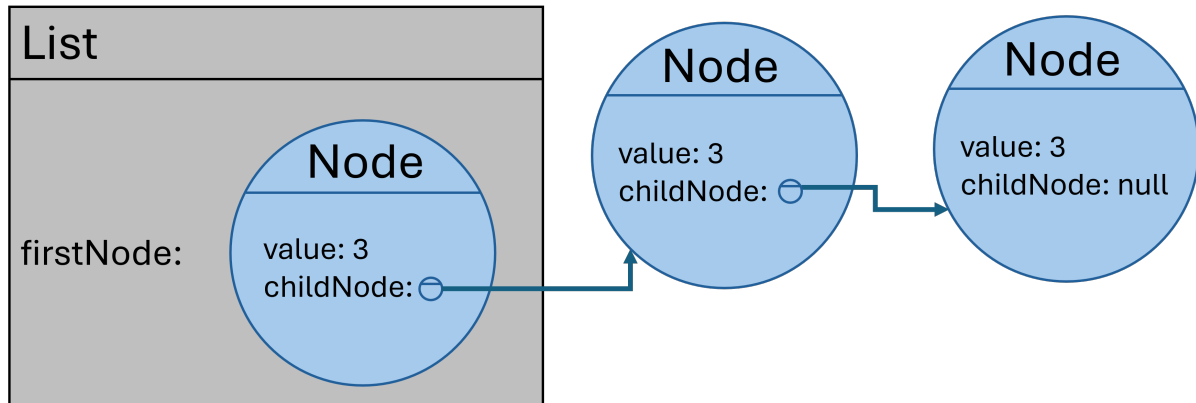
wird bei `length = 3` zu:

```
gri: 4, rim: 1, imm: 1, mmi: 1, mig: 1, ige: 1, ge : 1, e g: 2, gr: 4, ril:
3, ill: 3, lle: 2, ler: 1, er : 1, r g: 1, len: 1, en : 1, n g: 2, gl: 1,
glü: 1, lüh: 1, ühe: 1, hen: 1, end: 1, nd : 1, d g: 1, ge: 1, ger: 1, ern:
1, rn : 1, gro: 1, rob: 1, obe: 1, be : 1, llg: 1, lge: 1, gem: 1, emü: 1,
müs: 1, üse: 1, se-: 1, e-s: 1, -sp: 1, spi: 1, pie: 1, ieß: 1, eße: 1, ße.:
1
```

Aufgabe 2: Datenstrukturen selber erstellen

In Java kann jederzeit eine neue Datenstruktur gebaut werden, die dann auch an die Eigenheiten der Problemstellung angepasst sein kann.

Ihr habt ja sicher schon gemerkt, dass Arrays etwas sperrig zu nutzen sind, da deren Größe komplett statisch ist. Im Folgenden bauen wir eine Listen-Datenstruktur, welche dieses Problem löst. Sie soll aus einzelnen Knotenobjekten bestehen, die dann untereinander verbunden werden.

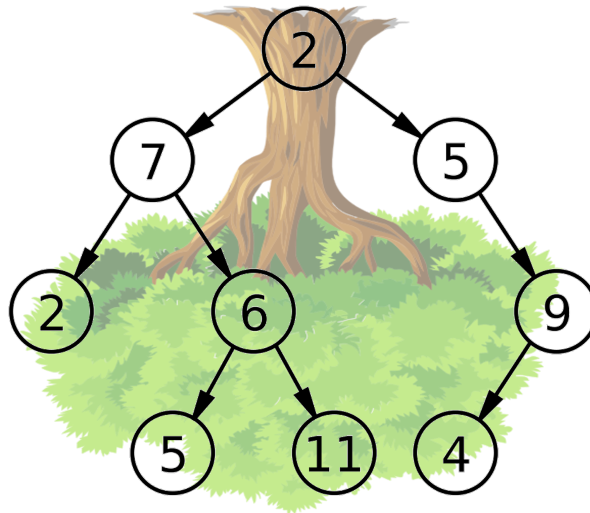


1. Erstelle eine Knotenklasse namens `Node`. Jede Knotenklasse soll einen Wert `value` vom Typ `int`, sowie einen Folgeknoten `nextNode` vom Typ `Node` als Attribute haben. Erstelle einen sinnvollen Konstruktor.
2. Erstelle jetzt die Listenklasse. Jede `List` hat einen Startknoten `startNode`. Programmiere wieder einen sinnvollen Konstruktor.
3. Noch besteht jede dieser Listen nur aus einem einzelnen Startknoten. Füge um das zu ändern der `List` Klasse eine `add(Node newElement)` Funktion hinzu. Sie soll eine neue `Node` ans Ende der Liste anfügen.
4. Gib der `List` Klasse außerdem eine `remove(Node element)` Funktion. Achte darauf, dass dabei eventuelle Folgeelemente nicht "Elternlos" zurückgelassen werden.
5. Programmiere eine `printElements()` Funktion, die alle Elemente in der Liste anschaulich in der Konsole ausgibt.
6. In dieser Aufgabe soll die Summe der Werte unserer Liste berechnet werden. Schreibe dazu eine Funktion in der Klasse `Node`, die die bisherige Summe entgegennimmt, ihren eigenen Wert hinzuaddiert und sich dann im nächsten Knoten selbst aufruft. Wenn die Methode auf den letzten Knoten in der Liste stößt, soll die berechnete Gesamtsumme zurückgegeben werden. Füge zuletzt der Klasse „`List`“ eine Funktion hinzu, die die Summenfunktion im ersten Knoten (natürlich mit bisheriger Summe 0) aufruft.

Du hast gerade eine einfache `LinkedList` programmiert. `LinkedLists` sind sehr wichtig als Datenstruktur mit Variabler Größe und sie existieren auch in nativem Java: <https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html>

HIGHPERFORMER-Aufgabe: Bäume

Bäume sind auch eine wichtige Datenstruktur in Java. Bei einem Baum ist jeder Knoten mit Kindknoten verbunden, welche wiederum mit weiteren Kindknoten verbunden sind. So werden die Daten in einer Art Baumstruktur gespeichert. Binärbäume sind besondere Bäume, bei welchen jeder Knoten maximal zwei Kindknoten haben darf:



1. Erstelle eine `BinaerBaum`-Klasse, die einen Wurzelknoten besitzt, sowie eine `Knoten`-Klasse, die einen `int`-Wert, sowie genau zwei Kindknoten (`linkesKind` und `rechtesKind`) speichert. Wenn ein Knoten bspw. kein rechtes Kind hat soll der Wert von `rechtesKind` gleich `null` sein. Da jeder Knoten maximal zwei Kinder hat ist der Baum ein sogenannter binärer Baum.
2. Programmiere eine Funktion in der `BinaerBaum`-Klasse, die eine neue Zahl in den Baum einfügt. Diese Funktion soll den Baum Knoten für Knoten durchgehen und bei jedem Knoten zufällig zum rechten oder linken Kind "abbiegen". Sobald auf ein `null` gestoßen wird, wird der einzufügende Wert dort mit einem neuen Knoten hinzugefügt.
3. Programmiere eine Funktion in der `BinaerBaum`-Klasse, die den Baum anschaulich in der Konsole ausgibt.
4. Programmiere zwei Funktionen, eine die den Baum eindeutig in eine `LinkedList` (die aus Aufgabe 2.) umwandelt und eine die dasselbe umgekehrt macht. Überlege dir hierfür eine Möglichkeit wie die Struktur des Baums in der Liste abgebildet werden kann.
Hinweis: Du kannst deine Funktionen Testen indem du einen Baum zu einer `LinkedList` umwandeln lässt und dann mit der anderen Funktion zurück zu einem Baum umwandelst und dann prüfst ob die Struktur erhalten geblieben ist.
5. Programmiere eine Funktion `maxPathSum` in der `BinaerBaum`-Klasse, die einen `BinaerBaum` entgegennimmt und die maximale Pfadsumme für diesen Baum berechnet. Es werden alle Knoten entlang eines Pfades von oben nach unten addiert und die höchste Summe soll zurückgegeben werden. Bei dem obigen Beispielbaum wäre die maximale Pfadsumme 26 für den Pfad $2 - 7 - 6 - 11$.

Feedback

Bitte füllt den Feedbackbogen aus, damit wir den Vorkurs für die nächsten Jahre verbessern können. Der Bogen ist anonym und dauert diesmal ein bisschen länger.

Ihr findet ihn unter: <https://forms.gle/HhisY91RZc8eDkPy5>