Heinrich Heine Universität Düsseldorf

Programmierung – WS 2025/26
Dr. Markus Brenneis
Abgabefrist für Feedback:
15.12.25, 10 Uhr

9. Übungsblatt (Lösungsvorschlag[1])

Aufgabe 1: BugHunt: Generische Liste nach VL 15

N. hat eine Listen-Klasse umgeschrieben, sodass sie nicht mehr nur Integer speichern kann, sondern generisch ist. Allerdings funktioniert die Testklasse `ListPlayground` nicht richtig und auch die Klasse `List` lässt sich nicht kompilieren.

Korrigieren Sie die Fehler in beiden Klassen im Zusammenhang mit Generics, sodass die Liste wie erwartet funktioniert und es auch keine Compiler-Warnungen mehr gibt. Außerdem scheint es auch Logikfehler in den Methoden `toString` und `anyMatch` zu geben, die korrigiert werden müssen.

Nach Ihren Korrekturen sollte `ListPlayground` das Folgende auf der Standardausgabe ausgeben:

```
Die Länge der Liste ist: 3
Das erste Element ist: Hallo
Das Element mit Index 2 ist: !
Das letzte Element ist: !
Die gesamte Liste ist: Hallo -> Welt -> ! ->
Und nochmal die ganze Liste: Hallo -> Welt -> ! ->
Irgendein Studi in der Liste in der alten PO? false
Irgendein Studi in der Liste in der alten PO? true
Irgendein Studi in der Liste in der alten PO? true
```

Die Code-Dateien von N. finden Sie in den vorgegebenen Dateien zu diesem Übungsblatt.

Aufgabe 2: Matrixmultiplikation nach VL 14

Das Multiplizieren von Matrizen ist eine häufige Operation, die z. B. bei künstlichen neuronalen Netzen, der Simulation von Quantencomputern und bei 3D-Computergrafiken benötigt wird. Da das Multiplizieren per Hand recht fehleranfällig ist und lange dauert, wollen wir eine Klasse `Matrix` schreiben, die eine zweidimensionale Matrix mit ganzzahligen Einträgen repräsentiert.

Folgende, soweit möglich paket-private, Methoden bzw. Konstruktoren soll es in Ihrer Klasse geben:

[1] Bei den meisten Programmieraufgaben gibt es mehr als einen funktierenden Lösungsweg. Diskutieren Sie gerne untereinander Ihre Lösungsansätze und lerne Sie damit verschiedene Lösungsstrategien und verschiedene Anwendungsmöglichkeiten der Java-Funktionalitäten kennen. Ihre Abgabe müssen Sie aber final selbst formulieren/eintippen.

- Konstruktor `Matrix(int[][])`
  Der Konstruktor bekommt ein `int[][]`-Array übergeben, das die Werte der neu zu erstellenden Matrix enthält. `new Matrix(new int[][]{ {1, 2}, {3, 4}, {5, 6} })` würde eine Matrix mit 2 Spalten und 3 Zeilen anlegen.
  Falls dem Konstruktor eine leere Matrix, `null` oder kein rechteckiges Array übergeben wird (also wenn nicht jede Zeile gleich viele Einträge enthält oder eine Zeile `null` ist), soll eine `IllegalArgumentException` geworfen werden. Folgende Aufrufe sollen beispielsweise diese Exception auslösen:
  `new Matrix(new int[][]{ {1, 2}, {3, 4}, {5, 6, 7} })`
  `new Matrix(new int[][]{ {1, 2}, null, {5, 6} })`
  `new Matrix(new int[0][0])`
  Legen Sie eine Defensive Copy des übergebenen Arrays an.
- `int get(int row, int column)`
  Gibt den Wert an der angegebenen Zeile und Spalte der Matrix zurück. Das Element oben links ist in Zeile 0 und Spalte 0. Im ersten Konstruktor-Beispiel oben wären oben links der Wert 1 und in Zeile 1 und Spalte 0 der Wert 3 usw.
  Falls die Zeilen- oder Spaltennummer ungültig ist, soll eine `IndexOutOfBoundsException` (aber keine `ArrayIndexOutOfBoundsException`) geworfen werden.
- `Matrix multiply(Matrix)`
  Multipliziert die aktuelle Instanz (this) mit der übergebenen Matrix und gibt das Produkt als eine **neue** Matrix zurück (this wird nicht verändert).[2]
  Rechenbeispiel:

$$\underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{7} & \mathbf{8} \\ 2 & 6 & 5 & 3 \end{pmatrix}}_{\text{this}} \cdot \underbrace{\begin{pmatrix} \mathbf{2} & 8 & 9 \\ \mathbf{1} & 12 & 15 \\ \mathbf{42} & 3 & 6 \\ \mathbf{50} & 21 & 4 \end{pmatrix}}_{\text{übergebene Matrix}} = \underbrace{\begin{pmatrix} \mathbf{698} & 221 & 113 \\ 370 & 166 & 150 \end{pmatrix}}_{\text{neue Matrix}}$$

  Dabei ist $698 = 1 \cdot 2 + 2 \cdot 1 + 7 \cdot 42 + 8 \cdot 50$.
  Zwei Matrizen A und B können nur dann miteinander multipliziert werden, wenn die Anzahl der Spalten in A gleich der Anzahl der Zeilen in B ist. Falls diese Bedingung nicht erfüllt ist, soll `multiply` eine `IllegalArgumentException` werfen. Diese Exception soll außerdem auch dann geworfen werden, wenn der Methode `null` übergeben wird.
- Überschreiben Sie `toString()`, sodass die einzelnen Zeilen der Matrix untereinander, und die einzelnen Zahlen mit Leerzeichen getrennt zurückgegeben werden. Beispiel:

```
-22 1 3
4 5 16
```

  Ob ein Leerzeichen am Zeilenende oder ein Newline am Ende steht, ist egal. *Zusatzfrage: Warum kann diese Methode nicht paket-privat sein?*

Die Klasse `Matrix` darf nur private Instanzvariablen haben; evtl. vorhandene, zusätzliche Hilfsmethoden müssen privat sein.

Bonusfrage: Warum ist es sinnvoll, dass der Konstruktor eine Defensive Copy anlegt? Schreiben Sie ein Beispiel, wo es ohne Defensive Copy zu Problemen kommen würde. *Wenn*

[2] Sollten Sie nicht (mehr) wissen, wie Matrixmultiplikation funktioniert, informieren Sie sich z. B. mithilfe einer Suchmaschine.

Sie Feedback zu Ihrer Antwort haben wollen, können Sie wie gewohnt Feedback anfordern und Ihre Antwort als Kommentar oben in die `Matrix.java` *schreiben.*

Aufgabe 3: Interpolationssuche nach VL 16

Die Interpolationssuche ist ein von der binären Suche abgeleitetes Suchverfahren, bei dem versucht wird, das Array schlauer als einfach in der Mitte aufzuteilen.

Wie bei der binären Suche wird das vorsortierte Eingabearray an einer bestimmten Stelle geteilt. Anschließend wird festgestellt, ob der gesuchte Wert im vorderen oder hinteren Teil des Arrays liegt und der entsprechende Teil wird wieder aufgeteilt. Dieser Vorgang wird so lange wiederholt, bis der gesuchte Wert gleich dem Element ist, an dem das Array aufgeteilt wird.

Der Index x des Trennungselements wird dabei mit folgender Formel bestimmt[3]:

$$x = \begin{cases} l & \text{falls } A[r] = A[l] \\ l + \left\lfloor \frac{v - A[l]}{A[r] - A[l]} \cdot (r - l) \right\rfloor & \text{sonst} \end{cases} \tag{1}$$

Hierbei sind l und r die linke bzw. rechte Grenze des Suchbereichs, v ist der gesuchte Wert und A das Array, in dem gesucht wird.

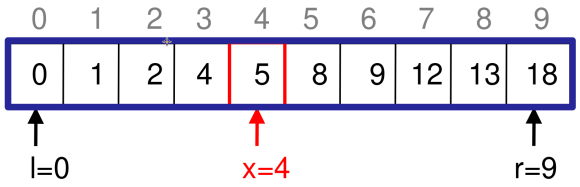
Bei der Interpolationssuche wird die Suche erfolglos abgebrochen, sobald $A[l] \leq v \leq A[r]$ *nicht* mehr gilt.

Im folgenden Beispiel wird das gesuchte Element v mit der Interpolationssuche schneller gefunden als mit der binären Suche:

1.

$$v = 8,\ l = 0,\ r = 9$$

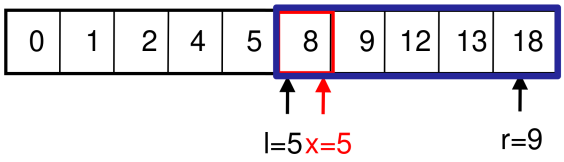
$$x = 0 + \frac{8 - 0}{18 - 0} \cdot (9 - 0) = 4 \tag{2}$$



2.

$$v = 8,\ l = 5,\ r = 9$$

$$x = 5 + \frac{8 - 8}{18 - 8} \cdot (9 - 5) = 5 \tag{3}$$



[3] $\lfloor x \rfloor$ steht für das Abrunden auf eine ganze Zahl.

Schreiben Sie eine Klasse `Search` mit folgenden **Klassen**-Methoden:

- `private int split(int[] haystack, int needle, int left, int right)`
  Diese Methode bestimmt mithilfe der oben gegebenen Formel, an welcher Stelle das (sortierte) Array geteilt werden soll, und gibt den entsprechenden Index zurück.
- `int search(int[] haystack, int needle)`
  Diese Methode sucht das Element `needle` in dem übergebenen, bereits sortierten Array. Hierbei soll die Methode `split()` verwendet werden, um den Suchraum schrittweise einzuschränken, bis das gesuchte Element gefunden wurde.
  Falls das Element nicht gefunden wurde, soll -1 zurückgegeben werden, ansonsten der Index, an dem es gefunden wurde.

Implementieren Sie außerdem eine `main`-Methode in derselben Klasse, die eine beliebige Anzahl ganzer Zahlen als Argumente entgegennimmt. Die erste Zahl ist dabei das zu suchende Element, die restlichen Zahlen sollen in einem Array gespeichert werden. Anschließend soll mit `search()` nach der ersten Zahl gesucht werden. Geben Sie das Resultat von `search()` als ganze Zahl auf der Standardausgabe aus. Sie dürfen davon ausgehen, dass mindestens ein Argument angegeben wird und die Zahlen für das Array sortiert sind; Sie müssen keine Fehlerfälle behandeln.

Beispiel:

```
% java Search 8 0 1 2 4 5 8 9 12 13 18
5
% java Search 8
-1
```