Heinrich Heine Universität Düsseldorf

Programmierung – WS 2025/26
Dr. Markus Brenneis
Abgabefrist für Feedback:
08.12.25, 10 Uhr

8. Übungsblatt

Aufgabe 1: Listen und Arrays nach VL 14

Lisa möchte ein Programm schreiben, das über das Internet nach und nach Produktnamen und Preise von einem Kassensystem einlesen. Wie viele Produkte sie einlesen muss, steht vorab nicht fest. Nach dem Einlesen aller Produkte möchte sie die Namen der Produkte und den Gesamtpreis ausgeben.

Hilmar möchte eine Anwendung schreiben, um 1.000.000 Datenpunkte von einem Helligkeits-Sensor einzulesen und Berechnungen auf diesen Daten durchzuführen. Dabei muss er oft auf verschiedene Indices zugreifen.

Welche Datenstruktur (Array oder verkettete Liste) würden Sie jeweils Lisa und Hilmar für ihre Anwendungen empfehlen? Begründen Sie Ihre Antwort. *Wenn Sie individuell Feedback erhalten wollen, geben Sie Ihre Lösung als Plaintext-Datei mit dem Namen* `antwort.txt` *ab; schreiben Sie* `// Feedback gewünscht` *in die Datei.*

Aufgabe 2: Verkettete Liste nach VL 14

In den vorgegebenen Dateien zu diesem Blatt finden Sie die Klasse `List` mit der inneren Klasse `Node`, wie sie Sie aus der Vorlesung kennen, sowie noch weitere Klassen und Interfaces, die für manche Teilaufgaben benötigt werden.

Erweitern Sie die `List`-Klasse um folgende (soweit nicht anders angegeben: paket-private) Methoden:

Hinweise:

- Wenn Sie die automatischen Tests nutzen: Die Tests verwenden `add()` und den Default-Konstruktor, um Listen zu erstellen und um dann z. B. zu prüfen, ob das Löschen von Elementen funktioniert. Implementieren Sie diese Methoden also unbedingt korrekt, bevor Sie Ihren Code hochladen.
- Testen Sie Ihre Implementierung in einer `main`-Methode.

(a) `void add(int)`: fügt den übergebenen Wert am **Ende** der Liste ein

(b) `int get(int)`: gibt den Wert am übergebenen Index zurück; falls der Index ungültig ist, soll eine `IndexOutOfBoundsException` geworfen werden.

(c) `int length()`: gibt die Anzahl der Listenelemente zurück

(d) `void addAll(int[])`: fügt jede Zahl des übergebenen Arrays zur Liste hinzu (Einfügen am Listenende); die Reihenfolge in der Liste soll der Reihenfolge im Array entsprechen

(e) `List(int[])`: zusätzlicher Konstruktor, der die Liste direkt mit den übergebenen Zahlen initialisiert; die Reihenfolge in der Liste soll der Reihenfolge im Array entsprechen
Tipp: Benutzen Sie `addAll`.

(f) `public String toString()`: gibt die Elemente der Liste als einen String zurück; dabei sollen alle Zahlen (beginnend bei `head`) durch Kommata getrennt zurückgegeben werden; verwenden Sie ausschließlich Kommata zwischen den Zahlen und fügen Sie keine weiteren Zeichen (wie z. B. Leerzeichen oder ein Komma nach der letzten Zahl) ein; bei einer leeren Liste ist die Rückgabe folglich ein leerer String
Beispiel: `1,9,42,3,7`

(g) `int find(int)`: durchsucht die Liste nach der übergebenen Zahl und gibt den Index des **ersten** Elements, das der gesuchten Zahl gleicht, zurück (Index-Zählung beginnt – wie bei Arrays – bei 0); wenn die Zahl nicht vorhanden ist, soll -1 zurückgegeben werden
Beispiel: [1,5,3,4,3,4] → find(3) → 2

(h) `int findLast(int)`: durchsucht die Liste nach der übergebenen Zahl und gibt den Index des **letzten** Elements, das der gesuchten Zahl gleicht, zurück; wenn die Zahl nicht vorhanden ist, soll -1 zurückgegeben werden
Beispiel: [1,5,3,4,3,4] → findLast(3) → 4

(i) `MaybeInt findFirst(Int2BooleanFunction p)`: gibt das erste Element zurück, das das übergebene Prädikat erfüllt
Beispiel 1: [-9,2,-8,4,0] → findFirst(new IsPositive()) → MaybeInt.of(2)
Beispiel 2: [-9,-8,0] → findFirst(new IsPositive()) → MaybeInt.empty()

(j) `void remove(int)`: entfernt das Element an dem gegebenen Index aus der Liste; sollte der übergebene Index nicht existieren, soll nichts geschehen (denken Sie an den Sonderfall einer leeren Liste)

(k) `void removeFirstOccurrence(int)` und `void removeLastOccurrence(int element)`: suchen in der Liste das erste bzw. letzte Vorkommen des übergebenen Wertes und entfernen es aus der Liste
Beispiel: [1,5,3,4,3,4] → removeLastOccurence(3) → [1,5,3,4,4]
Tipp: Verwenden Sie bereits geschriebene Methoden wieder, um Arbeit zu sparen.
Tipp 2: Schreiben Sie `Occurrence` *richtig.*

(l) *Bonusaufgabe:* `List map(Int2IntFunction f)`: lässt die Liste (`this`) unverändert und gibt eine neue Liste mit den gleichen Elementen wie die ursprüngliche Liste zurück, allerdings wurde auf jedes Element die Funktion `f` angewendet.
Beispiel: [-1,2,3] → map(new Square()) → [1,4,9]