

Aufgabe 7 Programmieren I


Hinweise

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 12. Dezember 2021 um 23:59 Uhr im ISIS-Kurs erfolgt sein. Es gelten die Ihnen bekannten Übungsbedingungen.
- Lösungen zu diesen Aufgaben sind als gezippter Projektordner abzugeben. Eine Anleitung zum Zippen von Projekten finden Sie auf der Seite des ISIS-Kurses. *Bitte benutzen Sie einen Dateinamen der Form VornameNachname.zip.*
- Bitte beachten Sie, dass Abgaben im Rahmen der Übungsleistung für die Zulassung zur Klausur relevant sind. Durch Plagieren verirken Sie sich die Möglichkeit zur Zulassung zur Klausur in diesem Semester.

Erstellen Sie für diese Hausaufgabe ein neues Projekt mit Namen **Aufgabe7**.

Aufgabe 7.1 „Selection Sort“ für Listen—genauer *SinglyLinkedList*-Instanzen (2 Punkte)

Implementieren Sie „Selection Sort“ für einfach verkettete Listen; also implementieren Sie eine Methode `selectionSort` in der Klasse `SinglyLinkedList` ohne Formalparameter und ohne Rückgabewert, welche die Liste nach dem „Selection Sort“-Algorithmus sortiert. Sie dürfen dazu annehmen, dass sich alle Objekte, die in der Liste gespeichert sind, nach `Integer` casten lassen. Testen Sie Ihre Implementierung mit der `main`-Methode aus Abbildung 1.

Hinweis Benutzen Sie die Implementierung von Listen aus der Vorlesung, verfügbar auf  GitHub. Sie müssen erklären können, wie Ihr Algorithmus mit dem „Selection Sort“-Algorithmus auf den Folien zusammenhängt.

```
public static void main(String[] args) {  
    SinglyLinkedList sll = new SinglyLinkedList();  
    int[] someInts = {4,7,5,8,32,-10,0,1,1};  
    for(int i : someInts){  
        sll.append(i);  
    }  
    System.out.println("Die Eingabeliste ist diese: " + sll + ".");  
    sll.sort();  
    System.out.println("Sortiert sieht das so aus: " + sll + ".");  
}
```

Die Eingabeliste ist diese: [4,7,5,8,32,-10,0,1,1].
Sortiert sieht das so aus: [-10,0,1,1,4,5,7,8,32].

Process finished with exit code 0

Abbildung 1: Selection Sort

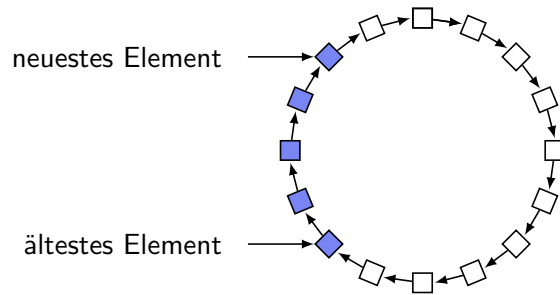


Abbildung 2: Pufferring mit ListElements

Aufgabe 7.2 Pufferring mit Listen (3 Punkte)

Die Idee vom Pufferring mit Listen ist in Abbildung 2 illustriert. Implementieren Sie einen Pufferring unter Verwendung von `ListEntry`-Instanzen, die einen Ring formen, und zwar in einer Klasse `CircularListBuffer` (siehe auch **W**: *Circular buffer*), und zwar wie folgt:

- Implementieren Sie einen Konstruktor, der eine Ganzzahl als Formalparameter hat, welche die (unveränderliche) Größe angibt.
- Implementieren Sie Methode `put`, die ein beliebiges Objekt als Formalparameter hat und einen boolean als Rückgabewert. Wenn noch Platz im Puffer ist, dann soll das Objekt gespeichert werden und `true` zurück gegeben werden; ansonsten wird lediglich `false` zurückgegeben.
- Implementieren Sie die Methode `remove` ohne Formalparameter und einem beliebigen Objekt als Rückgabewert. Das Objekt der Rückgabe ist entweder das älteste Objekt im Puffer oder `null`, wenn der Puffer leer ist.
- Implementieren Sie die `toString`-Methode geeignet (siehe Abbildung 3).

```
public static void main(String[] args) {
    CircularListBuffer cb = new CircularListBuffer(15);
    int[] someInts = {4, 7, 5, 8, 32, -10, 0, 1, 1};
    for (int i : someInts) {
        cb.put(i);
    }
    System.out.println("Der Pufferinhalt ist: " + cb + ".");
}
```

Der Pufferinhalt ist: [4, 7, 5, 8, 32, -10, 0, 1, 1].

Abbildung 3: Test für CircularListBuffer

Aufgabe 7.3 Zusammenfügen eines Arrays von sortierten Listen (2 Punkte)

Erstellen Sie eine Klasse `ListCoalescing` mit einer Klassenmethode `merge`, die ein Array von `SinglyLinkedLists` (die aufsteigend sortiert sind) als Formalparameter hat und als Rückgabe eine `SinglyLinkedList` liefert; die Rückgabe soll eine sortierte Liste sein, die alle Elemente aller Listen im Array beinhaltet (inklusive mehrfacher Kopien). Sie dürfen dazu annehmen, dass sich alle Objekte, die in den Listen gespeichert sind, nach `Integer` casten lassen.


Benutzen Sie *nicht* die Sortiermethode aus der vorherigen Aufgabe, eine Bibliothek, oder ähnliches!

Testen Sie Ihr Programm, z.B. so wie in Abbildung 4.

```
public static void main(String[] args) {
    int[][] test = {
        {5,7,8,30},
        {10,20},
        {19,22,50},
        {28,35,40,45}
    };
    SinglyLinkedList[] x = new SinglyLinkedList[test.length];
    for (int i = 0; i < test.length; i++){
        x[i] = new SinglyLinkedList();
        for (int k : test[i]){
            x[i].append(k);
        }
    }
    System.out.println("merged" + merge(x));
}
```

merged[5,7,8,10,19,20,22,28,30,35,40,45,50]

Abbildung 4: Test für `merge`

Hinweis Benutzen Sie die Implementierung von Listen aus der Vorlesung, verfügbar auf  GitHub.

Aufgabe 7.4 Total flach (2 Zusatzpunkte)

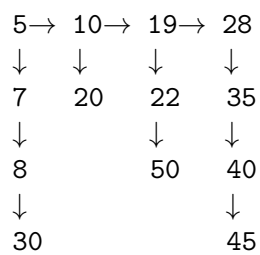
Implementieren Sie die rekursive Variation des „List flattening“ (siehe Abbildung 5) als statische Methode von `SinglyLinkedList`. Diese Methode `fullyFlatten` hat keine Formalparameter und keine Ausgabe. Insbesondere soll Ihre Implementierung auch Listen von Listen von Listen als Eingabe in eine einzige „flache“ Liste umformen. Ignorieren Sie `null`-Einträge.

Hinweis Sie müssen die Elemente nicht sortieren.

¹Abbildung 5 stammt von <https://www.geeksforgeeks.org/flattening-a-linked-list/>.

Given a linked list where every node represents a linked list and contains two pointers [...]:

- (i) Pointer to next node in the main list [...]
- (ii) Pointer to a linked list where this node is headed [...]



Write a function `flatten()` to flatten the lists into a single linked list. For example, for the above input list[, a list of four “vertical” lists of different lengths, the] output list should be

5→7→8→30→10→20→19→22→50→28→35→40→45.

Abbildung 5: List flattening¹