Übungsblatt 6 (Abgabe bis Dienstag, den 26.11.2024, um 12:00 Uhr)

Prof. Dr. J. Giesl

D. Cloerkes, J.-C. Kassing, N. Lommen, E. Meyer

Allgemeine Hinweise:

- Die **Deadline** zur **Abgabe** der Hausaufgaben ist am **Dienstag, den 26.11.2024, um 12:00 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine*n Abgabepartner*in zu finden. Es darf **nur ein*e** Abgabepartner*in die Abgabe hochladen. Diese*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als `.zip`-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass unsere Hiwis ihn mit Feedback versehen können.

  Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von `.java`-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen, etwa die Datei `SimpleIO.java`, dürfen nicht mit abgegeben werden.

  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, wenn die entsprechenden Klassen aus dem Lernraum hinzugefügt werden. Ansonsten werden keine Punkte vergeben.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden. Klicken Sie dazu im Lernraum rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Aufgabe 3 (Rekursive Datenstrukturen): (3+3+5+5+6+8+8 = 38 Punkte)

In dieser Aufgabe sollen eine Datenstruktur für adaptive einfach verkettete Listen sowie einige Algorithmen darauf implementiert werden. Diese ist eine besondere dynamische Datenstruktur, die wir deshalb als adaptiv bezeichnen, weil sie die Reihenfolge ihrer Elemente an die Häufigkeit der Suche nach ihnen anpassen kann: Jedes Mal, wenn ein Wert gesucht wird, kann er wahlweise um eine Position nach vorne oder ganz nach vorne rutschen. Dadurch soll erreicht werden, dass häufig gesuchte Elemente weiter vorne stehen und Suchen somit tendenziell schneller sind. Wir verwenden dafür die Klasse `AdaptiveList`:

```
public class AdaptiveList {

    private int value;
    private AdaptiveList next;

    public AdaptiveList(int value, AdaptiveList next) {
        this.value = value;
        this.next = next;
    }

    public int getValue() {
        return this.value;
    }

    public AdaptiveList getNext() {
        return this.next;
    }

    public setValue(int value) {
        this.value = value;
    }

    public setNext(AdaptiveList next) {
        this.next = next;
    }
}
```

Jedes `AdaptiveList`-Objekt repräsentiert ein Element der einfach verketteten Liste. Im seinem `int`-Attribut `value` steht der Wert, der an dieser Stelle der Liste gespeichert ist. In seinem Attribut `next` steht das Nachfolgeelement, das wiederum vom Typ `AdaptiveList` ist. Dadurch wird der Rest der Liste dargestellt. Im Fall `next == null` handelt es sich um das letzte Element der Liste. Gehen Sie davon aus, dass eine Liste immer aus mindestens einem Element besteht; die leere Liste betrachten wir nicht.
Außerdem sind ein Konstruktor sowie die beiden Getter und Setter gegeben. Da die konkrete Implementierung der Listenstruktur möglichst einfach änderbar sein soll, ist von diesen Selektoren (und ähnlichen Methoden) konsequent Gebrauch zu machen.

In der gesamten Aufgabe dürfen Sie nur eine einzige Schleife verwenden (die Verwendung von Rekursion ist hingegen überall erlaubt). Überlegen Sie also gut, wo eine Schleife am nützlichsten ist.

Ihre Implementierung soll genau die folgenden Methoden beinhalten. Berücksichtigen Sie dabei die Konzepte der Datenkapselung. In dieser Aufgabe dürfen Sie keine Hilfsmethoden schreiben und keine Bibliotheksfunktionen verwenden. Sie dürfen aber die Methoden vorangehender Teilaufgaben nutzen, auch wenn Sie diese nicht selbst implementiert haben.
Zugriffsmodifikatoren (inkl. `static`) und Rückgabetypen der zu implementierenden Methoden müssen Sie selbst begründet setzen.

a) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `singletonList(int value)`, die eine einelementige Liste zurückgibt, in der ausschließlich der Wert `value` gespeichert ist.

b) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `isLast()`. Für ein `AdaptiveList`-Objekt `xs` soll `xs.isLast()` zurückgeben, ob es sich dabei um das letzte Element der Liste handelt (d.h. um eine einelementige Liste).

c) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `prepend(int value)`, die die aktuelle Liste vorne um ein Element mit dem `int`-Wert `value` erweitern soll. Dieses neue erste Element soll dann zurückgegeben werden. Außerdem soll die Liste, auf der `prepend` aufgerufen wurde, nach dem Aufruf ebenfalls dieses neue Element als erstes Element haben, und danach alle anderen in derselben Reihenfolge wie vor dem Aufruf.

Beispiel:

Falls `xs` die Liste mit der Zahl `[17]` ist, so soll der Aufruf `xs.prepend(4)` die Liste also in `[4,17]` ändern. Diese Liste soll auch als Ergebnis zurückgegeben werden.

d) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `append(int value)`, die eine Liste hinten um ein Element mit dem `int`-Wert `value` erweitern soll. Das erste Element der neuen, erweiterten Liste soll dann zurückgegeben werden. Auch die Liste, auf der `append` aufgerufen wurde, soll nach den ursprünglichen Elementen, deren Reihenfolge nicht verändert werden soll, dieses neue Element als letztes Element haben.

Beispiel:

Falls `xs` die Liste mit den Zahlen `[4,17]` ist, so soll der Aufruf `xs.append(25)` die Liste also in `[4,17,25]` ändern. Diese Liste soll auch als Ergebnis zurückgegeben werden.

e) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `contains(int value)`, die beim Aufruf auf einem `AdaptiveList`-Objekt zurückgeben soll, ob die dadurch repräsentierte Liste den Wert `value` enthält. Die Liste soll nicht verändert werden.

f) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `containsAdaptive(int value)`, die beim Aufruf auf einem `AdaptiveList`-Objekt zurückgeben soll, ob die dadurch repräsentierte Liste den Wert `value` enthält. Falls das nicht der Fall ist, soll die Liste nicht verändert werden. Ansonsten soll der erste `int`-Wert `value` im Vergleich zu seiner Position vor dem Aufruf um eine Stelle nach vorne rutschen: Der Wert, der vor dem Aufruf direkt vor ihm gelegen hat, soll also nach dem Aufruf direkt hinter ihm liegen. Die Position der restlichen Werte soll nicht verändert werden.

Beispiel:

Falls `xs` die Liste mit den Zahlen `[4,17,25]` ist, so soll der Aufruf `xs.containsAdaptive(25)` die Liste also in `[4,25,17]` ändern und das Ergebnis `true` zurückgeben.

g) Ergänzen Sie die Klasse `AdaptiveList` um eine Methode `containsTopPriority(int value)`, die beim Aufruf auf einem `AdaptiveList`-Objekt zurückgeben soll, ob die dadurch repräsentierte Liste den Wert `value` enthält. Falls das nicht der Fall ist, soll die Liste nicht verändert werden. Ansonsten soll der erste `int`-Wert `value` nach dem Aufruf an den Anfang der Liste verschoben werden. An seiner ursprünglichen Position soll nach dem Aufruf sein ehemaliger Nachfolger stehen, d.h. die sonstige Reihenfolge der Werte soll sich nicht ändern.

Beispiel:

Falls `xs` wieder die Liste mit den Zahlen `[4,17,25]` ist, so soll der Aufruf `xs.containsTopPriority(25)` die Liste also in `[25,4,17]` ändern und das Ergebnis `true` zurückgeben.

Aufgabe 5 (Entwurf einer Klassenhierarchie): (12 Punkte)

Im Folgenden beschäftigen wir uns mit dem Entstehungsprozess von Büchern:

- Personen haben einen Namen und ein Alter.
- Ein Autor ist eine Person. Von besonderem Interesse ist, wie viele Bücher der Autor bereits veröffentlicht hat. Der Autor kann ein neues Buch schreiben.
- Ein Buch hat einen Titel und eine beliebig große Anzahl an Seiten, welche jeweils ein einfacher Text sind.
- Ein Lektor ist eine Person. Er kann zu einem Buch eine Rezension erstellen.
- Eine Rezension bezieht sich auf ein Buch. Außerdem umfasst eine Rezension eine beliebig große Anzahl an Anmerkungen.
- Eine Anmerkung bezieht sich anhand einer Seitenzahl auf eine konkrete Buchseite und anhand einer Zeilennummer auf eine konkrete Zeile auf dieser Seite. Sie enthält einen Kommentar, welcher ein einfacher Text ist.
- Wenn dem Autor eine Rezension eingereicht wird, erstellt er eine korrigierte Version des Buchs.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für den oben dargestellten Sachverhalt. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden, falls dies sinnvoll ist. Ergänzen Sie außerdem geeignete Methoden, um das Schreiben, Rezensieren und Korrigieren eines Buchs zu modellieren. Verwenden Sie hierbei die Notation aus der entsprechenden Tutoriumsaufgabe.

Hinweise:

- Um diese Aufgabe und insbesondere das erstellte Klassendiagramm verständlich zu halten, wurde in dieser Aufgabe auf das Gendern verzichtet.

Aufgabe 6 (Deck 6): (Codescape)

Lösen Sie die Missionen von Deck 6 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn Sie Ihre Lösung vor der einheitlichen Codescape Deadline am Freitag, den 24.01.2025, um 23:59 Uhr abschicken.