Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, R. Palenta, A. Reuss, S. Schulze Frielinghaus WS 2016/17

Übungsblatt 7 Abgabefrist: 12.12.2016 (vor 05:00 Uhr)

Σ Punkte: 20 | Σ Bonuspunkte: 2

Aufgabe 7.1 (P) Doppelt verkettete Liste

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)!

In dieser Aufgabe soll eine doppelt verkettete Liste implementiert werden. In einer doppelt verketteten Liste hat jedes Listenelement neben einer Referenz auf das nächste Element auch eine Referenz auf das vorherige Element. Gibt es kein nächstes oder vorheriges Element, ist die Referenz `null`.

Verwenden Sie das vorgegebene Programmgerüst. Der vorgegebene Code darf nicht verändert werden. Sie dürfen jedoch weitere Membervariablen oder Methoden in der Klasse `DoublyLinkedList` hinzufügen (nicht notwendig!). In der Klasse `Entry` darf kein weiterer Code hinzugefügt werden. Die Klasse `class Entry` ist eine innere Klasse der Klasse `public class DoublyLinkedList`. D. h. aus der Klasse `DoublyLinkedList` kann auf die Membervariablen der Klasse `Entry` zugegriffen werden. Die Klasse `DoublyLinkedList` repräsentiert die doppelt verkettete Liste. Die Klasse `Entry` repräsentiert ein Listenelement in dieser Liste. Implementieren Sie die folgenden Methoden und den Konstruktor:

1. Der Konstruktor `public DoublyLinkedList()` erzeugt eine leere `DoublyLinkedList`. Eine `DoublyLinkedList` ist leer, wenn die Referenz `head` `null` ist.

2. Die Methode `public int size()` liefert als Rückgabewert die Anzahl der Elemente der `DoublyLinkedList`.

3. Die Methode `public void add(int info)` fügt am Ende der `DoublyLinkedList` ein neues `Entry`-Element hinzu. Die Membervariable `elem` dieses `Entry`-Elements hat den Wert des Methodenparameters `info`. Implementieren Sie diese Methode *rekursiv*. Dazu benötigen Sie eine zusätzliche Hilfsmethode. Überlegen Sie sich, welche Parameter diese Hilfsmethode benötigt, und implementieren Sie sie entsprechend. Deklarieren Sie Ihre Hilfsmethode als `private`.

4. Die Methode `public void add(int index, int info)` fügt an der Position `index` der `DoublyLinkedList` ein `Entry`-Element ein. Die Membervariable `elem` dieses `Entry`-Elements hat den Wert des Methodenparameters `info`. Der Index beginnt bei 0, d. h. ist `index == 0`, wird der `head` der `DoublyLinkedList` verändert, ist `index == 1`, wird das neue Element als zweites Element eingefügt.

5. Die Methode `public int remove(int index)` entfernt das `Entry`-Element an Position `index` und liefert als Rückgabewert die Membervariable `elem` des entfernten `Entry`-Elements. Der Index beginnt bei 0, d. h. bei `index == 0` wird das erste `Entry`-Element der `DoublyLinkedList` entfernt usw. Ist der Index ungültig (z. B. negativ), wird kein Element entfernt und der Rückgabewert ist `Integer.MIN_VALUE`.

6. Die Methode `public void shiftLeft(int index)` verschiebt die Elemente der `DoublyLinkedList` um `index` Positionen nach links. D. h. das Element an Position

`index` wird das erste Element. Die Elemente, die nach links „aus“ der Liste geschoben würden, werden am Ende der Liste entsprechend der Reihenfolge angehangen. Die Verschiebung einer Liste mit n Elementen erfolgt nur, wenn `index` im Bereich 0 bis $n-1$ liegt. Beispiele: $[0,1,2,4]$ wird bei `shiftLeft(2)` zu $[2,4,0,1]$, $[0,1]$ wird bei `shiftLeft(1)` zu $[1,0]$, bei `shiftLeft(0)` bleibt die Liste unverändert. Es dürfen nur die `next`-, `prev`- und `head`-Referenzen der `Entry`-Elemente sowie die Objektvariable `head` der `DoublyLinkedList` verändert werden. D. h. insbesondere, dass teilweise oder vollständige Kopien der `DoublyLinkedList` nicht verwendet werden dürfen.

Aufgabe 7.2 (P) Endrekursion

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)!

Die Fakultätsfunktion ist für positive Zahlen wie folgt definiert:

$$n! = \prod_{i=1}^{n} i \qquad (n \geq 1)$$
$$0! = 1$$

Schreiben Sie ein Programm `Fak.java`, das die Fakultätsfunktion auf folgende Weisen berechnet.

1. In der Methode `public static int facRec(int n)` soll die Fakultätsfunktion *rekursiv* (aber nicht end-rekursiv) berechnet werden. Der Parameter `n` gibt dabei den Integer an, für den die Fakultät berechnet werden soll.
2. In der Methode `public static int facTailRec(int n)` soll die Fakultätsfunktion *endrekursiv* berechnet werden. Dafür benötigen Sie eine zusätzliche Hilfsmethode `private static int facTailRecHelper(int n, int k)`. Der Parameter `n` gibt dabei den Integer an, für den die Fakultät berechnet werden soll. Der Paramter `k` gibt das bisher berechnete Ergebnis an.
3. In der Methode `facIt(int n, int k)` soll die Fakultätsfunktion *iterativ* durch Umwandlung der Endrekursion aus der Methode `facTailRec(int n, int k)` berechnet werden. D h. die Berechnung erfolgt in einer `while(true){...}` Schleife.

Aufgabe 7.3 (P) Verbotenes Glücksspiel

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)

Wir wollen ein kleines Ratespiel implementieren. Die Regeln sind einfach: Zu Beginn wird vom Programm eine zufällige (ganze) Zahl `z` zwischen 10 und 100 bestimmt. Anschließend wird reihum gespielt, wobei jeder Spieler eine Zahl nennt. Die erste Zahl kann beliebig ≥ 10 gewählt werden, alle folgenden müssen jeweils höher als die zuletzt genannte Zahl sein. Es verliert derjenige Spieler, welcher zuerst eine Zahl nennt, die größer als `z` ist.

a) Implementieren Sie dieses Spiel unter Verwendung der Vorlage `Raten.java`. Der vorhandene Code darf *nicht verändert*, sondern soll ergänzt werden. Die entsprechenden Stellen sind mittels `//TODO` gekennzeichnet. Der Konstruktor bekommt die Anzahl Spieler übergeben. Die Vorlage enthält die Methode `readIntRange(String msg, int lower, int upper)`. Diese können Sie zum Einlesen einer Zahl aus einem bestimmten Zahlenbereich *von der Konsole* verwenden, sofern gewünscht. Zum Erzeugen einer Zufallszahl können Sie die Methode `public static int generateNumber(int lower, int upper)` verwenden.
Überprüfen Sie die eingegebenen Zahlen auf Gültigkeit und lassen Sie den Spieler ggf. die Eingabe wiederholen.

b) Der Konstruktor kann mit einem beliebigen `int`-Wert aufgerufen werden, z. B. mit -1 oder auch mit einer Milliarde. Diskutieren Sie, wie die Klasse `Raten` intern damit umgehen soll. Implementieren Sie die von Ihnen favorisierte Lösung.

Aufgabe 7.4 (H) Kampf der Damen [13 Punkte]

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)!

Wir betrachten eine Variante des bekannten Damenproblems. Es treten zwei Spielerinnen, Weiß und Schwarz genannt, gegeneinander an. Gespielt wird nach den folgenden Regeln:

1. Zuerst wählt Weiß eine Zahl a aus der Menge $\{5, 6, 7, 8\}$.
2. Schwarz antwortet mit der Wahl einer ganzen Zahl b aus dem Bereich zwischen $a-1$ und $a+1$ (jeweils inklusive).
3. Im Folgenden wird auf einem rechteckigen Spielbrett mit den Seitenlängen a (Breite) und b (Länge) gespielt. D. h. es gibt $a \cdot b$ Felder. Jedes Feld wird mit einer zweistelligen Zahl bezeichnet, wobei die Zehnerstelle aus dem Bereich 1 bis a, die Einerstelle aus dem Bereich 1 bis b ist. Beispiel: Auf einem Spielbrett der Breite $a = 5$ und der Länge $b = 6$ bezeichnet 56 das Eckfeld rechts hinten.
4. Danach entscheidet Weiß, wer den weiteren Spielverlauf beginnt.
5. Abwechselnd werden nun Runden gespielt nach den folgenden Regeln, bis eine Spielerin nicht mehr ziehen kann oder aufgibt. Die Spielerin, welche den letzten Zug gemacht hat, gewinnt das Spiel.
6. Auf einem freien Feld wird eine *Dame* (Schachfigur) platziert. Die Dame muss dabei so stehen, dass sie keinen direkten Kontakt zu bereits auf dem Spielbrett platzierten Damen hat, d. h. weder in einer der beiden Diagonalen, in denen sie steht, noch in der Linie, in der sie steht, noch in der Reihe, in der sie steht, darf eine andere Dame stehen. In anderen Worten: Wenn die Dame diagonal oder geradeaus zieht, erreicht sie immer den Rand des Spielbretts, ohne dass ihr eine andere Dame im Weg steht.

Implementieren Sie diese Variante des Damenproblems unter Verwendung der Vorlage `DameSpiel.java`. Der vorhandene Code enthält bereits eine Ausgabe des Spielbretts. Er darf **nicht verändert**, sondern soll ergänzt werden. Die entsprechenden Stellen sind mittels `//TODO` gekennzeichnet.

Die Attribute `nrRows` und `nrColumns` speichern die Länge und Breite des Spielbretts. Das Attribut `board` repräsentiert das Spielbrett. Dabei entspricht die erste Dimension der Breite, die zweite der Länge. Das Attribut `whiteToMove` zeigt an, ob Weiß am Zug ist. Die Attribute `white` und `black` schließlich speichern die im Konstruktor übergebenen Spielernamen.

Denken Sie daran, die Benutzerin nach jedem Spielzug über den Fortgang des Spielgeschehens zu informieren. Die Eingabe eines Zugs soll als Eingabe einer Ganzzahl im beschriebenen Format erfolgen. Zum Abbrechen oder Aufgeben soll die -1 eingelesen werden. Denken Sie auch daran, Eingaben auf Gültigkeit zu überprüfen! Insbesondere soll die Eingabe **wiederholt** werden, wenn ein Zug eingegeben wurde, der nicht den Spielregeln entspricht.

Lösungsvorschlag 7.4

siehe DameSpiel.java

Korrekturbemerkungen:
Die Zuordnung von nrRows, nrColumns zu Länge/Breite war nicht klar definiert. Jede konsistente Verwendung der Attribute wird akzeptiert.

- Konstruktor: 0.5 Punkte
- `determineBoardSize`: 1 Punkt
- `initBoard`: 0.5 Punkte
- `determineFirstPlayer`: 0.5 Punkte
- `reportWinner`: 1 Punkt
- `mainLoop`: 9.5 Punkte, davon
  - Option *Aufgeben / Spiel beenden* statt Zugeingabe: 1 Punkt
  - Überprüfung auf gültigen Zug: 5 Punkte, davon
    - gültige Koordinaten: 1 Punkt
    - freies Feld: 0.5 Punkte
    - Alle Richtungen überprüfen: 3 Punkte
    - Kombination obiger Elemente: 0.5 Punkte
  - Hauptschleife mit Ausgabe des Spielgeschehens: 1.5 Punkte
  - Zug einlesen: 0.5 Punkte
  - Zug ausführen: 1 Punkt
  - Prüfen, ob noch Züge möglich sind: 0.5 Punkte

Aufgabe 7.5 (H) HeadList [7 Punkte]

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)!

In dieser Aufgabe soll eine einfach verkettete Liste implementiert werden, in der jedes Listenelement zusätzlich eine Referenz auf den „Kopf“ (das erste Element der Liste) hat. D. h. auch das erste Element hat eine Referenz auf sich selbst. Wir bezeichnen eine solche Liste als `HeadList`.

Verwenden Sie das vorgegebene Programmgerüst. Der vorgegebene Code darf nicht verändert werden. Sie dürfen weitere Membervariablen oder Methoden in der Klasse `HeadList` hinzufügen (nicht notwendig!). In der Klasse `Entry` darf kein weiterer Code hinzugefügt werden. Die Klasse `class Entry` ist eine innere Klasse der Klasse `public class HeadList`. D. h. aus der Klasse `HeadList` kann auf die Membervariablen der Klasse `Entry` zugegriffen werden. Die Klasse `HeadList` repräsentiert die beschriebene Liste `HeadList`. Die Klasse `Entry` repräsentiert ein Listenelement in dieser Liste. Implementieren Sie die folgenden Methoden:

1. Die Methode `public void add(int info)` fügt am Ende der `HeadList` ein neues `Entry`-Element hinzu. Die Membervariable `elem` dieses `Entry`-Elements hat den Wert des Methodenparameters `info`.

2. Die Methode `private void setHead(Entry newHead)` setzt die `first`-Referenzen aller in der `HeadList` enthaltenen `Entry`-Elemente auf `newHead`.

3. Die Methode `public int remove(int index)` entfernt das `Entry`-Element an Position `index` und liefert als Rückgabewert die Membervariable `elem` des entfernten `Entry`-Elements. Der Index beginnt bei 0, d. h. bei `index == 0` wird das erste `Entry`-Element der `HeadList` entfernt usw. Ist der Index ungültig (z. B. negativ), wird kein Element entfernt und der Rückgabewert ist `Integer.MIN_VALUE`.

4. Die Methode `public void reverse()` dreht die `HeadList` um. D. h. das letzte `Entry`-Element wird zum ersten, das vorletzte `Entry`-Element wird zum zweiten usw. Beispiel: $[1, 2, 3, 4, 5]$ wird zu $[5, 4, 3, 2, 1]$. Es dürfen dabei nur die `next`- und `head`-Referenzen verändert werden. D. h. insbesondere, dass teilweise oder vollständige Kopien der `HeadList` nicht verwendet werden dürfen.

Lösungsvorschlag 7.5

Korrekturbemerkungen:

- `add(int info)`: 1.5 Punkte
- `setHead(Entry newHead)`: 1 Punkt
- `remove(int index)`: 1.5 Punkte allg. Funktionalität, 0.5 Punkte index out of bounds, 0.5 Punkte Sonderfälle (erstes, letztes Element wird entfernt)
- `reverse()`: 1.5 Punkte allg. Funktionalität, 0.5 Punkte Sonderfälle (leere und einelementige Liste)

- Sollte lediglich eine doppelt verkettete Liste implementiert worden sein, kann es maximal 3 Punkte geben. Bei 3 Punkten muss auch `reverse()` entsprechend einer doppelt verketteten Liste korrekt implementiert sein.
- In der Methode `setHead(Entry newHead)` darf der `head` der Klasse `HeadList` auf `newHead` gesetzt werden - muss aber nicht! D.h. wenn die Methode wie an den Stellen im Lösungsvorschlag verwendet werden kann, gibt es volle Punktzahl. Werden Teile der Liste abgeschnitten o.ä. und es liegt nicht an einem falschen Aufruf der Methode (Parameter ist nicht der aktuelle/zu setzende `head`), gibt es Punktabzug.

Die Punktzahl der Testklasse ist die Mindestpunktzahl, *sofern nicht gegen Anforderungen der Aufgabe verstoßen wurde.* In der Testklasse können maximal 6 Punkte erreicht werden, da `setHead(Entry newHead)` nicht explizit getestet werden kann. Werden 6 Punkte erreicht, ist diese Methode aber sehr wahrscheinlich korrekt implementiert, falls sie verwendet wurde.

Aufgabe 7.6 (H) Endrekursion [2 Bonuspunkte]

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang)!

In dieser Aufgabe sei % der Modulo-Operator wie in *Java* und / der Operator für ganzzahlige Division ebenfalls wie in *Java*. D. h. $12345/10 = 1234$ und $12345\%10 = 5$.

Gegeben sind die folgenden Funktionen $f : \mathbb{N} \to \mathbb{N}$ und $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

$$f(x) = g(x, 0)$$

$$g(x, y) = \begin{cases} x^y & x < 10 \\ (x\%10)^y + g(x/10, y + 1) & x \geq 10 \end{cases}$$

Eine rekursive, aber nicht endrekursive Implementierung der Funktion f finden Sie im mitgelieferten Codegerüst. Implementieren Sie die Funktion f *endrekursiv* in der Methode `public static int ftailrec(int x)`. Sie benötigen dazu eine Hilfsfunktion. Überlegen Sie, welche und wie viele Parameter diese Hilfsfunktion benötigt, und implementieren Sie die Hilfsfunktion. Verwenden Sie *keine* Schleifen und keine Membervariablen in Ihrer Implementierung.

Erlaubte Java-Methoden

MiniJava:

```
public static int readInt()
public static int readInt(String s)
public static int read()
public static int read(String s)
public static String readString()
public static String readString(String s)
public static void write(String output)
public static void write(int output)
public static int drawCard()
public static int dice()
```

returns an integer from the interval $[2, 11]$

returns an integer from the interval $[1, 6]$

String:

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from `0` to `length() - 1`. The first char value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Example: `String s = "Hello Students"; char c = s.charAt(7);` saves the character `'t'` in the variable `c`.

```
public boolean isEmpty()
```

Returns `true` if, and only if, `length()` is `0`.

```
public int length()
```

Returns the length of this string. The length is equal to the number of Unicode code units in the string.

Example: `String s = "Hello Students"; int l = s.length();` saves the value `14` in the variable `l`.

```
public boolean equals(Object obj)
```

Compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.
Example: `String s = "Hello Students"; boolean v = s.equals("Hello Students");` here `v` is evaluated to `true`.

System:

`System.out.print(x)` prints the object `x` to the console

`System.out.println(x)` prints the object `x` to the console and terminates the line

Selbstgeschriebene Methoden, die selber nur erlaubte Methoden verwenden, sind erlaubt, sofern dies nicht explizit in der Aufgabenstellung verboten wurde. Methoden, die im gegebenen Programmgerüst definiert wurden, dürfen verwendet werden.