

13. Übung zur Vorlesung Einführung in die Programmierung

Aktuelle Hinweise:

- Dies ist das letzte reguläre Übungsblatt dieser Veranstaltung. In der nächsten Übungsstunde werden wir eine Probeklausur besprechen, welche etwas früher als üblich herausgegeben wird, damit Sie die Probeklausur vor der Übung zur Probe bearbeiten können.
- Die letzten beiden Vorlesungen am 6.2. und 8.2. sind derzeit als Fragestunde und zur Wiederholung eingeplant. Bitte schicken Sie uns zur Vorbereitung Ihre Fragen und Themen-Vorschläge zur Wiederholung/Vertiefung per eMail an: jost@tcs.ifl.lmu.de

A13-1 Doppelt verkettete Listen I In der Vorlesung am 25.01.18 wurde eine Implementierung von doppelt verketteten Listen vorgestellt, welche Sie von der Vorlesungshomepage herunterladen können.

a)

```
MyList2<String> ls = new MyList2<>();  
ls.addLast("O"); ls.addLast("V");  
ls.addFirst("T"); ls.addLast("I");  
iter = ls.iterator();  
iter.add("L"); iter.next();  
iter.remove(); iter.next();  
iter.set("U"); iter.next();  
iter.add("E");  
ls.addLast("P");
```

Schnell mit Papier & Bleistift ausrechnen: Welchen Inhalt hat die liste `ls`?




b) Erweitern Sie Klasse `MyList2` um eine Methode `public void reverse()`, welche die Reihenfolge aller Elemente in der Liste umdreht. Verwenden Sie dazu keinen Iterator!

c) Dieser Test offenbart, dass Dr. Jost einen Fehler bei der Implementierung gemacht hat:

```
MyList2<Integer> l1 = new MyList2<>();  
l1.addLast(3);  
l1.addFirst(2);  
l1.addLast(4);  
l1.addFirst(1);  
ListIterator<Integer> it = l1.iterator();  
while(it.hasPrevious()){ System.out.print(it.previous() + " "); }  
while(it.hasNext()      ){ System.out.print(it.next()      + " "); }  
while(it.hasPrevious()){ System.out.print(it.previous() + " "); }  
while(it.hasNext()      ){ System.out.print(it.next()      + " "); }
```

Helfen Sie Dr. Jost! Finden, beschreiben und beheben Sie den Fehler fix!

A13-2 Effiziente Datenstrukturen Betrachten Sie folgende Anwendungsszenarien. Würden Sie jeweils dazu eher ein Array oder eine verkettete Liste verwenden? Begründen Sie Ihre Antwort jeweils kurz mit 2–5 Sätzen!

- a) Angenommen Sie müssten eine Applikation zur Verwaltung von Besprechungsterminen schreiben. Mit welcher Datenstruktur verwalten Sie die Objekte, welche die Besprechungen repräsentieren? 
- b) Angenommen Sie müssten eine Applikation für eine Telefonzentrale eines großen Konzerns schreiben, mehrmals pro Sekunde zu jeder Telefonnummer den Namen des Teilnehmers ermitteln muss. An die Hardware können maximal 70000 Telefonapparate angeschlossen werden. 
- c) Sie möchten eine Programmiersprache mit expliziter Speicherverwaltung schreiben. Der Programmierer kann Speicherblöcke in beliebiger Größe anfordern, nach Belieben deren Bits verändern (für die Aufgabe irrelevant) und anschließend zur Wiederverwendung freigeben. 

A13-3 Knobelaufgabe* EiP-Teilnehmerin Birke Ballantz hat die Aufgabe A12-2 „Einfügen in Verkettete Listen“ einfach gut gefallen. Als sie dann in der Vorlesung doppelt verkettete Listen kennenlernte, hatte sie eine Idee und implementierte eine Datenstruktur, welche das sortierte Einfügen effizienter erledigen kann! Birke's Datenstruktur hat zwei Verweise pro Kettenglied, so wie eine doppelt verkettete Liste auch. Allerdings merkt sich die Klasse beim Erstellen einen `Comparator` und kann alle enthaltenen Elemente damit vergleichen. Beim Einfügen werden kleinere Elemente mit dem Verweisen nach vorne eingefügt und größere Elemente mit den Verweisen nach hinten!

Birke's Datenstruktur mit Klassennamen `Birke` sollte diesem Übungsblatt beliegen.

- a) Was gibt das folgendes Hauptprogramm aus? Skizzieren Sie auch grob das Speicherbild!

```
Comparator<Integer> cmp = Comparator.naturalOrder();
Birke<Integer> birke = new Birke<>(cmp);
birke.insert(7);
birke.insert(3);
birke.insert(5);
birke.insert(4);
birke.insert(6);
for(Integer i : birke) { System.out.print(i+", "); }
```

- b) Was ändert sich an Ausgabe und Speicherskizze, wenn vor der Schleife noch die Befehle `birke.insert(9);` und `birke.insert(8);` eingefügt werden?
- c) Birke hat tatsächlich eine nützliche Datenstruktur gefunden; doch leider ist diese als Liste denkbar schlecht geeignet. Diskutieren Sie, warum Birke's Datenstruktur keine gute Liste ist!



- d) Implementieren Sie die Methode `public Iterator<E> sortedIterator()` welche mit `//TODO Ihre Aufgabe!` markiert ist. Diese Methode soll einen Iterator zurückliefern, mit dem man alle Elemente in Birke's Datenstruktur vom kleinsten bis zum größten durchgehen kann.

Die Anweisung `while(iter.hasNext()) System.out.print(iter.next()+" ");` sollte dann für das Beispiel aus der ersten Teilaufgabe `3, 4, 5, 6, 7,` ausgegeben.

Sie können dem Muster des vorhandenen Iterators folgen. Eventuell finden Sie die Aufgabe aber leichter, wenn Sie nach dem Muster der Methode `size` vorgehen, wobei Sie anstatt einem `int` eine frische `LinkedList<E>` einsetzen, deren Iterator Sie dann einfach zurückgeben.

- e) Diskutieren Sie, was diese Datenstruktur gut kann! Betrachten Sie zuerst die Methoden `contains` und `insert`: Was ist deren Laufzeitkomplexität?

H13-1 *Generisches Mapping* (5 Punkte; Abgabe: Multipliziere.java, Map.java)

Gegeben ist folgendes Interface: `public interface Function<X,Y> { public Y apply(X x); }`

- a) Schreiben Sie eine Klasse `Multipliziere`, welche das Interface `Function<Integer,Integer>` implementiert. Die Methode `apply` soll ein gegebenes Argument mit einem festen Faktor multiplizieren. *Beispiel:*

```
Multipliziere maldrei = new Multipliziere(3);
Multipliziere malvier = new Multipliziere(4);
System.out.print(maldrei.apply(malvier.apply(5))); // druckt 60
```

- b) Schreiben Sie eine statische, generische Methode `map`, welche zwei Argumente bekommt: Das erste Argument ist Objekt, welche das Interface `Function<X,Y>` implementiert; das zweite Argumente hat den Typ `LinkedList<X>`. Als Ergebnis soll die Methode eine verkettete Liste mit Elementen aus `Y` zurückgeben. Jedes Element der Ergebnisliste wurde durch Anwendung des ersten Arguments auf ein Element der Argumentliste berechnet; die Reihenfolge entspricht der Argumentliste. *Beispiel:*

```
Function<Integer,Boolean> even = new Even();
System.out.println(map(even,xs));
```

Für die Liste `xs = [72, 9, 21, 174, 6, 93]` wird in diesem Beispiel `[true, false, false, true, true, false]` gedruckt, wobei die Klasse `Even` eine Funktion implementiert, welche `true` nur für gerade Argumente zurück gibt.

Abgabe: Lösungen zu den Hausaufgaben können bis Sonntag, den 4.2.18, mit UniWorX nur als `.zip` abgegeben werden. Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip/).