Algorithmen und Programmierung 2, SS 2016 — 12. Übungsblatt

Abgabe bis Freitag, 1. Juli 2016, 12:00 Uhr (Aufgabe 53 korrigiert am 7. Juli)

53. Verkettete Liste mit Zeiger zum Listenende, Programmieraufgabe, 10 Punkte

    Erweitern Sie die Klasse `VerketteteListe` von Aufgabe 51, indem Sie eine Unterklasse `VerketteteSchlange` erstellen, die die folgenden zusätzlichen Operationen *in konstanter Zeit* unterstützt.

    (a) `public void anhängen(int i)`. Fügt einen neuen Knoten mit Wert i am Ende der Liste ein.

    (b) `public void anhängen(VerketteteSchlange L2)`. Hängt die verkettete Liste `L2` an das Ende der Liste an und vereinigt somit die beiden Listen zu einer Liste.

    Die neue Klasse soll auch alle bisherigen Operationen weiterhin unterstützen. Dazu müssen Sie gegebenenfalls die alten Methoden überschreiben. Verwenden Sie dabei möglichst viele Methoden der ursprünglichen Klasse, indem Sie zum Beispiel etwa `super.einfüge()` aufrufen, und programmieren Sie nur das neu, was nötig ist. Achten Sie darauf, dass die Operationen auch bei leeren Listen korrekt funktionieren.

54. Zusatzfrage, 0 Punkte

    Kann man die Liste `L2` noch verwenden, nachdem sie in der Methode `anhängen` von Aufgabe 53b „verbraucht“ worden ist?

55. Turm von Hanoi, Programmieraufgabe, 10 Punkte

    Schreiben Sie ein Java-Programm, das den Turm von Hanoi löst. Verwenden Sie dafür die Klasse `hanoi.Turm` aus dem Paket `hanoi`[1] Die drei Stifte sind mit `'a'`, `'b'`, und `'c'` bezeichnet. Das Programm `TestTurm`[2] zusammen mit der Klasse `HanoiLoesung`[3] zeigt ein Beispiel, wie die Klasse `Turm` verwendet wird. In der Klasse `Turm` sind folgende Methoden implementiert (siehe Aufgabe 58b für die Bedeutung von „`final`“):

```
package hanoi;
public class Turm {
   public Turm(int [] anfangsturm) throws TurmException
     /** anfangsturm enthält die Größen aller Scheiben, aufsteigend sortiert.
         Gleiche Scheiben sind erlaubt. Die Scheiben werden auf Stift 'a' gesetzt. */
   public final int setzeUm(char s, char z) throws TurmException; { ...
     /** bewegt die oberste Scheibe vom Stift s (Start) zum Stift z (Ziel).
         Vorbedingungen:
           • s, z = 'a', 'b', oder 'c', und s ≠ z.
           • Stift s ist nicht leer.
           • Oberste Scheibe d auf Stift s ≤ oberste Scheibe auf z, oder z ist leer.
         Der Rückgabewert ist d.     */
   }
   public final boolean fertig(); // testet, ob die Stifte 'a' und 'b' leer sind
   public int[] anfangsturm();    // erstellt eine Kopie des Ausgangsturms
   public final int bewegungen(); // die Anzahl der bisherigen Bewegungen
}
```

[1] Speichern Sie die Dateien `Turm.class` und `TurmException.java` von `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/hanoi/` in ein Unterverzeichnis mit Namen `hanoi`.

[2] `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/TestTurm.java`

[3] `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/HanoiLoesung.java`

Schreiben Sie Ihre eigene Klasse `HanoiLoesung`, die mit der Methode `anfangsturm` den Turm untersucht und anschließend die korrekte Folge von Bewegungen durchführt. (Diese Klasse wird dann von einem Testprogramm aufgerufen, das ähnlich wie das Programm `TestTurm` arbeitet. Schreiben Sie trotzdem Ihre eigene Testsuite.)

56. Schnittstellen, 0 Punkte

    Schreiben sie eine Schnittstelle `interface TurmInterface` für Klassen, die die Methoden von `Turm` von Aufgabe 55 unterstützen.

57. Generische Methoden, Programmieraufgabe, 10 Punkte

    Außer Klassen können auch Methoden in JAVA mit einem Datentypen parametrisiert werden. Schreiben Sie eine generische statische Methode `swap`, die zwei gegebene Elemente eines Feldes miteinander vertauscht:

```
class Swap {
    public static <T> void swap(...) ... Definition von swap ...
    public static void main(String[] args) {
        Integer [] a = {1,2,0,2,3};
        Swap.<Integer>swap(a, 2, 3); // vertauscht a[2] mit a[3]
        swap(a, 0, 4); // auch ohne explizite Angabe des Datentyps
        // Der Typ wird aus dem Argument a erschlossen. (Typinferenz)
    }
}
```

58. Endgültige Klassen, 0 Punkte

    (a) Studieren Sie den Quellcode `Turm.java`[4], der eine mögliche Implementierung der Klasse `Turm` zeigt. Gibt es eine Möglichkeit das Testprogramm von Aufgabe 55 zu überlisten, sodass man auch mit einer falschen Lösung erreichen kann, dass `fertig()` am Ende den Wert `true` zurückgibt?

    (b) Nehmen wir an, wir dürfen die Klasse `Turm` erweitern und die Methode

    `HanoiLoesung.löse(t)`

    auch mit einer Unterklasse von `Turm` aufrufen. Methoden, die als „`final`“ deklariert werden, können von einer Unterklasse nicht überschrieben werden. Gibt es trotzdem einen Weg, das Testprogramm zu überlisten?

    (c) Kann man auch erreichen, dass die Methode `bewegungen` nicht die korrekte Zahl an Aufrufen von `setzeUm` meldet?

    (d) Wie kann man diese Schlupflöcher gegebenenfalls stopfen?

    (e) Warum ist es nicht notwending, die Methode `anfangsturm` als `final` zu deklarieren? Warum wurde der Konstruktor `Turm` nicht als `final` deklariert?

59. Die `Collections`-Klassen von JAVA, 0 Punkte

    Studieren Sie die Dokumentation der Klasse `PriorityQueue`[5] und finden Sie eine möglichst einfache Möglichkeit, mit Hilfe der Methoden und Klassen des `Collection`-Rahmens die Elemente einer Prioritätswarteschlange `pq` in absteigend sortierter Reihenfolge auszudrucken,

    (a) wenn die Prioritätswarteschlange `pq` dabei nicht verändert werden soll,

    (b) wenn die Prioritätswarteschlange `pq` danach nicht mehr gebraucht wird.

[4] `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/Turm.java`

[5] siehe zum Beispiel `http://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html`, `http://docs.oracle.com/javase/tutorial/collections/intro/`, oder `http://docs.oracle.com/javase/tutorial/collections/interfaces/queue.html`.