Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Einführung in die Informatik
Sommersemester 2011

Prof. Dr. Wolfram Burgard
PD Dr. Cyrill Stachniss
Dipl. Inf. Axel Rottmann
Dipl. Inf. Markus Kuderer

Übungsblatt 12

Abgabe bis Donnerstag, 04.08.2011, 12:00 Uhr

Aufgabe 12.1

Implementieren Sie für die Klasse `SingleLinkedList` aus der Vorlesung die Methode `splitEvenOdd`, die die Liste in zwei neue Listen aufteilt. Die bestehende Liste enthält danach keine Elemente mehr. Die neue Liste `even` soll alle Elemente enthalten, die bisher an gerader Position gespeichert waren. Die neue Liste `odd` entsprechend die Elemente, die bisher an ungerader Position gespeichert waren. Implementieren Sie Unittests mittels `JUnit`, um Ihre Methode zu testen.

```
public void splitEvenOdd(SingleLinkedList even, SingleLinkedList odd) {
   ...
}
```

Aufgabe 12.2

Um binäre Suchbäume (siehe Vorlesungsfolien) für die effiziente Datenspeicherung einzusetzen, muss für dessen Objekte eine Ordungsrelation definiert sein. Java bietet für Objekte dieser Art das Interface `Comparable<Object>`. Alle Klassen, die dieses Interface implementieren, müssen die Methode `int compareTo(Object o)`[1] besitzen. Der Aufruf `A.compareTo(B)` gibt für zwei `Comparable<Objekt>`-Objekte A, B einen Wert größer 0 zurück, wenn $A > B$, einen Wert kleiner 0 wenn $A < B$ und 0 wenn $A = B$.

Betrachten Sie den Auszug der Klasse `BinaryTree`, die Objekte des Typs `Comparable<Object>` enthält.

```
class BinaryTree {
    BinaryTree(Comparable<Object> o) {
         content  = o;
         left = null;
         right = null;
    }
    ...
    private Comparable<Object> content;
    private BinaryTree left;
    private BinaryTree right;
}
```

[1] http://download.oracle.com/javase/6/docs/api/java/lang/Comparable.html

Erweitern Sie `BinaryTree` um folgende Methoden:

- `public Comparable<Object> getContent()`
- `public BinaryTree getLeft()`
- `public BinaryTree getRight()`
- `boolean contains(Comparable<Object> o)`
  Überprüft, ob ein Knoten mit dem Inhalt `o` existiert (d.h. ein Knoten, für den die Methode `compareTo(o)` 0 zurückgibt).
- `void insert(Comparable<Object> o)`
  Fügt einen Knoten mit Inhalt `o` in den Baum ein, sofern noch kein Knoten mit diesem Inhalt existiert. Dabei besteht der linke Teilbaum ausschließlich aus Knoten mit kleineren Inhalten (`compareTo(o)` < 0) als die Wurzel und der rechte Teilbaum entsprechend nur aus größeren.
- `void delete(Comparable<Object> o)`
  Löscht den Knoten mit dem Inhale `o`, falls er existiert. Durch welchen Schlüssel des Baums kann man diesen ersetzen, ohne die gesamte Struktur zu verändern?
- `void preorder()`
  Gibt zuerst den Inhalt der Wurzel und dann die Inhalte der linken und rechten Teilbäume aus.
- `void inorder()`
  Gibt zuerst den Inhalt des linken Teilbaums, dann den Inhalt der Wurzel und danach Inhalt des rechten Teilbaums aus.

Aufgabe 12.3

Erweitern Sie die Klasse `Student` von Übungsblatt 11 so, dass sie das Interface `Comparable<Object>` implementiert. Die dafür notwendige Methode `compareTo` soll dabei zunächst den Nachnamen der Studenten lexikographisch vergleichen. Falls der Nachname der Studenten identisch ist, soll der Vorname der Studenten für den Vergleich verwendet werden.

Hinweis: Casten Sie das übergebene Argument zu Student:

```
public int compareTo(Object o) {
  Student s = (Student) o;
  ...
  }
```

Aufgabe 12.4

Testen Sie Ihre Implementierung mit Hilfe der Klasse `TestStudentTree`, die Sie auf der Vorlesungsseite finden. Darin werden mehrere Studenten in einen Binärbaum eingefügt. Welche Methode aus 12.2 müssen Sie in Zeile 22 verwenden, um die `Student`-Objekte in sortierter Reihenfolge auszugeben?