FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

Objektorientierte Programmiertechniken
LVA 185.A01, VL 2.0, 2011 W

4. Übungsaufgabe

Themen:
Vererbung, Untertypbeziehungen, Zusicherungen

Termine:

Ausgabe:	16.11.2011
reguläre Abgabe:	23.11.2011, 13:45 Uhr
nachträgliche Abgabe:	30.11.2011, 13:45 Uhr

Abgabeverzeichnis:
Gruppe/Aufgabe4

Programmaufruf:
java Test

Grundlage:
Die ersten beiden Kapitel im Skriptum, Schwerpunkt auf Kapitel 2

Aufgabe

Welche Aufgabe zu lösen ist:
Es sollen folgende Typen (also Klassen, abstrakte Klassen oder Interfaces) definiert werden:

- *StringTree* repräsentiert einen binären Baum, dessen Knoten jeweils mit einem String gelabelt sind. *StringTree* bietet die folgenden Methoden an:
  - *boolean contains(String node)* gibt *true* zurück, wenn der Baum einen Knoten enthält, der mit *node* gelabelt ist, ansonsten *false*.
  - *String search(String node)* gibt den Pfad an, über den man den mit *node* gelabelten Knoten erreichen kann. Der Pfad ist dabei als Sequenz von Schritten codiert, die man vom Wurzelknoten aus zum gesuchten Knoten gehen muss. Ein Schritt ist entweder "*left*" oder "*right*". Der Pfad zum Wurzelknoten wäre also "", der Pfad zu dessen linkem Kind "*left*", der Pfad zu dessen rechtem Kind "*left right*", und so weiter. Enthält der Baum den gesuchten Knoten nicht, soll *search* eine Fehlermeldung (wie etwa "*Knoten wurde nicht gefunden*") zurückliefern. Enthält der Baum mehrere Konten mit demselben Label, so ist undefiniert, welcher davon

gefunden wird.
    - *void add(String node)* fügt einen neuen Knoten mit dem Label *node* in den Baum ein. Die genaue Stelle, an welcher der Knoten eingefügt wird, wird in Untertypen festgelegt.
    - *String toString()* gibt eine Repräsentation des gesamten Baums als String zurück. Die Struktur des Baums sollte dabei erkennbar sein. Ein Beispiel für eine solche Repräsentation wäre etwa eine Directory-ähnliche Struktur:

      ```
      a
      - b
        - c
        - d
      - e
      ```

      In diesem Beispiel ist "*a*" das Label des Wurzelknotens, "*b*" das des linken Kindes und "*e*" das des rechten Kindes. Der Knoten "*b*" hat "*c*" und "*d*" als Kinder.
- *ReplaceableTree* ist ein Untertyp von *StringTree*, bei dem ein Teilbaum durch einen anderen Teilbaum ersetzt werden kann:
    - *void replace(String position, String subTree)* ersetzt den Teilbaum an Position *position* durch den Teilbaum *subTree*. Hierbei hat *position* die Form des Pfads zum Wurzelnoten des zu ersetzenden Teilbaums, wie dieser von *search* zurückgegeben würde, und *subTree* hat die Form der *toString*-Ausgabe eines entsprechenden Baums. Im Fehlerfall (*position* existiert nicht, einer der Parameter hat nicht die korrekte Form) soll keine Ersetzung durchgeführt werden.
    - *search* durchsucht den Baum mittels eines beliebigen Suchalgorithmus, wie etwa Tiefen- oder Breitensuche.
    - *add* fügt ein neues Element so nahe wie möglich an der Wurzel des Baumes ein, wobei die Pfade aller schon bestehenden Knoten unverändert bleiben. Gibt es mehrere freie Stellen gleich weit von der Wurzel entfernt (aber keinen näheren freien Platz), so wird der neue Knoten an der am weitesten links stehenden freien Stelle eingefügt.
- *SortedTree* ist ebenso ein Untertyp von *StringTree*. Die Label der Knoten in jedem linken Teilbaum müssen kleiner als die der Knoten im rechten Teilbaum sein, und die im rechten Teilbaum müssen größer oder gleich denen im linken Teilbaum sein. (Tipp: Mittels *compareTo(String anotherString)* lassen sich zwei Strings vergleichen).
    - *search* soll sich diese Sortierung zu Nutze machen, um das Durchsuchen des Baums in logarithmischer Zeit (in Abhängigkeit von der Anzahl der Knoten) zu ermöglichen.
    - *add* soll einen neuen Knoten so in den Baum einfügen, dass die Sortierung der Knoten gewahrt bleibt.
    - *String traverse()* gibt das Ergebnis einer Traversierung des Baums zurück. Die Label der Knoten sollen dabei, durch Leerzeichen getrennt, in der Reihenfolge, die in Untertypen vorgegeben ist, ausgegeben werden.
- *PreorderTree* ist ein Untertyp von *SortedTree*, in dem *traverse* den Baum in *Preorder* traversiert. (Zuerst wird das Label des Knotens

selbst ausgegeben, dann die Preorder-Traversierung des linken Teilbaums, dann die Preorder-Traversierung des rechten Teilbaums.)
- *InorderTree* ist ein Untertyp von *SortedTree*, in dem *traverse* den Baum in *Inorder* traversiert. (Zuerst wird die Inorder-Traversierung des linken Teilbaums ausgegeben, dann das Label des Knotens selbst und schließlich die Inorder-Traversierung des rechten Teilbaums.)
- *PostorderTree* ist ein Untertyp von *SortedTree*, in dem *traverse* den Baum in *Postorder* traversiert. (Zuerst wird die Postorder-Traversierung des linken Teilbaums ausgegeben, dann die Postorder-Traversierung des rechten Teilbaums und schließlich das Label des Knotens selbt.)
- *IntTree* ist ein binärer Baum, dessen Knoten mit Zahlen vom Typ *int* gelabelt sind. Ansonsten verhalten sich Instanzen wie die von *ReplaceableTree* (Teilbäume ersetzbar, Einfügen möglichst nahe an der Wurzel) und bieten auch entsprechende Methoden an.

Versehen Sie (abstrakte) Klassen und Interfaces mit allen notwendigen Zusicherungen und stellen Sie sicher, dass Sie nur dort eine Vererbungsbeziehung (extends oder implements) verwenden, wo tatsächlich eine Untertypbeziehung besteht.

Schreiben Sie eine Klasse *Test* zum nicht-interaktiven Testen Ihrer Lösung. Dabei soll jede von Ihnen implementierte Methode mindestens einmal ausgeführt werden. Achten Sie auch besonders auf die Abdeckung von Grenzfällen und Ausnahmesituationen durch Testfälle.

Schreiben Sie (abgesehen von Klassen, die nur zum Testen dienen) so wenig Programmcode wie möglich. Sie können so viele zusätzliche (abstrakte) Klassen und Interfaces einführen, wie Sie für die Wiederverwendung von Code als vorteilhaft erachten.

Wie die Aufgabe zu lösen ist:
Verwenden sie keine Arrays oder vorgefertigte Container-Klassen (wie LinkedList, Stack, Queue, etc.) zur Lösung dieser Aufgabe. Wenn Sie Container-Klassen benötigen, müssen Sie diese selbst schreiben.

Verwenden Sie außerdem keine Generizität. Diese wird erst in späteren Aufgaben behandelt.

Aufgabe 5 wird auf der Lösung von Aufgabe 4 aufbauen. Achten Sie daher besonders darauf, dass Ihr Code vollständig und gut erweiterbar ist.

Was im Hinblick auf die Beurteilung wichtig ist:
Schwerpunkte der Beurteilung liegen auf der Verwendung von Vererbung, Untertypbeziehungen und Zusicherungen. Dazu gehört auch die Vermeidung unnötigen oder wiederholt vorkommenden Programmcodes. Fehler in diesen Bereichen führen zu sehr hohen Punkteabzügen. Daneben werden Fehler in der Sichtbarkeit von Klassen, Variablen und Methoden auch mit deutlichen Punkteabzügen

bedacht. Punkteabzüge gibt es natürlich auch für unzureichendes Testen, fehlende oder falsche Funktionalität und schlampige Strukturierungen des Codes.

Es werden nach wie vor keinerlei Ausnahmen bezüglich des Abgabetermins gemacht. Bitte verwenden Sie keine Pakete oder andere Unterverzeichnisse im Abgabeverzeichnis. Schreiben Sie nicht mehr als eine Klasse in jede Datei (ausgenommen geschachtelte Klassen), halten Sie sich an übliche Namenskonventionen in Java und verwenden Sie die Namen, die in der Aufgabenstellung vorgegeben sind. Damit erhöhen Sie die Lesbarkeit Ihrer Programme ganz wesentlich und vermeiden unnötige Punkteverluste.

Anfang | HTML 4.01 | letzte Änderung: 2011-11-16 (Puntigam)