Übungen zur Vorlesung
Praktische Informatik II
Sommersemester 2001

Serie 2

Listen und Bäume

Aufgabe 1: Listen (40%)

Auf dem Aufgabenzettel der Serie 7 der Praktischen Informatik 1 im letzten Semester wurde die algebraische Sicht auf Listen eingeführt. In dieser Aufgabe sollen auf Basis dieser Sichtweise Listen von Strings objektorientiert implementiert werden.
Jedes Element der Liste ist eine Instanz der Listenklasse `MyList`, die Verweise auf den Rest der Liste und einen String enthalten sollen. Um dem Konzept des *Data-Hiding* zu genügen, sollen alle Daten der Objekte `private` deklariert sein. Alle Methoden, die auf den Listen–Objekten arbeiten sind nicht–destruktiv, d.h. Listen, die als Parameterwerte der Objekt–Methoden verwendet werden, bleiben mit unverändertem Inhalt erhalten.
Implementieren Sie hierzu folgende Methoden:

MyList() **[Konstruktor]**
erzeugt eine leere Liste, die keine Verweise auf Strings oder ein anderes Listenobjekt enthält.

MyList(String, MyList) **[Konstruktor]**
fügt den übergebenen String vorne an die Liste an.

String head() **[Selektor]**
gibt den String des ersten Elementes der Liste zurück.

MyList tail() **[Selektor]**
gibt die Restliste, d.h. alles bis auf das erste Element einer Liste zurück.

boolean isEmpty() **[Recognizer]**
gibt zurück, ob es sich bei einem Listen-Objekt um eine leere Liste handelt.

Implementieren Sie zusätzlich noch die folgenden Methoden für die Klasse `MyList`. Diese Methoden sollen lediglich die fünf zuvorgenannten Methoden verwenden und nicht auf die privaten Daten der Objekte zugreifen.

boolean isSorted()
überprüft, ob eine Liste sortiert ist.

MyList merge(MyList)
erzeugt eine neue sortierte Liste durch Zusammenfügen zweier sortierter Listen.

void print()
gibt den Inhalt der Liste auf den Bildschirm aus.

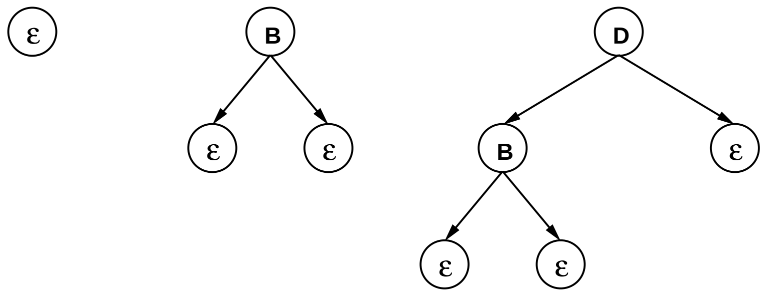
Zum Testen lesen Sie beliebige Strings zeilenweise aus einer Textdatei ein und füllen die Liste mit diesen Daten. Hierfür können Sie die Klasse `BufferedReader` aus dem Java Package `java.io` verwenden.

Aufgabe 2: Binäre Bäume (60%)

Bäume sind ein in der Informatik vielfach verwendeter Datentyp zur strukturierten (oder sortierten) Speicherung von Daten. Sie stellen eine Verallgemeinerung des Listenkonzepts dar. Die hier betrachtete Variante von binären Bäumen läßt sich folgendermaßen charakterisieren:
Ein binärer Baum ist entweder

- ein leerer Baum ε oder
- ein Knoten, welcher einen linken und einen rechten Nachfolger besitzt, die wiederum jeweils binäre Bäume sind, sowie ein Datum.

Es folgen drei Beispiele für Bäume, wobei das erste Beispiel einen leeren Baum darstellt:



Der Unterschied zu Listen besteht also darin, dass nicht eine lineare Kette von Elementen angelegt wird, sondern dass für jeden Knoten *zwei* Nachfolger angegeben werden müssen. Für eine Umsetzung des Datentyps Baum in Form einer Java–Klasse bedeutet dies, dass sie ein ähnliches Aussehen wie die `MyList`–Klasse erhält, jedoch neben dem Nutzdatum einen *linken* und einen *rechten* Nachfolger besitzt.
Implementieren Sie eine Java–Klasse für binäre Bäume über Strings mit den folgenden grundlegenden Operationen:

MyTree **[Konstruktor]**
erzeugt einen leeren Baum.

MyTree(MyTree, String, MyTree) **[Konstruktor]**
erzeugt einen neuen Baum, welcher als linken Nachfolger den ersten Parameter, als rechten Nachfolger den dritten Parameter und als Wert des aktuellen Knotens den zweiten Parameter erhält.

MyTree getLeft() **[Selektor]**
liefert den linken Teilbaum des Baum–Objekts zurück.

MyTree getRight() **[Selektor]**
liefert den rechten Teilbaum des Baum–Objekts zurück.

String getData() **[Selektor]**
liefert den Wert des obersten Knotens des Baum–Objekts zurück.

boolean isEmpty() **[Recognizer]**
liefert einen Wahrheitswert zurück, der angibt, ob es sich bei dem Baum–Objekt um einen leeren Baum handelt.

Implementieren Sie darüberhinaus, aufsetzend auf diesen Basisoperationen, die folgenden komplexeren Funktionen:

MyTree insertSorted(String)
: fügt ein Element sortiert in den Baum ein. Dabei sollen alle Werte, die kleiner als der Wert des aktuellen Knotens sind, in dem linken Baum, alle anderen im rechten Baum einsortiert werden. Mehrfach vorkommende Werte werden auch mehrfach im Baum gespeichert. Das Ergebnis dieser Methode ist ein neuer Baum, der ursprüngliche Baum darf dabei nicht verändert werden.

void printSorted()
: gibt alle im Baum gespeicherten Werte sortiert auf den Bildschirm aus.

boolean existsIn(String)
: liefert genau dann `true`, wenn der übergebene Wert an beliebiger Stelle als Datum in dem Baum vorkommt.

Verwenden Sie schließlich die Lösung aus Aufgabe 1 und implementieren Sie die folgende Methode:

void importList(MyList)
: fügt alle Elemente der übergebenen Liste sortiert in den Baum des Objektes ein.

Zum Testen lesen Sie, wie in Aufgabe 1, beliebige Strings aus einer Textdatei ein und fügen die Daten in den Baum ein.

Abgabe: 9.–11. Mai 2001 in den Tutorien. Die Abgabe soll sowohl elektronisch (Programm-Quellcode) als auch in gedruckter Form (mit LaTeX gesetzter kommentierter und erläuterter Quellcode) erfolgen. Dabei ist auch auf geeignete Testfälle und deren Dokumentation zu achten!