Technische Universität Darmstadt
Fachbereich Informatik
Prof. Dr. Johannes Fürnkranz

Allgemeine Informatik 2 im SS 2007

Übungsblatt 5

Bearbeitungszeit: 30.05. bis 05.06.2007

Aufgabe 1: Mengen

Die Klasse **`Menge`** soll eine mathematische Menge beschreiben: sie enthält beliebig viele Elemente (hier ganze Zahlen), man kann Elemente hinzufügen (sofern sie noch nicht vorhanden sind), Elemente löschen und abfragen, ob ein Element enthalten ist oder wie groß die Menge ist.
Die interne Struktur unserer Menge ist die einer **verketteten Liste**: jedes Element enthält einen Verweis auf das nächste Element, beim letzten Element der Liste zeigt dieser Verweis ins Nichts: **`null`**. Der einzige Unterschied zu einer normalen Liste ist, dass die Methode **`fuegeEin(int data)`** sicherstellt, dass das einzufügende Element nicht bereits vorhanden ist.

```
public class Menge {
  private Element firstElement;

  public int groesse() { … }
  public boolean istEnthalten(int data) { … }
  public boolean fuegeEin(int data) { … }
  public boolean entferne(int data) { … }
}

class Element {
  int data; // Inhalt des Elements
  Element next; // Nächstes Element
}
```

Die Klasse **`Menge`** (und die Klasse **`Element`**) finden Sie auf unserer Webseite im Bereich Übungen als BlueJ-Projekt. Sie müssen die Datei **uebung05.zip** zuerst entpacken (im Terminal `unzip uebung05.zip` eingeben). Starten Sie dann BlueJ und öffnen Sie das Projekt **uebung05** – fangen Sie bitte kein eigenes Projekt an!

Erweitern Sie nun die Klasse **`Menge`** um folgende Methoden:

a) **`public String toString()`**
Gibt die Struktur der Menge als Text zurück, und zwar in folgender Form:
`“4 -> 2 -> 7 -> 3 -> (X)“`
Begonnen beim ersten Element wird jedes Element und dahinter **`“ -> “`** an einen zunächst leeren **`StringBuffer`** gehängt (mit der Methode **`append`** aus der Klasse **`StringBuffer`**). Der **`null`**-Verweis des letzten Elements wird als **`“(X)“`** angehängt. Dann wird der **`StringBuffer`** als **`String`** zurückgegeben (Methode **`toString()`** aus der Klasse **`StringBuffer`**).

b) **`public void union(Menge m)`**

Vereinigt die aktuelle Menge mit der übergebenen Menge **`m`**. Nach dem Aufruf soll die übergebene Menge **`m`** die Elemente aus beiden Elementen enthalten. Dazu fügen Sie einfach der Reihe nach alle Elemente der aktuellen Menge in die Menge **`m`** ein.

c) **FREIWILLIGE FLEISSAUFGABE: `public void intersect(Menge m)`**
Bildet die Schnittmenge aus der aktuellen und der übergebenen Menge **`m`**. Nach dem Aufruf soll die übergebene Menge die Schnittmenge enthalten. Sie müssen dazu nacheinander beide Mengen durchlaufen und alle Elemente aus **`m`** entfernen, die nicht auch in der aktuellen Menge enthalten sind. Um die Menge **`m`** zu durchlaufen, benötigen Sie die Methode **`Element firstElement()`**, da Sie auf das private Attribut **`firstElement`** nicht direkt zugreifen können.

Mit der Methode **`test()`** der Klasse **`MengenTest`** können Sie die Funktion Ihrer Methoden überprüfen.
Machen Sie sich außerdem klar, warum nach dem Aufruf der Methoden **`union`** (oder **`intersect`**) deren Ergebnis in der übergebenen Menge vorliegt (Stichwort: **Referenzen**).

Aufgabe 2: Binäre Bäume

Als rekursiven Datentyp haben Sie bereits die **Liste** (bzw. den Sonderfall **Menge**) als eine Anzahl von verketteten Elementen kennen gelernt. Ein anderer in der Informatik sehr bekannter rekursiver Datentyp ist der **binäre Baum**. Jedes Element hat hier nicht ein Nachfolgerelement, sondern zwei - wobei beide natürlich auch **`null`** sein können.
In einem Baum heißt jedes Element **Knoten**. Alle Knoten ohne Nachfolger (bei denen also beide Nachfolgerelemente **`null`** sind) heißen **Blätter**, das oberste Element ist die **Wurzel**. Jeder Knoten kann auch als Wurzel eines so genannten Teilbaums aufgefasst werden.

Ihre Aufgabe ist es, einen alphabetisch sortierten Baum zu implementieren. Dabei ist der linke Nachfolger kleiner oder gleich dem aktuellen Element, der rechte größer (vgl. Abb. 1).

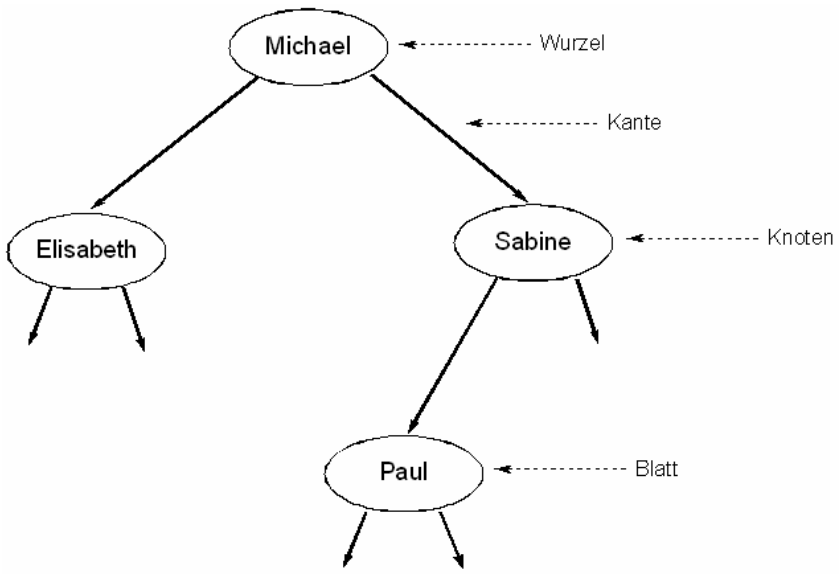

Abb. 1: Struktur eines binären Baums

Im BlueJ-Projekt **uebung05** ist auch ein Grundgerüst der Klasse **`Tree`** enthalten.

Diese enthält bereits:

- die Attribute (Daten-String, linker und rechter Teilbaum als **`Tree`**),

- einen Konstruktor, der einen String übergeben bekommt und linken und rechten Teilbaum mit **`null`** initialisiert,
- die Methoden **`hasLeft()`** und **`hasRight()`** zum Überprüfen, ob der Knoten einen linken bzw. rechten Teilbaum hat.

Erweitern Sie nun die Klasse **`Tree`** um folgende Methoden:

a) **`public void insert(String data)`**
Fügt ein neues Element in den Baum ein.
Dazu vergleichen Sie zuerst den übergebenen String mit dem eigenen String. Ist er alphabetisch größer, kommt er in den rechten Teilbaum, sonst in den linken.
Dann testen Sie, ob links bzw. rechts schon ein Teilbaum existiert (mit den Methoden **`hasLeft()`** bzw. **`hasRight()`**).
Wenn nein, legen Sie mit dem übergebenen String ein neues Objekt der Klasse **`Tree`** an und setzen es als linken bzw. rechten Teilbaum.
Wenn ja, übergeben Sie das Einfügen einfach rekursiv an den linken bzw. rechten Teilbaum.
Zum alphabetischen Vergleich verwenden Sie die Methode **`compareTo`** der Klasse **`String`** (siehe **Java-API**).

b) **`public String toString()`**
Gibt den Inhalt des Baums als Text zurück. Dazu legen Sie zuerst einen leeren **`String`** an. Wenn ein linker Teilbaum existiert, hängen Sie diesen (**`left.toString()`**) an. Dann hängen Sie den eigenen String und ein Leerzeichen an, danach den rechten Teilbaum, sofern er existiert. Zuletzt geben Sie den **`String`** zurück.
Diese Reihenfolge (linker Teilbaum, eigener String, rechter Teilbaum) nennt sich „**in-order**“. Bei „**pre-order**“ würde zuerst der eigene String ausgeben, bei „**post-order**“ zuletzt.

Mit der Methode **`test()`** der Klasse **`TreeTest`** können Sie die Funktion Ihrer Methoden überprüfen.

Viel Spaß!