TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Praktikum Grundlagen der Programmierung
Prof. Dr. Helmut Seidl

WS 2007/2008
Aufgabenblatt 6
23. November 2007

Abgabe: Montag, 03.12.07 per Mail an den jeweiligen Tutor

Praktikum Grundlagen der Programmierung

Aufgabe 33 (Ü) Testen auf Gleichheit

Java bietet zum Testen auf Gleichheit den Operator == und die für alle Objekte definierte öffentliche Methode **boolean** `equals(Object obj)` an.

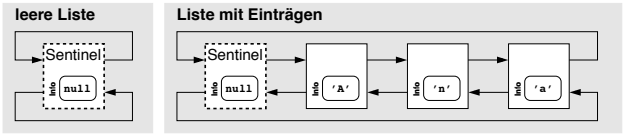
```
int a = 1;
int b = 1;
String c = new String("ab");
String d = new String("a" + "b");
```

a) Gegeben sei folgendes Code-Fragment:
   - Welche Vergleiche sind möglich? Welche Ergebnisse liefern diese Vergleiche? Schreiben Sie ein kleines Testprogramm, um dies herauszufinden.
   - Wie unterscheiden sich `int`-Variablen von `String`-Variablen?
   - Was unterscheidet den Operator == von der Methode `equals()`?

b) Implementieren Sie eine Klasse `Person` mit Attributen für Vor- und Nachname in der eine öffentliche Methode **boolean** `equals(Person p)` sinnvoll definiert ist.

Aufgabe 34 (Ü) Doppelt verkettete Liste

In dieser Übung soll eine *doppelt verkettete* Liste realisiert werden. In einer solchen Liste wird in jedem Listen-Eintrag (Klasse `Entry`) eine Referenz auf den Vorgänger, eine Referenz auf den Nachfolger sowie eine Referenz auf die eigentliche Information gespeichert. Wir implementieren die doppelt verkettete Liste zyklisch. Dabei gibt es einen ausgezeichneten Listen-Eintrag, der keine Information speichert (das so genannte *Sentinel-Element*) und sowohl vor dem ersten als auch hinter dem letzten eigentlichen Listen-Eintrag steht (siehe Abbildung). Der erste (bzw. letzte) eigentliche Listen-Eintrag ist dann der Nachfolger (bzw. Vorgänger) des Sentinel-Elementes. Insbesondere wird die leere Liste wie in der nachfolgenden Abbildung illustriert repräsentiert.



Implementieren Sie die Klassen `Entry` und `LinkedList`. Ein Objekt der Klasse `LinkedList` soll eine zyklisch doppelt verkettete Liste repräsentieren und muss dementsprechend eine Referenz auf das Sentinel-Element halten. In der Klasse `LinkedList` soll Folgendes definiert sein:

a) Ein Konstruktor zum Erzeugen von leeren Listen.
b) Eine Methoden **void** `add(String s)`, die den String `s` am Ende der Liste anhängt.
c) Eine Methode `String toString()`, die eine Textrepräsentation der Liste zurück liefert.

Aufgabe 35 (Ü) Textdokumente

In dieser Aufgabe sollen Sie ein Klassenmodell zur Repräsentation von Textdokumenten erstellen. Ein Textdokument besteht aus einer Liste von Absätzen. Der textuelle Inhalt eines Absatzes soll einfach als String gespeichert werden. Zu jedem Absatz soll weiterhin gespeichert werden, ob er normal, fett oder kursiv dargestellt werden soll.

Ihr Klassenmodell soll folgende Operationen unterstützen:

a) Das Hinzufügen eines Absatzes.
b) Die Generierung einer String-Darstellung eines Textdokuments, wobei Absätze durch Leerzeilen getrennt werden sollen.

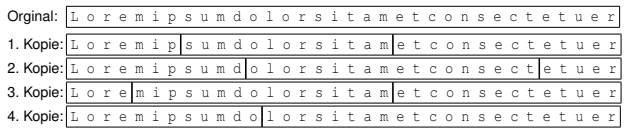
Aufgabe 36 (H) Doppelt verkettete Liste (6 Punkte)

Erweitern Sie Ihre Implementierung aus Aufgabe 34 um folgende Methoden:

a) `int size()`, die die Länge der Liste zurück liefert;
b) `void addFirst(String s)`, die den String `s` am Anfang der Liste einfügt;
c) `Entry first()` und `Entry last()`, die, falls die Liste nicht leer ist, den ersten bzw. den letzten echten Listeneintrag zurückliefern. Ist die Liste leer, soll `null` zurückgeliefert werden;
d) `Entry next(Entry e)` und `Entry prev(Entry e)`, die den Vorgänger- bzw. Nachfolger-Eintrag zurückliefern. Ist dieser das Sentinel-Element, soll **`null`** zurückgeliefert werden;
e) `void remove(Entry e)`, die das übergebene `Entry`-Objekt aus der Liste entfernt;
f) `void removeFirst()` und `void removeLast()`, die den ersten bzw. letzten Eintrag löschen.

Aufgabe 37 (H) Texte rekonstruieren (10 Punkte)

Für diese Aufgabe stellen wir uns folgendes Szenario vor: Ein Text wurde mehrmals kopiert. Jede Kopie des Textes ist leider zerstückelt worden. Ihre Aufgabe ist es eine Java-Klasse `Retter` zu definieren, die es ermöglicht aus solchen (Text-)Fragmenten den ursprünglichen Text wiederherzustellen. Die Ausgangssituation lässt sich also wie folgt illustrieren:



Verkleben zweier Fragmente. Betrachte zunächst zwei Fragmente f_1 und f_2. Z.B. `Loremipsumd` und `sumdolorsitam`. Jetzt wird ein möglichst langer String gesucht, der End-Stück von f_1 und Anfangs-Stück von f_2 oder Anfangs-Stück von f_2 und End-Stück von f_1 ist. Im Bespiel ist das der String `sumd`. Die beiden Fragmente werden dann mit dem identifizierten String als Klebestelle verklebt. Im Beispiel ergibt sich `Loremipsumdolorsitam`.

Der Algorithmus. Der Algorithmus zur Rekonstruktion verwaltet eine Menge F von Fragmenten und funktioniert wie folgt. Enthält die Menge F der Fragmente nur noch ein Element, so ist man fertig. Das einzige Element in F ist der hoffentlich korrekt rekonstruierte Text. Es ist allerdings nicht garantiert, dass das Ergebnis (einzige Element in F) dem Original entspricht.

Enthält die Menge F mehr als ein Element, so wähle aus F zwei Fragmente f_1 und f_2 mit maximal langer Klebestelle, d.h. f_1 und f_2 müssen derart gewählt werden, dass, für jede anderen Fragmente f_1' und f_2' aus F, die Länge der Klebestelle von f_1' und f_2' kleiner oder gleich der Länge der Klebestelle von f_1 und f_2 ist. Verklebe die Fragmente f_1 und f_2 zu einem Fragment f. Entferne f_1 und f_2 aus F und füge f zu F hinzu. Wiederhole dies bis F nur noch genau ein Element enthält.

Ihre Klasse Retter. Ihre Klasse `Retter` soll Folgendes zur Verfügung stellen:

a) Eine Methode `void addFragment(String f)`, die das Fragment `f` der Menge F hinzufügt.
b) Eine Methode zum Entfernen von Fragmenten aus der Menge F.

c) Eine Methode `String rette()`, die den rekonstruierten Text zurück liefert.

Hinweise:

a) Auf der Homepage können die Klassen `Kleber` und `Zerstueckler` heruntergeladen werden.
b) Zum Verkleben zweier Fragmente stellt die Klasse `Kleber` die statische Methode `String verklebe(String f1, String f2)` bereit, die das verklebte Fragment zurückliefert. Gibt es keine Klebestelle, so wird die Konkatenation von `f1` und `f2` (d.h. `f1 + f2`) zurückgeliefert. Die Länge der Klebestelle ergibt sich also als `f1.length() + f2.length() - Kleber.verklebe(f1,f2).length()`.
c) Um Ihre Implementierung testen zu können stellen wir Ihnen die Klasse `Zerstueckler` zur Verfügung. Die Main-Methode in der Datei `Zerstueckler.java` zeigt Ihnen am Beispiel wie diese Klasse zu verwenden ist.