Übungsblatt 10

Abgabe: 22.01.2007

Doppelt verkettete Ringlisten

Doppelt verkettete Ringlisten sind folgendermaßen konstruiert (vergleiche hierzu den zu dieser Aufgabe zur Verfügung gestellten Programmrahmen `MyList.java`):

- Jedes Listenelement `m` ist mit einer Referenz `m.n` auf den **Nachfolger** und einer Referenz `m.p` auf den **Vorgänger** versehen (sog. doppelte Verkettung).
- Jede Liste enthält immer ein sog. **Ankerelement**. Dieses trägt niemals Nutzdaten (Komponente 'data' ist die null-Referenz).
- Die leere Liste wird allein durch das Ankerelement repräsentiert, bei dem Nachfolger und Vorgänger auf sich selbst zeigen.
- Bei einer nicht leeren Liste ist der Nachfolger des Ankerelementes der **Listenkopf**, d.h. das erste Nutzdaten tragende Element der Liste. Der Vorgänger des Ankerelementes ist das **letzte Element** der Liste. Umgekehrt hat der Listenkopf immer das Ankerelement als Vorgänger, und das letzte Element der Liste hat immer das Ankerelement als Nachfolger.

Der Vorteil einer doppelt verketteten Ringliste besteht darin, dass Einfüge- und Löschoperationen, sowie Konkatenation ohne spezielle Fallunterscheidungen "Liste leer/nicht leer", "Löschen des letzten Elementes" etc. auskommen.

Aufgabe 1: Schicksalhafte Verkettungen (50%)

Kopieren Sie den zur Verfügung gestellten Programmrahmen auf eine Datei `MyList.java` und implementieren Sie die dort beschriebenen Methoden zur Bearbeitung doppelt verketteter Ringlisten. Beachten Sie die in den Methodenspezifikationen angegebenen Hinweise darauf, dass ggf. keine `if-else` Anweisungen erforderlich sind. Begründen Sie in Ihrer Lösung – ggf. durch Fallunterscheidungen – warum dies korrekt ist.

Aufgabe 2: Erschöpfend getestet – bis zur Erschöpfung (50%)

Entwerfen Sie eine umfassende Testsuite, wobei Sie die im Programmrahmen vorgegebene Methode `testAssert()` – ggf. mit kleinen Erweiterungen – verwenden, um eine einheitliche Formatierung der Ausgabe zu erzeugen. Die Testsuite soll sich auf die Methoden

- `rlApp()`
- `rlIns()`
- `rlFind()`
- `rlDel()`
- `rlInsSorted()`
- `rlCat()`
- `rlEquals()`

konzentrieren; die anderen Methoden müssen dabei mit verwendet werden und werden dabei "implizit" mit getestet.

Begründen Sie für jede der getesteten Methoden, warum die von Ihnen gewählten Testfälle zur Prüfung ausreichend sind.

Verwenden Sie insbesondre die Methode `rlEquals()`, um mit den Listenoperationen erzielte Ergebnisse gegen die erwarteten Resultate abzugleichen.