JAVA-ÜBUNGSAUFGABEN FELDERVERWALTUNG

Die folgenden Aufgaben haben mit der Verwaltuung von Feldern zu tun und sind nicht einfach!

1) Felder konstanter Länge verwalten

Es soll die Klasse Feldmanager erzeugt werden.
Diese Klasse Feldmanager verwaltet mit den entsprechenden Methoden ein Feld.
Wenn ein Feld erstellt wird, hat es eine bestimmte vorgegebene Länge maxLen.
Das Feld soll einen Zeiger auf das letzte Element des Feldes besitzen (zeigerLetztesElement).
Wenn das Feld erzeugt wird, hat dieser Zeiger den Wert -1, also: zeigerLetztesElement = -1.
Nach jedem An - oder Einfügen wird der Wert dieses Zeigers um 1 ehöht, bei jedem Löschen eines Elements aus dem Feld wird der Zeigerwert um 1 verringert.
Der Wert von zeigerLetztesElement darf maximal maxLen-1 sein.
Dies muß durch die einzelnen Methoden des Feldmanager garantiert werden.

1.1) Attribute der Klasse Feldmanager erstellen

Erstellen Sie die Klasse Feldmanager mit genau den folgenden Attributen
(und den entsprechenden get und set-Methoden)

```
// das Feld, das verwaltet wird
private int[] feld;
// Index (Stelle) des letzten Elements im Feld feld
private int letztesElement;
// Die beim Erzeugen des Feldes angegebene Feldlänge
private int maxLen;
```

1.2) Methode der Klasse Feldmanager erstellen

Die Klasse Feldmanager muß zusätzlich noch folgende Methoden enthalten:

`public void anfügen(int zahl)`
fügt eine Zahl an das Ende (d.h. zeigerLetztesElement = letztesElement) des Feldes an, vorausgesetzt, das Feld wird dadurch nicht größer als maxLen.

`public void druckeFeld()`
gibt alle Elemente des Feldes auf dem Bildschirm aus.

`public void entferneLetztesElement()`
entfernt das letzte Element des Feldes, wobei die Länge des Feldes angepaßt (um 1 verringert) wird.

`public void entferneAnPosition(int pos)`
Lösche das Element an der Stelle pos
Beispiel:
Feld: 4 5 9 1 3
Lösche Element an der Stelle 2
Feld: 4 5 1 3

`public void einfügenRechts(int pos, int zahl)`
Für pos muß gelten: -1<=pos <=zeigerLetztesElement
die Zahl zahl wird rechts von pos, also an der Stelle pos+1 eingefügt (nicht überschrieben)
Vorher werden die entsprechenden Zahlen noch nach rechts verschoben.
Beispiel:
Feld: 7 5 9 3
Einfügen der Zahl 8 an rechts von pos = 2 ergibt:
Feld: 7 5 9 8 3

`public void einsortierenRechts(int zahl)`
Fügt das Element zahl rechts von der Stelle ein, wo zahl das 1. Mal vom Wert her größer (oder gleich) ist als das dort sich befindliche Feldelement.
Ansonsten wird es rechts von der Stelle -1 eingefügt.
Beispiel:
Feld: 4 5 9 1 3
Rechts einsortieren der Zahl 8 ergibt:
Feld: 4 5 8 9 1 3

`public void einsortierenLinks(int zahl)`
Fügt das Element zahl links von der Stelle ein, wo zahl das 1. Mal vom Wert her größer (oder gleich) ist als das dort sich befindliche Feldelement.
Ansonsten wird es an das Feldende angefügt.
Beispiel:
Feld: 4 5 9 1 3
Links einsortieren der Zahl 2 ergibt:
Feld: 4 5 9 2 1 3

`public int entferneZahl(int zahl)`
Sucht die Zahl value in einer Liste und löscht diese.
Wenn mehrere gleiche Zahlen in der Liste vorkommen, wird die erste vorkommende Zahl gelöscht. Rückgabe: 0 (gefunden) bzw. -1 (nicht gefunden).
Beispiel:
Feld: 4 1 9 1 3
Entfernen des Zahl 1 ergibt:
Feld: 4 9 1 3

`public int sucheZahl(int zahl)`
sucht eine Zahl in der Liste und gibt ihre Position zurück.

`public void sortiereFeld(int modus)`
sortiert eine Liste auf- oder absteigend (durch Wert des Parameters modus bestimmt).

`public void löscheFeld()`
Lösche alle Elemente des Feldes

`public void ändereWert(int wertAlt, int wertNeu)`
Ersetzt das erste Vorkommen der Zahl wertAlt durch die neue Zahl wertNeu.

`public int löscheDuplikate()`
Löscht die Elemente einer Liste die mehr als 1 Mal vorkommen, so daß jedes Element genau ein Mal in der Liste vorkommt.
Beispiel:
Feld: 7 5 7 9 5 8 9
Duplikate löschen
Feld: 7 5 9 8

Bemerkungen:
B1)
In main() sollen alle obigen Methoden ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.

B2)
Weitere, selbst erfundene Methoden implementieren.

2) Felder variabler Länge verwalten

Es soll ein dynamisches Feld (dynamisches Array) mit Hilfe von "normalen" Feldern (d.h. Felder mit konstanter Länge) simuliert werden:
Jedes Mal, wenn an das Feldende eines Feldes (mit Länge len) ein Element (Integer-Zahl) angefügt werden soll, wird ein Feld der Länge len+1 erzeugt und der Inhalt des alten Feldes (an der Stelle 0 beginnend) in dieses neue Feld reinkopiert und an das Feldende des neuen Feldes das Element angefügt.
Das Feld soll einen Zeiger auf das letzte Element des Feldes besitzen (**stelleLetztesElement**). Wenn das Feld erzeugt wird, hat dieser Zeiger den Wert -1: stelleLetztesElement = -1
nach jedem An - oder Einfügen wird der Wert dieses Zeigers um 1 ehöht, bei jedem Löschen eines Elements aus dem Feld wird der Zeigerwert um 1 verringert.

2.1) Attribute der Klasse DynamischesArray erstellen

Erstellen Sie die Klasse DynamischesArray mit genau den folgenden Attributen
(und den entsprechenden get und set-Methoden)

```
private int[] feld;          // neue Feld
// Index (Stelle) des letzten Elements im Feld feld
private int stelleLetztesElement;
private int[] feldAlt;       // alte Feld
```

2.2) Methode anfügen(...) der Klasse DynamischesArray erstellen

`public void anfügen(int zahl)`
fügt eine Zahl an das Ende des Feldes an.
Dies soll dadurch realisiert werden, daß ein neues Feld erstellt wird (mit einer um 1 größeren Länge als das alte Feld). Die Zahl muß an das Ende des neuen Feldes angefügt werden.
Der Inhalt des alten Feldes muß dann noch in das neue Feld kopiert werden. Dann wird das alte Feld dem neuen Feld zugewiesen: "neueFeld = alteFeld"

2.3) Weitere Methode der Klasse DynamischesArray erstellen

Erstellen Sie - wie oben bei Feldern konstanter Länge - die dort angegebeben Methoden.
Beachten Sie, daß bei Methoden, die die Feldlänge ändern, der Wert des Attributs stelleLetztesEleemnt angepaßt werden muß.

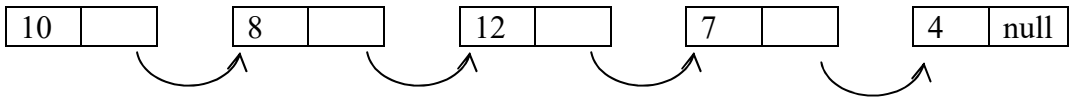
2.4)

Weitere, selbst erfundene Methoden implementieren.

Bemerkungen:
Die Java-Entwicklungsumgebung enthält schon die gegebene Klasse ArrayList
(siehe Java-Doku). Mit dieser kann die diese Aufgabe leichter gelöst werden.

3) Felder variabler Länge verwalten (als verkettete Listen)

Die gegebene Klasse ArrayList der Java-Entwicklungsumgebung (siehe Java-Doku) wird intern durch eine sogenannte verkettete Liste realisiert.
Als Beispiel soll dazu eine Liste von ganzen Zahlen gebastelt werden, die einfach verkettet ist (einfach verkettete Liste).



Im rechten Teil eines Datenelements steht im Prinzip ein Link (Adresse) auf das nächste Datenelement.
Die zugehörige Datenstruktur könnte man z.B. mit der folgenden selbstgebauten Klasse realisieren:

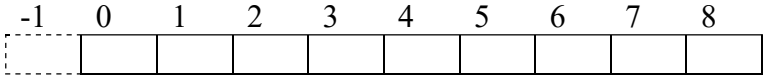
```
class DtElement{
    public int zahl;
    public DtElement next;

    public void setZahl(int zahl){
        this.zahl=zahl;
    }

    public void setNext(DtElement element){
        this.next=element;
    }
}
```

Bemerkungen:
B1)
Die Liste wäre doppelt verkettet, wenn die Pfeile jeweils in 2 Richtungen gehen.

B2)
Wenn man die verkettete Liste als Feld auffasst, in dem pos die Stelle des letzten Elements bezeichnet (wenn das Feld die Länge 0 hat ist diese Stelle = -1), dann kann man die Stelle angeben, wo der Wert eingefügt werden soll.
Nummerierung der Zellen für ein Feld der Länge 9 (das eine verkettete Liste simuliert):



Für pos muß gelten: -1<=pos <=Stelle des letzten Elements
Wenn also in einer der folgenden Methoden der Parameter pos vorkommt, dann wird so ein Feld betrachtet.

3.1) Einfach verkettete Listen
Implementieren Sie die Klasse LinkedList (deutsch: verkettete Liste) mit den Attributen:
```
public DtElement first;
public DtElement last;
```
und erstellen dort die wie folgt dokumentierten Funktionen:
(eine verkette Liste wird auch kurz nur mit Liste bezeichnet).

```
public void appendElement(int zahl)
```
fügt eine Zahl an das Ende der verketteten Liste an.

```
public void printList()
```
gibt alle Elemente der verketteten Liste auf dem Bildschirm aus.

```
public void gradeRight(int value)
```
Rechts einsortieren: Fügt den Wert rechts von der Stelle ein, wo value das 1. Mal vom Wert her größer (oder gleich) ist als das dort sich befindliche Feldelement.
Ansonsten wird es rechts von der Stelle -1 eingefügt.
Wenn nur diese Funktion verwendet wird, um Werte einzufügen, ergibt dies eine aufsteigend sortierte Liste.
Beispiel:
verkettete Liste: 4 5 9 1 3
Rechts einsortieren der Zahl 8 ergibt:
verkettete Liste: 4 5 8 9 1 3

```
public void gradeLeft(int value)
```
Links einsortieren: Fügt das Element zahl links von der Stelle ein, wo zahl das 1. Mal vom Wert her größer (oder gleich) ist als das dort sich befindliche Feldelement.
Ansonsten wird es an das Ende der Liste angefügt.
Wenn nur diese Funktion verwendet wird, um Werte einzufügen, ergibt dies eine absteigend sortierte Liste.
Beispiel:
verkettete Liste:: 4 5 9 1 3
Links einsortieren der Zahl 2 ergibt:
verkettete Liste: 4 5 9 2 1 3

```
public void insertRight(int pos, int value)
```
Für pos muß gelten: -1<=pos <= Stelle des letzten Elements
die Zahl zahl wird rechts von pos, also an der Stelle pos+1 eingefügt (nicht überschrieben)
Vorher werden die entsprechenden Zahlen noch nach rechts verschoben.
Beispiel:
verkettete Liste: 7 5 9 3
Einfügen der Zahl 8 an rechts von pos = 2 ergibt:
verkettete Liste: 7 5 9 8 3

```
public void deleteList()
```
löscht alle Elemente der verketteten Liste.

`public void deleteElement1(DtElement element, DtElement prec)`
Löscht das Listenelement "element" einer einfach verketteten Liste. Da die Liste nur einfach verkettet ist, kann man ihr noch die Adresse von "prec" mitgeben, also dem vorigen Element von "element", damit prec mit dem Nachfolger von "element" verlinkt werden kann.
Bei einer doppelt verketteten Liste wäre dies nicht nötig.
Natürlich könnte die Methode die Adresse prec selbst herausfinden (vom Listenanfang ausgehend), was aber zu einem schlechteren Laufzeitverhalten führen würde.

`public void deleteElement2(DtElement element ptec)`
Löscht nicht das Listenelement "prec" einer einfach verketteten Liste, sondern das Element rechts dieses Elements. Da die Liste nur einfach verkettet ist, kann die Funktion prec mit dem übernächsten verlinken bzw. das nächste löschen.
Bei einer doppelt verketteten Liste wäre dies nicht nötig: man müsste nur die Adresse des zu löschendem Elements angeben.

`public void deleteLastElement()`
Löscht das letzte Listenelement der Liste.

`public int deleteValue(int value)`
Sucht die Zahl value in einer Liste und löscht diese.
Wenn mehrere gleiche Zahlen in der Liste vorkommen, wird die erste vorkommende Zahl gelöscht. Rückgabe: 0 (gefunden) bzw. -1 (nicht gefunden).

`public void deleteAt(int pos)`
Für pos muß gelten: 0 <= pos <=Stelle des letzten Elements.
Löscht die Zahl an der Stelle pos.
Beispiel:
verkettete Liste: 4 5 9 1 3
Lösche Element an der Stelle 2
verkettete Liste: 4 5 1 3

`public int searchValue(int value)`
Sucht eine Zahl in einer Liste und gibt das Ergebnis zurück:
0: gefunden, -1: nicht gefunden.

`public void sortList(int modus)`
Sortiert eine Liste aufsteiegend (modus=0) oder absteiegend (modus=1).

`public void change(int valueOld, int valueNew)`
Ersetzt das erste Vorkommen der Zahl oldValue durch die neue Zahl newValue.

`public int deleteDuplicate()`
Löscht die Elemente einer Liste die mehr als 1 Mal vorkommen, so daß jedes Element genau ein Mal in der Liste vorkommt. Außerdem wird die Anzahl der Duplikate zurückgegeben.
Beispiel:
verkettete Liste: 7 5 7 9 5 8 9
Duplikate löschen
verkettete Liste: 7 5 9 8

Bemerkungen:
B1)
In main() sollen alle obigen Methoden ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.

B2)
Weitere, selbst erfundene Methoden implementieren.

B3)
Die Java-Entwicklungsumgebung enthält schon die gegebene Klasse ArrayList (siehe Java-Doku). Mit dieser kann die diese Aufgabe leichter gelöst werden.

3.2) Doppelt verkettete Listen

Implementieren Sie die oben dokumentierten Funktionen für doppelt verkettete Listen
Passen Sie die Parameter bzw. die Beschreibung - falls mötig - entsprechend an.
In main() sollen diese Funktionen ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.
Eine verkette Liste wird auch kurz nur mit Liste bezeichnet.

Ein paar Beispiele:

```
public void printList()
```

Gibt die Daten, also Zahlen (nicht Adressen) aller Elemente der doppelt verketteten Liste auf dem Bildschirm aus, wobei element nicht die Anfangsadresse der Liste, sondern die Adresse irgend eines Elements der Liste ist.
Die Elemente der Liste werden vom Listenanfang bis Listenende hintereinander ausgegeben.
Intern muß die Funktion dazu von "element" ausgehend den Listenanfang bestimmen.

```
/*************************************************************/
/**                                                         **/
/**  void deleteElement (struct dtelement *element)         **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *element : Adresse des zu löschenden Elements

Return:
  nichts

Beschreibung:
  Löscht das Listenelement "element" einer doppelt verketteten
  Liste.
*/
```

4) Warteschlange

Implementieren Sie eine Warteschlange (z.B. bei einem Drucker) als spezielle verkettete Liste, die dem Prinzip folgt: First in - First out.
Implementieren Sie dazu noch (siehe oben) die entsprechenden Funktionen (anfügen, usw.) .

5) Stack

Implementieren Sie einen Stapel (Stack), als spezielle verkettete Liste, der dem Prinzip folgt:
Last in - First out
Implementieren Sie dazu noch (siehe oben) die entsprechenden Funktionen (anfügen, usw.) .

6) Ringpuffer

Man hat einen Speicher mit N Elementen (von 0 bis N-1 insiziert). Das dem N-1 - ten Element folgende ist dann wieder das Element 0. D.h. das 0. Element wird überschrieben, danach 1. Element , usw. Implementieren Sie einen Ringpuffer.

7) Binärbaum

Ein Binärbaum hat eine Wurzel. Von jedem Knoten gehen 2 weitere Knoten ab. Damit kann man z.B. Zahlen einsortieren. Im linken Knoten werden Zahlen kleiner gleich und im rechten Knoten Zahlen größer als der Elternknoten abgespeichert.

								100								
				30								200				
		15				40				150				300		
														400		

Implementieren Sie eine Funktion, die Zahlen in einen Binärbaum einsortiert bzw. sucht.