Algorithmen und Datenstrukturen
Übungsblatt 2
Listen, Stacks, Bäume, Graphen
Prof. Dr. Tim Reichert

1. Stack basierend auf Array

1.1 Implementieren Sie einen Stack für Strings basierend auf einem Java Array und einem Zähler. Der Stack kann maximal N Elementen aufnehmen.

Beispiel für N=10:

```
MyStack stack = new MyStack(10);
stack.push("hello");
stack.push("world");
System.out.println(stack.pop());

Ausgabe: "world"
```

1.2 Erweitern Sie den Stack damit dieser variabel wächst und schrumpft. Eine Größenangabe für den Konstruktor ist nun nicht mehr nötig.
1.3 Erweitern Sie den Stack mit Hilfe von Generics damit dieser auf beliebige Datentypen spezialisiert werden kann.

Beispiel:

```
MyGenStack<Integer> stack = new MyGenStack<Integer>();
stack.push(1);
```

1.4 Algorithmus für verschachtelte arithmetische Ausdrücke.

Implementieren Sie ein Programm, welches einen arithmetischen Ausdruck als String von der Kommandozeile einließt und das Ergebnis des Ausdrucks ausgibt.

Die Ausrücke Exp haben die Form:

Op := + | - | * | /
Exp := Number
Exp := (Exp OP Exp)

Beispiele: (2 + 3), (3 – (2 * 3)), ((5 * 4) – (2 * (3 / 4)))

Der Algorithmus nach Dijkstra funktioniert mit einem Operatorstack und einem Zahlenstack wie folgt:

Öffnende Klammer => nichts tun
Zahl => Auf den Zahlenstack pushen
Operator => Auf den Operatorstack pushen

Schließende Klammer => Operator vom Operatorenstack poppen, Zahlen vom Zahlenstack poppen, Operator auf Zahlen anwenden, Ergebnis auf Zahlenstack pushen

Um das Ende des Strings zu erkennen, können Sie z.B. ein Semikolon verwenden.

2. Listen

Gegeben ist folgende Klasse:

```
public class Node<Item> {
      Item item;
      Node<Item> next;
}
```

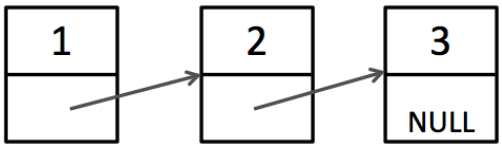
Mit Objekten dieser Klasse lassen sich Listenstrukturen (Linked List, verkettete Liste) bilden.

Beispiel:

```
Node<Integer> n1 = new Node<Integer>();
Node<Integer> n2 = new Node<Integer>();
Node<Integer> n3 = new Node<Integer>();
n1.item = 1;
n2.item = 2;
n3.item = 3;
n1.next = n2;
n2.next = n3;
n3.next = null;

System.out.println(n1.item);
System.out.println(n1.next.item);
System.out.println(n1.next.next.item);
```

```
Ergebnis:
1
2
3
```



2.1 Erzeugen Sie äquivalent zum Beispiel eine Liste mit 3 Nodes, so dass die Liste (1, 3, 4) entsteht. Schreiben Sie Code, der die existierende Struktur so verändert, dass durch Einfügen die Liste (1, 2, 3, 4) entseht. Entfernen Sie dann den Node mit der Zahl 3, so dass (1, 2, 4) entsteht.

2.2 Implementieren Sie eine Klasse Node mit doppelter Verkettung und lösen Sie damit die Aufgabenstellung aus 2.1 erneut.

2.3 Implementieren Sie auf Node (Sie dürfen zwischen einfach- und doppelt-verkettet wählen) eine verkettete Liste, die von der Node-Implementierung abstrahiert. Das heißt, die Liste verwendet intern Node-Objekte. Der Nutzer der Klasse bekommt diese aber nicht zu Gesicht. Die Implementierung bietet folgende Methoden als Interface:

isEmpty()
length()
elementAt(int index) //gibt das Element an position index zurück
prepend(Item e) //vorne einfügen
append(Item e) //am Ende einfügen
insertAt(int index, Item e)
remove(int index)

Beispiel:

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.append(1);
list.append(3);
list.insertAt(1,2);
```

Ergebnis: Listenstruktur mit (1, 2, 3) wie im Bild oben.

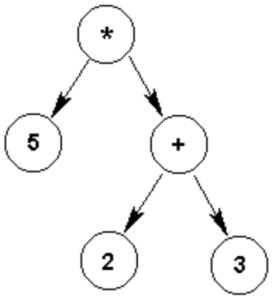
2.4 Implementieren Sie eine ArrayList mit demselben Interface. Das heißt, die Klassen ArrayList und LinkedList haben public-Methoden, die gleich heißen und dieselben Parameter und Rückgabewerte haben. Anders ist nur die Implementierung. Ein Array ersetzt die Node-Objekte. Ähnlich wie beim Stack wächst und schrumpft das Array.

3. Binärbäume

Hierarchische Strukturen können im Rechner durch Bäume abgebildet werden. Ähnlich wie bei verketteten Listen kann dies über einfache Knotenklasse erfolgen.

Gegeben ist folgende Klasse. Sie ist Grundlage für Bäume, bei denen jeder Knoten maximal zwei Nachfahren hat. Durch die parent-Referenz sind die entstehenden Strukturen "doppelt verkettet":

```
public class BinaryNode {
  String item;
  BinaryNode parent;
  BinaryNode left;
  BinaryNode right;
}
```



3.1 Bilden Sie die arithmetischen Ausdrücke mit dieser Struktur ab. Jede BinaryNode enthält entweder einen Operator oder eine Zahl. Die Zahlen sind die Blätter des Baumes:

a) 2 + 3,
b) 5 * (2 + 3) siehe Abbildung

b) 2 * 3 + 4 * 5
c) 2 * (3 + 4) * 5

3.2 Implementiren Sie eine Methode welche prüft, ob ein bestimmter String im Baum enthalten ist.

Beispiel:

// Baum für Ausdruck 2 + 3 * 4

```
find (tree, "4")

Ergebnis: true

find (tree, "5")

Ergebnis: false
```

3.3 Implementieren Sie eine Methode path, die den Weg von der Wurzel des Baums zu einem Element ausgibt. Ergebnis ist eine Liste. Ist das Element nicht enthalten, ist die Liste leer. Ist das Element in der Wurzel des Baums, dann enthält die Liste nur das Element selbst.

Beispiel:

// Baum aus Abbildung oben

```
path(tree, 3)

Ergebnis:

Liste ("*", "+", "3")
```

4. Bäume

Gegeben ist folgende Klasse:

```
public class TreeNode<Item> {
    TreeNode<Item> parent;
    Item item;
    ArrayList<TreeNode<Item>> children;
}
```

4.1 Bilden Sie mit dieser Klasse einen Teil der Studiengangsstruktur der Hochschule ab:

Hochschule
Informatik
Bachelor
AI
Mobile
Psychologie
SE
Business
Embedded

Games
....
Technik
...

4.2 Implementieren Sie eine Methode entsprechend der aus 3.3 und geben Sie mit dieser den Pfad zum Schwerpunkt Mobile oder alternativ Psychologie aus.
4.3 Implementieren Sie eine Methode printTree, welche alle Knoten des Baums so ausgibt, dass eine Textdarstellung wie in 4.1 entsteht.
4.4 Impelementieren Sie eine Methode printTreeLevels, welche jede Hierarchieebene für sich ausgibt.

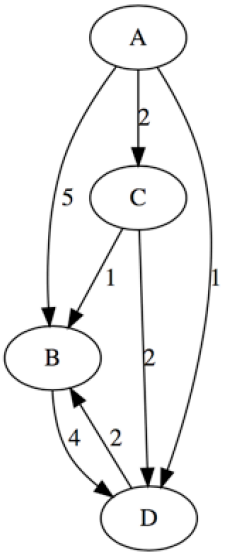
Beispiel:

```
// Baum aus 4.1
printTreeLevels(tree)

Ergebnis:

1 Hochschule
2 Informatik, Technik, Wirtschaft
3 Bachelor, Master
4 AI, SE, MI, ..
5 Mobile, Psychologie, Business, ….
```

5. Graphen



5.1 Implementieren Sie eine Graphdarstellung für gerichtete, gewichtete Graphen und bilden Sie den gezeigten Graph damit ab.
5.2 Implementieren Sie den Algorithmus von Dijkstra, so dass für alle Knoten der kürzeste Pfad zum Startknoten gefunden wird.