Gruppenänderungen: bei Yann Disser
(oder Madeleine Theile) melden
UND im Unix-Pool

Listen: Realisierung in Java z.B. mittels
ListNode SimpleList

Stacks: spezielle Listen (Last-In, First-Out oder LIFO-Listen genannt)

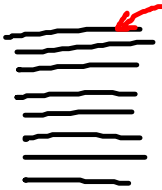
einfügen, löschen nur am Kopf möglich

Datenzugriff (immer nur auf das Element am Kopf

push() pop() top()

heißt „top" in der Stack-Sprache

Vorstellung: top



Stapel von Büchern in einem engen Karton

Stacks sind fundamental für viele Anwendungen.
Insbesondere in der Informatik

- Laufzeitstack in Java zur Verwaltung von Methodenaufrufen (später in VL)
- Erkennung korrekter Klammerausdrücke + Korrespondierender Klammerpaare

(( ) ( )) ( ) korrekt

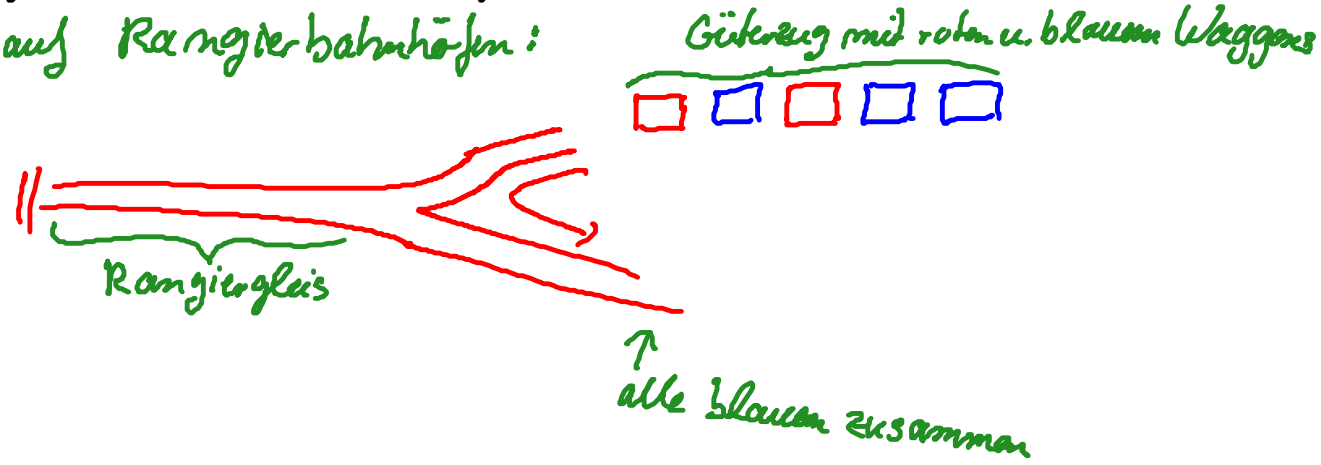
(( ) ( )) nicht korrekt

- Realisierung von Rekursionen
- Auswertung arithmetischer Ausdrücke

⋮

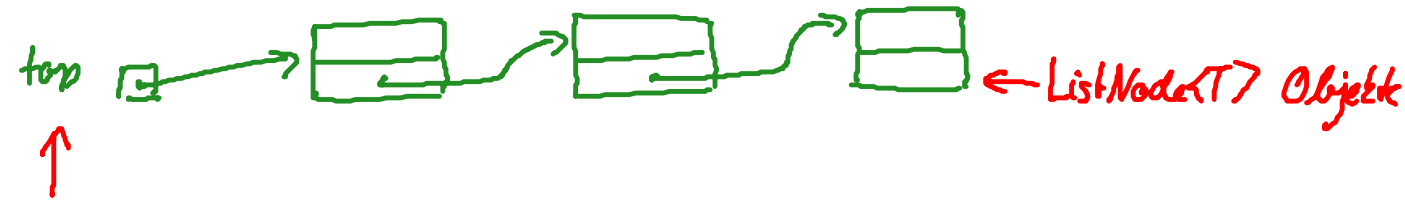
Anwendungen außerhalb der Informatik:

z.B. auf Rangierbahnhöfen:



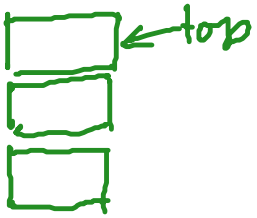
Stacks mit Hilfe von ListNode<T> implementieren:

Gedankenbild:



privates Datum eines Stacks = Referenz auf ListNode<T> Objekt

Bildliche Vorstellung (Stapel von Containern)



```
import java.util.NoSuchElementException;

/**
 * The <code>SimpleStack</code> class implements a Stack
 * of objects of Type T.
 *
 * @see ListNode
 */
public class SimpleStack<T> {
```

javadoc Kommentare

allgemeiner (generische) Typ

Name der Klasse mit allg. Typ Parameter

```
/**
 * a list pointer to the first node
 */
private ListNode<T>  top;

/**
 * Constructs an empty stack.
 *
 */
public SimpleStack() {
        top = null;
}
```

privates Objektfeld der Klasse SimpleStack

Default Konstruktor erzeugt: top null

"leeres Stack"

```
/**
 * Tests if this stack has no entries.
 *
 * @return <code>true</code> if the stack is empty;
 * <code>false</code> otherwise
 */
public boolean isEmpty() {
        return (top == null);
}
```

Stack leer ⟺ top zeigt auf kein Listenelement

↑ wird durch Implementation der Methoden sichergestellt.

```java
/**
 * Return the value of the top node.
 */
public T top() throws NoSuchElementException {
        if (top == null) {
                throw new NoSuchElementException("No top node.");
        }
        return top.getData();
}
```

vom Typ T

Normalfall

→ Methode der Klasse ListNode<T>


```java
/**
 * Inserts a new stack node at the top with <code>someData</code>
 * as data
 *
 * @param someData the data object of the new node
 */
public void push(T someData) {
        ListNode<T> newNode = new ListNode<T>(someData);
        if (isEmpty()) {
                top = newNode;
        } else {
                newNode.setNext(top);
                top = newNode;
        }
}
```

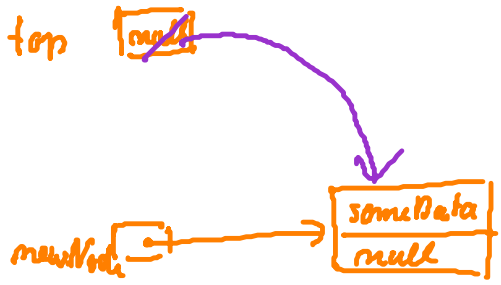
vom Typ T

1. Fall:

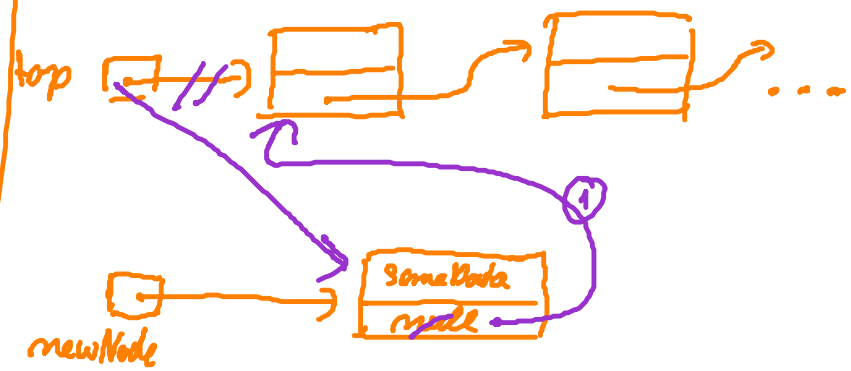
2. Fall:

① ②

1. Fall: nach push()



2. Fall: nach push()



```java
/**
 * Delete the top node from the stack.
 */
public void pop() throws NoSuchElementException {
        if (top == null) {
                throw new NoSuchElementException("No element "
                        + "for deletion.");
        }
        top = top.getNext();
}
```

Ausnahme, falls Stack leer

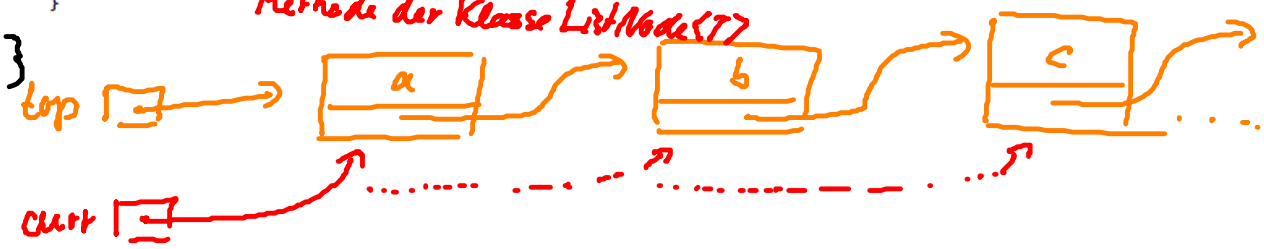
Normalfall

nach pop():

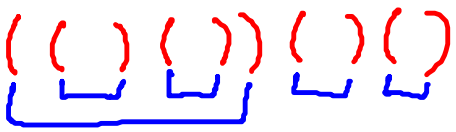

```java
/**
 * Returns a string representation of this stack, top element first
 *
 * @return a String representing this stack.
 */
public String toString( ) {
    String str = "";

    // use methods from class ListNode to traverse this stack
    for (ListNode<T> curr = top; curr != null; curr = curr.getNext())
        str = str + currentData().toString();

    return str;
}
}
```

Methode aus der Klasse T (wichtig, dass T Referenztyp!)

Methode der Klasse ListNode<T>



Verwendung der Klasse SimpleStack zur Erkennung von korrekten Klammerausdrücken und Ermittlung zugehöriger Klammerpaare.



Erinnerung: Ein (nichtleerer) regulärer Klammerausdruck wird nach folgenden Regeln („Grammatik") gebildet:

R1: ( ) korrekt

R2: A, B korrekt $\Rightarrow$ AB (die Konkatenation von A und B) ist korrekt

R3: A korrekt $\Rightarrow$ (A) korrekt

Beispiel:

(())() ist korrekt, denn er entsteht aus R2 angewendet auf A = (()) und B = ( ), wobei B korrekt nach R1 ist, und A aus R3 angewendet auf ( ) entsteht, und daher selbst korrekt ist.

Algorithmus muss also erkennen, ob ein String aus (,) und weiteren Zeichen bzgl. der Klammern nach diesen Regeln gebildet ist und muss die zugehörigen Paare finden

Dazu Stack verwenden.

Idee: Starte mit leerem Stack

Lese Klammerausdruck von links nach rechts

bei "(", so pushe diese auf den Stack

bei ")", so poppe das top-Element vom Stack

(dies ist eine "(") und erkläre sie zum Partner der gelesenen ")"

erzeuge Fehler, falls Stack leer ist

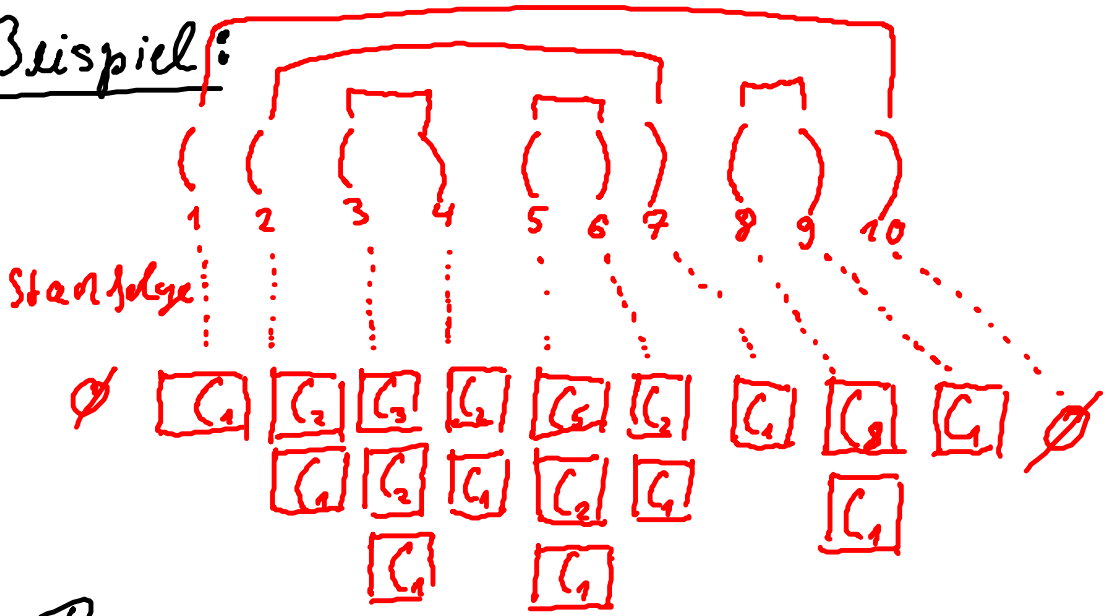
Erkläre Klammerausdruck als korrekt, falls

Stack-bedingung:

a) der Stack am Ende leer ist, und

b) nie versucht wurde, vom leeren Stack zu entfernen.

Beispiel:



↑ Stackbdg a) und b) sind erfüllt, Algo erklärt Klammerausdruck als korrekt.

Satz: (1) Klammerausdruck ist korrekt (im Sinne der Grammatik) $\Leftrightarrow$ Stackbdg. sind erfüllt.

(2) Ist dies der Fall, so erkennt der Algo zugeh. Paare richtig.

Beweis (Skript, evtl. später)

Programm "GenericStackDemo" → www

verwendet int[] partner für Positionen

$$partner[i] := \begin{cases} k & \text{falls charAt(i) und charAt(k)} \\ & \text{ein Paar bilden} \\ -1 & charAt(i) \neq (,) \end{cases}$$

Shell

Schreiben Sie einen String und beenden Sie ihn mit Return

((a+b)-(c-d))/(x-y) ← Eingabe

Der String ist korrekt mit folgender Klammerung

((a+b)-(c-d))/(x-y)

( )

( )

( )

( )

→ Frohe Weihnachten