Informatik II - Übung 5

Pascal Schärli

PascalCase@pascscha.ch

23.10.2020

ETH zürich

Nachbesprechung

Ein wachsender Stack

```
1 public Stack(int capacity) {
2     size = 0;
3     buffer = new int[capacity];
4 }
```

Ein wachsender Stack

```
1  public String toString() {
2      StringBuffer buf = new StringBuffer();
3      buf.append("[");
4      for (int i = 0; i < size; i++) {
5          if (i != 0) buf.append(", ");
6          buf.append(Integer.toString(buffer[i]));
7      }
8      buf.append("]");
9      return buf.toString();
10 }
```

Ein wachsender Stack

```
1 private void grow() {
2     int[] new_buffer = new int[2 * buffer.length];
3     for (int i = 0; i < size; i++) {
4         new_buffer[i] = buffer[i];
5     }
6     buffer = new_buffer;
7 }
```

Ein wachsender Stack

```
1 public void push(int number) {
2    if (size == buffer.length) {
3         grow();
4     }
5     buffer[size] = number;
6     size += 1;
7 }
```

Ein wachsender Stack

```
1 public int pop() throws EmptyStackException {
2     if (size == 0) throw new EmptyStackException();
3     size -= 1;
4     return buffer[size];
5 }
```

Ein wachsender Stack

```
1 public int peek() throws EmptyStackException {
2     if (size == 0) throw new EmptyStackException();
3     return buffer[size - 1];
4 }
```

Ein wachsender Stack

```
1  public boolean empty() {
2      return size == 0;
3  }
4
5  public int size() {
6      return size;
7  }
8
9  public int capacity() {
10     return buffer.length;
11 }
```

Ackermann-Funktion

```
A(2,1)
  A(2,0)
    A(1,1)
      A(1,0)
        A(0,1)
        <- 2
      <- 2
      A(0,2)
      <- 3
    <- 3
  <- 3
  A(1,3)
    A(1,2)
      A(1,1)
        A(1,0)
          A(0,1)
          <- 2
        <- 2
        A(0,2)
        <- 3
      <- 3
      A(0,3)
      <- 4
    <- 4
    A(0,4)
    <- 5
  <- 5
<- 5
```

Ackermann-Funktion als Stack

```
1  public int A(int n, int m) {
2      Stack stack = new Stack(10);
3      stack.push(n);
4      stack.push(m);
5
6      while (stack.size() != 1) {
7          System.out.println(stack);
8          final int cm = stack.pop();
9          final int cn = stack.pop();
10         if (cn == 0) {
11             stack.push(cm + 1);
12         }
13         else if (cm == 0) {
14             stack.push(cn - 1);
15             stack.push(1);
16         }
17         else {
18            stack.push(cn - 1);
19             stack.push(cn);
20             stack.push(cm - 1);
21         }
22     }
23     System.out.println(stack);
24     return stack.pop();
25 }
```

$$A(0,m) = m+1$$

$$A(n,0) = A(n-1,1)$$

$$A(n,m) = A(n-1,A(n,m-1))$$

Ackermann-Bytecode

```
 0: iload_1
 1: ifne 8
 4: iload_2
 5: iconst_1
 6: iadd
 7: ireturn
 8: iload_2
 9: ifne 21
12: aload_0
13: iload_1
14: iconst_1
15: isub
16: iconst_1
17: invokevirtual #16; //Method A:(II)I
20: ireturn
21: aload_0
22: iload_1
23: iconst_1
24: isub
25: aload_0
26: iload_1
27: iload_2
28: iconst_1
29: isub
30: invokevirtual #16; //Method A:(II)I
33: invokevirtual #16; //Method A:(II)I
36: ireturn
```

```java
public int A(int n, int m) {
    if (n == 0){
        return m + 1;
    }
    if (m == 0){
        return A(n - 1, 1);
    }
    return A(n - 1, A(n, m - 1));
}
```

Vorlesung

Swap

```
1  public static void swap(StringBuffer sbf1, StringBuffer sbf2)   {
2          StringBuffer temp = sbf1;
3          sbf1 = sbf2;
4          sbf2 =temp;
5  }
```

```
1  StringBuffer sbf1 = new StringBuffer("Hello");
2  StringBuffer sbf2 = new StringBuffer("World");
3
4  System.out.println(sbf1+" "+sbf2);
5  swap(sbf1,sbf2);
6  System.out.println(sbf1+" "+sbf2);
```

Output:

```
1  Hello World
2  Hello World
```

Call by Value / Reference

- Call by value:
  - an Funktion übergebenen Daten werden kopiert
  - keine Verbindung mehr zwischen den Daten beim Aufrufer und den Daten in der Funktion

- Call by reference:
  - Anstatt Daten zu kopieren werden Referenzen (Pointer) auf die Daten übergeben
  - Methodenaufrufe an einem so übergebenen Objekt arbeiten also auf demselben Objekt, das auch außerhalb sichtbar ist

Call by Value / Reference

- In C++ war beides möglich:
  - Call by value: Daten werden kopiert und übergeben
  - Call by reference: Referenz auf die Daten wird übergeben

- Java ist **IMMER** call by value!!
  - Bei der Übergabe einer Referenz auf ein Objekt wird die Adresse in eine lokale Variable kopiert!
  - Bei Übergabe eines primitiven Typs (char, int, float) wird der Wert in eine lokale Variable kopiert!

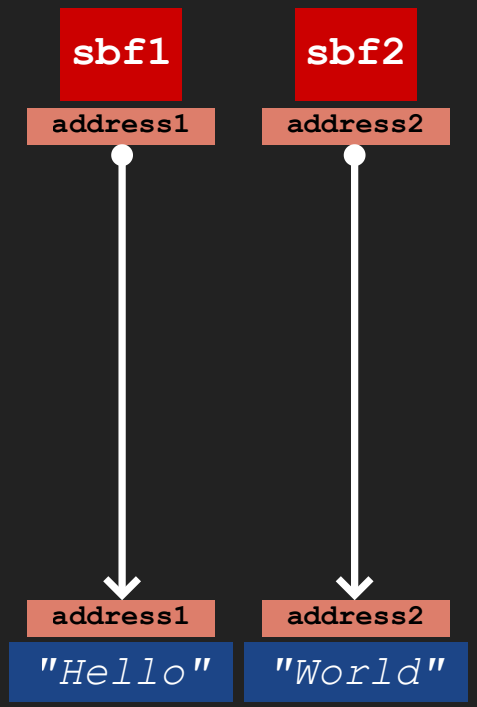
What went wrong?

```
1  public static void swap(StringBuffer s1, StringBuffer s2){
2          StringBuffer temp = s1;
3          s1 = s2;
4          s2 =temp;
5  }
```

```
1  StringBuffer sbf1 = new StringBuffer("Hello");
2  StringBuffer sbf2 = new StringBuffer("World");
3
4  System.out.println(sbf1+" "+sbf2);
5  swap(sbf1,sbf2);
6  System.out.println(sbf1+" "+sbf2);
```

Output:

```
1  Hello World
```



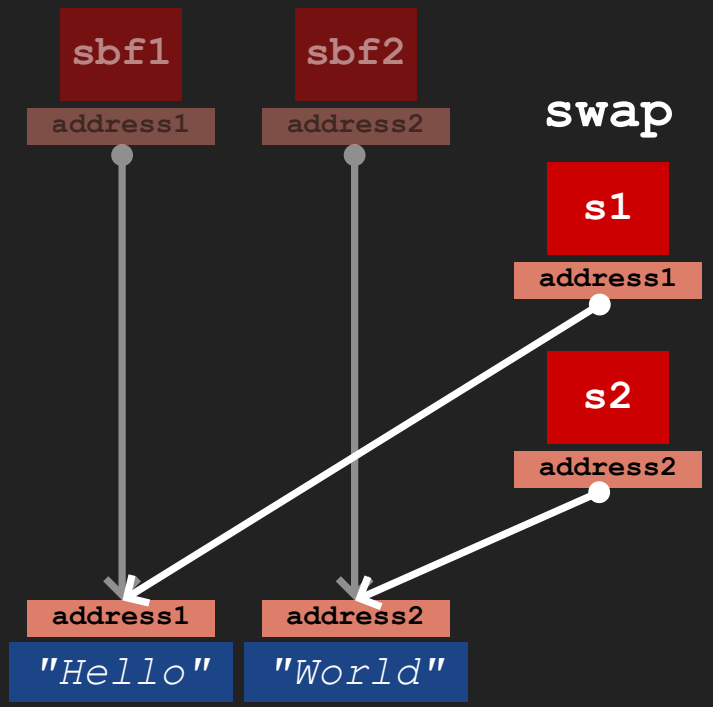
What went wrong?

```
1  public static void swap(StringBuffer s1, StringBuffer s2){
2          StringBuffer temp = s1;
3          s1 = s2;
4          s2 =temp;
5  }
```

```
1  StringBuffer sbf1 = new StringBuffer("Hello");
2  StringBuffer sbf2 = new StringBuffer("World");
3
4  System.out.println(sbf1+" "+sbf2);
5  swap(sbf1,sbf2);
6  System.out.println(sbf1+" "+sbf2);
```

Output:

```
1  Hello World
```



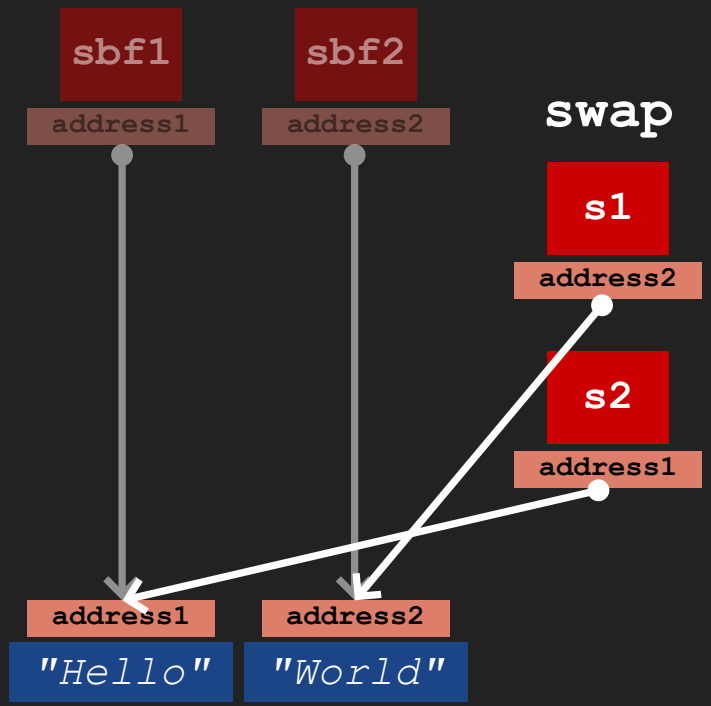
What went wrong?

```
1  public static void swap(StringBuffer s1, StringBuffer s2){
2          StringBuffer temp = s1;
3          s1 = s2;
4          s2 =temp;
5  }
```

```
1  StringBuffer sbf1 = new StringBuffer("Hello");
2  StringBuffer sbf2 = new StringBuffer("World");
3
4  System.out.println(sbf1+" "+sbf2);
5  swap(sbf1,sbf2);
6  System.out.println(sbf1+" "+sbf2);
```

Output:

```
1  Hello World
```



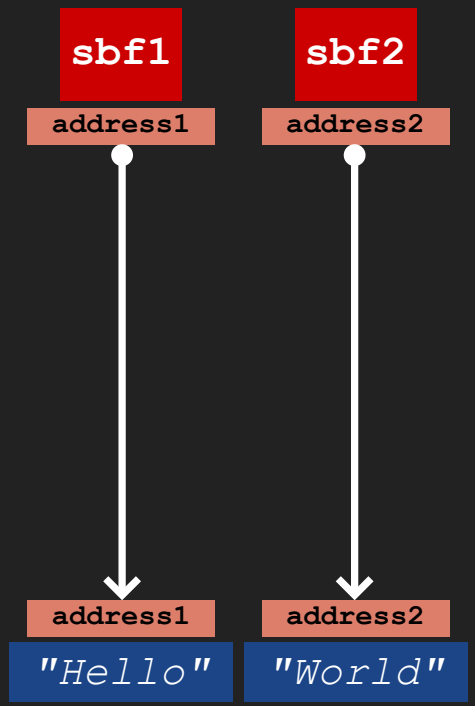
What went wrong?

```
1  public static void swap(StringBuffer s1, StringBuffer s2){
2          StringBuffer temp = s1;
3          s1 = s2;
4          s2 =temp;
5  }
```

```
1  StringBuffer sbf1 = new StringBuffer("Hello");
2  StringBuffer sbf2 = new StringBuffer("World");
3
4  System.out.println(sbf1+" "+sbf2);
5  swap(sbf1,sbf2);
6  System.out.println(sbf1+" "+sbf2);
```

Output:

```
1 Hello World
2 Hello World
```



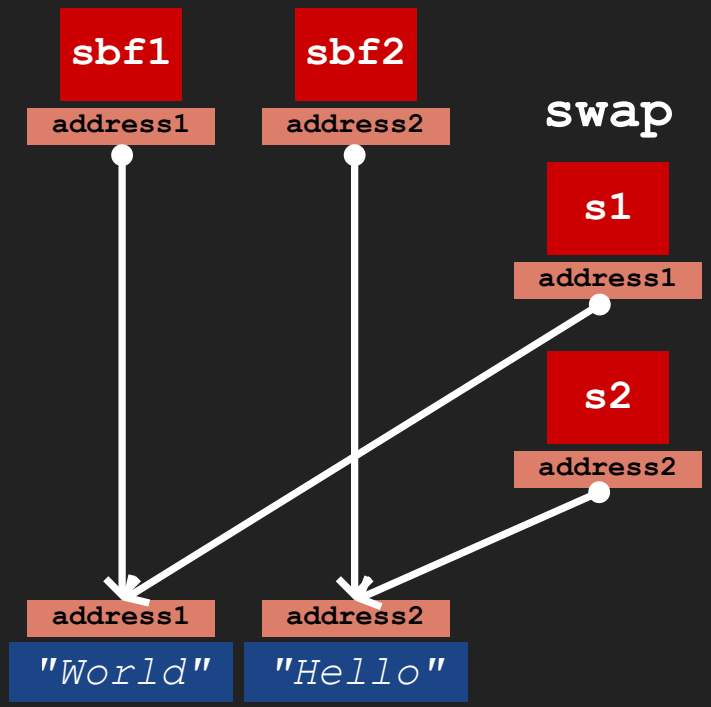
Fix

```java
public static void swap(StringBuffer s1, StringBuffer s2) {
    StringBuffer temp = new StringBuffer(s1);

    s1.delete(0, s1.length());
    s1.append(s2);

    s2.delete(0, s2.length());
    s2.append(temp);
}
```

```java
StringBuffer sbf1 = new StringBuffer("Hello");
StringBuffer sbf2 = new StringBuffer("World");

System.out.println(sbf1+" "+sbf2);
swap(sbf1,sbf2);
System.out.println(sbf1+" "+sbf2);
```

Output:

```
Hello World
World Hello
```



→ Man kann das zu Grunde liegende Objekt verändern!

Pakete - Beispiel

Node.java

```java
package node;

public class Node {
  public int value;
  public Node next;
  public Node(int value, Node next) {
    this.value = value;
    this.next = next;
  }
}
```

{1, 4, 21, 8}
→ **"1, 4, 21, 8, null"**

Lists.java

```java
package u5a1;
import node.Node;

public class Lists {
    public static String toString(Node node) {
        if (node == null) return "null";

        StringBuffer buf = new StringBuffer();
        buf.append(node.value).append(", ").append(toString(node.next));
        return buf.toString();
    }
}
```

How do you call that?

```java
public class Impedance {
    private final String name;
    protected double real, imag;

    public Impedance(String name, double real,
            double imag) {
        this.name = name;
        this.real = real;
        this.imag = imag;
    }

    public static Impedance add(String name,
            Impedance z1, Impedance z2) {
        double real = z1.getReal() + z2.getReal();
        double imag = z1.getImag() + z2.getImag();
        return new Impedance(name, real, imag);
    }

    public String toString() {
        return getName() + " = " + getReal() +
            " + i" + getImag();
    }

    public String getName() {
        return this.name;
    }

    public double getImag() {
        return this.imag;
    }

    public double getReal() {
        return this.real;
    }
}
```

Attribute

Konstruktor

Klassenmethoden

Instanzmethoden (nicht static)

Java Keywords

```java
public class Impedance {
    private final String name;
    protected double real, imag;

    public Impedance(String name, double real,
            double imag) {
        this.name = name;
        this.real = real;
        this.imag = imag;
    }

    public static Impedance add(String name,
            Impedance z1, Impedance z2) {
        double real = z1.getReal() + z2.getReal();
        double imag = z1.getImag() + z2.getImag();
        return new Impedance(name, real, imag);
    }

    public String toString() {
        return getName() + " = " + getReal() +
            " + i" + getImag();
    }

    public String getName() {
        return this.name;
    }

    public double getImag() {
        return this.imag;
    }

    public double getReal() {
        return this.real;
    }
}
```

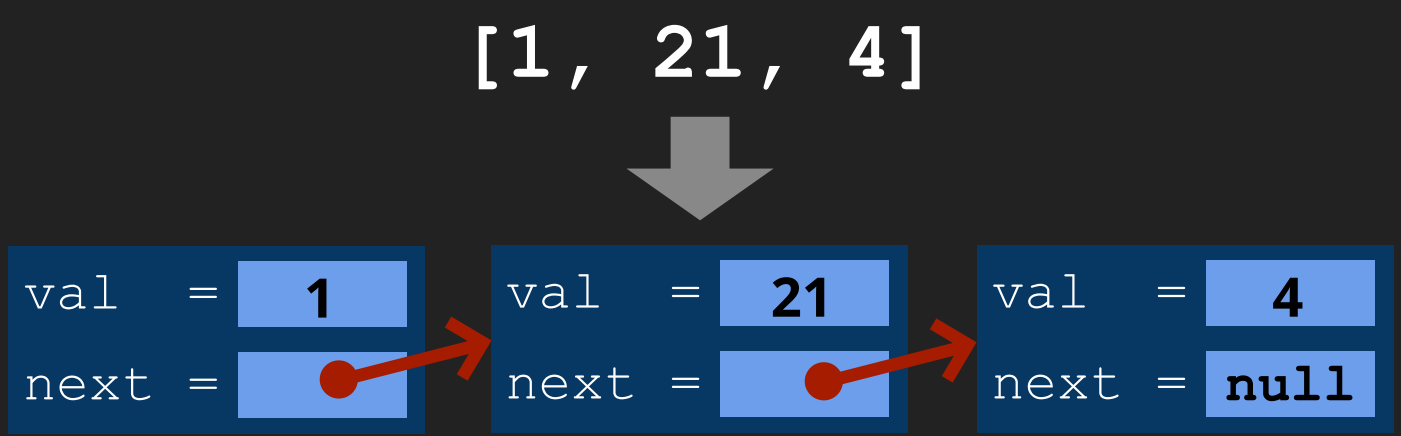
- **final**: Darf nicht verändert werden
- **public**: Für alle Sichtbar
- **private**: In der Klasse Sichtbar
- **protected**: in abgeleiteten Klassenund im package Sichtbar
- **static**: unabhängig von der Klasseninstanz, existiert nur einmal für alle Objekte dieser Klasse

Naming Conventions

- Pakete:
  - Alles Kleinbuchstaben (lowercase)
  - Beispiel: *mypackage, u4a2*
- Klassen:
  - Erster Buchstabe Gross (PascalCase)
  - Beispiel: RandomArray, StringBuffer
- Funktionen & Variabeln
  - Erster Buchstabe klein (camelCase)
  - Beispiel: *toString(), myFunction(), myVar*
- Konstanten (Static):
  - Alles Gross, Wörter mit _ getrennt (UPPER_SNAKE_CASE)
  - Beispiel: MY_CONSTANT, SPEED_OF_LIGHT

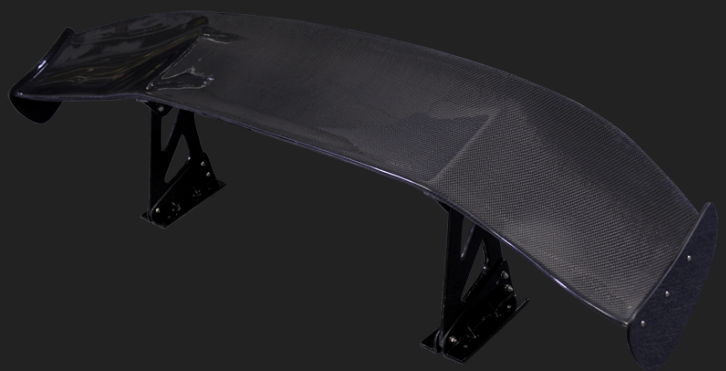
Linked List

```java
package list;

public class List {
        public int val;
        public List next;

        public List(int val, List next)
        {
                this.val = val;
                this.next = next;
        }
}
```




K!

Vorbesprechung



Einfach verkettete Listen

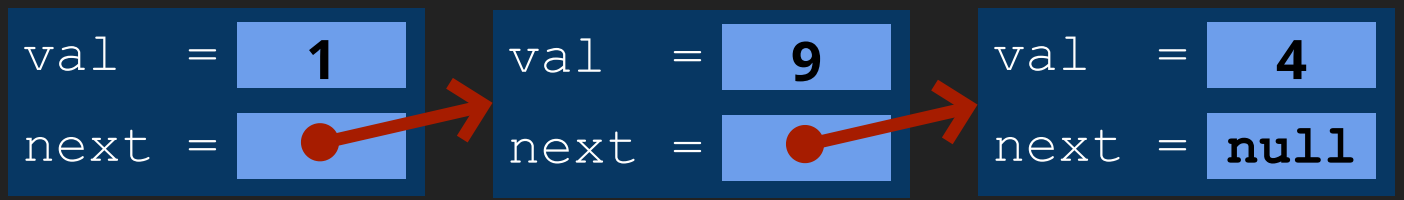
- Implementiert eine Linked-List
- **Achtung**: Spezielle Namensgebung
  - List $\rightarrow$ Eine Node/Knoten aus der Liste
  - Lists $\rightarrow$ Helperfunktionen um die Liste zu verändern
- Ihr müsst die Funktionen rekursiv implementieren!

```java
package list;

public class List {
    public int val;
    public List next;

    public List(int val, List next) {
        this.val = val;
        this.next = next;
    }
}
```

```java
package u5a1;

import java.util.NoSuchElementException;
import list.List;


public class Lists {
        // TODO
}
```

Einfach verkettete Listen

```java
public static String toString(List list){
    if (list == null) return "null";

    StringBuffer buf = new StringBuffer();
    buf.append(list.value).append(", ").append(toString(list.next));
    return buf.toString();
}
```




```
"1, " + toString(list.next) = "1, 9, 4, null"
"9, " + toString(list.next) = "9, 4, null"
"4, " + toString(list.next) = "4, null"
"null"                      = "null"
```

Einfach verkettete Listen

add $\rightarrow$ Neuer Wert am Anfang anhängen

list1 $\widehat{=}$ [1, 21]

```
add(list1, 4);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 1, 21]

size $\rightarrow$ Bestimmt die Grösse einer Liste

list1 $\widehat{=}$ [4, 1, 21]

```
size(list1);
```

$\Rightarrow$ 3

Einfach verkettete Listen

sum $\rightarrow$ Bestimmt Summe aller Elemente

list1 $\widehat{=}$ [4, 1, 21]

```
sum(list1);
```

$\Rightarrow$ 26

last $\rightarrow$ Bestimmt den letzten Knoten einer Liste

list1 $\widehat{=}$ [4, 1, 21]

```
list2 = last(list1);
```

$\Rightarrow$ list2 $\widehat{=}$ [21]

Einfach verkettete Listen

sublist $\rightarrow$ Schneidet den Anfang einer Liste ab

list1 $\widehat{=}$ [4, 1, 21]

```
list2 = sublist(list1,1);
```

$\Rightarrow$ list2 $\widehat{=}$ [1, 21]

valueAt $\rightarrow$ Bestimmt den Wert von einem bestimmten Element

list1 $\widehat{=}$ [4, 1, 21]

```
valueAt(list1,2);
```

$\Rightarrow$ 21

Einfach verkettete Listen

index $\rightarrow$ bestimmt den ersten Index bei dem ein bestimmter Wert in der Liste gespeichert ist

list1 $\widehat{=}$ [4, 1, 21]

index(list1,21);

$\Rightarrow$ 2

Modifizierung von Listen

- Klasse MutableLists erweitert die Klasse Lists mit Methoden zum Bearbeiten von Listen
- Rekursive Implementation
- Benützt eure Methoden in Lists aus der vorherigen Aufgabe

Modifizierung von Listen

append $\rightarrow$ Neuer Wert am Ende anhängen

list1 $\widehat{=}$ [4, 1]

```
append(list1,21);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 1, 21]

concat $\rightarrow$ Verbindet zwei Listen

list1 $\widehat{=}$ [4, 1], list2 $\widehat{=}$ [21, 8]

```
concat(list1,list2);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 1, 21, 8]

Modifizierung von Listen

insertAt $\rightarrow$ Wert an bestimmter Stelle einfügen

list1 $\widehat{=}$ [4, 1]

```
insertAt(list1,1,3);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 3, 1]

insertAt $\rightarrow$ Liste an bestimmter Stelle einfügen

list1 $\widehat{=}$ [4, 1], list2 $\widehat{=}$ [9, 10]

```
insertAt(list1,1,list2);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 9, 10, 1]

Modifizierung von Listen

remove $\rightarrow$ Wert an bestimmter Stelle entfernen

list1 $\widehat{=}$ [4, 1, 21, 8]

```
remove(list1,2);
```

$\Rightarrow$ list1 $\widehat{=}$ [4, 1, 8]

Sortieren von Listen

- Klasse SortedLists erweitert die Klasse Lists mit Methoden zum Bearbeiten von Listen

- Rekursive Implementation
- Zuerst funkiton insertSorted() machen, welche einen Wert in einen bereits sortierten Array einfügt.
- list.next sortieren, dann list.value mit insertSorted in den sortierten Rest einfügen

Sortieren von Listen

insertSorted $\rightarrow$ Fügt einer Sortierten Liste einen weiteren Wert hinzu

list1 $\widehat{=}$ [1, 4, 21]

```
insertSorted(list1,8);
```

$\Rightarrow$ list1 $\widehat{=}$ [1, 4, 8, 21]

Noch ein wachsender Stack

- Klasse Stack hat eine Private Variable vom Typ List, welches die Daten im Stack speichert.
- Implementiert damit einen Stack, Ihr könnt eure List***s*** Hilfsfunktionen verwenden.
- Falls ihr etwas neu Schreibt $\rightarrow$ Rekursive Implementation

Noch ein wachsender Stack

push $\rightarrow$ Wert oben auf den Stack legen

list $\widehat{=}$ [1, 21]

```
push(4);
```

$\Rightarrow$ list $\widehat{=}$ [4, 1, 21]

pop $\rightarrow$ Element oben vom Stack wegnehmen

list $\widehat{=}$ [4, 1, 21]

```
pop();
```

$\Rightarrow$ list $\widehat{=}$ [1, 21]

$\Rightarrow$ return 4

Noch ein wachsender Stack

peek $\rightarrow$ Oberster Wert vom Stack lesen

list $\widehat{=}$ [1, 21]

```
peek();
```

$\Rightarrow$ return 1

empty $\rightarrow$ True falls Liste leer ist

size $\rightarrow$ Grösse der Liste

Viel Spass!

VIDEO ORIENTATION	PROS	CONS
HORIZONTAL 	• LOOKS NORMAL TO OLD PEOPLE • FORMAT USED BY A CENTURY OF CINEMA	• HUMANS ARE TALLER THAN THEY ARE WIDE • I'M NOT TURNING MY PHONE SIDEWAYS
VERTICAL 	• HOW MOST NORMAL PEOPLE SHOOT AND WATCH VIDEO NOW SO WE MAY AS WELL ACCEPT IT	• HUMAN WORLD IS MOSTLY A HORIZONTAL PLANE
DIAGONAL 	• BOLD AND DYNAMIC • EQUALLY ANNOYING TO ALL VIEWERS • GOOD COMPROMISE	• NONE