

# Listen

## Algorithmen und Datenstrukturen 2

- Grüne Farbe: Bitte im Script nachtragen

**Lösung: public boolean remove(Object o) (Schleppzeiger) : Komplexität:  $O(n)$**

```
Node<E> current = first, before = null;
```

```
// Remove first element
```

```
while (current != null &&  
!Objects.equals(o, current.elem)) {
```

```
if (before == null) {
```

```
before = current;
```

```
first = current.next;
```

```
current = current.next;
```

```
// Remove element in between
```

```
} // End of list – not found
```

```

} else {

```

```
if (current == null) {
```

```
before.next = current.next;
```

```
return false;
```

}

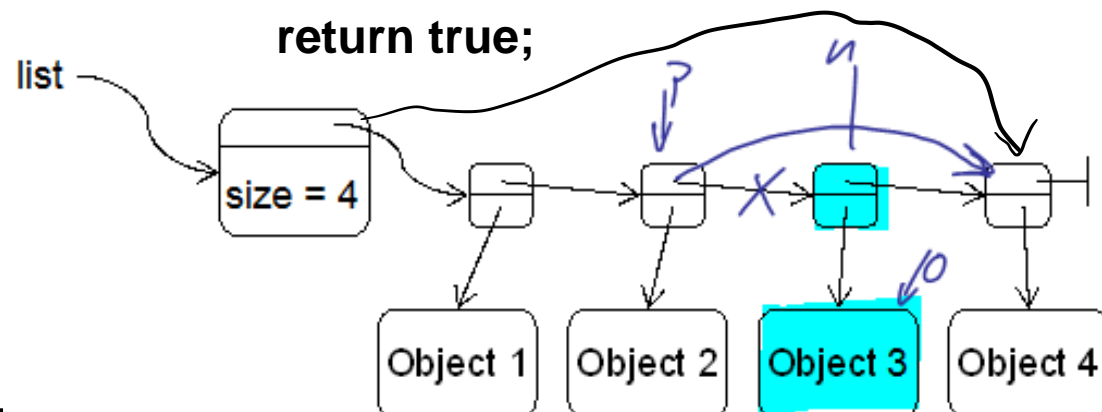
```
} // Tail pointer check
```

```
current.next = null; size--;
```

```
if (current == last) {
```

```
return true;
```

```
last = before;
```



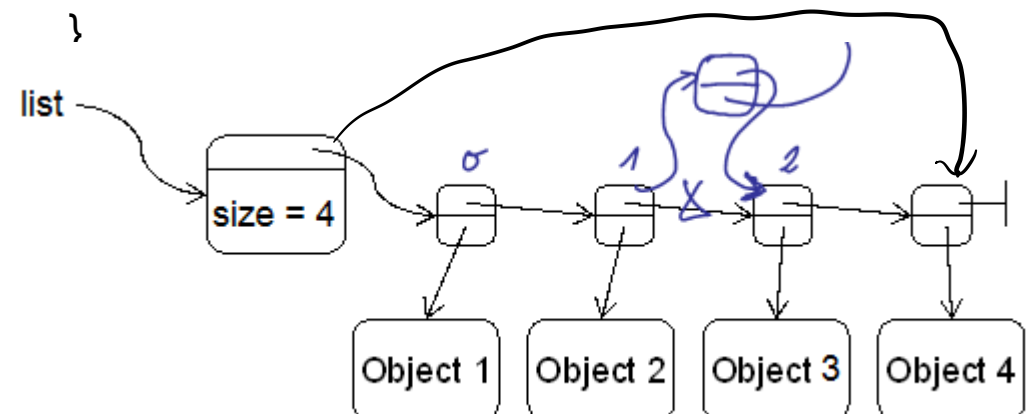
**Lösung: public void add(int index, E item) : Komplexität:  $O(n)$**

```

if (index < 0 || index > size) {
    throw new
    IndexOutOfBoundsException()
}
if (index == 0) {
    // First position ( $O(1)$ )
    addHead(item);
} else if (index == size) {
    // Last position ( $O(1)$ )
    addTail(item);
} else {
    // In Between  $O(n)$ 
    addElementAt(index, item);
}
    
```

```

private void addElementAt (int index, E
item) {
    Node<E> current = first;
    for (int i = 1; i < index; i++) {
        current = current.next;
    }
    Node<E> toAdd = new Node<E>(item);
    toAdd.next = current.next;
    current.next = toAdd; ++size;
}
    
```



## Lösung: Erlauben von null-Werten in der Collection

Anpassen der equals-Funktion, da sonst NullPointerException sowie auch Entfernen der Null-checks bei den Methoden.

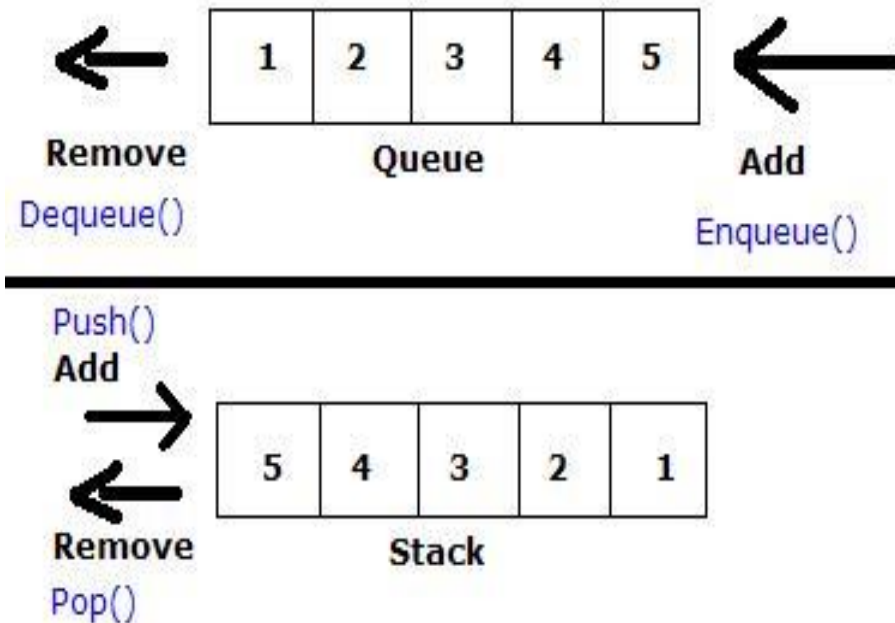
```
private boolean elementValueEquals(Object o, Node<E> current) {  
    if (o == null) {  
        return current.elem == null;  
    }  
    return o.equals(current.elem);  
}
```

Oder einfach:

```
Objects.equals(o, current.item) // (macht das selbe wie oben)
```

## Implementation: Stacks / Queues

- Basieren auf Listen
- Stack: LIFO (Last-in-first-out)
- Queue: FIFO (First-in-first-out)



## Feedback: Optionale Detail-Diskussion: For-Loop vs. While-Loop

### For-Loop:

- Anzahl Durchläufe bekannt

```
private void addElementAt(int index, E item) {  
    Node<E> current = first;  
    for (int i = 1; i < index; i++) {  
        current = current.next;  
    }  
    Node<E> toAdd = new Node<E>(item);  
    toAdd.next = current.next;  
    current.next = toAdd;  
    ++size;  
}
```

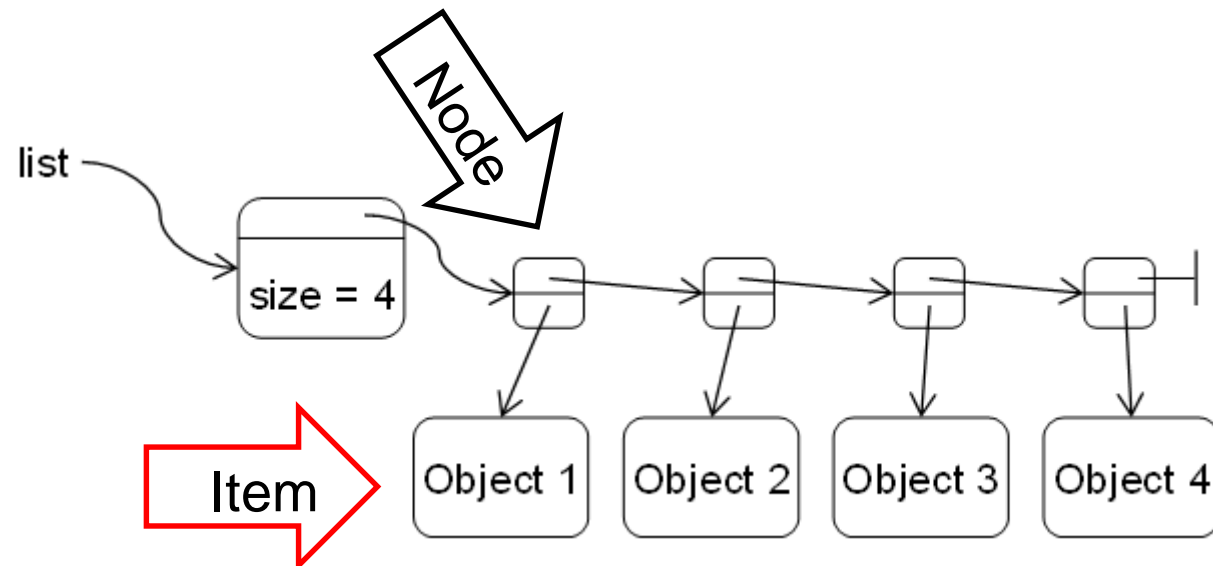
### While-Loop

- Anzahl Durchläufe nicht bekannt

```
public boolean contains(Object o) {  
    Node<E> current = first;  
    while (current != null &&  
        !elementValueEquals(o, current)) {  
        current = current.next;  
    }  
    return current != null;  
}
```

## Feedback: Achtung, equals auf dem richtigen Objekt

- ~~Falsch~~: `return first.equals(number);`
- **Richtig**: `return first.elem.equals(number);`



## Feedback: Achtung, Flüchtigkeitsfehler

- **Node<E> current = first;**
- **while (first != null && !first.item.equals(number)) {...}**



## Feedback: Integer entfernen

- **List.remove(1);**

```
@Override  
public E remove(int index) {
```

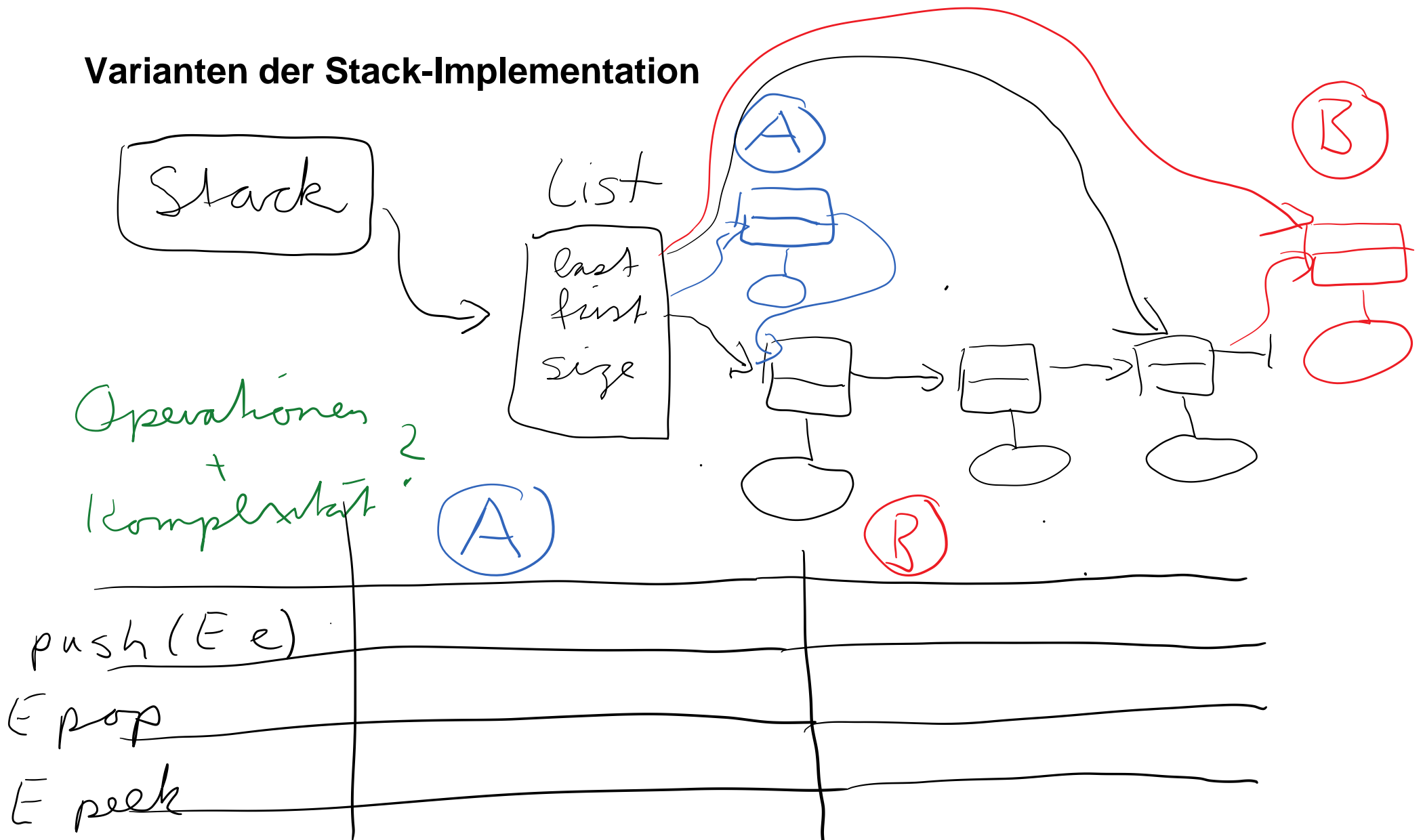
- **List.remove(Integer.valueOf(1));**

```
@Override  
public boolean remove(Object o) {
```

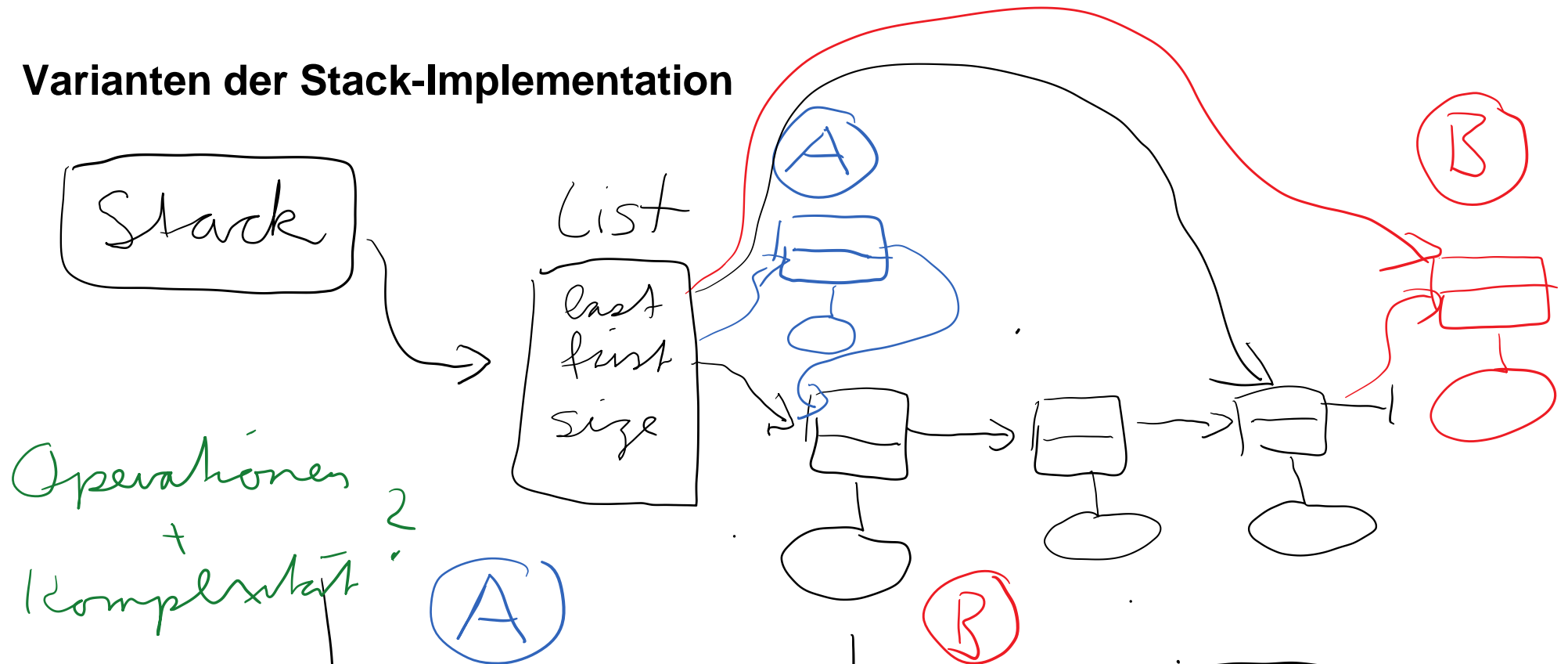
**First- und Lastzeiger anpassen (wichtig, nicht vergessen!)**

- **Bei allen add-Methoden**
- **Bei allen remove-Methoden**

## Varianten der Stack-Implementation



## Varianten der Stack-Implementation



Operationen  
+  
Komplexität?

	(A)		(B)	
push(E e)	add(o, e)	$O(1)$	add(e)	$O(1)$
E pop	remove(o)	$O(1)$	remove(size()-1)	$O(n)$ !
E peek	get(o)	$O(1)$	get(size()-1)	$O(1)$
		$O(1)$		$O(n)$