

Praktische Informatik 2

Grundlegende Datenstrukturen

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für
Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen



Abstrakte Datentypen (In Java: Klassen)

- Ein abstrakter Datentyp besteht aus den Daten selbst (z.B. Objektzustand) und den auf den Daten auszuführenden Operationen (z.B. Methoden)
- Interne Abläufe werden versteckt (Geheimnisprinzip)
- Beispiele
 - **plus : int × int → int**
 - **null : → int**
 - **neg : int → int**
- Semantik der Operationen kann formal beschrieben werden, z.B. mit Axiomen

```
plus(a, b) = plus(b, a)
plus(a, null) = a
plus(a, plus(b, c)) = plus(plus(a, b), c)
plus(a, neg(a)) = null
```

Stapel (Stack)

- Dient zum Zwischenspeichern von Elementen
- Der Zugriff erfolgt nach dem **last-in, first-out** Prinzip (**lifo**), d.h. das **zuletzt** abgelegte Element wird **zuerst** zurückgeliefert
- Stapel erzeugen: **new** : $\rightarrow \text{Stack}\langle E \rangle$
- Eintrag ablegen: **push** : $\text{Stack}\langle E \rangle \times E \rightarrow \text{Stack}\langle E \rangle$
- Kopf auslesen: **top** : $\text{Stack}\langle E \rangle \rightarrow E$
- Eintrag entnehmen (und zurückliefern): **pop** : $\text{Stack}\langle E \rangle \rightarrow \text{Stack}\langle E \rangle (\times E)$
- Auf Leere testen: **empty** : $\text{Stack}\langle E \rangle \rightarrow \text{boolean}$

```
top(push(s, e)) = e
pop(push(s, e)) = s
empty(new)
¬ empty(push(s, e))
```

Beschränkter Stapel: Beispiel

Stack<Integer>		top() → 8								
size	3									
values	17	5	<u>8</u>	0	0	0	0	0	0	0

Stack<Integer>		push(6)								
size	4									
values	17	5	8	<u>6</u>	0	0	0	0	0	0

Stack<Integer>		pop() → 6								
size	3									
values	17	5	<u>8</u>	6	0	0	0	0	0	0

Beschränkter Stapel: Demo

The screenshot shows an IDE window titled "ADTs - Stack.java". The left sidebar displays a project structure with folders for "DLRList", "IndexedList", "Queue", "SLList", "SLNode", and "Stack". The main editor area shows the following Java code:

```
2  
3  
4  /**  
5   * Beschränkter Stapel (last-in, first-out).  
6   * @param <E> Der Typ der Werte, die im Stapel gespeichert werden  
7   */  
8  public class Stack<E>  
9  {  
10     /** Die gespeicherten Werte. */  
11     private final E[] values;  
12  
13     /** Die Anzahl der Werte im Stapel. */  
14     private int size = 0;  
15  
16     /**  
17     * Konstruktor für einen leeren Stapel.  
18     * @param capacity Die Maximalkapazität des Stapels.  
19     */  
20     /unchecked/  
21     public Stack(final int capacity)  
22     {
```

The status bar at the bottom indicates "Build completed successfully in 1 sec, 553 ms (2 minutes ago)" and "10:30 LF UTF-8 4 spaces master".

Warteschlange (Queue)

- Dient zum Zwischenspeichern von Daten
- Zugriff erfolgt nach dem **first-in, first-out** Prinzip (**fifo**), d.h. das zuerst abgelegte Element wird zuerst zurückgeliefert
- Warteschlange erzeugen: **new** : \rightarrow **Queue<E>**
- Eintrag ablegen: **push** : **Queue<E>** \times **E** \rightarrow **Queue<E>**
- Kopf auslesen: **top** : **Queue<E>** \rightarrow **E**
- Eintrag entnehmen (und zurückliefern): **pop** : **Queue<E>** \rightarrow **Queue<E>** (\times **E**)
- Auf Leere testen: **empty** : **Queue<E>** \rightarrow **boolean**

```
top(push(new, e)) = e
pop(push(new, e)) = new
empty(new)
¬ empty(push(q, e))
top(push(q, e)) = top(q), q ≠ new
pop(push(q, e)) =
    push(pop(q), e), q ≠ new
```

Beschränkte Warteschlange: Beispiel

Queue<Integer>	
first	2
size	8
values	17 5 <u>8</u> 0 7 4 42 19 11 0

top() → 8

Queue<Integer>	
first	2
size	9
values	6 5 <u>8</u> 0 7 4 42 19 11 0

push(6)

Queue<Integer>	
first	3
size	8
values	6 5 <u>8</u> 0 7 4 42 19 11 0

pop() → 8

Beschränkte Warteschlange: Demo

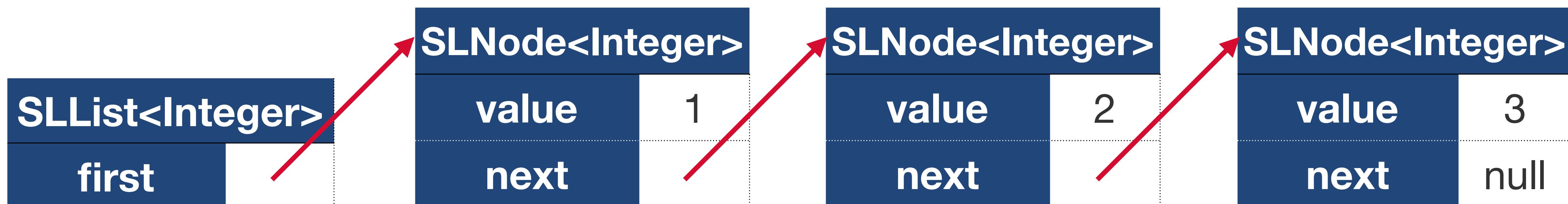
The screenshot shows an IDE window titled "ADTs - Stack.java". The left sidebar displays a project structure with folders for "DLRList", "IndexedList", "Queue", "SLList", "SLNode", and "Stack". The main editor area shows the following Java code:

```
2
3
4  /**
5   * Beschränkter Stapel (last-in, first-out).
6   * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
7   */
8  public class Stack<E>
9  {
10     /** Die gespeicherten Werte. */
11     private final E[] values;
12
13     /** Die Anzahl der Werte im Stapel. */
14     private int size = 0;
15
16     /**
17     * Konstruktor für einen leeren Stapel.
18     * @param capacity Die Maximalkapazität des Stapels.
19     */
20     /unchecked/
21     public Stack(final int capacity)
22     {
```

The status bar at the bottom indicates "Build completed successfully in 1 sec, 553 ms (2 minutes ago)" and "10:30 LF UTF-8 4 spaces master".

Einfach verkettete Liste

- Jedes Element enthält einen Zeiger auf seinen Nachfolger
- Kann nur in einer Richtung durchlaufen werden
- Einfügen immer am Anfang der Liste oder hinter einem bekannten Listenelement
- Ein Element kann nur gelöscht werden, wenn sein Vorgänger bekannt ist
- Einfügen und Löschen sind in konstanter Zeit möglich
- Speicherplatzverbrauch höher als bei Arrays



Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

push(4)

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

push(4)

SLNode<Integer>	
value	4
next	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

push(4)

SLNode<Integer>	
value	4
next	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

push(4)

SLNode<Integer>	
value	4
next	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

pop() → 4

push(4)

SLNode<Integer>	
value	4
next	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Stapel mit einfach verketteter Liste

Stack<Integer>	
first	

pop() → 4

push(4)

SLNode<Integer>	
value	4
next	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	null

Warteschlange mit einfach verketteter Liste

Queue<Integer>	
first	
last	

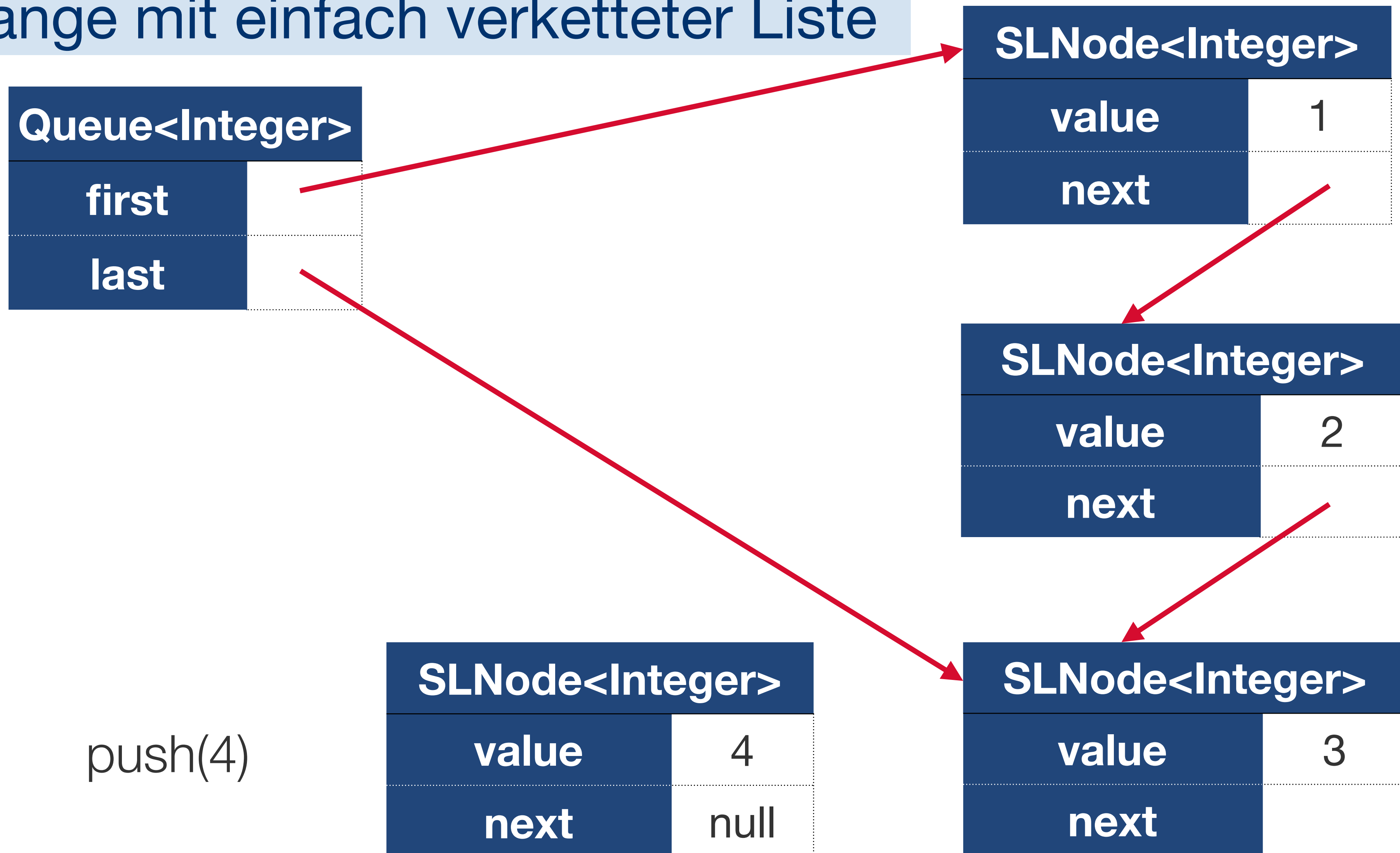
SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	

push(4)

Warteschlange mit einfach verketteter Liste



Warteschlange mit einfach verketteter Liste

Queue<Integer>	
first	
last	

SLNode<Integer>	
value	1
next	

SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	4
next	null

SLNode<Integer>	
value	3
next	

push(4)

Warteschlange mit einfach verketteter Liste

Queue<Integer>	
first	
last	

SLNode<Integer>	
value	1
next	

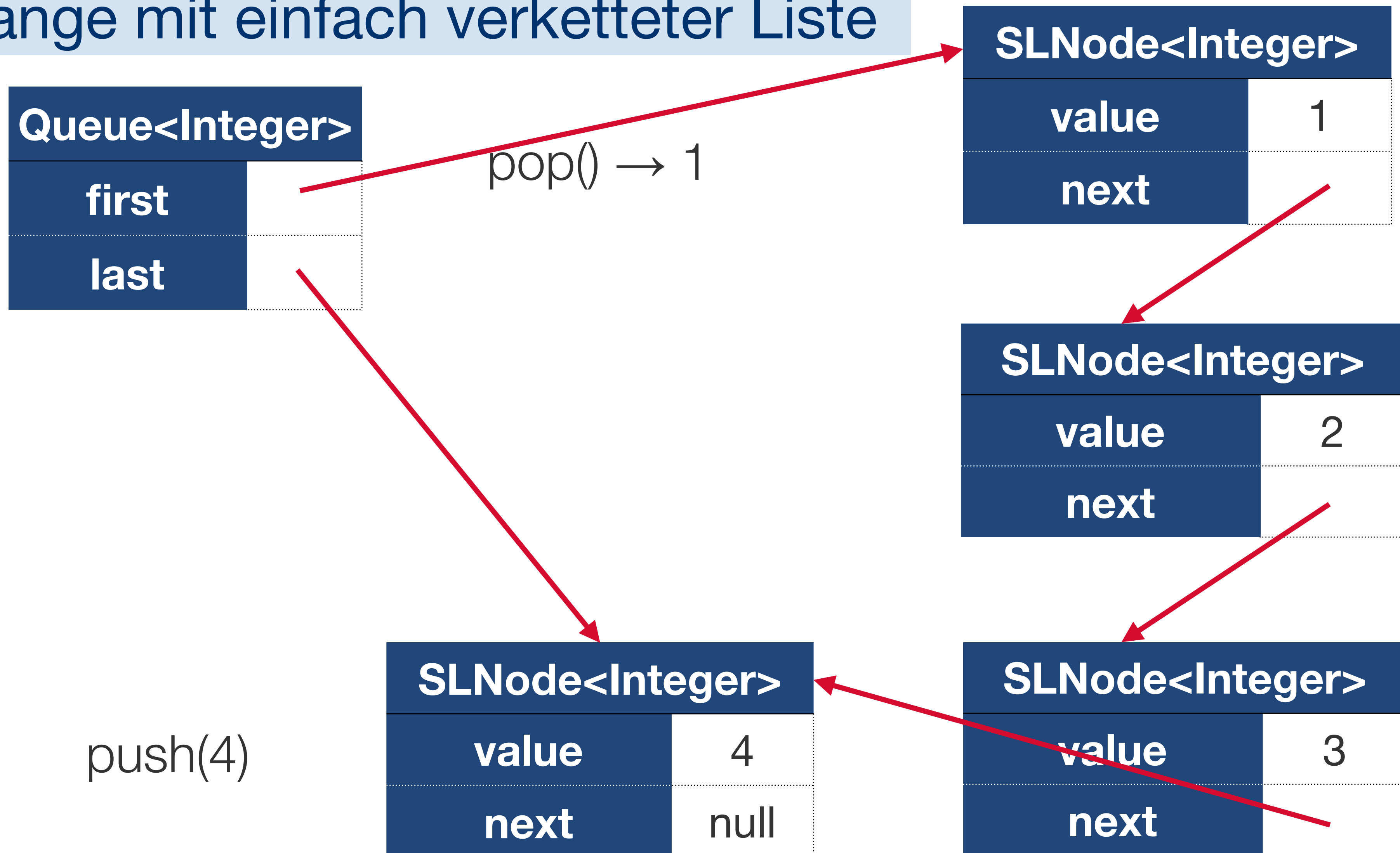
SLNode<Integer>	
value	2
next	

SLNode<Integer>	
value	3
next	

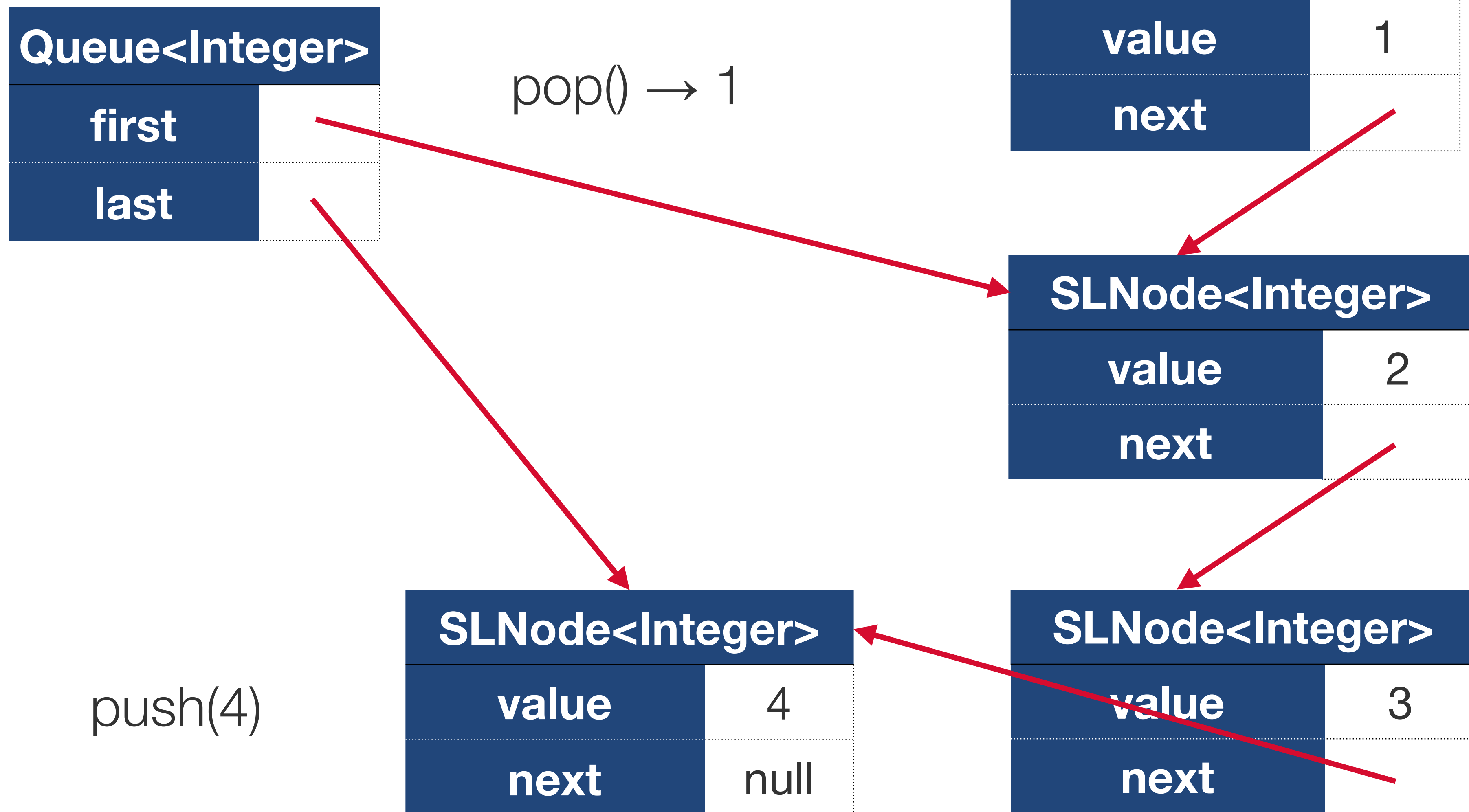
SLNode<Integer>	
value	4
next	null

push(4)

Warteschlange mit einfach verketteter Liste



Warteschlange mit einfach verketteter Liste



Operationen auf Listen

- Liste erzeugen: **new** : $\rightarrow \text{SLList}\langle E \rangle$
- Ersten Knoten liefern: **first** : $\text{SLList}\langle E \rangle \rightarrow \text{SLNode}\langle E \rangle$
- Eintrag einfügen: **insert** : $\text{SLList}\langle E \rangle \times E \times \text{SLNode}\langle E \rangle \rightarrow \text{SLList}\langle E \rangle$
- Knoten löschen: **remove** : $\text{SLList}\langle E \rangle \times \text{SLNode}\langle E \rangle \rightarrow \text{SLList}\langle E \rangle$
- Auf Leere testen: **empty** : $\text{SLList}\langle E \rangle \rightarrow \text{boolean}$
- Knoten erzeugen: **new** : $E \times \text{SLNode}\langle E \rangle \rightarrow \text{SLNode}\langle E \rangle$
- Nächsten Knoten liefern: **next** : $\text{SLNode}\langle E \rangle \rightarrow \text{SLNode}\langle E \rangle$
- Wert liefern: **value** : $\text{SLNode}\langle E \rangle \rightarrow E$



$\text{SLList}\langle E \rangle$



$\text{SLNode}\langle E \rangle$

Einfach verkettete Liste: Demo

```
ADTs - Stack.java
05_ADTs > src > de > uni_bremen > pi2 > Stack > values
Project: 05_ADTs
Structure: DLRLIST, IndexedList, Queue, SLList, SLNode, Stack
External Libraries
Scratches and Consoles
JShell Console: adts.snippet, fibonacci.snippet, hanoi.snippet, lambda.snippet, maxSumme.snippet, node.snippet, priority_queue.snippet, queens.snippet, red_black.snippet, ring_buffer.snippet, rucksack.snippet, search.snippet, search_tree.snippet, uebung3.snippet
Favorites

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

/**
 * Beschränkter Stapel (last-in, first-out).
 * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
 */
public class Stack<E>
{
    /** Die gespeicherten Werte. */
    private final E[] values;

    /** Die Anzahl der Werte im Stapel. */
    private int size = 0;

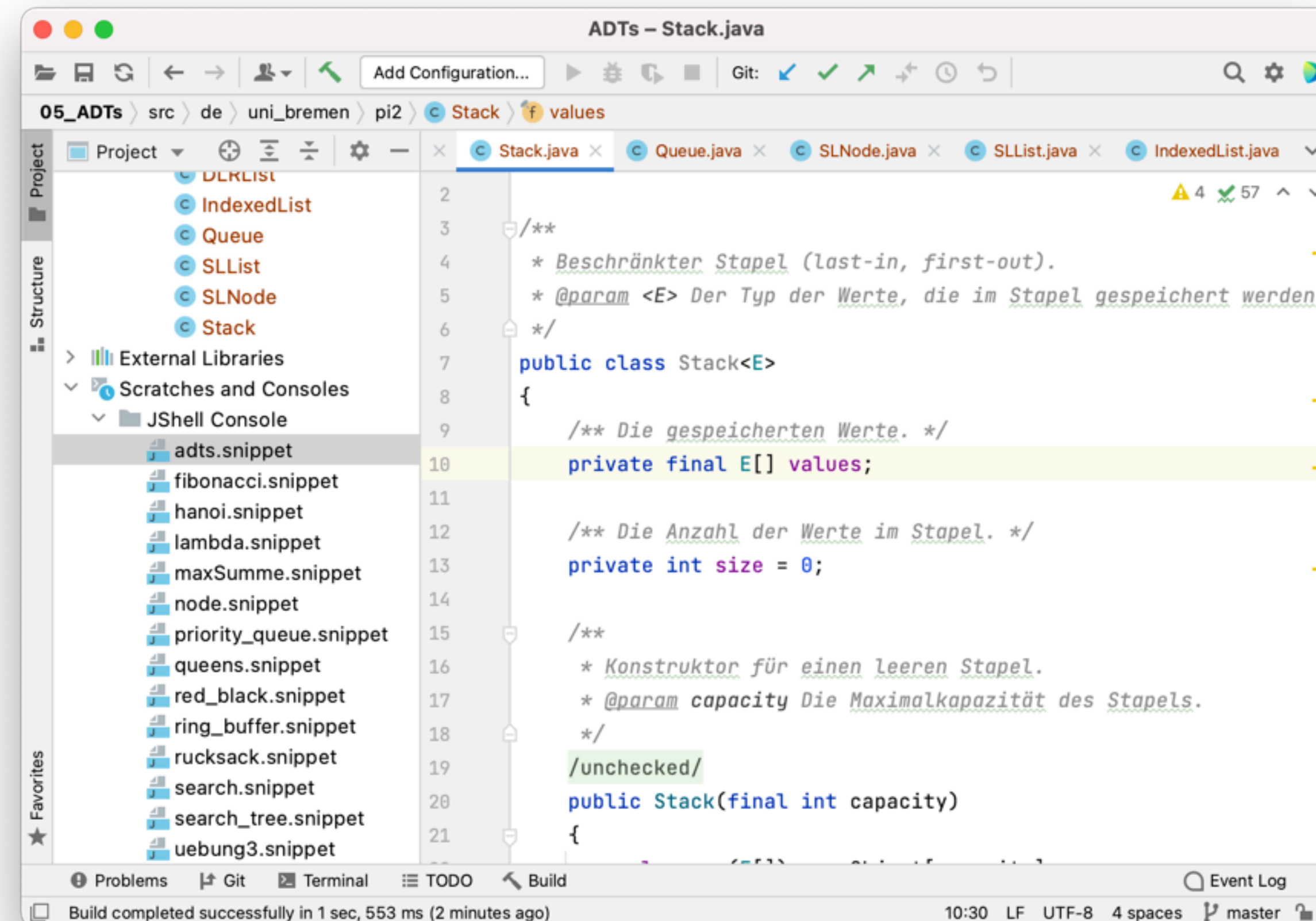
    /**
     * Konstruktor für einen leeren Stapel.
     * @param capacity Die Maximalkapazität des Stapels.
     */
    /unchecked/
    public Stack(final int capacity)
    {
```

Build completed successfully in 1 sec, 553 ms (2 minutes ago) 10:30 LF UTF-8 4 spaces master

Einfach verkettete Liste mit Index

- Bildet ein Array dynamischer Größe mithilfe einer Liste nach, wodurch Einfügen und Löschen ohne Umkopieren möglich sind
- Liste erzeugen: **new** : $\rightarrow \text{IndexedList}\langle E \rangle$
- Eintrag schreiben: **set** : $\text{IndexedList}\langle E \rangle \times \text{int} \times E \rightarrow \text{IndexedList}\langle E \rangle$
- Eintrag auslesen: **get** : $\text{IndexedList}\langle E \rangle \times \text{int} \rightarrow E$
- Einfügen: **insert** : $\text{IndexedList}\langle E \rangle \times \text{int} \times E \rightarrow \text{IndexedList}\langle E \rangle$
- Löschen: **remove** : $\text{IndexedList}\langle E \rangle \times \text{int} \rightarrow \text{IndexedList}\langle E \rangle$
- Größe: **size** : $\text{IndexedList}\langle E \rangle \rightarrow \text{int}$

Einfach verkettete Liste mit Index: Demo



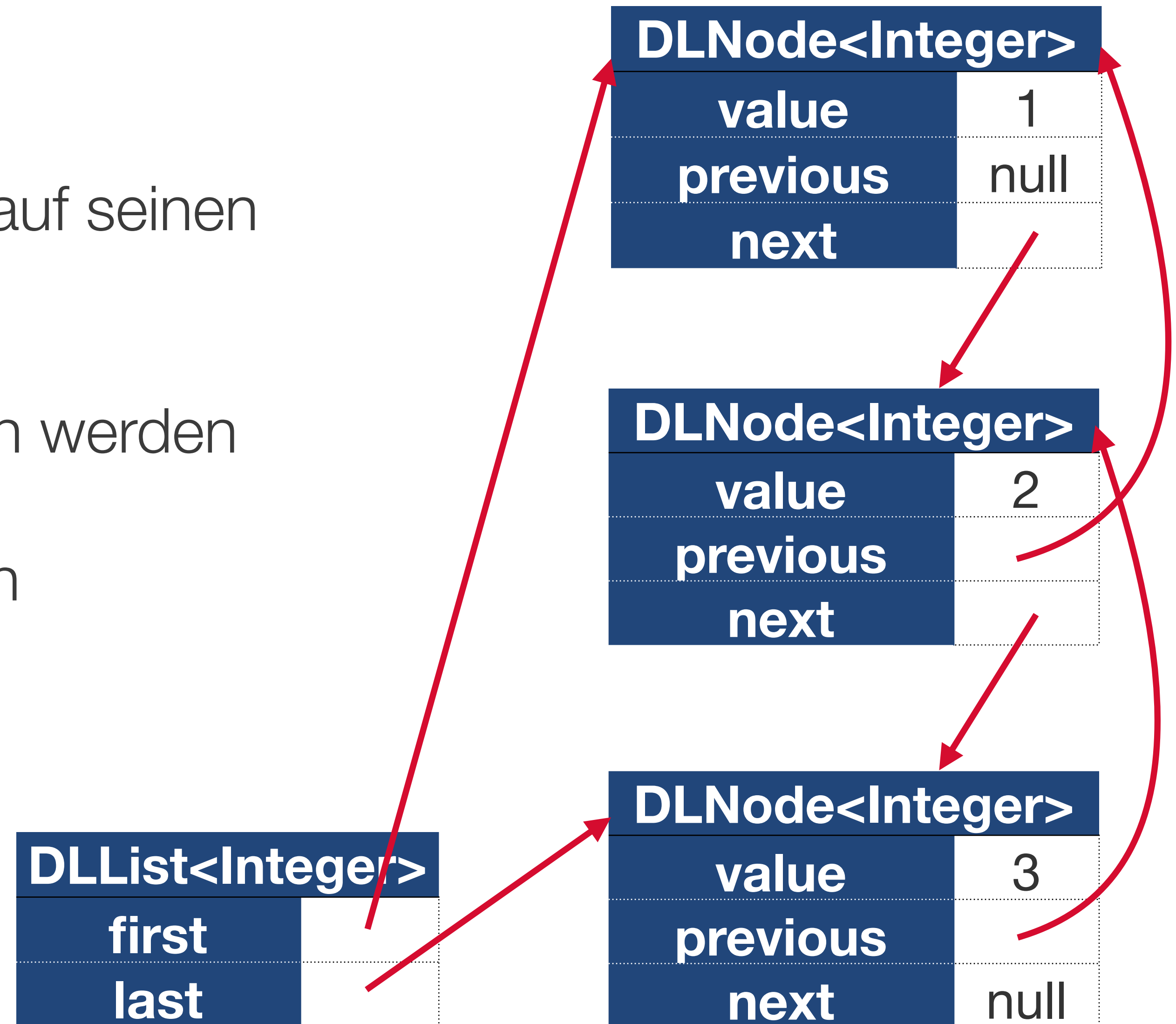
The screenshot shows an IDE window titled "ADTs - Stack.java". The left sidebar displays a project structure with folders for "DLRLIST", "IndexedList", "Queue", "SLList", "SLNode", and "Stack". Below these are "External Libraries", "Scratches and Consoles", and "Favorites". The main editor area shows the following Java code:

```
2
3
4  /**
5   * Beschränkter Stapel (last-in, first-out).
6   * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
7   */
8  public class Stack<E>
9  {
10     /** Die gespeicherten Werte. */
11     private final E[] values;
12
13     /** Die Anzahl der Werte im Stapel. */
14     private int size = 0;
15
16     /**
17     * Konstruktor für einen leeren Stapel.
18     * @param capacity Die Maximalkapazität des Stapels.
19     */
20     /unchecked/
21     public Stack(final int capacity)
22     {
```

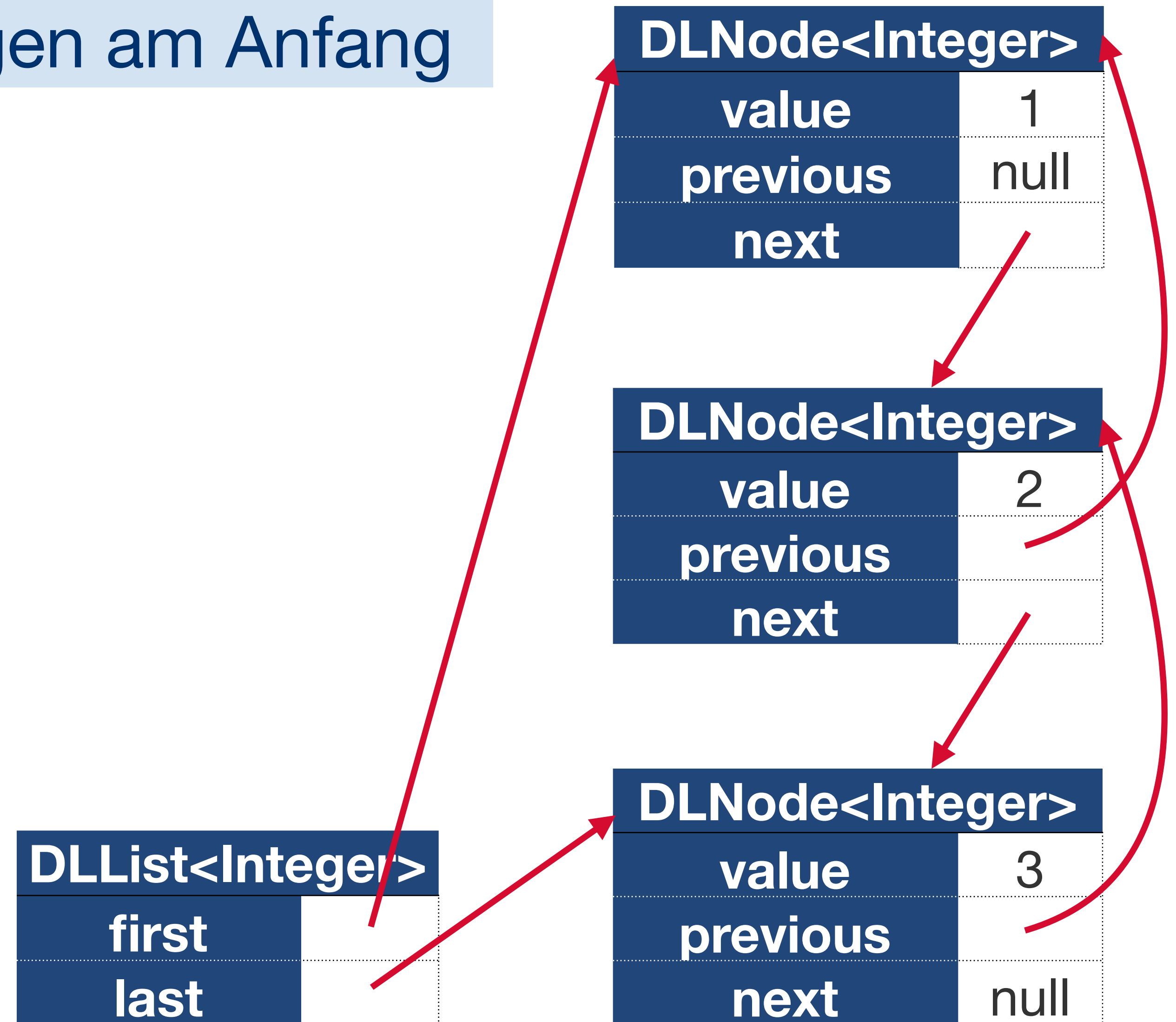
The status bar at the bottom indicates "Build completed successfully in 1 sec, 553 ms (2 minutes ago)" and shows the time "10:30" along with other settings like "LF", "UTF-8", "4 spaces", and "master".

Doppelt verkettete Liste

- Jedes Element enthält einen Zeiger auf seinen Vorgänger und seinen Nachfolger
- Kann vor- und rückwärts durchlaufen werden
- Einfügen vor und hinter existierenden Listenelementen
- Löschen von Elementen problemlos möglich



Doppelt verkettete Liste: Einfügen am Anfang



Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

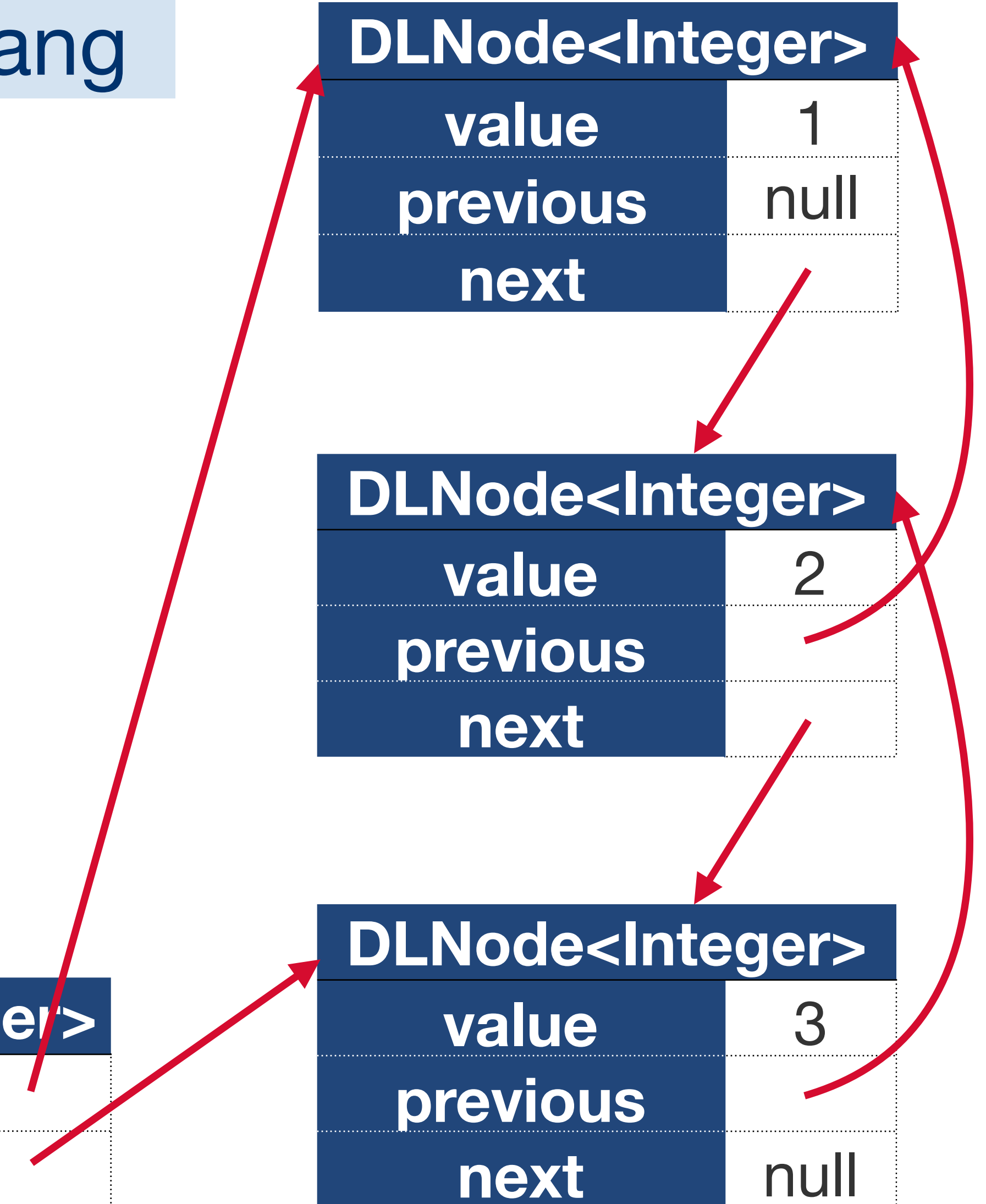
DLNode<Integer>	
value	4
previous	
next	

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null



Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

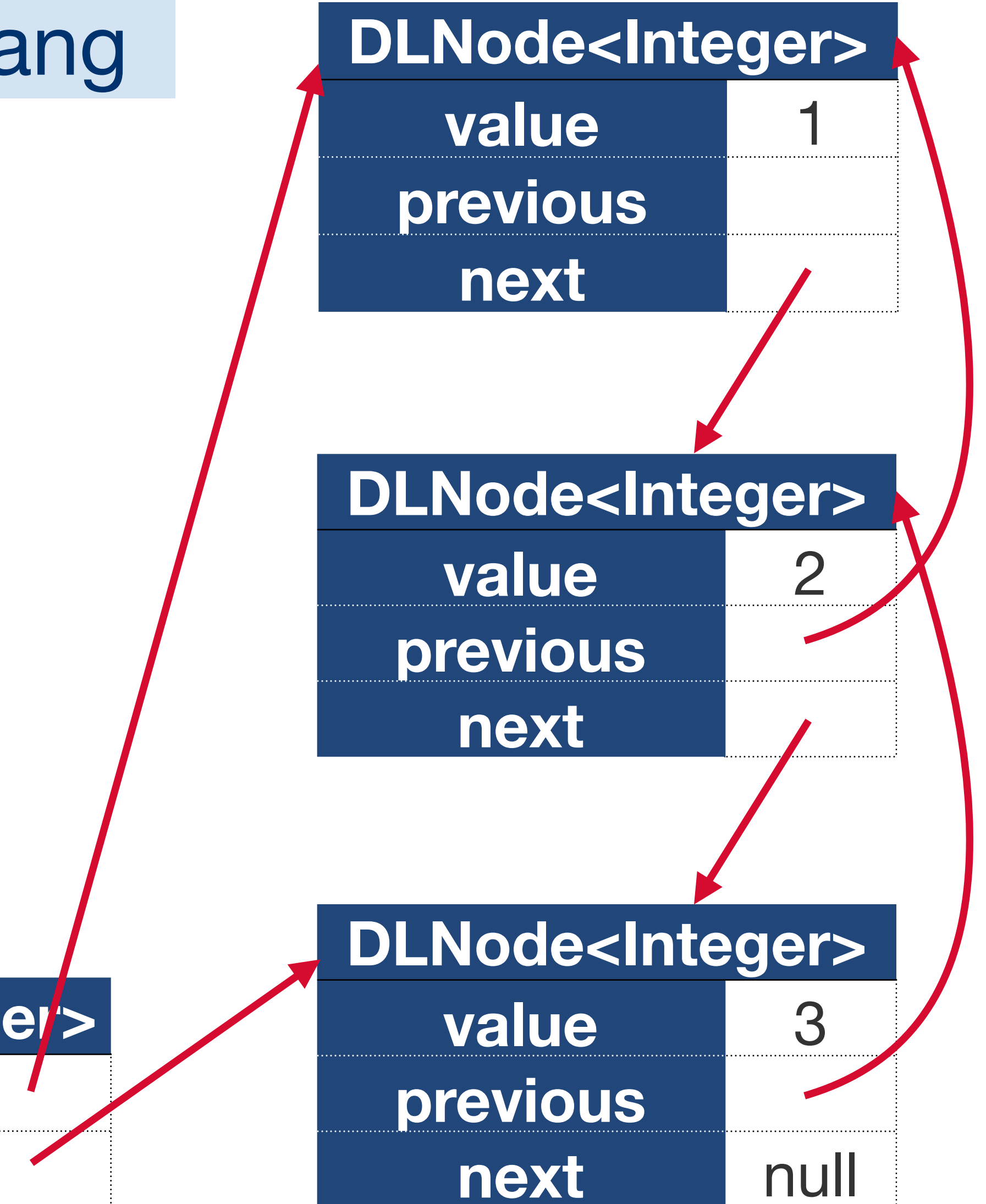
DLNode<Integer>	
value	4
previous	null
next	

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null



Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

DLNode<Integer>	
value	4
previous	null
next	

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null

Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)

DLNode<Integer>	
value	4
previous	null
next	

DLList<Integer>	
first	
last	

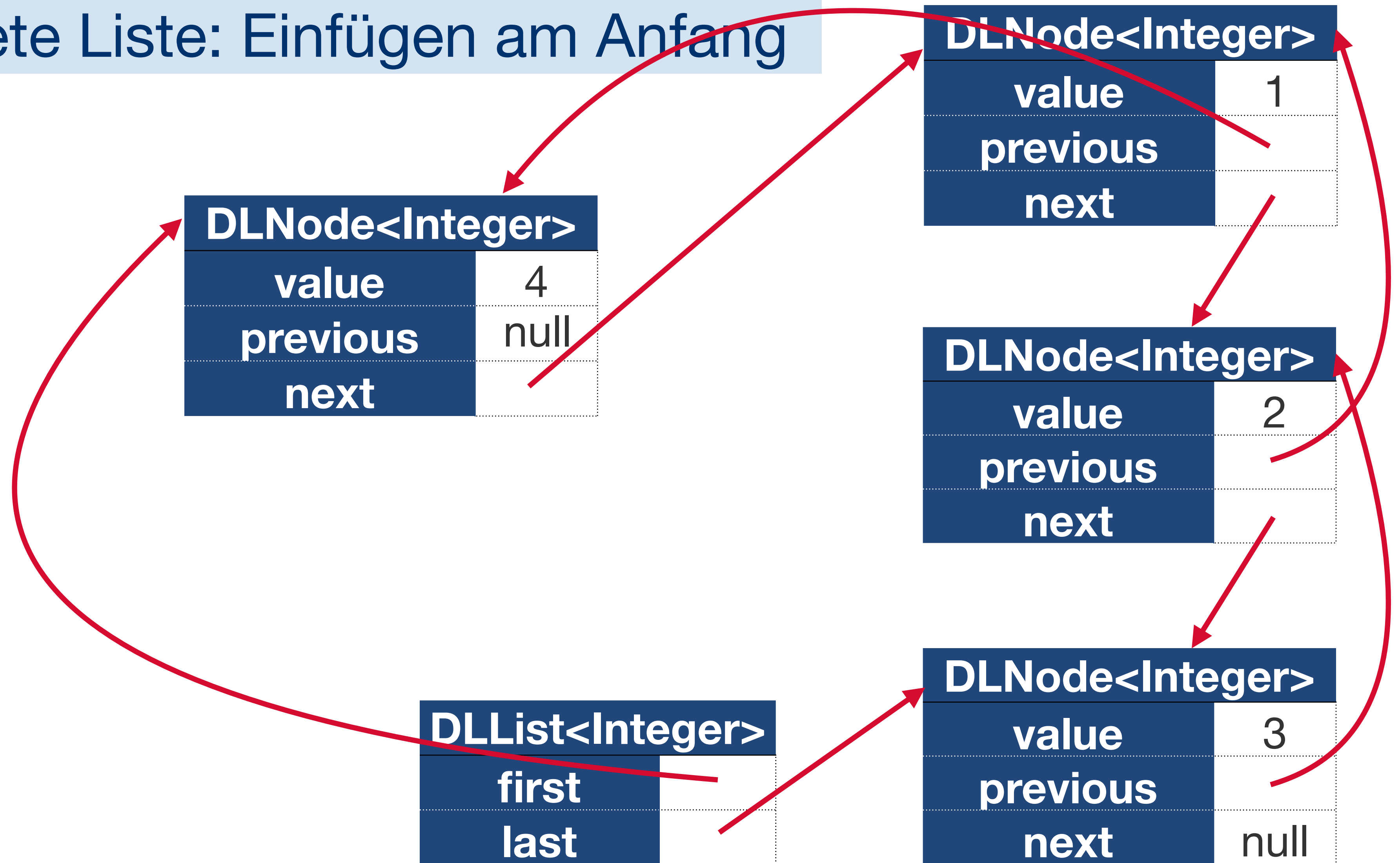
DLNode<Integer>	
value	1
previous	
next	

DLNode<Integer>	
value	2
previous	
next	

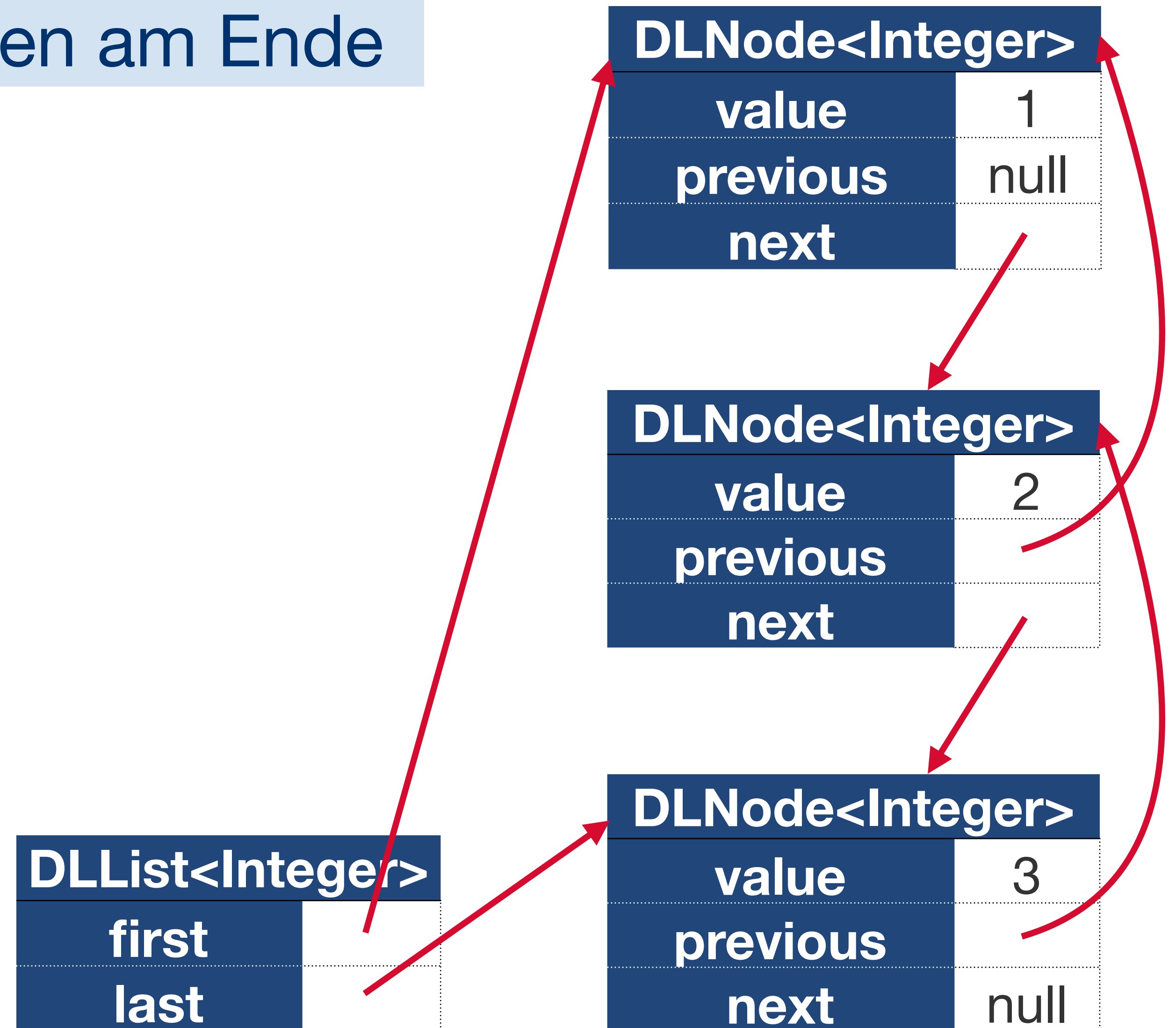
DLNode<Integer>	
value	3
previous	
next	null

Doppelt verkettete Liste: Einfügen am Anfang

insert(4, first)



Doppelt verkettete Liste: Einfügen am Ende



Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)

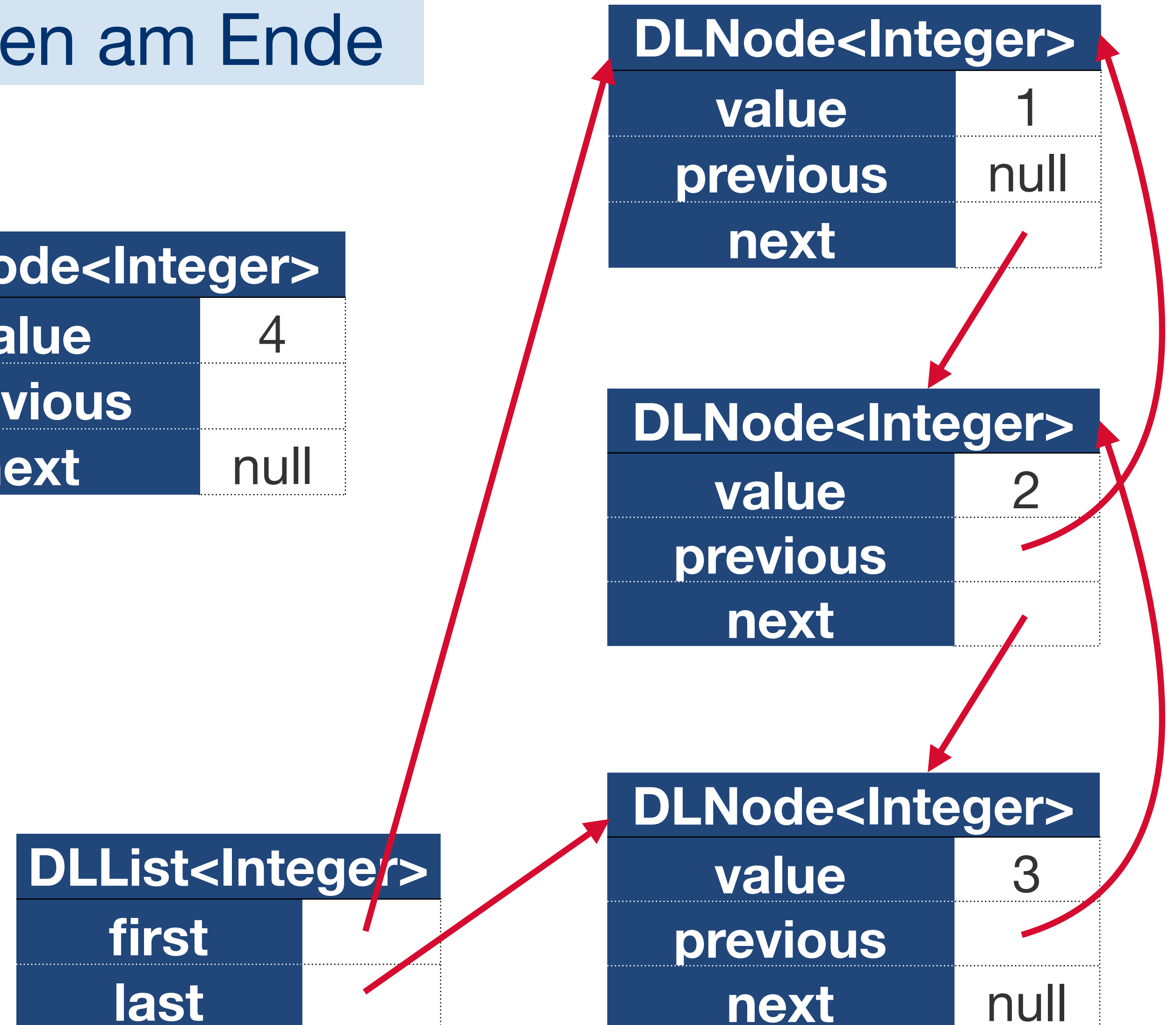
DLNode<Integer>	
value	4
previous	
next	null

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null



Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)

DLNode<Integer>	
value	4
previous	
next	null

DLList<Integer>	
first	
last	

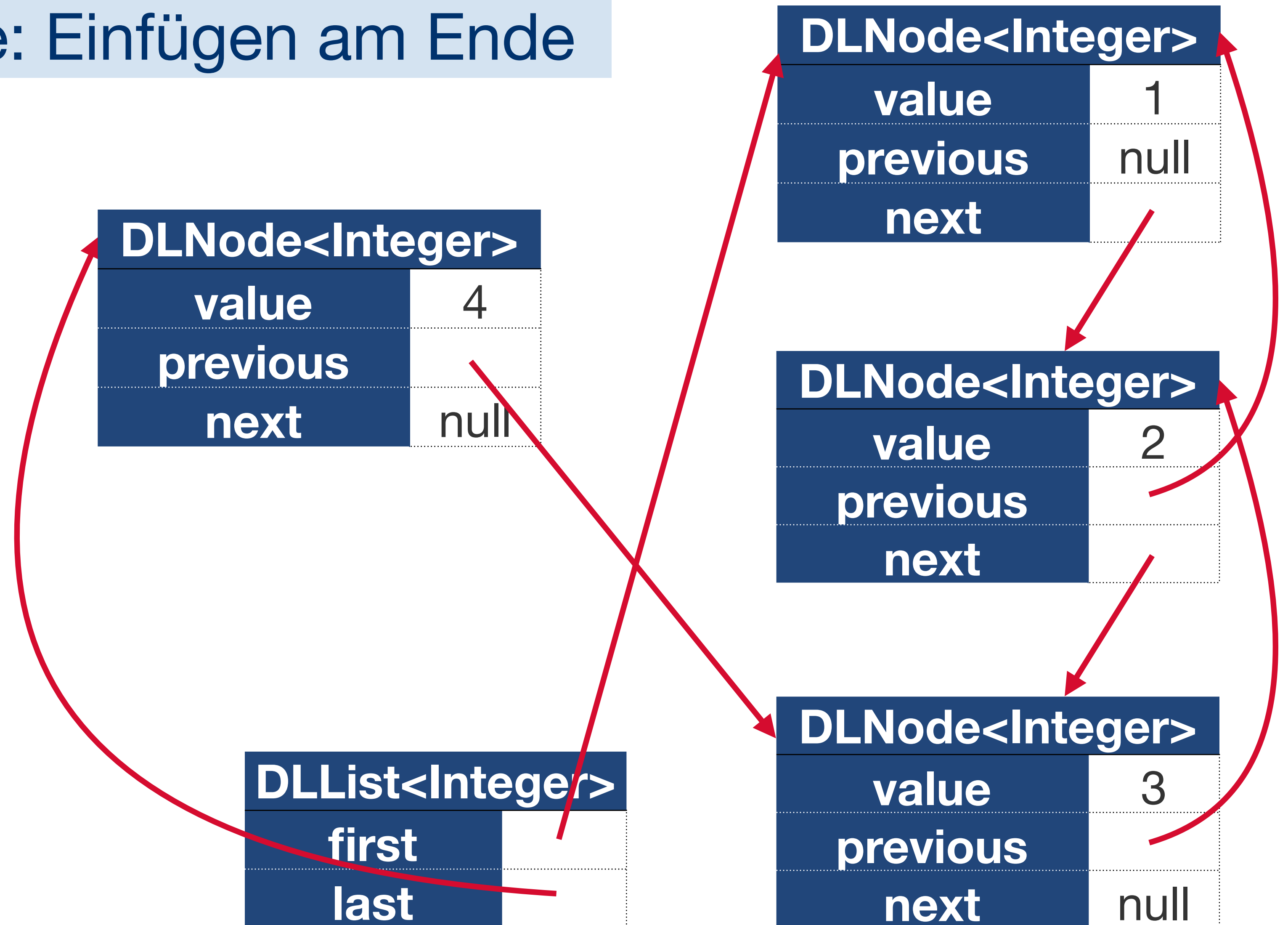
DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null

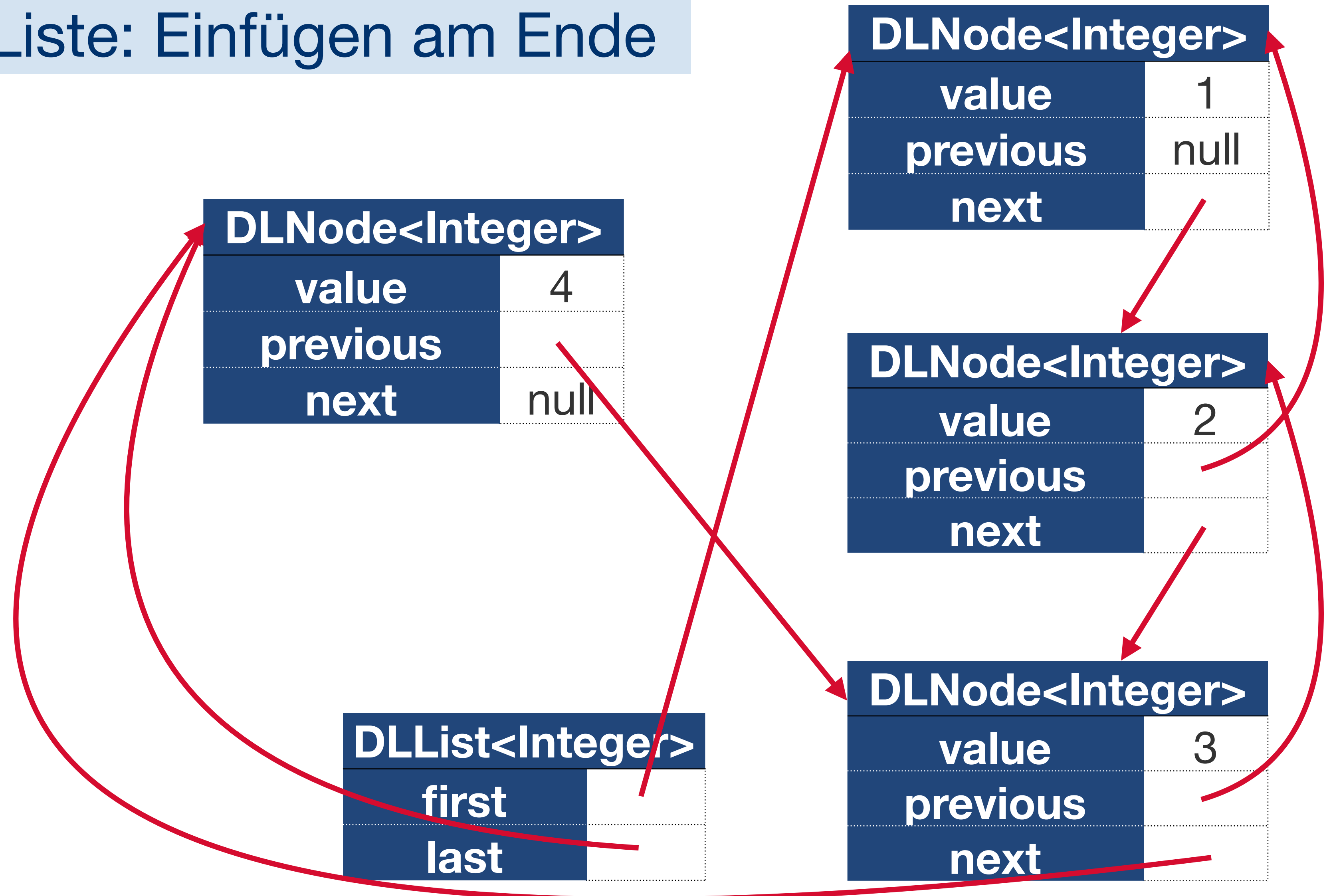
Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)

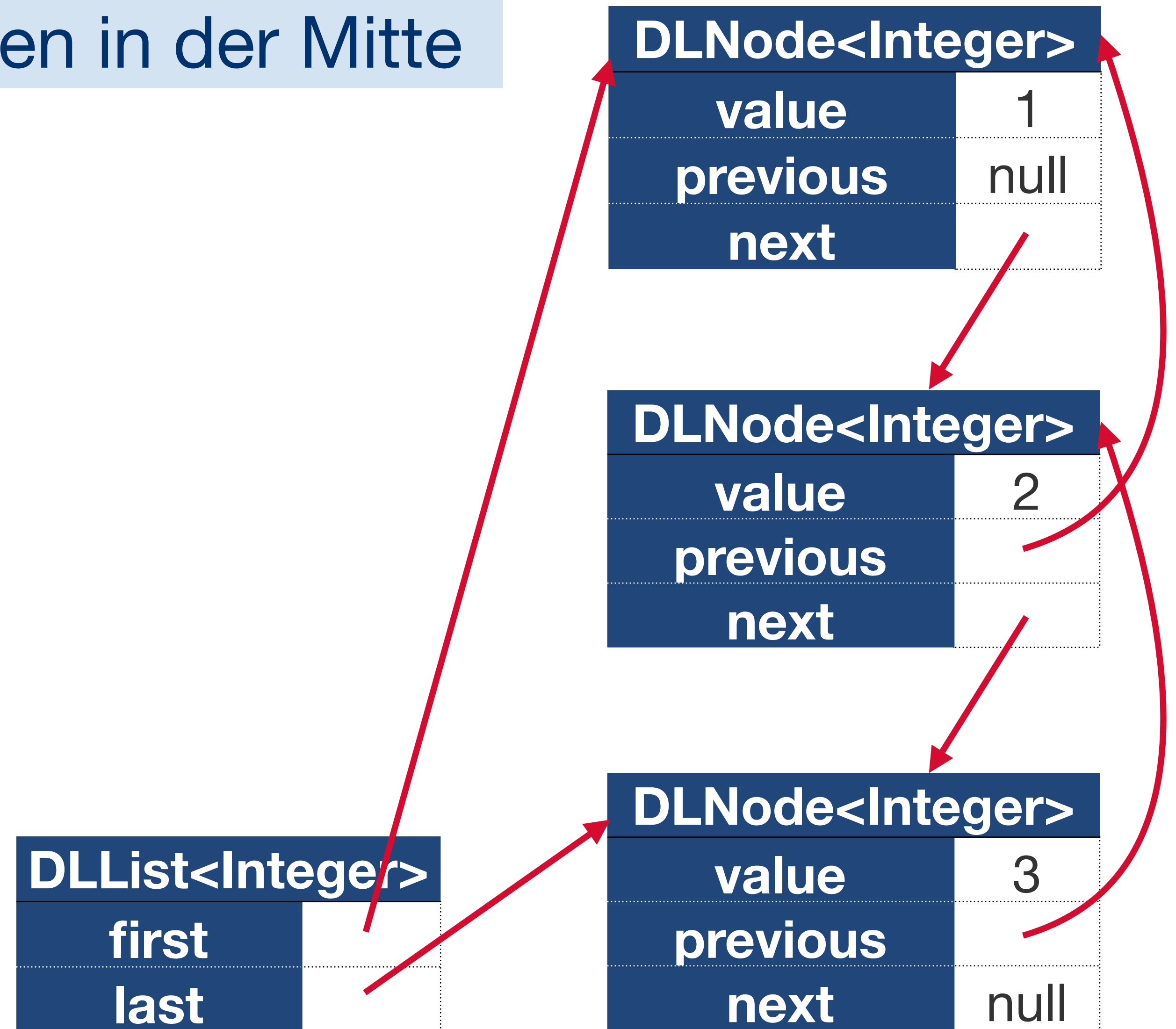


Doppelt verkettete Liste: Einfügen am Ende

insert(4, null)



Doppelt verkettete Liste: Einfügen in der Mitte



Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)

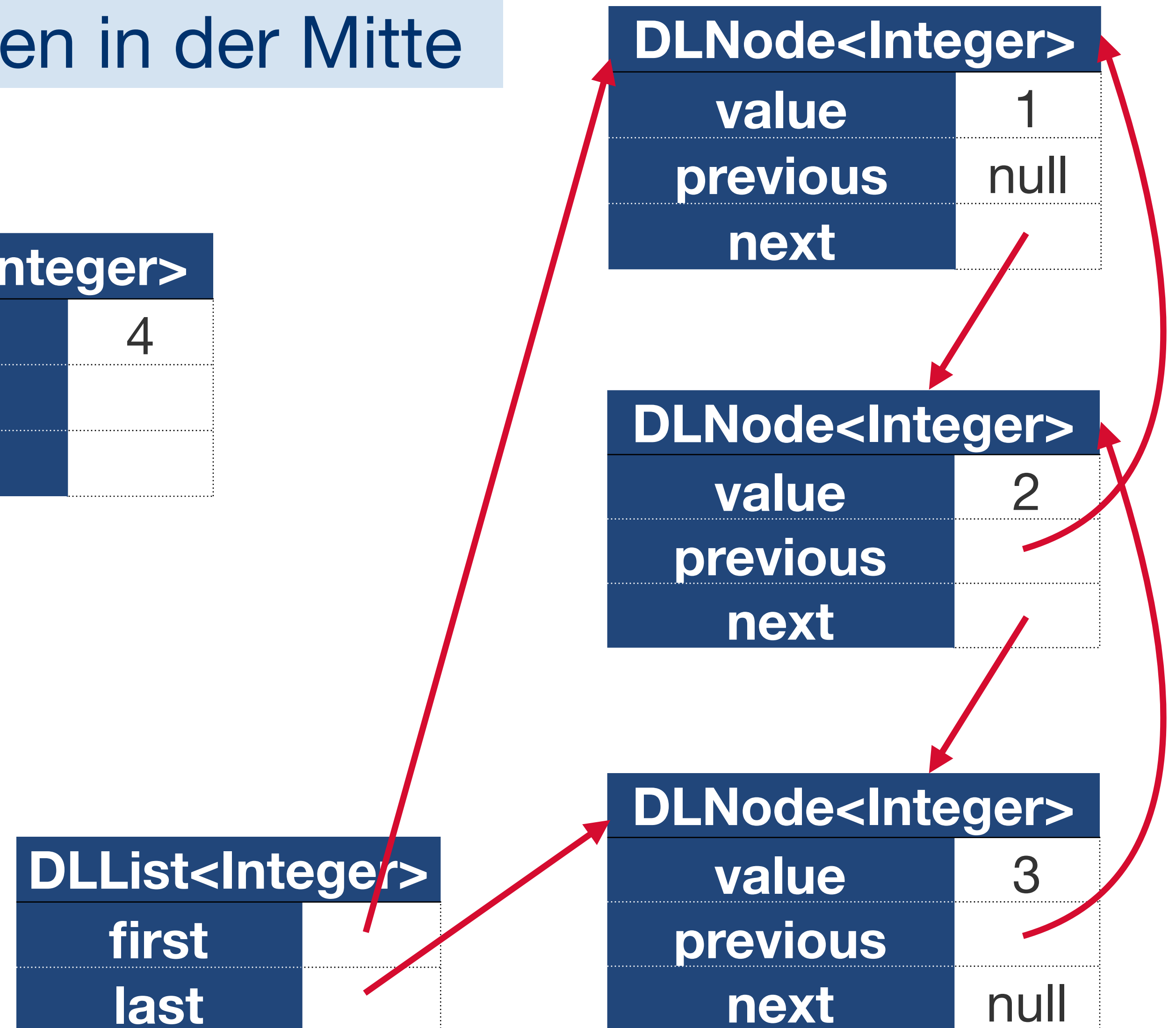
DLNode<Integer>	
value	4
previous	
next	

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null



Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)

DLNode<Integer>	
value	4
previous	
next	

DLList<Integer>	
first	
last	

DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null

Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)

DLNode<Integer>	
value	4
previous	
next	

DLList<Integer>	
first	
last	

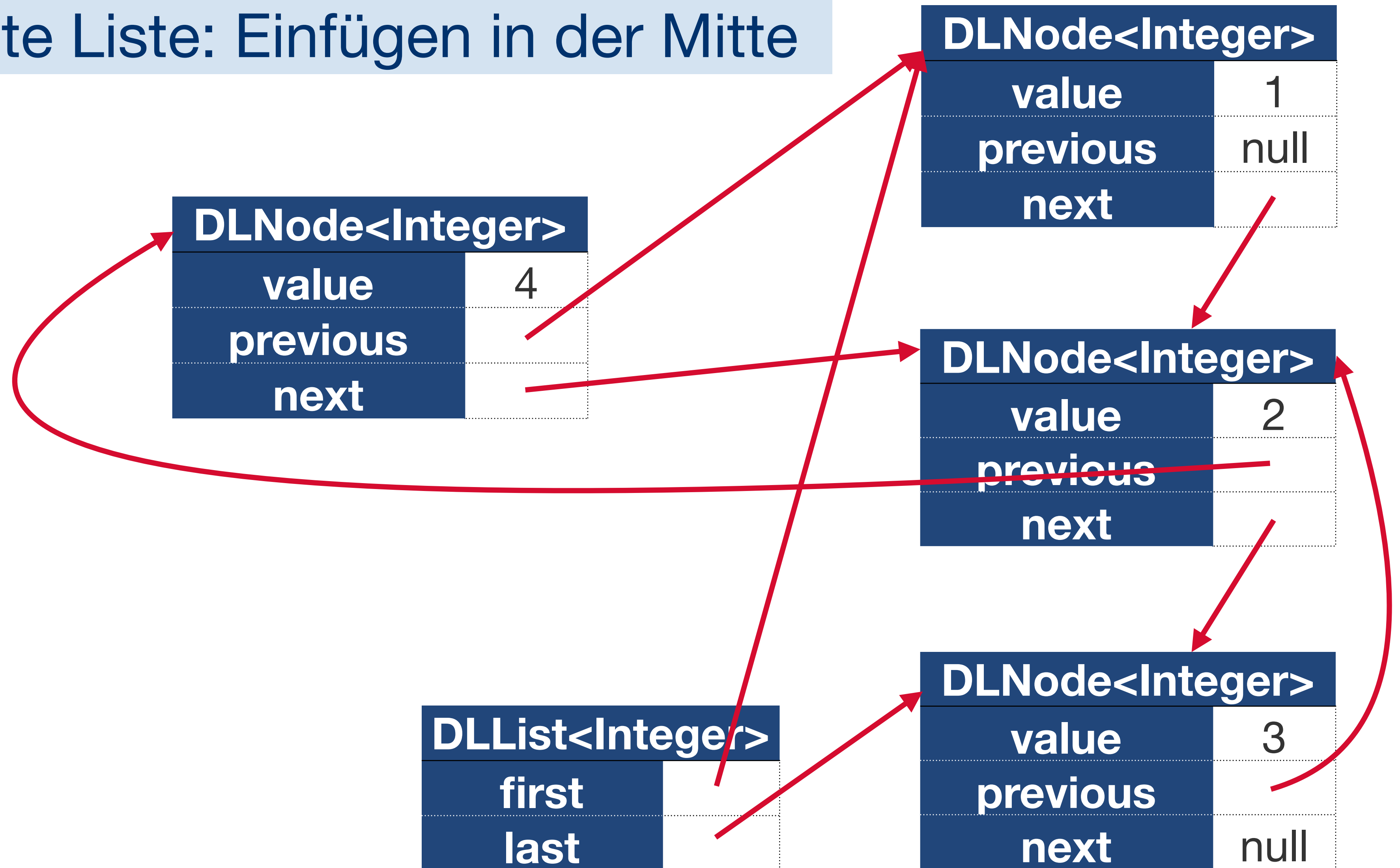
DLNode<Integer>	
value	1
previous	null
next	

DLNode<Integer>	
value	2
previous	
next	

DLNode<Integer>	
value	3
previous	
next	null

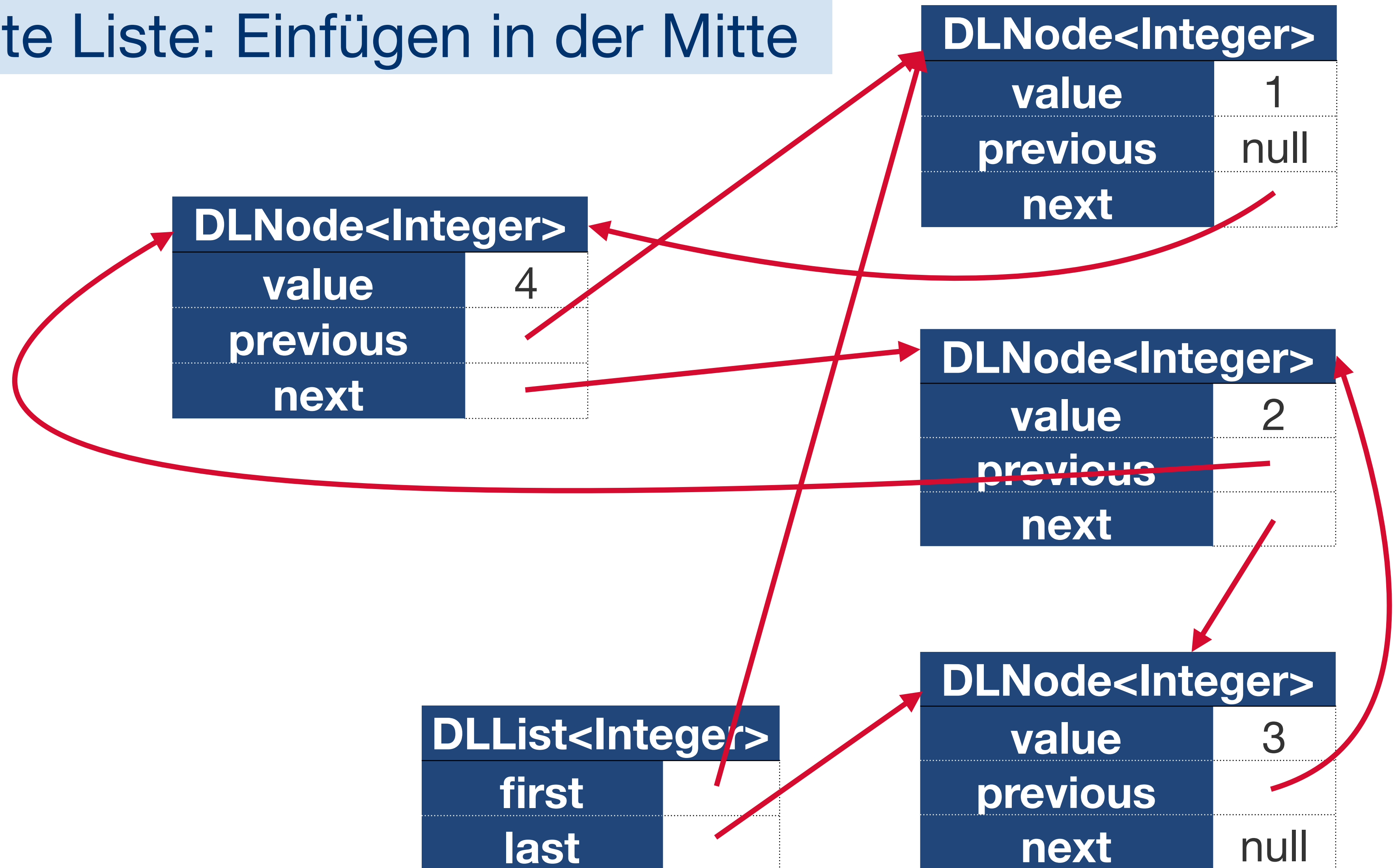
Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)

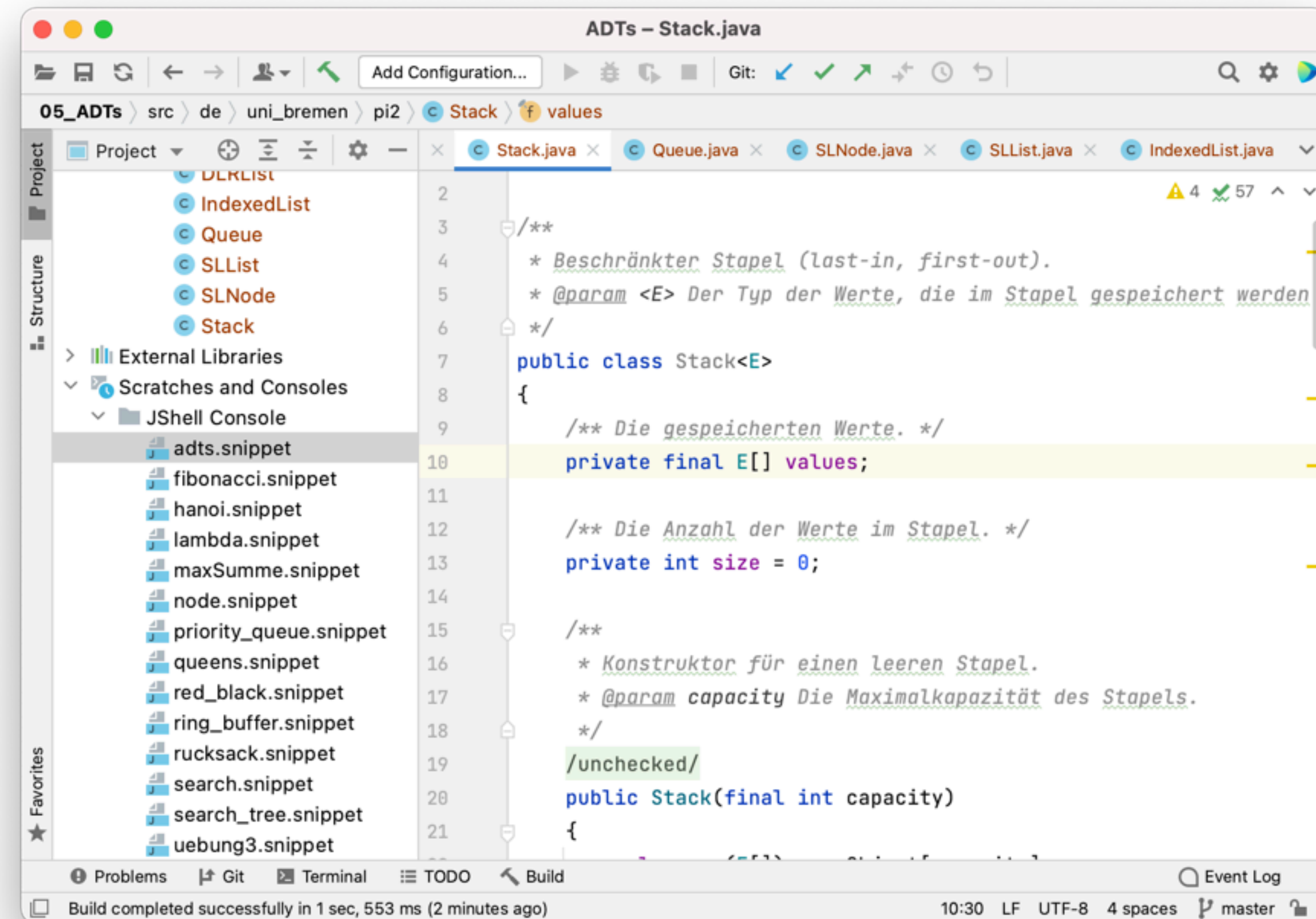


Doppelt verkettete Liste: Einfügen in der Mitte

insert(4, first.next)



Einfügen in doppelt verkettete Liste: Demo



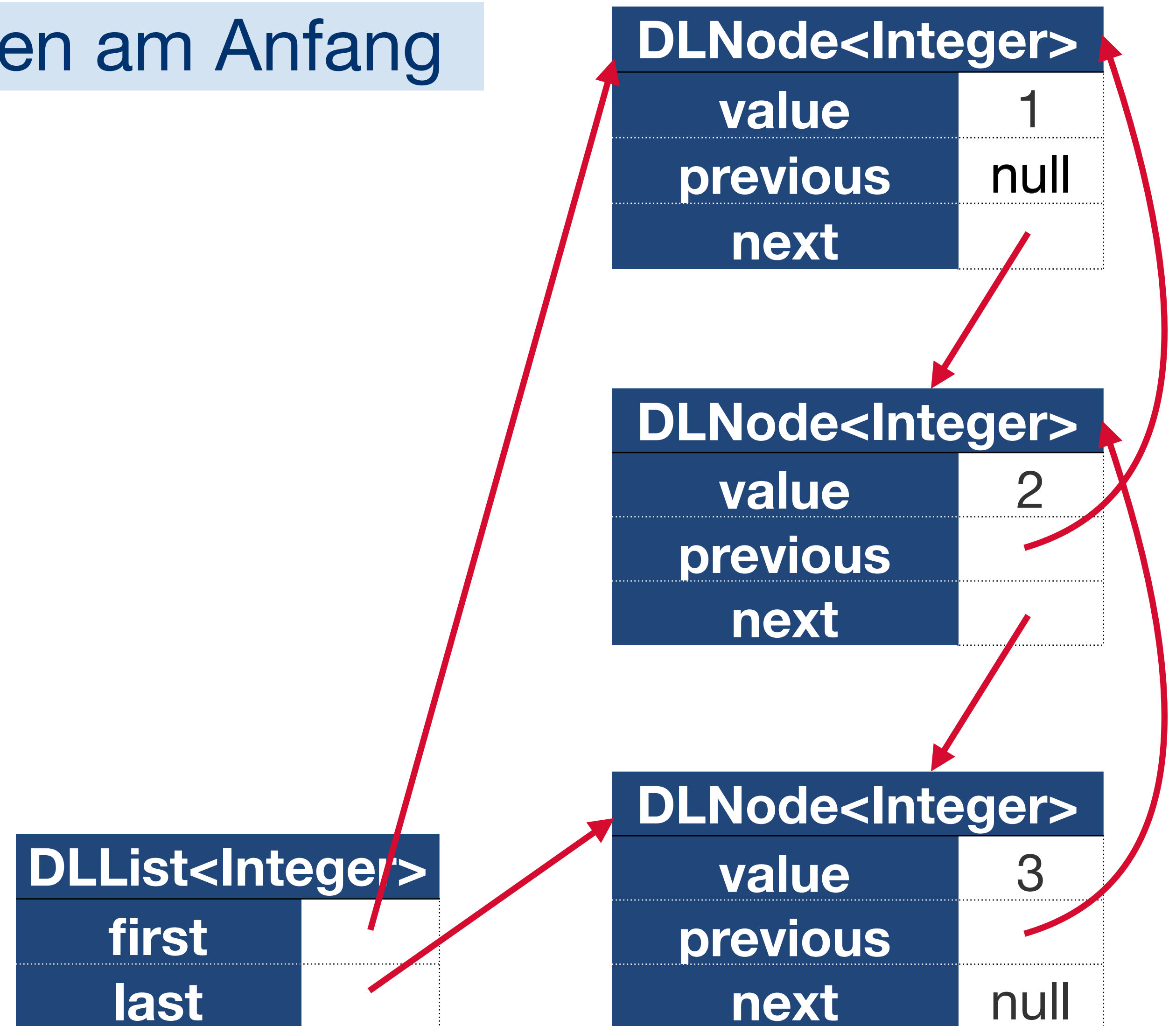
The screenshot shows an IDE window titled "ADTs - Stack.java". The main editor displays the following Java code:

```
2
3
4  /**
5   * Beschränkter Stapel (last-in, first-out).
6   * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
7   */
8  public class Stack<E>
9  {
10     /** Die gespeicherten Werte. */
11     private final E[] values;
12
13     /** Die Anzahl der Werte im Stapel. */
14     private int size = 0;
15
16     /**
17     * Konstruktor für einen leeren Stapel.
18     * @param capacity Die Maximalkapazität des Stapels.
19     */
20     /unchecked/
21     public Stack(final int capacity)
22     {
```

The IDE interface includes a Project Explorer on the left showing a project structure with folders like "DLRList", "IndexedList", "Queue", "SLList", "SLNode", and "Stack". The bottom status bar indicates "Build completed successfully in 1 sec, 553 ms (2 minutes ago)" and shows the time "10:30" and encoding "UTF-8".

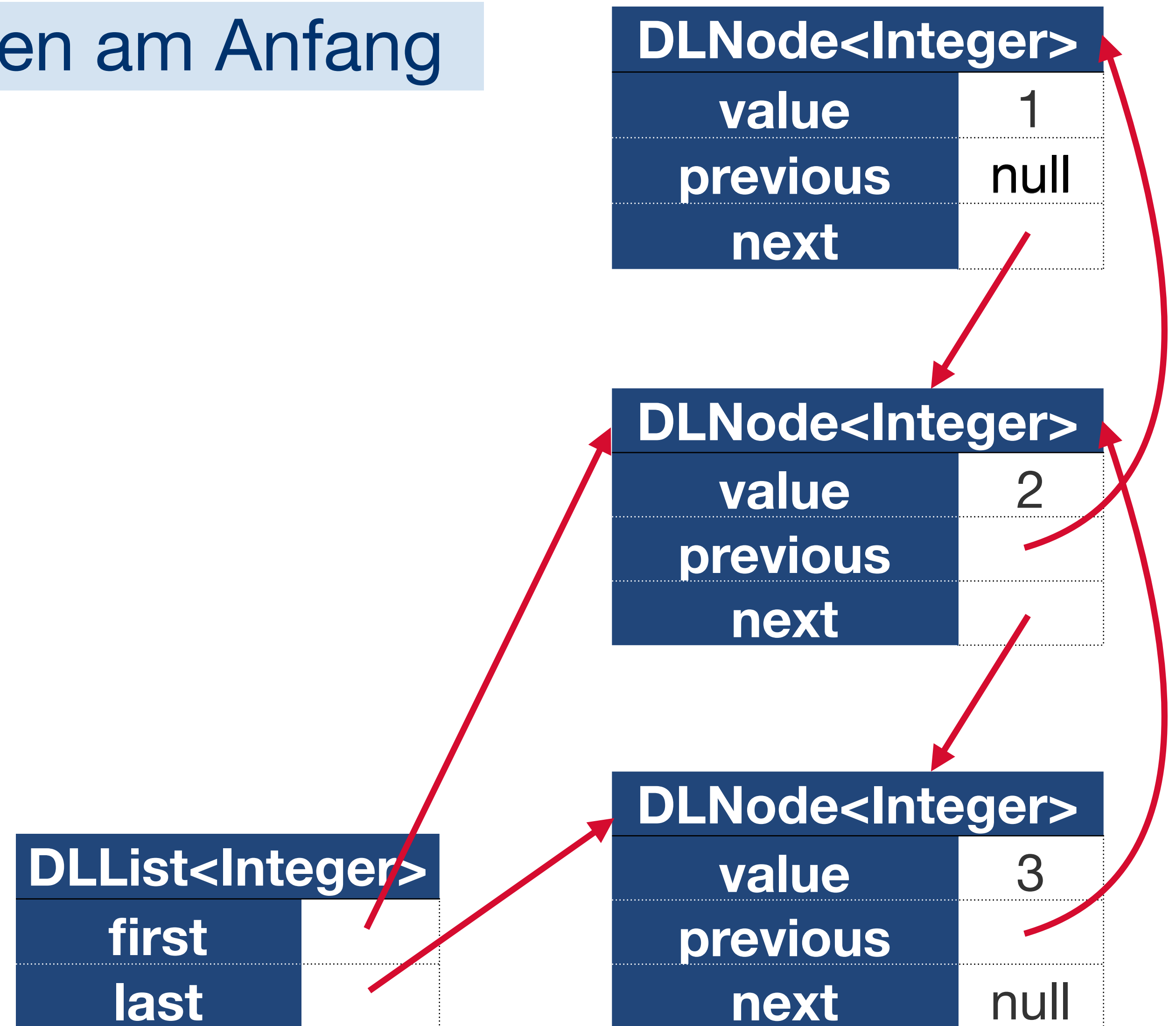
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



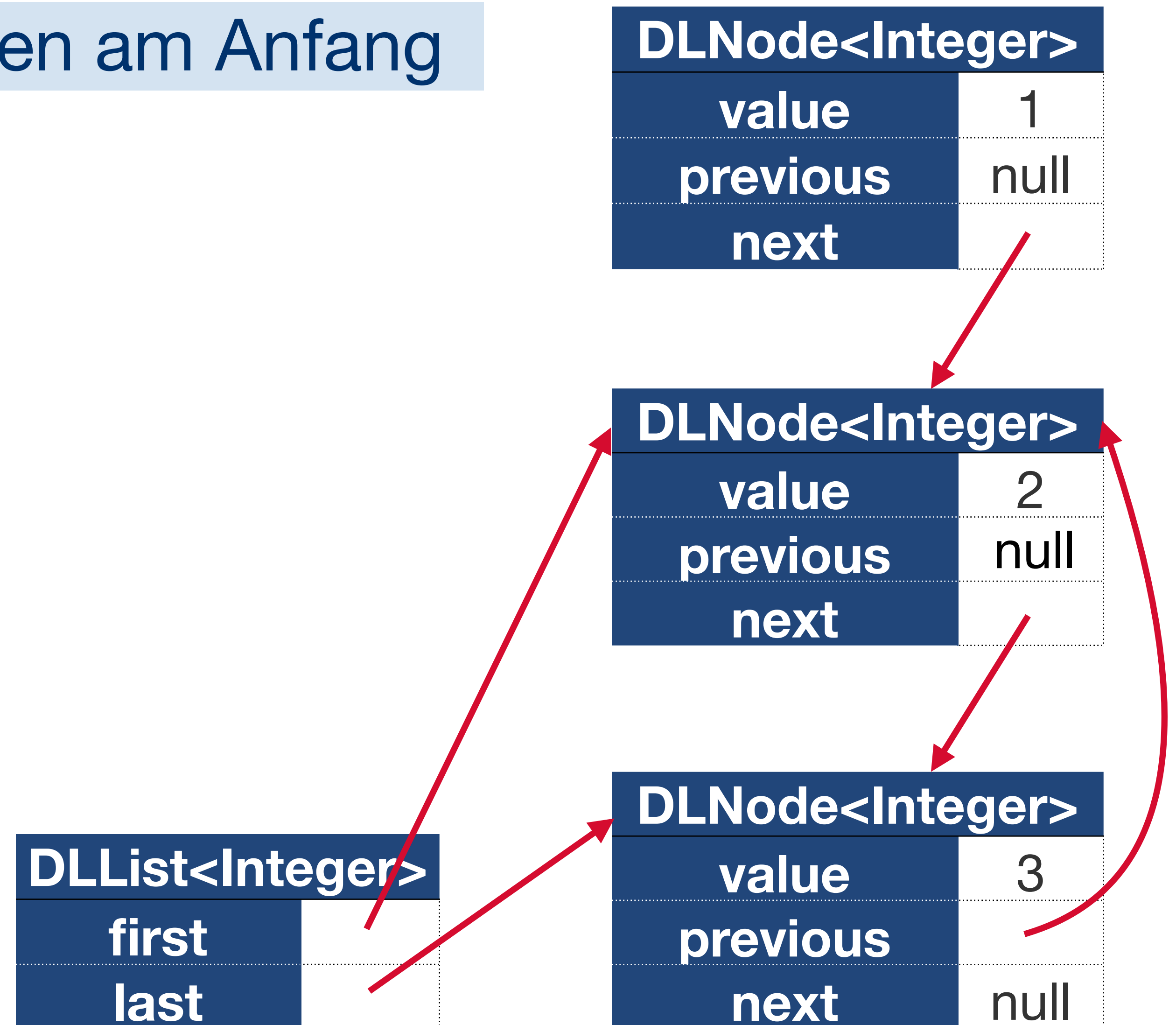
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



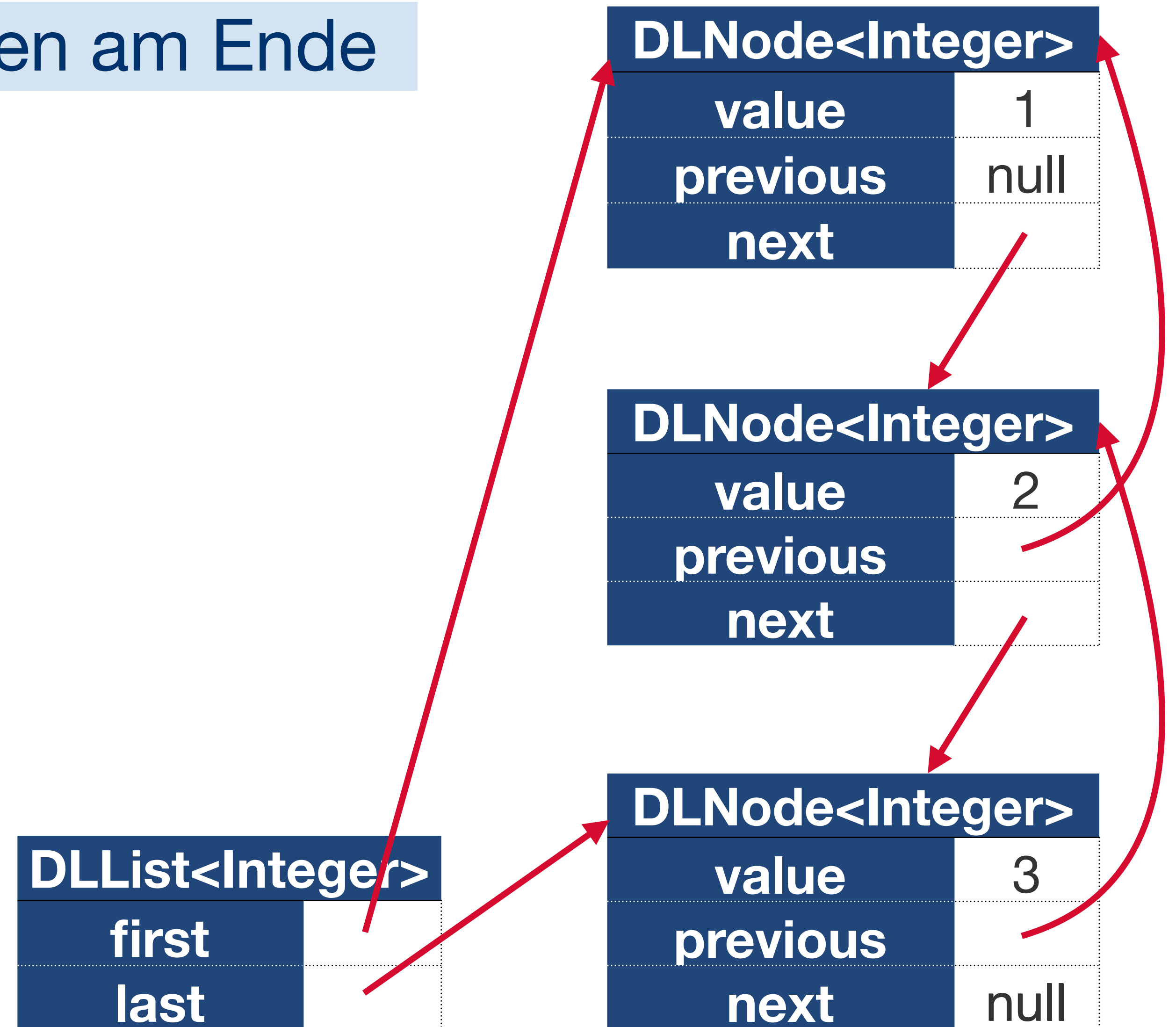
Doppelt verkettete Liste: Löschen am Anfang

remove(first)



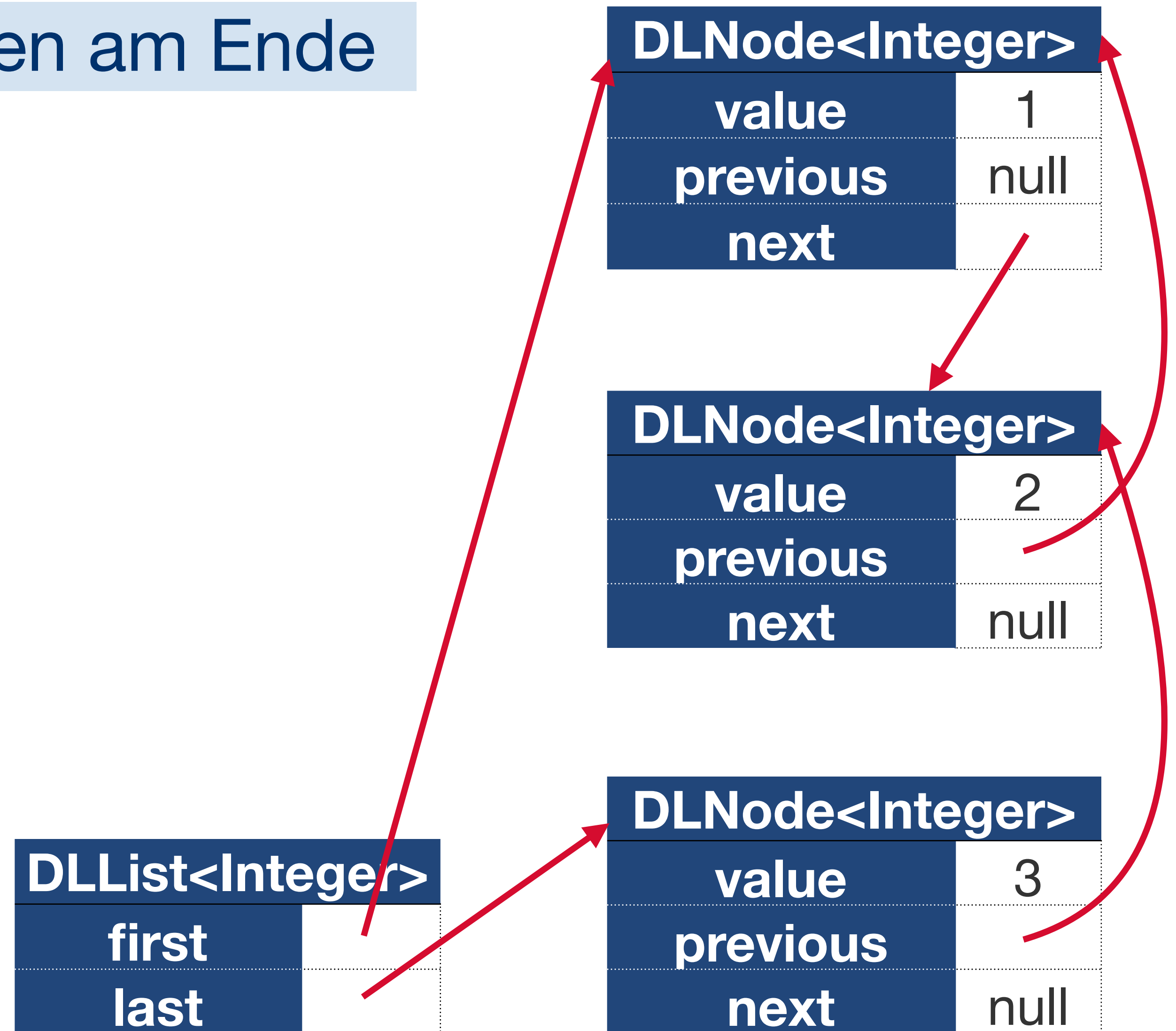
Doppelt verkettete Liste: Löschen am Ende

remove(last)



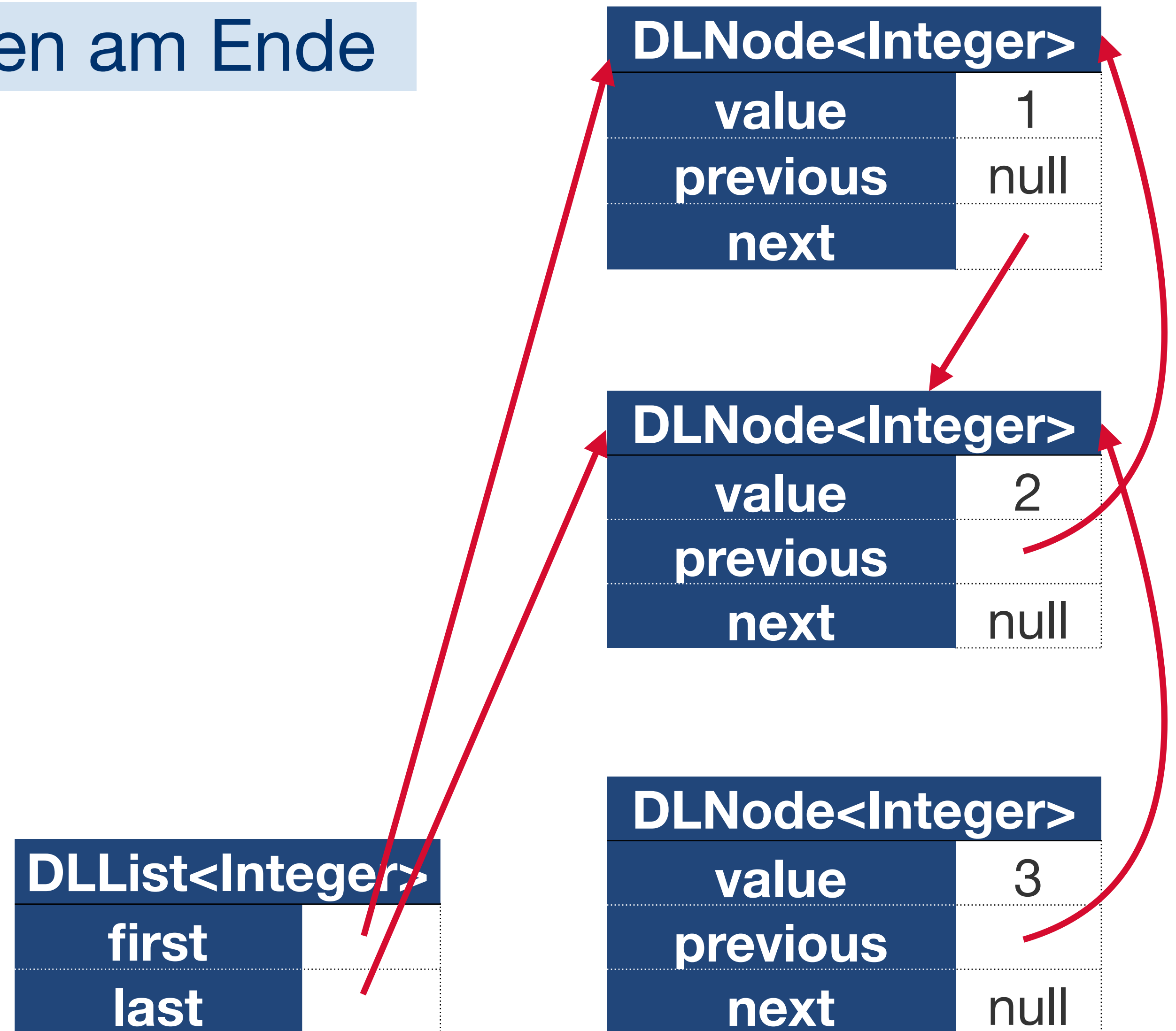
Doppelt verkettete Liste: Löschen am Ende

remove(last)



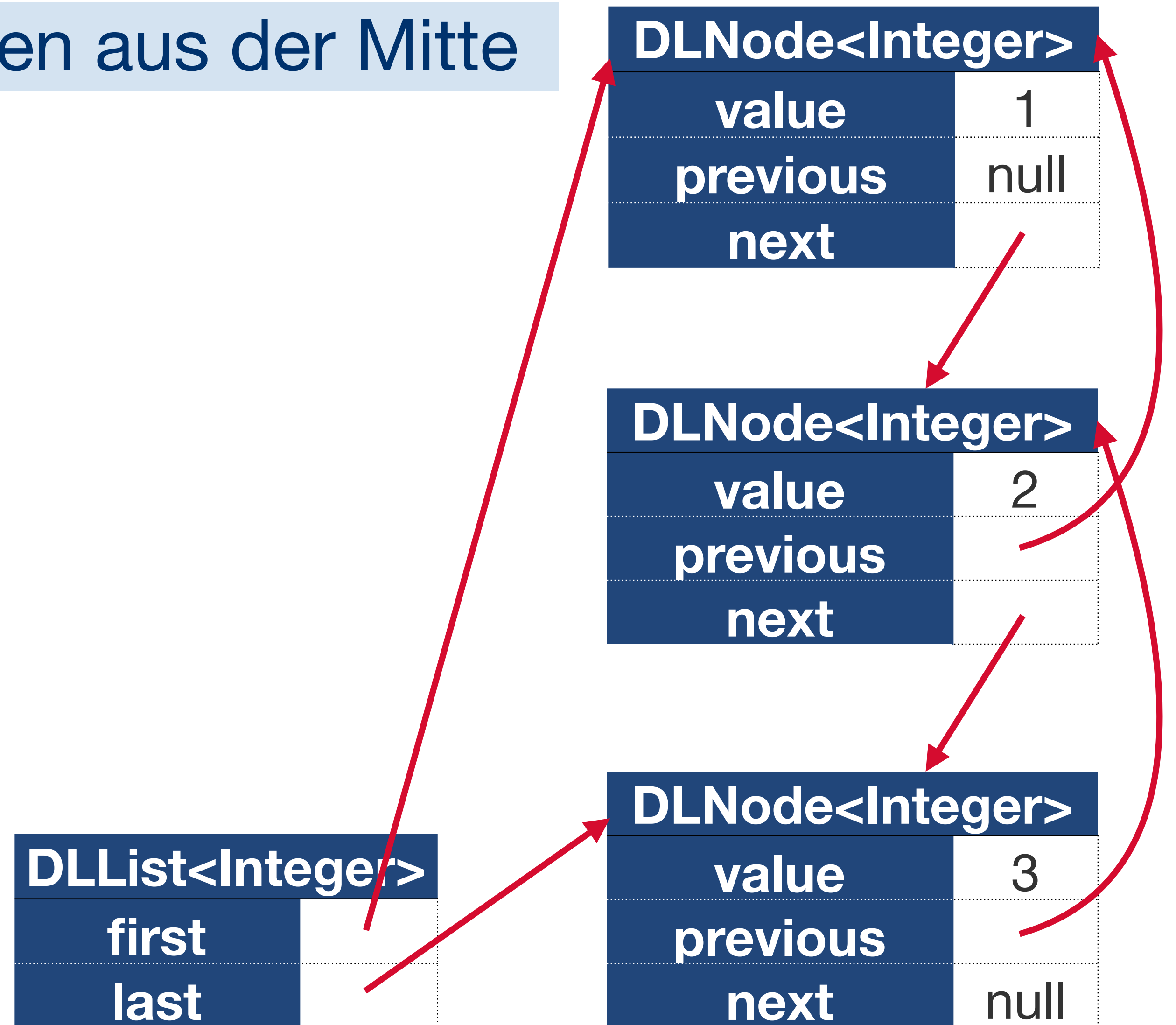
Doppelt verkettete Liste: Löschen am Ende

remove(last)



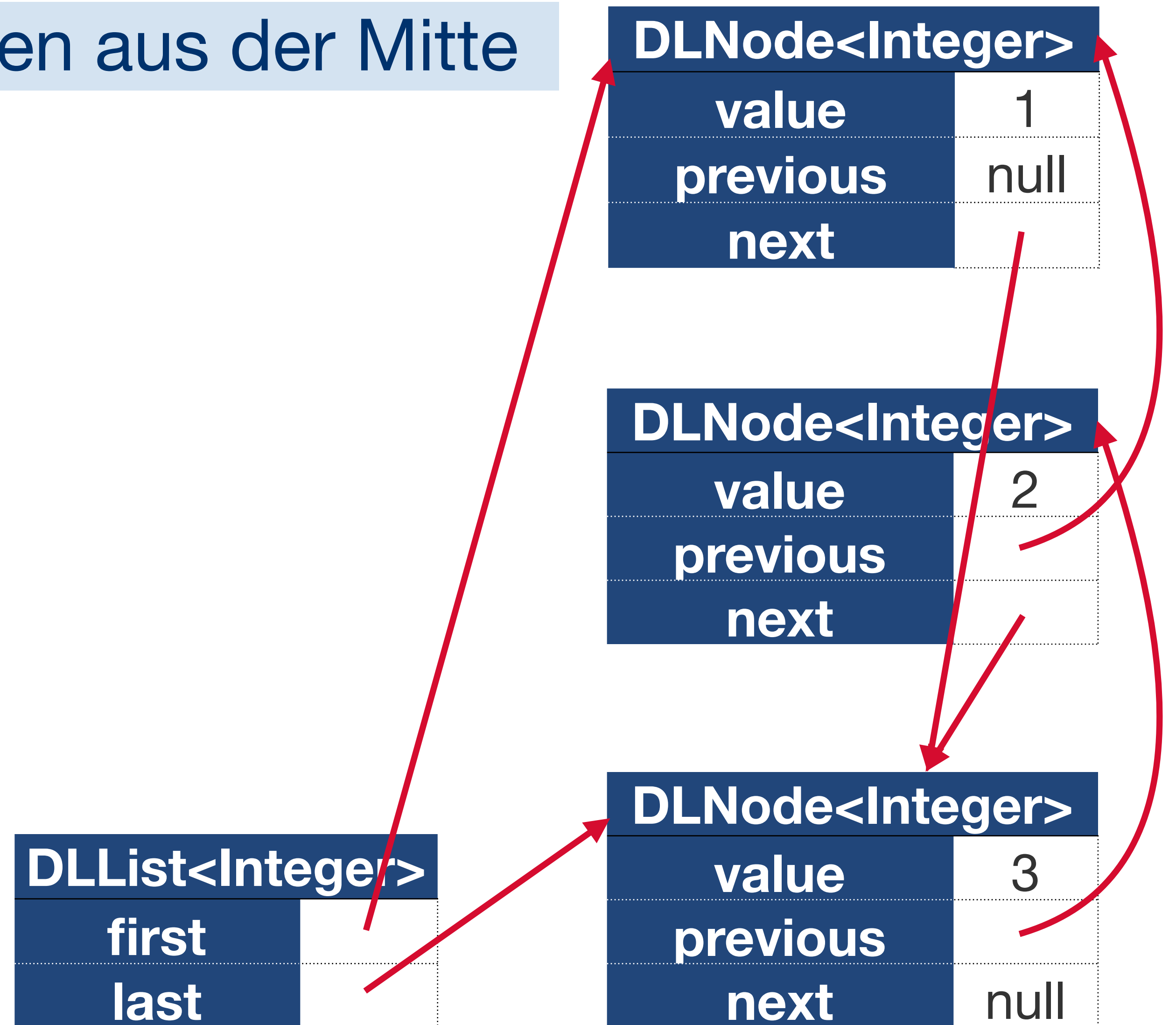
Doppelt verkettete Liste: Löschen aus der Mitte

`remove(first.next)`



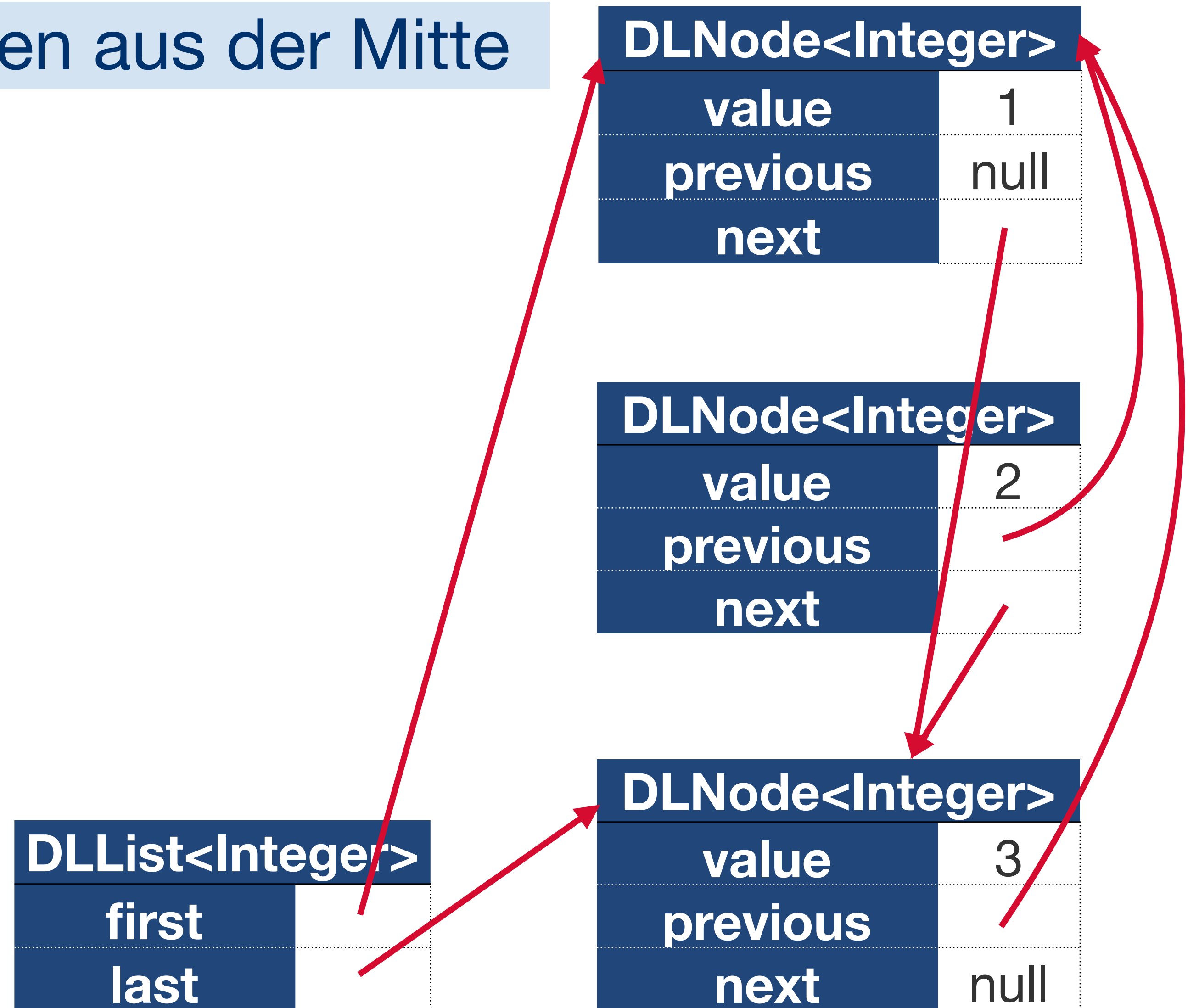
Doppelt verkettete Liste: Löschen aus der Mitte

`remove(first.next)`

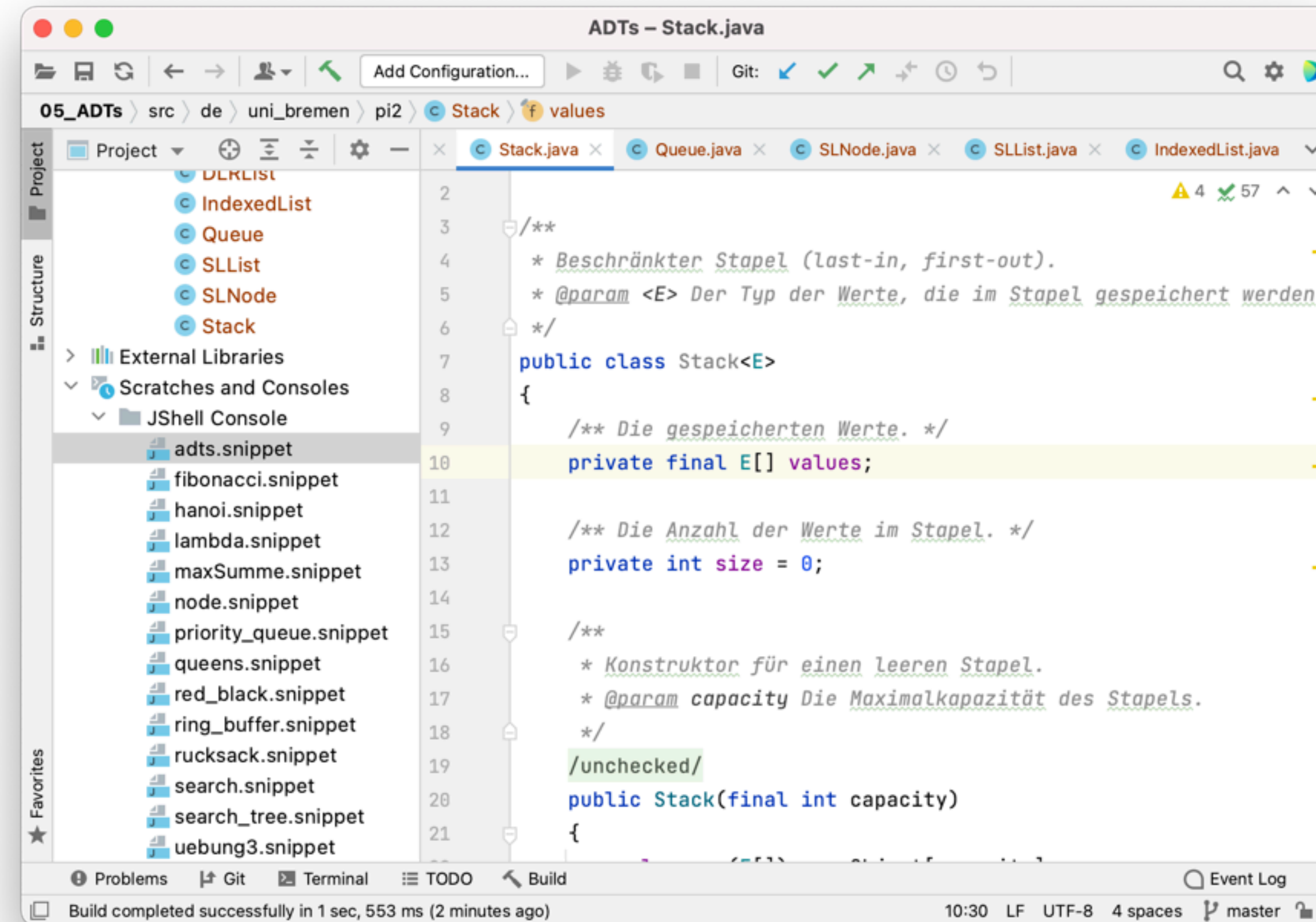


Doppelt verkettete Liste: Löschen aus der Mitte

`remove(first.next)`



Löschen aus doppelt verketteter Liste: Demo



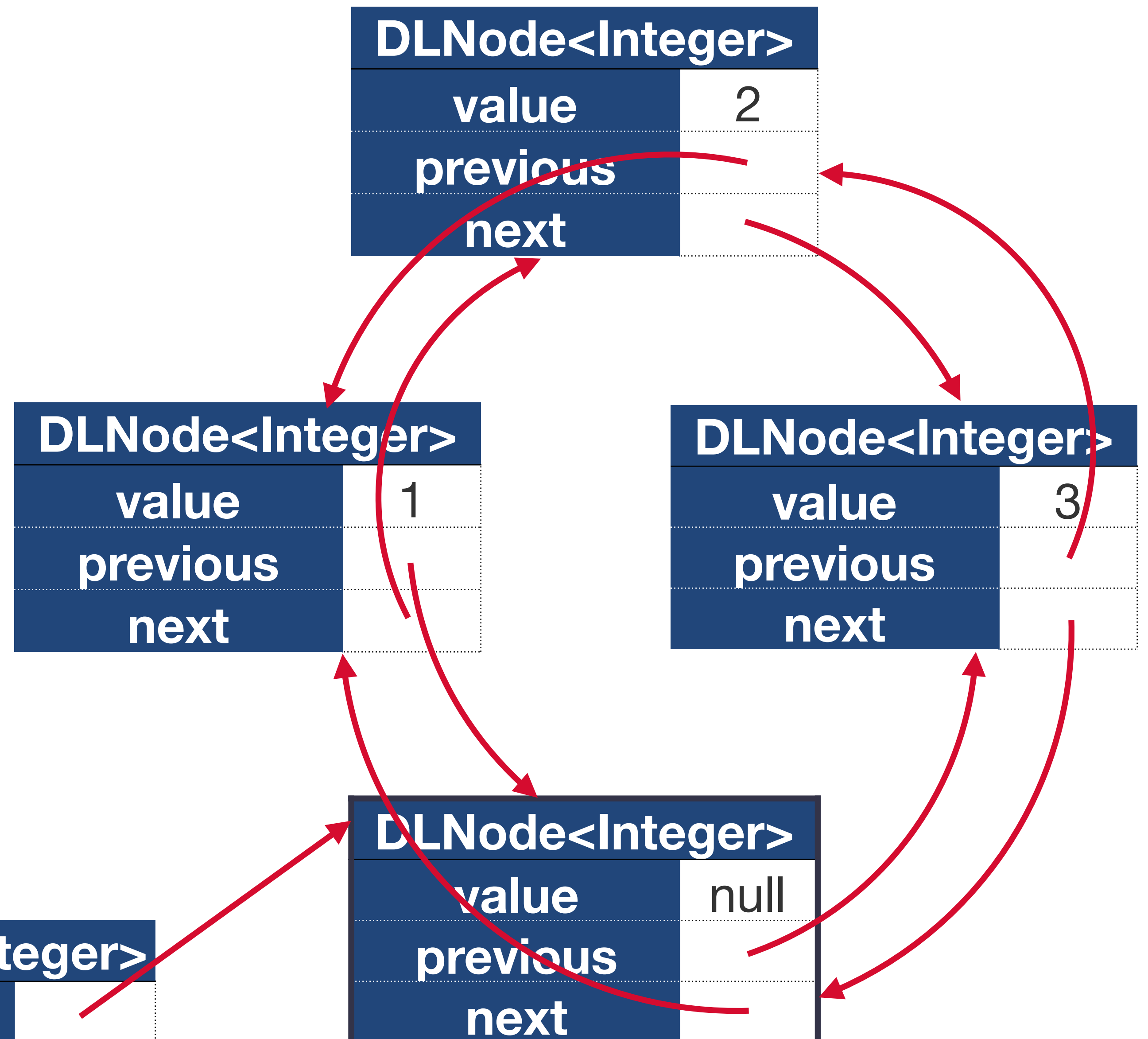
The screenshot shows an IDE window titled "ADTs - Stack.java". The main editor displays the following code:

```
2
3
4  /**
5   * Beschränkter Stapel (last-in, first-out).
6   * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
7   */
8  public class Stack<E>
9  {
10     /** Die gespeicherten Werte. */
11     private final E[] values;
12
13     /** Die Anzahl der Werte im Stapel. */
14     private int size = 0;
15
16     /**
17     * Konstruktor für einen leeren Stapel.
18     * @param capacity Die Maximalkapazität des Stapels.
19     */
20     /unchecked/
21     public Stack(final int capacity)
22     {
```

The line `private final E[] values;` is highlighted in yellow. The left sidebar shows a project structure with folders for "DLRList", "IndexedList", "Queue", "SLList", "SLNode", and "Stack". The bottom status bar indicates "Build completed successfully in 1 sec, 553 ms (2 minutes ago)".

Doppelt verkettete Ringliste

- Doppelt verkettete Liste hat Sonderfälle am Anfang und Ende der Liste
- Ringliste: leerer, zusätzlicher Knoten gleichzeitig als Anfang und Ende und Ende
- Jeder Knoten hat Vorgänger und Nachfolger



Doppelt verkettete Ringliste: Demo

```
ADTs - Stack.java
Add Configuration...
05_ADTs > src > de > uni_bremen > pi2 > Stack > values
Stack.java Queue.java SLNode.java SLList.java IndexedList.java
DLRList
IndexedList
Queue
SLList
SLNode
Stack
External Libraries
Scratches and Consoles
JShell Console
adts.snippet
fibonacci.snippet
hanoi.snippet
lambda.snippet
maxSumme.snippet
node.snippet
priority_queue.snippet
queens.snippet
red_black.snippet
ring_buffer.snippet
rucksack.snippet
search.snippet
search_tree.snippet
uebung3.snippet
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
/**
 * Beschränkter Stapel (last-in, first-out).
 * @param <E> Der Typ der Werte, die im Stapel gespeichert werden
 */
public class Stack<E>
{
    /** Die gespeicherten Werte. */
    private final E[] values;

    /** Die Anzahl der Werte im Stapel. */
    private int size = 0;

    /**
     * Konstruktor für einen leeren Stapel.
     * @param capacity Die Maximalkapazität des Stapels.
     */
    /unchecked/
    public Stack(final int capacity)
    {
```

Problems Git Terminal TODO Build Event Log
Build completed successfully in 1 sec, 553 ms (2 minutes ago) 10:30 LF UTF-8 4 spaces master

Zusammenfassung der Konzepte

- **Abstrakter Datentyp**
- **Stapel** und **Warteschlange** (**beschränkt/unbeschränkt**)
- **Einfach** und **doppelt verkettete (Ring-)Liste**
- **Liste mit Index**