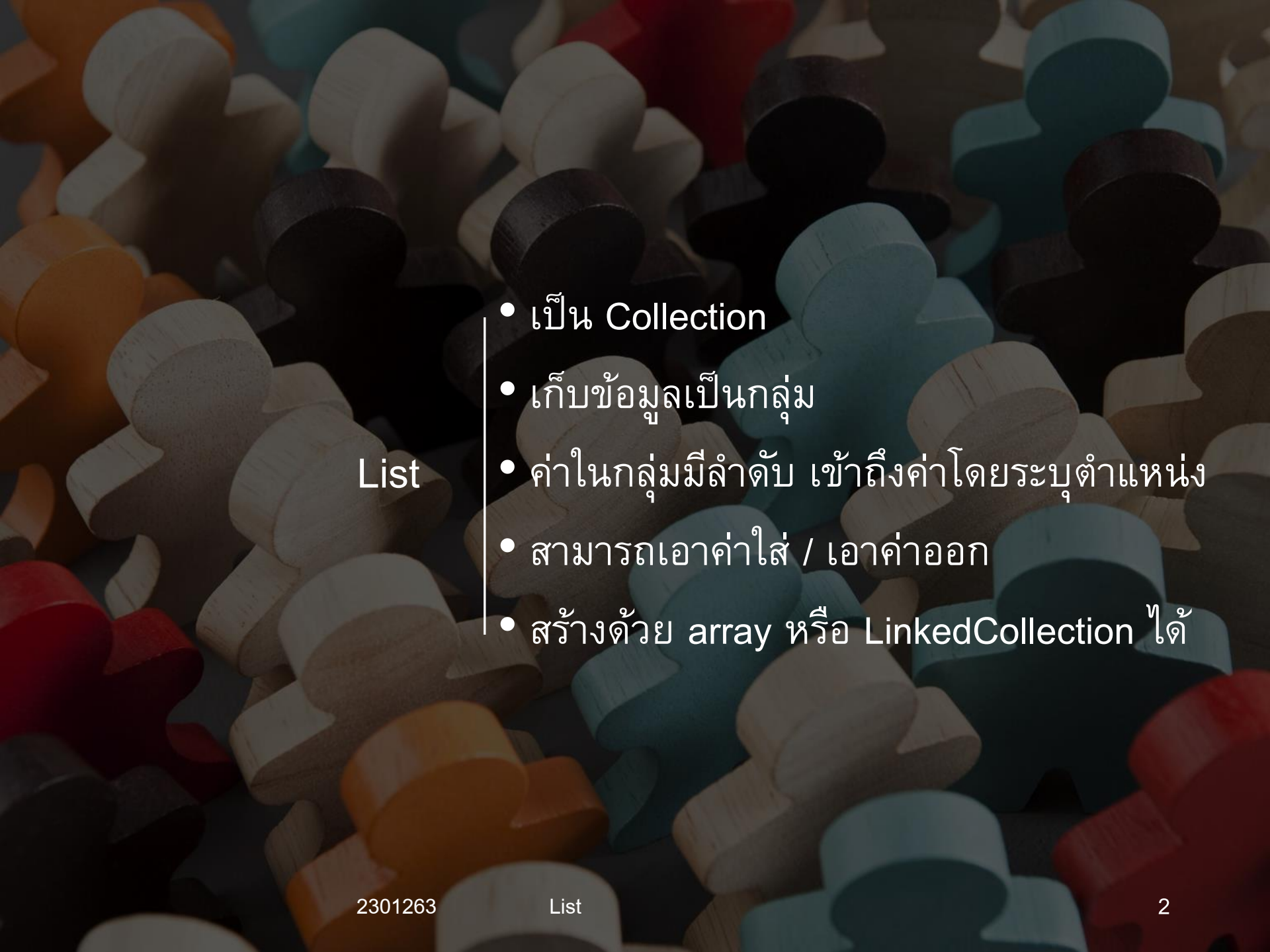
List

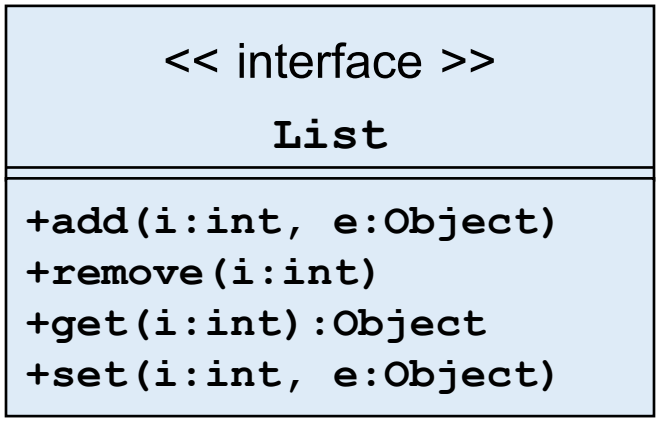
รายการ



List

- เป็น Collection
- เก็บข้อมูลเป็นกลุ่ม
- ค่าในกลุ่มมีลำดับ เข้าถึงค่าโดยระบุตำแหน่ง
- สามารถเอาค่าใส่ / เอาค่าออก
- สร้างด้วย array หรือ LinkedCollection ได้

interface



`void add(i:int, e:Object)`

เพิ่ม `e` ในลิสต์ที่ตำแหน่ง `i`

`void remove(i:int)`

ลบค่าที่ตำแหน่ง `i` ของลิสต์

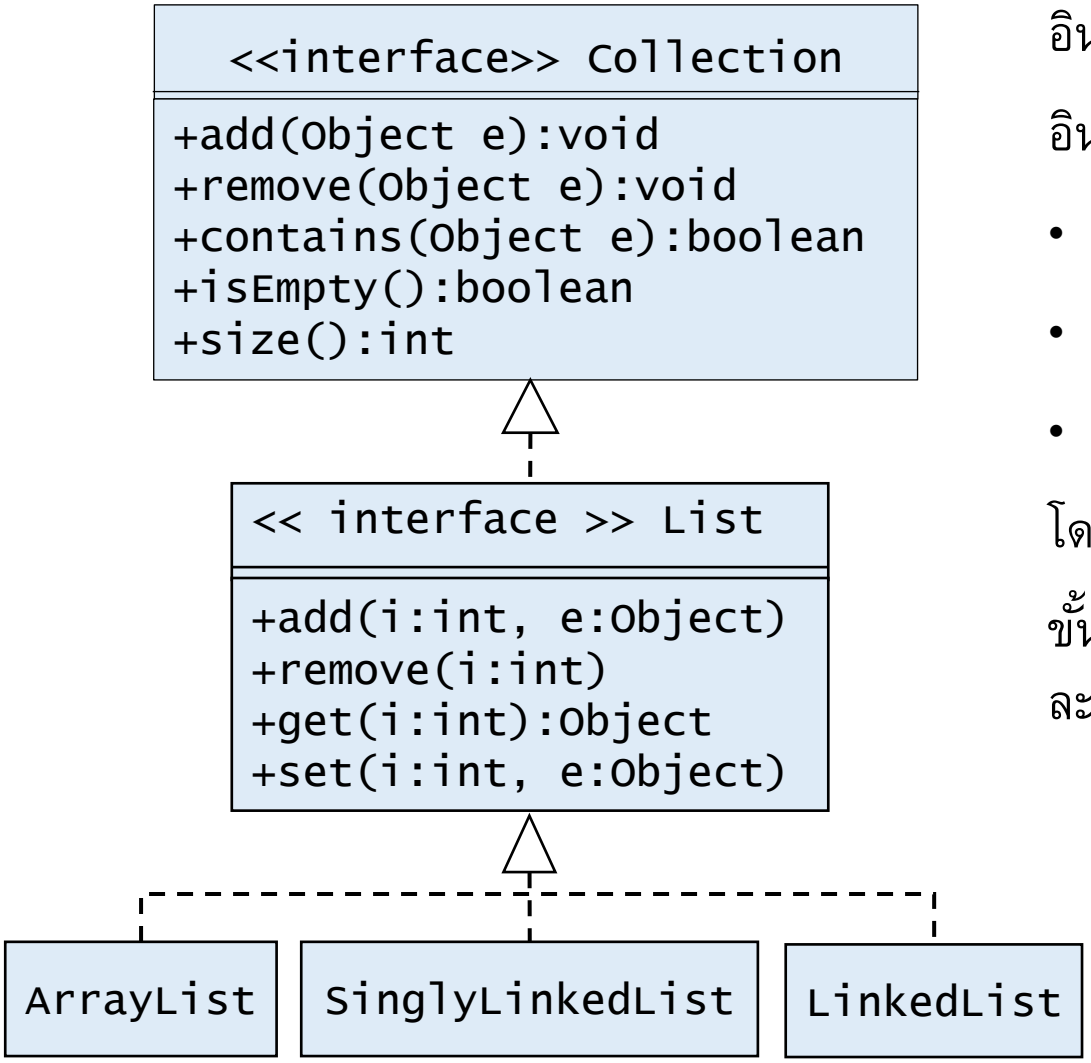
`Object get(i:int)`

คืนค่าที่ตำแหน่ง `i` ของลิสต์

`void set(i:int, e:Object)`

แก้ค่าที่ตำแหน่ง `i` ของลิสต์เป็น `e`

interface



อินเทอร์เฟส `List` เป็น `Collection`

อินเทอร์เฟส `List` ใช้สำหรับ

- คลาส `ArrayList`
- คลาส `SinglyLinkedList` และ
- คลาส `LinkedList`

โดยเก็บค่าในแต่ละคลาสต่างกัน

ขั้นตอนการทำงานของเมท็อดเดียวกันในแต่ละคลาสต่างกัน

Interface List

```
public interface List extends Collection{
   // includes methods in Collection
   public void add(int i, Object e)  // also in Collection
   public void remove(int i)         // also in Collection
   public Object get(int i)
   public void set(int i, Object e)
}
```

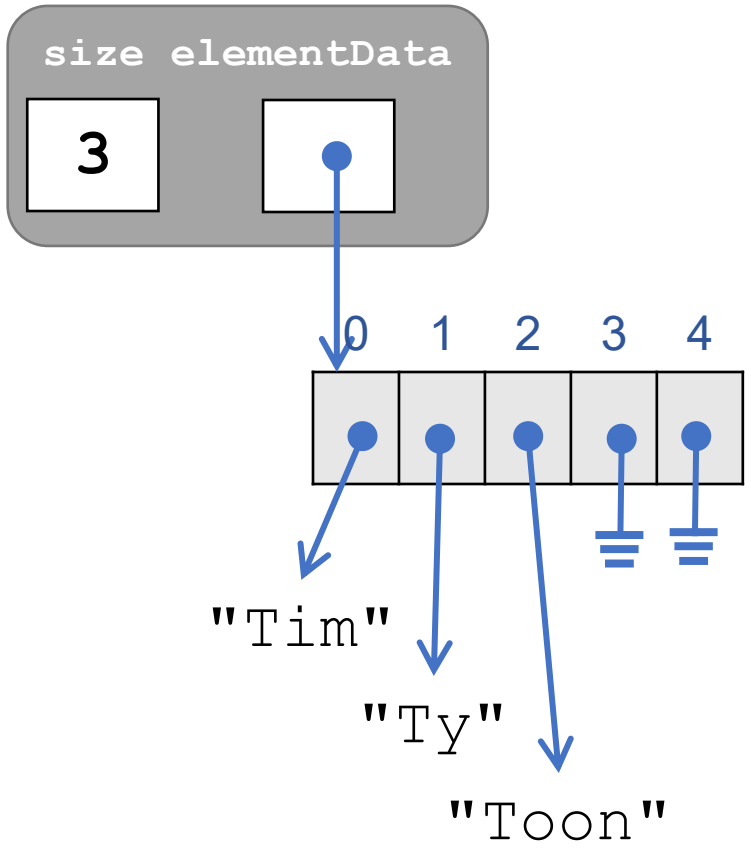
ArrayList

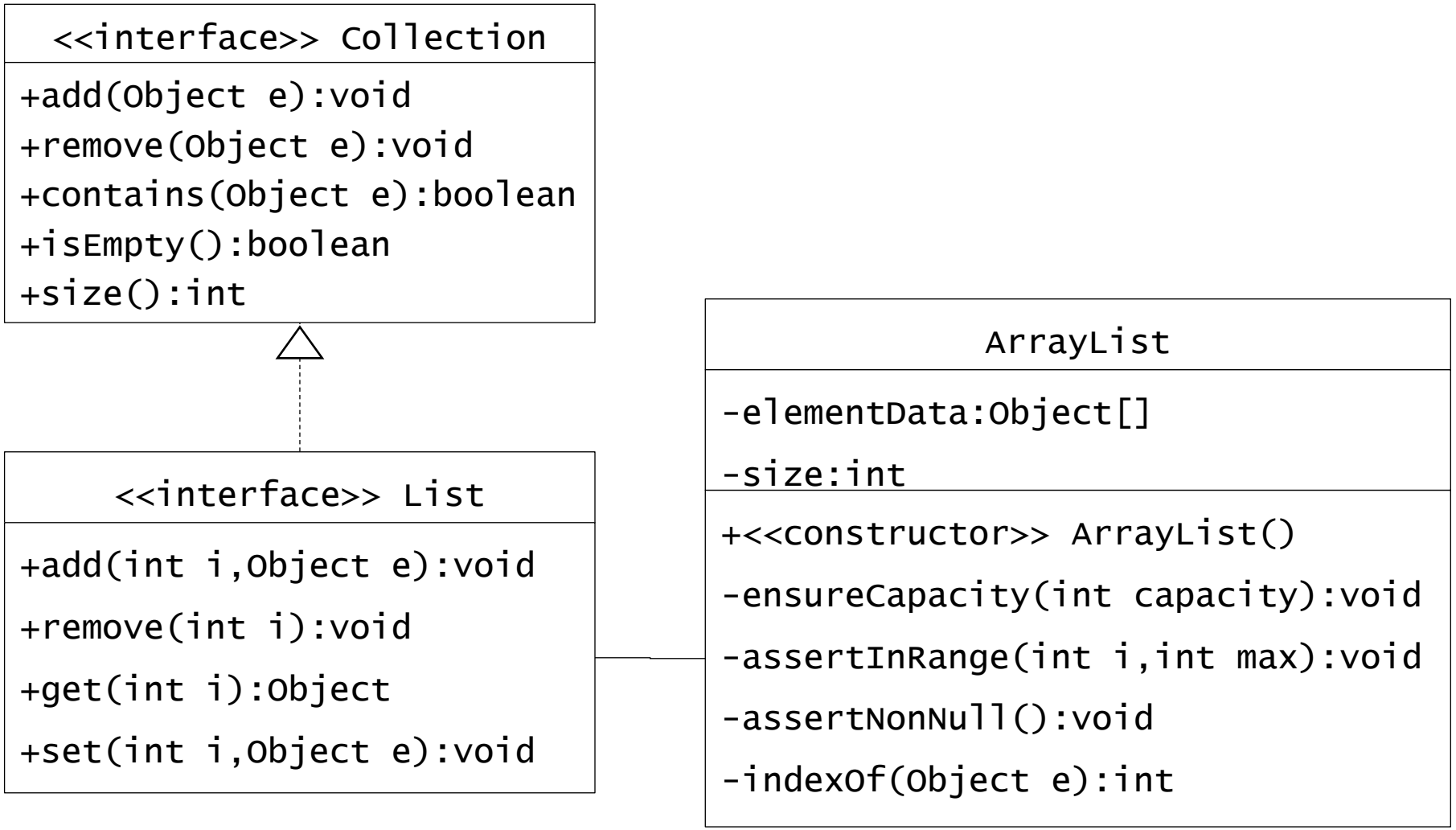
สร้าง List ด้วย Array

Class ArrayList

สร้าง List ด้วย array

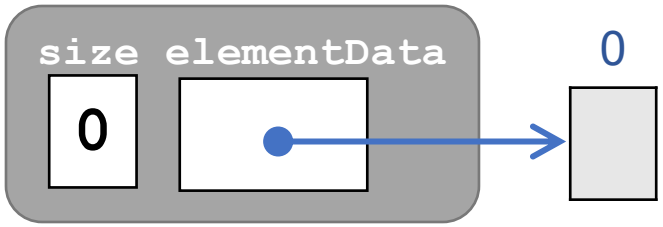
- Attribute **`elementData`** เก็บข้อมูลแต่ละตัวในชุด
  - เป็น array ของ Object
- Attribute **`size`** เก็บจำนวนข้อมูลในชุด
  - เป็น integer




<<interface>> Collection
+add(Object e):void
+remove(Object e):void
+contains(Object e):boolean
+isEmpty():boolean
+size():int
<<interface>> List
+add(int i,Object e):void
+remove(int i):void
+get(int i):Object
+set(int i,Object e):void
ArrayList
-elementData:Object[]
-size:int
+<<constructor>> ArrayList()
-ensureCapacity(int capacity):void
-assertInRange(int i,int max):void
-assertNonNull():void
-indexOf(Object e):int

Create new object: Class ArrayList

```
public class ArrayList implements List {
    private Object[] elementData;
    private int size;

    public ArrayList() {  // create an empty array list.
        elementData = new Object[1];
        size = 0;
    }
}
```



Methods in Class ArrayList

```
public class ArrayList implements List {
   private Object[] elementData;
   private int size;

   public ArrayList() {...}
   public int size() {...}                    // interface Collection
   public boolean isEmpty() {...}
   public boolean contains(Object e) {...}
   public void add(Object e) {...}
   public void remove(Object e) {...}
   public void add(int i, Object e) {...} // interface List
   public void remove(int i) {...}
   public Object get(int i) {...}
   public void set(int i, Object e) {...}
}
```

Some simple methods (Same as in ArrayCollection)

```
public int size() {
   return size;
}
public boolean isEmpty() {
   return size==0;
}
public boolean contains(Object e) {
   return indexOf(e)!= -1;
}
private int indexOf(Object e) {
   for (int i=0; i<size; i++)
      if (elementData[i].equals(e)) return i;
   return -1;
}
```

Method add

Class ArrayList

add: Class ArrayList

```
public void add(int i, Object e) { // add e at pos. i
   assertNonNull(e);          // is e not null ?
   assertInRange(i, size);    // is i valid position ?
   ensureCapacity(size+1);    // ensure enough space in array
   for (int j=size-1; j>=i; j--) // move data to make space
      elementData[j+1] = elementData[j];
   elementData[i] = e;        // put e in the array
   size++;
}

public void add(Object e) {   // add e as last element
   add(size, e);
}
```

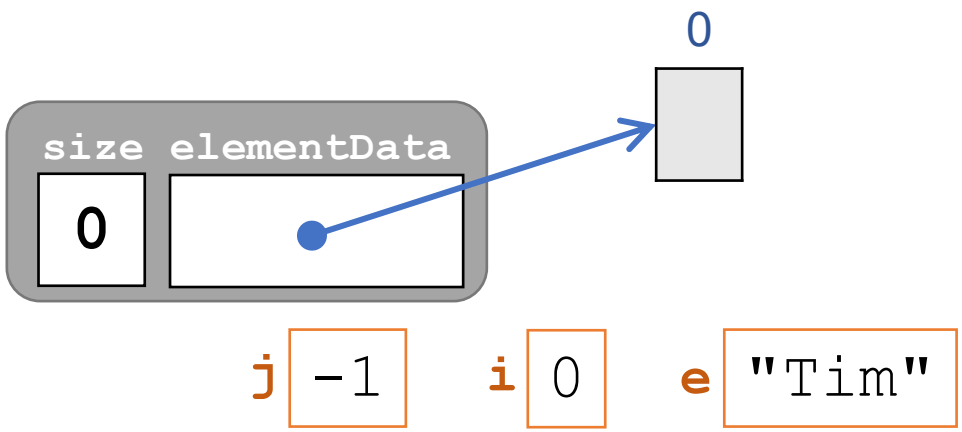
Add a value in an empty list

Example 1

Example 1: add in an empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)  // j<i , not get in loop
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++;
}
```

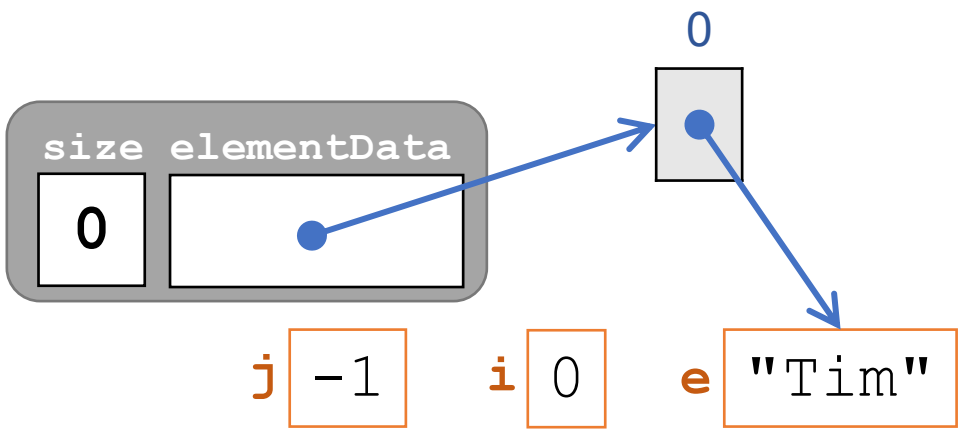
```
al = new ArrayList();
al.add(0,"Tim");
```



Example 1: add in an empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)
        elementData[j+1] = elementData[j];
    elementData[i] = e;     // link from array to e
    size++;
}
```

```
al = new ArrayList();
al.add(0,"Tim");
```

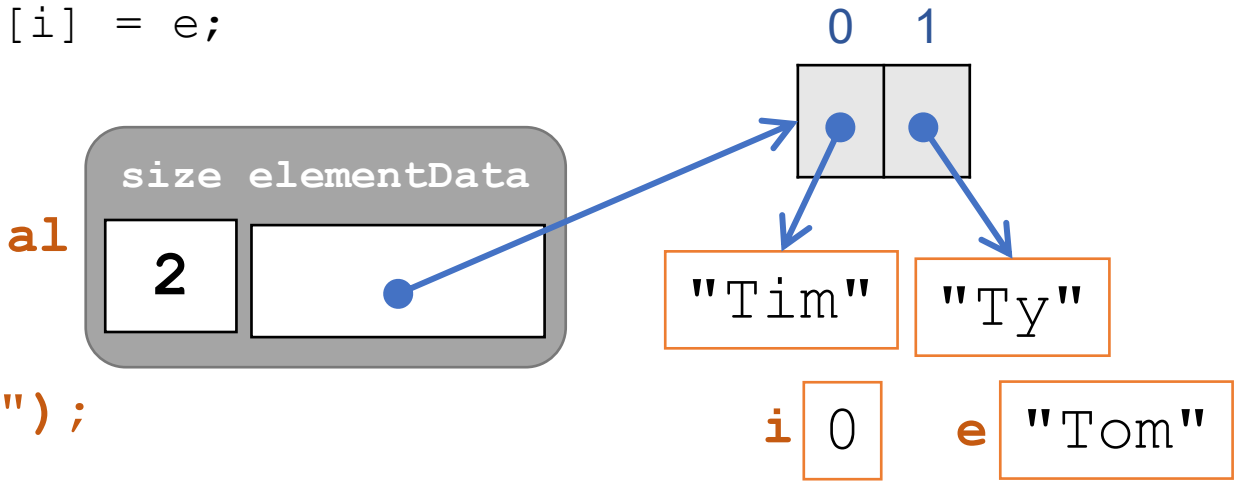


Add a value in a non-empty list

Example 2

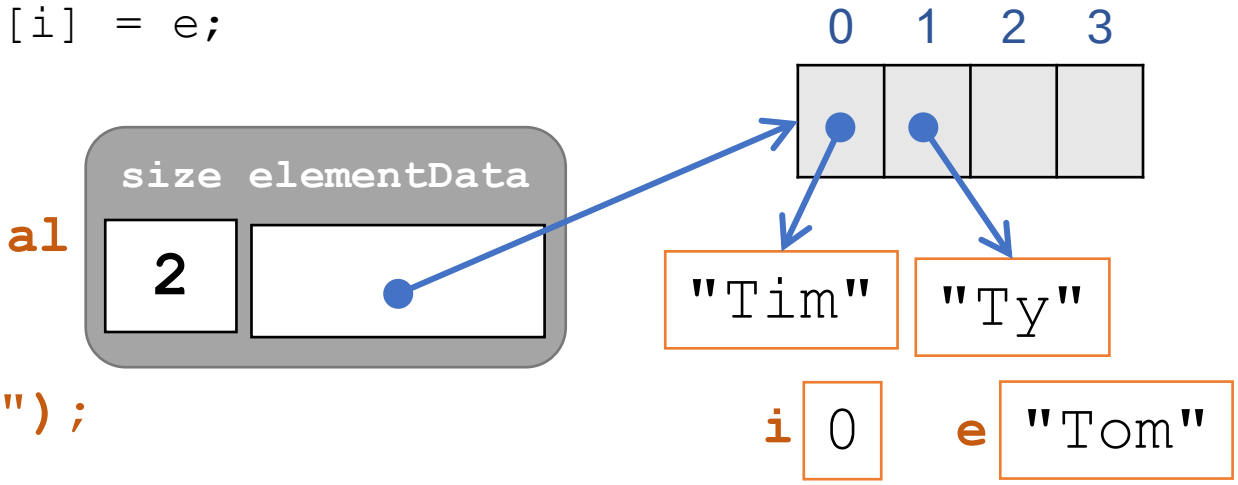
Example 2: add in a non-empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);       // make space
    for (int j=size-1; j>=i; j--)
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++;
}
```



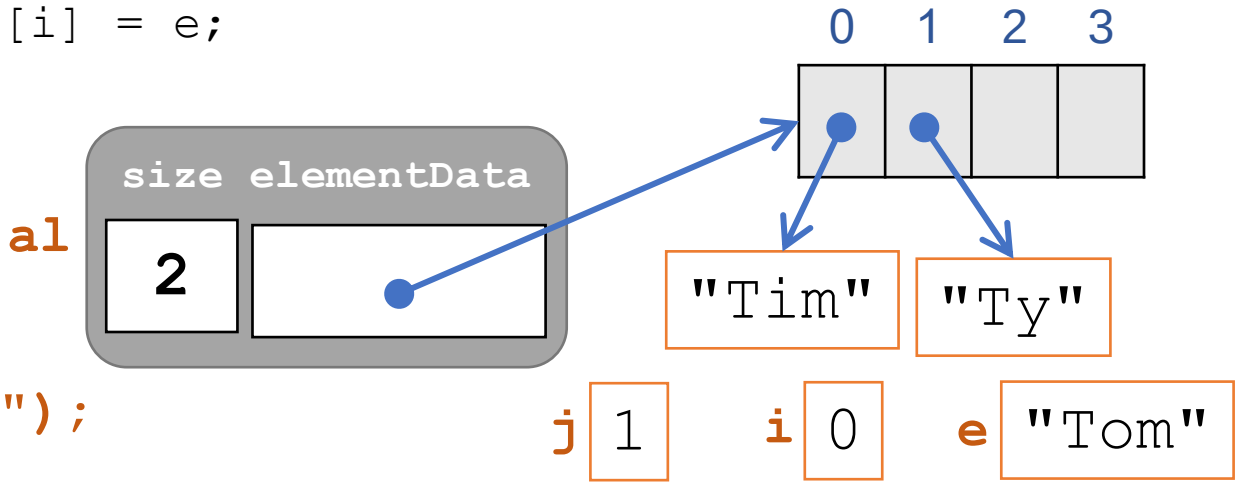
Example 2: add in a non-empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);       // make space
    for (int j=size-1; j>=i; j--)
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++;
}
```



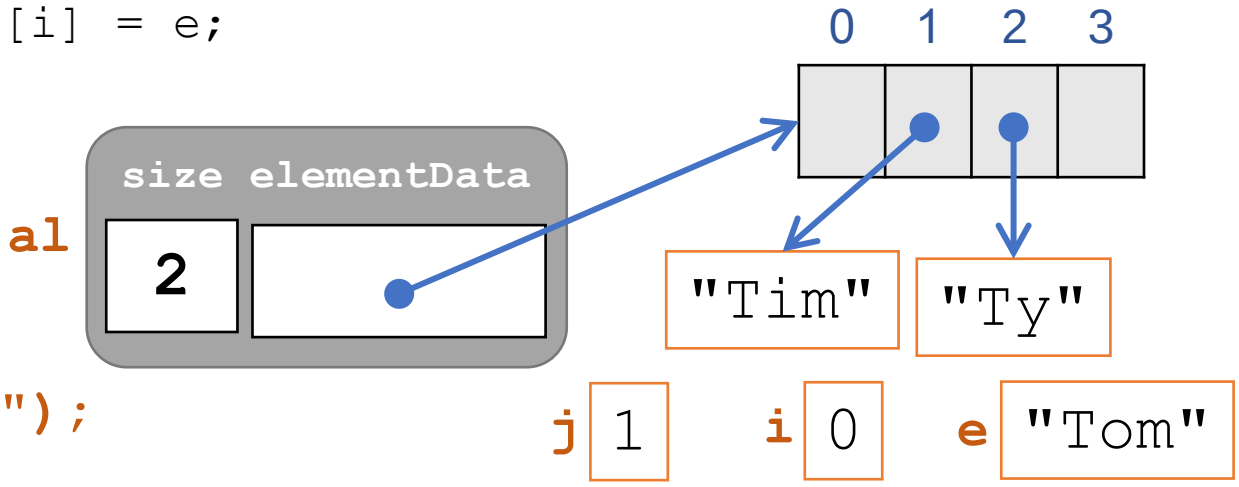
Example 2: add in a non-empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)  // shift data in array
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++;
}
```



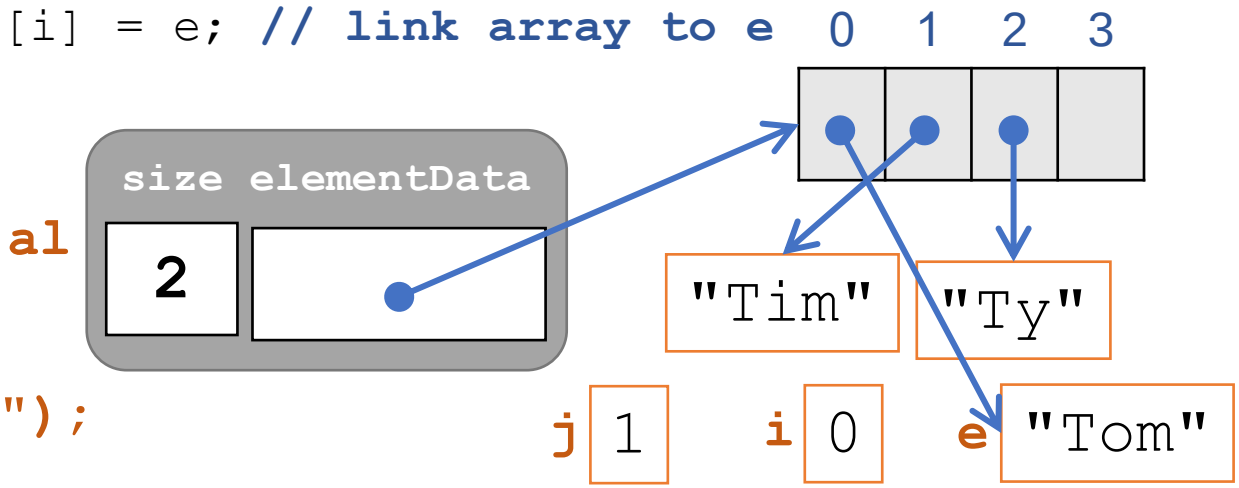
Example 2: add in a non-empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)  // shift data in array
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++;
}
```



Example 2: add in a non-empty list

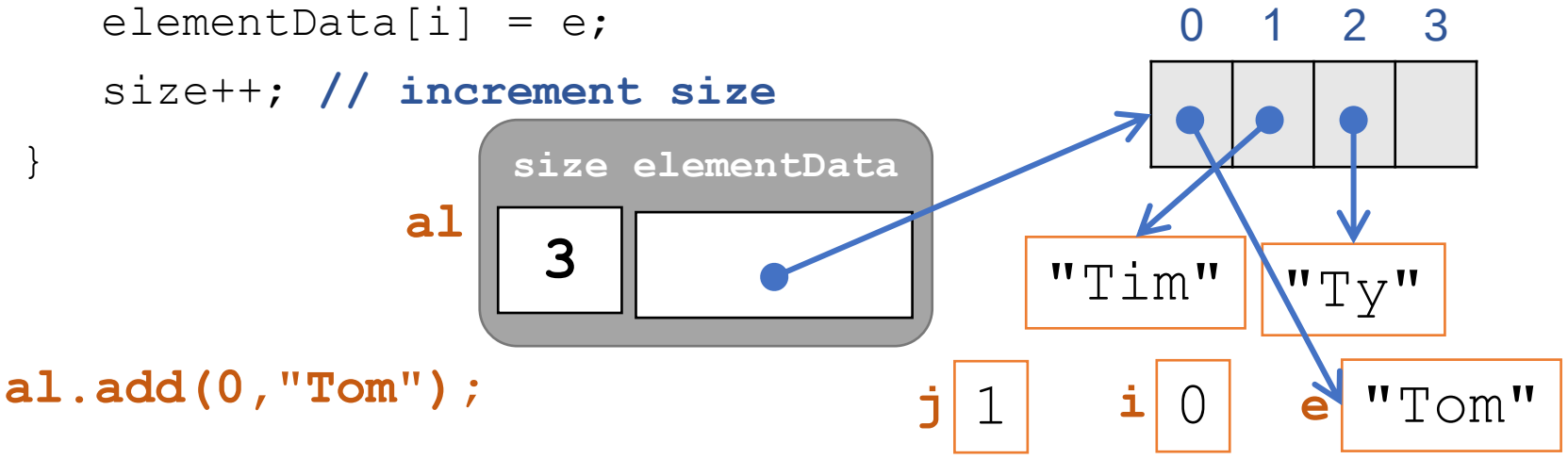
```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)
        elementData[j+1] = elementData[j];
    elementData[i] = e; // link array to e

    size++;
}
```




```
al.add(0,"Tom");
```

Example 2: add in a non-empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    ensureCapacity(size+1);
    for (int j=size-1; j>=i; j--)
        elementData[j+1] = elementData[j];
    elementData[i] = e;
    size++; // increment size
}
```



Method remove

Class ArrayList

remove: Class ArrayList

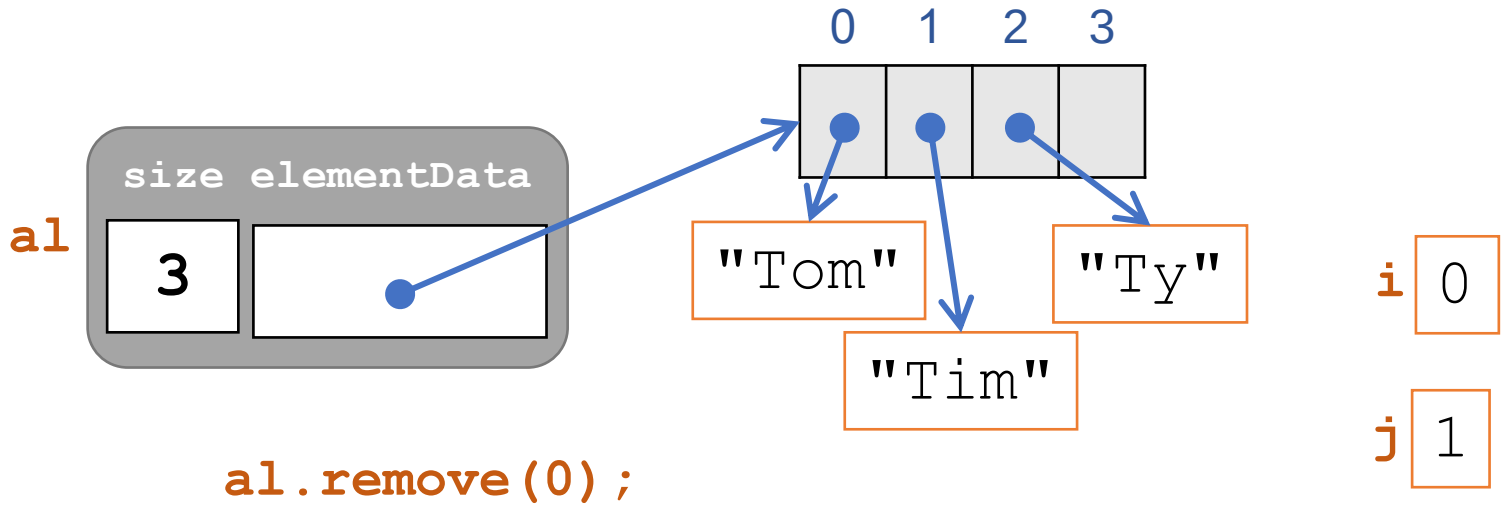
```
public void remove(int i) {
   assertInRange(i, size-1);  // is position i in the range?
   // move elements from position i+1, i+2,…, size-1
   // to the front 1 position
   for (int j=i+1; j<size; j++)
      elementData[j-1] = elementData[j];
   // decrement size & erase the last element
   elementData[--size] = null;
}
public void remove(Object e) {
   int i = indexOf(e);
   if (i>=0) remove(i);
}
```

Remove an element

Example

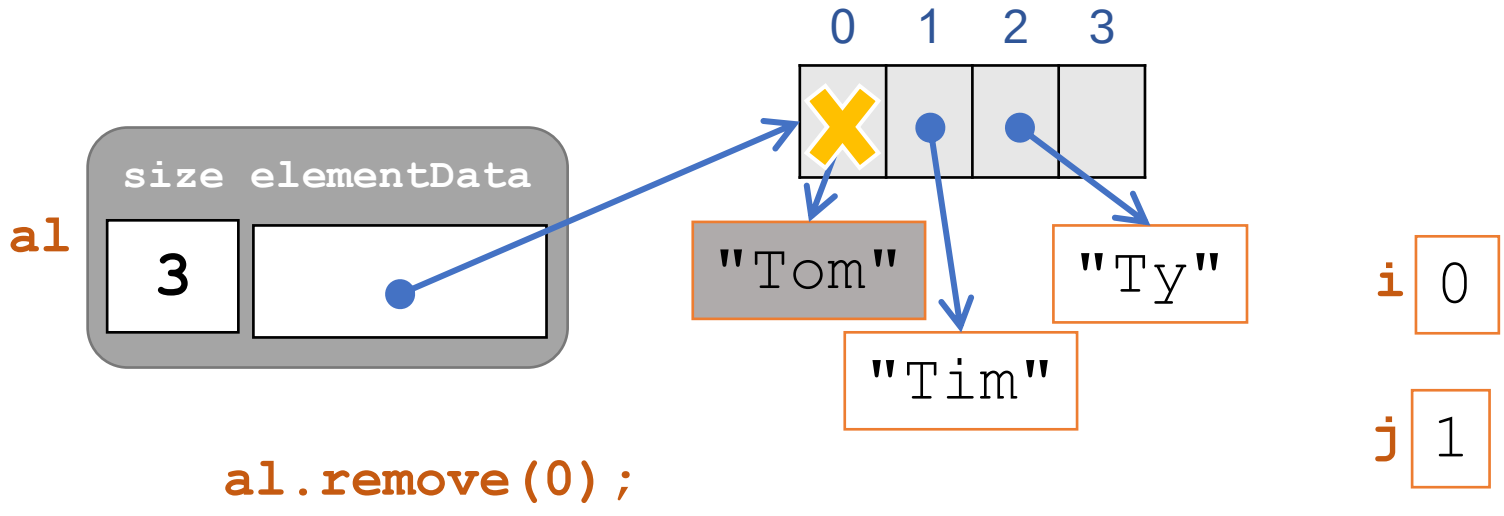
Example: remove an element

```
public void remove(int i) {
    assertInRange(i, size-1);
    for (int j=i+1; j<size; j++) // move data in array
        elementData[j-1] = elementData[j];
    elementData[--size] = null;
}
```



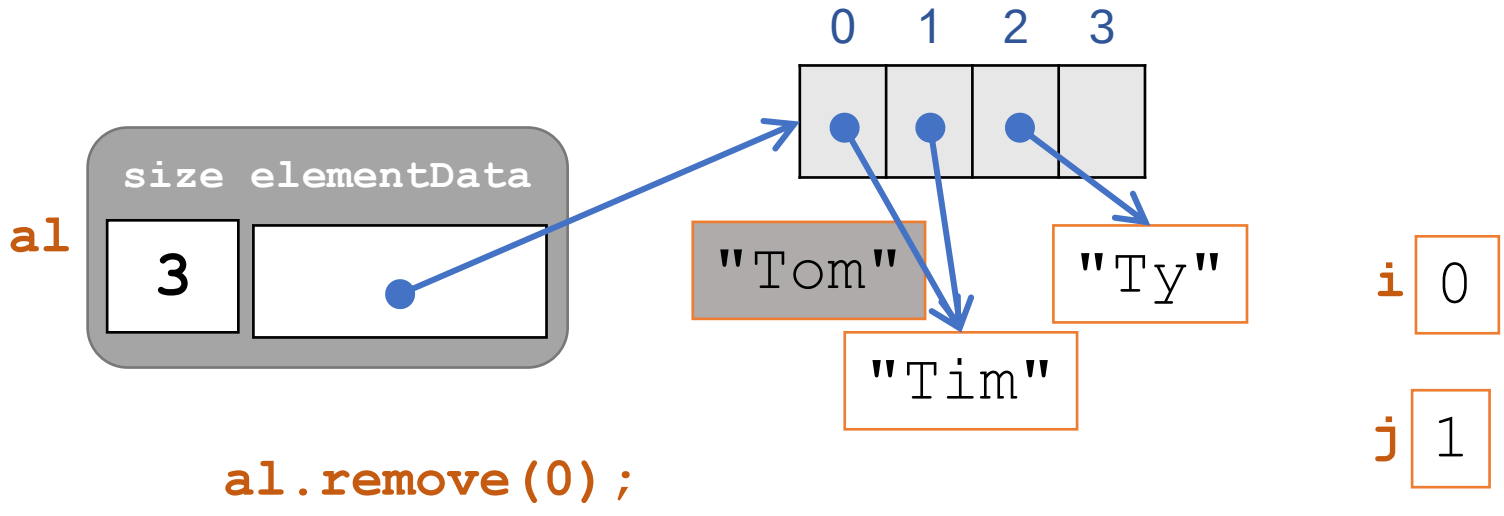
Example: remove an element

```
public void remove(int i) {
    assertInRange(i, size-1);
    for (int j=i+1; j<size; j++)
        elementData[j-1] = elementData[j];
    elementData[--size] = null;
}
```



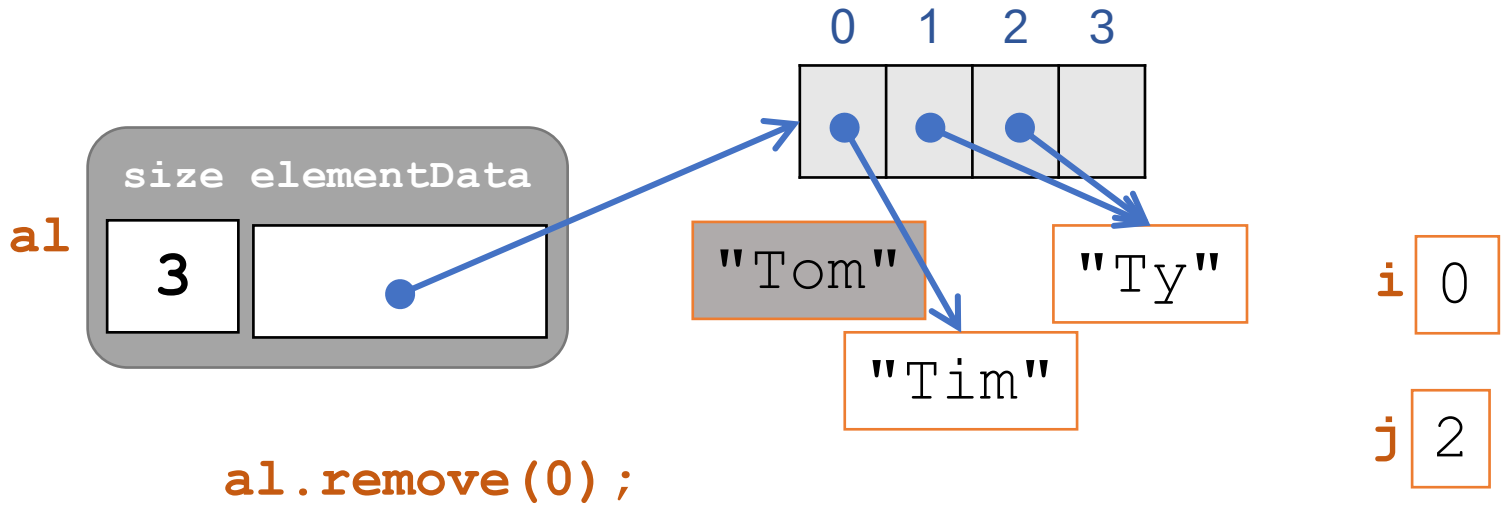
Example: remove an element

```
public void remove(int i) {
    assertInRange(i, size-1);
    for (int j=i+1; j<size; j++)
        elementData[j-1] = elementData[j]; // move from 1 to 0
    elementData[--size] = null;
}
```



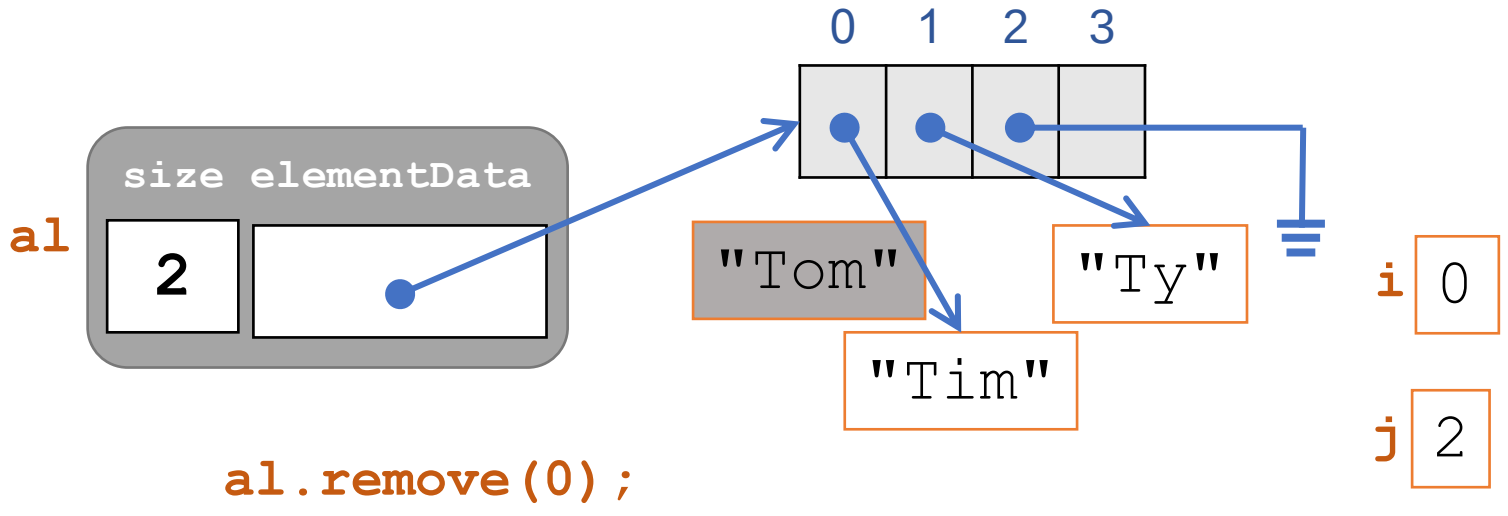
Example: remove an element

```
public void remove(int i) {
    assertInRange(i, size-1);
    for (int j=i+1; j<size; j++)
        elementData[j-1] = elementData[j]; // move from 2 to 1
    elementData[--size] = null;
}
```



Example: remove an element

```
public void remove(int i) {
    assertInRange(i, size-1);
    for (int j=i+1; j<size; j++)
        elementData[j-1] = elementData[j];
    elementData[--size] = null; // delete the last
}
```



Methods get and set

Class ArrayList

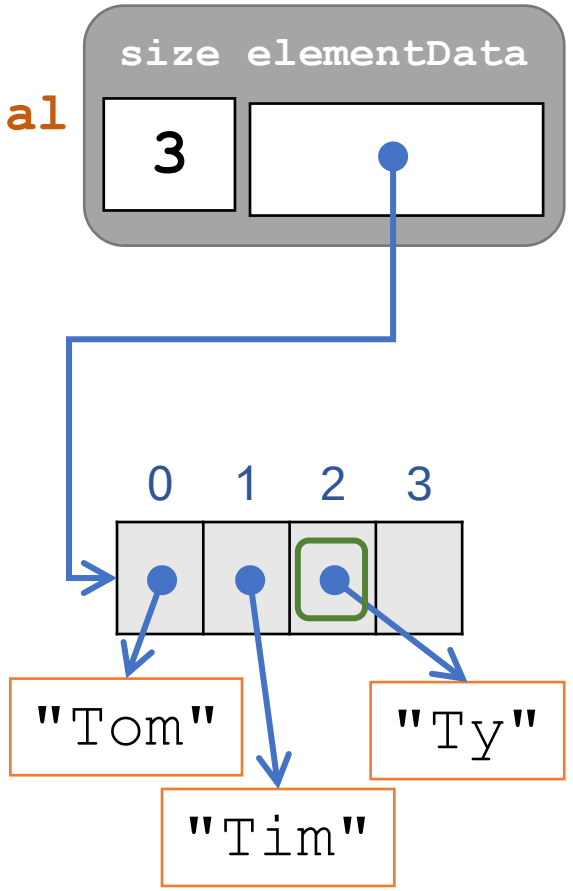
Get/set: Class ArrayList

```
public Object get(int i) {
   assertInRange(i, size-1);
   return elementData[i];
}
public void set(int i, Object e) {
   assertNonNull(e);
   assertInRange(i, size-1);
   elementData[i] = e;
}
```

Get: Class ArrayList

```
public Object get(int i) {
    assertInRange(i, size-1);
    return elementData[i];
}
```

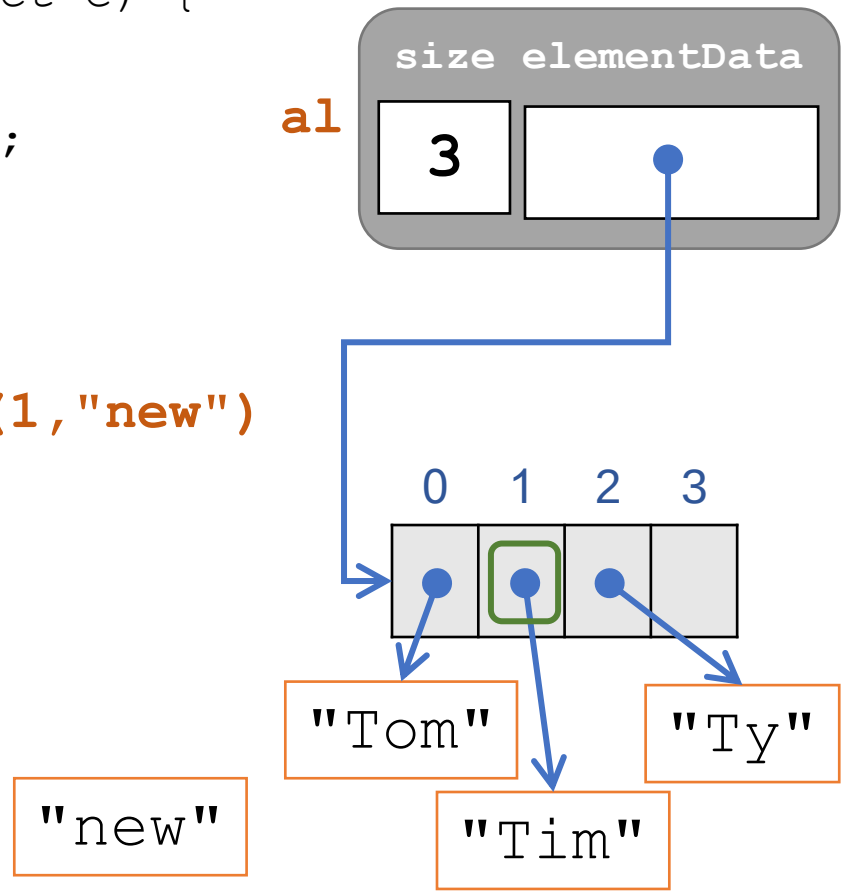
al.get(2)



Set: Class ArrayList

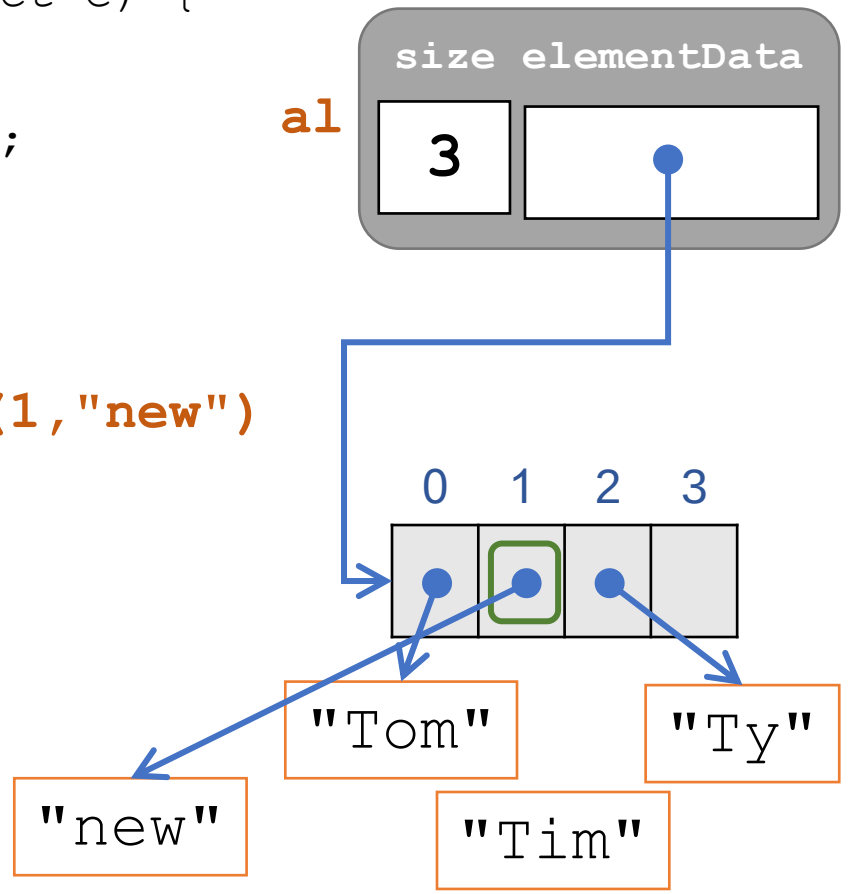
```
public void set(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size-1);
    elementData[i] = e;
}
```

al.set(1,"new")



Set: Class ArrayList

```
public void set(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size-1);
    elementData[i] = e;
}
```



Method equals

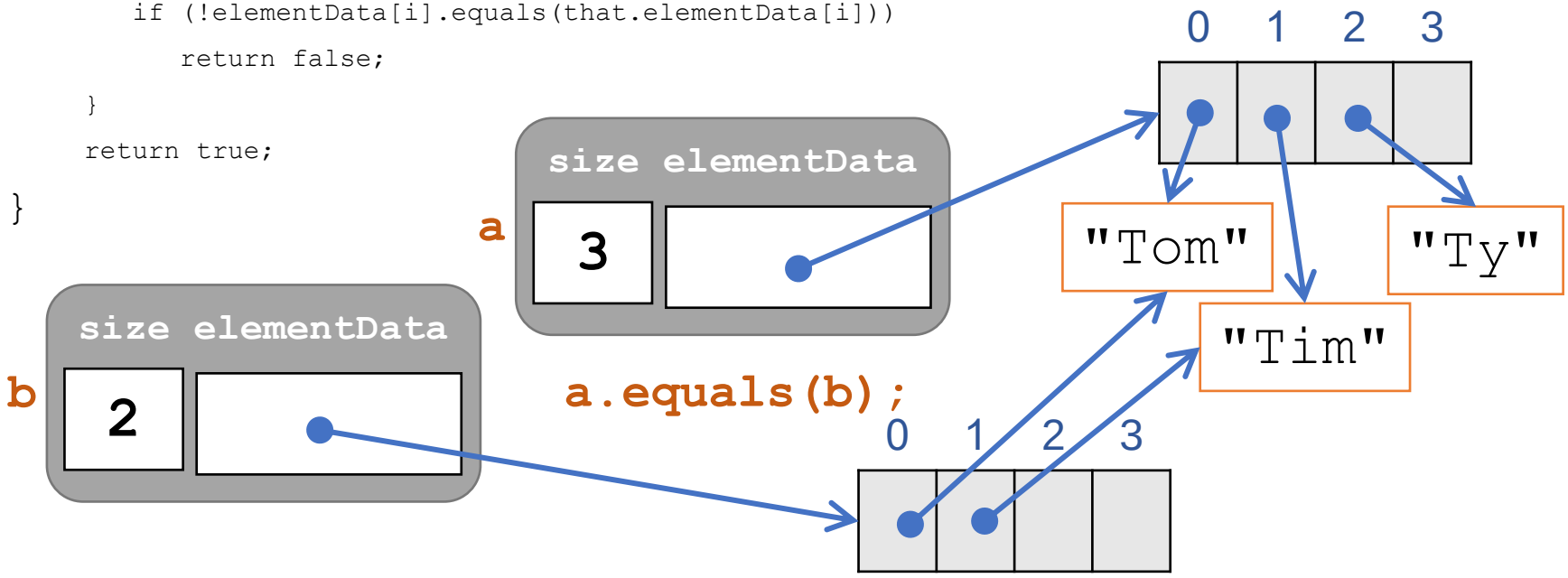
Class ArrayList

equals: Class ArrayList

```
public boolean equals(Object x) {
   // different class: false
   if (!(x instanceof ArrayList)) return false;
   // copy x as ArrayList
   ArrayList that = (ArrayList) x;
   // different size: false
   if (size != that.size) return false;
   // check each pair of elements
   for (int i=0; i<size; i++) {
      if (!elementData[i].equals(that.elementData[i]))
         return false;
   }
   return true;
}
```

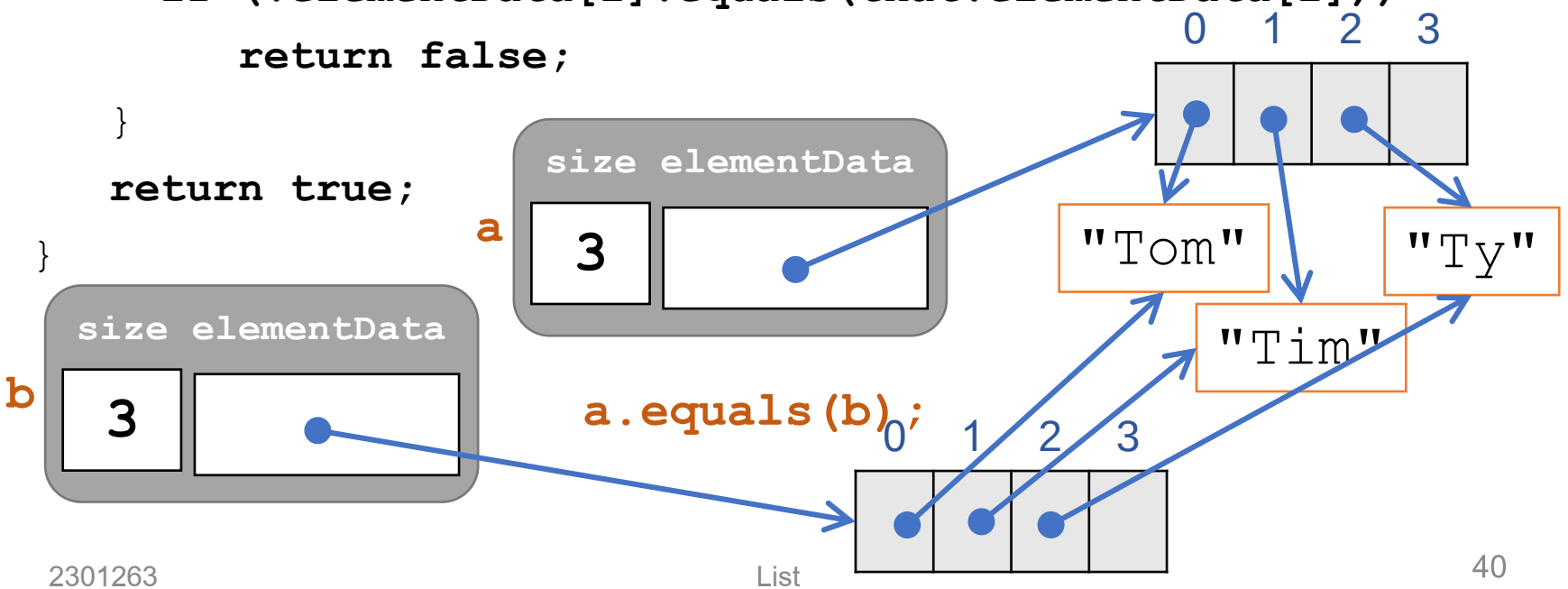
Example: equals

```
public boolean equals(Object x) {
    if (!(x instanceof ArrayList)) return false;
    ArrayList that = (ArrayList) x;
    if (size != that.size) return false;
    for (int i=0; i<size; i++) {
       if (!elementData[i].equals(that.elementData[i]))
          return false;
    }
    return true;
}
```



Example: equals

```
public boolean equals(Object x) {
    if (!(x instanceof ArrayList)) return false;
    ArrayList that = (ArrayList) x;
    if (size != that.size) return false;
    for (int i=0; i<size; i++) {
      if (!elementData[i].equals(that.elementData[i]))
          return false;
    }
    return true;
}
```



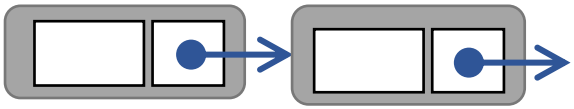
Exercises

เขียน `method`

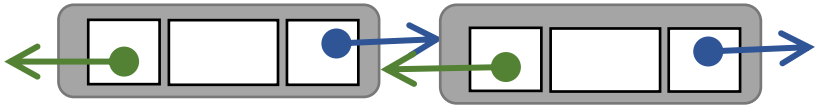
- `concat` ที่นำลิสต์มาต่อกัน
- `clone` ที่ `copy` ลิสต์เดิมมาใส่ลิสต์ใหม่
- `swap` ที่ สลับ `element` ที่ `i` กับ `j` ในลิสต์
- `removeAll` ที่ลบ `element` ทุกตัวที่มีค่าเท่ากับค่าที่กำหนดออกจากลิสต์

สร้าง List ด้วยการโยง

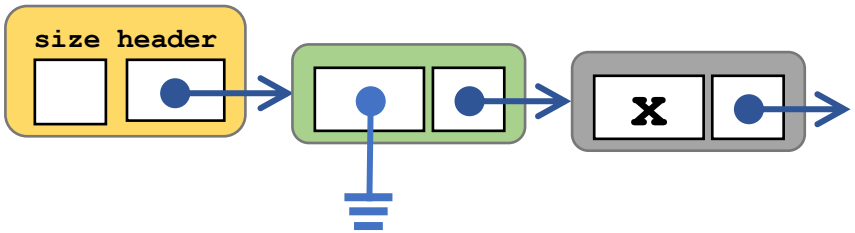
รายการโยงเดี่ยว



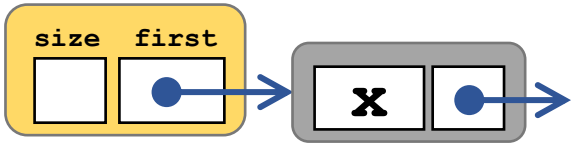
รายการโยงคู่



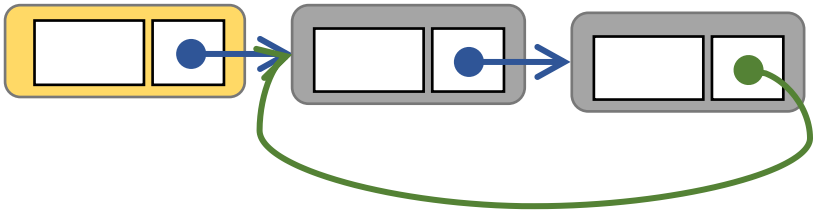
มีโหนดหัวที่เป็นโหนดว่าง



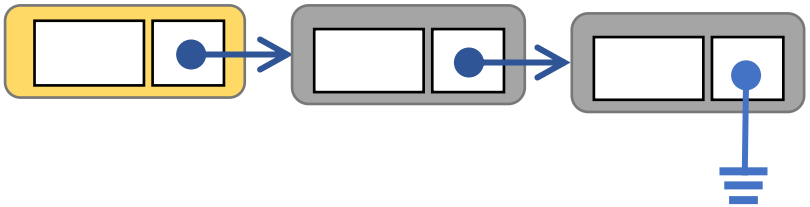
ไม่มีโหนดหัวที่เป็นโหนดว่าง

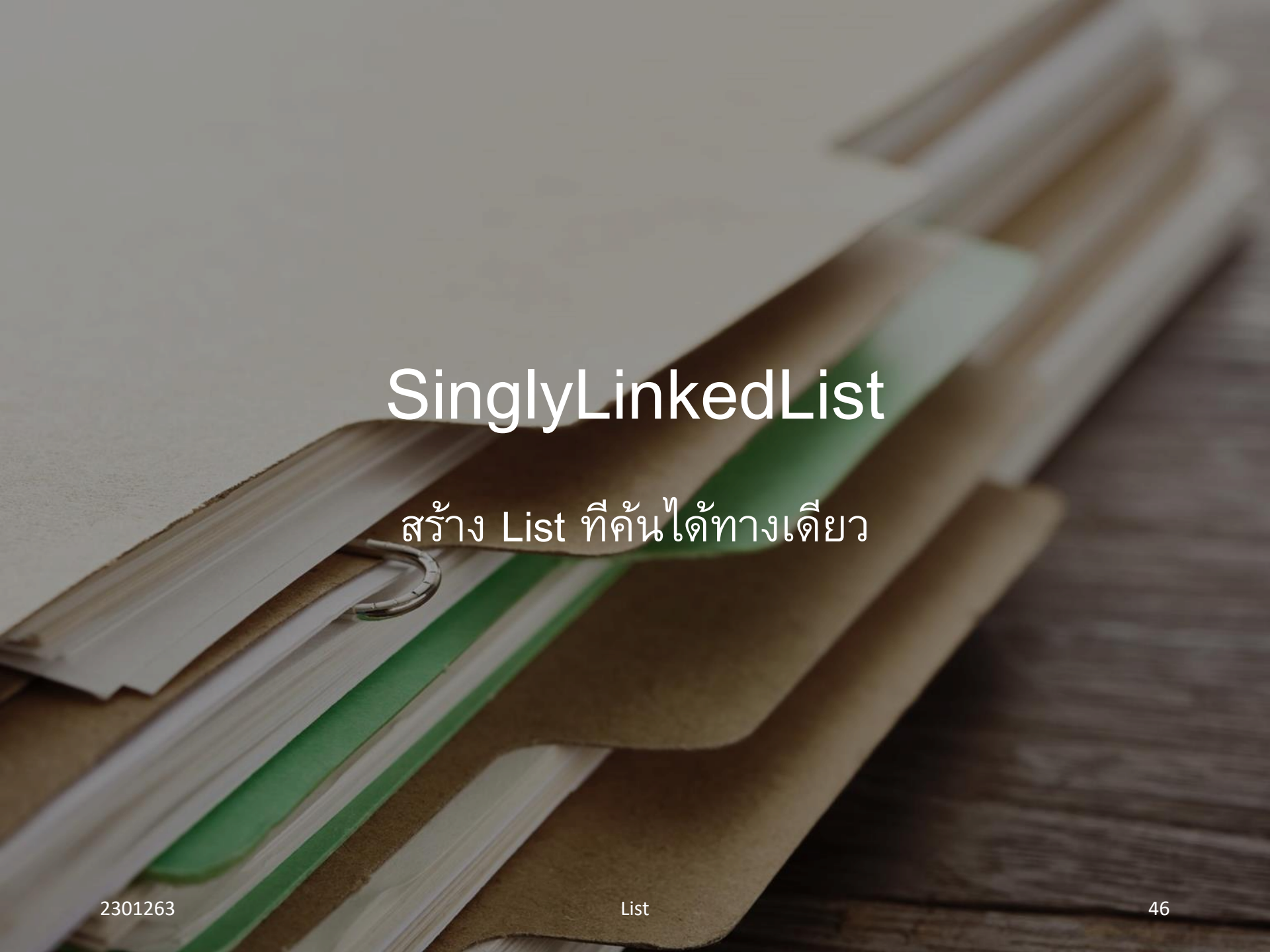


การโยงแบบวน



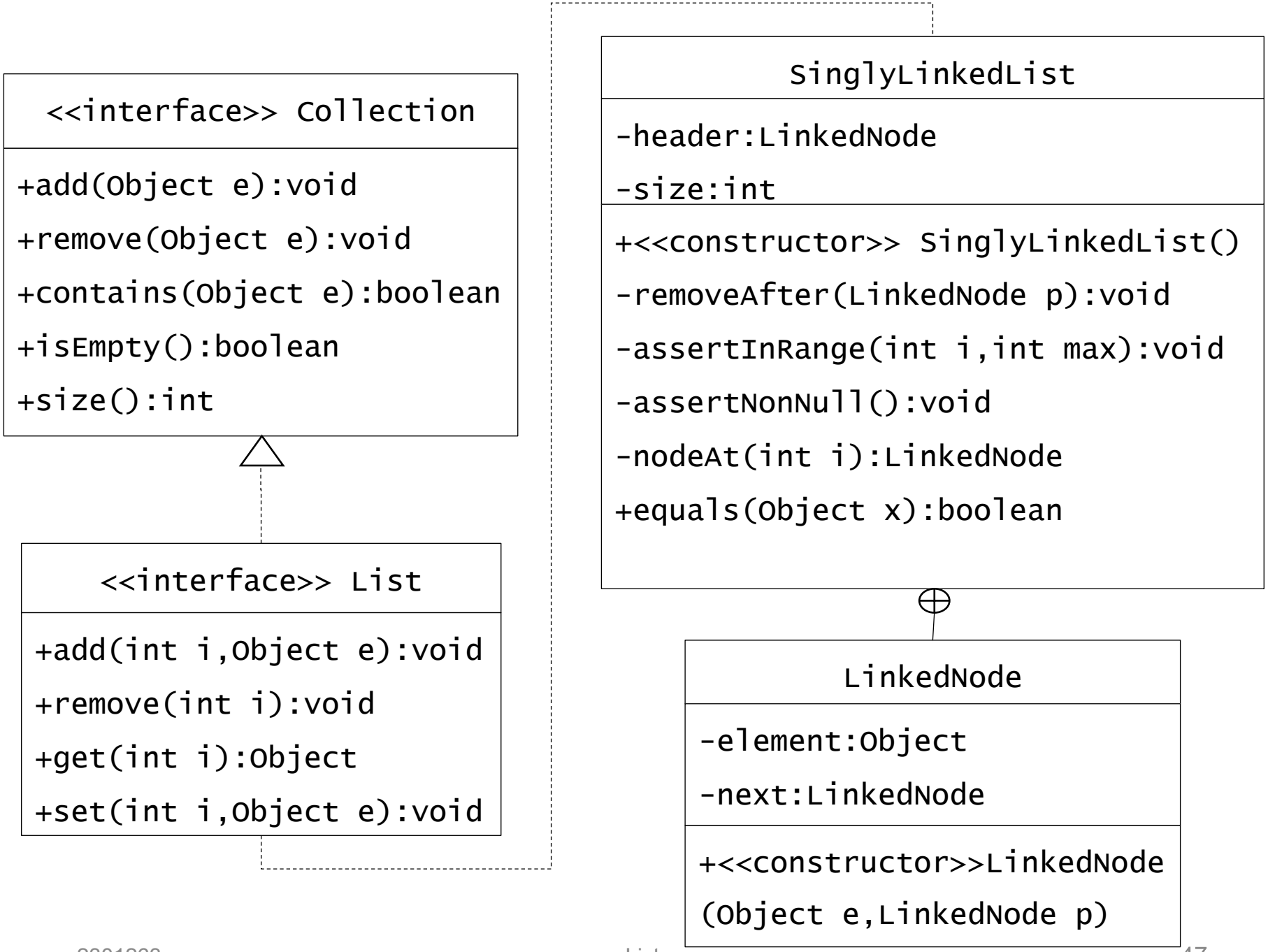
การโยงแบบไม่วน





SinglyLinkedList

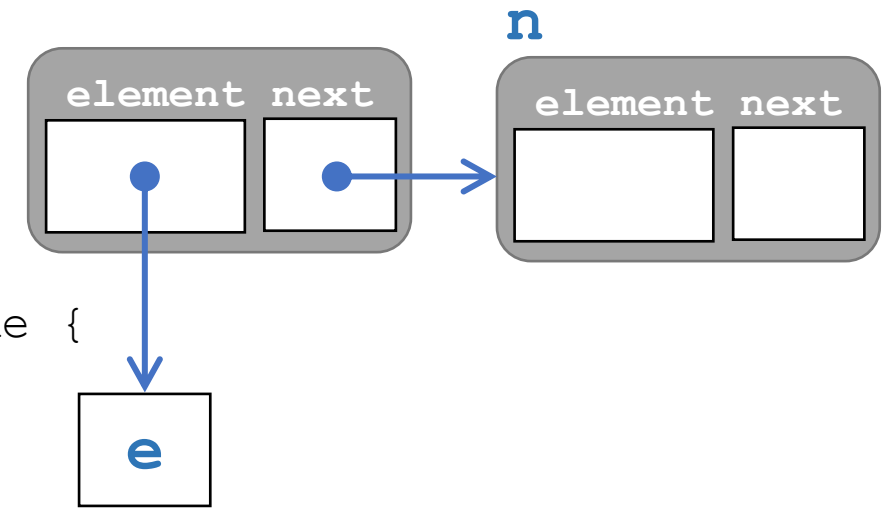
สร้าง List ที่ค้นได้ทางเดียว


<<interface>> Collection
+add(Object e):void
+remove(Object e):void
+contains(Object e):boolean
+isEmpty():boolean
+size():int
SinglyLinkedList
-header:LinkedNode
-size:int
+<<constructor>> SinglyLinkedList()
-removeAfter(LinkedNode p):void
-assertInRange(int i,int max):void
-assertNonNull():void
-nodeAt(int i):LinkedNode
+equals(Object x):boolean
<<interface>> List
+add(int i,Object e):void
+remove(int i):void
+get(int i):Object
+set(int i,Object e):void
LinkedNode
-element:Object
-next:LinkedNode
+<<constructor>>LinkedNode
(Object e,LinkedNode p)

Class LinkedNode

โหนด

- Attribute **`element`** เก็บข้อมูลของโหนด
- Attribute **`next`** ชี้ไปยังโหนดอื่น

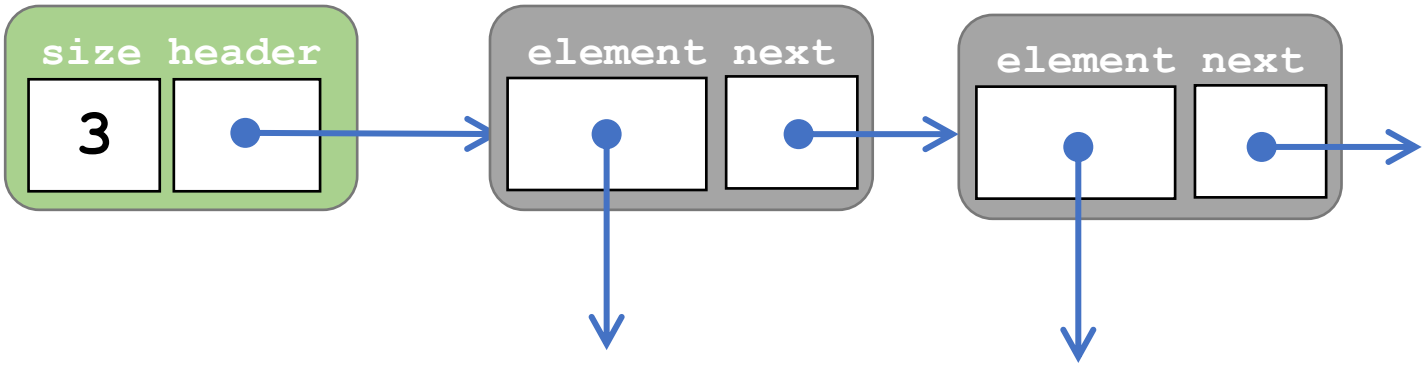



```
private static class LinkedNode {
    private Object element;
    private LinkedNode next;

    LinkedNode(Object e, LinkedNode n) {
        this.element = e;
        this.next = n;
    }
}
```

Class SinglyLinkedList

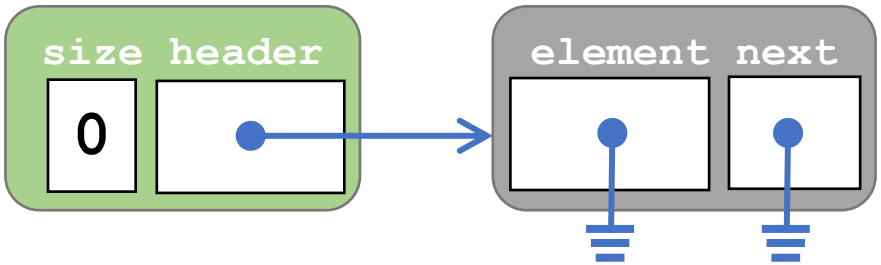
สร้าง List คล้ายกับ ListCollection

- Attribute **`header`** ไปที่โหนดแรกในลิสต์
- Attribute **`size`** เก็บจำนวนข้อมูลในชุด



Create new object: Class SinglyLinkedList

```
public class SinglyLinkedList implements List {
    private int size;
    private LinkedNode header;

    public SinglyLinkedList() {
        size = 0;
        header = new LinkedNode(null, null);
    }
}
```



Methods in Class SinglyLinkedList

```
public class SinglyLinkedList implements List {
   private int size;
   private LinkedNode header;
   public SinglyLinkedList() {...}
   public int size() {...}          // interface Collection
   public boolean isEmpty {...}
   public boolean contains(Object e) {...}
   public void add(Object e) {...}
   public void remove(Object e) {...}
   private void removeAfter(LinkedNode p) {...}
   private static void assertInRange(int i, int max) {...}
   private static void assertNonNull(Object e) {...}
   private LinkedNode nodeAt(int i) {...}
   public Object get(int i) {...}    // interface List
   public void set(int i, Object e) {...}
   public void add(int i, Object e) {...}
   public void remove(int i) {...}
}
```

Methods in Class SinglyLinkedList

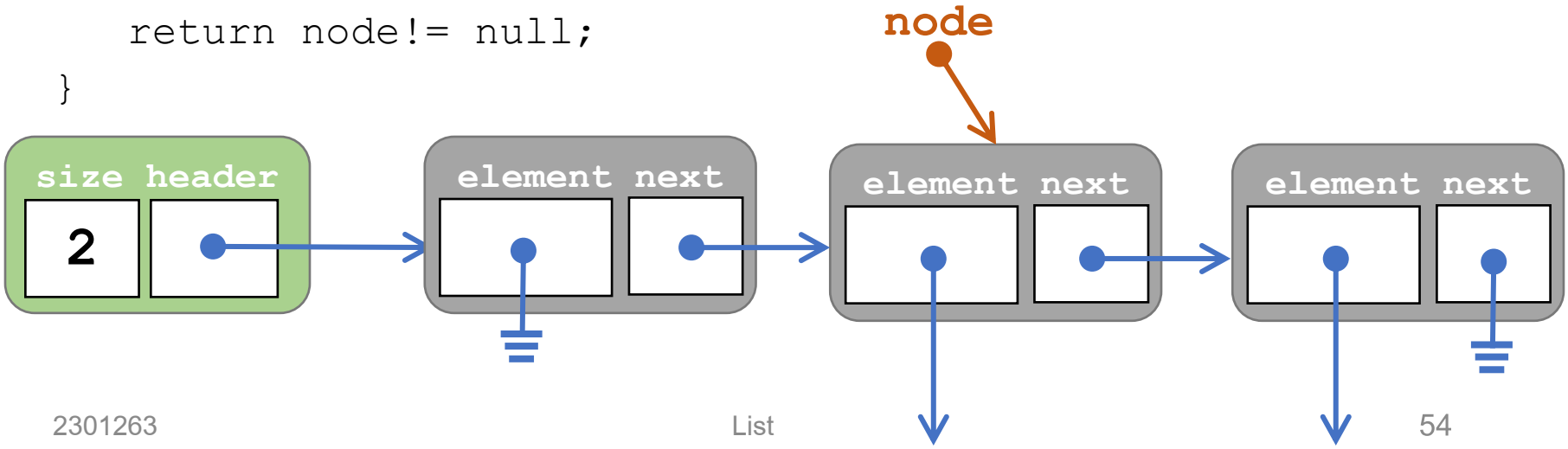
```
public class SinglyLinkedList implements List {
    private int size;
    private LinkedNode header;
    public SinglyLinkedList() {...}
    public int size() {...}                    // interface Collection
    public boolean isEmpty {...}
    public boolean contains(Object e) {...}
    public void add(Object e) {...}
    public void remove(Object e) {...}
    private void removeAfter(LinkedNode p) {...}
    private static void assertInRange(int i, int max) {...}
    private static void assertNonNull(Object e) {...}
    private LinkedNode nodeAt(int i) {...}
    public Object get(int i) {...}     // interface List
    public void set(int i, Object e) {...}
    public void add(int i, Object e) {...}
    public void remove(int i) {...}
}
```

Methods in Class SinglyLinkedList

```
public class SinglyLinkedList implements List {
    private int size;
    private LinkedNode header;
    public SinglyLinkedList() {...}
    public int size() {...}          // interface Collection
    public boolean isEmpty {...}
    public boolean contains(Object e) {...}
    public void add(Object e) {...}
    public void remove(Object e) {...}
    private void removeAfter(LinkedNode p) {...}
    private static void assertInRange(int i, int max) {...}
    private static void assertNonNull(Object e) {...}
    private LinkedNode nodeAt(int i) {...}
    public Object get(int i) {...}    // interface List
    public void set(int i, Object e) {...}
    public void add(int i, Object e) {...}
    public void remove(int i) {...}
}
```

Some simple methods (Same as in LinkedCollection)

```
public int size()          {   return size;   }

public boolean isEmpty() {   return size==0;}

public boolean contains(Object e) {
    LinkedNode node = header.next;
    while (node != null && !node.element.equals(e))
        node = node.next;
    return node!= null;
}
```



Method add

Class SinglyLinkedList

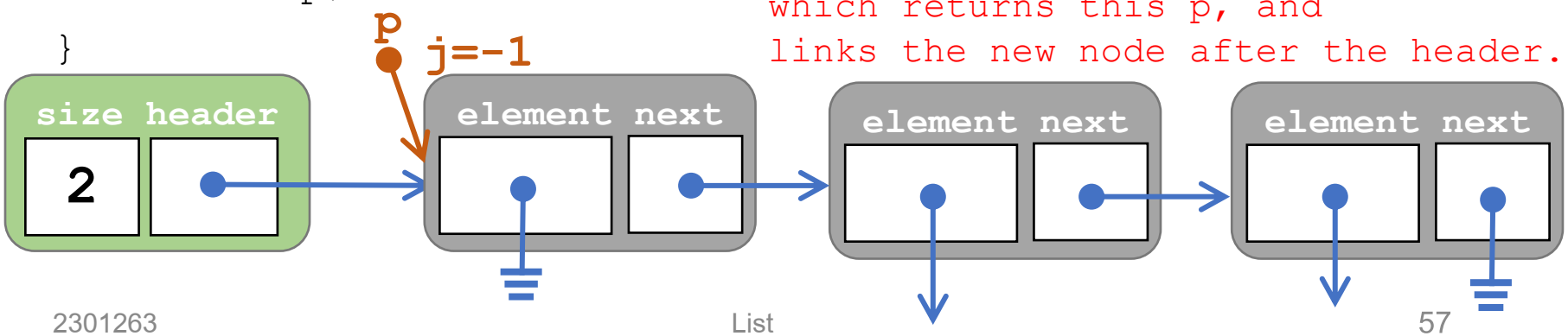
add: Class SinglyLinkedList

```
public void add(int i, Object e) {
   assertNonNull(e);
   assertInRange(i, size);
   LinkedNode p = nodeAt(i-1);
   p.next = new LinkedNode(e, p.next);   ++size;
}

private LinkedNode nodeAt(int i) {
   LinkedNode p = header;
   for (int j=-1; j<i; j++) p = p.next;
   return p;
}
```

How method nodeAt is used

```
public void add(int i, Object e) {
    assertNonNull(e);                    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);    ++size;
}
private LinkedNode nodeAt(int i) {
    LinkedNode p = header;
    for (int j=-1; j<i; j++) p = p.next;
    return p;
}
```

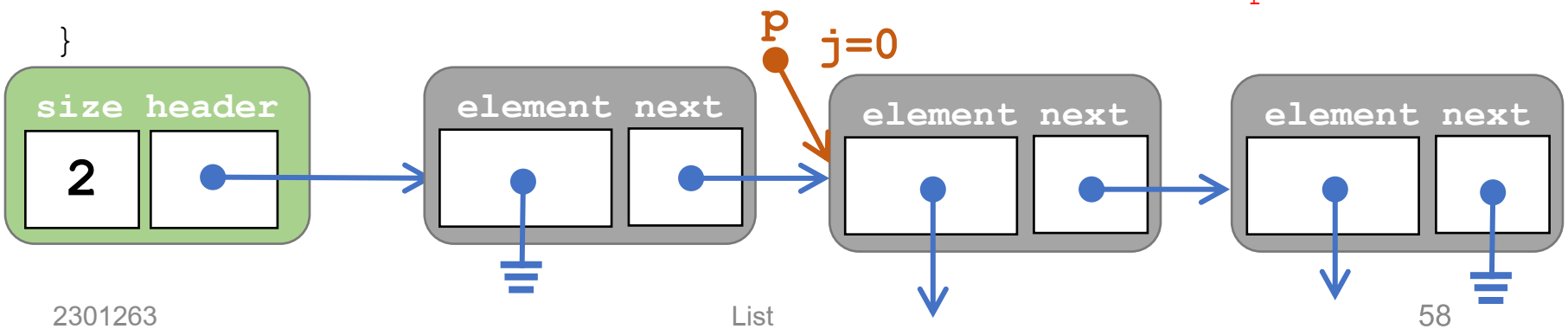
add(0,e) calls nodeAt(-1), which returns this p, and links the new node after the header.



How method nodeAt is used

```
public void add(int i, Object e) {
    assertNonNull(e);                    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);    ++size;
}
private LinkedNode nodeAt(int i) {
    LinkedNode p = header;
    for (int j=-1; j<i; j++) p = p.next;
    return p;
}
```

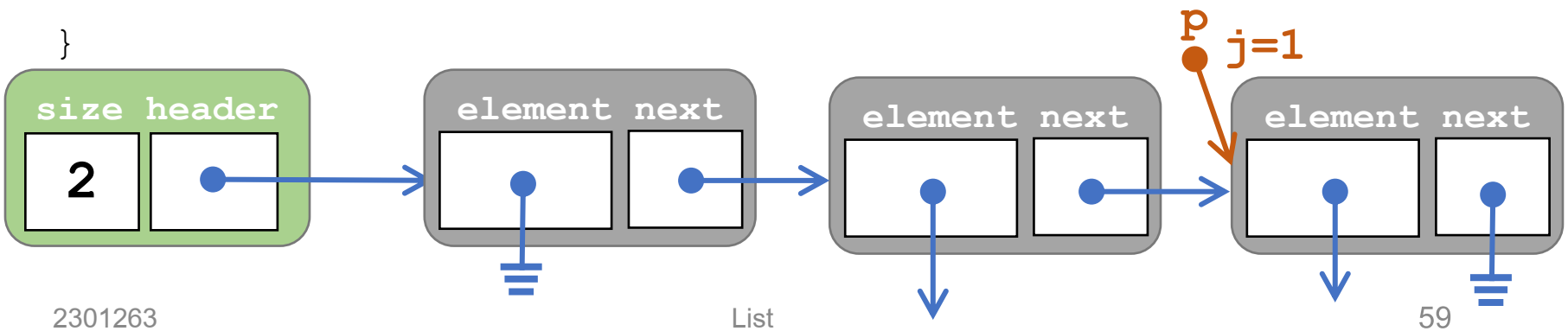
add(1,e) calls nodeAt(0), which returns this p,and links the new node after this node p.



How method nodeAt is used

```
public void add(int i, Object e) {
    assertNonNull(e);                    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);    ++size;
}
private LinkedNode nodeAt(int i) {
    LinkedNode p = header;
    for (int j=-1; j<i; j++) p = p.next;
    return p;
}
```

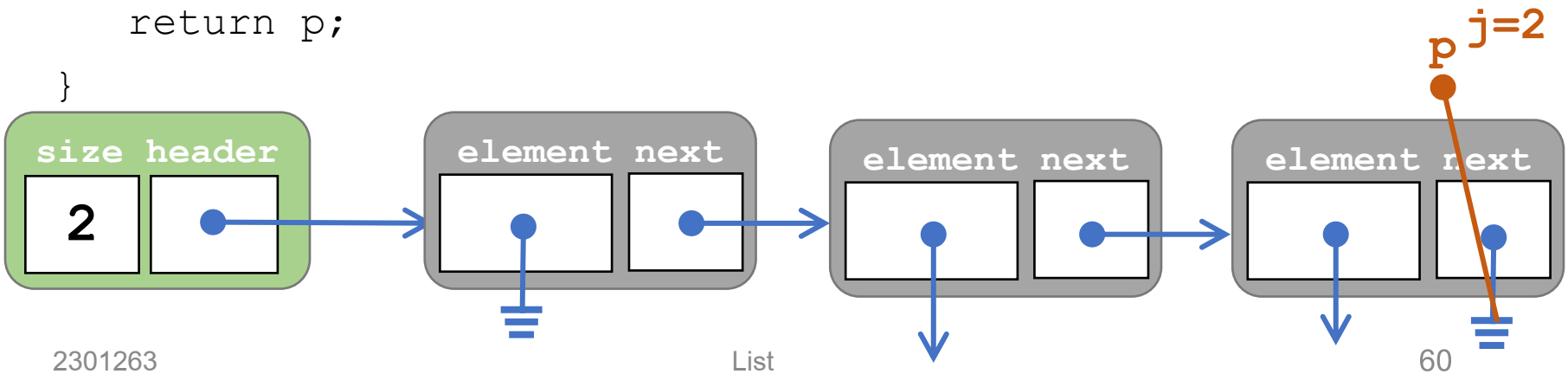
add(2,e) calls nodeAt(1), which returns this p,and links the new node after this node p.



How method nodeAt is used

```
public void add(int i, Object e) {
    assertNonNull(e);                    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);    ++size;
}
private LinkedNode nodeAt(int i) {
    LinkedNode p = header;
    for (int j=-1; j<i; j++) p = p.next;
    return p;
}
```

Could this happen?



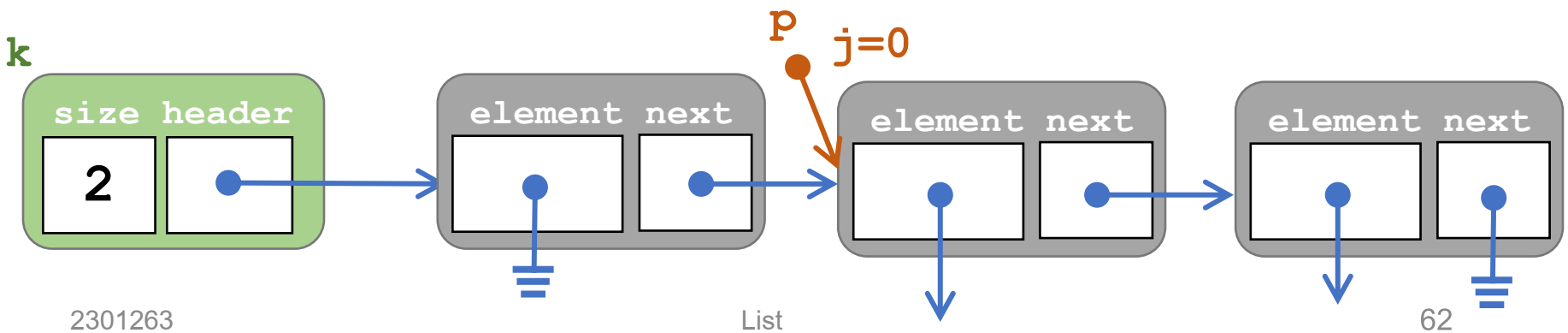
Add a value in the middle of the list

Example 1

Add in the middle of the list

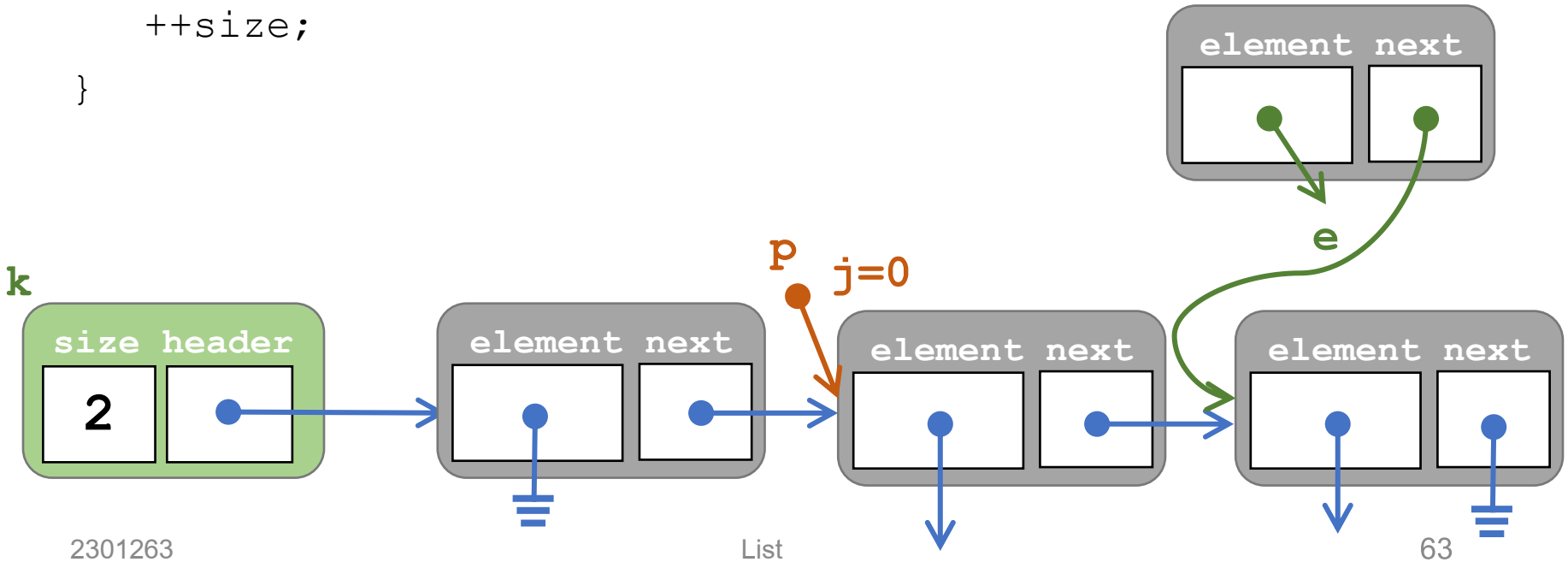
```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```

k.add(1,e) calls nodeAt(0), which returns this p,and links the new node after this node p.



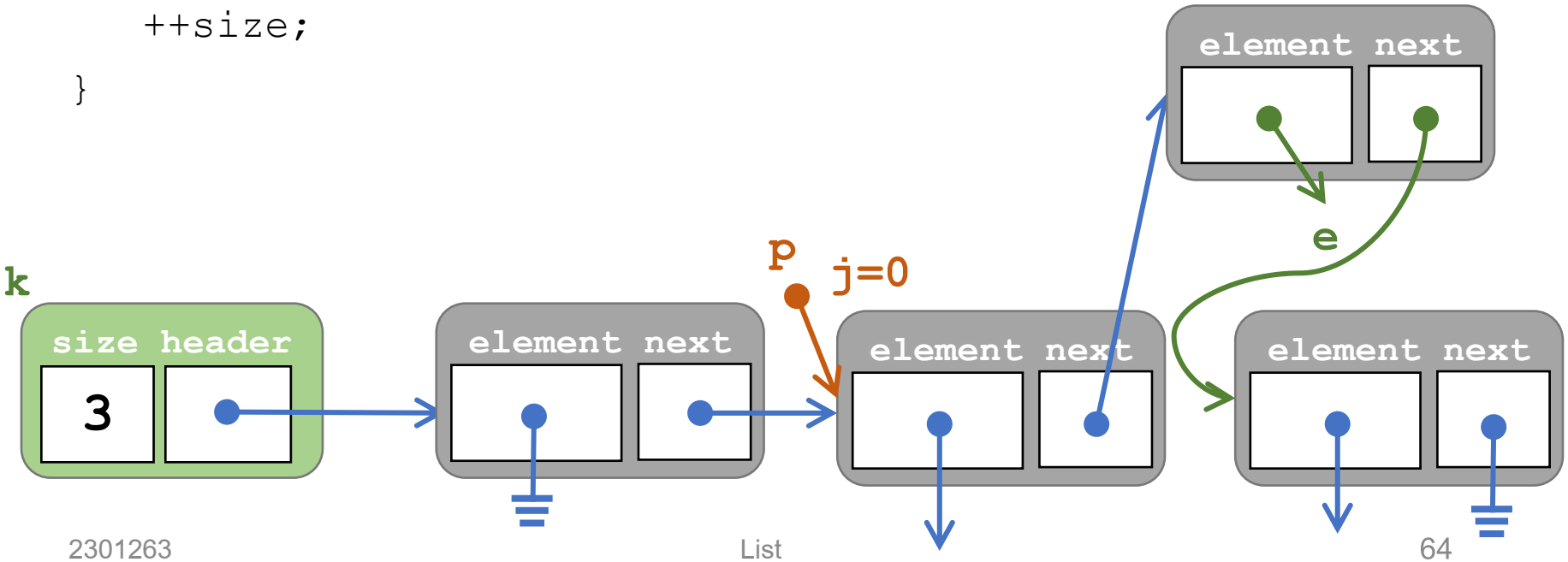
Add in the middle of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



Add in the middle of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



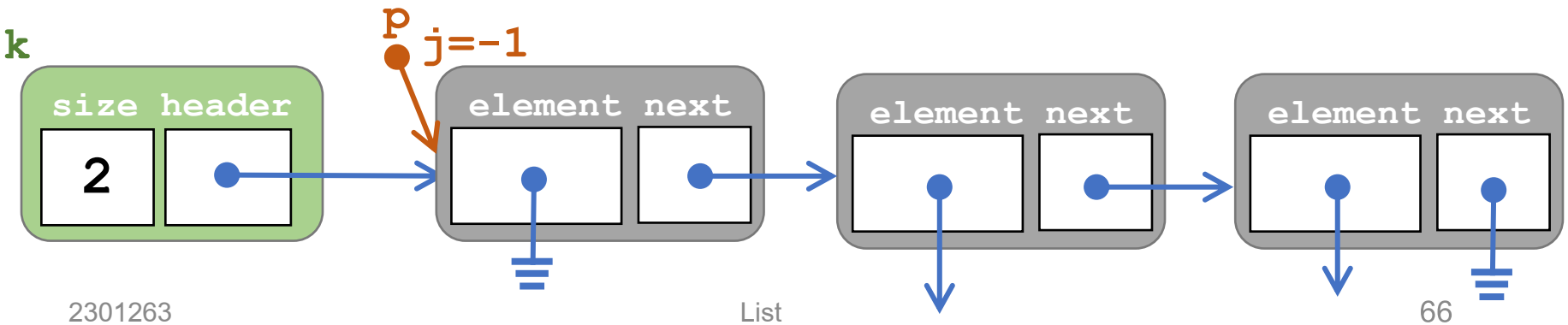
Add a value in a at the head of the list

Example 2

Add at the head of the list

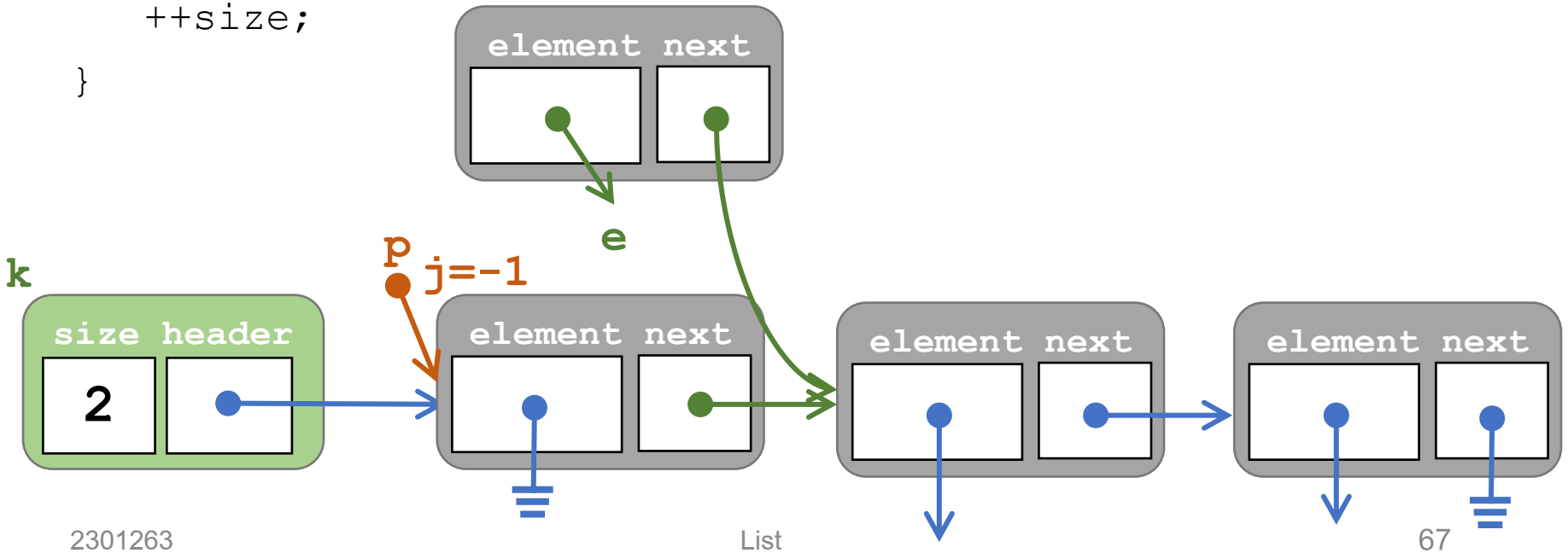
```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```

k.add(0,e) calls nodeAt(-1), which returns this p, and links the new node after this node p.



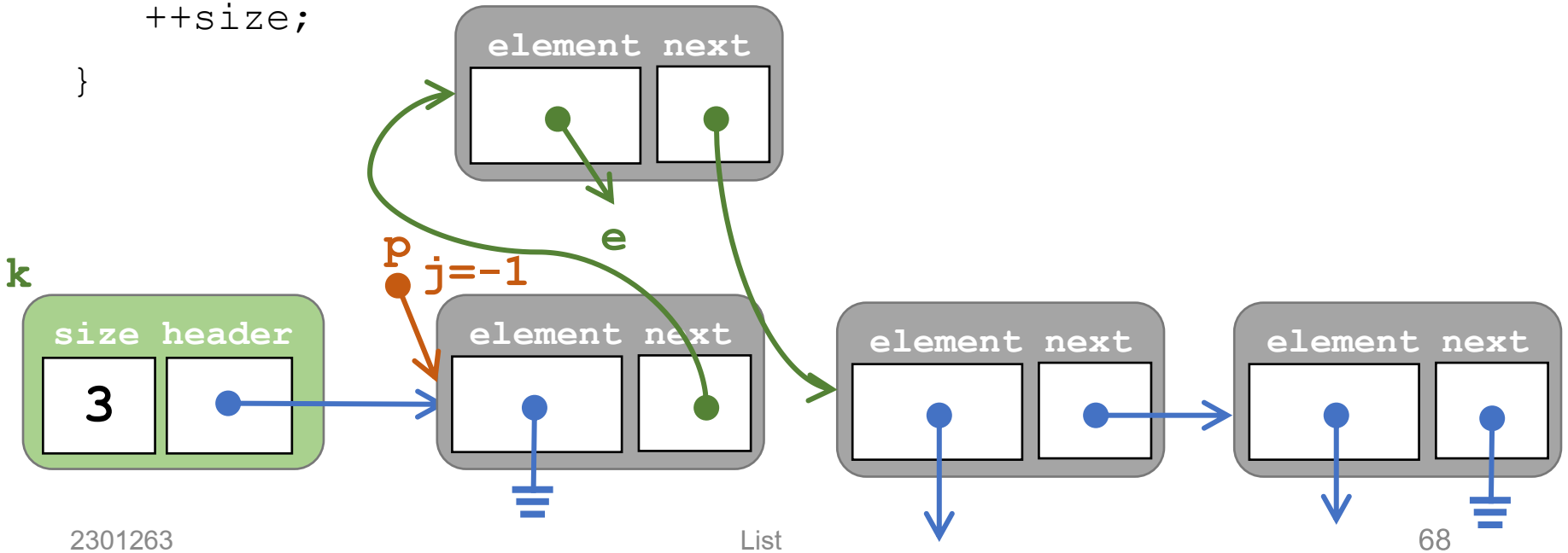
Add at the head of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



Add at the head of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



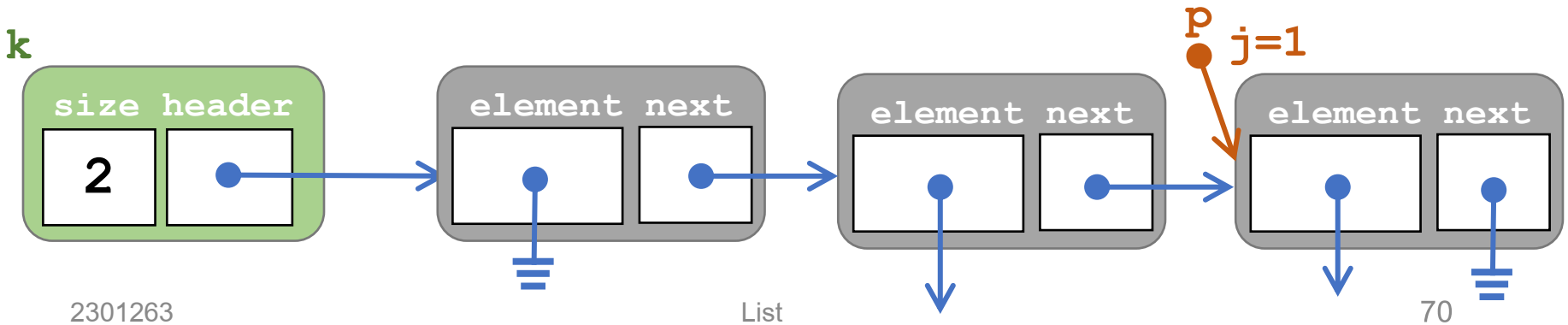
Add a value in a at the end of the list

Example 3

Add at the end of the list

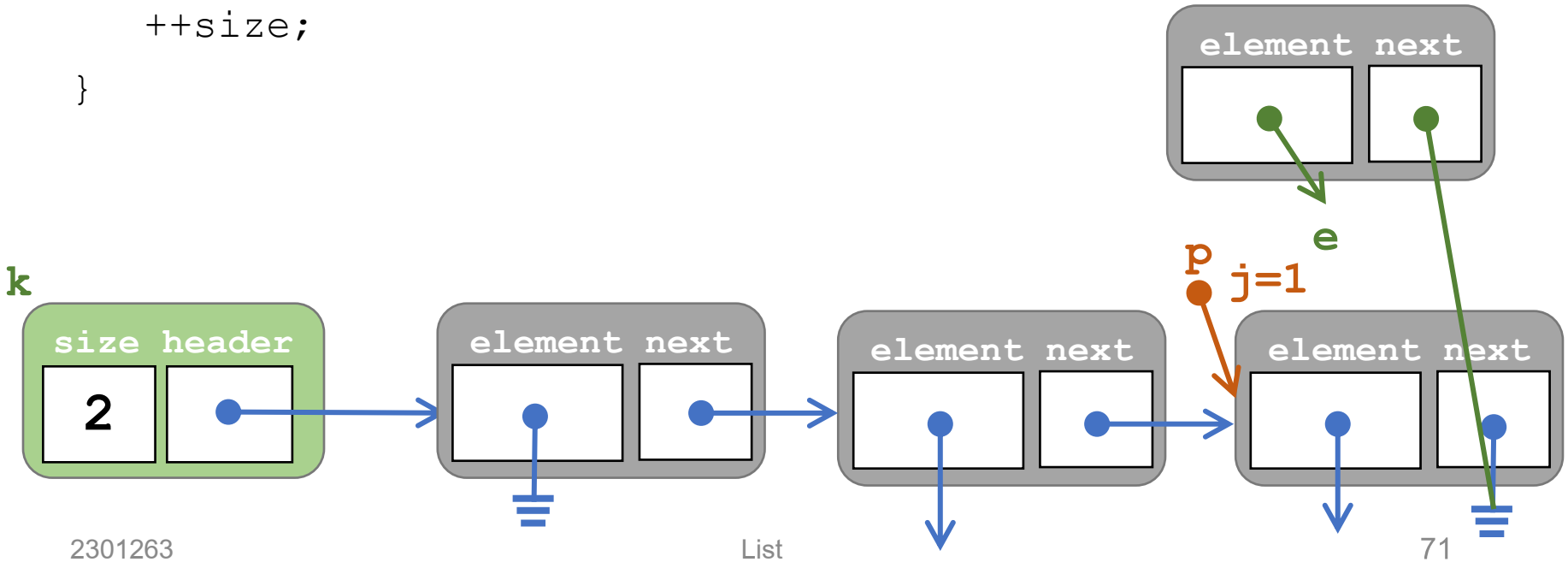
```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```

k.add(2,e) calls nodeAt(1), which returns this p, and links the new node after this node p.



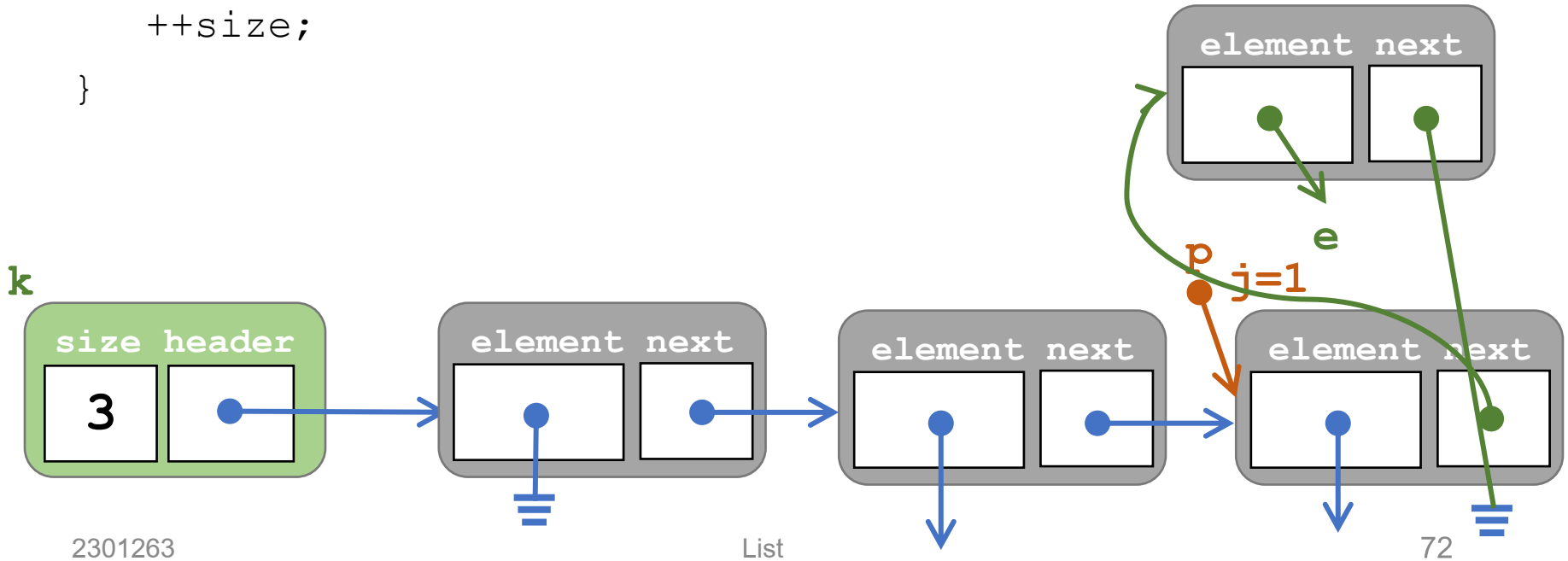
Add at the end of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



Add at the end of the list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



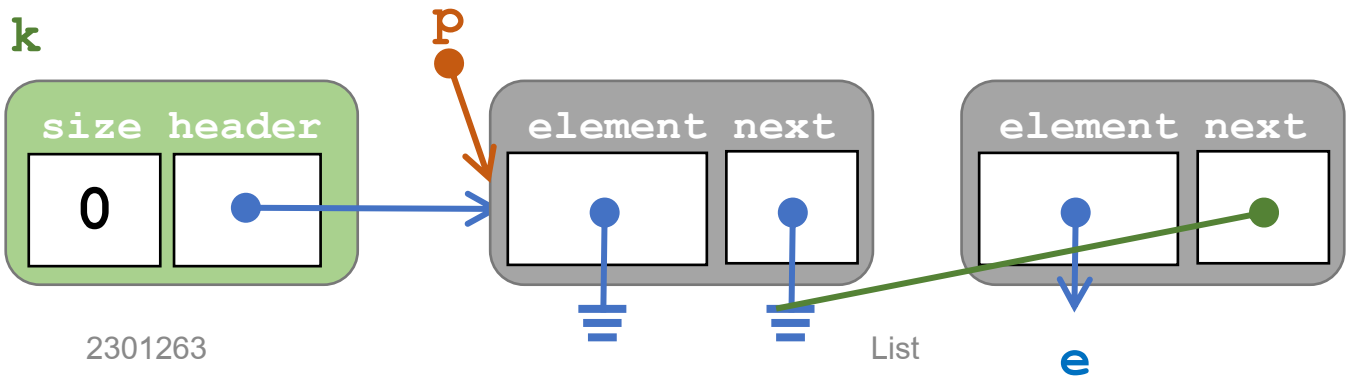
Add a value in an empty list

Example 4

Add in an empty list

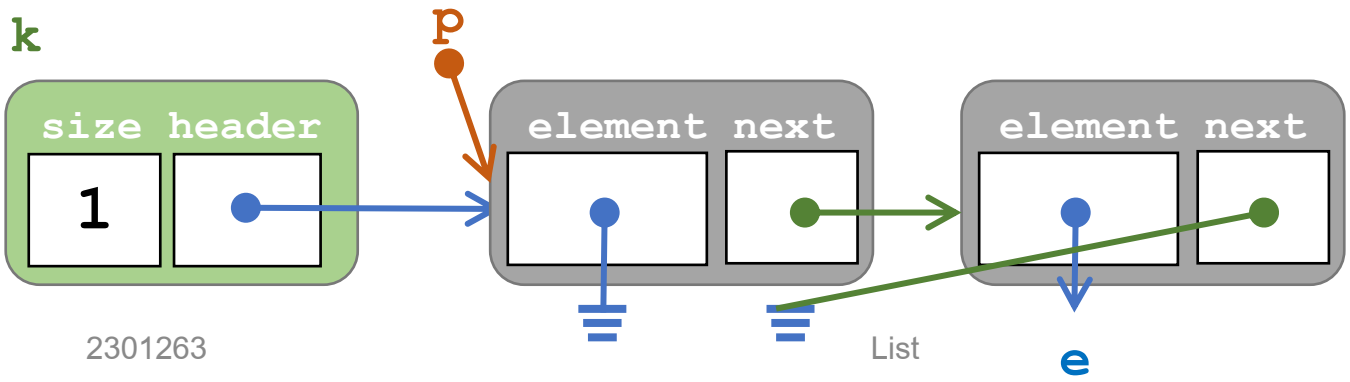
```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```

k.add(0,e) calls nodeAt(0), which returns this p, and links the new node after this node p.



Add in an empty list

```
public void add(int i, Object e) {
    assertNonNull(e);
    assertInRange(i, size);
    LinkedNode p = nodeAt(i-1);
    p.next = new LinkedNode(e, p.next);
    ++size;
}
```



Method remove

Class SinglyLInkedList

remove: Class SinglyLinkedList

```
public void remove(int i) {
   assertInRange(i, size-1);
   LinkedNode p = nodeAt(i-1);
   removeAfter(p);
}

private void removeAfter(LinkedNode p) {
   if (p.next != null) {
      p.next = p.next.next;
      --size;
   }
}
```

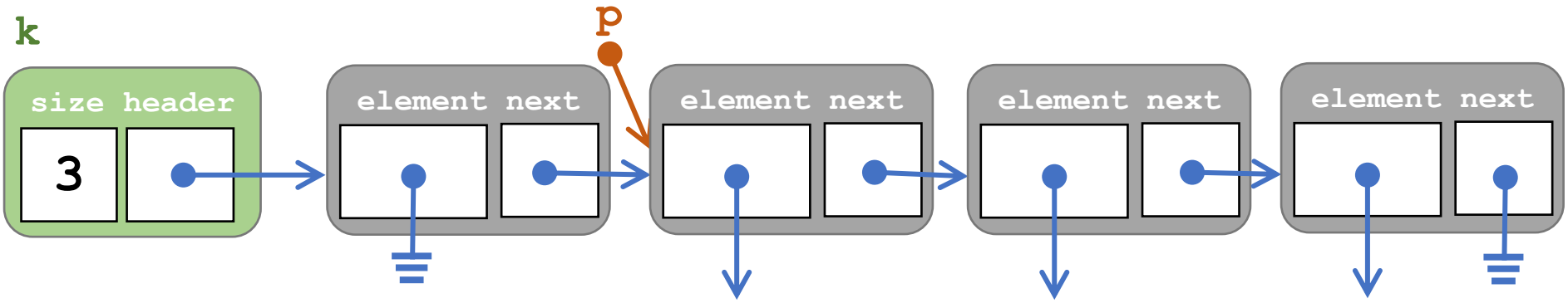
Remove from the middle of a list

Example 1

Remove from the middle of a list

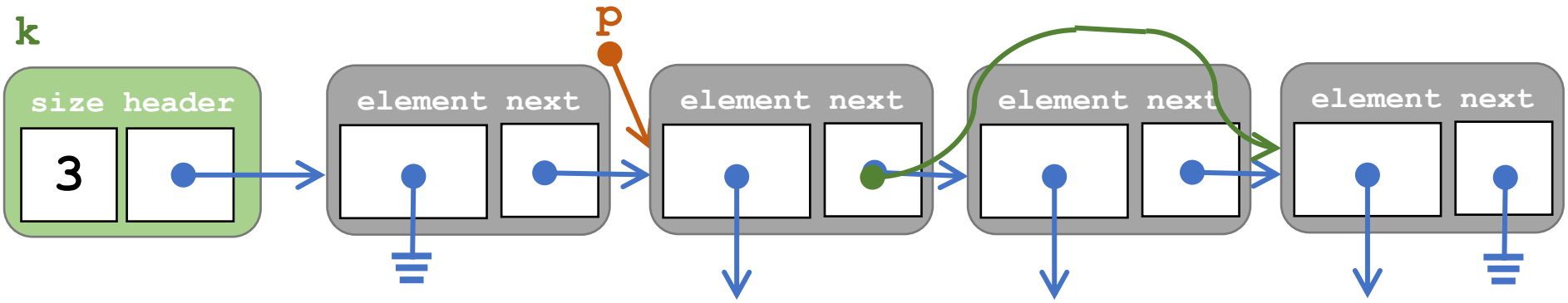
```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```

k.remove(1) calls nodeAt(0), which returns this p, and k.removeAfter(p) is called.



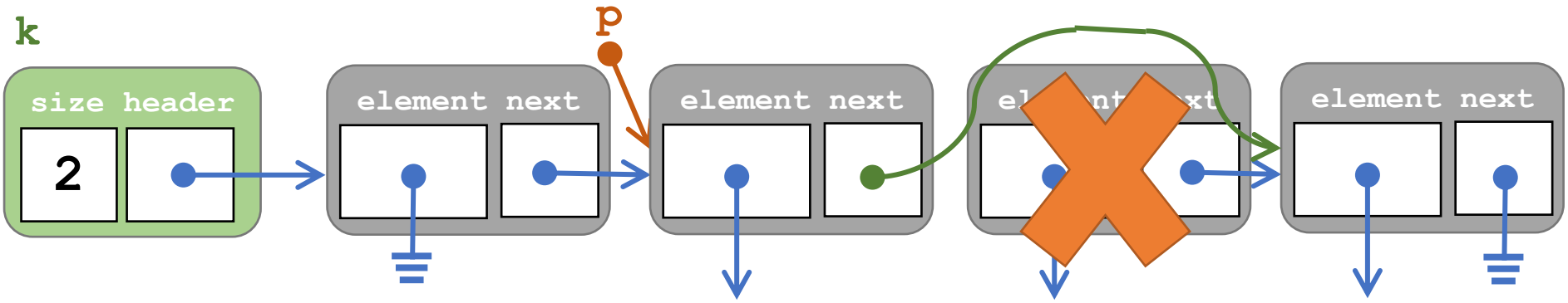
Remove from the middle of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



Remove from the middle of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



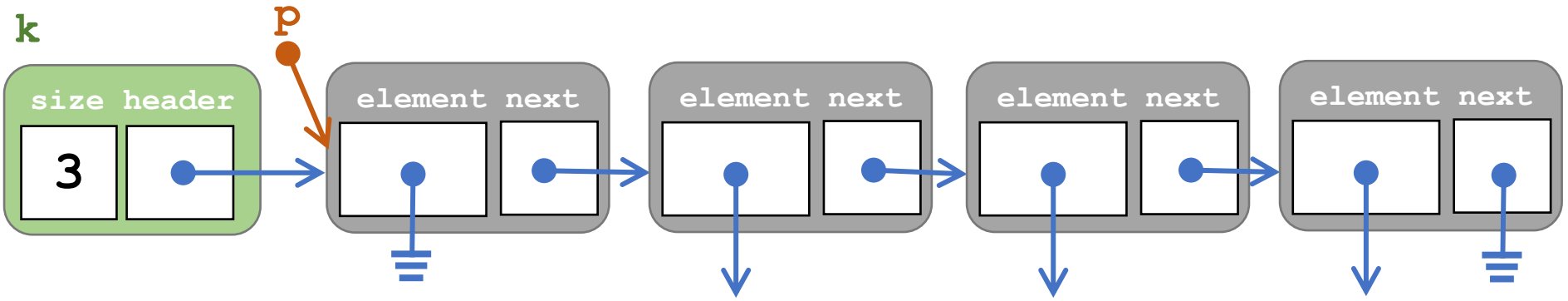
Remove at the head of the list

Example 2

Remove at the head of a list

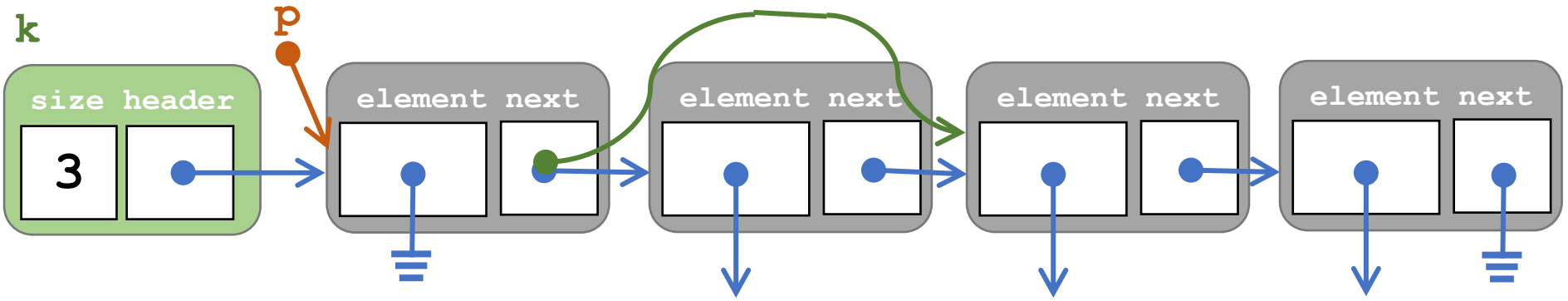
```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```

k.remove(0) calls nodeAt(-1), which returns this p, and k.removeAfter(p) is called.



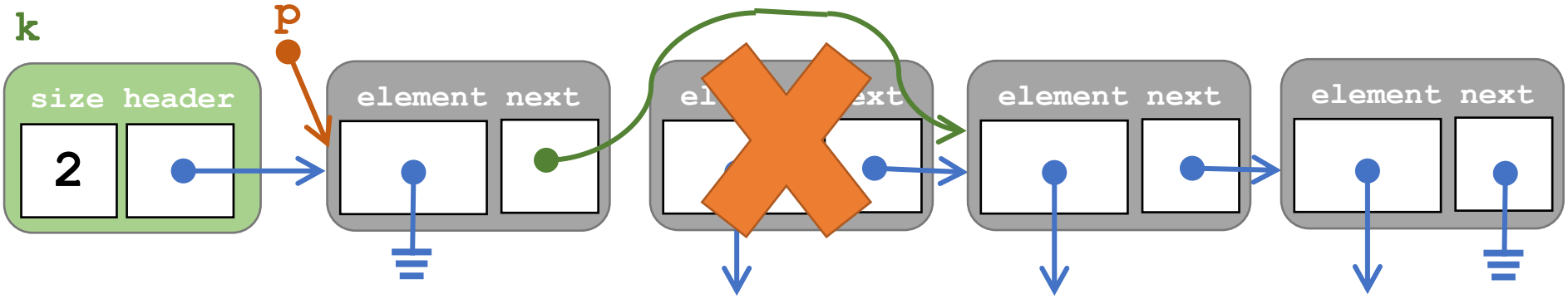
Remove at the head of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



Remove at the head of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



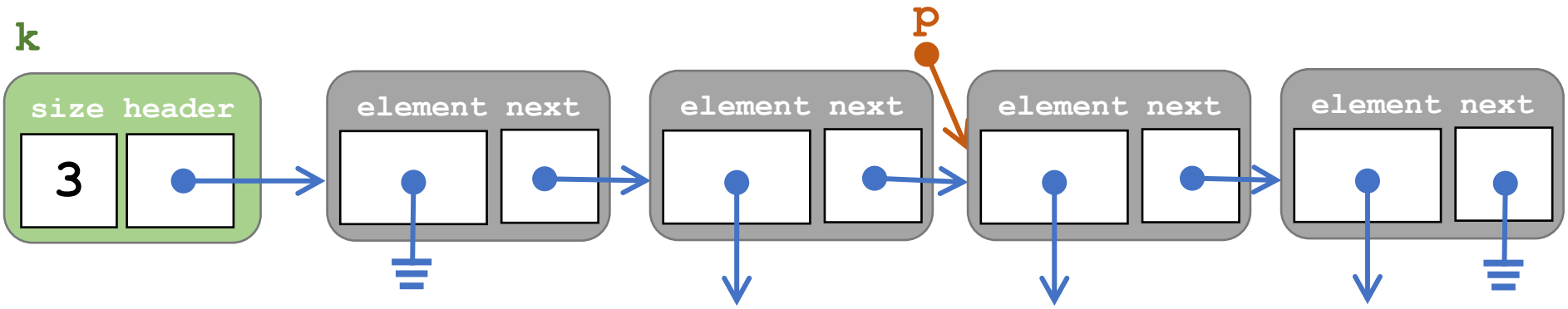
Remove at the end of the list

Example 3

Remove at the end of a list

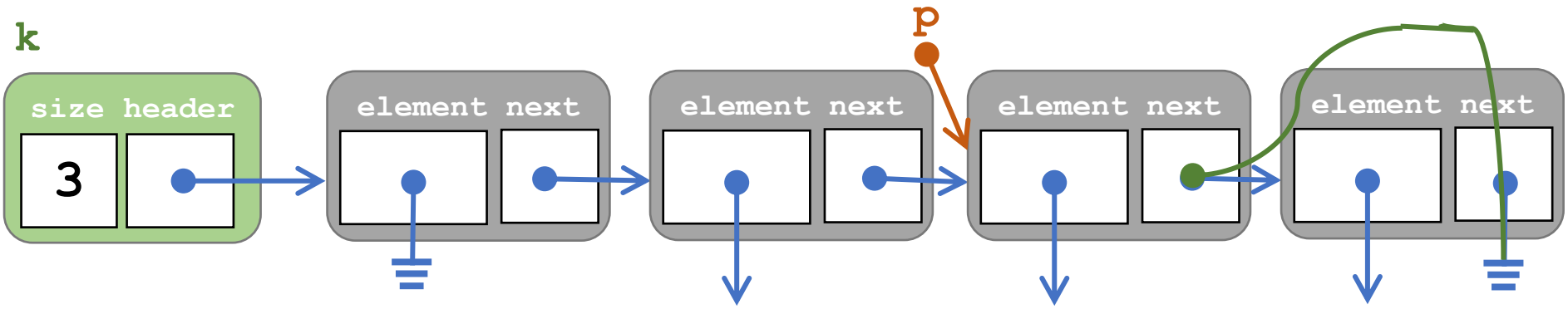
```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```

k.remove(2) calls nodeAt(1), which returns this p, and k.removeAfter(p) is called.



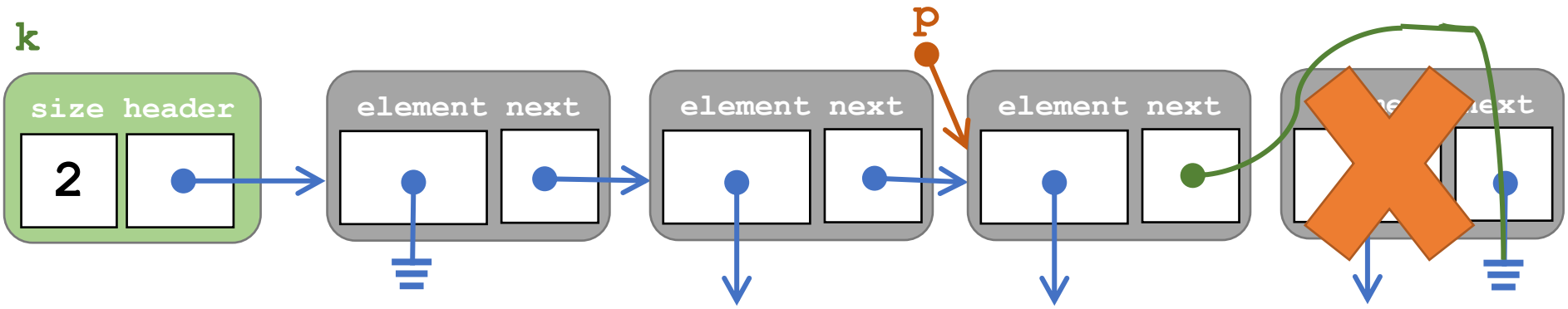
Remove at the end of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



Remove at the end of a list

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



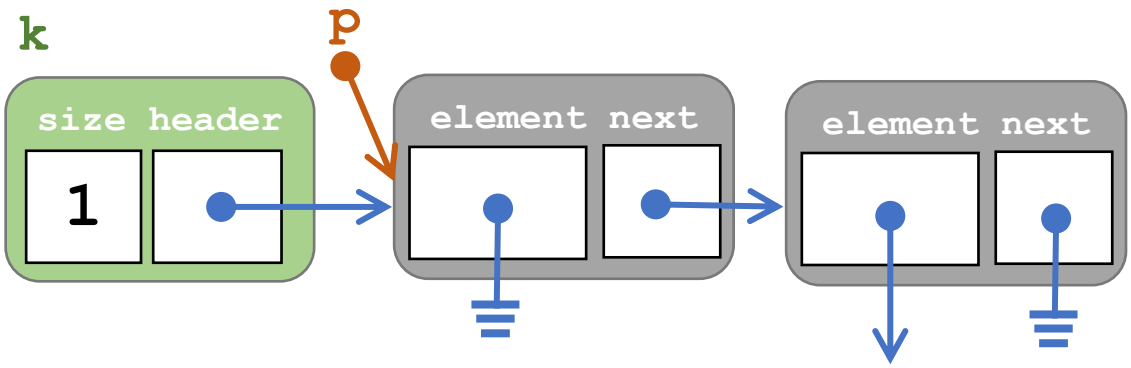
Remove from a list
with only one element

Example 4

Remove from a list with only one element

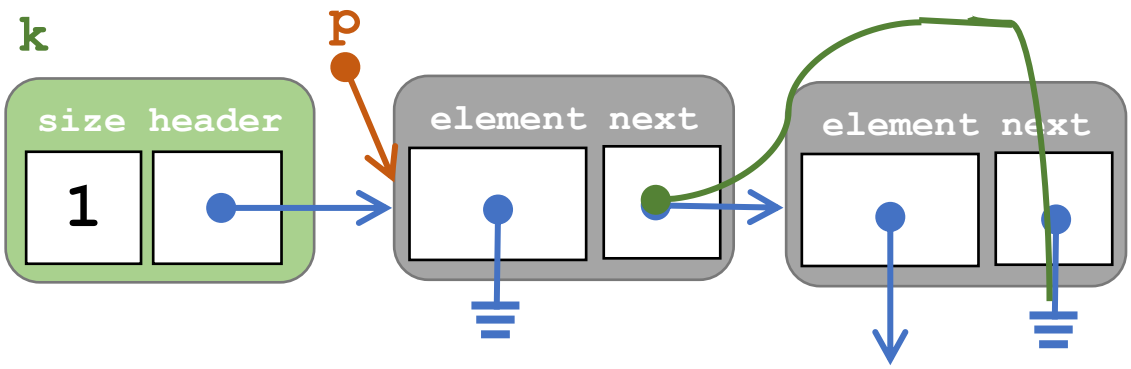
```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```

k.remove(0) calls nodeAt(-1), which returns this p, and k.removeAfter(p) is called.



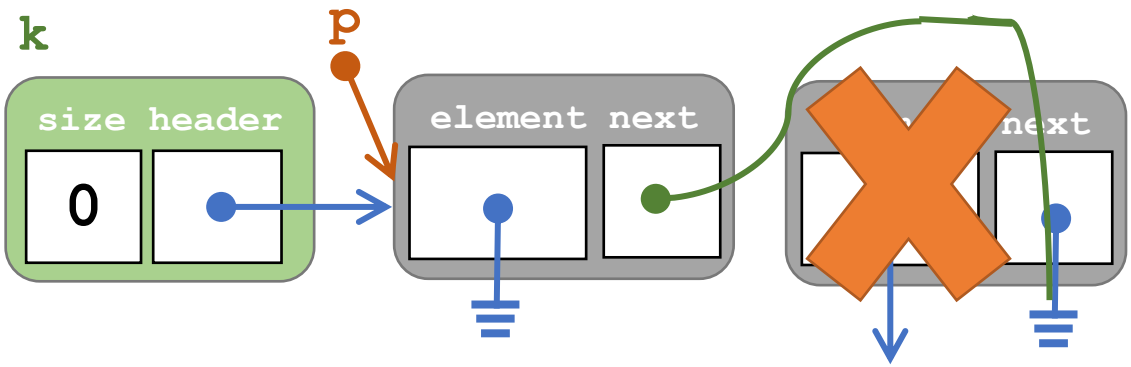
Remove from a list with only one element

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```



Remove from a list with only one element

```
public void remove(int i) {
    assertInRange(i, size-1);    LinkedNode p = nodeAt(i-1);
    removeAfter(p);
}
private void removeAfter(LinkedNode p) {
    if (p.next != null) {  p.next = p.next.next;  --size;  }
}
```

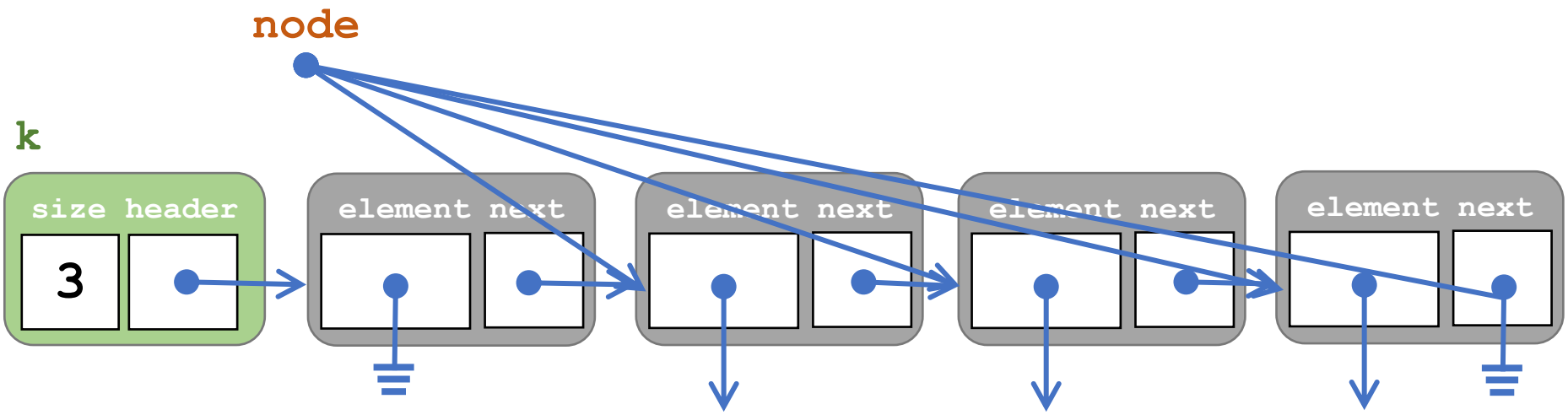


Method contains

Class SinglyLInkedList

contains: Class SinglyLinkedList

```
public boolean contains(Object e) {
    LinkedNode node = header.next;
    while (node != null && !node.element.equals(e))
        node = node.next;
    return node!= null;
}
```



Exercises

เขียน method

- concat ที่นำลิสต์มาต่อกัน
- clone ที่ copy ลิสต์เดิมมาใส่ลิสต์ใหม่
- insertAt ที่แทรก element e ที่ตำแหน่ง i ในลิสต์
- swap ที่ สลับ element ที่ i กับ j ในลิสต์
- removeAll ที่ลบ element ทุกตัวที่มีค่าเท่ากับค่าที่กำหนดออกจากลิสต์

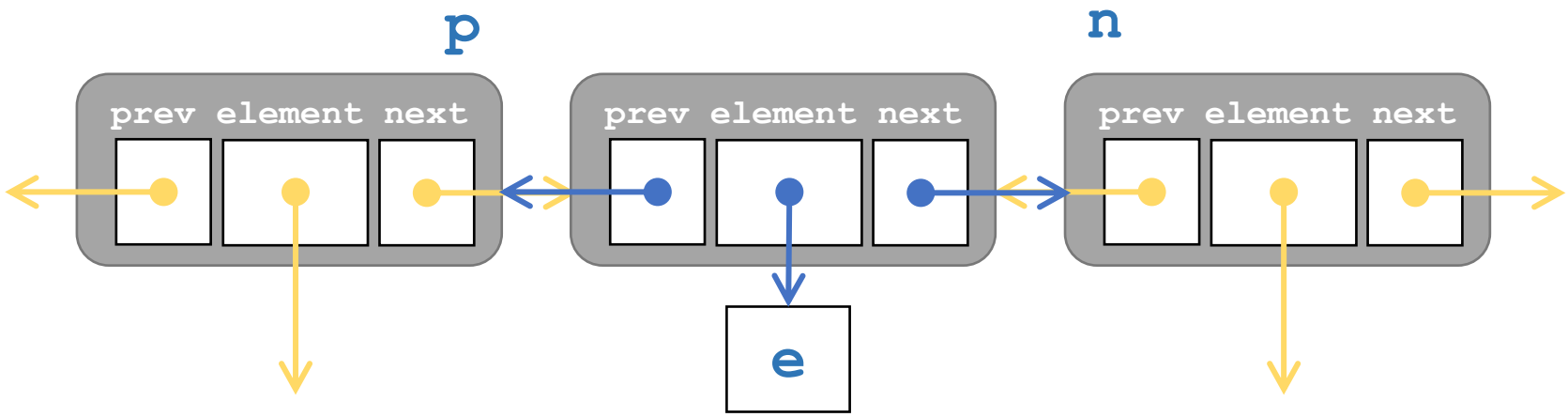
DoublyLinkedList

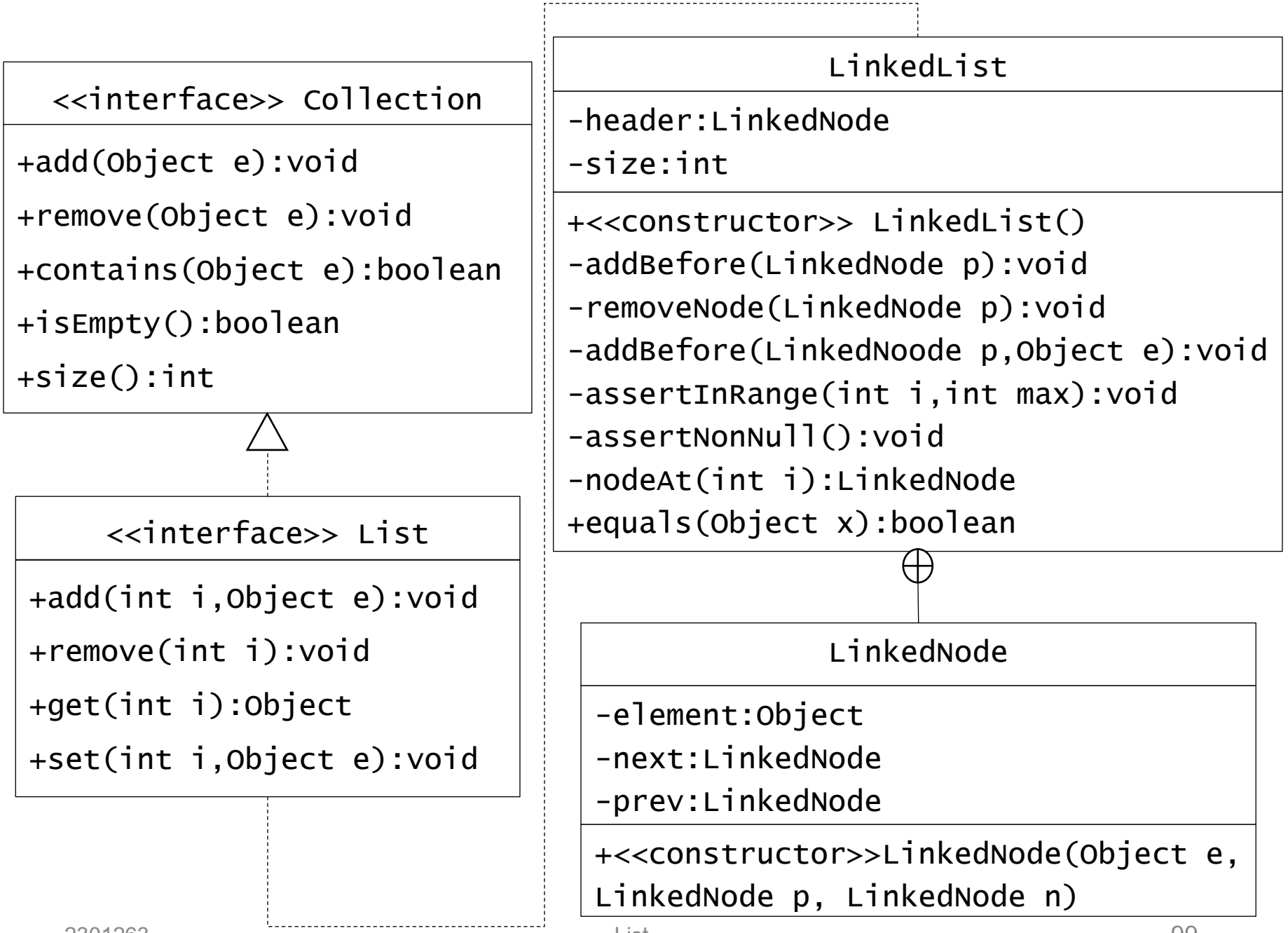
สร้าง List ที่ค้นย้อนได้

Class LinkedNode

โหนด

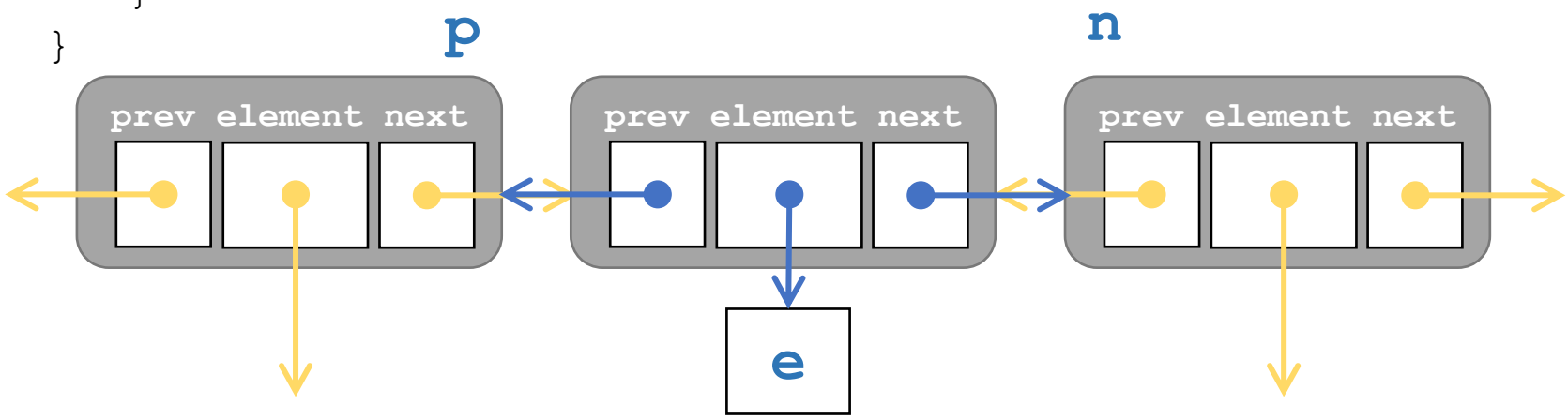
- Attribute **`element`** เก็บข้อมูลของโหนด
- Attribute **`prev`** ชี้ไปยังโหนดก่อนหน้า
- Attribute **`next`** ชี้ไปยังโหนดต่อไป




<<interface>> Collection
+add(Object e):void
+remove(Object e):void
+contains(Object e):boolean
+isEmpty():boolean
+size():int
<<interface>> List
+add(int i,Object e):void
+remove(int i):void
+get(int i):Object
+set(int i,Object e):void
LinkedList
-header:LinkedNode
-size:int
+<<constructor>> LinkedList()
-addBefore(LinkedNode p):void
-removeNode(LinkedNode p):void
-addBefore(LinkedNoode p,Object e):void
-assertInRange(int i,int max):void
-assertNonNull():void
-nodeAt(int i):LinkedNode
+equals(Object x):boolean
LinkedNode
-element:Object
-next:LinkedNode
-prev:LinkedNode
+<<constructor>>LinkedNode(Object e,
LinkedNode p, LinkedNode n)

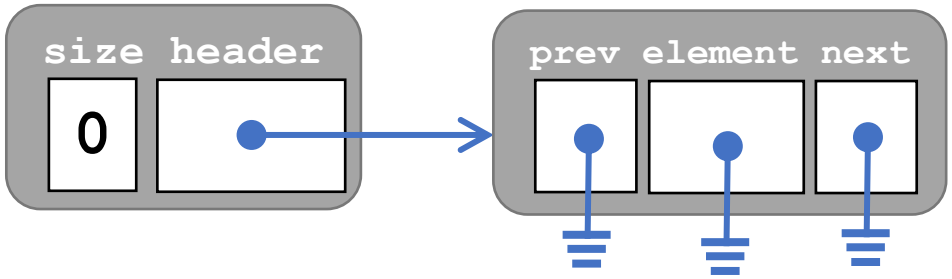
Class LinkedNode

```
private static class LinkedNode {
    private Object element;
    private LinkedNode prev;
    private LinkedNode next;

    LinkedNode(Object e, LinkedNode p, LinkedNode n) {
        this.element = e;
        this.prev = p;
        this.next = n;
    }
}
```



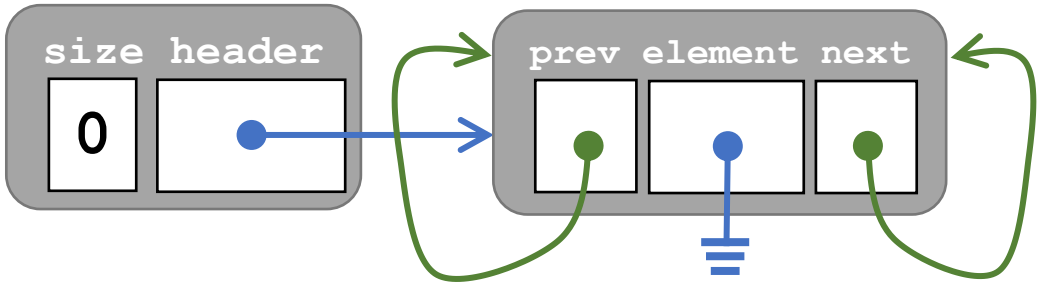
Create new object: Class LinkedList

```
public class LinkedList implements List {
    private int size;
    private LinkedNode header;

    public LinkedList() {
        size = 0;
        header = new LinkedNode(null, null, null);
        header.prev = header.next = header;
    }
```



Create new object: Class LinkedList

```
public class LinkedList implements List {
    private int size;
    private LinkedNode header;

    public LinkedList() {
        size = 0;
        header = new LinkedNode(null, null, null);
        header.prev = header.next = header;
    }
```



Methods in Class LinkedList

```
public class LinkedList implements List {
   private int size;
   private LinkedNode header;
   public LinkedList() {...}
   public int size() {...}          // interface Collection
   public boolean isEmpty {...}
   public boolean contains(Object e) {...}
   public void add(Object e) {...}
   public void remove(Object e) {...}
   private void removeAfter(LinkedNode p) {...}
   private static void assertInRange(int i, int max) {...}
   private static void assertNonNull(Object e) {...}
   private LinkedNode nodeAt(int i) {...}
   public Object get(int i) {...}    // interface List
   public void set(int i, Object e) {...}
   public void add(int i, Object e) {...}
   public void remove(int i) {...}
}
```

Methods in Class LinkedList

```
public class LinkedList implements List {
    private int size;
    private LinkedNode header;
    public LinkedList() {...}
    public int size() {...}              // interface Collection
    public boolean isEmpty {...}
    public boolean contains(Object e) {...}
    public void add(Object e) {...}
    public void remove(Object e) {...}
    private void removeAfter(LinkedNode p) {...}
    private static void assertInRange(int i, int max) {...}
    private static void assertNonNull(Object e) {...}
    private LinkedNode nodeAt(int i) {...}
    public Object get(int i) {...}    // interface List
    public void set(int i, Object e) {...}
    public void add(int i, Object e) {...}
    public void remove(int i) {...}
}
```

Methods in Class LinkedList

```
public class LinkedList implements List {
    private int size;
    private LinkedNode header;
    public LinkedList() {...}
    public int size() {...}          // interface Collection
    public boolean isEmpty {...}
    public boolean contains(Object e) {...}
    public void add(Object e) {...}
    public void remove(Object e) {...}
    private void removeAfter(LinkedNode p) {...}
    private static void assertInRange(int i, int max) {...}
    private static void assertNonNull(Object e) {...}
    private LinkedNode nodeAt(int i) {...}
    public Object get(int i) {...}    // interface List
    public void set(int i, Object e) {...}
    public void add(int i, Object e) {...}
    public void remove(int i) {...}
}
```

Some simple methods (Same as in SinglyLinkedList)

```
public int size() {
   return size;
}

public boolean isEmpty() {
   return size==0;
}

public boolean contains(Object e) {
   LinkedNode node = header.next;
   while (node != null && !node.element.equals(e))
      node = node.next;
   return node!= null;
}
```

Method add

Class LinkedList

Add: Class LinkedList

```
public void add(Object e) {
    addBefore(header, e);
}
public void add(int i, Object e) {
    assertInRange(i, size);
    addBefore(nodeAt(i), e);
}
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```

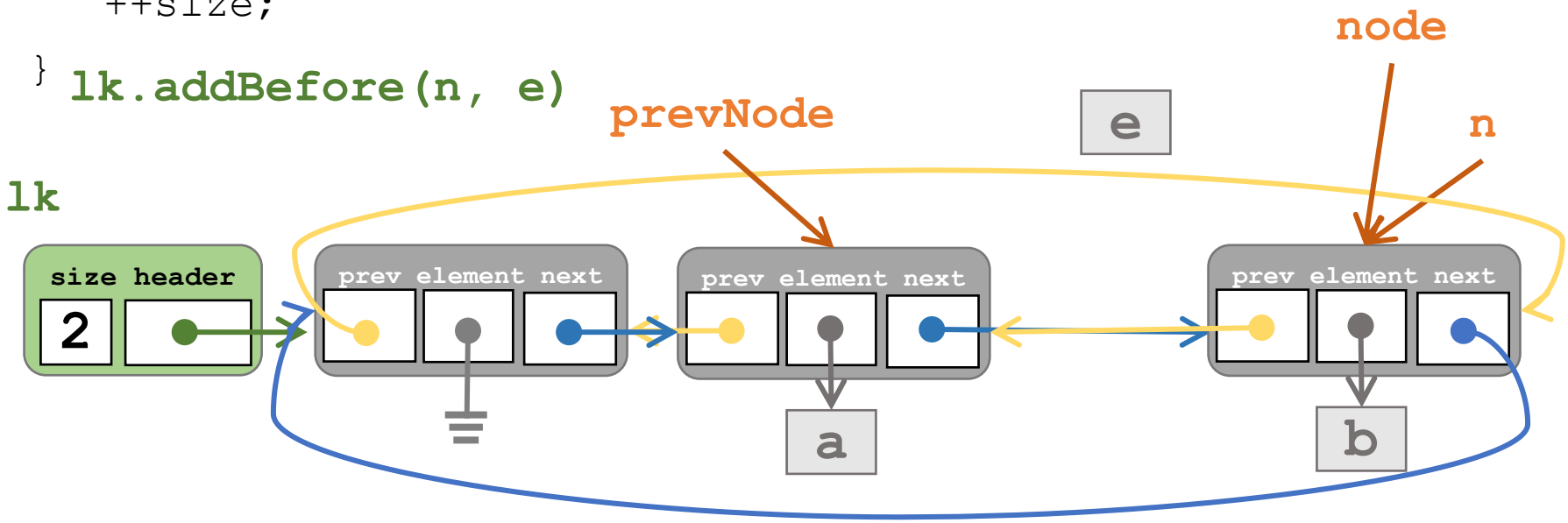
Add a node in the middle of the list

Example 1

addBefore

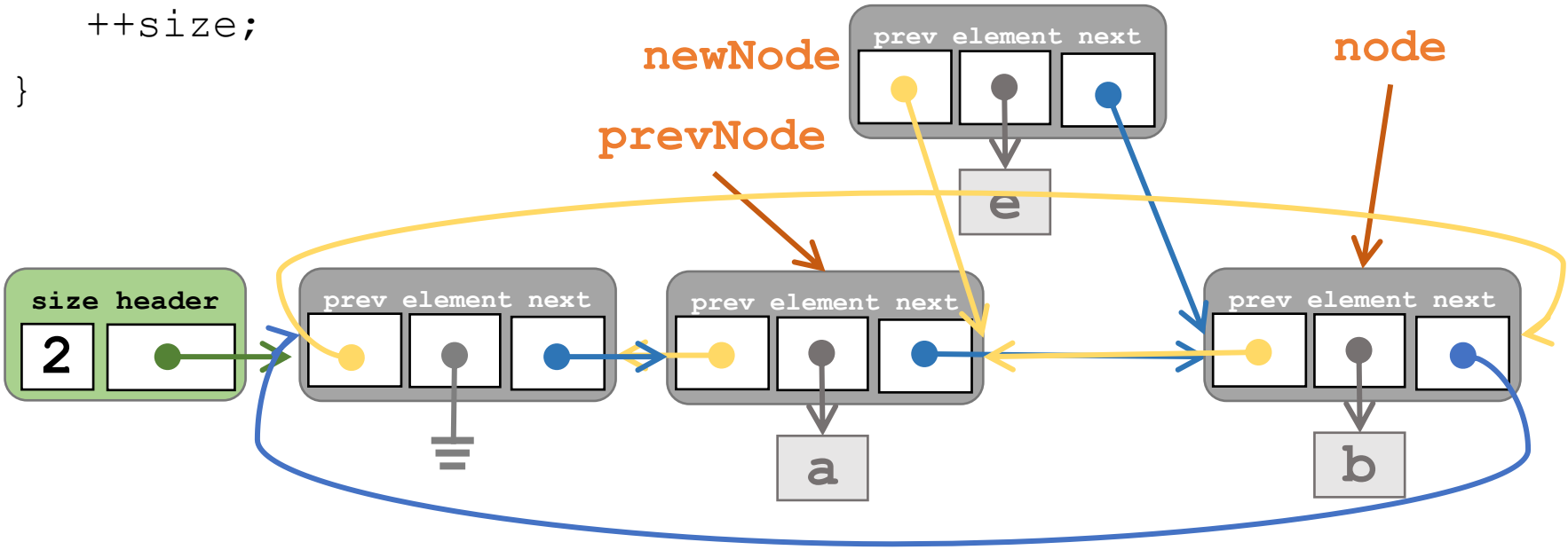
```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```

lk.addBefore(n, e)



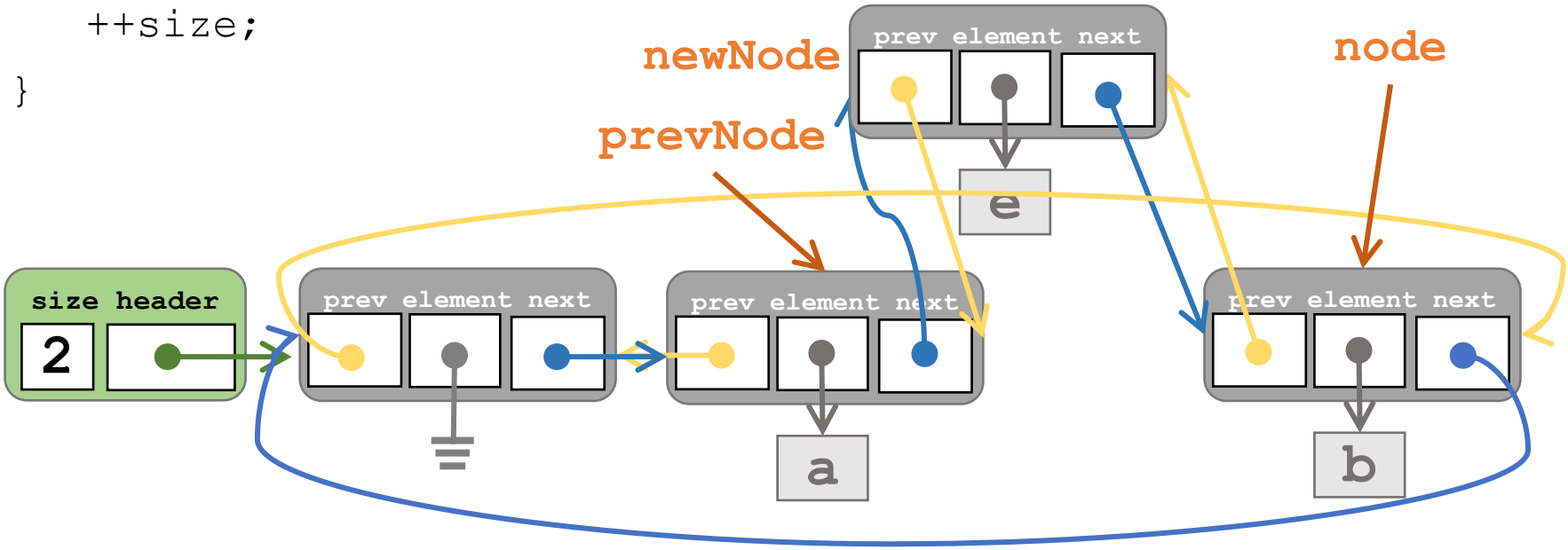
add: Class LinkedList

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



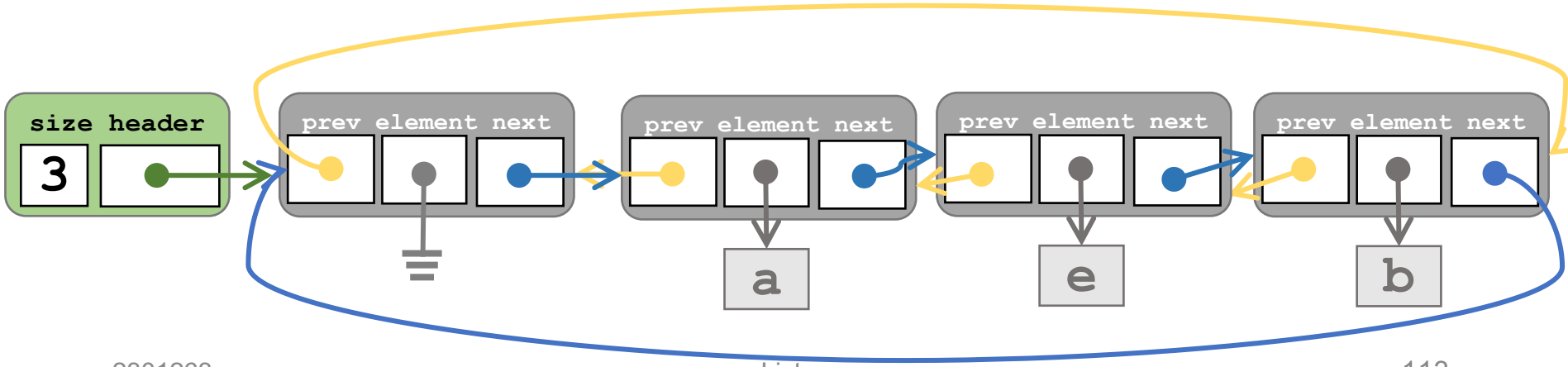
add: Class LinkedList

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



add: Class LinkedList

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



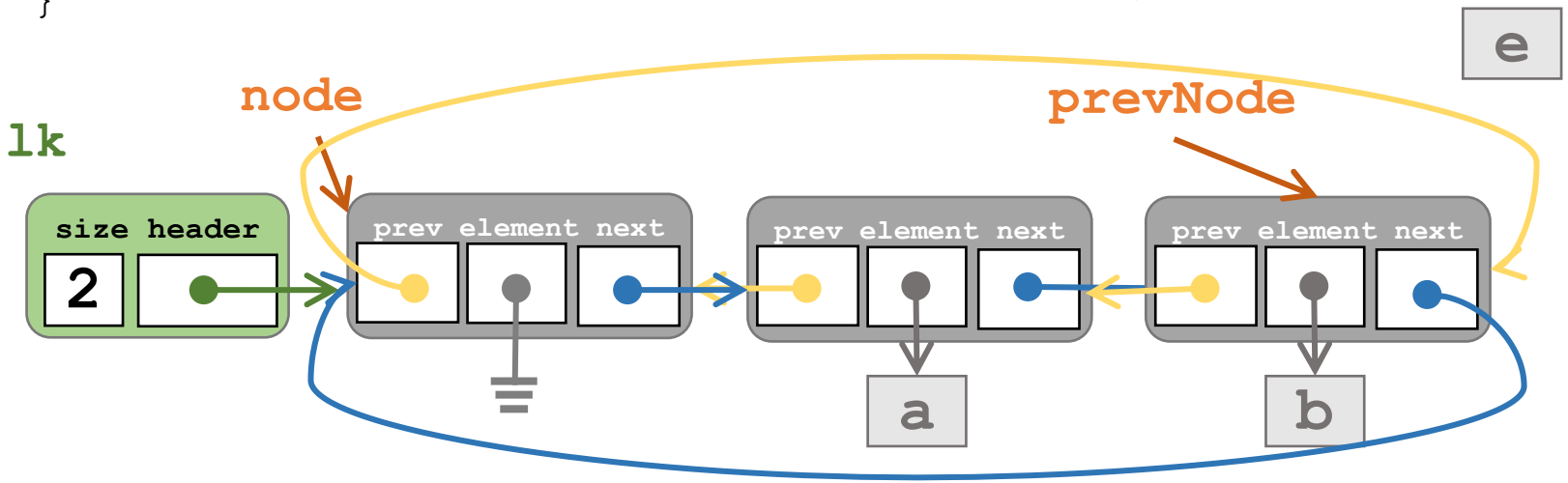
Add a node at the end of the list

Example 2

addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```

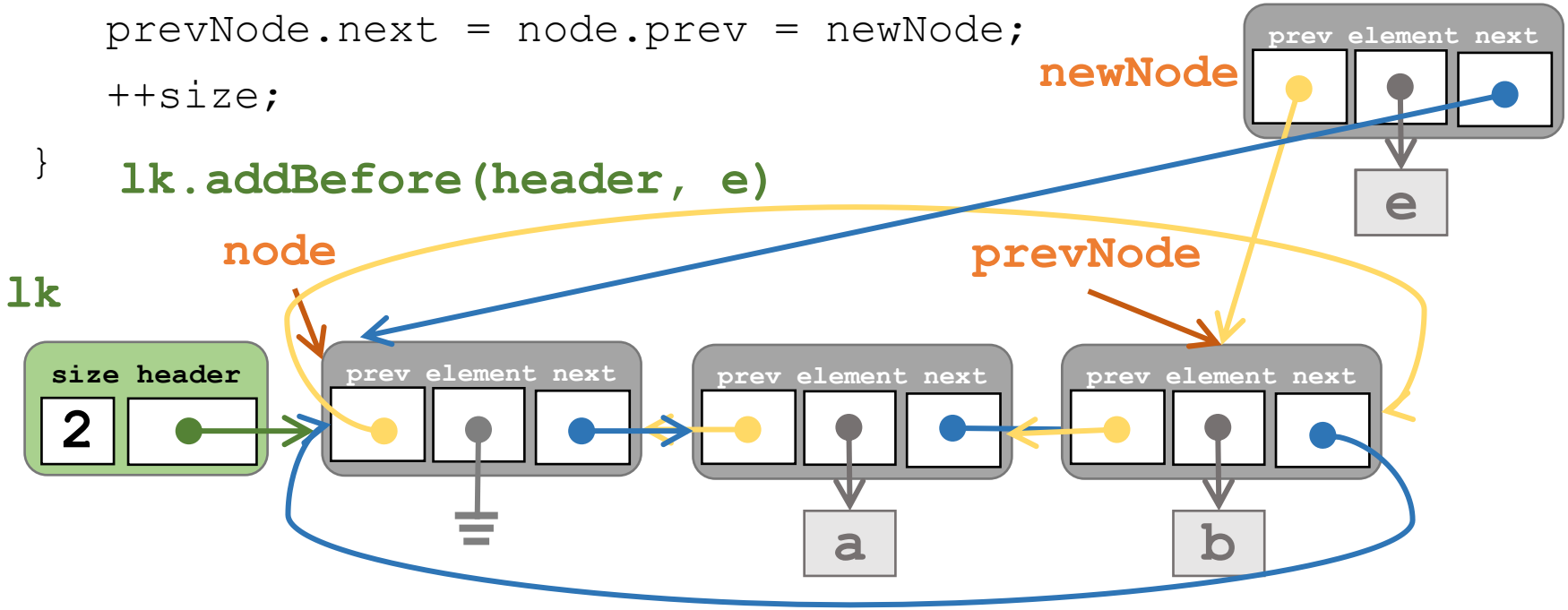
lk.addBefore(header, e)



addBefore

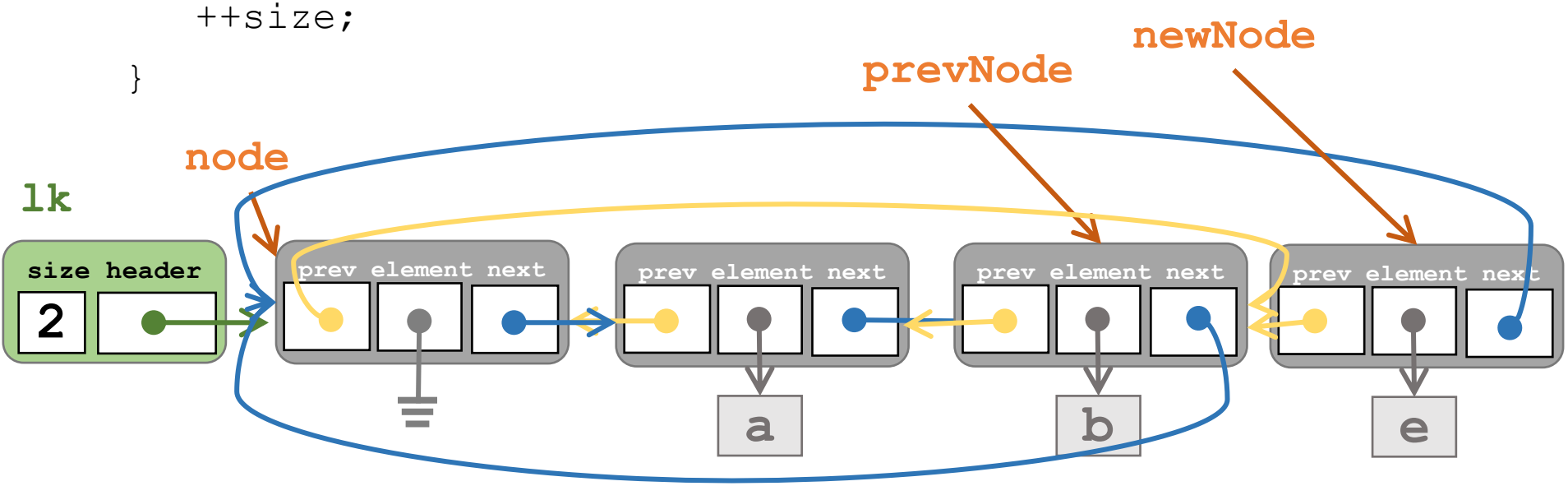
```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```

lk.addBefore(header, e)



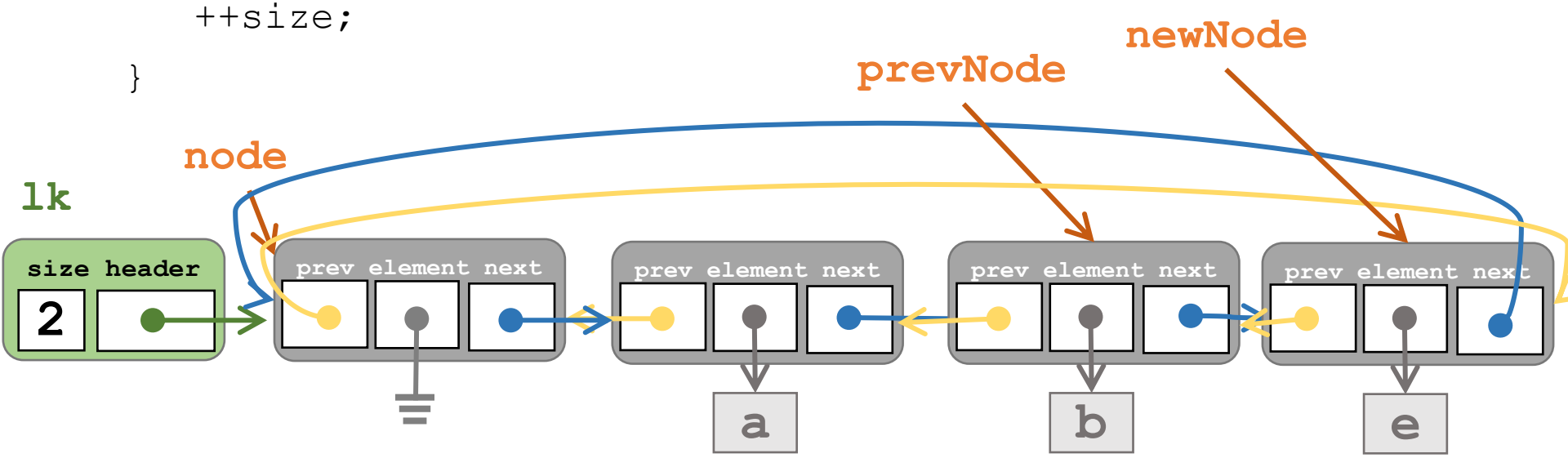
addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



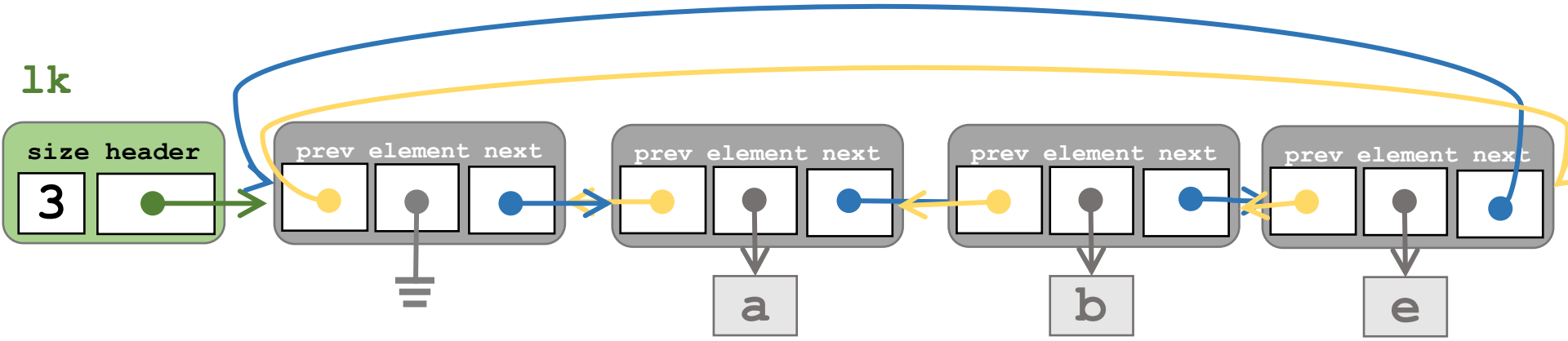
addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```

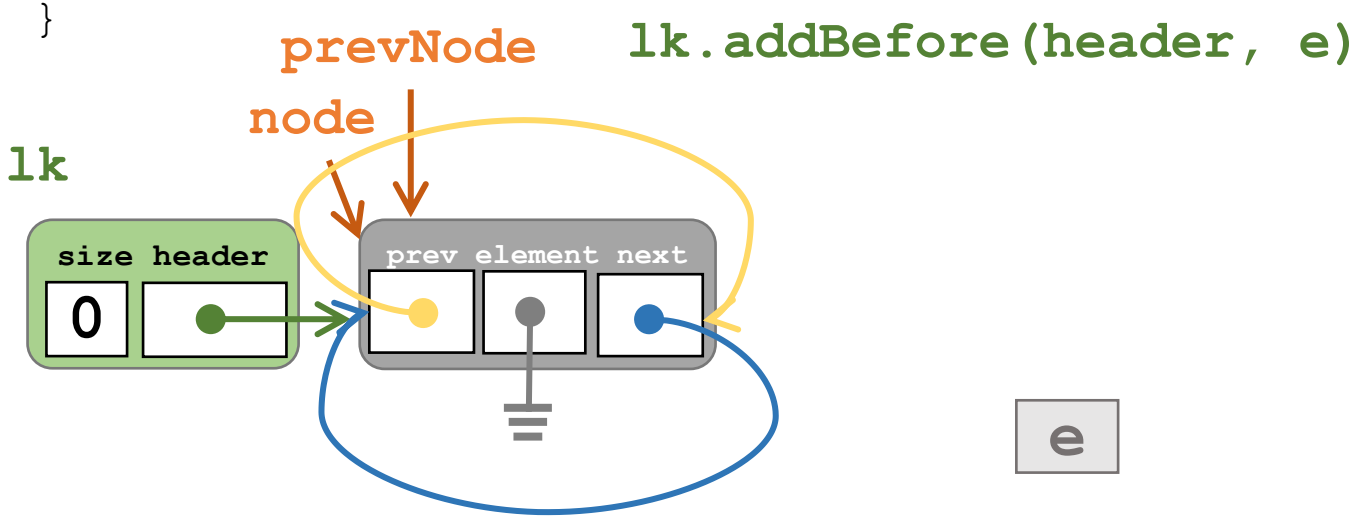


Add a node in an empty list

Example 3

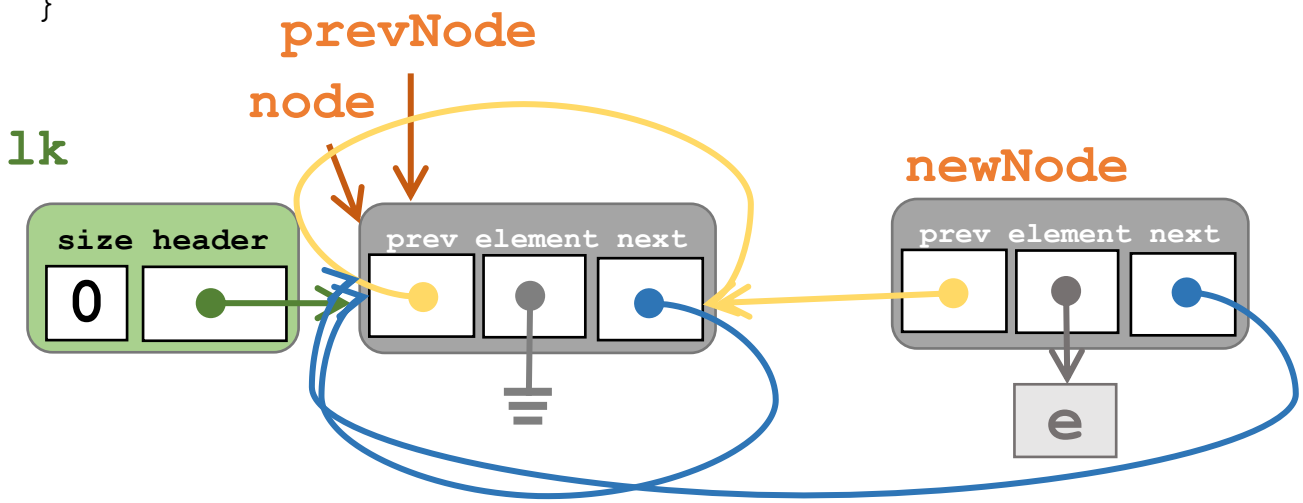
addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



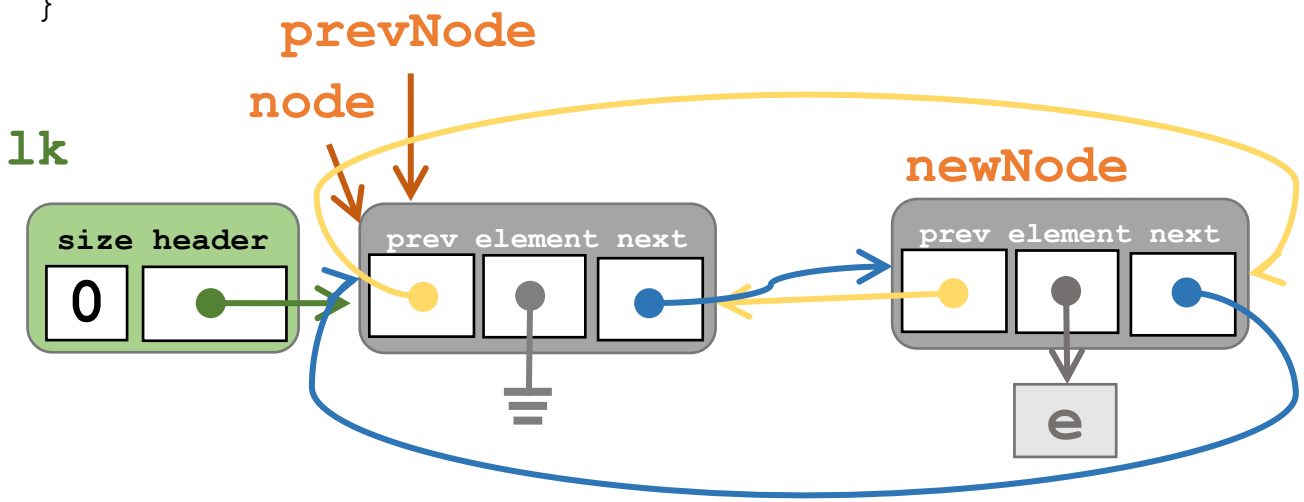
addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



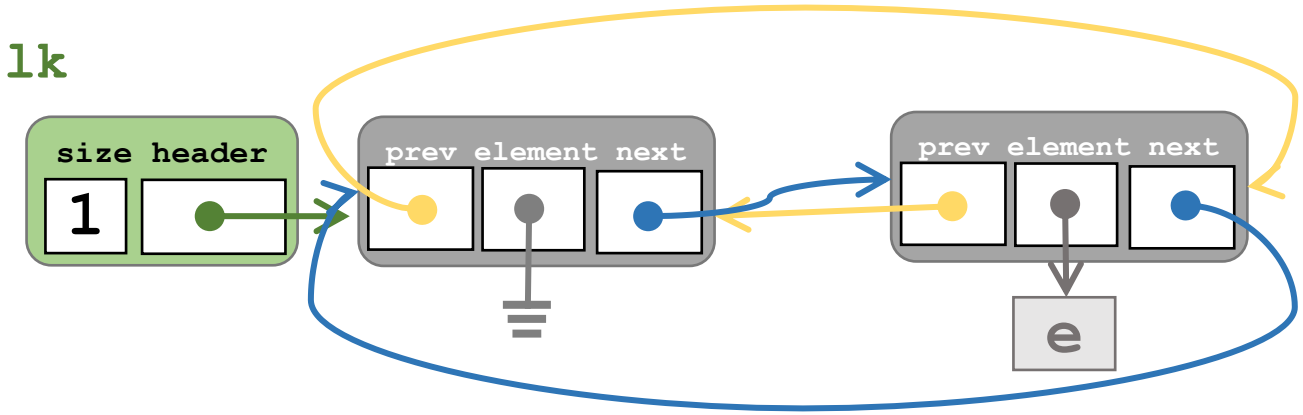
addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



addBefore

```
private void addBefore(LinkedNode node, Object e) {
    assertNonNull(e);
    LinkedNode prevNode = node.prev;
    LinkedNode newNode = new LinkedNode(e, prevNode, node);
    prevNode.next = node.prev = newNode;
    ++size;
}
```



Method remove

Class LinkedList

remove: Class LinkedList

```
public void remove(Object e) {
    LinkedNode node = nodeOf(e);
    if (node!=header) removeNode(node);
}
public void remove(int i) {
    assertInRange(i, size);     removeNode(nodeAt(i));
}
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode; nextNode.prev = prevNode;
    --size;
}
```

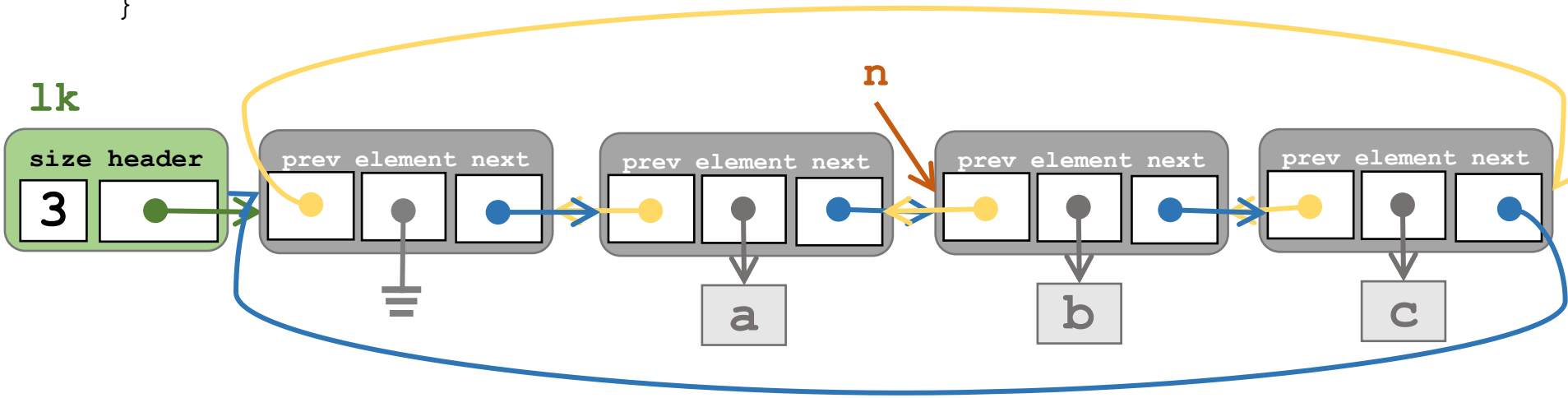
Remove a node in the middle of the list

Example 1

removeNode: Class LinkedList

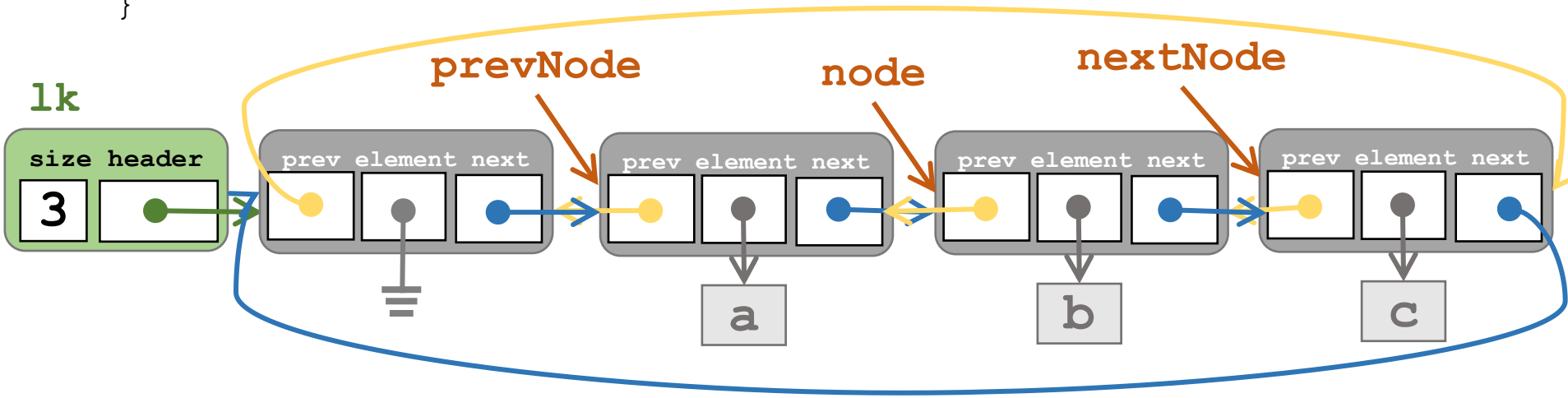
```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

lk.removeNode(n)



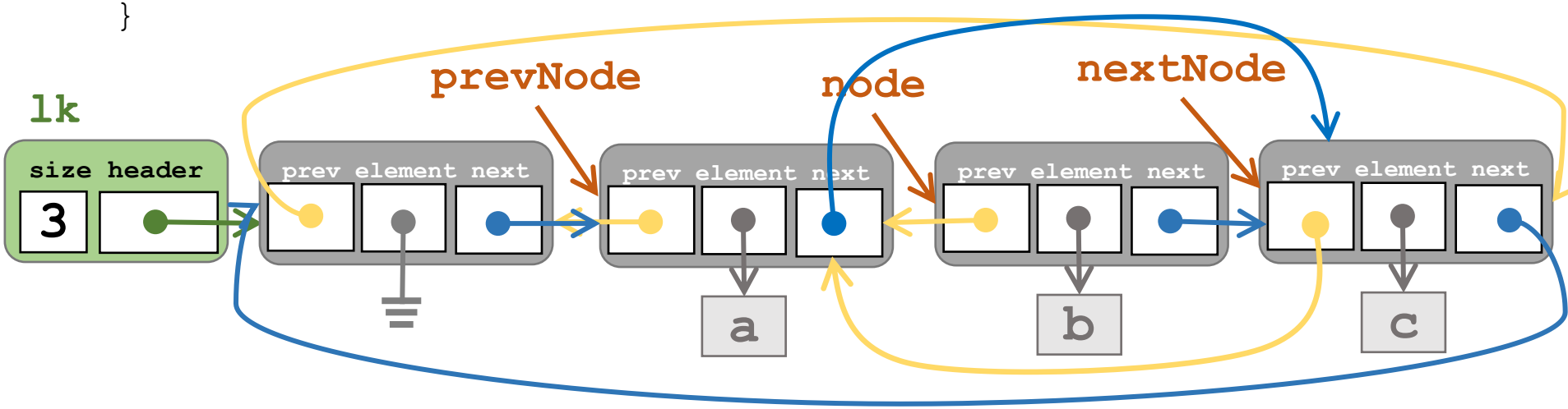
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



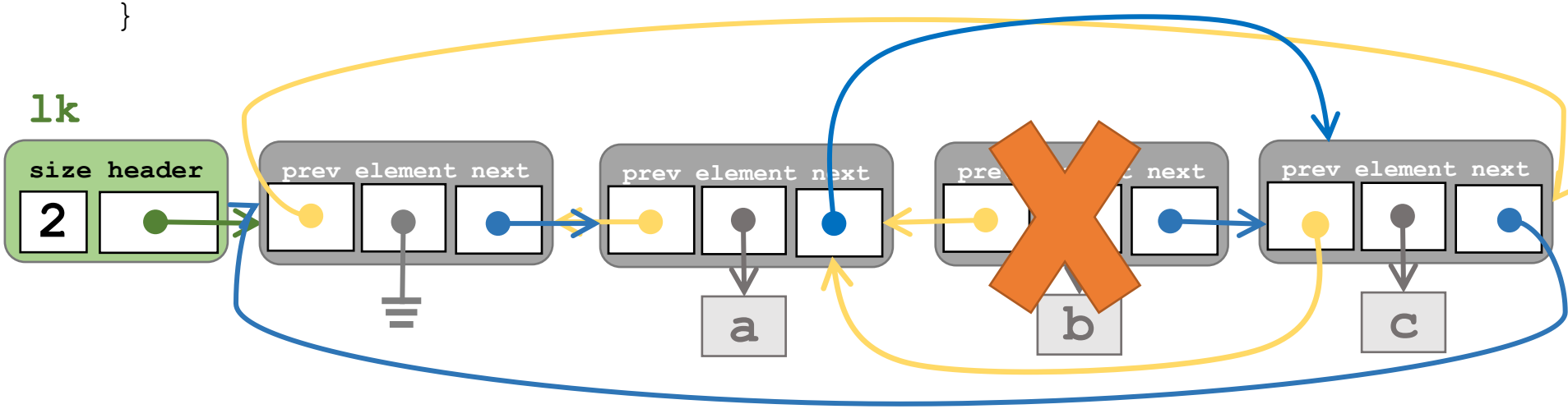
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

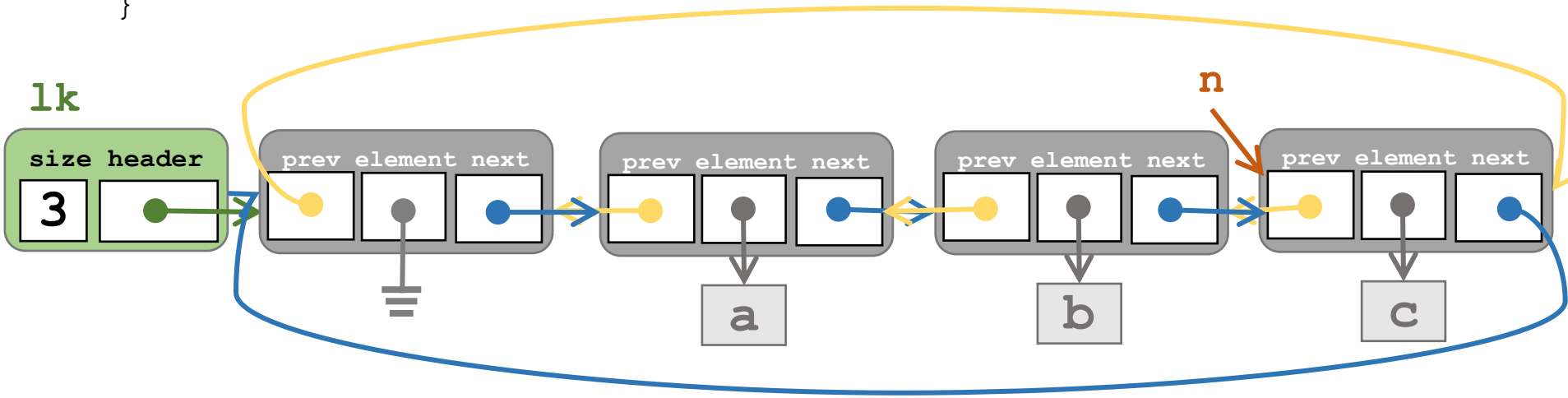


Remove a node at the end of the list

Example 2

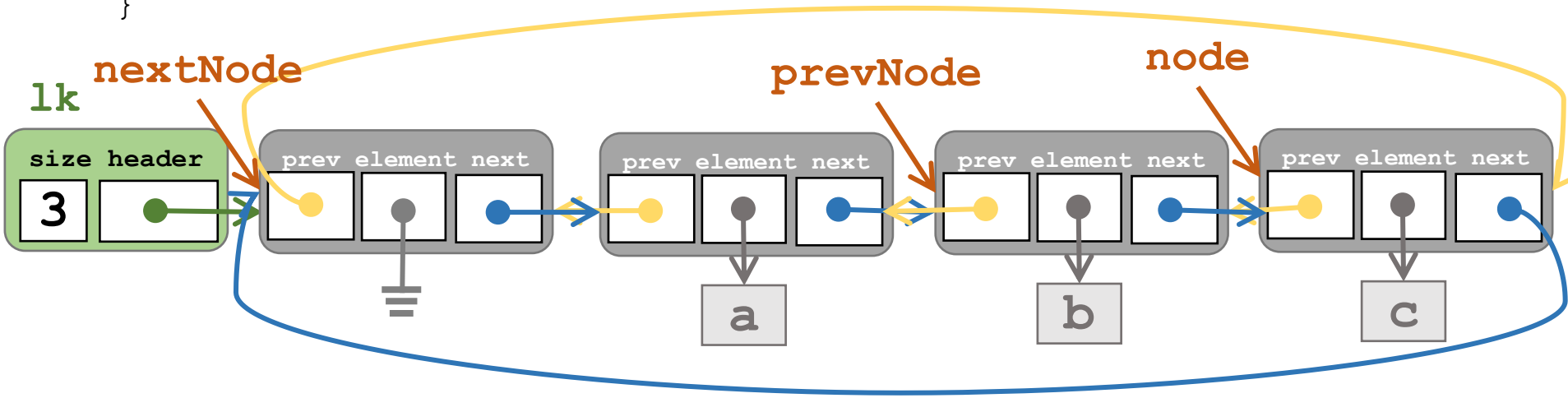
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



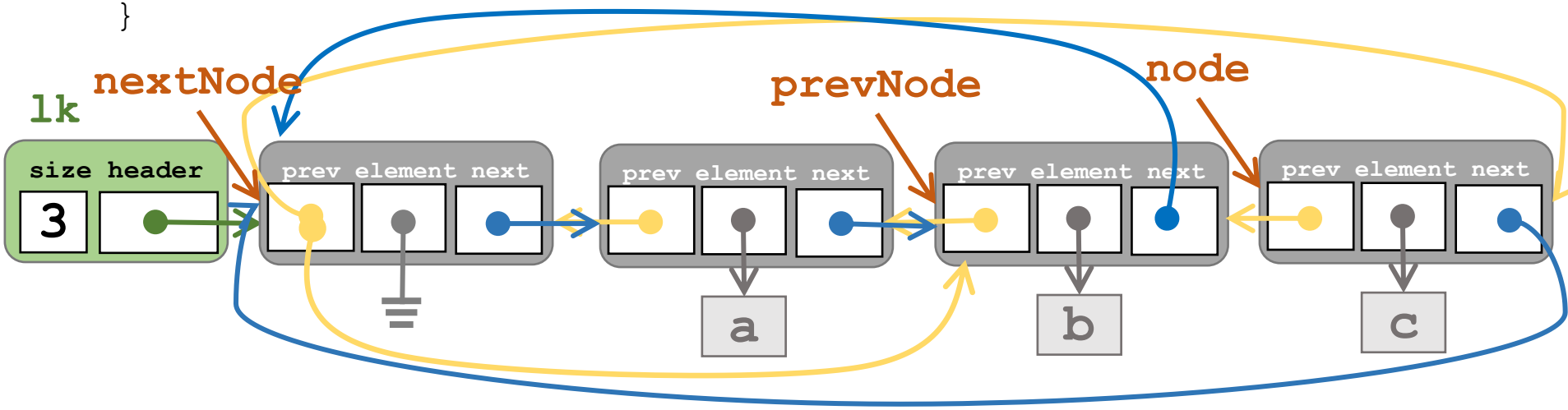
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



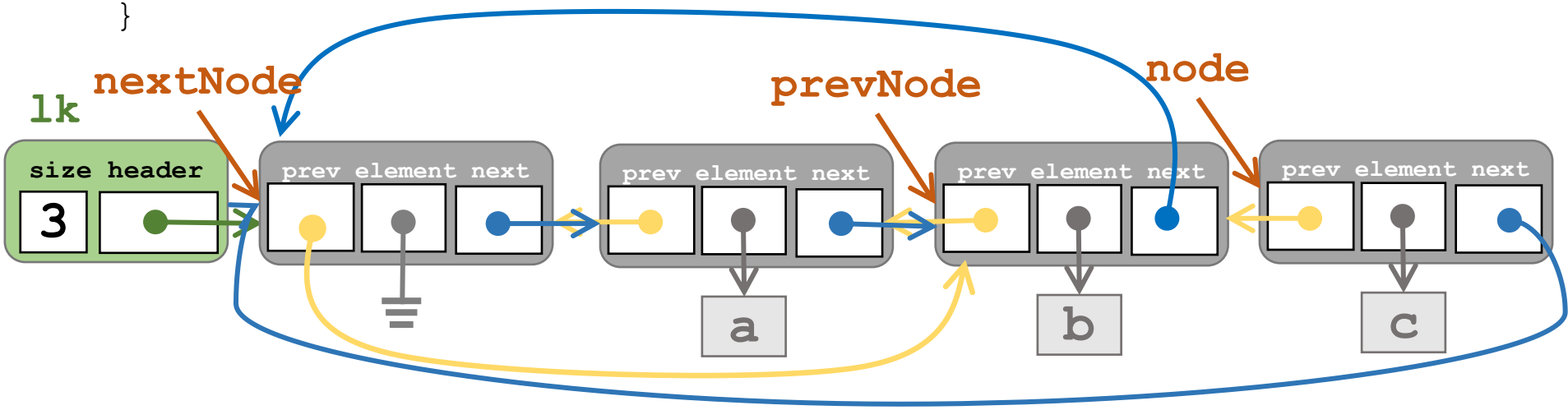
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



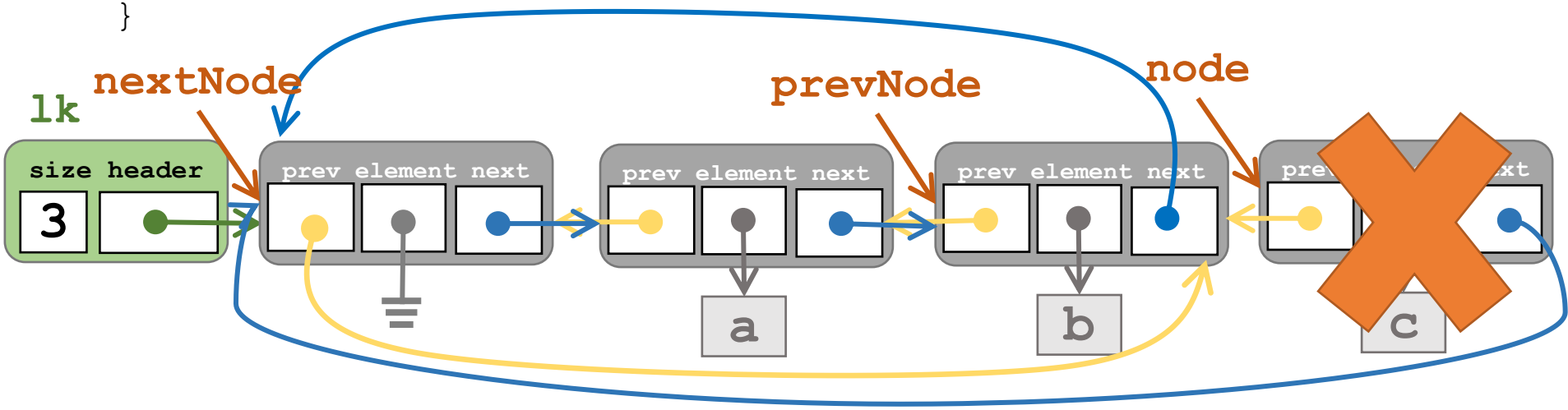
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



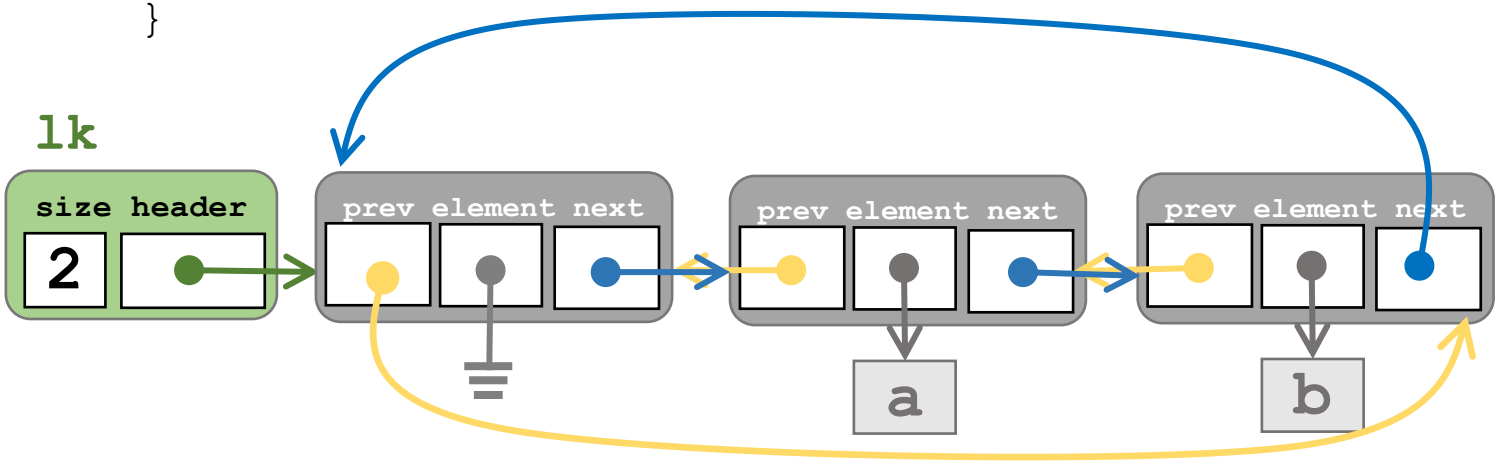
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

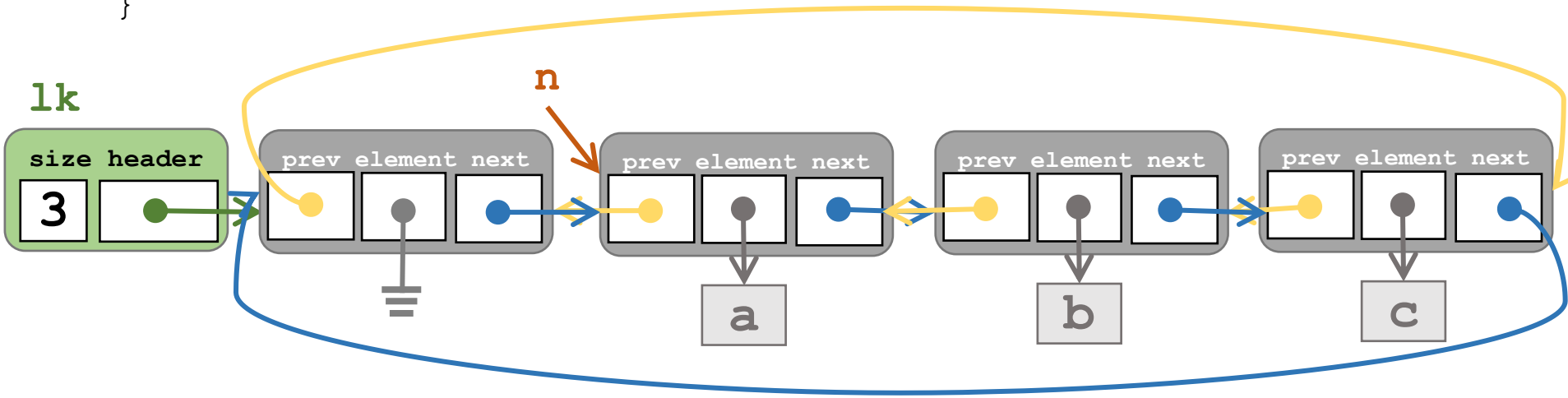


Remove a node at the head of the list

Example 3

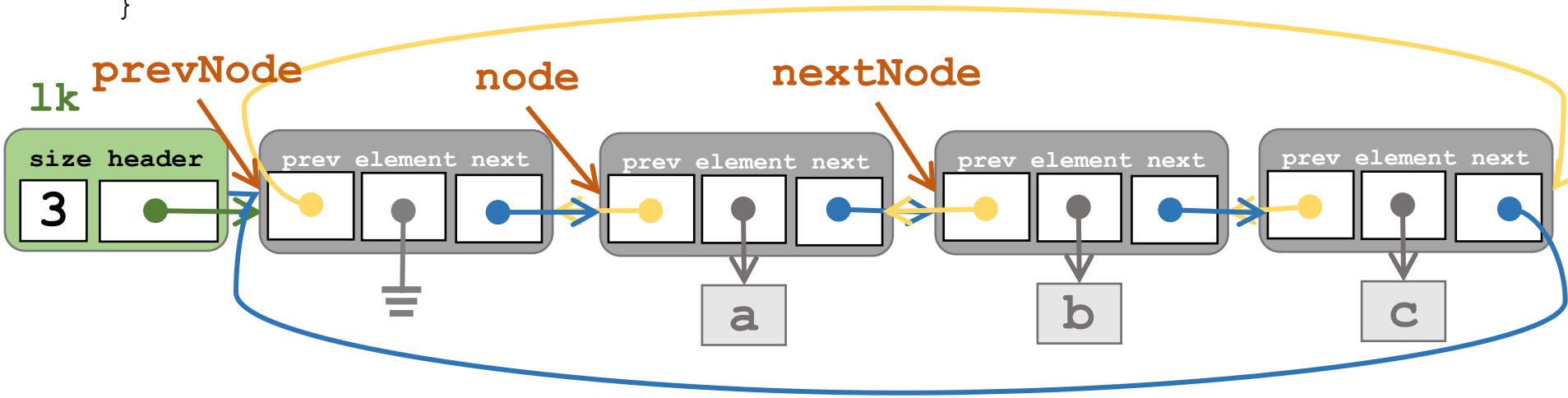
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



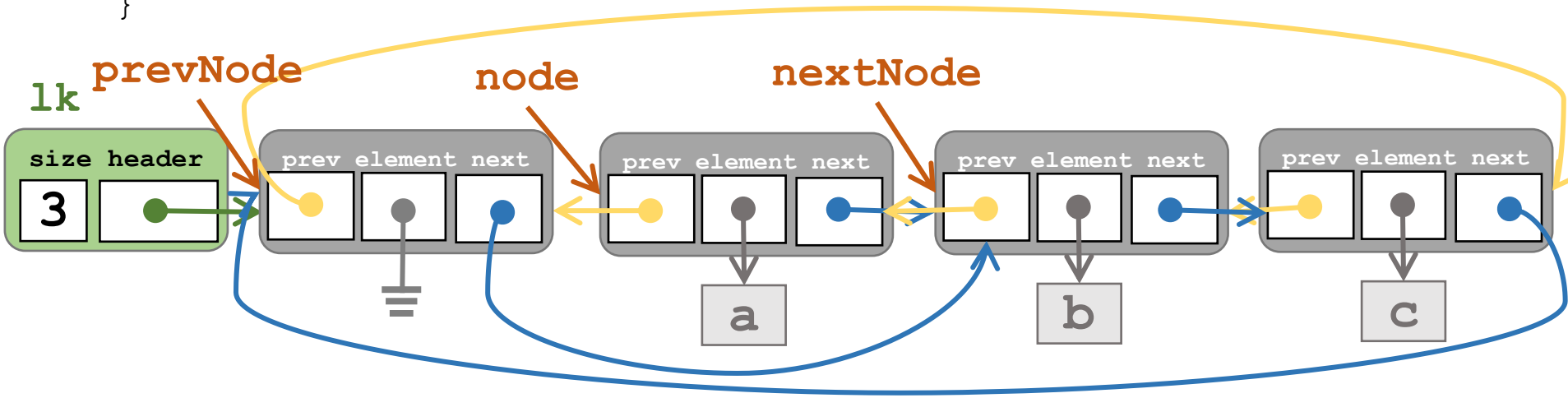
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



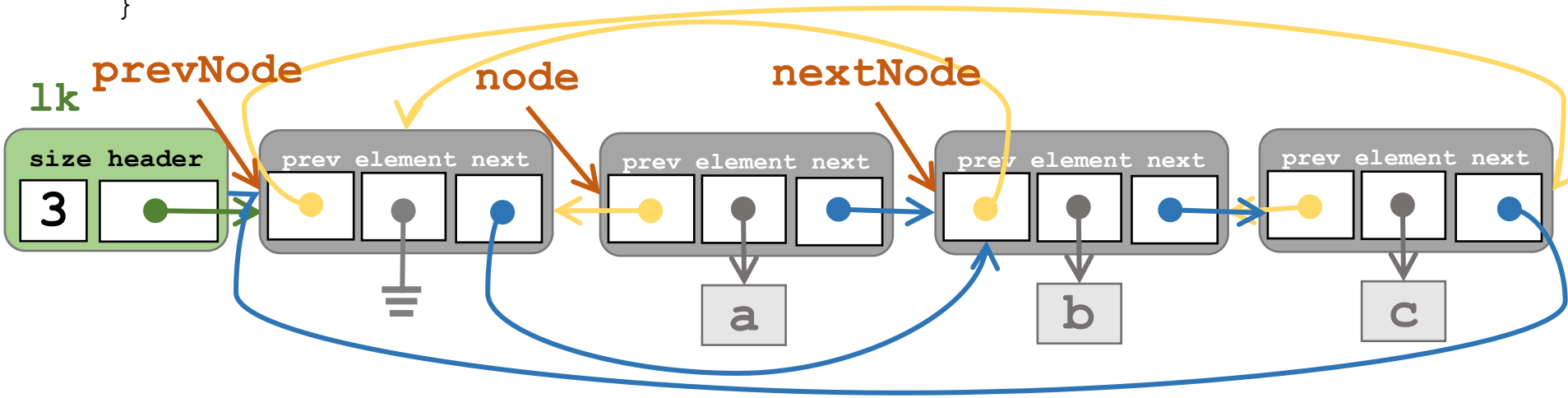
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



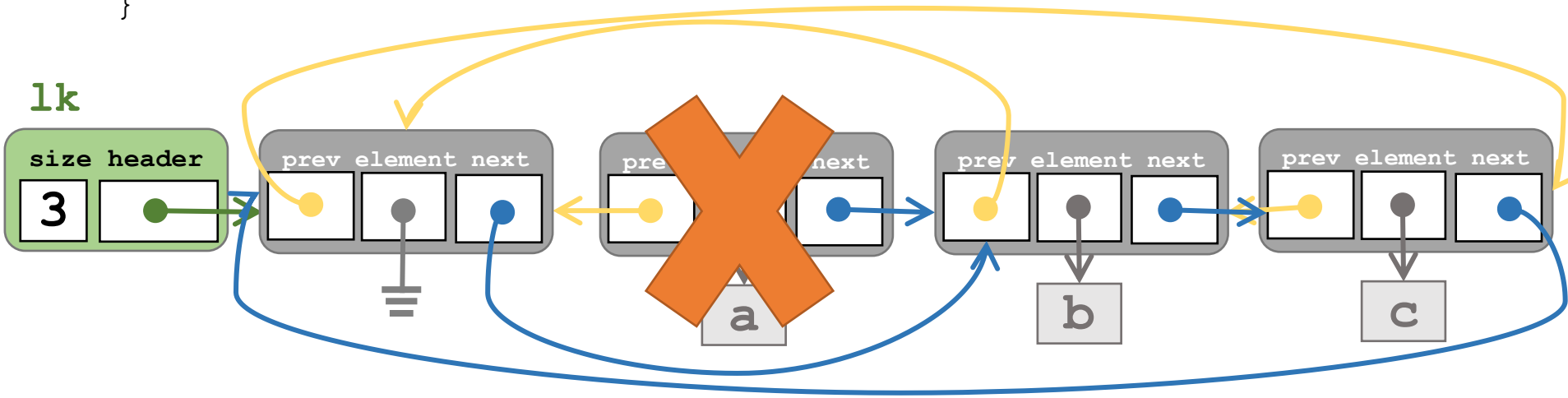
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



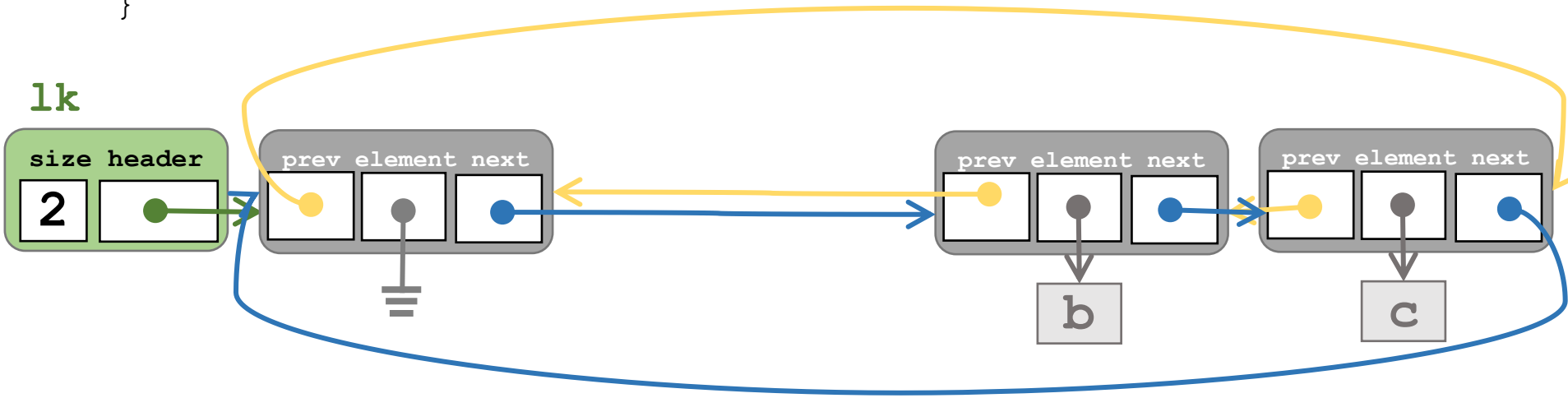
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



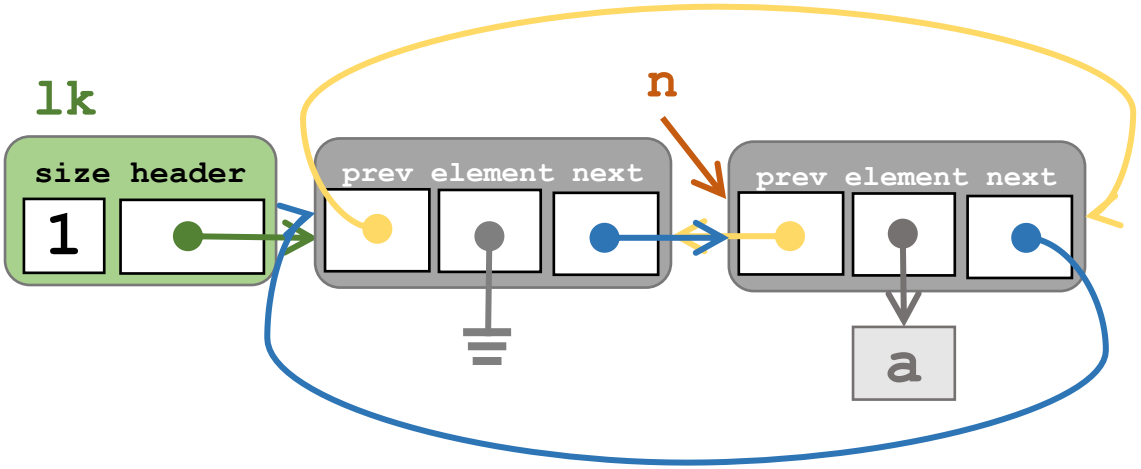
Remove a node from a list with one element

Example 4

removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

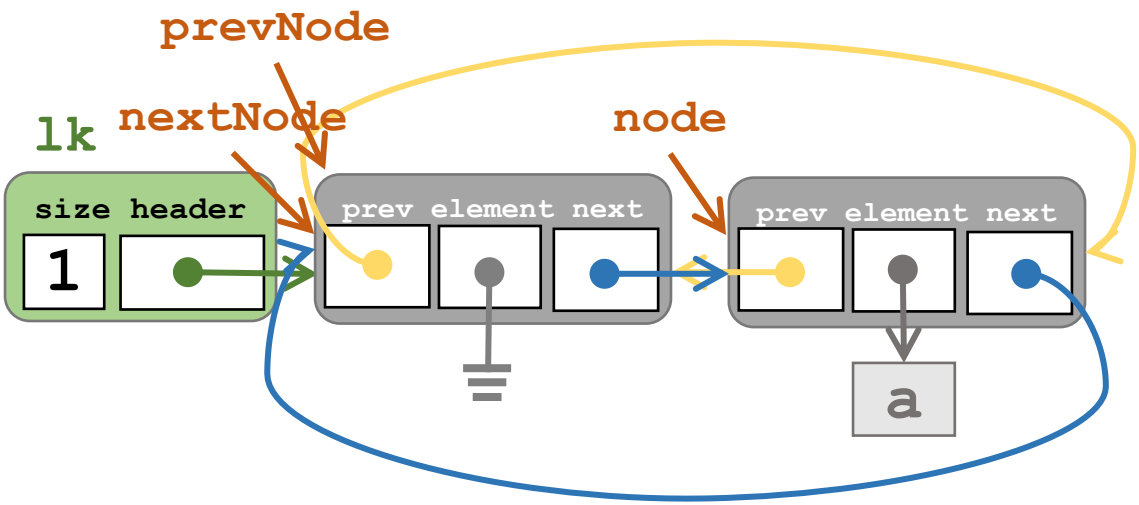
lk.removeNode(n)



removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

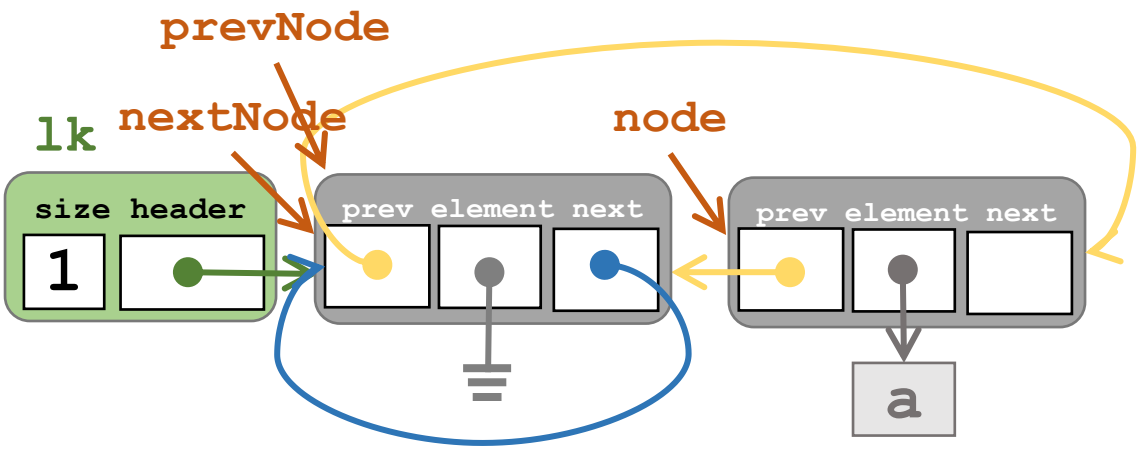
lk.removeNode(n)



removeNode: Class LinkedList

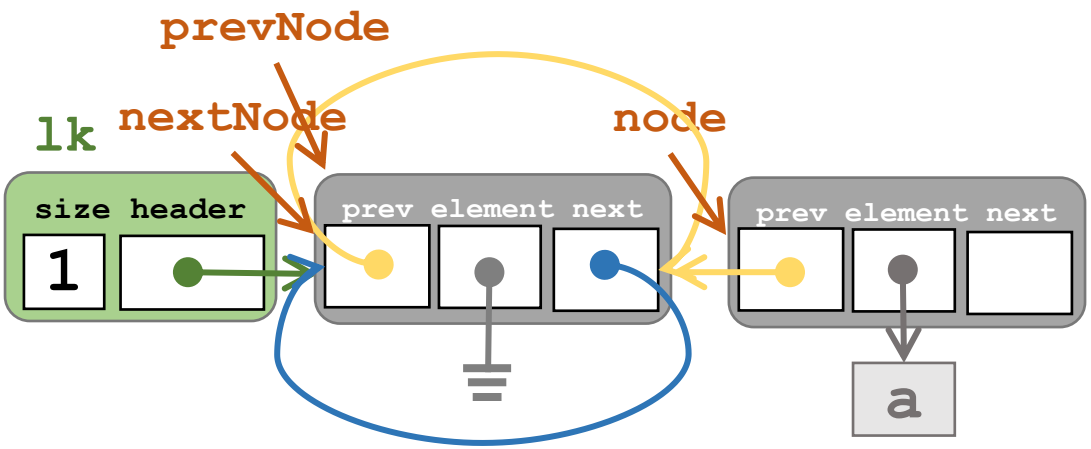
```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```

lk.removeNode(n)



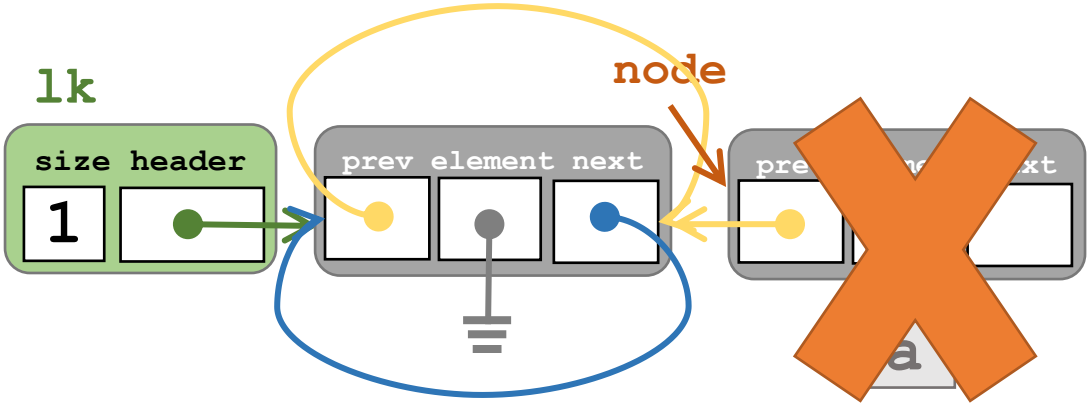
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



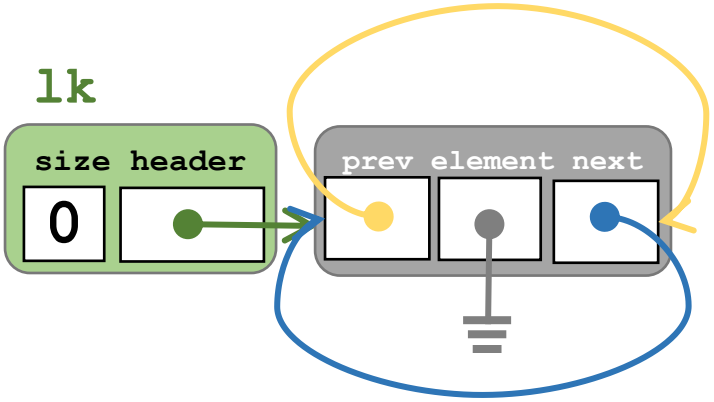
removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



removeNode: Class LinkedList

```
private void removeNode(LinkedNode node) {
    LinkedNode prevNode = node.prev;
    LinkedNode nextNode = node.next;
    prevNode.next = nextNode;
    nextNode.prev = prevNode;
    --size;
}
```



Exercises

เขียน `method`

- `concat` ที่นำลิสต์มาต่อกัน
- `clone` ที่ `copy` ลิสต์เดิมมาใส่ลิสต์ใหม่
- `insertAt` ที่แทรก `element e` ที่ตำแหน่ง `i` ในลิสต์
- `swap` ที่ สลับ `element` ที่ `i` กับ `j` ในลิสต์
- `removeAll` ที่ลบ `element` ทุกตัวที่มีค่าเท่ากับค่าที่กำหนดออกจากลิสต์