Listen
Algorithmen und Datenstrukturen 2

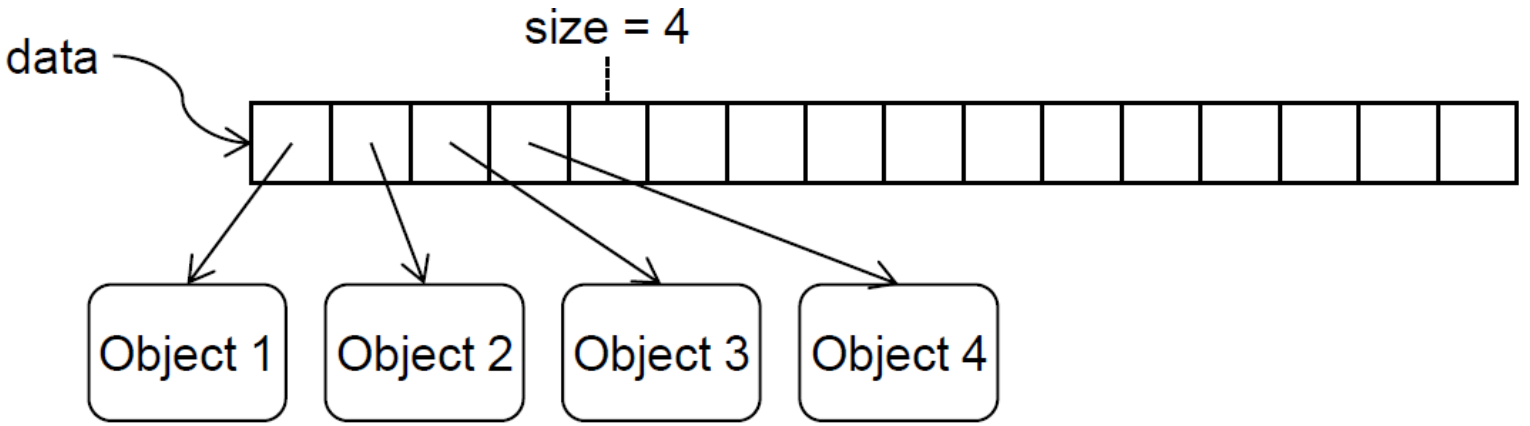
- Grüne Farbe: Bitte im Script nachtragen

Programm

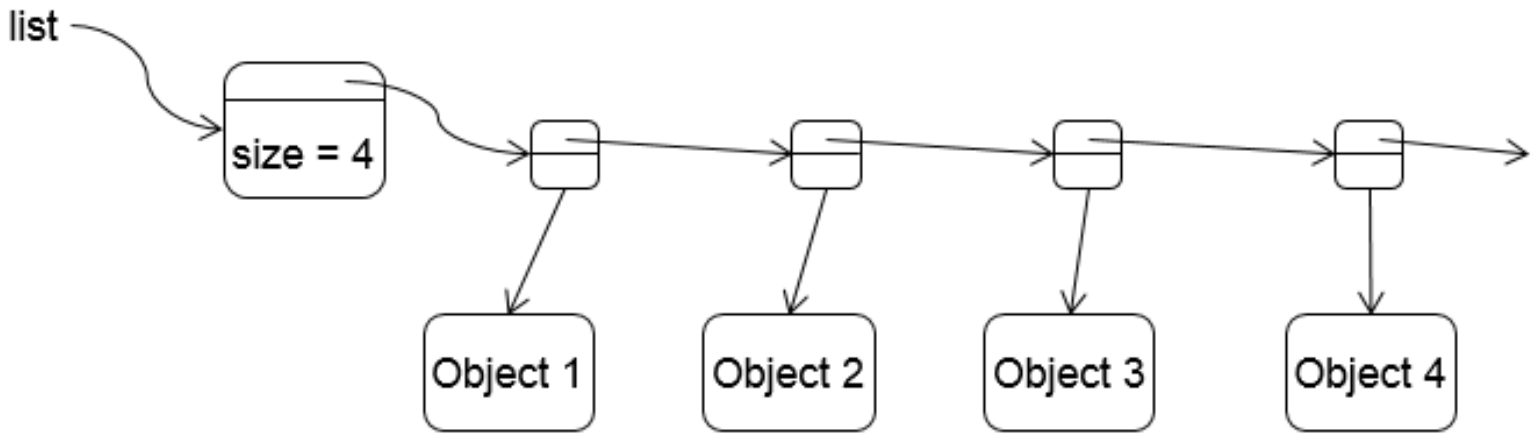
- Einführung Listen
- Implementation Listen
- Listen im Java Collection Framework
- Stack / Queues

Arrays und Listen im Vergleich (Struktur)

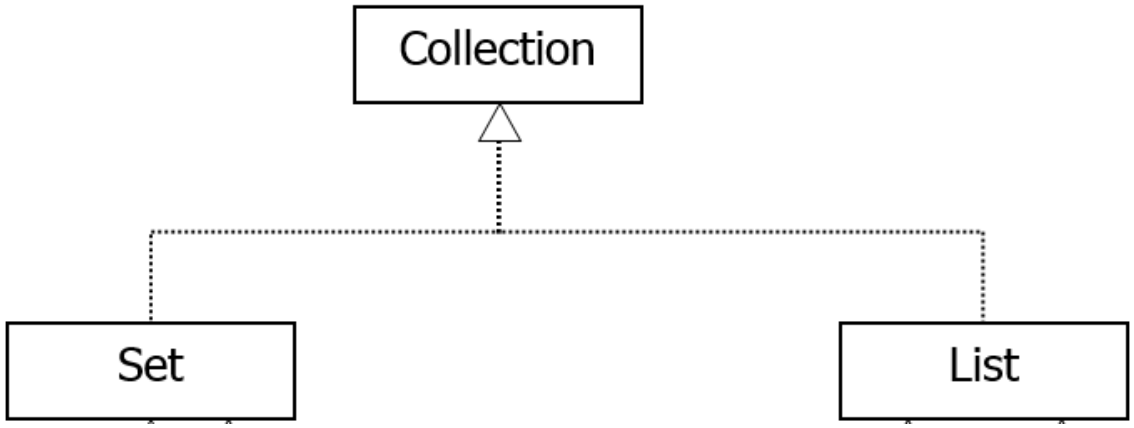
Array:



Liste:

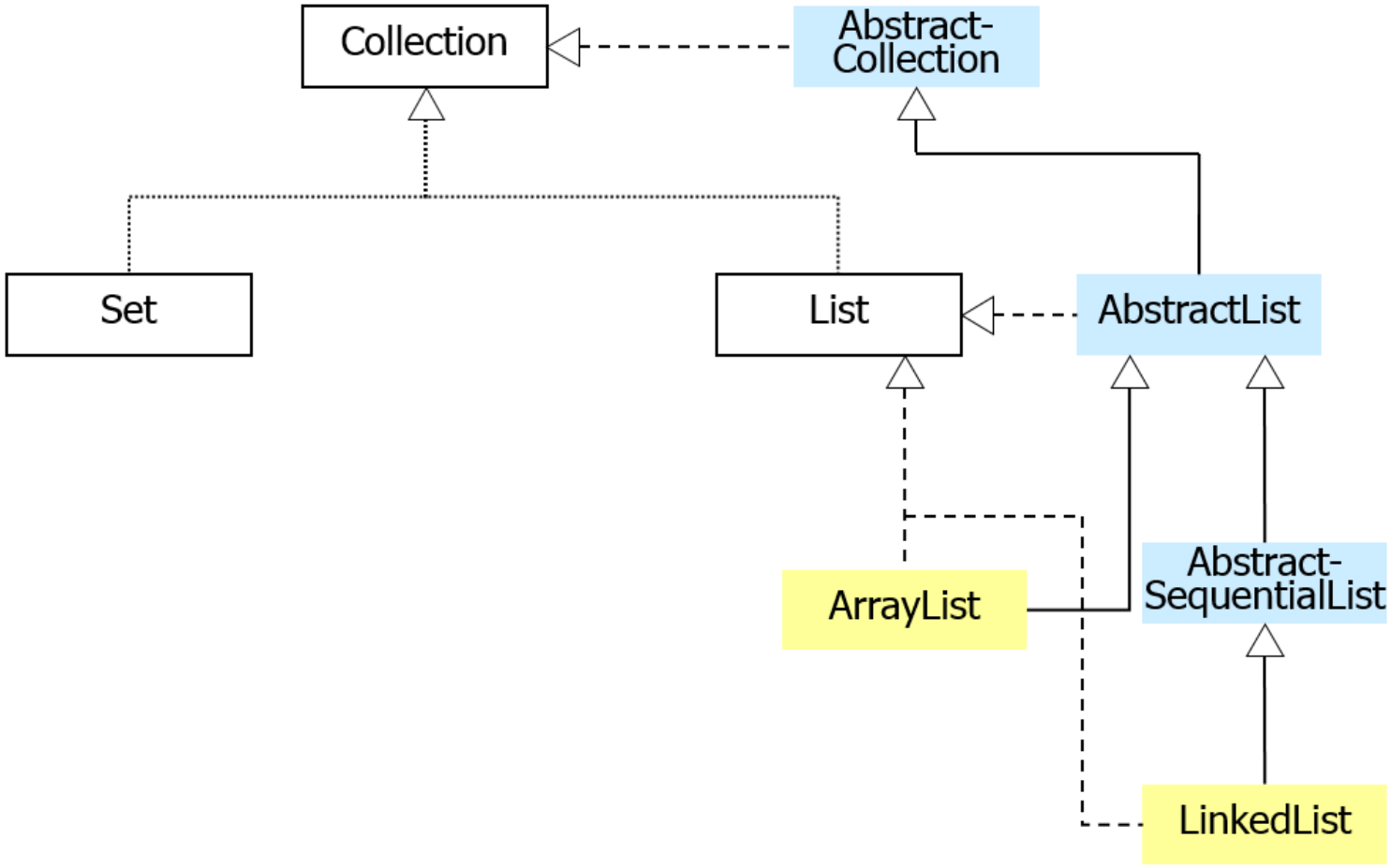


Listen im Java Collection Framework



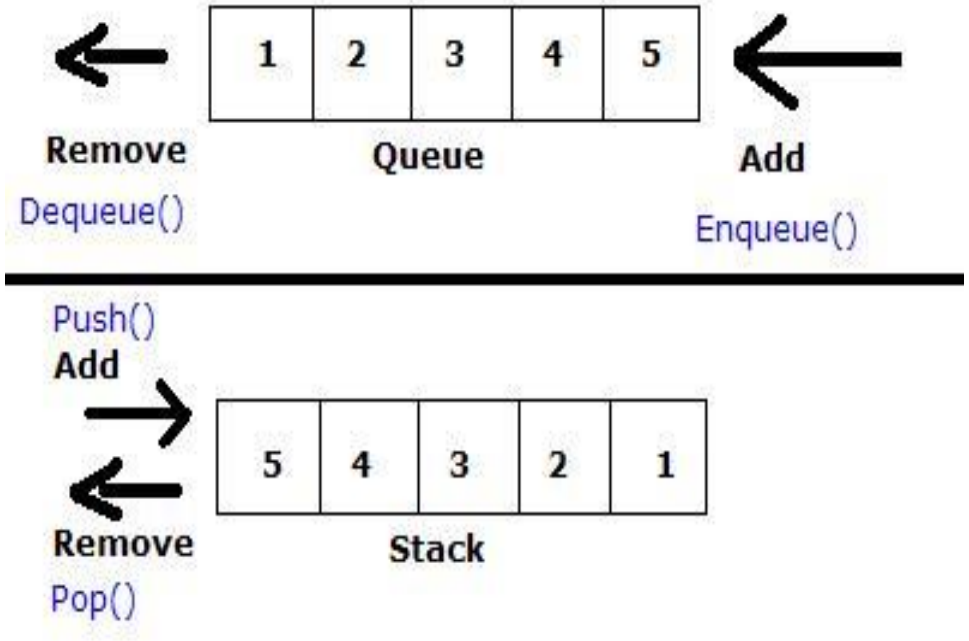
- Bag-Semantik
- Sequenzielle Anordnung von Elementen (hinzufügen per add(E e) am Ende)
- List-Interface definiert u.a. Zugriffe auf Index:
  - get(index)
  - remove(index)
  - add(index, element);

Vorhandene Listen im Java Collection Framework



Stacks / Queues

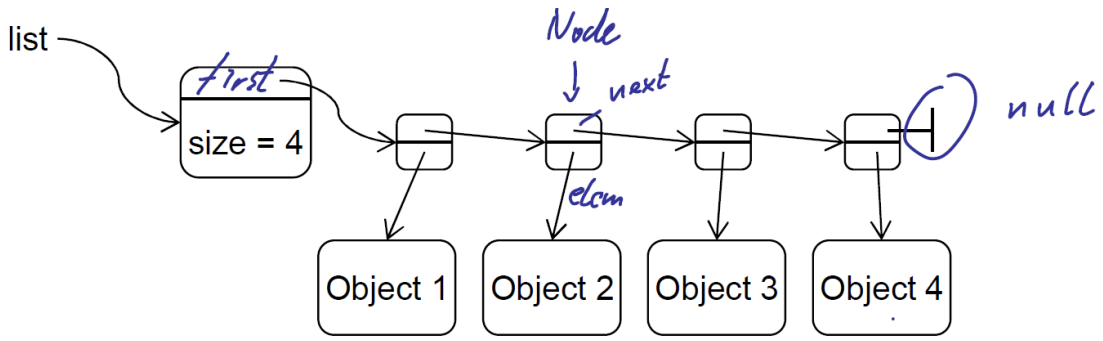
- Basieren auf Listen
- Stack: LIFO (Last-in-first-out)
- Queue: FIFO (First-in-first-out)



Arbeitsblatt: Listen

- Absprache mit Partner
- Jede Person implementiert selbst
- Zeichnet Struktur / Operationen auf !

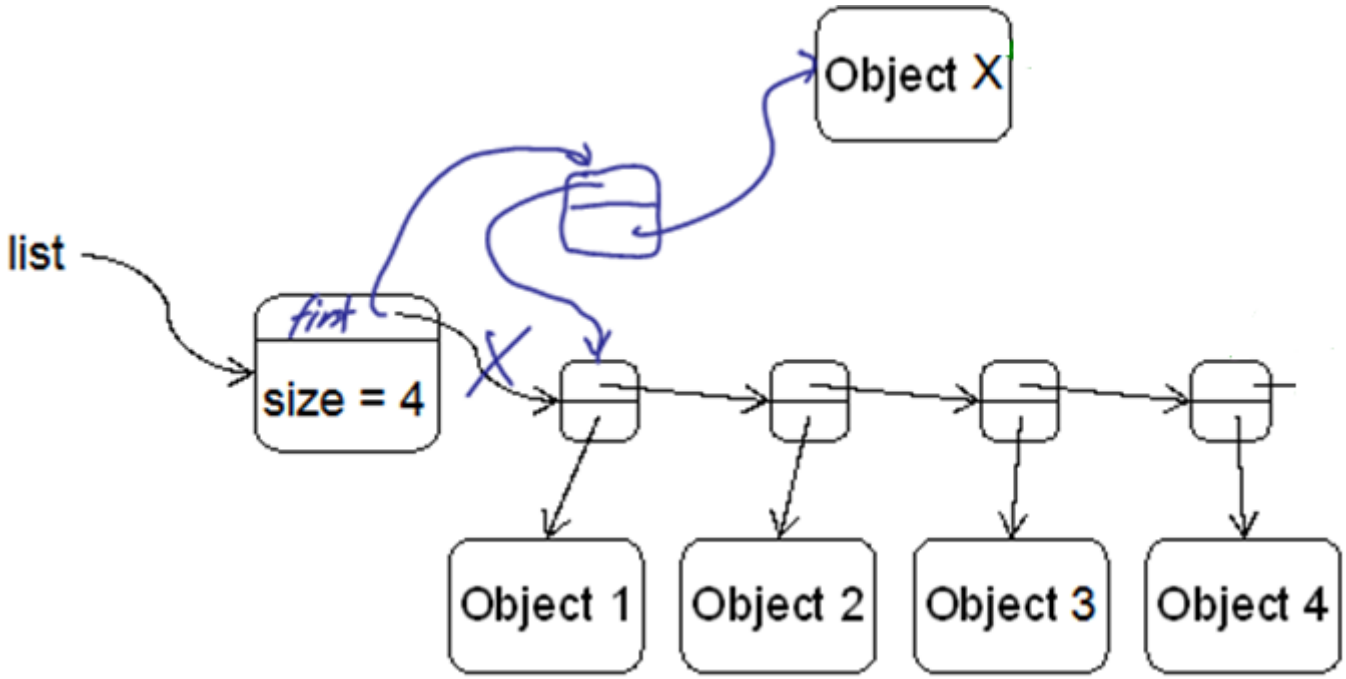
Beschriften der Objekte und Referenzen




```
public class LinkedList<E> implements List<E> {
  private int size = 0;
  private Node<E> first;

  private static class Node<E> {
    private final E elem;
    private Node<E> next;

    private Node(E elem) { this.elem = elem; }

    private Node(E elem, Node<E> next) { this.elem = elem; this.next = next; }
  }
}
```

Lösung Add-Methode: Einfügen am Anfang

```
private boolean add (E item) {
  Node<E> newElement = new Node<E>(item);
  newElement.next = first;
  first = newElement;
  ++size;
  return true;
}
```

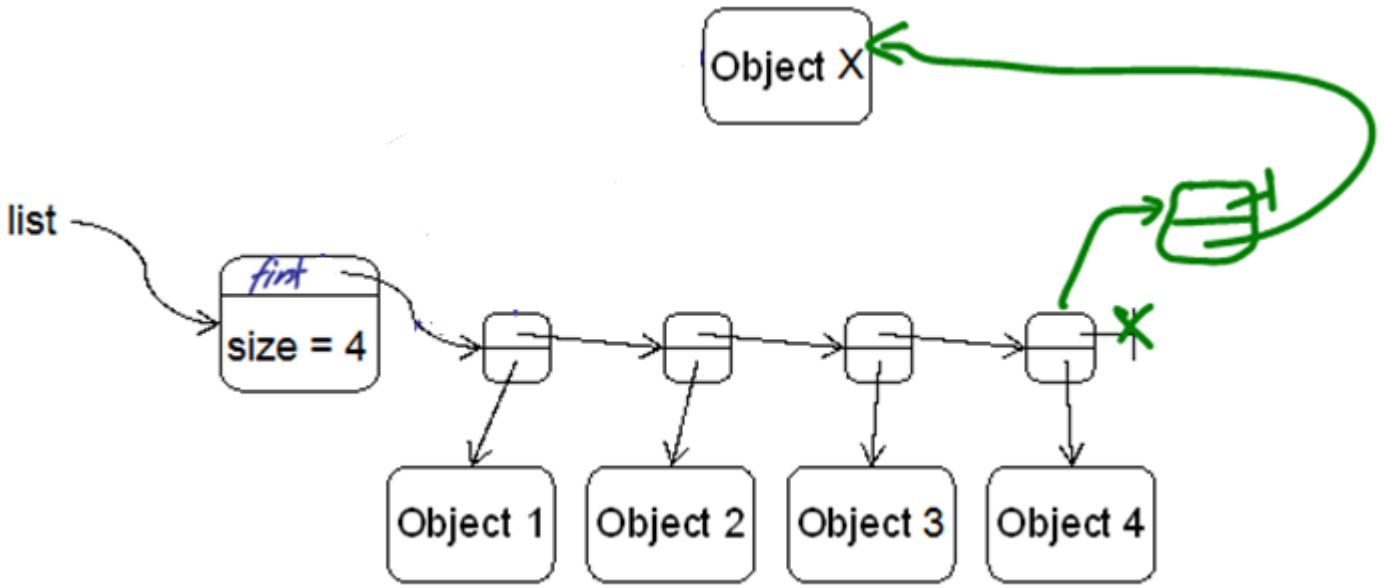


Komplexität: O(1)

Lösung Add-Methode: Einfügen am Ende (Variante ohne last-Zeiger)

```
Node<E> newNode = new Node<E>(item);
if (first != null) {
 Node<E> current = first;
 while (current.next != null) {
  current = current.next;
 }
 current.next = newNode;
} else { first = newNode; }
++size; return true;
```

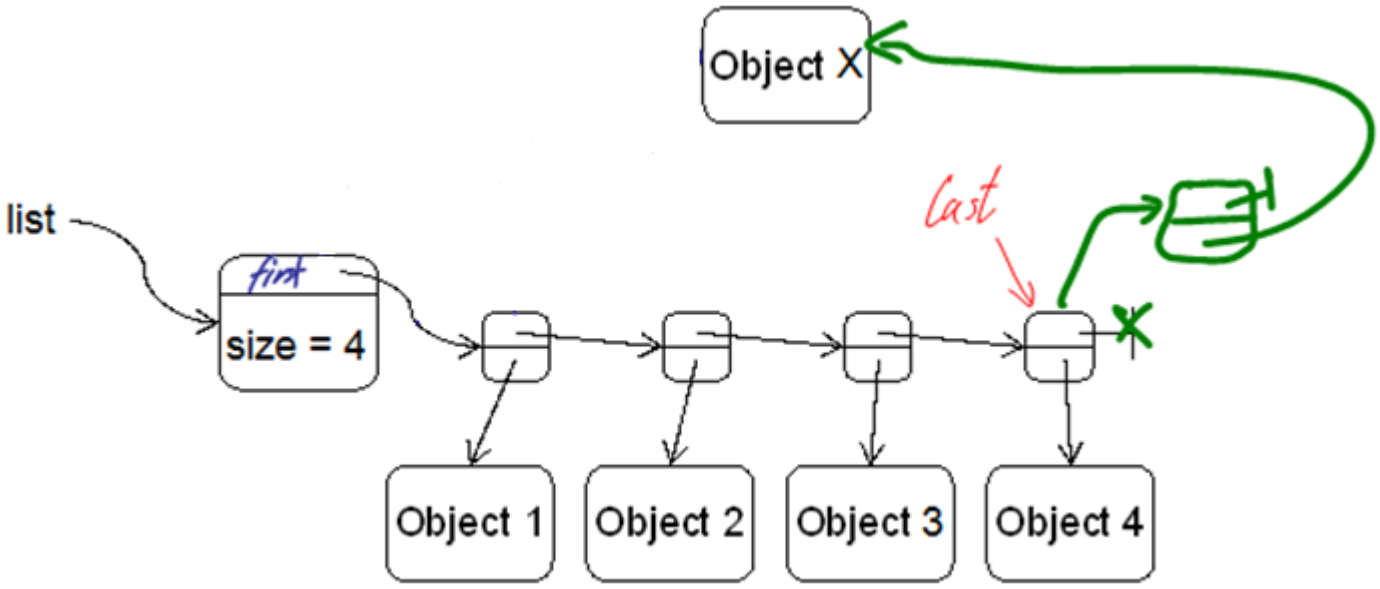
Komplexität: O(n)



Lösung Add-Methode: Einfügen am Ende (Variante mit last-Zeiger)

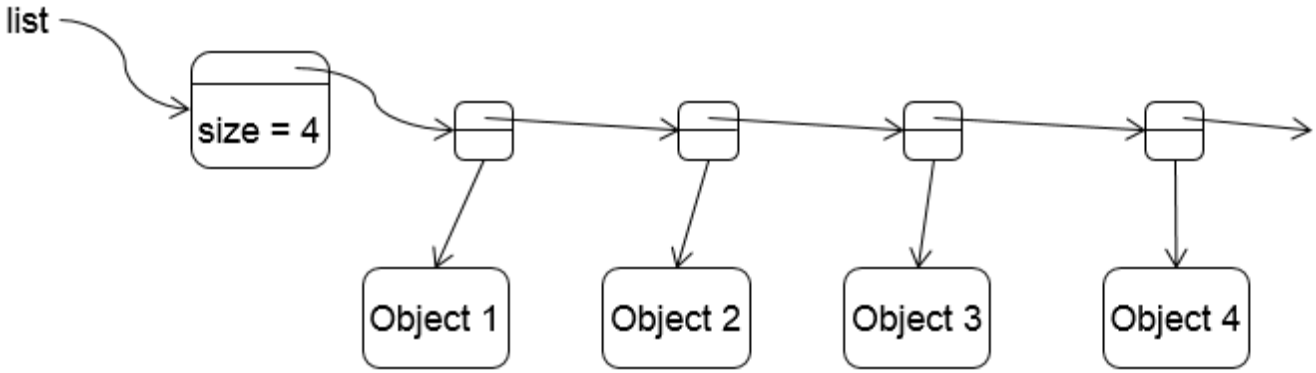
```
Node<E> newNode = new Node<E>(item);

If (first != null) {

 Node<E> current = last;

 current.next = newNode;

 last = newNode;

} else {

first = newNode;

last = first;

}  ++size; return true;
```

Komplexität: O(1)



Lösung: Contains-Methode

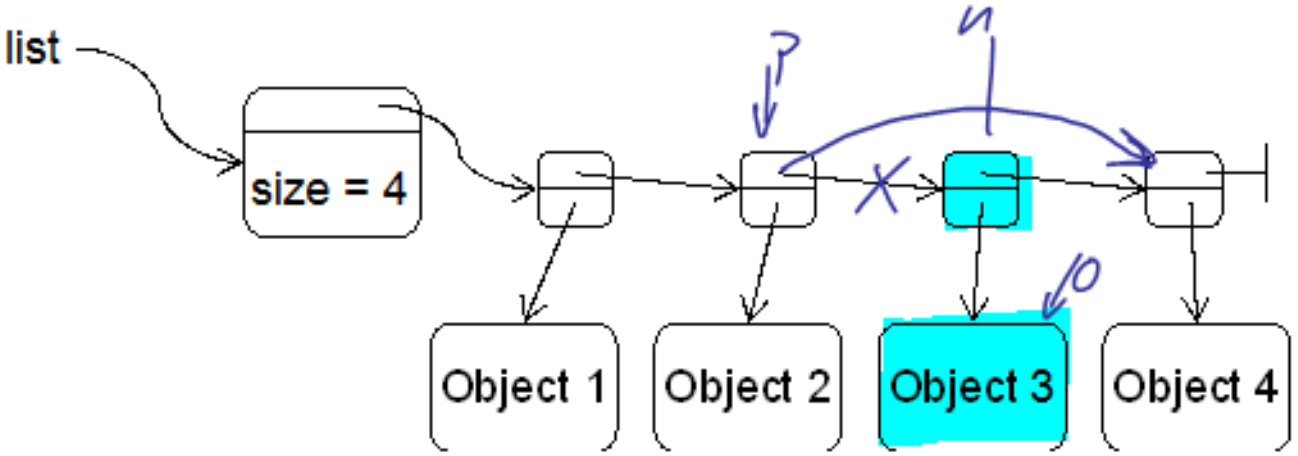
```
public boolean contains(Object o) {
Node<E> current = first;
while (current != null && !current.elem.equals(o)) {
current = current.next;
}
return current != null;
}
```



Komplexität: O(n)

Lösung: Remove-Methode (Schleppzeiger)

```
Node<E> n = first, p = null;
while (n != null && !n.elem.equals(o)) {p = n;n = n.next;}
if (n != null) {
  if (n == last) {last = p;}
  if (p != null) {p.next = n.next;}
  else {first = n.next;}
  size--;
  return true;
}
else return false;
```



Komplexität: O(n)