Programmierkurs Java

UE 12 – Collections

Dr.-Ing. Dietrich Boles

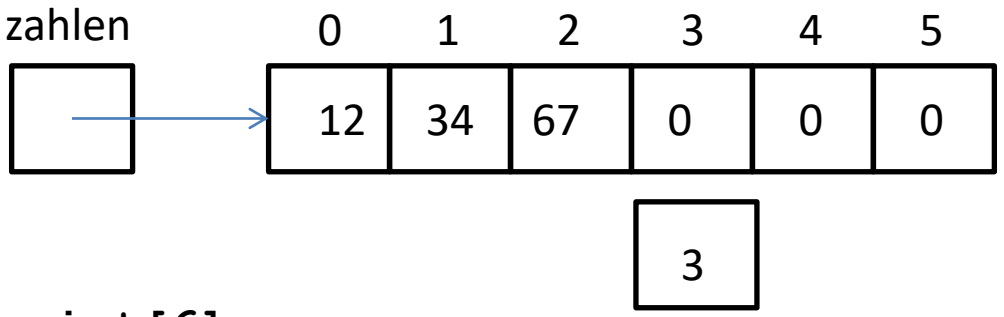
Gliederung

- Motivation
- Array
- ArrayList
- Stack
- Verkettete Liste
- LinkedList
- LinkedStack

Motivation

- Collection = Sammlung von Werten/Elementen
- Kapselung bestimmter Datenstrukturen
  - Liste: Zugriff über Index
  - Set: keine Duplikate
  - Stack: Zugriff nur auf jüngstes Element
  - Queue: Zugriff nur auf ältestes Element
  - Map: Key-Value-Paare
- Verschiedene Implementierungsmöglichkeiten
  - „Dynamisches“ Array
  - Verkettete Liste
- Effizienzbetrachtungen bei bestimmten Operationen

Array




```
int[] zahlen = new int[6];
int nextIndex = 0;
...
Hinzufügen hinten: zahlen[nextIndex++] = 8;
Hinzufügen vorne:
  verschieben von Index 0 bis nextIndex-1 um 1
  nextIndex++;
  zahlen[0] = 8;
Löschen an Index i:
  verschieben von Index i+1 bis nextIndex-1 um -1
  nextIndex--;
```

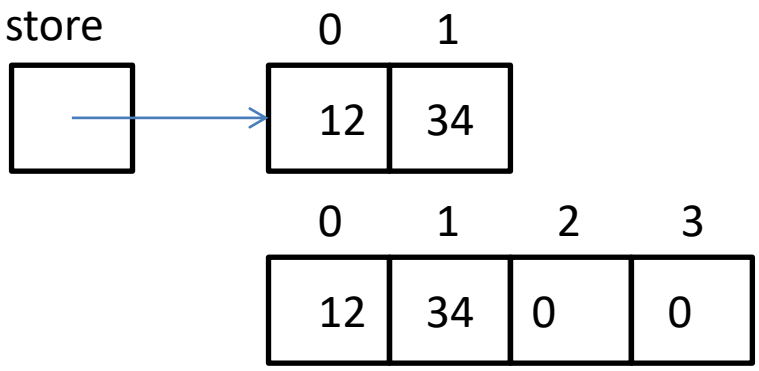
Problem: Array hat feste Größe

Klasse ArrayList (1)

```
// gesucht (Testprogramm)
ArrayList list = new ArrayList();
for (int i = 0; i < 10; i++) {
  list.add(i);
  list.add(i);
}
list.remove(5);
list.remove(9);
list.remove(0);
for (int i = 10; i < 20; i++) {
  list.add(i);
}
list.insert(1,  77);
for (int i = 0; i < list.size(); i++) {
  System.out.println(list.get(i));
}
```

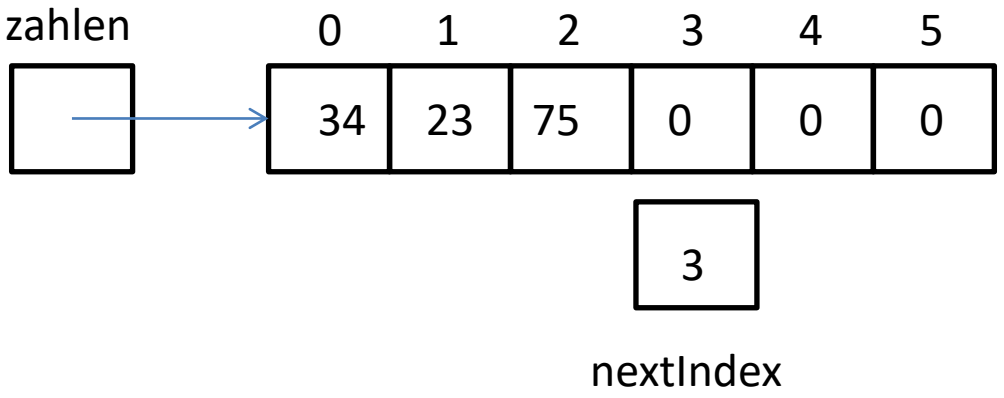
Klasse ArrayList (2)

```
class ArrayList {
  int[] store;
  int next;

  ArrayList() { store = new int[8]; next = 0; }

  void add(int value) {
    ensureCapacity();
    store[next] = value; next++;
  }

  void ensureCapacity() {
    if (next == store.length) {
      int[] buffer = new int[store.length * 2];
      for (int i = 0; i < store.length; i++)
        buffer[i] = store[i];
      store = buffer;
  } }

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

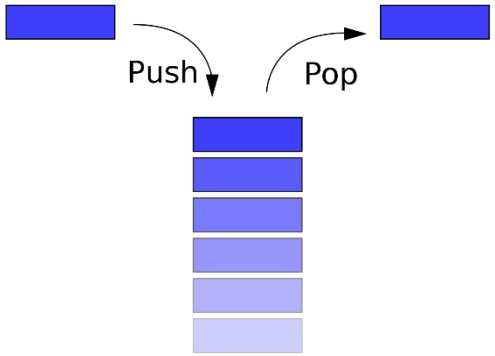


Klasse ArrayList (3)

```
  void insert(int index, int value) { // without index check
    ensureCapacity();
    for (int i = next; i > index; i--)
      store[i] = store[i - 1];
    store[index] = value; next++;
  }

  void prepend(int value) { insert(0, value); }

  int get(int index) {
    return store[index]; // without index check
  }

  void remove(int value) {
    for (int i = 0; i < next; i++) {
      if (store[i] == value) {
        for (int j = i; j < next - 1; j++)
          store[j] = store[j + 1];
        next--; i--;
} } } }
```




```
class Stack {
  // kein Zugriff von außerhalb!
  int[] zahlen = new int[6];
  int nextIndex = 0;
  ...
  // erlaubte Operationen
  push(8): zahlen[nextIndex++] = 8;
  pop():   return zahlen[--nextIndex];
}
```



Klasse Stack (1)

```
// gesucht (Testprogramm)
public static void main(String[] args) {
   Stack stack = new Stack();
   for (int i = 0; i < 10; i++) {
      stack.push(i);
   }
   while (!stack.isEmpty()) {
      System.out.println(stack.pop());
   }
}
```

Klasse Stack (2)

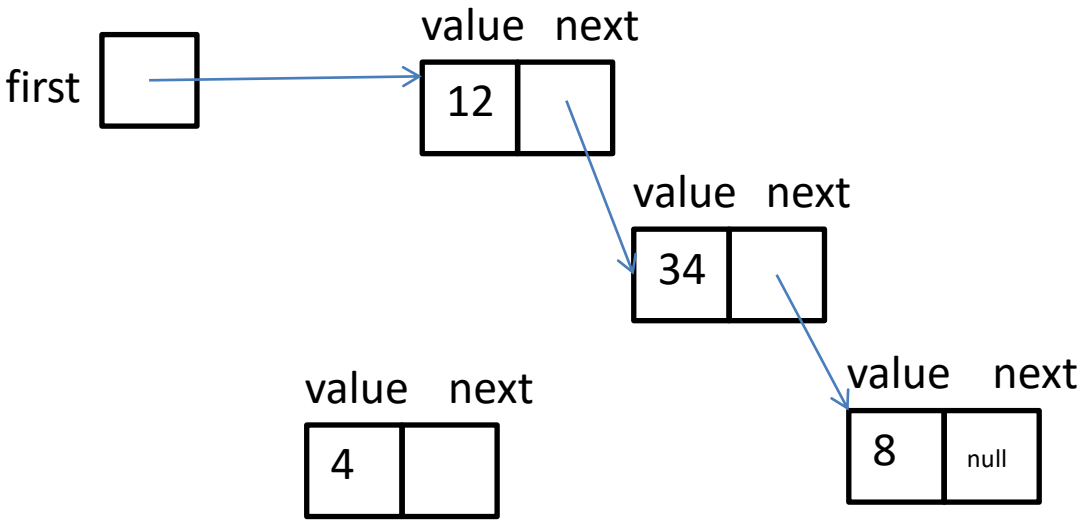
```
class Stack { // für int-Werte
  int[] store;
  int next;

  Stack() { store = new int[8]; next = 0; }

  void push(int value) {
    ensureCapacity();
    store[next] = value;
    next++;
  }

  int pop() {
    if (!isEmpty()) {
      next--; return store[next];
    }
    return 0; // Fehler! spaeter via Exceptions
  }
```

Klasse Stack (3)

```
  boolean isEmpty() {
    return next == 0;
  }

  void ensureCapacity() {
    if (next == store.length) {
      int[] buffer = new int[store.length * 2];
      for (int i = 0; i < store.length; i++)
        buffer[i] = store[i];
      store = buffer;
  } }

}
```

Verkettete Liste

```
class Element {
  int value;
  Element next;
}

class List {
  Element first;
}
```




```
Hinzufügen (newElem):
  suchen: elem.next == null
  elem.next = newElem;

Entfernen (oldElem):
  suchen: prevElem mit prevElem.next == oldElem
  prevElem.next = oldElem.next;
```

Klasse LinkedList (1)

```
// gesucht (Testprogramm)
LinkedList list = new LinkedList();
for (int i = 0; i < 10; i++) {
  list.add(i);
  list.add(i);
}
list.remove(5);
list.remove(9);
list.remove(0);
for (int i = 10; i < 20; i++) {
  list.add(i);
}
list.insert(1,  77);
for (int i = 0; i < list.size(); i++) {
  System.out.println(list.get(i));
}
```

Klasse LinkedList (2)

```
class ListElement {
  int value;
  ListElement next;

  ListElement(int v, ListElement n) {
    value = v;
    next = n;
} }

class LinkedList {
  ListElement first;

  LinkedList() {
    first = null;
  }

  void prepend(int value) {
    first = new ListElement(value, first);
  }
```

Klasse LinkedList (3)

```
int size() {
  int result = 0; ListElement elem = first;
  while (elem != null) { result++; elem = elem.next; }
  return result;
}

void add(int value) {
  if (first == null) first = new ListElement(value, null);
  else {
    ListElement elem = first;
    while (elem.next != null) elem = elem.next;
    elem.next = new ListElement(value, null);
} }

void insert(int index, int value) { // without index check
  if (index == 0) prepend(value);
  else {
    ListElement elem = first;
    for (int i = 0; i < index - 1; i++) elem = elem.next;
    elem.next = new ListElement(value, elem.next);
} }
```

Klasse LinkedList (4)

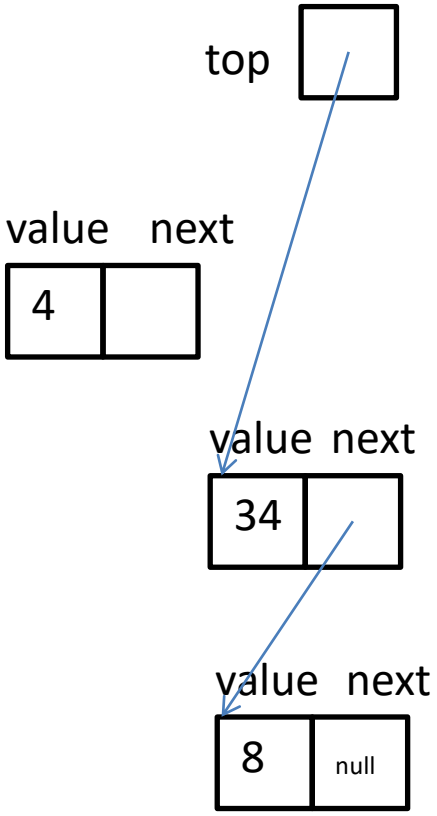
```
int get(int index) { // without index check
  ListElement elem = first;
  for (int i = 0; i < index; i++) elem = elem.next;
  return elem.value;
}
void remove(int value) {
  ListElement before = null;
  ListElement help = first;
  while (help != null) {
    if (help.value == value)
      if (before != null)
        before.next = help.next;
      else  // Element == first
        first = help.next;
    else
      before = help;
    help = help.next;
} }
```

Klasse LinkedStack (1)

```
  // gesucht (Testprogramm)
  public static void main(String[] args) {
    LinkedStack stack = new LinkedStack();
    for (int i = 0; i < 10; i++) {
      stack.push(i);
    }
    while (!stack.isEmpty()) {
      System.out.println(stack.pop());
    }
  }
}
```

Klasse LinkedStack (2)

```
class StackElement {
  int value;
  StackElement next;

  StackElement(int v, StackElement n) {
    value = v; next = n;
} }

class LinkedStack {
  StackElement top;

  LinkedStack() { top = null; }

  boolean isEmpty() { return top == null; }
```



Klasse LinkedStack (3)

```
 void push(int value) {
   top = new StackElement(value, top);
 }

int pop() {
  if (!this.isEmpty()) {
    int result = top.value;
    top = top.next;
    return result;
  }
  return 0; // Fehler! spaeter via Exceptions
}
```

Zusammenfassung

- Collection = Sammlung von Werten/Elementen
- ArrayList = Liste mit „dynamischem“ Array
- LinkedList = Liste mit verketteten Elementen
- Stack = gekapselte Last-In-First-Out (LIFO) Datenstruktur
- Queue = gekapselte First-In-First-Out (FIFO) Datenstruktur