Karlsruher Institut für Technologie

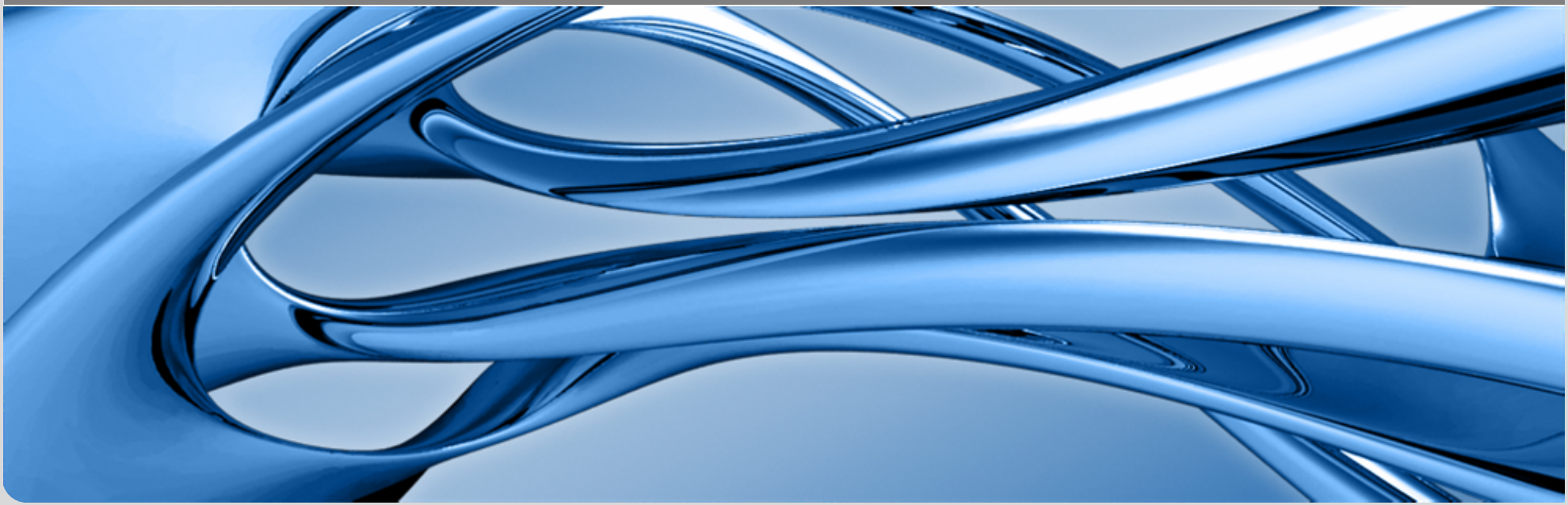
SDQ Software Design and Quality

Tutorium 6

Listen und abstrakte Datentypen

Christian Zielke | 4. Dezember 2018

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

Warum Listen?

- Wie ihr evtl schon bemerkt habt, sind Arrays sehr statisch
  $\rightarrow$ Größe fest, für Größenänderung viel Aufwand
- Listen sind hier flexibler
  $\rightarrow$ Es können beliebig Einträge hinzugefügt oder entfernt werden

Einfach verkettete Liste

Für eine einfach verkettete Liste benötigt man 2 Klassen:

- Listenelement
  - Im Element gespeichertes Objekt (Datentyp: Zu listender Datentyp)
  - Zeiger auf den Nachfolger (Datentyp: Listenelement)
  - Kann als geschachtelte Klasse der Liste realisiert werden
- Liste
  - Erstes Listenelement (Datentyp: Listenelement, bei leerer Liste `null`)
  - Funktionalität (Methoden)

Grundgerüst

```
1   class ListElement {
2     private Object entry;
3     private ListElement next;
4
5     public ListElement(Object entry) {
6       this.entry = entry;
7       this.next = null;
8     }
9     //Getter
10    //Setter for next
11  }
```

Grundgerüst

```
1   class List {
2
3     private ListElement head;
4
5     public List() {
6       this.head = null;
7     }
8
9     //Funktionalitaetsmethoden
10  }
```

Typische Funktionalitäten

- addFirst(Object newItem)
- addLast(Object newItem) (auch push genannt)
- insertAfter(Object prevItem, Object newItem)
- remove(Object item)
- removeFirst(), removeLast()
- removeAfter(Object prevItem)
- contains(Object item)
- popFirst(), popLast()
- ...

Doppelt verkettete Listen

- Listenelemente haben zusätzlich einen Vorgänger
- Vorgänger des ersten Elements ist das Letzte
- Nachfolger des letzten Elements ist das Erste
- Oder mit Zeigern auf Anfang und Ende
- Beschleunigen manche Funktionalitäten
  $\rightarrow$ Mehr dazu im SS - Algorithmen I

Iterator: Definition

- Ein **Iterator** ist ein Zeiger, der über die Elemente einer Sammlung wandern (iterieren) kann
- Bieten meist 2 grundlegende Funktionalitäten:
  - Existiert ein weiteres Element?
  - Gehe zum nächsten Element
- Wird als geschachtelte Klasse implementiert (siehe Beispiel)

Iterator: Beispiel

```
1   public class List {
2     ...
3     public ListIterator getListIterator() {
4       return new ListIterator(head);
5     }
6     public class ListIterator {
7
8       private ListElement current;
9
10      private ListIterator(ListElement start) {
11        this.current = start;
12      }
13      public boolean hasNext() {
14        return current != null;
15      }
16      public Object next() {
17        Object currentObj = current.entry;
18        this.current = current.next;
19        return currentObj;
20      }
21    }
22  }
```

Iterator: Beispiel

Benutzung des Iterators

```
1   List list = new List();
2
3   ...
4   ListIterator it = list.getListIterator();
5
6   while (it.hasNext()) {
7     Object element = it.next();
8     ...
9   }
```

Übungsaufgaben

Verkettete Liste

- Implementiere eine einfach verkettete Liste für Integer:
- public void addFirst(Integer i)
- public void remove(Integer i) //löscht alle Elemente mit Wert i
- public boolean contains(Integer i)
- public String toString()

Erweiterte Funktionalität

- Erweitere deine Liste um diese Methoden:
- public boolean isEmpty()
- public void addLast(Integer i)
- public Integer get(int i) //gibt Objekt an Stelle i zurück