

FOP Recap #10



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Listen und verzweigte Strukturen



Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Interface List

- Listen in Java

- Nutzung

- Listen vs. Arrays

Verzeigerte Datenstrukturen



- Listen in Java sind unterschiedlich zu Listen in Racket
- Der abstrakte Datentyp der Liste wird in Java durch das Interface `List` repräsentiert
- `List` ist generisch mit einem Typparameter `E`
- Wichtige Operationen, die in dem Interface festgelegt werden:
 - ▣ `size()`: Länge der Liste
 - ▣ `add(E element)`: Element am Ende hinzufügen
 - ▣ `remove(E element)`: Element aus Liste entfernen
 - ▣ `contains(E element)`: Prüfen ob Element in Liste ist
 - ▣ Viele weitere, siehe Dokumentation



- Im Package `java.util` finden sich einige Klassen, die `List` implementieren:
 - ▣ `ArrayList`
 - ▣ `LinkedList`
 - ▣ Eigene Implementation auch möglich
- Oft genutztes Muster:
 - ▣ Überall, wo Implementation egal ist (fast immer): Verwende statischen Typ `List<E>`
 - ▣ Wenn eine Liste erstellt werden muss, verwende Implementation aus Standardbibliothek

```
1 ArrayList<String> myList1 = new ArrayList<String>();  
2 // Diamond Operator: String wird impliziert:  
3 ArrayList<String> myList2 = new ArrayList<>();  
4 // List ist Supertyp von ArrayList:  
5 List<String> myList3 = new ArrayList<>();
```

Interface List

Listen vs. Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Eigenschaft	Arrays	Listen
Gesamtgröße	fest	dynamisch
Implementation	immer gleich	dynamischer Typ kann variieren
Objektattribute	length	keins
Generics	zu vermeiden	Sehr wichtig
Hilfsmethoden	nur in <code>java.util.Arrays</code>	durch Interface festgelegt, <code>java.util.Collections</code>



Interface List

Verzeigerte Datenstrukturen

- Grundideen

- Klasse `ListItem`

- Typische Form

- Mögliche interne Struktur

- Visuelle Repräsentation

- Zugriff auf Elemente

- Iterieren von Datenstrukturen

- Iterieren mit `ListItem`

- Beispiel-Implementierungen

- Typische Fehler



- Nutzen Objekte, die durch Referenzen (Zeiger/Pointer) aufeinander verweisen
- Starker Kontrast zu Arrays:
 - ▣ Haben veränderbare Länge
 - ▣ Effizientes Hinzufügen und Entfernen
 - ▣ Auch Nachteile: Können Endlos-Schleifen besitzen

Verzeigerte Datenstrukturen

Klasse `ListItem`



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4 }
```


Verzeigerte Datenstrukturen

Typische Form – LinkedList



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- add: Hinzufügen von Objekten
- remove: Entfernen von Objekten
- contains: Prüfen, ob Objekte in Liste sind
- ...

Verzeigerte Datenstrukturen

Mögliche interne Struktur – LinkedList



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4 }
```

```
1 public class LinkedList<T> {  
2     private ListItem<T> head;  
3     // private ListItem<T>tail;  
4 }
```

Verzeigerte Datenstrukturen

Mögliche interne Struktur – Double Linked



TECHNISCHE
UNIVERSITÄT
DARMSTADT

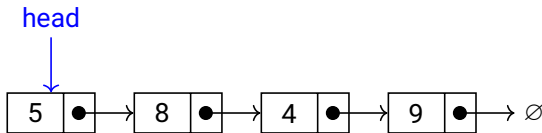
```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4     public ListItem<T> prev;  
5 }
```

Verzeigerte Datenstrukturen

Visuelle Repräsentation – LinkedList<Integer>



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Verzeigerte Datenstrukturen

Zugriff auf Elemente



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Keinen (direkten) Zugriff durch einen Index
- Zugriff durch "Folgen der Zeiger"

Verzeigte Datenstrukturen

Iterieren von Datenstrukturen — Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public static <T> void print(T[] array) {  
2     System.out.println("Array:");  
3     for(T obj : array) {  
4         System.out.println("-> " + obj);  
5     }  
6 }
```

```
1 public static <T> void print(T[] array) {  
2     System.out.println("Array:");  
3     for(int index = 0; index < array.length; index++) {  
4         System.out.println("-> " + array[index]);  
5     }  
6 }
```



```
1 public static <T> void print(List<T> list) {  
2     System.out.println("List:");  
3     // possible for every class that implements Iterable  
4     for(T obj : list) {  
5         System.out.println("-> " + obj);  
6     }  
7 }
```

```
1 public static <T> void print(List<T> list) {  
2     System.out.println("List:");  
3     for(int index = 0; index < list.size(); index++) {  
4         System.out.println("-> " + list.get(index));  
5     }  
6 }
```

Verzeigerte Datenstrukturen

Iterieren mit ListItem



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4 }
```

```
1 public static <T> void print(ListItem<T> head) {  
2     System.out.println("List:");  
3     ListItem<T> pointer = head;  
4     while(pointer != null) {  
5         System.out.println("-> " + pointer.key);  
6         pointer = pointer.next;  
7     }  
8 }
```


Verzeigerte Datenstrukturen

Iterieren mit ListItem



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4 }
```

```
1 public static <T> void print(ListItem<T> head) {  
2     System.out.println("List:");  
3     for(ListItem<T> pointer = head; pointer != null;  
4         pointer = pointer.next) {  
5         System.out.println("-> " + pointer.key);  
6     }  
7 }
```



```
1 public class ListItem<T> {  
2     public T key;  
3     public ListItem<T> next;  
4 }
```

```
1 public class LinkedList<T> {  
2     private ListItem<T> head;  
3     //....  
4 }
```

Verzeigerte Datenstrukturen

Beispiel-Implementierungen – addFirst



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void addFirst(T item) {  
2     ListItem<T> newItem = new ListItem<T>();  
3     newItem.key = item;  
4     newItem.next = head;  
5     head = newItem;  
6 }
```

Verzeigerte Datenstrukturen

Beispiel-Implementierungen – add



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void add(T item) {  
2     ListItem<T> newItem = new ListItem<T>();  
3     newItem.key = item;  
4  
5     if(head == null) {  
6         head = newItem;  
7     }  
8     else {  
9         // ....  
10    }  
11 }
```



```
1 public void add(T item) {  
2     ....  
3     else {  
4         // Get last item  
5         ListItem<T> tail = head;  
6         while(tail.next != null) {  
7             tail = tail.next;  
8         }  
9  
10        // Append item  
11        tail.next = newItem;  
12    }  
13 }
```



```
1 public boolean remove(T item) {  
2     if(head != null && head.key.equals(item)) {  
3         head = head.next;  
4         return true;  
5     }  
6     else {  
7         ....  
8     }  
9 }
```



```
1  else {
2      ListItem<T> pointer = head;
3      // Search for previous node
4      while(pointer.next != null &&
5      pointer.next.key.equals(item) == false) {
6          pointer = pointer.next;
7      }
8      if(pointer.next == null) {
9          return false;
10     }
11     // Set pointers
12     pointer.next = pointer.next.next;
13     return true;
14 }
```



```
1 public boolean contains(T item) {  
2     ListItem<T> pointer = head;  
3     while(pointer != null) {  
4         if(pointer.key.equals(item)) {  
5             return true;  
6         }  
7         pointer = pointer.next;  
8     }  
9     return false;  
10 }
```




```
1 public T get(int index) {  
2     int position = 0;  
3     ListItem<T> pointer = head;  
4     while(pointer != null) {  
5         if(position == index) {  
6             return pointer.key;  
7         }  
8         pointer = pointer.next;  
9         position += 1;  
10    }  
11    throw new IndexOutOfBoundsException(index);  
12 }
```

Verzeigerte Datenstrukturen

Beispiel-Implementierungen – insert



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public boolean insert(int index, T item) {  
2     if(index < 0) { return false; }  
3     if(index == 0) {  
4         addFirst(item);  
5         return true;  
6     }  
7     else {  
8         // Create new item  
9         ListItem<T> newItem = new ListItem<T>();  
10        newItem.key = item;  
11        ....  
12    }  
13 }
```



```
1  ....
2  // Search for previous item
3  int position = 0;
4  ListItem<T> pointer = head;
5
6  while(pointer != null && position+1 < index) {
7      pointer = pointer.next;
8      position += 1;
9  }
10  ....
```



```
1  ....
2  if(pointer == null) {
3      // Index was not reachable
4      return false;
5  }
6  // Insert newItem between pointer and pointer.next
7  newItem.next = pointer.next;
8  pointer.next = newItem;
9  return true;
```



- Edge-Cases nicht beachtet, zum Beispiel:
 - ▣ `head` ist `null`
 - ▣ `remove` bezieht sich auf `head`
 - ▣
- Zeiger werden vergessen, zum Beispiel:
 - ▣ `newItem.next` in `remove`
 - ▣
- Erzeugen einer Schleife durch fehlerhaftes Setzen von Zeigern
 - ▣ Führt zu Endlos-Schleifen



Live-Coding!