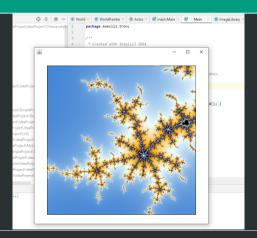
FOP Recap #11



Verzeigerte Strukturen



Willkommen zurück!

Das steht heute auf dem Plan



Wiederholung: List Arrays vs Listen

Verzeigerte Daten-Strukturer

Wiederholung: List



```
ArrayList<String> myList1 = new ArrayList<String>();
// Diamond Operator: String wird impliziert:
ArrayList<String> myList2 = new ArrayList<>();
// List ist Überklasse von ArrayList:
List<String> myList3 = new Arraylist<</pre>
```

Wiederholung: List



List<E>:

- size(): Länge der Liste
- add(E element): Element ans Ende hinzufügen
- remove(E element): Element aus Liste entfernen
- contains(E element): Prüfen ob Element in Liste ist
- und mehr ...

Wiederholung: List

Arrays vs Listen



Eigenschaft	Arrays	Listen
Gesamtgröße	fest	dynamisch
Hinzufügen v. Elementen	nein	ja

Das steht heute auf dem Plan



Wiederholung: List

Verzeigerte Daten-Strukturen Wie? Was? Typische Form Mögliche interne Struktur Visuelle Repräsentation Typische Algorithmen-Form Zugriff auf Elemente Iterieren von einem Array Iterieren mit List Ttem Beispiel-Implementierungen Typische Fehler

Wie? Was?



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

Typische Form - LinkedList



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

Mögliche interne Struktur - LinkedList



```
public class ListItem<T> {
    public T key;
    public ListItem<T> next;
}

public class LinkedList<T> {
    private ListItem<T> head;
    // private ListItem<T> tail;
}
```

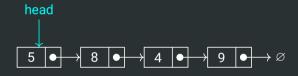
Mögliche interne Struktur - Double Linked



```
public class ListItem<T> {
    public T key;
    public ListItem<T> next;
    public ListItem<T> prev;
}
```

Visuelle Repräsentation - LinkedList<Integer>





Typische Algorithmen-Form



```
public class LinkedList<T> {
public static <T> void add(LinkedList<T> list, T obj) {
public static <T> ListItem<T> add(ListItem<T> head, T obj) {
```

Zugriff auf Elemente



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

Iterieren von einem Array



```
public static <T> void print(T[] array) {
    System.out.println("Array:");
    for(T obj : array) {
        System.out.println("-> " + obj);
public static <T> void print(T[] array) {
    System.out.println("Array:"):
    for(int index = 0: index < array.length: index++) {</pre>
        System.out.println("-> " + array[index]);
```

Iterieren mit ListItem



```
public class ListItem<T> {
   public T key:
   public ListItem<T> next:
public static <T> void print(ListItem<T> head) {
    System.out.println("List:");
    ListItem<T> pointer = head;
    while(pointer != null) {
        System.out.println("-> " + pointer.key);
        pointer = pointer.next:
```

Iterieren mit ListItem



```
public class ListItem<T> {
    public T kev:
    public ListItem<T> next;
public static <T> void print(ListItem<T> head) {
    System.out.println("List:");
    for(ListItem<T> pointer = head; pointer != null;
                             pointer = pointer.next) {
        System.out.println("-> " + pointer.kev):
```

Beispiel-Implementierungen



```
public class ListItem<T> {
    public T key;
    public ListItem<T> next;
}

public class LinkedList<T> {
    private ListItem<T> head;
    //....
}
```

Beispiel-Implementierungen - addFirst



```
public void addFirst(T item) {
    ListItem<T> newItem = new ListItem<T>();
    newItem.key = item;
    newItem.next = head;
    head = newItem;
}
```

Beispiel-Implementierungen - add



```
public void add(T item) {
       <u>ListItem<T></u> newItem = new ListItem<T>();
       newItem.key = item;
       if(head == null) {
           head = newItem;
       else {
10
```

Beispiel-Implementierungen - add



```
public void add(T item) {
       else {
           ListItem<T> tail = head:
           while(tail.next != null) {
               tail = tail.next:
10
           tail.next = newItem:
```

Beispiel-Implementierungen - remove



```
public boolean remove(T item) {
    if(head != null && head.key.equals(item)) {
        head = head.next;
        return true;
    }
    else {
        ....
    }
}
```

Beispiel-Implementierungen - remove



```
else {
       ListItem<T> pointer = head;
       while(pointer.next != null &&
       pointer.next.key.equals(item) == false) {
           pointer = pointer.next;
       if(pointer.next == null) {
           return false:
10
       pointer.next = pointer.next.next:
       return true:
14
```

Beispiel-Implementierungen — contains



```
public boolean contains(T item) {
       ListItem<T> pointer = head:
       while(pointer != null) {
           if(pointer.key.equals(item)) {
               return true:
6
           pointer = pointer.next;
       return false:
10
```

Beispiel-Implementierungen - get



```
public T get(int index) {
       int position = 0:
       ListItem<T> pointer = head;
       while(pointer != null) {
           if(position == index) {
               return pointer.key:
           pointer = pointer.next:
           position += 1:
10
       throw new IndexOutOfBoundsException(index):
```

Beispiel-Implementierungen - insert



```
public boolean insert(int index, T item) {
       if(index < 0) { return false; }</pre>
       if(index == 0) {
            addFirst(item);
            return true:
6
       else {
            // Create new item
            ListItem<T> newItem = new ListItem<T>():
            newItem.kev = item:
10
            . . . .
```

Beispiel-Implementierungen - insert



```
int position = 0:
   ListItem<T> pointer = head:
   while(pointer != null && position+1 < index) {</pre>
       pointer = pointer.next;
       position += 1;
10
```

Beispiel-Implementierungen - insert



```
if(pointer == null) {
    return false:
// Insert newItem between pointer and pointer.next
newItem.next = pointer.next;
pointer.next = newItem;
return true;
```

Typische Fehler



- 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!