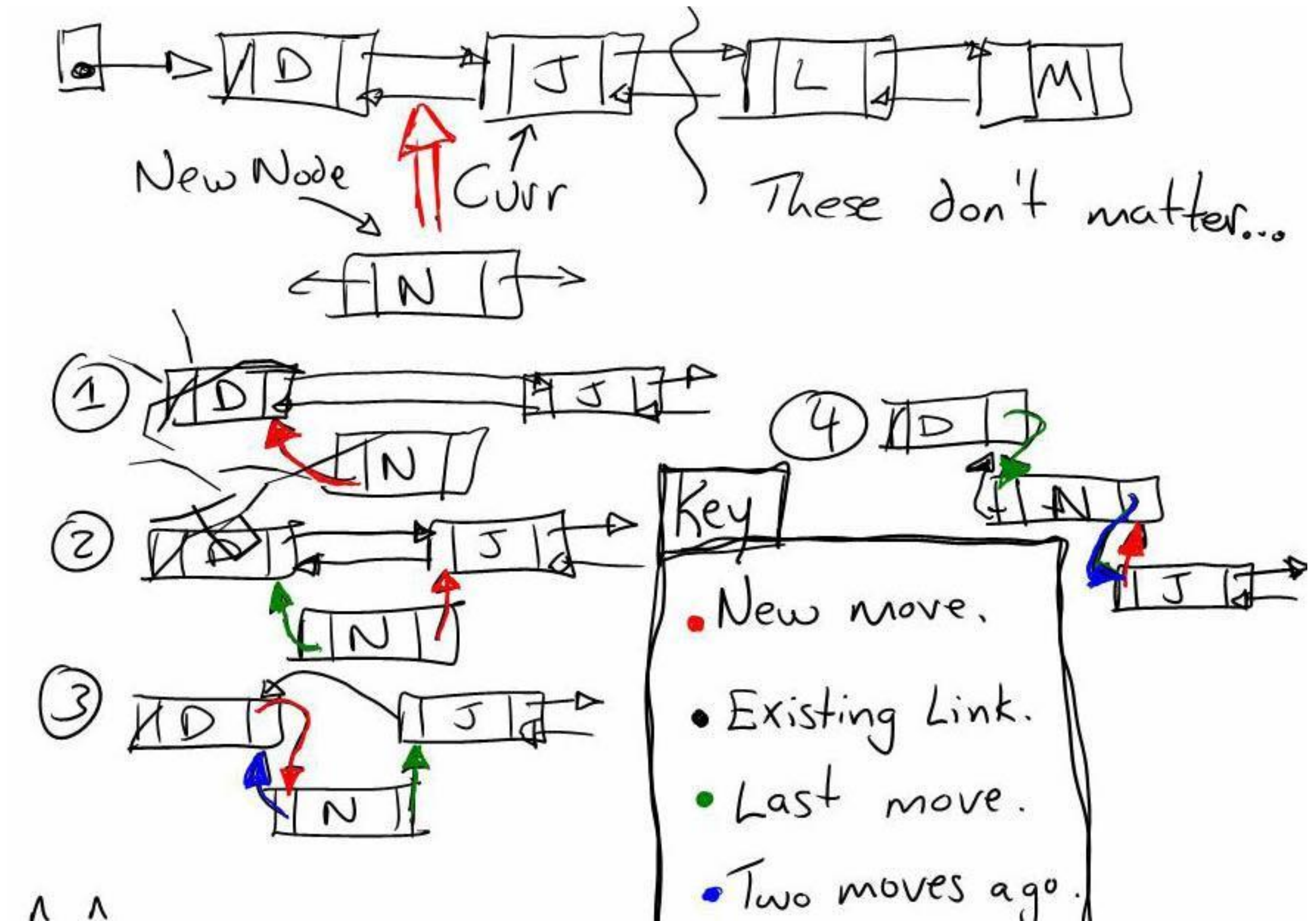
Verkettete Listen



Zielsetzungen

- Sie wissen was **verkettete Listen** sind und kennen verschiedene **Listentypen.**
- Sie kennen die entsprechenden **Java-Klassen** und deren wichtigsten Operationen.
- Sie wissen **in welcher Situation Listen und deren Java-Klassen** angewendet werden.

Ablauf

- **Teil 1: Verkettete Listen**
- **Teil 2: Java-Collections Framework**
- **Teil 3: Wann welche Liste/Klasse?**

Teil 1: Verkettete Listen

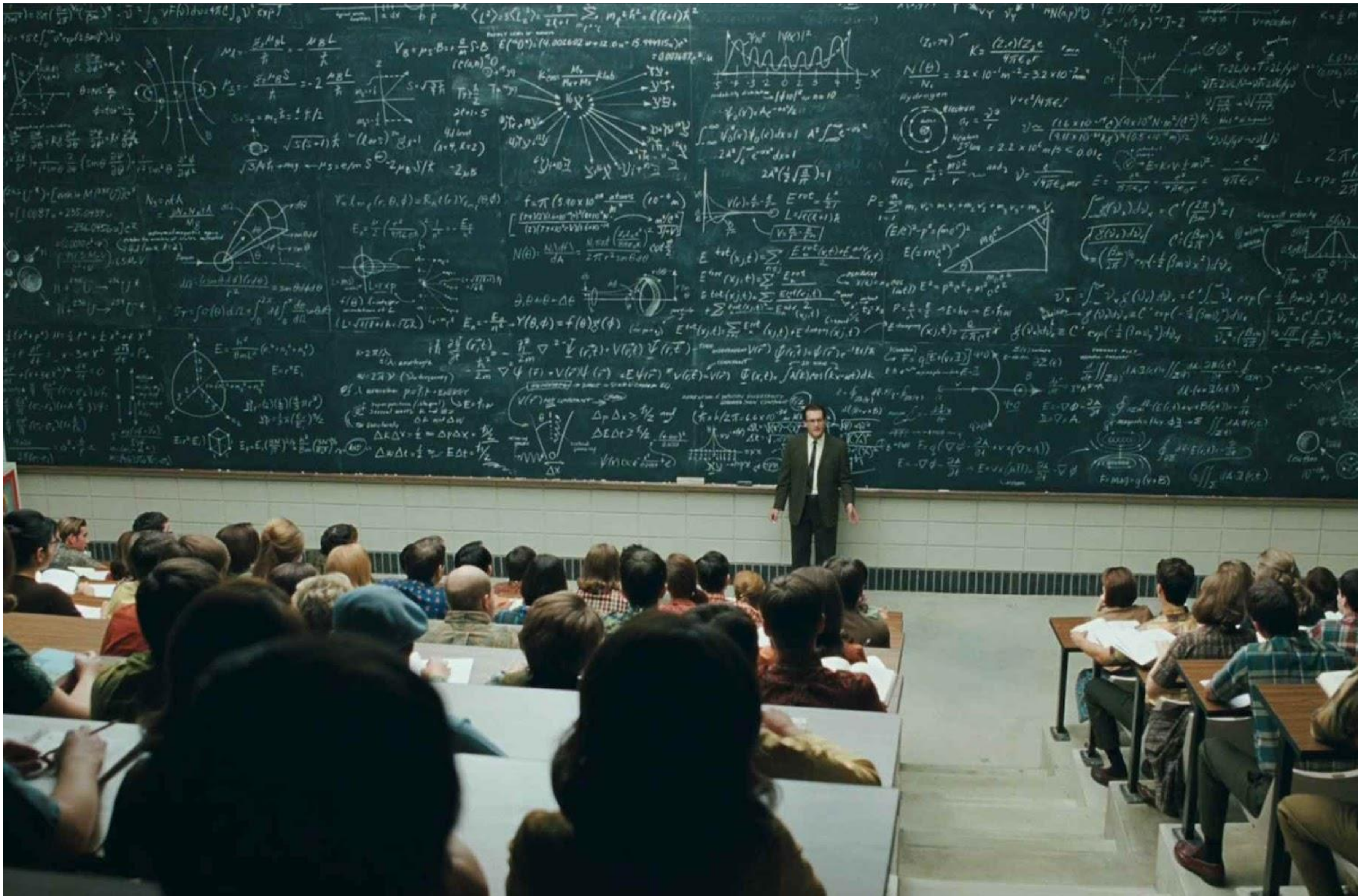
- Arrays
- Verkettete Listen
  - Einfach verkettete Liste
  - Zirkuläre Liste
  - Mehrfach (z.B. doppelt) verkette Liste
  - Sortierte Liste, Interface: Comparable
- Performancebetrachtungen

Arrays

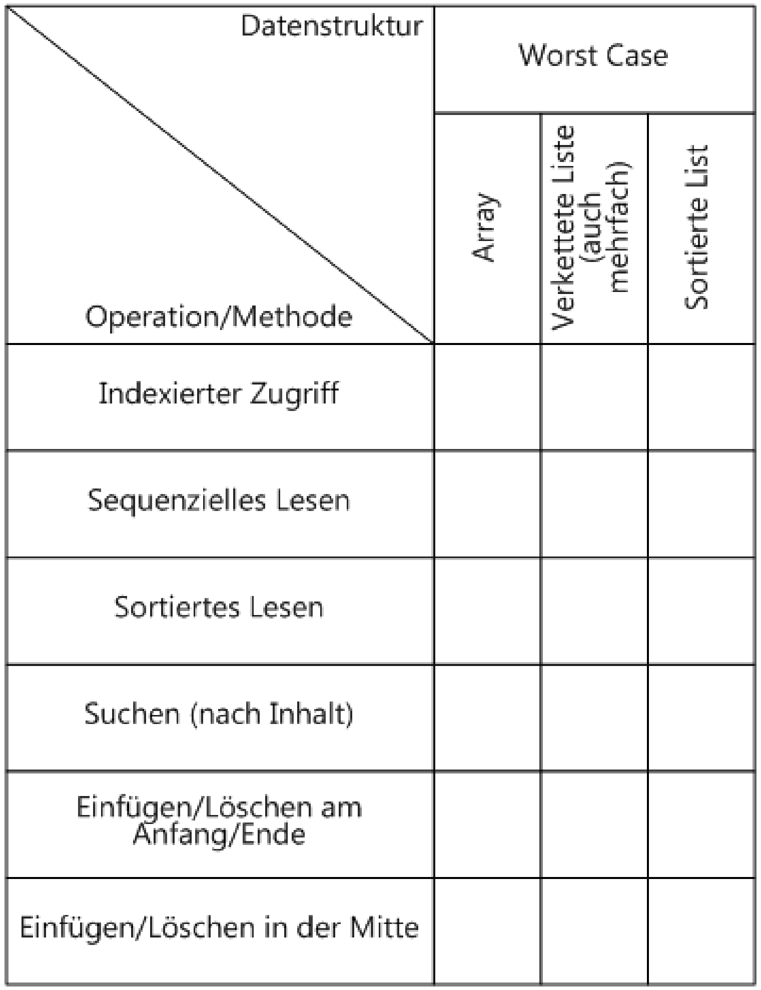
```
String[] playList = new String[5];
playList[0] = "TiK ToK";
playList[1] = "Domino";
playList[2] = "Dance Again";
playList[3] = "E.T.";
playList[4] = "The West";
```

Was sind die zwei Hauptprobleme des Arrays?

Verkettete Listen…

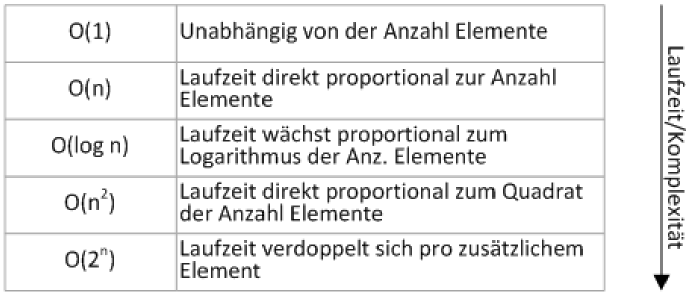


Performance



Datenstruktur / Operation/Methode	Worst Case		
	Array	Verkettete Liste (auch mehrfach)	Sortierte List
Indexierter Zugriff			
Sequenzielles Lesen			
Sortiertes Lesen			
Suchen (nach Inhalt)			
Einfügen/Löschen am Anfang/Ende			
Einfügen/Löschen in der Mitte			

Legende (Beispiele):



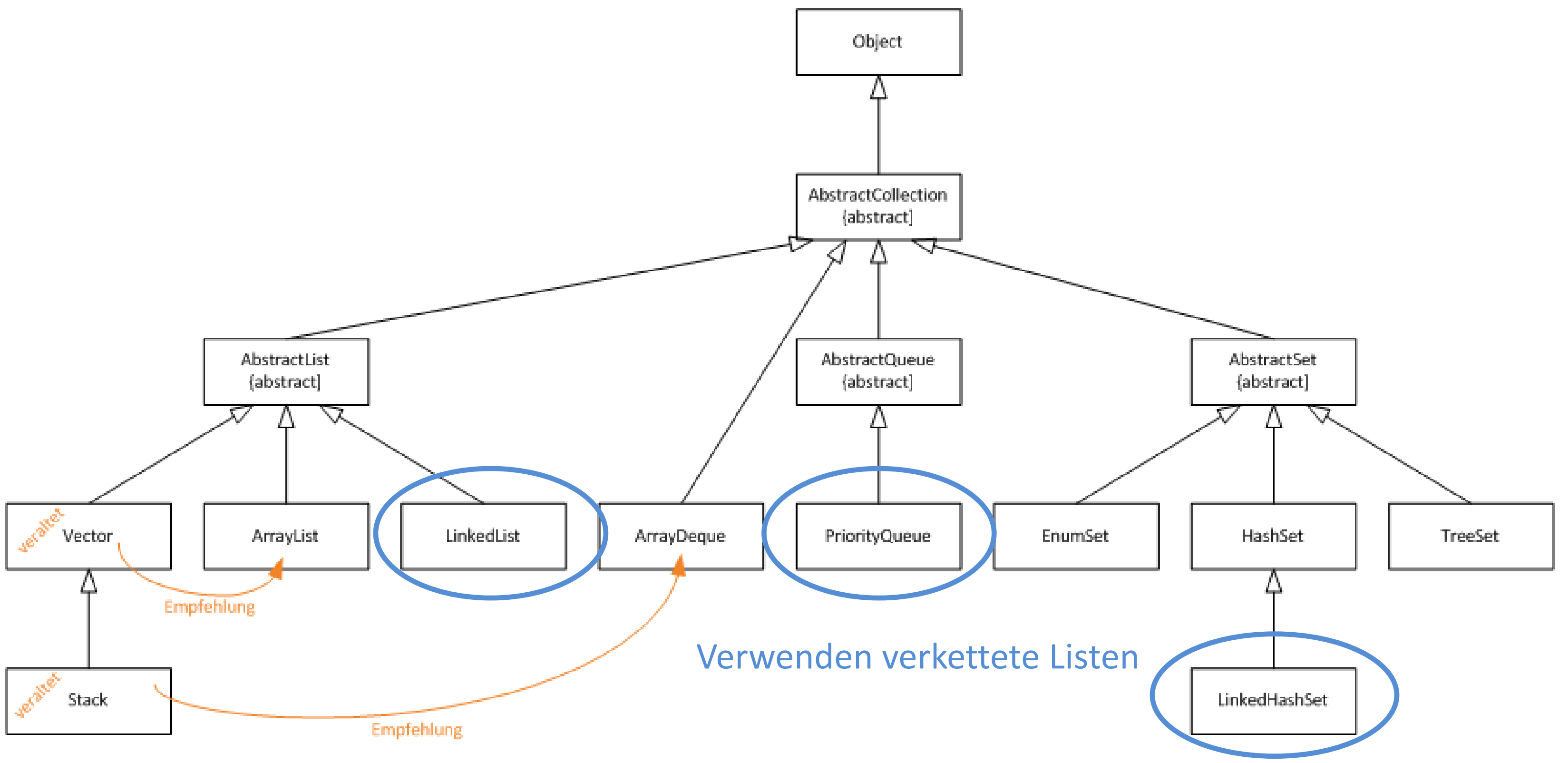
O(1)	Unabhängig von der Anzahl Elemente
O(n)	Laufzeit direkt proportional zur Anzahl Elemente
O(log n)	Laufzeit wächst proportional zum Logarithmus der Anz. Elemente
$O(n^2)$	Laufzeit direkt proportional zum Quadrat der Anzahl Elemente
$O(2^n)$	Laufzeit verdoppelt sich pro zusätzlichem Element

Laufzeit/Komplexität

Teil 2: Java-Collections Framework

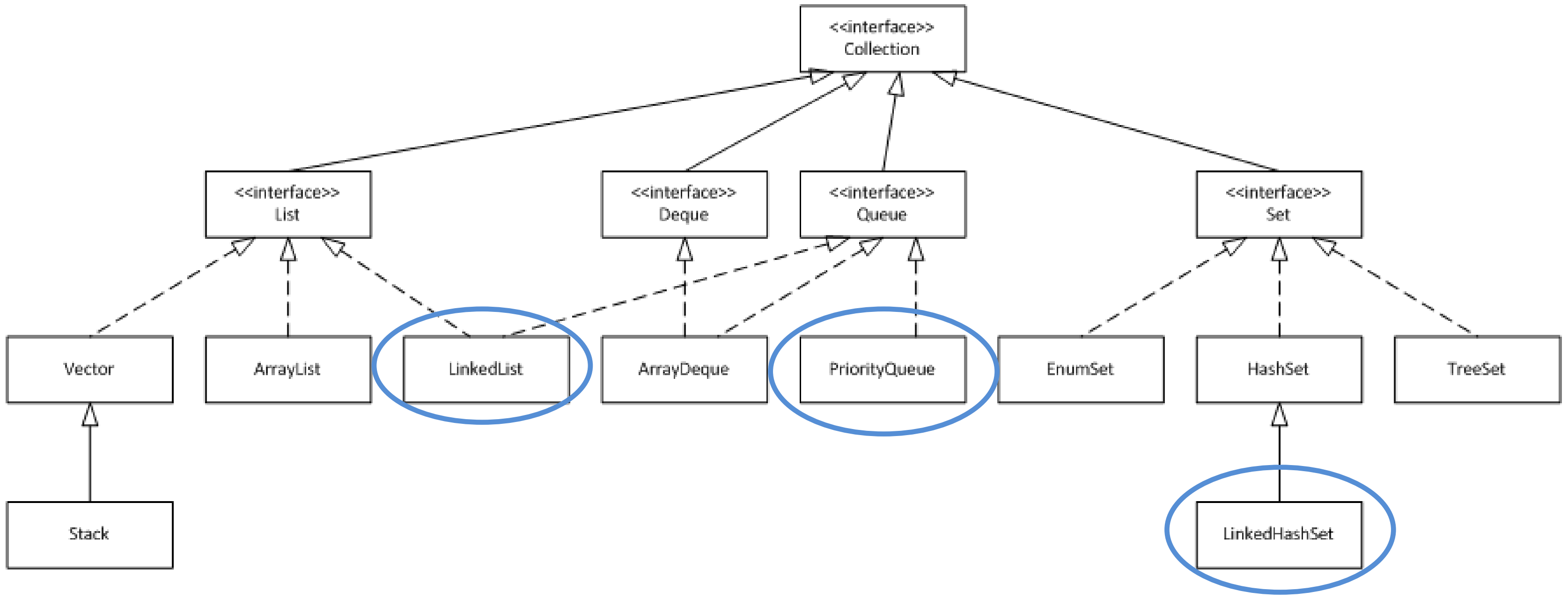
- Klassendiagramm Collections
- Methoden der Klasse LinkedList
- Verwendung von Listen in Programmen

Klassendiagramm AbstractCollection



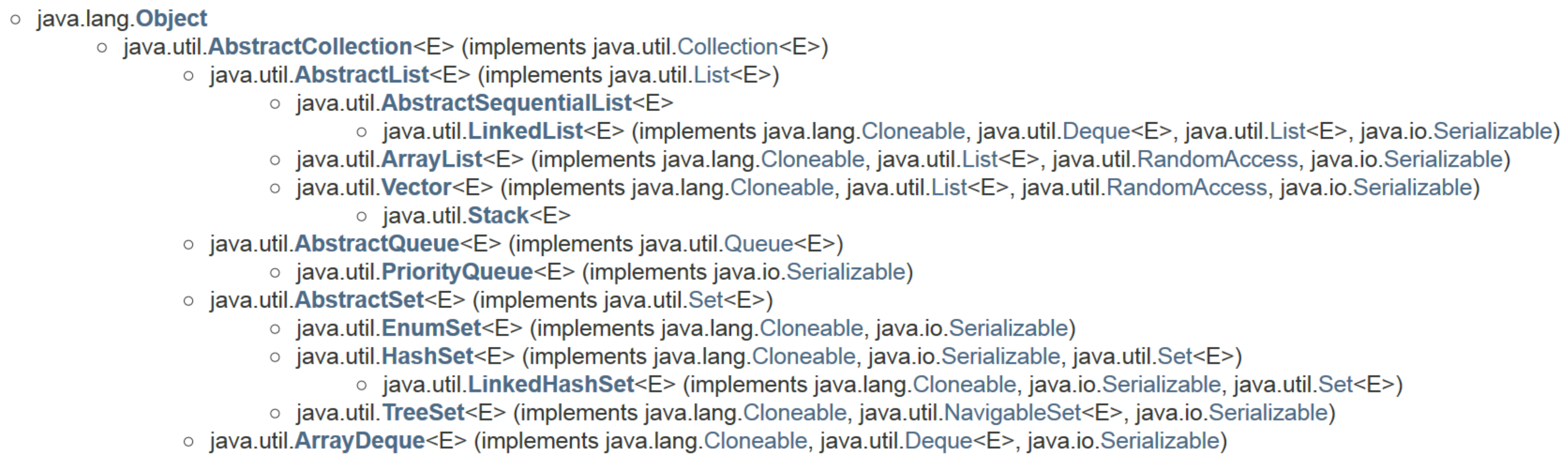
Alle Collection-Klassen sind generisch

Klassendiagramm Collection Interface



Klassenhierarchie - Doku

http://docs.oracle.com/javase/7/docs/api/java/util/package-tree.html



- java.lang.**Object**
  - java.util.**AbstractCollection**<E> (implements java.util.Collection<E>)
    - java.util.**AbstractList**<E> (implements java.util.List<E>)
      - java.util.**AbstractSequentialList**<E>
        - java.util.**LinkedList**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.util.List<E>, java.io.Serializable)
      - java.util.**ArrayList**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
      - java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
        - java.util.**Stack**<E>
    - java.util.**AbstractQueue**<E> (implements java.util.Queue<E>)
      - java.util.**PriorityQueue**<E> (implements java.io.Serializable)
    - java.util.**AbstractSet**<E> (implements java.util.Set<E>)
      - java.util.**EnumSet**<E> (implements java.lang.Cloneable, java.io.Serializable)
      - java.util.**HashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
        - java.util.**LinkedHashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
      - java.util.**TreeSet**<E> (implements java.lang.Cloneable, java.util.NavigableSet<E>, java.io.Serializable)
    - java.util.**ArrayDeque**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.io.Serializable)

Teil 2: Java-Collection Framework

Methoden LinkedList

`boolean`	`add(E e)` Appends the specified element to the end of this list.
`void`	`add(int index, E element)` Inserts the specified element at the specified position in this list.
`boolean`	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
`boolean`	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
`void`	`addFirst(E e)` Inserts the specified element at the beginning of this list.
`void`	`addLast(E e)` Appends the specified element to the end of this list.
`void`	`clear()` Removes all of the elements from this list.
`Object`	`clone()` Returns a shallow copy of this `LinkedList`.
`boolean`	`contains(Object o)` Returns `true` if this list contains the specified element.
`Iterator<E>`	`descendingIterator()` Returns an iterator over the elements in this deque in reverse sequential order.
`E`	`element()` Retrieves, but does not remove, the head (first element) of this list.
`E`	`get(int index)` Returns the element at the specified position in this list.
`E`	`getFirst()` Returns the first element in this list.
`E`	`getLast()` Returns the last element in this list.
`int`	`indexOf(Object o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`int`	`lastIndexOf(Object o)` Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
`ListIterator<E>`	`listIterator(int index)` Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
`boolean`	`offer(E e)` Adds the specified element as the tail (last element) of this list.
`boolean`	`offerFirst(E e)` Inserts the specified element at the front of this list.
`boolean`	`offerLast(E e)` Inserts the specified element at the end of this list.
`E`	`peek()` Retrieves, but does not remove, the head (first element) of this list.
`E`	`peekFirst()` Retrieves, but does not remove, the first element of this list, or returns `null` if this list is empty.
`E`	`peekLast()` Retrieves, but does not remove, the last element of this list, or returns `null` if this list is empty.
`E`	`poll()` Retrieves and removes the head (first element) of this list.
`E`	`pollFirst()` Retrieves and removes the first element of this list, or returns `null` if this list is empty.
`E`	`pollLast()` Retrieves and removes the last element of this list, or returns `null` if this list is empty.
`E`	`pop()` Pops an element from the stack represented by this list.
`void`	`push(E e)` Pushes an element onto the stack represented by this list.
`E`	`remove()` Retrieves and removes the head (first element) of this list.
`E`	`remove(int index)` Removes the element at the specified position in this list.
`boolean`	`remove(Object o)` Removes the first occurrence of the specified element from this list, if it is present.
`E`	`removeFirst()` Removes and returns the first element from this list.
`boolean`	`removeFirstOccurrence(Object o)` Removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
`E`	`removeLast()` Removes and returns the last element from this list.
`boolean`	`removeLastOccurrence(Object o)` Removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
`E`	`set(int index, E element)` Replaces the element at the specified position in this list with the specified element.
`int`	`size()` Returns the number of elements in this list.
`Object[]`	`toArray()` Returns an array containing all of the elements in this list in proper sequence (from first to last element).
`<T> T[]`	`toArray(T[] a)` Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

Stack	Queue	List

Verwendung von Listen in Prog.

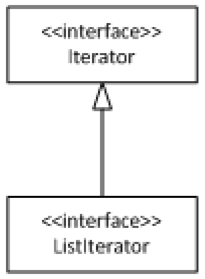
Die LinkedList versteckt ihre innere Datenstruktur (Information Hiding) und es gibt kein getNext().

Wie aber kann dann die Liste verarbeitet werden?

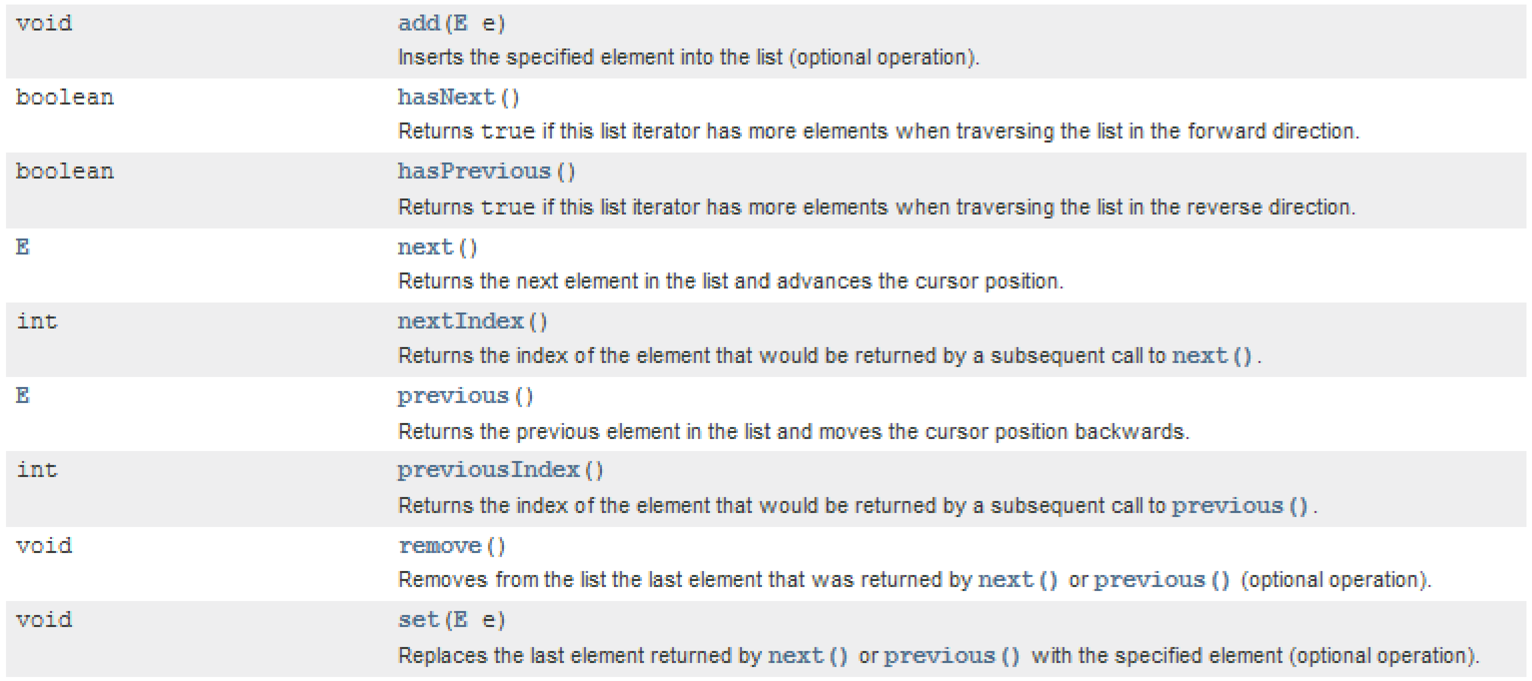
```
LinkedList<String>  playList = new LinkedList<String>();

for (int i = 0, n = playList.size(); i < n; i++) {
    System.out.println(playList.get(i));
}
```

Das ist SEHR ineffizient: $O(n^2)$

ListIterator Interface



LinkedList hat eine Methode die einen / mehrfach den ListIterator<E> mit Cursor und entsprechenden Methoden liefert.



`void`	`add(E e)` Inserts the specified element into the list (optional operation).
`boolean`	`hasNext()` Returns `true` if this list iterator has more elements when traversing the list in the forward direction.
`boolean`	`hasPrevious()` Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
`E`	`next()` Returns the next element in the list and advances the cursor position.
`int`	`nextIndex()` Returns the index of the element that would be returned by a subsequent call to `next()`.
`E`	`previous()` Returns the previous element in the list and moves the cursor position backwards.
`int`	`previousIndex()` Returns the index of the element that would be returned by a subsequent call to `previous()`.
`void`	`remove()` Removes from the list the last element that was returned by `next()` or `previous()` (optional operation).
`void`	`set(E e)` Replaces the last element returned by `next()` or `previous()` with the specified element (optional operation).

Lösung mit Iterator:

```
LinkedList<String> playList = new LinkedList<String>();

for (ListIterator iterator = playList.listIiterator();
        iterator.hasNext();) {
    String s = iterator.next();
    System.out.println(s);
    if(s == "Ende") break;
}
```

Lösung mit For Each Loop:

```
LinkedList<String> playList = new LinkedList<String>();

for (String s : playList) {
    System.out.println(s);
    if(s == "Ende") break;
}
```

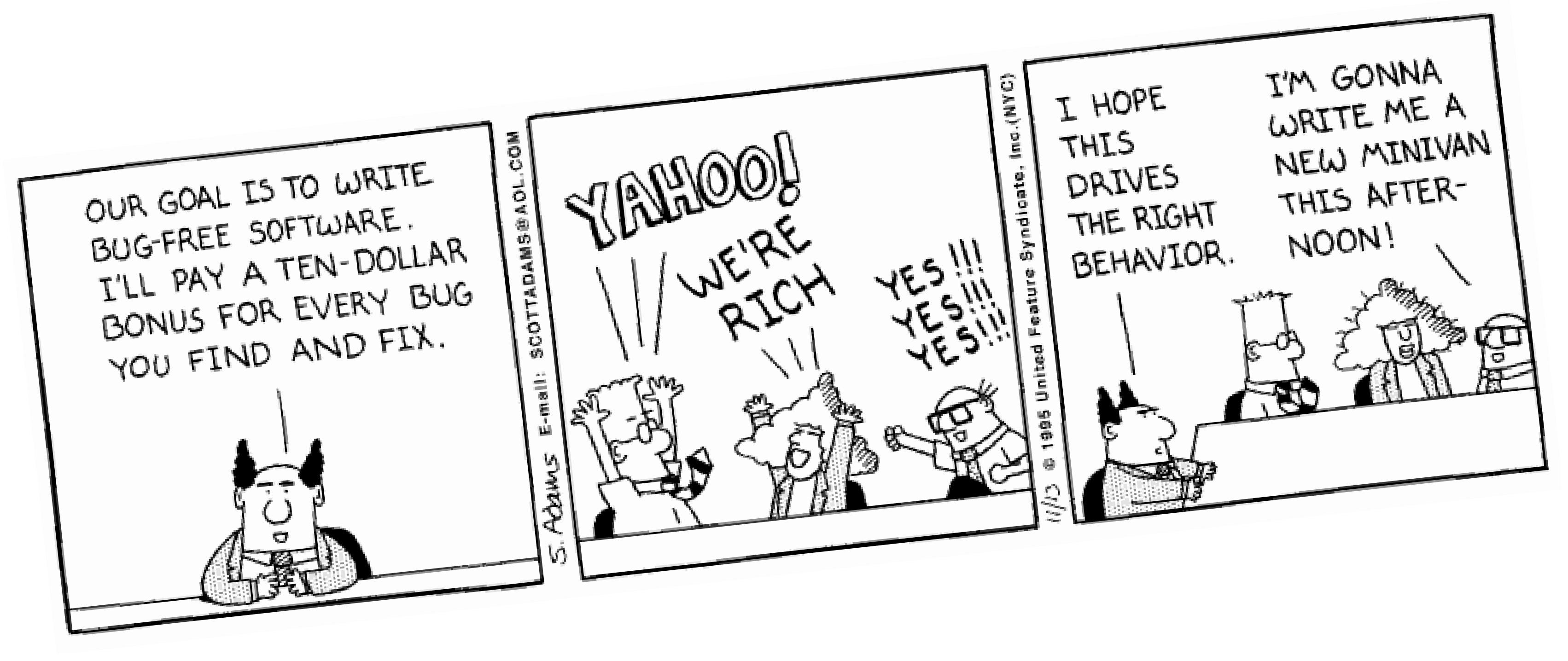
Teil 3: Wann welche Liste/Klasse?

1. Welche Funktionalität benötige ich?
2. Welche Klasse bietet den besten Kompromiss (O-Notation beachten)

	Java Klassen
Array	Array, ArrayList
Verkettete Liste	LinkedList
Doppelt verkettete Liste	LinkedList, LinkedHashSet
Sortierte Liste	PriorityQueue [1)]

[1)] Für Listen und Arrays existieren Sortiermethoden und es gibt weitere, geeignete Klassen ausserhalb der verketteten Listen, z.B. Bäume!

Questions?


OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!
S. Adams E-mail: SCOTTADAMS@AOL.COM
11/13 © 1995 United Feature Syndicate, Inc.(NYC)