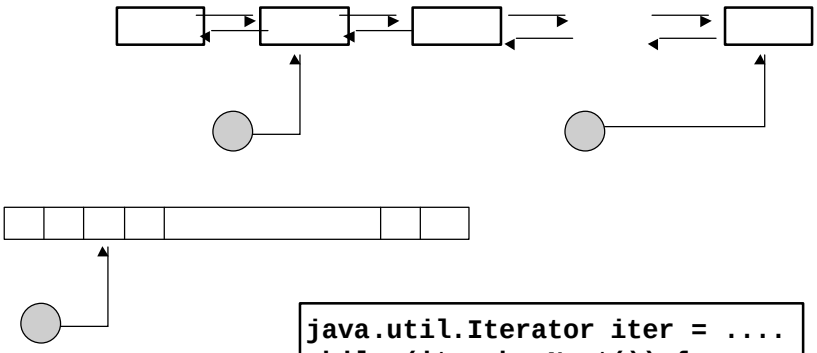
Abstraktion des Traversierens

3.1.2 Iteratoren



.... sind Objekte

```
java.util.Iterator iter = ....
while (iter.hasNext()) {
  ... // tu was
  iter.next()
}
```


Iterator Interface (java.util)

```
public abstract interface Iterator{
```

public boolean hasNext();

// Returns true if the iteration has more elements. (In other words, returns true if next would return an element rather than throwing an exception.)

public Object **next**();

// Returns the next element in the interation.

Throws:

NoSuchElementException - iteration has no more elements.

public void **remove**();

//Removes from the underlying collection the last element returned by the iterator (optional operation).

}

Listen Iterator

```
class DList {

 class DListIterator implements
       java.util.Iterator {
        // Definition der DListIterator-Klasse
  private ....    // aktueller Knoten
  public DListIterator(...) {...}
  public boolean boolean hasNext(){......}
  ....
  }

public java.util.Iterator iterator(){
   return new DListIterator(...);
  }     //implementiert iterator()
 ...... // Listenimplementierung
}
```


Listen Iterator: Verwendung

Klientenprogramm

```
DList myList = new Dlist(); // Listenobjekt
  ......
 java.Util.Iterator iter = myList.iterator();
   // erzeuge Iterator ...
   // ... und verwende ihn
 while (iter.hasNext())
        Object o = iter.next();
```

Doppelt verkettete Liste: Knoten

```
public class DList {

   class Node {
   Object obj;
   Node prev, next;
   public Node (Object o, Node p, Node n) {
     obj = o;
     prev = p;
     next = n; }
   public void setElement (Object o) {
       obj = o;     }   ....
   }
```


Implementierung DListe

```
  class ListIterator implements
       java.util.Iterator {
  ......
  }

 private Node head = null;
 private Node tail = null;
 public DList () {
  head = new Node ();
  tail = new Node ();
  head.setNext (tail);
  tail.setPrevious (head);    }
```

Implementierung DListe

```
public void addFirst (Object o) {
   Node n = new Node (o, head,
    head.getNext ());
    head.getNext ().setPrevious (n);
     head.setNext (n);
}
 public int size () {
  int s = 0;
  Node n = head;
  while (n.getNext () != null) {
     s++;
     n = n.getNext ();
  }
  return s;
 }
```


Implementierung DListe

```
public boolean isEmpty () {
  return head.getNext () == tail;
}
public addAfter (Object o) {
  ....
}

 public java.util.Iterator iterator () {
  return new ListIterator (head.getNext ());
 }.....
}
```

Listen Iterator

```
class ListIterator implements
            java.util.Iterator {

  private Node node = null;

  public ListIterator (Node n) {
    node = n; }
  public boolean hasNext () {
    return node.getNext () != null;    }
   // das letzte Listenelement verweist auf
    //  auf tail!
  public void remove () {
     .... }
```


Listen Iterator

```
public Object next () {
   if (! hasNext ())
    throw new
  java.util.NoSuchElementException ();
   Object o = node.getElement ();
   node = node.getNext ();
   return o; }
}
```