Abstraktionen

Was wir darstellen wollen:
Menge, Folge, ...

abstrakte Datentypen:
Feld, Liste, Keller, Schlange, Hash-Tabelle...

Realisierung in JAVA:
...verkettete Liste...

Abstrakte Datentypen

Ein Datenmodell wird in einer Programmiersprache durch eine Datenstruktur realisiert.

Während es laufend neue Programmiersprachen gibt, ist die Menge der Datenmodelle seit Jahrzehnten konstant.

Eine Darstellung von Daten wird immer in Hinblick auf die Operationen über den Daten definiert!

Ein abstrakter Datentyp ist durch eine Menge von Operationen gegeben, die bezogen auf ein Datenmodell definierte Wirkungen zeigen.

Listen

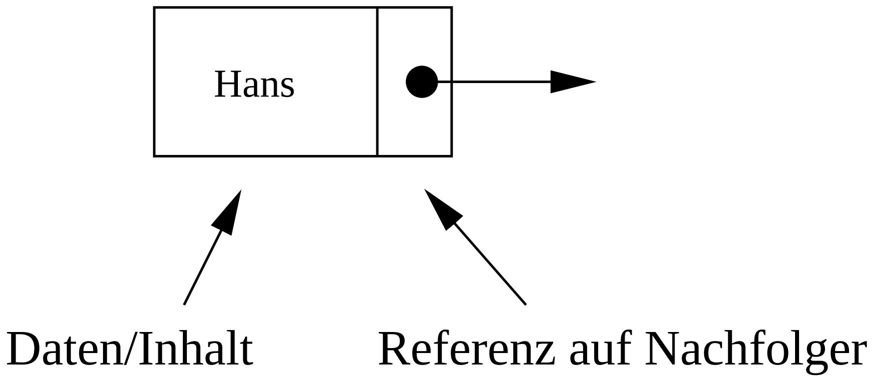
Elemente:
aktuelles Element,
Nachfolger,
Vorgänger,
leere Liste,
erstes Element,
letztes Element.

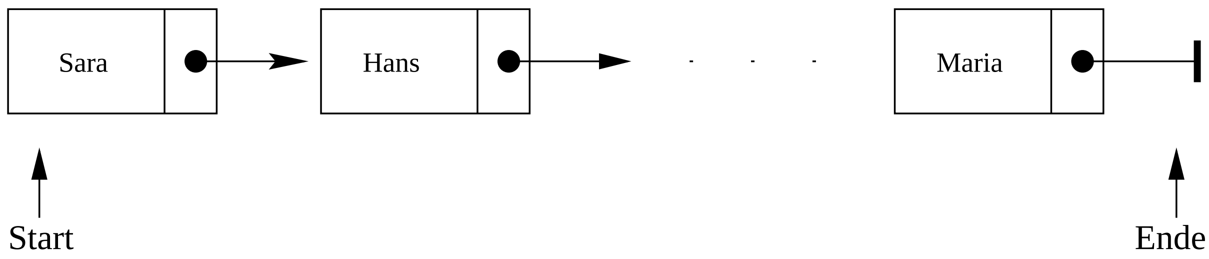
Operationen:
Durchgehen der Liste,
Zurückgeben eines Elements,
Löschen eines Elements,
Einfügen eines Elementes.

Verkette Liste

Realisierung von Listen:

- Zeiger auf aktuelles Element
- Element:
  - Zeiger auf den Nachfolger
  - Inhalt des Elements



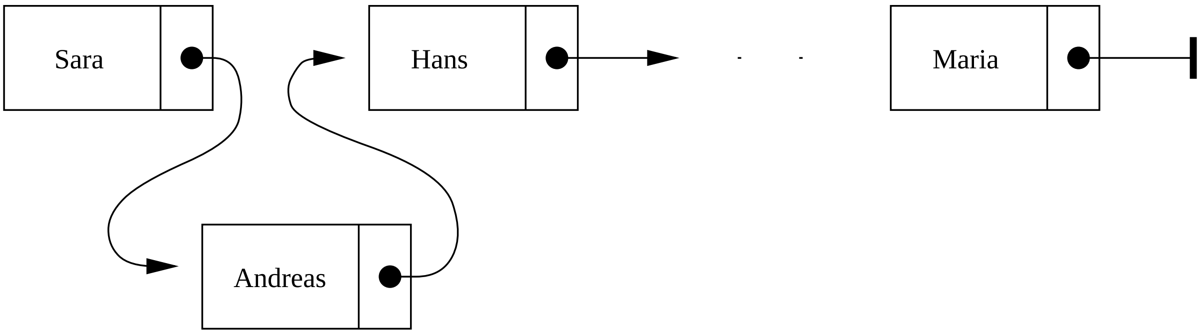


Beim letzten Element ist der Nachfolger *null*.

Einfügen eines neuen Elementes

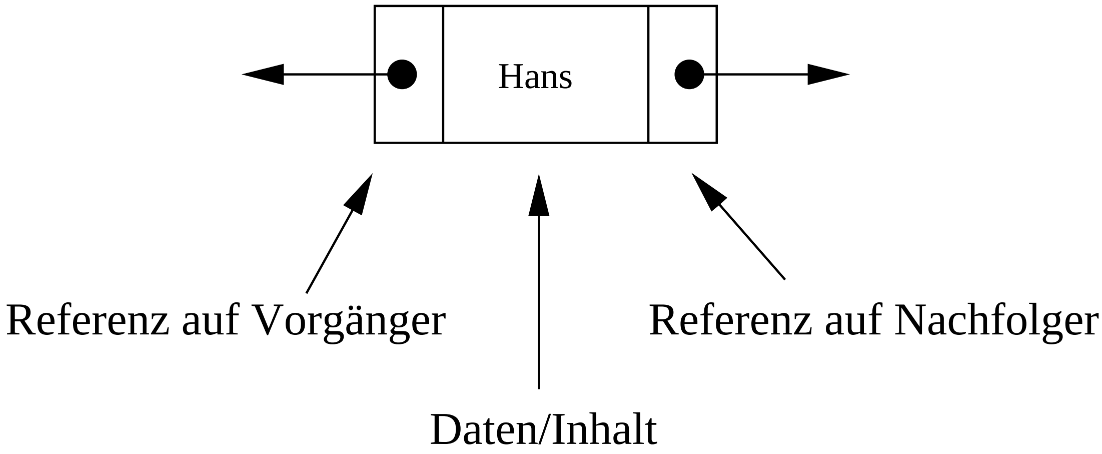
Andreas soll zwischen Sara und Hans.

- Nachfolger von Sara wird Andreas.
- Nachfolger von Andreas wird Hans.



Doppelt verkettete Liste

Abgehen der Liste in beiden Richtungen.



Ein Element hat

- einen Vorgänger
- einen Nachfolger
- einen Inhalt

..in JAVA

Paket: `java.util`

abstrakte Klasse: `AbstractSequential List`

Unterklasse: `LinkedList`

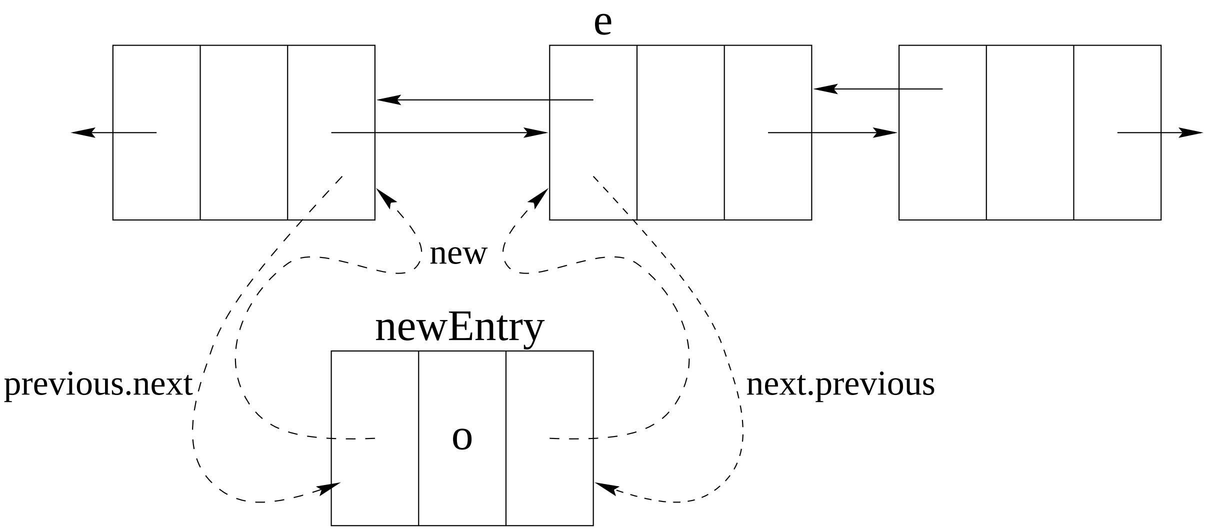
- add(int index, Object element)
- add(Object element)
- addFirst(Object element) – fügt vorn ein
- addLast(Object element) – fügt hinten an
- remove(Object o) – löscht das erste Auftreten des Elements o.
- getFirst() – gibt das erste Element
- getLast() – gibt das letzte Element
- size() – gibt die Anzahl von Elementen an

ein Element

```
private static class Entry {
    Object element;
    Entry next;
    Entry previous;

    Entry (Object element, Entry next, Entry previous) {
        this .element = element;
        this .next = next;
        this .previous = previous;
    }
}
```

Einfügen bei doppelt verketteter Liste



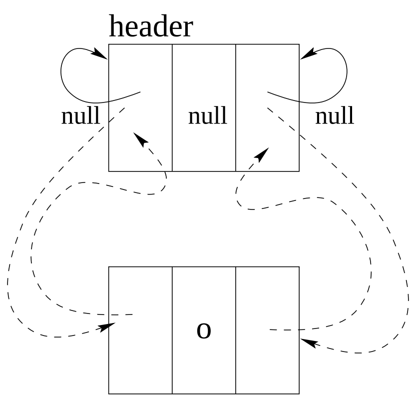
Methode von **LinkedList**:

```
private Entry addBefore (Object o, Entry e) {
    Entry newEntry = new Entry (o, e, e.previous);
    newEntry.previous.next = newEntry;
    newEntry.next.previous = newEntry;
    size++;                                   // Länge Liste
    modCount++;                    // Anzahl Modifikationen
    return newEntry;
}
```

eine verkettete Liste

```
public class LinkedList extends AbstractSequentialList
implements List, Cloneable, java.io.Serializable {
    private Entry header = new Entry (null, null, null);
    private int size = 0;

    /**
    * Constructs an empty Linked List.
    */
    public LinkedList() {
        header.next = header.previous = header;
    }
    /**
    * Inserts the given element at the beginning of this List.
    */
    public void addFirst (Object o) {
        addBefore (o, header.next);
    }
}
```



ein Zeiger

ListIterator realisiert den Zeiger auf das aktuelle Element. Über Methoden senden sich Listenelement und Zeiger Nachrichten.

Die Methode **listIterator()** der Klasse **LinkedList** erzeugt ein Objekt vom Typ **ListIterator** und hängt diesen neuen Zeiger vor das erste Element der Liste, **listIterator(int i)** hängt den erzeugten **ListIterator** vor das i-te Element.

hasNext ist wahr, falls der Nachfolger nicht $null$ ist.

In der Klassenbibliothek von JAVA

Beschreibung einer Methode der Klasse **LinkedList**:

add(Object o)

public boolean add(Object o)

Appends the specified element to the end of this List.

Specified by: add in interface List

Parameters: o - element to be appended to this List.

Returns: true (as per the general contract of Collection.add).

Parameters: Overrides: add in class AbstractList

100m-Lauf

Linked List	
add	Object
listIterator()	ListIterator
listIterator(int i)	ListIterator

ListIterator	
hasNext()	
next()	LinkedList

Lauf100	
teilnehmerEingabe()	LinkedList
zeitenEinlesen(LinkedList L)	ListIterator
main(String args[])	Teilnehmer

```
import java.lang.String;
import java.util.LinkedList;
import java.util.ListIterator;
import AlgoTools.*;
public class Lauf100 {
    static LinkedList laeufer;                    // Die Teilnehmer
    /**
    * Hier werden die Teilnehmer am 100m-Lauf eingelesen
    */
    static LinkedList teilnehmerEingabe () {
        LinkedList liste = new LinkedList ();
        String name;
        int alter;
        while (IO.readString ("Wollen Sie einen Laeufer
eingeben?").equals("ja")) {
            name = IO.readString ("Bitte geben Sie den
Namen des Laeufers ein: ");
            alter = IO.readInt ("Bitte geben Sie das Alter des
Laeufers ein: ");
            liste.add (new Teilnehmer (name, alter));
        }
        return liste;
    }
```

```
/*
    * Der Benutzer gibt hier die Zeiten der Laeufer ein
    */
    static void zeitenEinlesen (LinkedList L) {

        ListIterator i = L.listIterator ();
        Teilnehmer t;
        double zeit;

        while (i.hasNext ()) {
            t = (Teilnehmer)i.next ();
            t.zeit = IO.readDouble ("Bitte geben Sie die Zeit
von "+ t.name + ", "+ t.alter + "ein: ");
        }
    }
```

```
static void main (String[] argv)  {

        Teilnehmer sieger;              // Der Sieger des Laufes
        double bestZeit;                // Laufzeit des Siegers
        IO.println ("Hallo ! Willkommen beim 100m-Lauf !");
        // Eingabe der Laeufer
        laeufer = teilnehmerEingabe ();
        IO.println ("Jetzt fuehren Sie das Rennen durch.");
        // Eingabe der Zeiten durch die Preisrichter
        zeitenEinlesen (laeufer);
        // Besten ermitteln
        ListIterator i = laeufer.listIterator (0);
        sieger = (Teilnehmer)i.next ();
        bestZeit = sieger.zeit;
        while (i.hasNext ())  {
            Teilnehmer t;           // fuer die Suche des Siegers
            t = (Teilnehmer)i.next ();
            if (t.zeit < bestZeit)  {
                sieger = t;
                bestZeit = t.zeit;
            }
        }
```

```
// Besten ausgeben. IO.println ("\n Auswertung:");
        IO.println ("——————————————————————————");
        IO.println ("Der Gewinner des diesjaehrigen
100m-Laufes ist:");
        IO.println ("Name: "+ sieger.name + ", "+
sieger.alter);
        IO.println ("Mit einer Bestzeit von "+ bestZeit);
    }

 }
```

Teilnehmer

```
public class Teilnehmer {

    public String name;
    public int alter;
    public double zeit;
    Teilnehmer (String _name, int _alter) {
        name = _name;
        alter = _alter;
    }

}
```