RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

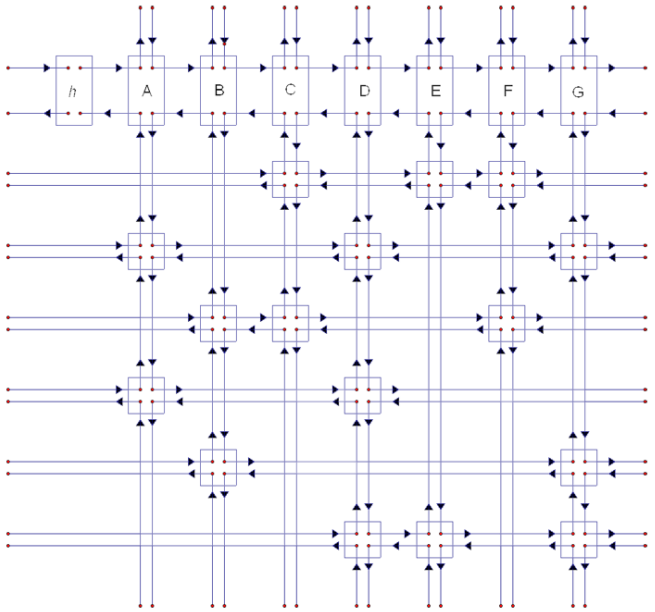
Sommersemester 2023

Datenstrukturen – Java Collection Framework

Einführung

Mit Hilfe einer Datenstruktur werden Daten zusammenhängend in einer bestimmten Struktur gespeichert

- Zugriff und Verwaltung von großen Datenmengen wird vereinfacht
- Effiziente Algorithmen werden ermöglicht
- Spezielle Strukturen für spezielle Problemstellungen

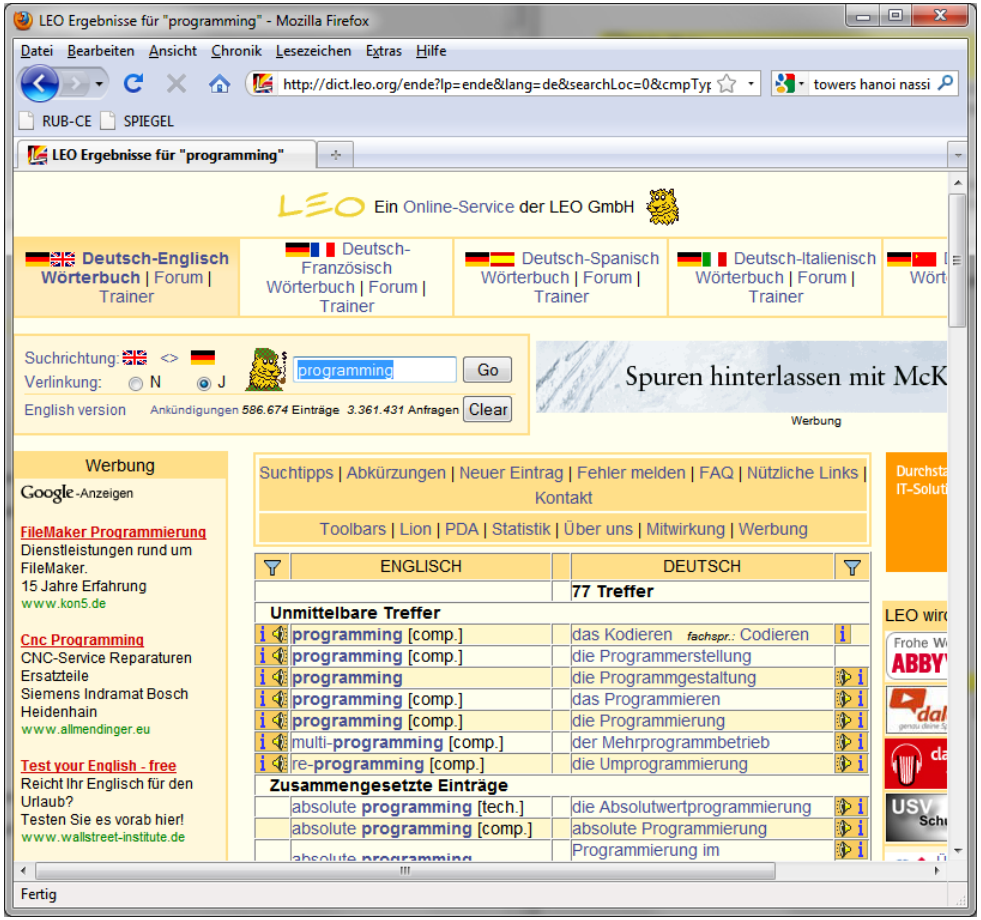


Datenstrukturen

Beispiele - Abbildungen

Elektronisches Wörterbuch

- Zuordnung von Wörtern
- Schnelle Suche

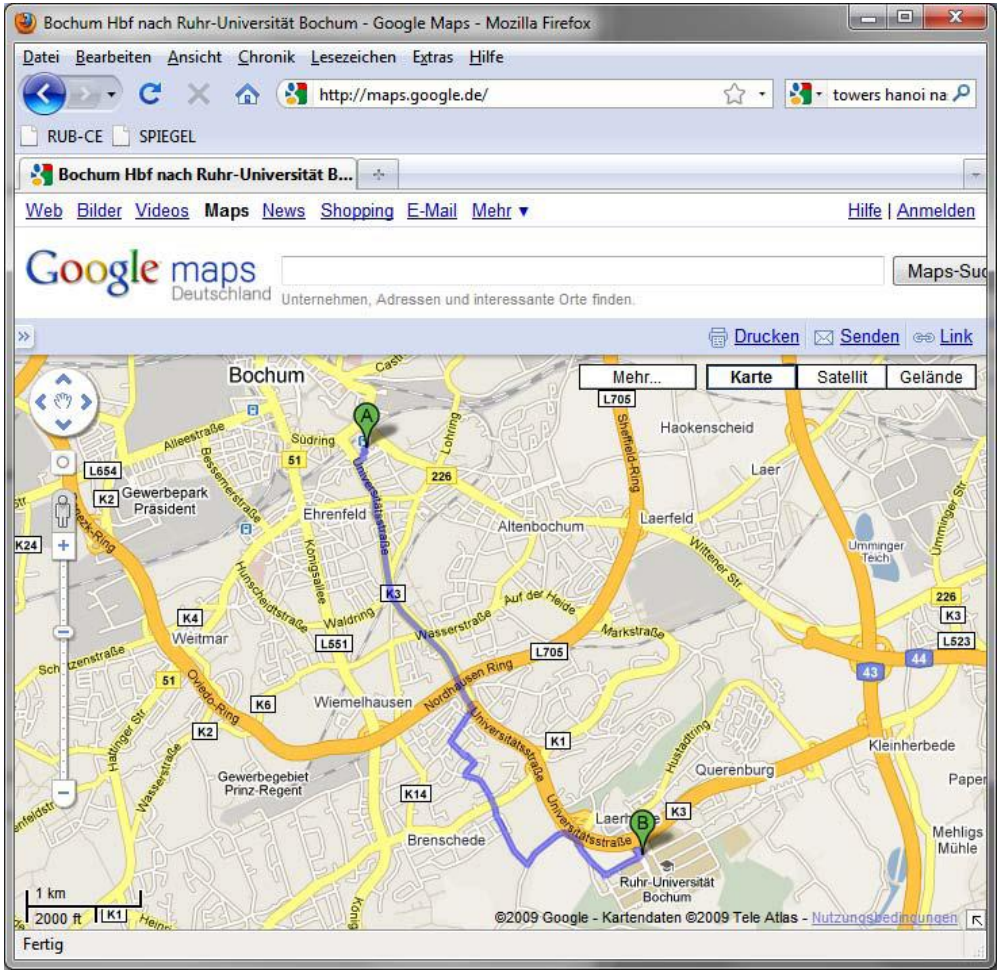


Datenstrukturen

Beispiele - Graphen

Routenplaner

- Knoten und Kanten
- Kürzeste Wege finden

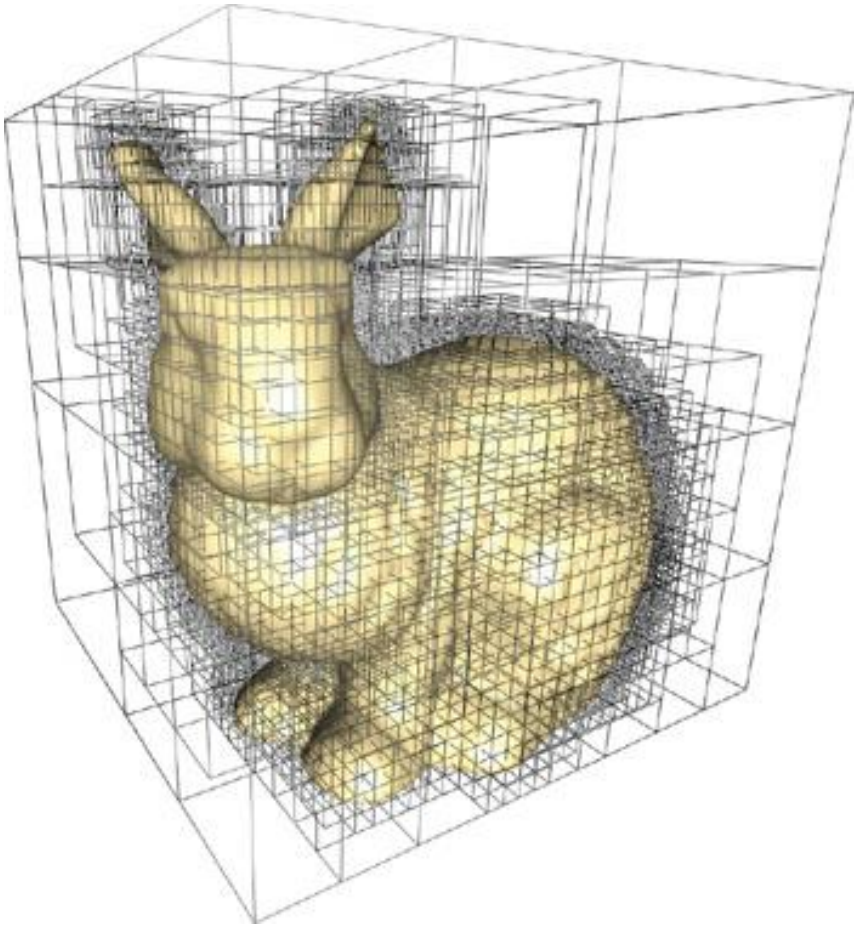


Datenstrukturen

Beispiele - Bäume

Gebietszerlegung

- Hierarchischer Aufbau
- Grundlage für Berechnungen

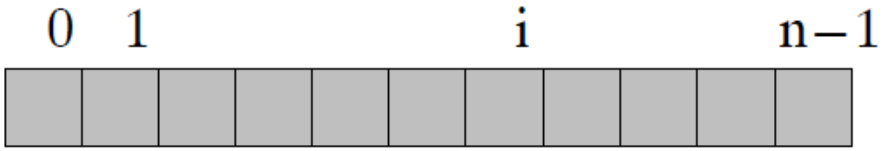


Datenstrukturen

Generelle Speicherkonzepte

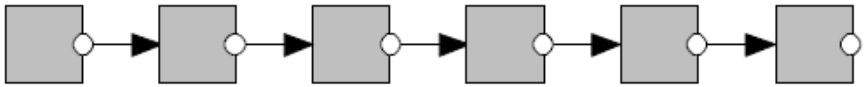
Array Strukturen

- Daten werden in Feldern abgelegt
- Direkter Zugriff mittels Index



Verknüpfte Strukturen

- Daten werden in erweiterten Objekten gespeichert
- Objekte besitzen Verweise auf andere Datenobjekte



Datenstrukturen

Folgen

Folgen sind strukturierte Mengen von Elementen

- Festgelegt Reihenfolge der Elemente
- Element kann mehrfach vorhanden sein
- `null` Elemente sind erlaubt

$$F := < e_0, e_1, ..., e_{n-1} >$$

Beispiel Fibonacci-Folge

- Für die beiden ersten Zahlen lauten null und eins
- Jede weitere Zahl ist die Summe ihrer beiden Vorgänger

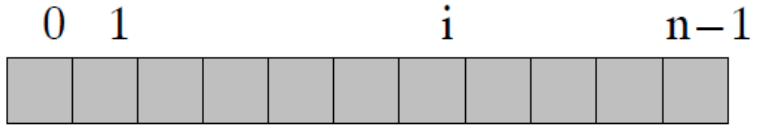
$$F := < 0, 1, 1, 2, 3, 5, 8, ... >$$

Datenstrukturen

Reihenfolge der Elemente

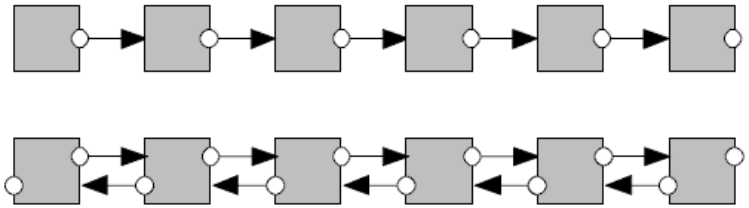
Absolute Anordnung

- Jedem Element wird ein eindeutiger Index zugeordnet
- Wird auch als Reihe bezeichnet



Relative Anordnung

- Jedem Element wird ein Vorgänger und/oder Nachfolger zugeordnet
- Wird auch als Kette bezeichnet

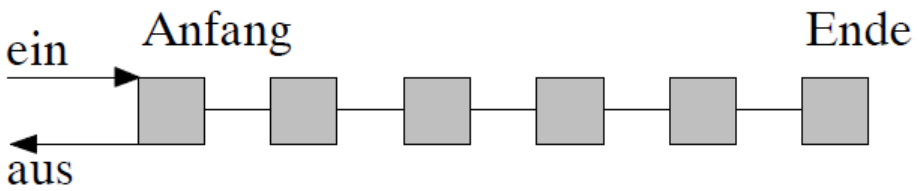


Datenstrukturen

Spezielle Folgen mit relativen Anordnungen

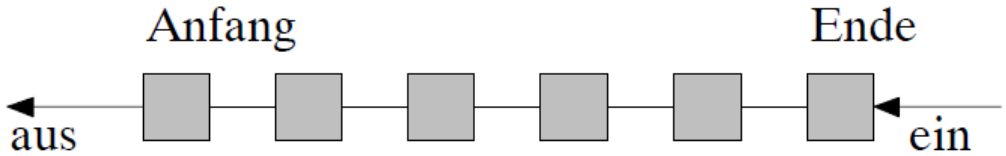
Stapel (last in, first out)

- Beim Aufbau wird ein neues Element am Anfang eingetragen
- Beim Abbau wird immer das Element am Anfang entnommen



Schlange (first in, first out)

- Beim Aufbau wird ein neues Element am Ende eingetragen
- Beim Abbau wird immer das Element am Anfang entnommen



Datenstrukturen

Klasse ArrayList

Die Klasse `ArrayList<E>` implementiert eine Folge mit absoluter Anordnung. Beim Einfügen und Löschen wird die Folge automatisch vergrößert und verkleinert. Folgende Methoden stehen zur Verfügung:

```
// Füge x am Ende ein
public boolean add(E x);
// Füge x an der Stelle index ein
public boolean add(int index, E x);
// Folge enthält x?
public boolean contains(E x);
// Anzahl Elemente
public int size();

// Gebe Element an der Stelle index zurück
public E get(int index);
// Ersetze Element x an der Stelle index
public E set(int index, E x);
// Entferne Element an der Stelle index
public boolean remove(int index);
```

Generische Klassen

Zur Vermeidung von mehrfachen Implementierungen können Platzhalter für Datentypen verwendet werden

- Implementierungen für `double` Zahlen und `Rectangle` Objekte

```
public class ArrayListDouble {
        public boolean add(Double x) …
public class ArrayListRectangle {
        public boolean add(Rectangle x) …
```

- Implementierung für beliebige Datentypen `E`

```
public class ArrayList<E> …
        public boolean add(E x) …
```

Datenstrukturen

Generische Klassen

Soll eine generische Klasse in einem Programm verwendet werden, muss ein Datentyp angegeben werden

- Mengen für `double` Zahlen und `Rectangle` Objekte

```
ArrayList<Double> listA;

ArrayList<Rectangle> listB;
```

- Es können nur Objekte verwendet werden, die zum angegebenen Datentyp passen

```
listA.add(12.0);        Rectangle r1;

                        listB.add(r1);

// Nicht möglich

listB.add(new Box(0.0, 0.0, 0.0));
```

Datenstrukturen

Beispiel ArrayList

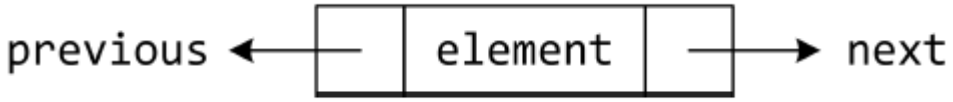
```
public class ArrayListDoubleDemo {

    public static void main(String[] args) {

        ArrayList<Double> list = new ArrayList<Double>();

        list.add(5.0);
        list.add(4.0);
        list.set(0, 1.0);

        list.add(1, 3.0);
        list.add(5.0);

        list.remove(2);
    }
}
```

Datenstrukturen

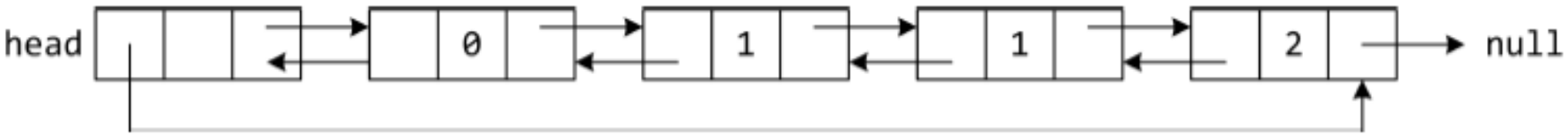
Klasse LinkedList

Eine relative Anordnung wird mit speziellen Verknüpfungsobjekten umgesetzt werden

- Für jedes Element wird ein internes Verknüpfungsobjekt erzeugt
- Das Verknüpfungsobjekt enthält die Verweise auf den Vorgänger und/oder Nachfolger



- Ein Startobjekt (head) kennzeichnet den Beginn der Kette



Datenstrukturen

Klasse LinkedList

Die Klasse `LinkedList<E>` implementiert eine Folge mit relativer Anordnung. Folgende Methoden stehen zusätzlich zur Verfügung:

```
// Füge x am Anfang ein
public boolean addFirst(E x);
// Füge x am Ende ein
public boolean addLast(E x);

// Gebe erstes Element zurück
public E getFirst();
// Gebe letztes Element zurück
public E getLast();

// Entferne erstes Element
public boolean removeFirst();
// Entferne letztes Element
public boolean removeLast();
```

Java Collection Framework

Viele Schnittstellen und Klassen für Datenstrukturen werden durch das Java Collection Framework (Package `java.util`) zur Verfügung gestellt

- Mengen: `HashSet`
- Folgen: **`ArrayList`**, **`LinkedList`**
- Spezielle Ketten: `Stack`, `PriorityQueue`
- Geordnete Mengen: `TreeSet`
- Abbildungen: `HashMap`, `TreeMap`
- ...

Datenstrukturen

Java Collection Framework

Die Schnittstellen und Klassen werden im Rahmen der Java Dokumentation beschrieben

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 6

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.util

Interface Collection<E>

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

```
public interface Collection<E>
extends Iterable<E>
```

The root interface in the *collection hierarchy*. A collection represents a group of objects, known as its *elements*. Some collections allow duplicate elements and others do not. Some are ordered and others unordered. The JDK does not provide any *direct* implementations of this interface: it provides implementations of more specific subinterfaces like `Set` and `List`. This interface is typically used to pass collections around and manipulate them where maximum generality is desired.

Bags or *multisets* (unordered collections that may contain duplicate elements) should implement this interface directly.

Datenstrukturen

Verwendung des Iterators

Häufig ist es notwendig alle Elemente einer generischen Datenstruktur zu durchlaufen. Für einen einheitlichen Zugriff auf alle Elemente einer Datenstruktur wird ein `Iterator<E>` verwendet. `Iterator`-Objekte werden häufig im Rahmen von Schleifen eingesetzt.

- Normale `for` Schleife

```
for (Iterator<Point2D> iter = points.iterator(); iter.hasNext(); ) {
      Point2D p = iter.next();
      System.out.println(p);
}
```

- Vereinfachte `for` Schleife

```
for (Point2D p : points) {
      System.out.println(p);
}
```

- Normale `while` Schleife

```
Iterator<Point2D> iter = points.iterator();
while (iter.hasNext()) {
      Point2D p = iter.next();
      System.out.println(p);
}
```