

Listen und abstrakte Datentypen

Programmieren Tutorium Nr.17

Aleksandr Zakharov | 10. Dezember 2025

Organisatorisches

Übungsblatt 3 Abgabe

10. Dezember 2025 12:00 Uhr - 18. Dezember 2025 6:00 Uhr

Organisatorisches



Übungsblatt 2



Wiederholung



Listen



Aufgabe: Liste



Abstrakte Datenstrukturen



Übungsblatt 2 - Besprechung

- Je Klasse eigene Datei
- Enum nicht nur anlegen, auch das passende Attribut
- Keine inline enums
- Objektorientierung! (Klassen als Datentyp nutzen, ...)
- Datenkapselung (`private ...`)
- Wiederverwendung eigener Methoden (Vermeidung von Code-Duplizierung)
- Utility-Klassen: `final class`, privaten Konstruktor und die Methoden/Attribute `static`. Und **keine Instanzen** dessen!
- `if(var == true) ⇔ if(var)`
- `equals()` verwenden
- Sinnvolle Bezeichner & Kommentare
- Checkstyle

Übungsblatt 2 - Besprechung

- Falscher Schleifentyp (`while` statt `for`, `for` statt `for-each` verwendet)
- Nicht die schon gegebene Funktionalität reimplementieren!
- Mit `chars` direkt rechnen, nicht die ASCII-Tabelle-Kodierungen als integers verwenden!
- Auskommentierten Quelltext nicht abgeben
- Magic Numbers
- Vergleichsrichtung: `"text".equals(var)`
- Bitte kein `while(true)`
- Kompilieren, Ausführen, **TESTEN**
- Im Artemis nachschauen ob es wirklich kompiliert hat!

Weitere Anmerkungen

- Der Name einer Variablen hat nichts mit der Objektidentität zu tun:

```
FireTruck truck = new FireTruck();  
FireTruck anotherTruck = truck;  
// Beide Variablen zeigen auf den GLEICHEN Truck  
truck == anotherTruck // true
```

Weitere Anmerkungen

- Der Name einer Variablen hat nichts mit der Objektidentität zu tun:

```
FireTruck truck = new FireTruck();  
FireTruck anotherTruck = truck;  
// Beide Variablen zeigen auf den GLEICHEN Truck  
truck == anotherTruck // true
```

- Eine Variable ist nicht notwendig, um ein Objekt zu erstellen

```
// Erzeugt einen FireTruck und ruft dessen Konstruktor auf  
new FireTruck();
```

Immutable Objects

- Ein „Immutable Object“ ist ein Objekt bei dem der Zustand lediglich einmal am Anfang gesetzt wird und danach nie wieder verändert werden kann.
Bsp.: alle Strings
- „Immutable Objects“ sind z.B. sehr hilfreich wenn die entsprechenden Objekte kopiert werden müssen, da dann lediglich die Referenz kopiert werden muss.
- Sie enthalten oftmals nur *Daten* \implies `equals` (`hashCode`) überschreiben lohnt sich wahrscheinlich
- Siehe Wikipedia - Immutable Object
- In Java ist es durch den Record-Klassentyp dargestellt (ihr könnt damit während den Ferien herumspielen)

Wiederholung

Wahr oder Falsch?

`int[5] integers = { 1, 2, 3, 4, 5 } kompiliert.`

Organisatorisches
○

Übungsblatt 2
○○○○

Wiederholung
●○○

Listen
○○○○○○○

Aufgabe: Liste
○

Abstrakte Datenstrukturen
○

Wiederholung

Wahr oder Falsch?

`int[5] integers = { 1, 2, 3, 4, 5 }` kompiliert.

Falsch!

Die Größe eines Arrays ist nicht Teil des Typs.

Wiederholung

Wahr oder Falsch?

`array.length()` gibt die Länge eines Arrays zurück.

Wiederholung

Wahr oder Falsch?

`array.length()` gibt die Länge eines Arrays zurück.

Falsch!

Die Länge eines Arrays ist ein *Attribut*, keine Methode.

Wiederholung

Wahr oder Falsch?

`array[2]` gibt das 2. Element eines Arrays zurück.

Organisatorisches



Übungsblatt 2



Wiederholung



Listen



Aufgabe: Liste



Abstrakte Datenstrukturen



Wiederholung

Wahr oder Falsch?

`array[2]` gibt das 2. Element eines Arrays zurück.

Falsch!

Die Indizierung startet bei **0**!. Daher gibt dies das dritte Element im Array zurück.

Listen

Arten

- Single linked List
- Double linked List
- ArrayList (hier nicht erklärt)

Listen

Arten

- Single linked List
- Double linked List
- ArrayList (hier nicht erklärt)

Aufbau

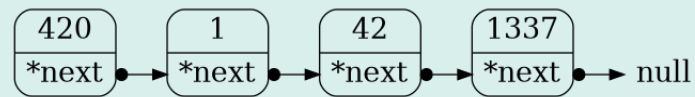
- Es gibt verschiedene `ListElements`, die sich gegenseitig referenzieren
- Durch diese rekursiven Referenzen lassen sich komplexe Datenstrukturen bauen
- Jedes `ListElement` hält die zu speichernden Daten und eine oder mehrere Referenzen auf andere Listenelemente
- Wie genau diese Elemente aufgebaut sind hängt von Implementierung und konkreter Datenstruktur ab

Single Linked List

Beispielimplementierung für ein Element

```
class Element {  
    //Das zu speichernde Element  
    long element;  
    //Referenz auf das naechste Element in der Liste  
    Element next;  
}
```

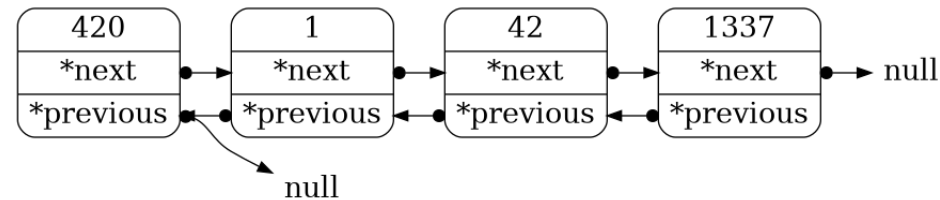
Schaubild



Double Linked List

Beispielimplementierung für ein Element

```
class Element {  
    //Das zu speichernde Element  
    long element;  
    //Referenz auf das naechste Element in der Liste  
    Element next;  
    //Referenz auf das letzte Element in der Liste  
    Element previous;  
}
```



Binäre Bäume

Eigenschaften

- Speichern pro Element ein Datum (Singular von Daten)
- Haben einen linken und einen rechten Teilbaum
- Auf der Folie zu Aufrufbäumen haben wir schon Binärbäume gesehen

Binäre Bäume

Eigenschaften

- Speichern pro Element ein Datum (Singular von Daten)
- Haben einen linken und einen rechten Teilbaum
- Auf der Folie zu Aufrufbäumen haben wir schon Binärbäume gesehen

Frage

Wie könnte man sowas implementieren?

Iteratoren

Problemstellung

- Viele Arten von Sammlungen und Mengen
- Verschieden aufgebaut und implementiert
- Unterschiedliche Schnittstellen (z.B. für Elementzugriff)

Gesucht wird ein einheitlicher Weg, über alle Elemente zu iterieren \Rightarrow Iterator-Objekte

Iteratoren

Problemstellung

- Viele Arten von Sammlungen und Mengen
- Verschieden aufgebaut und implementiert
- Unterschiedliche Schnittstellen (z.B. für Elementzugriff)

Gesucht wird ein einheitlicher Weg, über alle Elemente zu iterieren \Rightarrow Iterator-Objekte

Methoden

Sammlungen (Collections) bieten Methode zur Erzeugung eines Iterators an: `collection.iterator()` Iteratoren bieten die folgenden Methoden an:

- `hasNext()`: true, wenn es nächstes Element gibt, false sonst
- `next()`: gib das aktuelle Element zurück und bewege zum nächsten
- `remove()`: entferne aktuelles Element aus der Sammlung

Iteratoren

Beispiel

```
public void someMethod(List list) {  
    // erzeuge neuen Iterator  
    Iterator iterator = list.iterator();  
  
    while(iterator.hasNext()) {  
        // bekomme das aktuelle Element  
        Type element = iterator.next();  
  
        // Element ausgeben und aus Liste entfernen  
        System.out.println(element);  
        iterator.remove();  
    }  
}
```

Listen-API

Was bietet mir die API?

- Queue: `java.util.LinkedList<E>`
- Stack: `java.util.Stack<E>`
- PriorityQueue: `java.util.PriorityQueue<E>` (ginge aber auch mit *LinkedList*)

Was ist <E>?

Listen-API

Was bietet mir die API?

- Queue: `java.util.LinkedList<E>`
- Stack: `java.util.Stack<E>`
- PriorityQueue: `java.util.PriorityQueue<E>` (ginge aber auch mit *LinkedList*)

Was ist <E>?

Mit <E> ist der Datentyp der (z.B.) Liste generisch.

Die gleiche Listenimplementation kann also mit verschiedensten Datentypen umgehen, OHNE dass jedesmal eine komplett neue Listenimplementation erstellt werden muss.

Aufgabe: Liste

Liste von Punkten

Implementieren Sie die Klasse `LinkedList`, die Punkte (Klasse `Point`) mittels einer Liste verwaltet. Schreiben Sie hierzu die für Listen üblichen Schnittstellen: `LinkedListInterface.java`. Verwenden Sie hierzu die gegebene Klasse `Point.java`.

einige Schnittstellenmethoden (nicht vollständig)

- **public void** `add(Point p);`
- **public boolean** `contains(Point p);`
- **public** `Point get(int index);`
- **public** `Point getFirst();`
- **public** `Point getLast();`
- **public int** `indexOf(Point p);`
- **public void** `remove(int index);`

Abstrakte Datenstrukturen

Definition

Ein Abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen.

Abstrakte Datenstrukturen

Definition

Ein Abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen.

Genauer

- Definiert nur Schnittstellen auf diese Daten
- Kapselt dadurch die Daten nach außen
- Funktioniert im inneren komplexer und implementierungsabhängig

Abstrakte Datenstrukturen

Definition

Ein Abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen.

Genauer

- Definiert nur Schnittstellen auf diese Daten
- Kapselt dadurch die Daten nach außen
- Funktioniert im inneren komplexer und implementierungsabhängig

Beispiele

- Listen (verschiedene Arten)
- Graphen (Und alle „Unterdatenstrukturen“)
- Viel mehr...

Organisatorisches
○

Übungsblatt 2
○○○○

Wiederholung
○○○

Listen
○○○○○○○

Aufgabe: Liste
○

Abstrakte Datenstrukturen
●