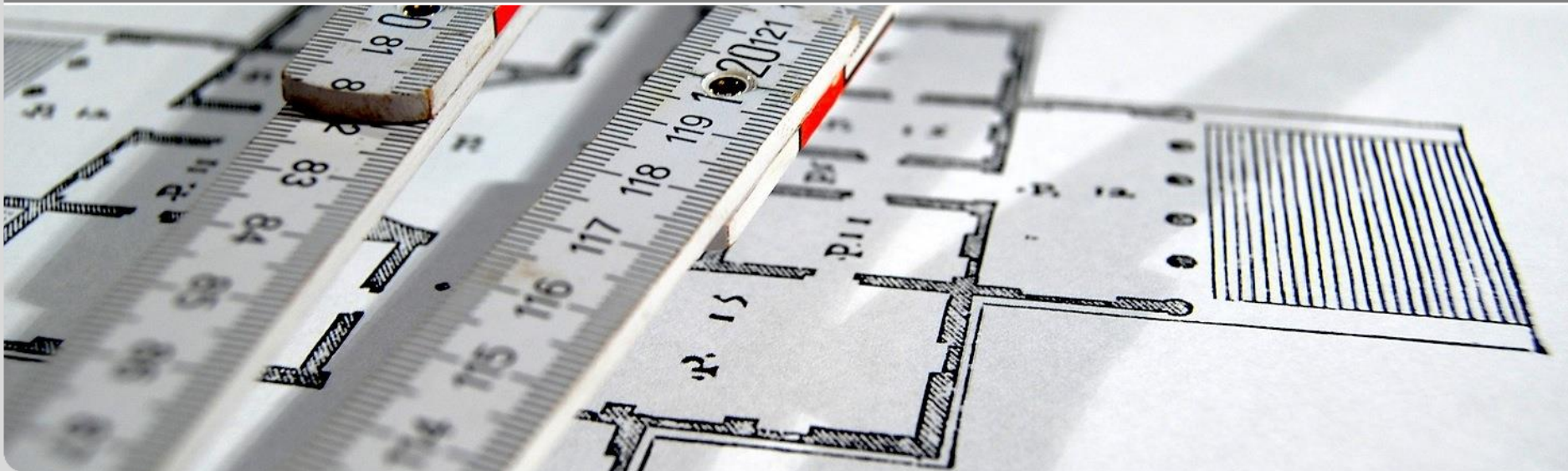


# Programmieren-Tutorium Nr. 10

12. Tutorium | Jonas Ludwig  
**Java Collections, Abschlussaufgaben**

Architecture-driven Requirements Engineering – Institut für Programmstrukturen und Datenorganisation – Fakultät für Informatik



# Was machen wir Heute?

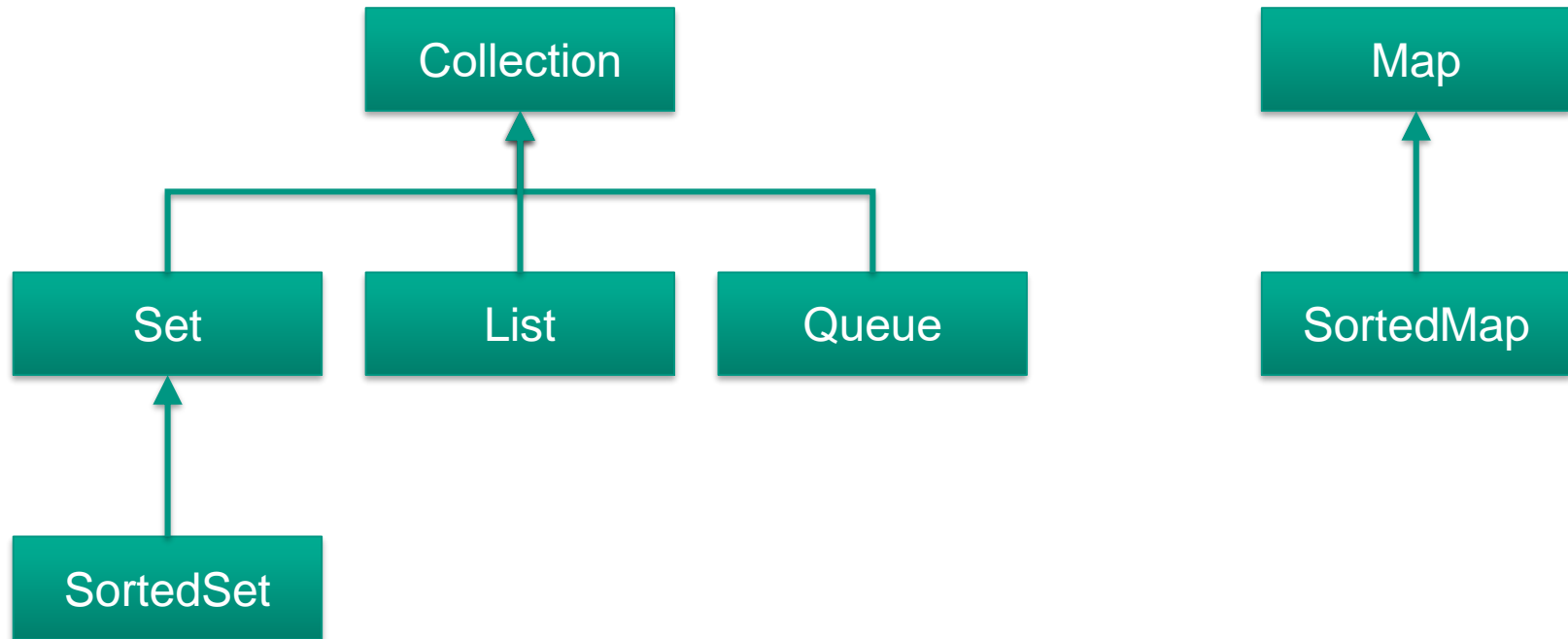
- Übungsblatt 5
- Java Collections
- Abschlussaufgaben

# Übungsblatt 5 - Korrektur

- Aufgabe A Ø 14.28 P von 20P
  - Magic Numbers vermeiden => Konstanten verwenden
  - Auf lesbaren Programmcode achten
  - Speichern von Attributen als String wenn nicht unbedingt notwendig vermeiden
  
- Tipp: Falls Abschlussaufgaben ähnlich: Musterlösung anschauen

# Java Collections Framework – Interfaces

## ■ Haupt-Interfaces des Collection-Frameworks:



- Die Haupt-Interfaces kapseln verschiedene Arten von Collections
- Ermöglichen die implementierungsunabhängige Manipulation
- Haupt-Interfaces bilden eine Hierarchie

# Das Interface `Collection<E>`

- Alle Collection-Interfaces sind generisch:
  - `public interface Collection<E> { ... }`
- Bei Deklaration einer Collection kann und sollte angegeben werden, welche Elemente enthalten sind:
  - `Collection<Point> c;`
  - Compiler kann prüfen, ob die der Collection hinzugefügten Elemente vom Typ `Point` sind
- `Collection<E>` trifft beispielsweise keine Aussage darüber, ob
  - die Elemente der Collection geordnet sind
  - die Collection Duplikate enthält
- `Collection<E>` wird verwendet, wenn möglichst wenig Einschränkungen gelten sollen bzw. bekannt sind
- `Set`, `List` und `Queue` sind spezifischere Collections

# Das Interface Collection<E>

- Collection<E> definiert folgende Methoden:

```
public interface Collection<E> extends Iterable<E> {  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(E element);    // optional  
    boolean remove(Object element); // optional  
    Iterator<E> iterator();  
    boolean containsAll(Collection<?> c);  
    boolean addAll(Collection<? extends E> c); // optional  
    boolean removeAll(Collection<?> c);    // optional  
    void clear();    // optional  
  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
}
```

# Das Interface `Set<E>`

- Ein `Set<E>` ist eine `Collection<E>`, die keine Duplikate enthält
- Modelliert eine mathematische Menge
- `Set<E>` enthält ausschließlich die Methoden von `Collection<E>`
- Zwei `Set<E>`-Instanzen sind gemäß der `equals`-Methoden identisch, wenn diese dieselben Elemente enthalten
- Die Java-API stellt unter anderem die konkrete Implementierung `HashSet` zur Verfügung

- Beispiel:

```
List<Point> l = new LinkedList<Point>();  
l.add(new Point(3.0, 5.0));  
l.add(new Point(3.0, 5.0));  
System.out.println(l.size()); // 2  
System.out.println((new HashSet<Point>(l)).size()); // 1
```

# Das Interface `SortedSet<E>`

- Ein `SortedSet<E>` ist ein geordnetes `Set<E>`
- Die Elemente eines `SortedSet<E>` sind aufsteigend sortiert
- Ein `SortedSet<E>` definiert, u.a., folgende zusätzliche Methoden:

```
public interface SortedSet<E> extends Set<E> {  
    SortedSet<E> subSet(E fromElement, E toElement);  
    E first();  
    E last();  
}
```

- Die Klasse `TreeSet<E>` implementiert `SortedSet<E>`
- Beispiel:

```
Set<Integer> s = new HashSet<Integer>(); // unsortiert  
s.add(42); s.add(13); s.add(-5);  
SortedSet<Integer> ss = new TreeSet<Integer>();  
ss.addAll(s);  
System.out.println(s); // Ausgabe: [42, -5, 13]  
System.out.println(ss); // Ausgabe: [-5, 13, 42]
```



# Das Interface `List<E>`

- Eine `List<E>` ist eine geordnete `Collection<E>`
- Eine `List<E>` kann Duplikate enthalten
- Zugriff auf einzelne Elemente mittels index-Position
- `List<E>` definiert, unter anderem, folgende zusätzliche Methoden:

```
public interface List<E> extends Collection<E> {  
    E get(int index);  
    E set(int index, E element); // optional  
    boolean add(E element);      // optional  
    void add(int index, E element); // optional  
    E remove(int index);         // optional  
    boolean addAll(int index, Collection<? extends E> c);  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
}
```

# Das Interface `List<E>`

Die Java-API stellt unter anderem die `List<E>`-Implementierungen `ArrayList` und `LinkedList` zur Verfügung

## `ArrayList<E>`


Stärken: `get`, `add`  $\in O(1)$

Schwächen: `add` nur amortisiert!

## `LinkedList<E>`

Stärken: `add`  $\in O(1)$

Schwächen: `get`  $\in O(n)$



`LinkedList` nur  
dann, wenn `add` sehr  
häufig genutzt wird

# Die Klasse Collections: Algorithmen

- Die Klasse Collections enthält (statische) Methode zum Umgang mit Collections:
- `public static boolean disjoint(Collection<?> c1, Collection<?> c2)`  
true, falls kein Element sowohl in c1 als auch in c2 enthalten ist
- `public static int frequency(Collection<?> c, Object o)`  
Anzahl der Elemente in c , die , gemäß equals , identisch zu o sind
- `public static void reverse(List<?> l)`  
Kehrt die Reihenfolge der Elemente in l um
- `public static <T> boolean replaceAll(List<T> l, T oldV, T newV)`  
Ersetzt jedes zu oldV identische Element in l mit newV
- `public static <T extends Comparable<? super T>> void sort(List<T> l)`  
Sortiert l gemäß der Methode compareTo des Typs T
- `public static <T> void sort(List<T> l, Comparator<? super T> c)`  
Sortiert l gemäß dem Comparator c

# Die Klasse Collections – Aufgabe

```
List<Integer> l = new ArrayList<Integer>();  
l.add(18); l.add(46); l.add(18); l.add(12);  
Set<Integer> s = new HashSet<Integer>();  
s.addAll(l);
```

```
System.out.println(l); // [18, 46, 18, 12]
```

```
Collections.reverse(l);
```

```
System.out.println(l); // [12, 18, 46, 18]
```

```
Collections.sort(l);
```

```
System.out.println(l); // [12, 18, 18, 46]
```

```
System.out.println(s); // [12, 18, 46]
```

```
System.out.println(Collections.disjoint(l, s)); // false
```

```
System.out.println(Collections.frequency(l, 18)); // 2
```

```
System.out.println(Collections.frequency(s, 18)); // 1
```

```
Collections.replaceAll(l, 18, 22);
```

```
System.out.println(l); // [12, 22, 22, 46]
```

# Das Interface Map<K, V>

Eine Map<K,V> bildet Schlüssel vom Typ K auf Werte vom Typ V ab

Eine Map<K,V> modelliert daher eine mathematische Funktion

Eine Map<K,V> enthält keinen Schlüssel mehrmals

Jeder Schlüssel bildet auf höchstens einen Wert ab

```
public interface Map<K,V> {  
    V put(K key, V value);  
    V get(Object key);  
    V remove(Object key);  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
    void putAll(Map<? extends K, ? extends V> m);  
    void clear();  
    public Set<K> keySet();  
    public Collection<V> values();  
}
```

# Das Interface Map<K, V>

- Die Java-API stellt unter anderem die Map<K,V>-Implementierungen **HashMap**, **TreeMap** und **LinkedHashMap** zur Verfügung
- **HashMap<K,V>**
  - Gibt keine Garantie auf die Reihenfolge der Elemente
  - $\text{get}, \text{put} \in O(1)$
- **TreeMap<K,V>**
  - Sortiert die Elemente nach der Ordnung ihrer Schlüssel
  - $\text{containsKey}, \text{get}, \text{put}, \text{remove} \in O(\log(n))$
- **LinkedHashMap<K,V>**
  - Ordnet die Elemente nach der Reihenfolge beim Einfügen
  - $\text{add}, \text{remove} \in O(1)$

# Das Interface Map<K,V> – Aufgabe

```
Map<String,Integer> berge = new TreeMap<String,Integer>();
berge.put("Mount Everest", 8848);
berge.put("K2", 8611);
berge.put("Kangchendzoenga", 8586);
berge.put("Lhotse", 8516);

System.out.println(berge.containsKey("Lhotse"));           // Ausgabe: true
System.out.println(berge.get("K2"));                       // Ausgabe: 8611
System.out.println(berge.get("Zugspitze"));                // Ausgabe: null
System.out.println(berge.keySet());                        // Ausgabe: [K2, Kangchendzoenga, Lhotse, Mount Everest]

berge.remove("K2");

System.out.println(berge.size());                           // Ausgabe: 3
System.out.println(berge.values());                         // Ausgabe: [8586, 8516, 8848]
```

# Die Klasse Math

- Die Klasse Math stellt Methoden und Konstanten zur Verfügung, die grundlegende mathematische Funktionen implementieren, u.a.:

- `static double E;` // Eulersche Zahl e, 2.71828...
- `static double Pi;` // Kreiszahl Pi, 3.14159...
- `static double abs(double a);` // Absolutbetrag von a
- `static double floor(double a);` // Abrunden von a
- `static double ceil(double a);` // Aufrunden von a
- `static double sin(double a);` // Sinus des Winkels a
- `static double cos(double a);` // Kosinus des Winkels a
- `static double exp(double a);` //  $e^a$
- `static double log(double a);` // Log von a zur Basis e
- `static double log10(double a);` // Log von a zur Basis 10
- `static double max(double a, double b);` // Maximum von a und b
- `static double min(long a, long b);` // Minimum von a und b
- `static double pow(double a, double b);` //  $a^b$
- `static long round(double a);` // Runden von a
- `static double sqrt(double a);` // Quadratwurzel von a



# Die Klasse String

Die Klasse String stellt Basisfunktionalitäten für Zeichenketten zur Verfügung. Wichtige, bisher nicht betrachtete, Funktionalitäten sind:

`public int compareTo(String s)`

Vergleicht die Strings `this` und `s`

```
"Adam".compareTo("Adam"); // Ergebnis == 0
```

```
"Adam".compareTo("Eva"); // Ergebnis < 0
```

```
"Eva".compareTo("Adam"); // Ergebnis > 0
```

```
"Adam".compareTo("adam"); // Ergebnis < 0
```

```
"Eva".compareTo("adam"); // Ergebnis < 0
```

`public int compareToIgnoreCase(String s)`

Vergleicht die Strings `this` und `s` lexikographisch

```
"Adam".compareToIgnoreCase("Adam"); // Ergebnis == 0
```

```
"Adam".compareToIgnoreCase("Eva"); // Ergebnis < 0
```

```
"Adam".compareToIgnoreCase("adam"); // Ergebnis < 0
```

# Die Klasse String

- `public int indexOf(char c)`  
`public int indexOf(String s)`  
Liefert den Index, an dem c bzw. s zum ersten Mal auftritt
- `public String substring(int beginIdx, int endIdx)`  
`"smiles".substring(1, 5); // Ergebnis : "mile"`
- `public String trim()`  
Schneidet Whitespaces am Anfang und Ende des Strings ab
- `public String toLowerCase()`  
`public String toUpperCase()`  
Liefert den String, konvertiert in Klein- bzw. Großbuchstaben
- `public boolean startsWith(String s)`  
`public boolean endsWith(String s)`  
Liefert true, falls this mit s beginnt bzw. endet

# Feedback



<https://b.socrative.com/login/student/>

Raum-Name: **PROGGENTUT**

# Abschlussaufgaben

- Eine Anmeldung ist nur nach bestandenem Übungsschein möglich
  - Der Übungsschein wird voraussichtlich bis zum **11.02.2020** verbucht sein
- Anmeldezeitraum für die Abschlussaufgaben im Studierendenportal zwischen **11.02.2020** und **20.02.2020** jeweils **12:00 Uhr**
- Abschlussaufgabe 1: **10.02.2020 – 10.03.2020**
- Abschlussaufgabe 2: **24.02.2020 – 24.03.2020**
- 100% der Abschlussnote des gesamten Moduls
  - Funktionalität: Korrektheit, Stabilität, ...
  - Methodik: saubere Modellierung, lesbarer Code, ...

# Frühere Abschlussaufgaben

## SS 15

Scrabble mit arithmetischen Ausdrücken

Graphen-basiertes Fluginformationssystem

## WS 15/16

Navigationssystem mit Berechnung des kürzesten Pfades

Brettspiel mit einem unendlichen Torusspielfeld

## SS 16

Mensch ärgere Dich nicht mit optionalen Regeln

Studierendenportal zur Verwaltung und Berechnung von Noten

# Frühere Abschlussaufgaben

- WS 16/17
  - Literaturverzeichnis: Verwaltung von Schriftstücken
  - 3x3 Candy-Crush
  
- SS 17
  - Brettspiel: Ablaufen eines Feldes
  - Zustandstracker mit Historie und einfacher Auswertung
  
- WS 17/18
  - Brettspiel ähnlich „4 Gewinnt“ mit Torusspielfeld
  - Verwaltungssystem für Olympische Spiele

# Tipps für die Abschlussaufgaben

## Herangehensweise

- Genau lesen

- Bei Unklarheiten im Forum fragen

- Frühzeitig anfangen

- Am Ende viel Testen!

## Ausgabe

- Genau die erwartete Ausgabe liefern

- An `toString()` denken

- Trennung von Logik und Benutzungsschnittstelle

## Quelltext

- `equals()`, `compareTo()` nutzen wenn sinnvoll

- Exceptions einbauen, sollte der Nutzer aber nie sehen!

- Gute JavaDoc!

# Verschiedenes

## ■ Zum Nachlesen

- <http://openbook.rheinwerk-verlag.de/javainsel/>
- <http://overapi.com/java>

## ■ Zum Üben

- <https://projecteuler.net/>
- <https://www.w3resource.com/java-exercises/>
- <https://code-exercises.com/>
- <https://codingbat.com/java>
- Eigene kleine Projekte, z.B.:
  - Taschenrechner (Eingeben von Ausdruck, Programm berechnet Ergebnis)
  - Adressbuch o.ä.

## ■ Weiterführende Vorlesungen

- Softwaretechnik (SWT)
- Betriebssysteme
- Programmierparadigmen

... und viele mehr



# Fragen?

# Auf ein gutes Gelingen!