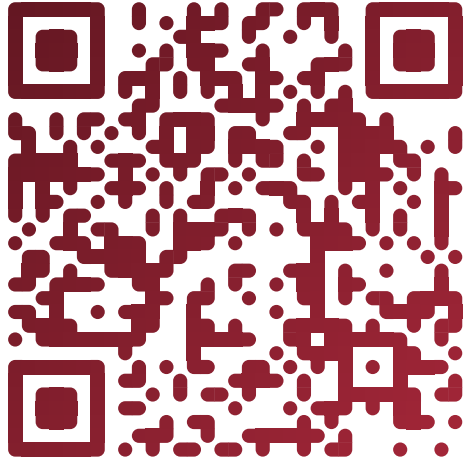


Zulassungsklausur!



Collections

Programmers like Hoarding

6. Großübung

Ulm University | Florian Sihler, Raphael Straub und Matthias Tichy | 28. Mai 2024



Software Engineering
Programming Languages



universität
uulm



Halloli!

Gibt es vorab Wünsche?

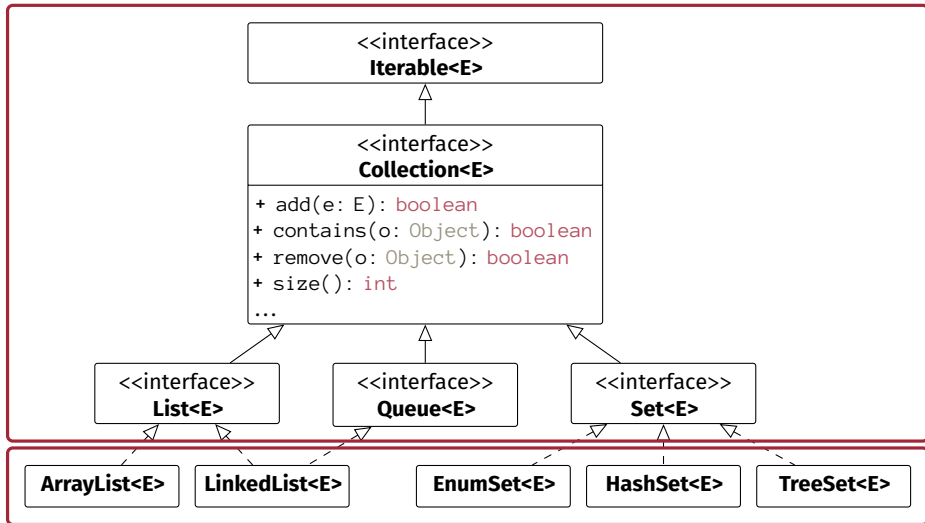
Themenübersicht

1. Einführung

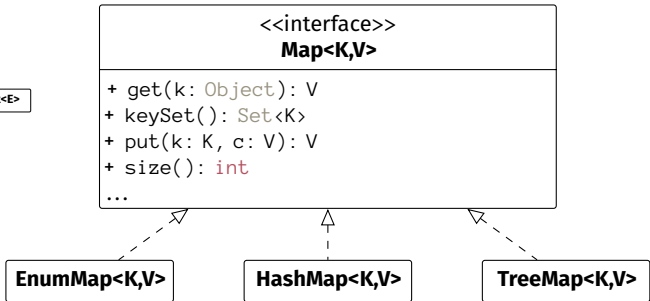
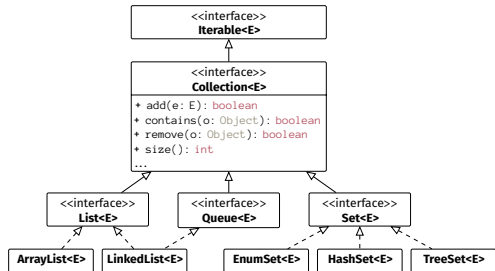
2. Javadoc Lesen

3. Hilfs-Klassen

4. Aufgaben



Implementierungen



Themenübersicht

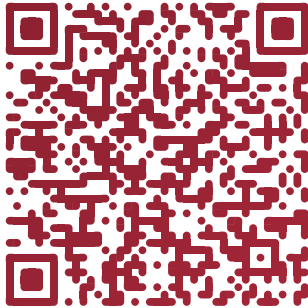
1. Einführung

2. Javadoc Lesen


3. Hilfs-Klassen

4. Aufgaben

Javadoc Lesen



Ein bisschen Friede, ein bisschen Doku...

 Javadoc-Kommentar

```
/**
 * Diese fantastische Klasse wird eines Tages die Welt beherrschen.
 */
public class SampleObject {
    /** Wie süß ist dieses Objekt bitte? */
    public final int cuteness;

    /**
     * @param cuteness Wie süß das Objekt sein soll. Muss mindestens 100 sein.
     * @throws IllegalArgumentException Wenn das Objekt nicht süß genug ist.
     */
    public SampleObject(int cuteness) {
        if (cuteness < 100) {
            throw new IllegalArgumentException("Das Objekt muss mindestens 100 süß sein!");
        }
        this.cuteness = cuteness;
    }
    // ...
}
```

Javadoc-Kommentare

- Dokumentation lesen zu können, ist eine wichtige Fähigkeit
Insbesondere auch, auswählen, was wichtig ist
- An der Zulassungs-Klausur steht das gesamte Javadoc von Java 21 zur Verfügung
- An der Klausur erhaltet ihr geeignete Ausschnitte

Themenübersicht

1. Einführung

2. Javadoc Lesen

3. Hilfs-Klassen

4. Aufgaben

Hilfs-Klassen

Objects

Arrays

Collections

- Bieten viele nützliche Methoden zum ...
 - Vergleichen
 - Kopieren
 - Sortieren
 - Durchsuchen
 - Erzeugen
- Versucht ruhig, sie in den nächsten Aufgaben zu verwenden

Themenübersicht

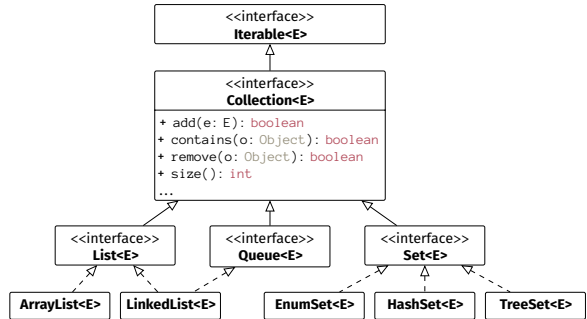
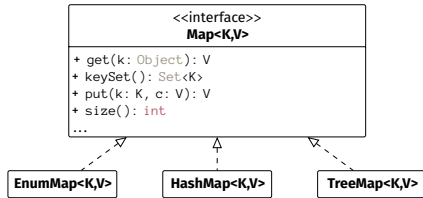
1. Einführung

2. Javadoc Lesen

3. Hilfs-Klassen

4. Aufgaben

Den Durchschnitt berechnen



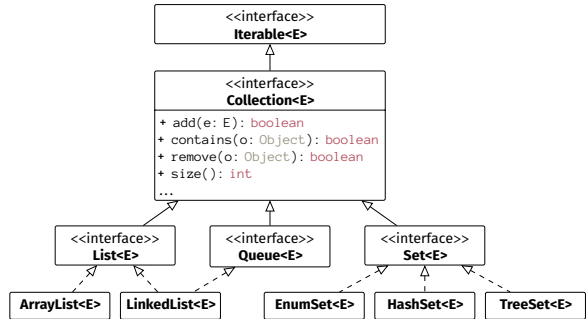
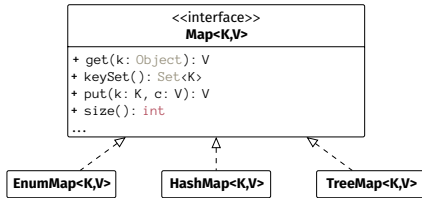
Implementieren Sie eine Funktion `average(Collection<Integer> numbers)`, die den Durchschnitt aller Zahlen in der Collection berechnet. Wenn die Collection leer ist, soll die Funktion ein leeres Optional zurückgeben.

Lösungsvorschlag

```
public static Optional<Double> average(Collection<Integer> numbers) {  
    if (numbers.isEmpty()) {  
        return Optional.empty();  
    }  
    double sum = 0;  
    for (int number : numbers) {  
        sum += number;  
    }  
    return Optional.of(sum / numbers.size());  
}
```

```
public static OptionalDouble averageStream(Collection<Integer> numbers) {  
    return numbers.stream()  
        .mapToDouble(Integer::doubleValue)  
        .average();  
}
```


Die Differenz zweier Mengen

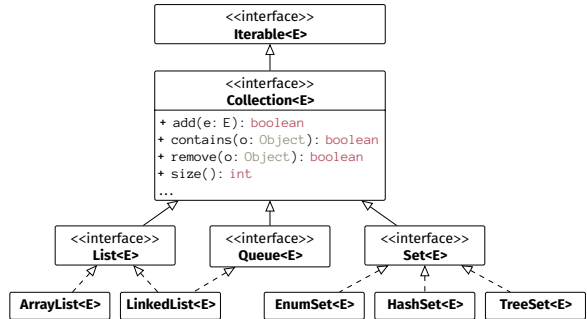
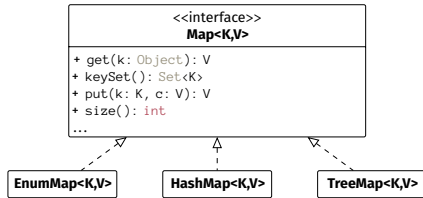


Implementieren Sie **public static** `<T> Set<T> difference(Set<T> set1, Set<T> set2)`. Die Funktion soll nur die Elemente von `set1` zurückgeben, die nicht in `set2` enthalten sind. Die Funktion soll die übergebenen Mengen dabei nicht verändern.

Lösungsvorschlag

```
public static <T> Set<T> difference(Set<T> set1, Set<T> set2) {  
    Set<T> result = new HashSet<>(set1);  
    result.removeAll(set2);  
    return result;  
}
```

Der Herr Graph!



- Erstellen Sie eine Klasse *DirectedGraph<E>*, die einen gerichteten, ungewichteten Graphen. Die einzelnen Knoten sollen dabei durch (eindeutige) *Strings* benannt werden und einen Wert vom generischen Typ *E* speichern können. Jeder Knoten kann beliebig viele Nachbarn besitzen.
- Implementieren Sie die Methode **public void** *addNode(String name, E value)*, die einen neuen Knoten hinzufügt.
- Implementieren Sie die Methode **public void** *addEdge(String from, String to)*, die eine gerichtete Kante von einem Knoten *from* zu einem Knoten *to* hinzufügt.
- Implementieren Sie die Methode **public Set<String>** *nodes()*, die die Namen aller Knoten des Graphen zurückgibt.

Lösungsvorschlag

```
public class DirectedGraph<E> {  
    private Map<String, E> nodeValues = new HashMap<>();  
    private Map<String, List<String>> edges = new HashMap<>();  
  
    public void addNode(String node, E value) {  
        nodeValues.put(node, value);  
    }  
  
    public void addEdge(String from, String to) {  
        edges.computeIfAbsent(from, k -> new ArrayList<>()).add(to);  
    }  
  
    public Set<String> getNodes() {  
        return nodeValues.keySet();  
    }  
}
```