# Hinweise zur Probeklausur

- Wir empfehlen, die Probeklausur unter so realistischen Bedingungen wie möglich zu schreiben. Das heißt, Sie sollten sich 120 Minuten möglichst **ungestört** mit der **ausgedruckten** Probeklausur beschäftigen und nur die für die echte Klausur erlaubten Hilfsmittel verwenden.
- Die Punkte pro Aufgabe entsprechen etwa dem Zeitaufwand in Minuten. Falls Sie bei einer Teilaufgabe nicht weiterkommen, überspringen Sie erstmal diese Teilaufgabe; ansonsten verlieren Sie in der echten Klausur unnötig Zeit.
- Sie dürfen bei jeder Aufgabe auf andere Methoden in der Aufgabe zurückgreifen, die sie selbst in der Aufgabe implementieren müssen oder die von uns vorgegeben sind. Ersteres ist auch dann erlaubt, falls sie diese Teilaufgabe nicht bearbeitet haben.
- Deckblatt, Hinweise usw. entsprechen voraussichtlich schon denen der echten Klausur. Sie können sich also bereits jetzt damit beschäftigen und alle Fragen dazu klären, dann müssen Sie das Deckblatt in der tatsächlichen Klausur nur noch überfliegen.
- Bei der echten Klausur bekommen Sie vorab einen Platz zugewiesen und auf diesem Platz liegt bereits eine Klausur, auf die Ihr Name gedruckt ist. Außerdem ist auf dem Deckblatt ein QR-Code, in dem Ihre Matrikelnummer steht. Wir nutzen den Code, um die Eintragung der Punkte in unsere Ergebnistabelle teilweise zu automatisieren.
- Die Klausur wird doppelseitig gedruckt und oben links getackert; der Bereich der Tackerung ist mit einem Dreieck markiert und darf nicht beschrieben werden. Zusammenhängende Aufgaben werden nach Möglichkeit (wie auch in der Probeklausur) beim Blättern nebeneinander zu liegen kommen.
    - Es kann sein, dass aus Layout-Gründen eine Seite in der Klausur leer ist. Diese sind dann mit einem Hinweis versehen, dass die leere Seite Absicht ist.
    - Insbesondere werden die letzten beiden Seiten (doppelseitig) auf ein Blatt gedruckt. In dieser Probeklausur und in den echten Klausuren enthält dieses letzte Blatt Dokumentation von Klassen aus dem JDK. Dieses letzte Blatt können Sie zum einfacheren Bearbeiten der zugehörigen Aufgaben heraustrennen. Notizen, die Sie auf dem Dokumentations-Anhang machen, werden **nicht** korrigiert.
- Die angehängte Dokumentation wird in den echten Klausuren sehr wahrscheinlich identisch zu der Dokumentation in dieser Probeklausur sein. Es ist also sinnvoll, wenn Sie sich jetzt schon mit der Dokumentation vertraut machen und Fragen dazu im Forum oder Tutorium stellen.
- Die Probeklausur wird in der letzten Vorlesungswoche in den Tutorien besprochen (nach der Besprechung des letzten Übungsblatts, sodass Personen, die die Probeklausur noch nicht gerechnet haben, gehen können, bevor sie gespoilert werden). Außerdem gibt es voraussichtlich ein Lösungs-Video (wird an der Stelle verlinkt, wo auch die Lösungen zu den Übungsaufgaben sind).
- In der Probeklausur können teilweise Abwandlungen alter Klausur- und Übungsaufgaben vorkommen. Dies kann in einer echten Klausur ebenfalls der Fall sein, muss es aber nicht.
- Bei jeder Klausur muss eine Auswahl aus den behandelten Themen getroffen werden. Die Auswahl der Themen und die Verteilung der Punkte auf die einzelnen Themen kann in der echten Klausur anders aussehen. Insbesondere können in den echten Klausuren auch Aufgaben zu den Lernzielen der letzten Vorlesungen vorkommen.

- Folgende Themen kommen beispielsweise in dieser Probeklausur nicht vor, sind aber klausurrelevant (unvollständige Aufzählung): Überladen, mehrdimensionale Arrays, abstrakte Klassen, super, verkettete Listen, Fangen von Exceptions, Zweck von Buildtools, Versionsverwaltungssystemen und IDEs
- Welche Themen klausurrelevant sind, wird in der Vorlesung kommuniziert. Insbesondere kennen die Hilfskräfte (auch diejenigen, die das Tutorium zur Vorbereitung auf die zweite Klausur halten) die Klausuren im Vorfeld nicht und können daher keine verbindlichen Aussagen zur Klausurrelevanz treffen.

- Auch in den echten Klausuren wird die letzte Aufgabe aus dem Themenbereich der objektorientierten Programmierung stammen und viele Punkte geben.

- Sie können die Probeklausur nutzen, um herauszufinden, mit welchen Aufgabentypen Sie besonders gut zurechtkommen; diese Aufgaben wären ein guter Startpunkt in der echten Klausur. Wenn Sie mit bestimmten Aufgabentypen Schwierigkeiten haben, versuchen Sie diese zu identifizieren und Unklarheiten zu beseitigen; überspringen Sie erstmal Aufgaben, bei denen Sie hängen.

- Wir bemühen uns, dass die Probeklausur eher **schwieriger** und zeitlich **knapper** kalkuliert ist als die echten Klausuren, und dass beide echten Klausuren gleich schwierig sind. Da der Schwierigkeitsgrad aber keine objektive Einschätzung ist, sondern von den eigenen Stärken und Schwächen abhängt, können wir das nicht garantieren. Die Klausuren orientieren sich an den im Semester bearbeiteten Materialien und den Lernzielen der Vorlesung. Die echten Klausuren sind keine Probeklausur „mit anderen Zahlen“.
    - Der Schwierigkeitsgrad der Zulassungstests ist **nicht** repräsentativ für die Klausur. Orientieren Sie sich für Schwierigkeit und Zeit eher an den jüngeren Altklausuren.

- Zum weiteren Üben eigenen sich insbesondere die jüngeren Altklausuren, die Sie im Klausurarchiv der Fachschaft Informatik[1] finden. Beachten Sie, dass es in diesem Semester Vorlesungsthemen gab, die in den letzten Jahren nicht behandelt wurden (und andersherum).

[1] https://fscs.hhu.de/de/klausurarchiv/

HHU Düsseldorf
Institut für Informatik
Dr. Markus Brenneis

*Daten des Prüflings*

Matrikelnummer: _____

Nachname: _____

Vorname: _____

# Probeklausur

**Programmierung WS 25/26**
**Januar 2026**

- Prüfen Sie nach Beginn der Schreibzeit, ob alle Seiten enthalten sind.
- Sie erhalten von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie unten auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben.
- Fachbegriffe werden wie in der Vorlesung definiert verwendet. Die Aufgaben beziehen sich auf die in der Vorlesung vorgestellte Java-Version 25. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Erlaubte Hilfsmittel: eine beidseitig beschriebene oder bedruckte A4-Seite, Wörterbuch (Wörterbuch muss vor Beginn der Aufsicht zur Kontrolle vorgelegt werden.)
- Schalten Sie technische Geräte aus. Täuschungsversuche führen zum sofortigen Ausschluss und dem Ergebnis „nicht bestanden“.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

Mit meiner Unterschrift bestätige ich, dass alle Seiten vorhanden sind und ich die Hinweise gelesen habe: _____

zur Bewertung abgegebene, lose Blätter: _____ **Viel Erfolg!**

---

Diesen Teil bitte nicht ausfüllen:

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| Punkte | 5 | 6 | 18 | 8 | 5 | 5 | 15 | 2 | 26 | 90 |
| Erreicht | | | | | | | | | | |

## Aufgabe 1

_____ _____ / 5 Punkte

__/5 (a) Gegeben seien die folgenden java-Dateien:

IntFunction.java

```java
interface IntFunction<T> {
    T calc(int x);
}
```

Square.java

```java
class Square implements IntFunction<Integer> {
    Integer calc(int x) {
        return x * x;
    }
}
```

Test.java

```java
static void main(String[] args) {
    if(args.length == 0) {
        System.out.println("arg missing");
        break;
    }
    x = Integer.parseInt(args[0]);
    IntFunction<Integer> sq = new Square();
    System.out.println(sq.calc(x));
}
```

Beim Compilieren der Dateien gibt es folgende Fehlermeldungen:

```
Test.java:4: error: break outside switch or loop
        break;
        ^
Test.java:6: error: cannot find symbol
    x = Integer.parseInt(args[0]);
    ^
  symbol:   variable x
  location: class Test
Test.java:8: error: cannot find symbol
    System.out.println(sq.calc(x));
                               ^
  symbol:   variable x
  location: class Test
Square.java:2: error: calc(int) in Square cannot implement calc(int) in IntFunction
    Integer calc(int x) {
            ^
  attempting to assign weaker access privileges; was public
  where T is a type-variable:
    T extends Object declared in interface IntFunction
4 errors
error: compilation failed
```

Geben Sie an, in welchen Dateien und Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehlerursachen jeweils kurz (max. 1 Satz) und geben Sie die korrigierte Codezeile **vollständig** an, sodass der Code das tut, was bei der Programmierung wahrscheinlich vorgesehen war; falls eine Zeile entfernt werden muss, tragen Sie `-leer-` als Korrektur ein. Geben Sie keine Folgefehler an, die durch Korrektur eines anderen Fehlers behoben werden.

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Datei, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

## Aufgabe 2

_____ _____ / 6 Punkte

Methoden von `java.util.List<E>` sind zur Erinnerung im Anhang dokumentiert.

(a) Geben Sie für die beiden folgenden Codeausschnitte jeweils an, ob die idiomatische Iterations-Art verwendet wird. Falls der Code nicht idiomatisch ist, geben Sie idiomatischen Code mit gleicher Semantik an.

Vorgabe — Java

```
import java.util.List;

static void printEveryOtherWord(List<String> words) {
    int i = 0;
    for(String word: words) {
        if(i % 2 == 0) {
            System.out.println(word);
        }
        i++;
    }
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
import java.util.List;

static void printEveryOtherWord(List<String> words) {









}
```

Vorgabe — Java

```
import java.util.List;

static void printShortWords(List<String> words) {
    words.stream()
        .filter(word -> word.length() < 5)
        .forEach(System.out::println);
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
import java.util.List;

static void printShortWords(List<String> words) {









}
```

(b) Geben Sie an, was das folgende Programm ausgibt, wenn es ausgeführt wird.

Foreach.java — Java

```
import java.util.List;
import java.util.LinkedList;

static void main() {
    List<String> woerter = new LinkedList<>();
    woerter.add("a");
    woerter.add("be");
    woerter.add("ce");
    woerter.forEach(w -> System.out.print(w + ","));
}
```

---

(c) Die $n$-te sog. Partialsumme der alternierenden harmonischen Reihe ist definiert als:

$$f(n) = \underbrace{1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} \ldots}_{n \text{ Terme (bis einschl. } 1/n)} = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$$

*(Ignorieren Sie den letzten Teil mit dem Σ, wenn Sie das Zeichen nicht kennen.)*
Die folgende Methode `alternierendHarmonisch` soll $f(n)$ in Java umsetzen. Sie funktioniert allerdings nicht korrekt:

AlternierendHarmonisch.java — Java

```
1  private static double alternierendHarmonisch(int n) {
2      double ergebnis = 0;
3      for(int k = 1; k < n; k++) {
4          ergebnis += Math.pow(-1, k+1) / k;
5      }
6      return ergebnis;
7  }
8
9  static void main() {
10     System.out.println(alternierendHarmonisch(1)); // erwartet: 1
11     System.out.println(alternierendHarmonisch(2)); // erwartet: 0.5
12     System.out.println(alternierendHarmonisch(3)); // erwartet: 0.8333
13     System.out.println(alternierendHarmonisch(4)); // erwartet: 0.5833
14 }
```

```
% java AlternierendHarmonisch.java
0.0
1.0
0.5
0.8333333333333333
```

Erklären Sie in 1–3 ausformulieren Sätzen, warum die Methode nicht wie gewünscht funktioniert und wie sie repariert werden kann. Zur Verdeutlichung können Sie auch in den vorgegebenen Code schreiben.

## Aufgabe 3

_____ _____ / 18 Punkte

Zum Lernen des Alphabets haben wir uns folgendes Spiel ausgedacht: Personen sagen der Reihen nach Wörter. Der Anfangsbuchstabe des nächsten Worts muss immer weiter hinten im Alphabet liegen als der Anfangsbuchstabe des vorhergehenden Wortes. Ä, Ö und Ü zählen als A, O bzw. U. Beispiele:

- gültig: Banane, Müsli, Öl, Soja
- ungültig: Chamäleon, Kuh, Zug, Affe (A nicht weiter hinten als Z)
- ungültig: Eisbär, Ente (E nicht weiter hinten als E)

*Hinweise:*

- `'A' < 'Z' == true`
- `(int)'A' == 65`, `(int)'Ä' == 169`, `(int)'Ö' == 214`, `(int)'Ü' == 220`
- Methoden der String-Klasse finden Sie in der angehängten Dokumentation.

Vervollständigen Sie die Klasse `Game` durch Hinzufügen der folgenden Methoden:

___/5½ (a) Eine private Klassenmethode `boolean validWord(String)`, die genau dann `true` zurückgibt, wenn das Wort mindestens 1 Zeichen lang ist und mit einem Großuchstaben von A bis Z, Ä, Ö oder Ü beginnt. Sie dürfen davon ausgehen, dass der Parameter nicht `null` ist.

___/6 (b) Eine private Klassenmethode `boolean isBefore(String, String)`, die genau dann `true` zurückgibt, wenn der zweite übergebene String auf den ersten übergegebenen String folgen darf (gemäß der Spielregeln). Sie dürfen hier davon ausgehen, dass beide Strings mit einem Großuchstaben von A bis Z, Ä, Ö oder Ü beginnen und aus jeweils mindestens einem Zeichen bestehen.

___/6½ (c) Eine paket-private main-Methode, die beliebig viele Wörter als Argumente erwartet. Wenn die Wortfolge gemäß der Spielregeln gültig ist, soll `gültig` auf der Standardausgabe ausgegeben werden, ansonsten `ungültig`.[1]

**Beispiele:**

```
% java Game aa bb
ungültig
% java Game Aa Cc
gültig
% java Game Aa Äa
ungültig
```

Game.java Java

```
class Game {
```

[1] Da alle Wörter in einer leeren Wortfolge die Spielregeln einhalten, ist eine leere Wortfolge auch gültig.

```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Nr. 998 (Z98)

Game.java (Fortsetzung) Java

```
}
```

## Aufgabe 4

_____ _____ / 8 Punkte

Gegeben ist eine `HashSet`-Implementierung, wie Sie sie aus der Vorlesung kennen:

HashSet.java — Java

```java
public class HashSet {

    private final static int SIZE = 50;
    // analog zur Vorlesung: genau die nicht besetzten Plätze sind null
    private Studi[] studis = new Studi[SIZE];

    public void insert(Studi studi) {
        // Implementierung nicht abgedruckt

    }

    @Override
    public String toString() {
        // Implementierung nicht abgedruckt

    }
```

Studi.java — Java

```java
public record Studi(String name, Integer nr) {}
```

Im JDK ist im Package `java.util.function` das funktionale Interface `Predicate<T>` enthalten, das wie folgt definiert ist:

Predicate.java — Java

```java
public interface Predicate<T> {
    boolean test(T t);
}
```

Wir wollen die Klasse `HashSet` um eine öffentliche Methode `countIf` erweitern. Der Methode wird ein `Predicate<String>` übergeben und sie gibt zurück, für wie viele Studi-Namen im HashSet das übergebene Prädikat `true` ist.

Beispiel — Java

```java
HashSet studis = new HashSet();
studis.insert(new Studi("Alex", 3473847));
studis.insert(new Studi("Marlin", 3182782));
studis.insert(new Studi("Robin", 3583729));
Predicate<String> p = n -> n.endsWith("n");
System.out.println(studis.countIf(p)); // 2
System.out.println(studis.countIf(n -> n.charAt(0) == 'A')); // 1
System.out.println(studis.countIf(n -> n.length() > 4)); // Aufgabenteil a)
```

___/1 (a) Welchen **Datentyp** hat der Ausdruck `n.charAt(0) == 'A'` (vorletzte Zeile) im obigen Beispiel?

---

Welcher **Wert** wird vom letzten `println`-Statement im Beispielcode ausgegeben?

---

___/1 (b) Ergänzen Sie die Codevorgabe `HashSet.java` oben, sodass das Interface `java.util.Predicate` unter dem Namen `Predicate` verwendet werden kann.

___/6 (c) Implementieren Sie die öffentliche Methode `countIf`, sodass sie wie oben im Beispiel angegeben verwendet werden kann.

```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

HashSet.java Java

```
}
```

## Aufgabe 5

_____ _____ / 5 Punkte

Der Konstruktor der Klasse `Search` speichert Kopien der übergebenen Elemente in einer neuen ArrayList (Shallow Copy) und sortiert sie. Die Instanzmethode `contains` kann benutzt werden, um mithilfe von **binärer Suche** Elemente in der gespeicherten Liste zu suchen.

Search.java — Java

```
import java.util.ArrayList;
import java.util.Collections;

final public class Search<T extends Comparable<T>> {
  private final ArrayList<T> items;

  public Search(ArrayList<T> items) {
    this.items = new ArrayList<T>(items);
    Collections.sort(this.items);
  }

  public boolean contains(T needle) {
    return Collections.binarySearch(this.items, needle) >= 0;
  }
}
```

Artikel.java — Java

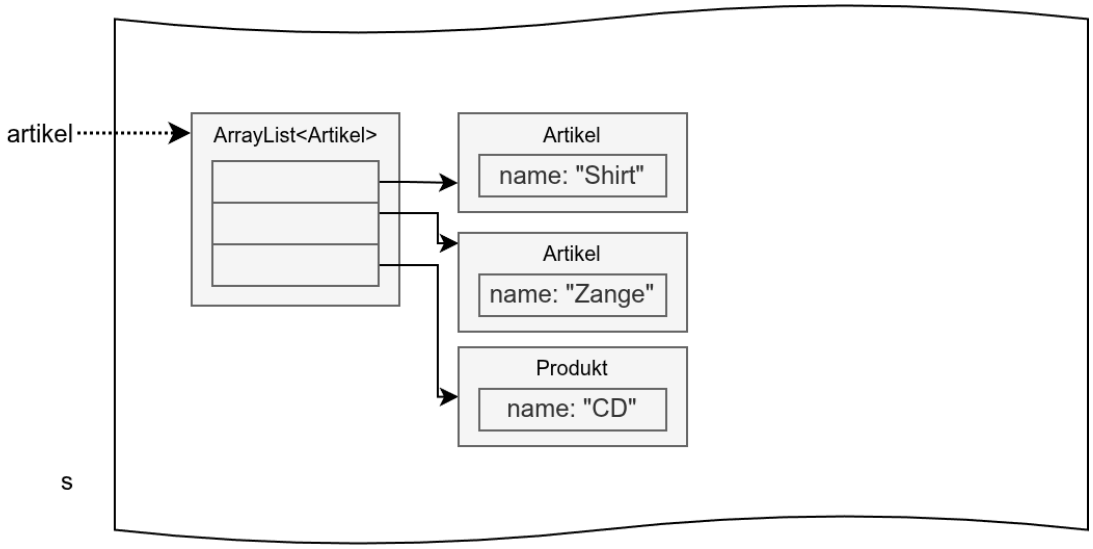
```
class Artikel implements Comparable<Artikel> {
  private String name;

  Artikel(String name) {
    this.name = name;
  }

  String getName() {
    return name;
  }

  void setName(String name) {
    this.name = name;
  }

  @Override
  public boolean equals(Object other) {
    // zwei Artikel sind gleich, wenn ihre Namen gleich sind
    if(!(other instanceof Artikel otherArtikel)) {
      return false;
    }
    return this.name.equals(otherArtikel.name);
  }

  @Override
  public int compareTo(Artikel other) {
    return this.name.compareTo(other.name); // alphabetisch nach Name
  }
}
```

Test.java — Java

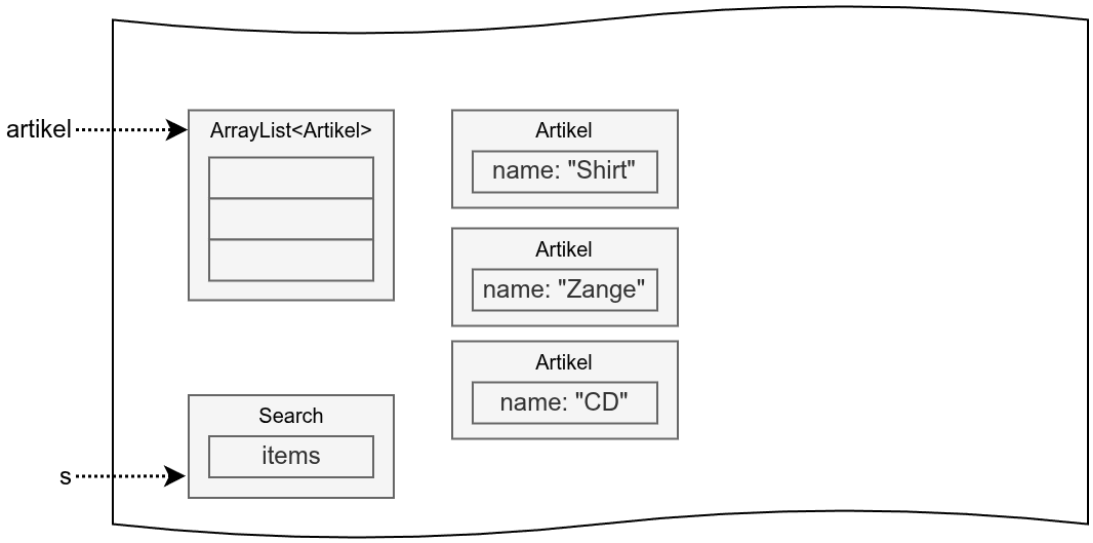
```
1  import java.util.ArrayList;
2
3  static void main() {
4    ArrayList<Artikel> artikel = new ArrayList<>();
5    artikel.add(new Artikel("Shirt"));
6    Artikel art2 = new Artikel("Zange");
7    artikel.add(art2);
8    artikel.add(new Artikel("CD"));
9
10   Search<Artikel> s = new Search<>(artikel);
11   System.out.println(s.contains(new Artikel("CD"))); // true
12   System.out.println(s.contains(new Artikel("Zange"))); // true
13   System.out.println(s.contains(art2)); // true
14   System.out.println(s.contains(new Artikel("Kabel"))); // false
14 }
```

Die Zeilennummern beziehen sich immer auf die Datei `Test.java`.

___/$2\frac{1}{2}$ (a) Nach dem Ausführen von Zeile 8 sieht der Heap (vereinfacht) so aus:




**Zeichnen** Sie in die folgende Abbildung ein, wie der Heap nach Ausführen von **Zeile 10** aussieht. Zeichnen Sie insbesondere ein, welche **Objekte** in der Liste `artikel` stehen, und das Objekt, auf das die Instanzvariable `items` des Objekts `s` **zeigt** und welche **Objekte** in dieser Liste stehen.



___/$2\frac{1}{2}$ (b) Ist es möglich – ohne Veränderung der Klassen `Search` und `Artikel` – Code in der `Test.java` zu **ergänzen**, sodass `s.contains(art2)` in Zeile 13 `false` zurückgibt, obwohl `art2` weiterhin in `s.items` enthalten ist?

- *wenn ja:* Geben Sie konkreten **Code** an. Erklären Sie, wie dieser Code den **Heap** beeinflusst und **warum** er das falsche Ergebnis liefert.
- *wenn nein:* **Begründen** Sie Ihre Antwort, indem Sie erklären, durch **welchen Code** die Klasse `Search` verhindert, dass falsche Ergebnisse geliefert werden. Sie dürfen dazu Stellen im vorgegebenen Code markieren.

```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```



## Aufgabe 6

_____ _____ / 5 Punkte

___/3 (a) Für das Organisationsteam einer Vorlesung wurde ein Java-Programm entwickelt, das anhand einer eingelesenen Punktetabelle ausgibt, welche Studis die Zulassung erreicht haben und wie viele Punkte maximal, minimal und im Median erlangt wurden. Dazu liest das Programm die Tabellenzeilen über die Standardeingabe ein und speichert die eingelesenen Zahlen zunächst in einer Datenstruktur. Wie viele Zeilen die Tabelle hat, ist vorab nicht bekannt.

Mo schlägt vor, die Tabellenzeilen in einem sortierten Array zu speichern. Was spricht **für** diese Wahl? Was spricht **dagegen**? Begründen Sie jeweils in 1–2 ausformulierten Sätzen.

___/2 (b) Wir haben die Noten einer Klausur in der Plaintext-Datei `noten` gespeichert. Außerdem haben wir ein bereits kompiliertes Java-Programm `Schnitt`, das den Durchschnitt von Zahlen berechnen und ausgeben kann.

```
% ls
noten Schnitt.class Schnitt.java
% cat noten
1.7 1.3 1.7 2.3 2.3 1 4 3 5 3.7
```

Schnitt.java — Java

```java
import java.util.Scanner;
import java.util.LinkedList;
import java.util.List;

static void main(String[] args) {
    Scanner eingabe = new Scanner(System.in);
    List<Double> zahlen = new LinkedList<>();

    // liest Zahlen ein
    String[] werte = eingabe.nextLine().split(" ");
    for(String w: werte) {
        zahlen.add(Double.parseDouble(w));
    }

    // berechnet den Durchschnitt (den Code müssen Sie nicht nochvollziehen)
    double schnitt = zahlen.stream()
        .mapToDouble(Double::doubleValue)
        .average()
        .orElse(Double.NaN);

    // gibt Durchschnitt aus
    System.out.println(schnitt);
}
```

Geben Sie einen Aufruf des Programms `Schnitt` an, der den Durchschnitt der Zahlen in der Datei `noten` berechnet und das Ergebnis in die Datei `schnitt` schreibt.

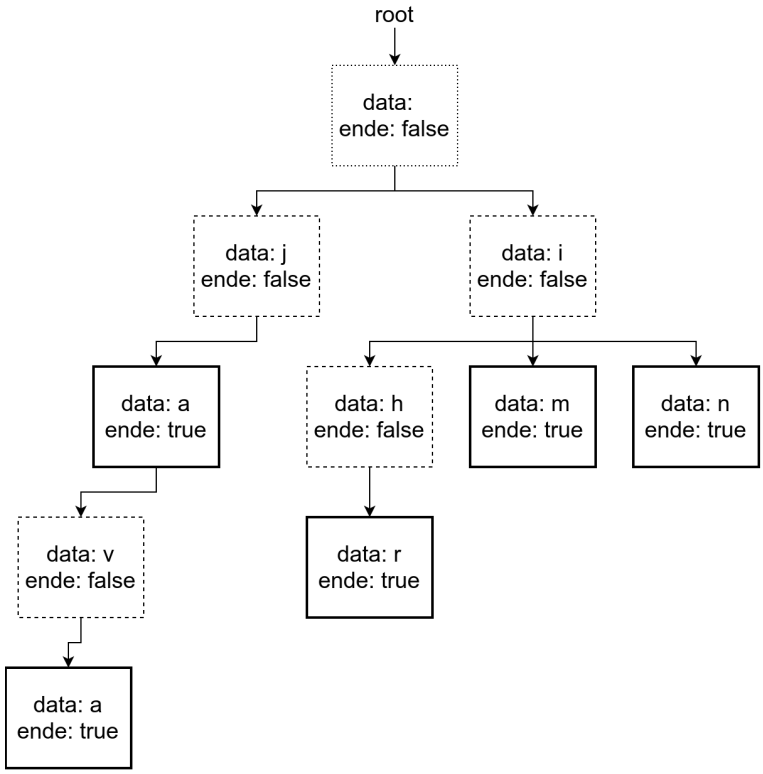
## Aufgabe 7

_____ _____ / 15 Punkte

Ein Trie ist eine baumartige Datenstruktur. Anders als bei einem binären Suchbaum kann ein Knoten (Node) allerdings eine beliebige Anzahl von Nachfolgeknoten haben.

In einen Trie wird ein Wort eingefügt, indem die Knoten – beginnend nach dem Wurzelknoten – die einzelnen Buchstaben des Wortes speichern und der letzte Knoten eine `ende`-Markierung erhält.

In folgendem Beispiel-Trie sind die fünf Wörter ja, java, ihr, in und im gespeichert:




Wir betrachten die Klasse `Trie` mit folgenden Methoden:

Trie.java Java

```
import java.util.ArrayList;

public class Trie {
    private class Node {
        private char data;
        private ArrayList<Node> next; // niemals null, _nicht_ sortiert, keine Duplikate
        private boolean ende;

        private Node(char data, ArrayList<Node> next, boolean ende) {
            this.data = data;
            this.next = next;
            this.ende = ende;
        }
    }

    private Node root = new Node(' ', new ArrayList<Node>(), false);

    // fügt wort in den Trie ein
    public void add(String wort) {
        // ... (Code nicht abgedruckt)

    }
```

Sie sollen später u. a. eine Methode `size()` programmieren, die die Anzahl der gespeicherten Wörter zurückgibt.

Beispielverwendung der Methoden; es wird der Trie von der vorherigen Seite erzeugt:

Beispielcode — Java

```
Trie trie = new Trie();
trie.add("java");
trie.add("ja");
trie.add("im");
trie.add("in");
trie.add("ihr");

System.out.println(trie.size()); // 5
System.out.println(trie.contains("ja")); // true
System.out.println(trie.contains("jav")); // false
System.out.println(trie.contains("java")); // true
System.out.println(trie.contains("javas")); // false
System.out.println(trie.contains(" ")); // false
// trie.contains(""); // löst IllegalArgumentException aus
```

___/1 (a) Wir ergänzen den Beispielcode um diese Zeilen:

Java

```
trie.add("ih");
System.out.println(trie.size());
```

Welcher **Wert** wird vom `println`-Statement ausgegeben?

---

**Zeichnen** Sie in die Abbildung auf der vorherigen Seite ein, wie sich der Trie durch diesen Code verändert hat.

___/5½ (b) Schreiben Sie eine **öffentliche Instanzmethode** `int size()`, die zurückgibt, wie viele Wörter im Trie gespeichert sind (d. h. wie viele Knoten mit `ende == true` existieren). Wenn Sie Hilfsmethoden schreiben, müssen diese minimale Sichtbarkeit haben.

___/8½ (c) Schreiben Sie eine **öffentliche Instanzmethode** `boolean contains(String)`, die genau dann `true` zurückgibt, wenn der übergebene String im Trie gespeichert ist. Wenn der übergebene String die Länge 0 hat oder der Methoden-Parameter `null` ist, soll eine `IllegalArgumentException` geworfen werden.

Trie.java (Fortsetzung) — Java

Trie.java (Fortsetzung) Java

```
}
```

* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*



## Aufgabe 8

_____ _____ / 2 Punkte

Im Terminal ist die Ordnerstruktur im aktuellen Verzeichnis dargestellt:

```
/home/mareike/projects % tree
.
|-- src
    |-- main
        |-- java
            |-- progra
                |-- Main.java
                |-- util
                    |-- Math.java
    |-- test
        |-- java
            |-- progra
                |-- MainTest.java
```

Der Classpath ist `src/main/java:src/test/java`.

Kreuzen Sie an, welche der Klassen im Package `progra` liegen:

- ☐ `Main`
- ☐ `Math`
- ☐ `MainTest`

In Woche 5 der Vorlesung wurde gesagt, dass paket-private Klassen nur für Klassen im selben Ordner sichtbar sind. Kreuzen sie die richtige Aussage an:

- ☐ Diese Aussage ist korrekt. Der Sinn von paket-privaten Klassen ist, die Abhängigkeit zwischen Codebestandteilen und damit die Komplexität zu begrenzen.
- ☐ Die Aussage ist **nicht** richtig.

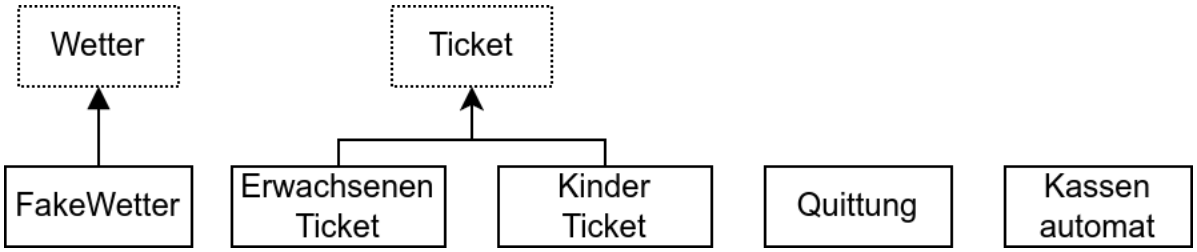
## Aufgabe 9

_____ _____ / 26 Punkte

In dieser Aufgabe sollen Sie Code für einen Kassenautomaten eines Schwimmbads programmieren.

**Hinweise:**

- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.
- Wählen Sie sinnvolle Datentypen, wo es keine genaue Vorgabe gibt.
- Sofern nicht anders durch die Aufgabenstellung oder Java vorgegeben, sollen Interfaces, Klassen, Konstruktoren und Methoden **paket-privat** sein und Klassen *nicht* abstrakt.
- Alle Instanz- und Klassenvariablen müssen **privat** sein.
- Wenn kein Konstruktorverhalten vorgeschrieben ist, reicht der Default-Konstruktor.
- Das genaue Format von Textausgaben ist Ihnen überlassen.
- Innerhalb dieser Aufgabe müssen Sie keine Exceptions abfangen. Sie müssen nur dann Parameter validieren, wenn dies verlangt ist.
- An Stellen, wo Klassen verlangt sind, dürfen Sie auch Records benutzen, wenn diese mit den Anforderungen der Aufgabenstellung kompatibel sind.
- Gehen Sie davon aus, dass alle in dieser Aufgabe erstellten und vorgegebenen Klassen und Interfaces im Default-Package liegen.




Bereits fertig sind zwei Interfaces:

Wetter.java

```java
interface Wetter {
    double temperatur();
}
```

Ticket.java

```java
interface Ticket {
    int preis();
}
```

___/6

(a) Erstellen Sie Klassen `ErwachsenenTicket` und `KinderTicket`, die das `Ticket`-Interface sinnvoll implementieren. Jedes Erwachsenen-Ticket kostet 300 Cent, jedes Kinder-Ticket 150 Cent. Überschreiben Sie außerdem die `toString`-Methode, sodass jeweils die Art des Tickets (Erwachsene/Kinder) und der Preis zurückgegeben werden.

`ErwachsenenTicket.java` Java

`KinderTicket.java` Java

___/3 (b) Schreiben Sie eine Klasse `FakeWetter`, die das `Wetter`-Interface sinnvoll implementiert. Der Konstruktor bekommt eine Temperatur übergeben. Die `temperatur`-Methode gibt immer diese Temperatur zurück.

FakeWetter.java Java

___/12 (c) Schreiben Sie eine Klasse `Quittung`.

Eine Quittung speichert eine `Wetter`-Instanz und eine `ArrayList` von Tickets. Der **Konstruktor** `Quittung(int, int, Wetter)` bekommt eine Anzahl von Erwachsenen-Tickets, eine Anzahl von Kinder-Tickets und eine `Wetter`-Instanz übergeben; er legt entsprechend viele Tickets in der `Ticket`-Liste ab.

Relevante Methoden von `ArrayList` finden Sie in der angehängten Dokumentation. Denken Sie daran, dass `ArrayList` in einem anderen Package liegt.

Schreiben Sie eine Methode `int gesamtpreis()`, die den **Gesamtpreis** aller gespeicherten Tickets zurückgibt. Es gibt einen **Rabatt** von 200 Cent, wenn der Gesamtpreis (ohne Rabatt) größer als 1000 Cent und die Temperatur (wie vom `Wetter`-Objekt angegeben) größer als 30 ist. Benutzen Sie in der Methode keine `int`-Literale mit Werten größer als 1, sondern speichern Sie die Werte als Klassenkonstanten (statische, finale Attribute).

Überschreiben Sie die `toString`-Methode, sodass die **String-Repräsentationen aller Tickets** und der **Gesamtpreis** zurückgegeben werden.

Quittung.java Java

```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```



Quittung.java (Fortsetzung) Java

```
}
```

___/4 (d) Ergänzen Sie die `main`-Methode der Klasse `Kassenautomat`. Die Methode nimmt das eingeworfene Geld (in Cent), die Anzahl der Erwachsenen- und Kinder-Tickets entgegen und soll dann folgendes tun:

1. Eine Instanz von `FakeWetter` mit Temperatur 30 wird erstellt.
2. Eine Quittung wird erstellt.
3. Wenn genug Geld eingeworfen wurde, um die Quittung zu bezahlen, wird die String-Repräsentation der Quittung ausgegeben.
4. Andernfalls wird `zu wenig Geld` ausgegeben

Kassenautomat.java — Java

```
class Kassenautomat {
  static void main(String[] args) {
    int geldGegeben = Integer.parseInt(args[0]);
    int anzahlErwachsene = Integer.parseInt(args[1]);
    int anzahlKinder = Integer.parseInt(args[2]);

    Wetter w = ____________________________________________;

    Quittung q = ____________________________________________;


    if(____________________________________________) {

      ____________________________________________;

    } else {

      ____________________________________________;

    }
  }
}
```

___/1 (e) Könnte bei der Lücke `Wetter w = ______;` auch ein Lambda-Ausdruck eingesetzt werden? Begründen Sie Ihre Antwort. (Falls die Antwort Ja ist, müssen Sie den Lambda-Ausdruck *nicht* angeben.)

**Notizen auf dieser Seite werden nicht korrigiert!**

Auszug aus der Dokumentation für ausgewählte öffentliche Klassen und Methoden aus dem JDK, gekürzt auf die klausurrelevanten Informationen.

### Interface `java.util.Collection<E>`

alle Collections können in for-each-Schleifen verwendet werden

- `boolean add(E e)` – fügt `e` in die Collection ein; gibt `true` zurück, wenn sich der Inhalt der Collection durch den `add`-Aufruf verändert hat
- `int size()` – gibt die Anzahl der Elemente in der Collection zurück
- `Stream<E> stream()` – erstellt einen Stream, der die Elemente der Collection enthält

### Interface `java.util.Set<E>`

`extends Collection<E>`

### Class `java.util.HashSet<E>`

`implements Set<E>`

- Konstruktor `HashSet()` – erstellt neues, leeres HashSet

### Interface `java.util.List<E>`

`extends Collection<E>`

- `boolean add(E e)` – fügt `e` hinten in die Liste ein
- `E get(int index)` – gibt das Element am gegebenen Index zurück
- `E remove(int index)` – löscht das Element am gegebenen Index und gibt das gelöschte Element zurück
- `E set(int index, E element)` – ersetzt das bereits vorhandene Element am gegebenen Index durch das übergebene Element; gibt das alte Element am Index zurück

### Class `java.util.ArrayList<E>`

`implements java.util.List<E>`

Array mit dynamischer Größe

- Konstruktor `ArrayList()` – erstellt eine neue, leere ArrayList
- Konstruktor `ArrayList(Collection<? extends E> c)` – erstellt eine neue ArrayList, die die Elemente von `c` enthält (Shallow Copy)

### Class `java.util.LinkedList<E>`

`implements java.util.List<E>`

doppelt verkettete Liste

- Konstruktor `LinkedList()` – erstellt eine neue, leere LinkedList

**Notizen auf dieser Seite werden nicht korrigiert!**

### Class `java.util.Collections`

- `static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)` – sucht `key` mithilfe binärer Suche in `list`; gibt den Index von `key` zurück, falls gefunden; gibt eine negative Zahl zurück, falls nicht gefunden; geht davon aus, dass `list` gemäß natürlicher Sortierung sortiert ist
- `static <T extends Comparable<? super T>> void sort(List<T> list)` – sortiert die Elemente der Liste aufsteigend nach ihrer natürlichen Sortierung

### Class `java.util.Arrays`

- `static String toString(int[] a)` – gibt eine String-Repräsentation des Arrays in der Form `[element1, element2, element3]` zurück

### Interface `java.lang.Comparable<T>`

durch Implementieren dieses Interfaces definieren Klassen ihre natürliche Sortierung

- `int compareTo(T o)` – gibt eine negative Zahl, Null oder eine positive Zahl zurück, wenn dieses Objekt kleiner, gleich bzw. größer als das übergebene Objekt ist

### Class `java.lang.String`

`implements Comparable<String>` – natürliche Sortierung ist alphabetisch aufsteigend

- `char charAt(int index)` – gibt den Character am übergebenen Index zurück
- `boolean contains(String s)` – gibt genau dann `true` zurück, wenn dieser String den übergebenen String enthält
- `boolean endsWith(String s)` – gibt genau dann `true` zurück, wenn dieser String mit dem übergebenen String endet
- `int length()` – gibt die Länge dieses Strings zurück
- `String replace(String target, String replacement)` – gibt einen neuen String mit gleichem Inhalt zurück, allerdings wurde jedes Vorkommen von `target` durch `replacement` ersetzt
  Beispiel: `"abcb".replace("b", "x")` gibt String `"axcx"` zurück
- `String[] split(String s)` – teilt diesen String überall dort auf, wo der übergebene String gefunden wird
  Beispiel: `"A;B;C".split(";")` erzeugt Array mit den Strings `"A"`, `"B"`, `"C"`
- `char[] toCharArray()` – gibt ein Array zurück, das die Zeichen des Strings enthält
- `String toLowerCase()` – gibt einen neuen String mit gleichem Inhalt zurück, aber alle enthaltenen Buchstaben wurden in Kleinbuchstaben umgewandelt
- `String toUpperCase()` – gibt einen neuen String mit gleichem Inhalt zurück, aber alle enthaltenen Buchstaben wurden in Großbuchstaben umgewandelt
- `static String valueOf(int i)` – gibt die String-Repräsentation des Parameters zurück; es gibt überladene Varianten für alle primitiven Datentypen