



Name: _____

Abiturprüfung 2021

Informatik, Leistungskurs

Aufgabenstellung:

Die Schülerinnen und Schüler eines Projektkurses zum Thema „Klimawandel“ möchten eine Software zur Verwaltung und Analyse von Wetterdaten entwickeln.

Dem Projektkurs liegen dafür die Daten von Wetterstationen in ganz Deutschland mit ihren eindeutigen Stationsnamen und der jeweiligen Stadt vor. Jede Wetterstation kann – je nach Ausstattung – verschiedene Arten von Messwerten (z. B. die Temperatur in °C oder die Windgeschwindigkeit in km/h) ermitteln. Für jede Messung wird die Messwertart, der jeweilige Messwert und der Zeitpunkt gespeichert (vgl. Abbildung 1).

Stationsname	Stadt	Messungen		
Botanischer Garten	Berlin	Messwertart	Messwert	Zeitpunkt
		Temperatur	17.0	2020-08-04 09:01
		Temperatur	18.4	2020-08-04 10:02
		Temperatur	20.6	2020-08-04 12:07
Leuchtturm	Kiel	Messwertart	Messwert	Zeitpunkt
		Windgeschwindigkeit	2.0	2019-08-03 10:02
		Windgeschwindigkeit	20.0	2019-12-31 12:07
Zugspitze	Garmisch-Partenkirchen	Messwertart	Messwert	Zeitpunkt
		Temperatur	−10.3	2019-12-31 09:01
		Windgeschwindigkeit	20.0	2019-12-31 09:01
		Temperatur	4.0	2020-08-04 13:01
		Windgeschwindigkeit	32.0	2020-08-04 13:01
Teststation	Soest	Messwertart	Messwert	Zeitpunkt
		(noch keine Messungen vorhanden)		

Abbildung 1: Beispieldaten von ausgewählten Wetterstationen



Name: _____

Eine erste Modellierung für die Verwaltung der Wetterdaten ist im folgenden Implementationsdiagramm dargestellt. Die Dokumentationen der Klassen befinden sich im Anhang.

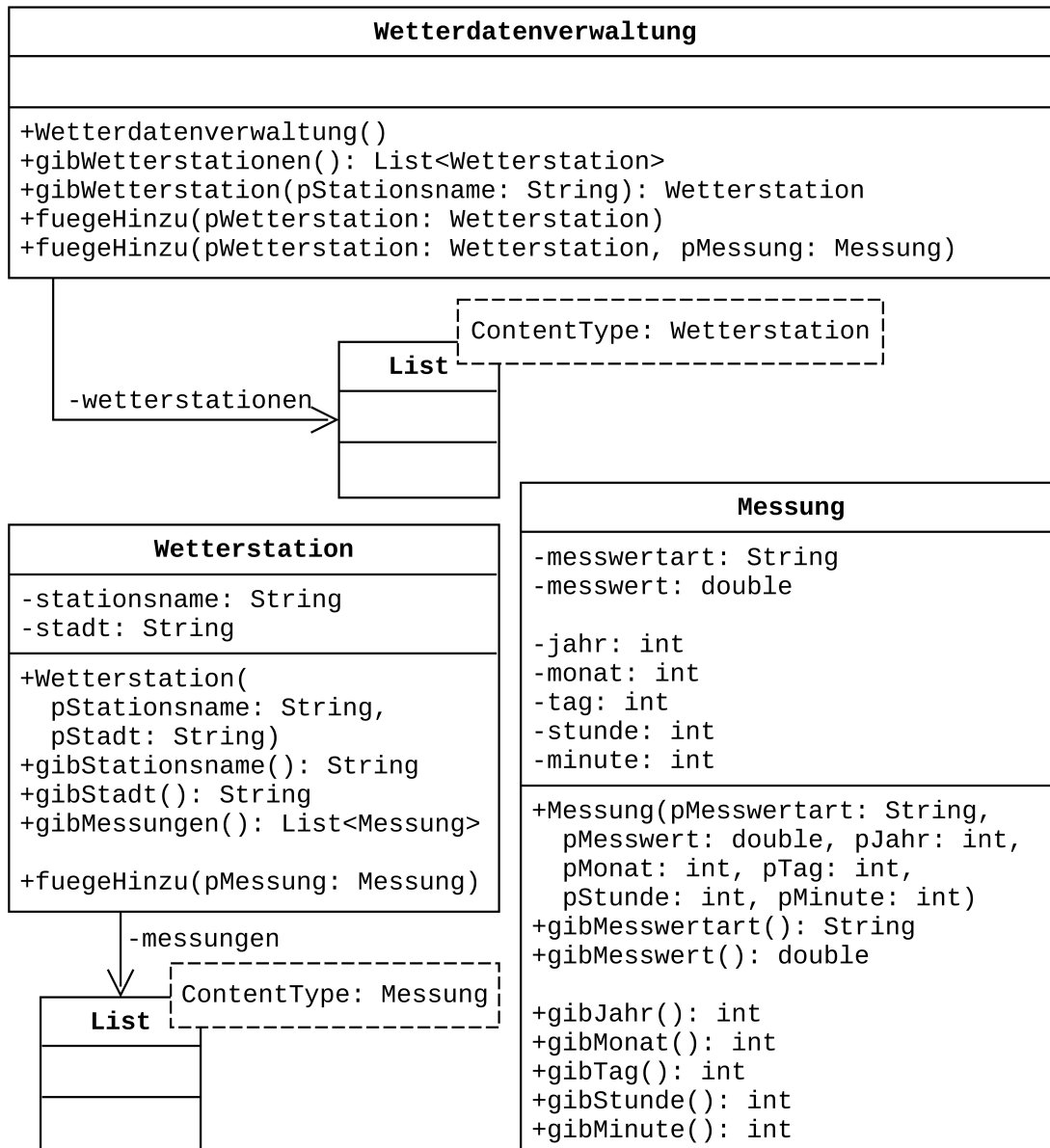


Abbildung 2: Teilmodellierung als Implementationsdiagramm

- a) Beschreiben Sie die Beziehungen zwischen den Klassen der in Abbildung 2 gegebenen Teilmodellierung.

Begründen Sie im Sachkontext, warum es sinnvoll ist, die Datensammlungen für die Wetterstationen und die Messungen jeweils als lineare Liste und nicht als Feld (Array) zu realisieren.

(5 Punkte)



Name: _____

- b) Nachdem schon einige Arbeit in das Projekt geflossen ist, entdeckt eine Schülerin in der Klasse Wetterdatenverwaltung die undokumentierte Methode `ermittleEtwas`.

```
1 public Wetterstation ermittleEtwas(String pMesswertart,
2                                     int pJahr, int pMonat, int pTag) {
3     Wetterstation gesuchteStation = null;
4     Messung tempMessung = null;
5     wetterstationen.moveToFirst();
6     while (wetterstationen.hasAccess()) {
7         Wetterstation station = wetterstationen.getContent();
8         List<Messung> messungen = station.gibMessungen();
9         messungen.moveToFirst();
10        while (messungen.hasAccess()) {
11            Messung m = messungen.getContent();
12            if (m.gibMesswertart().equals(pMesswertart)
13                && m.gibJahr() == pJahr
14                && m.gibMonat() == pMonat
15                && m.gibTag() == pTag
16                && (gesuchteStation == null
17                    || m.gibMesswert() > tempMessung.gibMesswert())) {
18                gesuchteStation = station;
19                tempMessung = m;
20            }
21            messungen.next();
22        }
23        wetterstationen.next();
24    }
25    return gesuchteStation;
26 }
```

Hinweis: Gehen Sie hier davon aus, dass die Wetterstationen in der Reihenfolge vorliegen, die durch die Beispieldaten in Abbildung 1 gegeben ist.

Analysieren Sie die Methode und geben Sie für die Beispieldaten aus Abbildung 1 die Rückgabewerte für die Methodenaufrufe

`ermittleEtwas("Temperatur", 2020, 8, 4)`,
`ermittleEtwas("Windgeschwindigkeit", 2019, 12, 31)` und
`ermittleEtwas("Windgeschwindigkeit", 2030, 1, 1)` an.

Erläutern Sie, unter welchen Umständen die Methode `null` zurückliefert.

Entscheiden Sie, ob in Zeile 16 der Teil der Bedingung
`gesuchteStation == null`

durch

`tempMessung == null`

ersetzt werden könnte.

Erläutern Sie die Aufgabe der Methode im Sachzusammenhang.

(12 Punkte)



Name: _____

- c) Der Klasse `Wetterstation` soll die Methode `ermittleSortierteMessungen` hinzugefügt werden. Sie soll für die im Parameter übergebene Messwertart `pMesswertart` eine nach den Messwerten aufsteigend sortierte Liste der Messungen der jeweiligen Wetterstation zurückliefern. Wenn keine Messungen für die gesuchte Wetterstation oder Messwertart vorliegen, dann soll eine leere Liste zurückgeliefert werden.

Die Methode erhält den folgenden Methodenkopf:

```
public List<Messung> ermittleSortierteMessungen(String  
                                              pMesswertart)
```

Entwickeln und erläutern Sie ein algorithmisches Verfahren für die Methode.

Implementieren Sie die Methode.

(13 Punkte)

- d) Nachdem der erste Prototyp der Software getestet wurde, wird klar, dass das Projekt erweitert werden muss, um wirklich aussagekräftige Ergebnisse zu liefern. Folgende Anforderungen sollen zusätzlich umgesetzt werden:
- Der Zeitpunkt einer Messung soll so präzisiert werden, dass in Zukunft zusätzlich zum Jahr, zum Monat, zum Tag, zur Stunde und zur Minute auch noch die Sekunde erfasst werden soll. Der so beschriebene Zeitpunkt soll in eine eigene Klasse ausgelagert werden, welche auch jeweils die Abfrage der genannten Attribute einzeln ermöglichen soll.
 - Um möglichst schnell zu jeder Messung die zugehörige Wetterstation ermitteln zu können, soll eine Messung auch Kenntnis über ihre Wetterstation haben. Eine Messung soll ihre Wetterstation auch zurückliefern können.
 - Der Projektkurs möchte in Zukunft auch mobile Messungen durchführen können, z. B. mit Wetterballons oder Treibbojen. Eine mobile Messung soll weiterhin einer Wetterstation zugeordnet sein. Zusätzlich soll nur für jede mobile Messung ihre Position in Form von zwei Werten gespeichert werden (z. B. 51.955013, 7.611646). Beide Werte sollen auch abgerufen werden können.

Modellieren Sie die oben genannten Anforderungen als Erweiterung des Implementationsdiagramms aus Abbildung 2.

Erläutern Sie, wie Sie die zweite und dritte Anforderung in Ihrem Implementationsdiagramm realisieren.

Hinweis: Unveränderte Klassen, Methoden und Attribute aus dem Implementationsdiagramm in Abbildung 2 müssen nicht dargestellt bzw. angegeben werden.

(13 Punkte)



Name: _____

- e) Ein Schüler schlägt vor, dass man die Klassen Wetterstation und Messung durch eine neue Klasse Wetteraufzeichnung ersetzen sollte. Im Folgenden werden Beispieldaten für Wetteraufzeichnungen dargestellt:

Stations-name	Stadt	Messwertart	Messwert	Zeitpunkt
Botanischer Garten	Berlin	Temperatur	17.0	2020-08-04 09:01
Botanischer Garten	Berlin	Temperatur	18.4	2020-08-04 10:02
Botanischer Garten	Berlin	Temperatur	20.6	2020-08-04 12:07
Leuchtturm	Kiel	Windgeschwindigkeit	2.0	2019-08-03 10:02
Leuchtturm	Kiel	Windgeschwindigkeit	20.0	2019-12-31 12:07
Zugspitze	Garmisch-Partenkirchen	Temperatur	-10.3	2019-12-31 09:01
Zugspitze	Garmisch-Partenkirchen	Windgeschwindigkeit	20.0	2019-12-31 09:01
Zugspitze	Garmisch-Partenkirchen	Temperatur	4.0	2020-08-04 13:01
Zugspitze	Garmisch-Partenkirchen	Windgeschwindigkeit	32.0	2020-08-04 13:01
Teststation	Soest			

Abbildung 3: Beispieldaten für Wetteraufzeichnungen

Die Klasse Wetterdatenverwaltung würde dann eine Liste von Objekten der Klasse Wetteraufzeichnung verwalten.

Der Schüler behauptet, dass man aus dieser einen Liste von Wetteraufzeichnungen schließlich auch alle Informationen auslesen könne, sein Vorschlag einfacher zu implementieren sei und sonst keine nennenswerten Nachteile habe.

Beurteilen Sie die Behauptung des Schülers.

(7 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung



Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse **Wetterdatenverwaltung**

Objekte der Klasse **Wetterdatenverwaltung** verwalten Wetterstationen mit ihren Messungen.

Auszug aus der Dokumentation der Klasse **Wetterdatenverwaltung**

Wetterdatenverwaltung()

Ein Objekt der Klasse **Wetterdatenverwaltung** wird initialisiert.

List<Wetterstation> gibWetterstationen()

Eine Liste mit allen Wetterstationen wird zurückgeliefert.

Wetterstation gibWetterstation(String pStationsname)

Die Wetterstation mit dem Namen `pStationsname` wird zurückgeliefert. Wenn zu diesem Namen keine Wetterstation existiert, dann wird `null` zurückgeliefert.

void fuegeHinzu(Wetterstation pWetterstation)

Die Wetterstation `pWetterstation` wird der **Wetterdatenverwaltung** hinzugefügt, wenn der Stationsname bisher noch nicht existiert. Sonst geschieht nichts.

void fuegeHinzu(Wetterstation pWetterstation, Messung pMessung)

Wenn die Wetterstation `pWetterstation` in der **Wetterdatenverwaltung** existiert, dann wird dieser Station die Messung `pMessung` hinzugefügt. Sonst geschieht nichts.

Double ermittleEtwas(String pMesswertart, int pJahr)

Diese Methode soll in Teilaufgabe b) analysiert werden.



Name: _____

Die Klasse Wetterstation

Objekte der Klasse **Wetterstation** verwalten ihren Stationsnamen, ihre Stadt und ihre Messungen.

Auszug aus der Dokumentation der Klasse Wetterstation

Wetterstation(String pStationsname, String pStadt)

Ein Objekt der Klasse **Wetterstation** wird mit dem Stationsnamen **pStationsname** und der Stadt **pStadt** initialisiert.

String gibStationsname()

Der Stationsname wird zurückgeliefert.

String gibStadt()

Die Stadt wird zurückgeliefert.

List<Messung> gibMessungen()

Eine Liste mit allen Messungen dieser Wetterstation wird zurückgeliefert.

void fuegeHinzu(Messung pMessung)

Die Messung **pMessung** wird hinzugefügt.

List<Messung> ermittleSortierteMessungen(String pMesswertart)

Diese Methode soll in Teilaufgabe c) hinzugefügt werden.



Name: _____

Die Klasse Messung

Objekte der Klasse **Messung** verwalten ihre Messwertart, ihren Messwert und ihren Zeitpunkt.

Auszug aus der Dokumentation der Klasse Messung

```
Messung(String pMesswertart, double pMesswert, int pJahr,  
        int pMonat, int pTag, int pStunde, int pMinute)
```

Ein Objekt der Klasse Messung wird mit der Messwertart pMesswertart, dem Messwert pMesswert und dem Zeitpunkt der Messung initialisiert.

```
String gibMesswertart()
```

Die Messwertart wird zurückgeliefert.

```
double gibMesswert()
```

Der Messwert wird zurückgeliefert.

```
int gibJahr()
```

Das Jahr wird zurückgeliefert.

```
int gibMonat()
```

Der Monat wird zurückgeliefert.

```
int gibTag()
```

Der Tag wird zurückgeliefert.

```
int gibStunde()
```

Die Stunde wird zurückgeliefert.

```
int gibMinute()
```

Die Minute wird zurückgeliefert.



Name: _____

Die generische Klasse `List`

Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste kann durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

`List()`

Eine leere Liste wird erzeugt.

`boolean isEmpty()`

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

`boolean hasAccess()`

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

`void next()`

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

`void toFirst()`

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`void toLast()`

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`ContentType getContent()`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.



Name: _____

void setContent(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

void append(ContentType pContent)

Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess() == false`). Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

void insert(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pContent == null` ist, bleibt die Liste unverändert.

void concat(List<ContentType> pList)

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList == null` oder eine leere Liste ist, bleibt die Liste unverändert.

void remove()

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.

Unterlagen für die Lehrkraft

Abiturprüfung 2021

Informatik, Leistungskurs

1. Aufgabenart

Modellierung, Implementation und Analyse kontextbezogener Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung und Algorithmen

2. Aufgabenstellung¹

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2021 (Stand: August 2020)

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf. Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. Inhaltsfelder und inhaltliche Schwerpunkte

Daten und ihre Strukturierung

- Objekte und Klassen
 - Entwurfsdiagramme und Implementationsdiagramme
 - Lineare Strukturen
- Lineare Liste (Klasse List)*

Algorithmen

- Analyse, Entwurf und Implementierung von Algorithmen
- Algorithmen in ausgewählten informatischen Kontexten

Formale Sprachen und Automaten

- Syntax und Semantik einer Programmiersprache
 - Java

2. Medien/Materialien

- entfällt

¹ Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Ein Objekt der Klasse `Wetterdatenverwaltung` verwaltet im Attribut `wetterstationen` seine Wetterstationen in einer linearen Liste mit Objekten der Klasse `Wetterstation`. Jedes Objekt der Klasse `Wetterstation` verwaltet seine Messungen jeweils in einer weiteren linearen Liste, welche Objekte vom Typ `Messung` enthält. Das Objekt dieser Liste wird als `messungen` bezeichnet.

Da es möglich ist, dass zur Laufzeit weitere Wetterstationen und Messungen hinzugefügt werden können, ist es sinnvoll, eine dynamische Datenstruktur zu verwenden. Die lineare Liste erfüllt dieses Kriterium, wohingegen das Feld eine statische Datenstruktur ist.

Teilaufgabe b)

Der Methodenaufruf `ermittleEtwas("Temperatur", 2020, 8, 4)` liefert das Objekt der Klasse `Wetterstation` vom Botanischen Garten in Berlin.

Der Methodenaufruf `ermittleEtwas("Windgeschwindigkeit", 2019, 12, 31)` liefert das Objekt der Klasse `Wetterstation` vom Leuchtturm in Kiel.

Der Methodenaufruf `ermittleEtwas("Windgeschwindigkeit ", 2030, 1, 1)` liefert `null`.

Die Methode liefert `null` zurück, wenn für die im Parameter übergebene Messwertart, das Jahr, den Monat und den Tag in keiner Wetterstation keine Messung vorliegt.

Der Teil der Bedingung in Zeile 16

```
    gesuchteStation == null
```

könnte durch

```
    tempMessung == null
```

ersetzt werden, weil beide Variablen (`gesuchteStation` und `tempMessung`) erst in Zeile 18 und 19 auf ein Objekt verweisen, wenn die erste geeignete Messung gefunden wurde.

Die Methode ermittelt die Wetterstation mit dem höchsten Messwert für die im Parameter übergebene Messwertart und das Datum. Wenn mehrere Wetterstationen den gleichen höchsten Messwert gespeichert haben, wird die Wetterstation zurückgeliefert, die am weitesten vorne in der Liste der Wetterstationen steht.

Teilaufgabe c)

Ein algorithmisches Verfahren:

Erzeuge eine lokale Liste `relevante`, die Objekte vom Typ `Messung` verwaltet, in der zunächst alle Messungen für diese Wetterstation mit der zugehörigen Messwertart referenziert werden.

Erzeuge eine lokale Liste `sortierte`, die Objekte vom Typ `Messung` verwaltet, in der anschließend die relevanten Messungen schrittweise sortiert eingefügt werden.

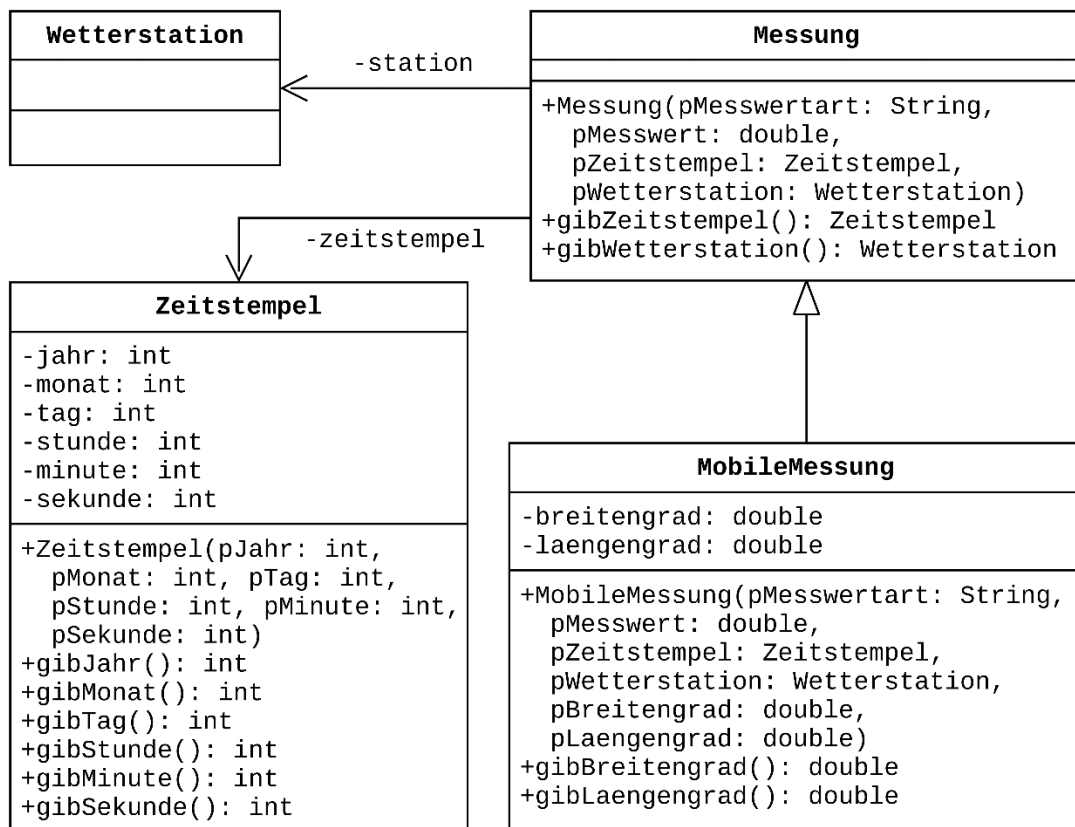
Durchlaufe die Liste `messungen` mit allen Objekten vom Typ `Messung` und betrachte jedes Objekt einer Messung. Hänge die jeweilige Messung, falls diese der im Parameter übergebenen Messwertart `pMesswertart` entspricht, an die Liste `relevante` an.

Durchlaufe die Liste `relevante` mit allen Objekten vom Typ `Messung` und füge schrittweise jedes Objekt sortiert in die Liste `sortierte` ein. Suche dafür in der Liste `sortierte` – beginnend am Anfang – die richtige Einfügestelle.

Die Liste `sortierte` wird zurückgeliefert.

Eine mögliche Implementation der Methode:

```
public List<Messung> ermittleSortierteMessungen(
    String pMesswertart) {
    List<Messung> relevante = new List<Messung>();
    List<Messung> sortierte = new List<Messung>();
    messungen.toFirst();
    while (messungen.hasAccess()) {
        Messung messung = messungen.getContent();
        if (messung.gibMesswertart().equals(pMesswertart)) {
            relevante.append(messung);
        }
        messungen.next();
    }
    relevante.toFirst();
    while (relevante.hasAccess()) {
        Messung messung = relevante.getContent();
        sortierte.toFirst();
        while (sortierte.hasAccess()
            && messung.gibMesswert() >=
                sortierte.getContent().gibMesswert()) {
            sortierte.next();
        }
        if (sortierte.hasAccess()) {
            sortierte.insert(messung);
        } else {
            sortierte.append(messung);
        }
        relevante.next();
    }
    return sortierte;
}
```

Teilaufgabe d)

Realisierung der zweiten Anforderung:

Ein Objekt der Klasse Messung verwaltet eine Referenz auf ein Objekt der Klasse Wetterstation, welche nun zusätzlich im Konstruktor übergeben wird. Die Wetterstation für eine Messung kann über die Methode gibWetterstation abgefragt werden.

Realisierung der dritten Anforderung:

Da bei einer mobilen Messung zusätzlich zwei Werte für die Position gespeichert werden sollen, erbt die Klasse MobileMessung alle Attribute und Methoden der Klasse Messung und erweitert diese um die zwei Attribute breitengrad und laengengrad jeweils vom Typ double. Die beiden Werte für die Position werden im Konstruktor der Klasse MobileMessung als Parameter übergeben und können über die zwei Getter-Methoden abgerufen werden.

Teilaufgabe e)

Durch eine vollständige Verschmelzung der Klassen `Wetterstation` und `Messung` zur neuen Klasse `Wetteraufzeichnung` könnten alle Informationen weiterhin ausgelesen werden.

Der Alternativvorschlag des Schülers ist nicht wesentlich einfacher zu implementieren, weil grundsätzlich die gleichen Attribute, Datentypen und Datenstrukturen verwendet werden.

Im Ursprungsmodell werden für jede Wetterstation nur die eigenen Messungen verwaltet, d. h. man könnte direkt auf diese Liste der Messungen zugreifen.

Um im vom Schüler vorgeschlagenen Alternativmodell die Messungen einer bestimmten Station zu ermitteln, muss zwangsläufig die gesamte Liste aller Wetteraufzeichnungen durchlaufen werden, was deutlich aufwändiger ist und einen nennenswerten Nachteil darstellt.

Der Vorschlag des Schülers ist grundsätzlich umsetzbar, aber dennoch abzulehnen.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK ²	ZK	DK
1	beschreibt die Beziehungen zwischen den Klassen der in Abbildung 2 gegebenen Teilmodellierung.	3			
2	begründet im Sachkontext, warum es sinnvoll ist, die Datensammlungen für die Wetterstationen und die Messungen jeweils als lineare Liste und nicht als Feld (Array) zu realisieren.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (5)					
.....					
.....					
	Summe Teilaufgabe a)	5			

Teilaufgabe b)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	analysiert die Methode und gibt für die Beispieldaten aus Abbildung 1 die Rückgabewerte für die drei Methodenaufrufe an.	5			
2	erläutert, unter welchen Umständen die Methode null zurückliefert.	2			
3	entscheidet, ob in Zeile 16 der Teil der Bedingung ersetzt werden könnte.	2			
4	erläutert die Aufgabe der Methode im Sachzusammenhang.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (12)					
.....					
.....					
	Summe Teilaufgabe b)	12			

² EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	entwickelt und erläutert ein algorithmisches Verfahren für die Methode.	5			
2	implementiert die Methode.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (13)					
.....					
.....					
	Summe Teilaufgabe c)	13			

Teilaufgabe d)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	modelliert die erste Anforderung (i) als Erweiterung des Implementationsdiagramms aus Abbildung 2.	2			
2	modelliert die zweite Anforderung (ii) als Erweiterung des Implementationsdiagramms aus Abbildung 2.	3			
3	modelliert die dritte Anforderung (iii) als Erweiterung des Implementationsdiagramms aus Abbildung 2.	4			
4	erläutert, wie die zweite und dritte Anforderung in seinem Implementationsdiagramm realisiert wird.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (13)					
.....					
.....					
	Summe Teilaufgabe d)	13			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	beurteilt die Behauptung des Schülers.	7			
Sachlich richtige Lösungsalternative zur Modelllösung: (7)					
.....					
.....					
	Summe Teilaufgabe e)	7			
	Summe insgesamt	50			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	50			
Punktzahl der gesamten Prüfungsleistung	150			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOST				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOST

Die Klausur wird abschließend mit der Note _____ (_____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	Punkte	Erreichte Punktzahl
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0



Name: _____

Abiturprüfung 2021

Informatik, Leistungskurs

Aufgabenstellung:

Ein junges Startup-Unternehmen verleiht in Köln Elektro-Roller, sogenannte Scooter. Alle Scooter sind mit einem GPS-Empfänger ausgestattet und übermitteln über eine Internetverbindung im Minutentakt ihre aktuelle Position (Standort) sowie den aktuellen Batterie-Zustand (Akkuladung) an den Server des Unternehmens. Um den Abstand eines Scooters zur Firmenzentrale leichter berechnen zu können, übermitteln die Scooter ihre Position relativ zur Firmenzentrale des Unternehmens (Koordinatenursprung). Die jeweils letzte Position wird serverseitig als Punkt in einem Koordinatensystem gespeichert. Abbildung 1 zeigt die aktuelle Situation im näheren Umfeld der Firmenzentrale.

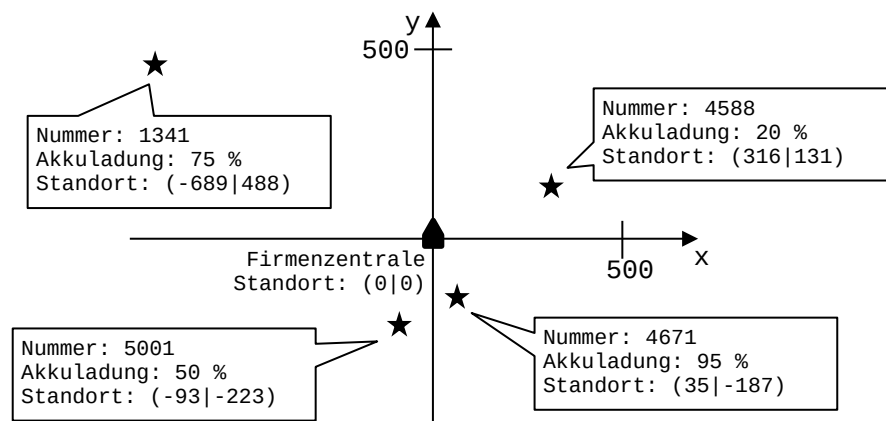


Abbildung 1: Aktuelle Positionen der Scooter in der Nähe der Firmenzentrale

Das Unternehmen hat sich dazu entschieden, die Scooter in einem binären Suchbaum zu verwalten. Die eindeutige vierstellige (Serien-)Nummer der Scooter dient dazu als Suchschlüssel innerhalb des Baums.

Das Unternehmen nutzt bereits die vier in Abbildung 1 dargestellten Scooter. Der sich daraus ergebende binäre Suchbaum ist in Abbildung 2 dargestellt.

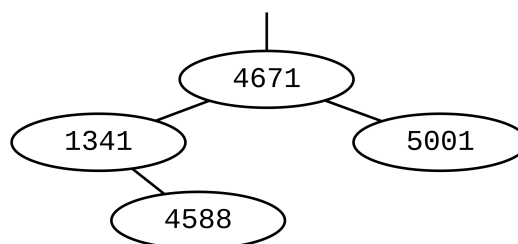


Abbildung 2: Suchbaum mit den Suchschlüsseln der vier Scooter



Name: _____

- a) Das Unternehmen besitzt noch mehr Scooter, welche in den vorhandenen Datenbestand eingefügt werden sollen. Zunächst werden vier Scooter mit den Nummern 4999, 4400, 1000 und 8500 in genau dieser Reihenfolge in den Suchbaum aus Abbildung 2 eingefügt.

Stellen Sie den nach dem Einfügen entstandenen Suchbaum grafisch dar.

Stellen Sie den Suchbaum grafisch dar, der aus dem Suchbaum mit den acht Suchschlüsseln entsteht, wenn der Scooter mit der Nummer 4671 gestohlen wurde und deshalb aus dem System gelöscht werden muss.

(7 Punkte)

Das Unternehmen verwaltet mithilfe seiner Software neben den Daten der Scooter auch die Daten der Kundinnen und Kunden, die einen Scooter ausleihen möchten. Aus Gründen der Vereinfachung ist dabei die Verwaltung so modelliert, dass jede Kundin und jeder Kunde lediglich durch eine eindeutige E-Mail-Adresse und eine Kreditkartennummer zu Abrechnungszwecken beschrieben wird.

Die Software wurde auf Basis des in Abbildung 3 dargestellten Ausschnitts des Implementationsdiagramms entwickelt.



Name: _____

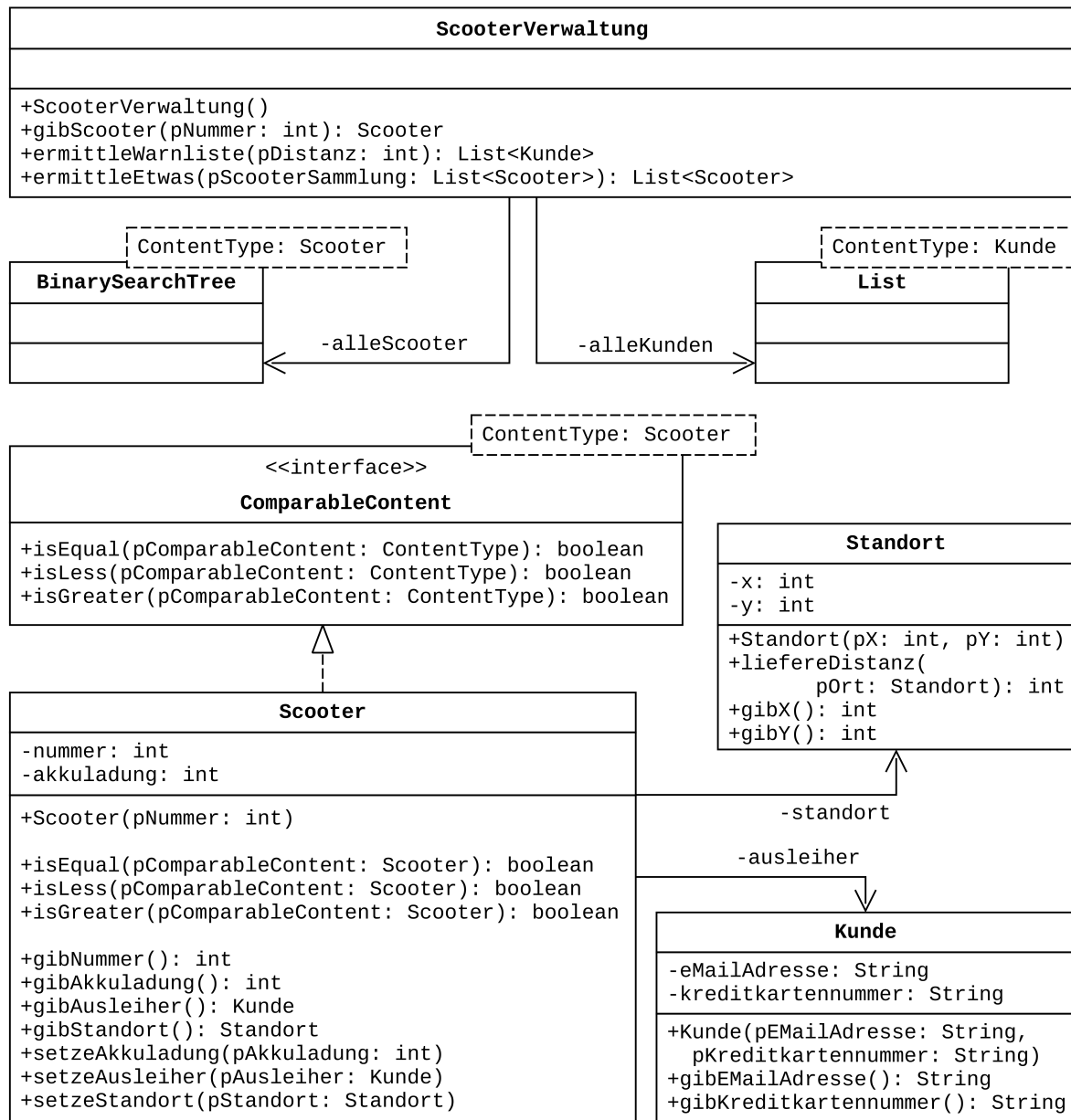


Abbildung 3: Ausschnitt des Implementationsdiagramms

- b) Erläutern Sie die Beziehungen zwischen den Klassen *ScooterVerwaltung*, *Scooter*, *Kunde*, *List* und *BinarySearchTree* sowie dem Interface *ComparableContent*.

Begründen Sie im Sachkontext, weshalb die Verwaltung der Scooter in Form eines binären Suchbaums effizienter ist als die Verwaltung in einer linearen Liste, wohingegen die Verwaltung der Kundinnen und Kunden in Form einer Liste vertretbar ist.

(8 Punkte)



Name: _____

Jeden Abend muss überprüft werden, ob alle Scooter vorschriftsmäßig geparkt wurden. Zu diesem Zweck hat sich das Unternehmen eine Drohne angeschafft, welche abends alle Scooter abfliegt und jeweils ein Foto des Scooters an die Firmenzentrale schickt.

c) Ein Praktikant hat die folgende Methode der Klasse ScooterVerwaltung implementiert.

```
1 public List<Scooter> ermittleEtwas(  
2     List<Scooter> pScooterSammlung) {  
3     Standort ort = new Standort(0, 0);  
4     List<Scooter> ergebnis = new List<Scooter>();  
5     while (!pScooterSammlung.isEmpty()) {  
6         pScooterSammlung.moveToFirst();  
7         Scooter naechster = pScooterSammlung.getContent();  
8         pScooterSammlung.next();  
9         while (pScooterSammlung.hasAccess()) {  
10            Scooter test = pScooterSammlung.getContent();  
11            if (ort.liefereDistanz(test.gibStandort())  
12                < ort.liefereDistanz(naechster.gibStandort())) {  
13                naechster = test;  
14            }  
15            pScooterSammlung.next();  
16        }  
17        pScooterSammlung.moveToFirst();  
18        while (pScooterSammlung.getContent() != naechster) {  
19            pScooterSammlung.next();  
20        }  
21        pScooterSammlung.remove();  
22        ergebnis.append(naechster);  
23        ort = naechster.gibStandort();  
24    }  
25    return ergebnis;  
26 }
```

Die Methode `ermittleEtwas` werde mit dem Parameter `pScooterSammlung` aufgerufen, welcher eine Liste referenziert, die die in Abbildung 1 dargestellten Scooter in der Reihenfolge 4588, 4671, 1341 und 5001 enthält.

Analysieren Sie den Quelltext der Methode, indem Sie sie auf die Beispieldaten anwenden und die so erzeugte Rückgabe-Liste ermitteln.

Erläutern Sie die Funktionsweise der Methode.

Erläutern Sie im Sachkontext, welche Information diese Methode liefert.

(12 Punkte)



Name: _____

- d) Das allabendliche Anfliegen der Scooter durch eine Drohne wird problematisch, wenn sich die Scooter zu weit von der Firmenzentrale entfernt befinden. Deshalb sollen Kundinnen und Kunden, die sich z. B. mehr als 10 000 Meter von der Firmenzentrale entfernen, über ihre hinterlegte E-Mail-Adresse eine Warnmeldung erhalten.

Zur Ermittlung der Liste aller Personen, die eine solche Warnmeldung erhalten sollen, stellt die Klasse ScooterVerwaltung die Methode `ermittleWarnliste` zur Verfügung, welche den folgenden Methodenkopf besitzt:

```
public List<Kunde> ermittleWarnliste(int pDistanz)
```

Die Dokumentation dieser Methode ist in der Anlage zu finden.

Entwickeln Sie eine Strategie für einen Algorithmus dieser Methode.

Implementieren Sie die Methode entsprechend der Dokumentation.

(11 Punkte)

- e) Ein Praktikant bemerkt, dass die Entfernung eines Scooters von der Firmenzentrale eine sehr wichtige Information ist, welche in verschiedenen Kontexten benötigt wird, und nennt hierfür zwei Beispiele:

- Für entlehene Scooter ist diese Entfernung zum Beispiel bei der Versendung von Warnmeldungen für zu weit entfernte Scooter wichtig.
- Für nicht entlehene Scooter ist diese Entfernung dagegen ein wichtiges Kriterium für den Flug der Drohne, die das vorschriftsmäßige Parken der Scooter überprüft.

Er schlägt deshalb vor, diese Entfernung zwischen Scooter und Firmenzentrale als Sortierkriterium in dem binären Suchbaum zu verwenden.

Erläutern Sie, welche Veränderungen in der Modellierung vorgenommen werden müssen, um den Vorschlag des Praktikanten umzusetzen.

Begründen Sie, warum der Vorschlag des Praktikanten problematisch ist, wenn verschiedene Scooter die gleiche Entfernung zur Firmenzentrale haben.

Beurteilen Sie mit Bezug auf die beiden vom Praktikanten genannten Anwendungsbeispiele, inwiefern der Modellierungsvorschlag des Praktikanten sinnvoll ist.

(12 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung



Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse Scooter

Objekte dieser Klasse verwalten die Daten eines Scooters.

Dokumentation der Klasse Scooter

Scooter(int pNummer)

Ein Objekt der Klasse wird initialisiert. Die eindeutige Nummer `pNummer` wird gespeichert.

boolean isEqual(Scooter pComparableContent)

Wenn die eindeutige Nummer des Scooter-Objekts, von dem die Methode aufgerufen wird, gleich der Nummer des Scooter-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

boolean isLess(Scooter pComparableContent)

Wenn die eindeutige Nummer des Scooter-Objekts, von dem die Methode aufgerufen wird, kleiner als die Nummer des Scooter-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

boolean isGreater(Scooter pComparableContent)

Wenn die eindeutige Nummer des Scooter-Objekts, von dem die Methode aufgerufen wird, größer als die Nummer des Scooter-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

int gibNummer()

Die Methode liefert die eindeutige Nummer des Scooters.

int gibAkkuladung()

Die Methode liefert den Batterie-Zustand des Scooters. Der Wert entspricht der Akkuladung in Prozent, d. h., der Wert liegt im Wertebereich 0 (Akku ist leer) bis 100 (Akku ist vollständig aufgeladen).

Kunde gibAusleiher()

Die Methode liefert das Objekt der Klasse `Kunde`, welcher den Scooter momentan ausgeliehen hat. Ist der Scooter momentan nicht verliehen, so wird `null` zurückgegeben.

Standort gibStandort()

Die Methode liefert ein Objekt der Klasse `Standort`, welches die Koordinaten relativ zur Firmenzentrale enthält. Z. B. bedeutet der Standort `(-300 | 400)`, dass sich der Scooter 300 Meter in westlicher Richtung und 400 Meter in nördlicher Richtung von der Firmenzentrale befindet.



Name: _____

void setzeAkkuladung(int pAkkuladung)

Die Methode verändert den Wert der Akkuladung auf den Wert pAkkuladung. Ist der Wert kleiner als 0 oder größer als 100, so geschieht nichts.

void setzeAusleiher(Kunde pAusleiher)

Die Methode speichert die Daten der Person, die den Scooter momentan ausgeliehen hat. Ist der Parameter pAusleiher gleich null, so ist der Scooter wieder frei und kann von jemand anderem ausgeliehen werden.

void setzeStandort(Standort pStandort)

Die Methode setzt den Standort des Scooters auf pStandort.

Die Klasse ScooterVerwaltung

Objekte dieser Klasse verwalten Scooter sowie Kundinnen und Kunden und ermöglichen die Ausleihe von Scootern.

Dokumentation der Klasse ScooterVerwaltung

ScooterVerwaltung()

Ein Objekt der Klasse wird initialisiert. Es verwaltet noch keine Scooter und keine Daten von Kundinnen und Kunden.

Scooter gibScooter(int pScooternummer)

Liefert das Scooter-Objekt mit der (Serien-)Nummer pScooternummer. Falls es kein Scooter-Objekt mit der entsprechenden (Serien-)Nummer gibt, dann wird null zurückgeliefert.

List<Kunde> ermittleWarnliste(int pDistanz)

Die Methode liefert eine Liste aller Kundinnen und Kunden, die sich mit ihrem ausgeliehenen Scooter um mehr als pDistanz Meter Luftlinie von der Firmenzentrale entfernt haben. Die Firmenzentrale befindet sich im Standort (0|0).

Gibt es keine Kundin und keinen Kunden, die bzw. der sich zu weit entfernt hat, oder ist der Parameter pDistanz < 0, so wird eine leere Liste zurückgegeben.

Scooter, die momentan nicht ausgeliehen wurden, sich aber in größerer Distanz befinden, werden nicht berücksichtigt, weil es keine Person gibt, an den eine Warnmeldung verschickt werden könnte.

List<Scooter> ermittleEtwas(List<Scooter> pScooterSammlung)

Die Methode wird in Teilaufgabe c) analysiert.



Name: _____

Die Klasse Kunde

Objekte dieser Klasse verwalten die Daten einer Kundin bzw. eines Kunden.

Dokumentation der Klasse Kunde

Kunde(String pEmailAdresse, String pKreditkartennummer)

Ein Objekt der Klasse wird initialisiert. Die eindeutige E-Mail-Adresse `pEmailAdresse` der Kundin bzw. des Kunden und die zugehörige Kreditkartennummer `pKreditkartennummer` werden gespeichert.

String gibEmailAdresse()

Die eindeutige E-Mail-Adresse der Kundin bzw. des Kunden wird zurückgeliefert.

String gibKreditkartennummer()

Die Kreditkartennummer der Kundin bzw. des Kunden wird zurückgeliefert.

Die Klasse Standort

Objekte dieser Klasse verwalten die Koordinaten eines Standorts für Scooter. Die Koordinaten werden relativ zur Firmenzentrale gespeichert, welche sich im Koordinatenursprung befindet.

Dokumentation der Klasse Standort

Standort(int pX, int pY)

Ein Objekt der Klasse wird initialisiert. Die Koordinaten (`pX|pY`) werden im Objekt gespeichert.

int liefereDistanz(Standort pOrt)

Die auf volle Meter abgerundete Distanz (Luftlinie in Metern) zwischen dem gespeicherten Ort und dem durch `pOrt` referenzierten Ort wird zurückgeliefert. Sollte `pOrt` den Wert `null` haben, so wird der Wert 0 zurückgeliefert.

int gibX()

Die x-Koordinate des Standorts wird zurückgeliefert. Diese entspricht der Entfernung in westlicher (bei negativem Wert) bzw. in östlicher (bei positivem Wert) Richtung in Metern von der Firmenzentrale.

int gibY()

Die y-Koordinate des Standorts wird zurückgeliefert. Diese entspricht der Entfernung in südlicher (bei negativem Wert) bzw. in nördlicher (bei positivem Wert) Richtung in Metern von der Firmenzentrale.



Name: _____

Die generische Klasse `List`

Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste kann durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

`List()`

Eine leere Liste wird erzeugt.

`boolean isEmpty()`

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

`boolean hasAccess()`

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

`void next()`

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

`void toFirst()`

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`void toLast()`

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`ContentType getContent()`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.



Name: _____

void setContent(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

void append(ContentType pContent)

Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess() == false`). Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

void insert(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pContent == null` ist, bleibt die Liste unverändert.

void concat(List<ContentType> pList)

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList == null` oder eine leere Liste ist, bleibt die Liste unverändert.

void remove()

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.



Name: _____

Die generische Klasse `BinarySearchTree`

Mithilfe der generischen Klasse `BinarySearchTree` können beliebig viele Objekte des Typs `ContentType` in einem Binärbaum (binärer Suchbaum) entsprechend einer Ordnungsrelation verwaltet werden.

Ein Objekt der Klasse `BinarySearchTree` stellt entweder einen leeren Baum dar oder verwaltet ein Inhaltsobjekt vom Typ `ContentType` sowie einen linken und einen rechten Teilbaum, die ebenfalls Objekte der Klasse `BinarySearchTree` sind.

Die Klasse der Objekte, die in dem Suchbaum verwaltet werden sollen, muss das generische Interface `ComparableContent` implementieren. Dabei muss durch Überschreiben der drei Vergleichsmethoden `isLess`, `isEqual`, `isGreater` (siehe Dokumentation des Interfaces) eine eindeutige Ordnungsrelation festgelegt sein.

Die Objekte der Klasse `ContentType` sind damit vollständig geordnet. Für je zwei Objekte `c1` und `c2` vom Typ `ContentType` gilt also insbesondere genau eine der drei Aussagen:

- `c1.isLess(c2)` (Sprechweise: `c1` ist kleiner als `c2`)
- `c1.isEqual(c2)` (Sprechweise: `c1` ist gleichgroß wie `c2`)
- `c1.isGreater(c2)` (Sprechweise: `c1` ist größer als `c2`)

Alle Objekte im linken Teilbaum sind kleiner als das Inhaltsobjekt des Binärbaumes. Alle Objekte im rechten Teilbaum sind größer als das Inhaltsobjekt des Binärbaumes. Diese Bedingung gilt auch in beiden Teilbäumen.



Name: _____

Dokumentation der generischen Klasse `BinarySearchTree<ContentType>` extends `ComparableContent<ContentType>`

`BinarySearchTree()`

Der Konstruktor erzeugt einen leeren Suchbaum.

`boolean isEmpty()`

Diese Anfrage liefert den Wahrheitswert `true`, wenn der Suchbaum leer ist, sonst liefert sie den Wert `false`.

`void insert(ContentType pContent)`

Falls bereits ein Objekt in dem Suchbaum vorhanden ist, das gleichgroß ist wie `pContent`, passiert nichts. Andernfalls wird das Objekt `pContent` entsprechend der Ordnungsrelation in den Baum eingeordnet. Falls der Parameter `null` ist, ändert sich nichts.

`ContentType search(ContentType pContent)`

Falls ein Objekt im binären Suchbaum enthalten ist, das gleichgroß ist wie `pContent`, liefert die Anfrage dieses, ansonsten wird `null` zurückgegeben. Falls der Parameter `null` ist, wird `null` zurückgegeben.

`void remove(ContentType pContent)`

Falls ein Objekt im binären Suchbaum enthalten ist, das gleichgroß ist wie `pContent`, wird dieses entfernt. Falls der Parameter `null` ist, ändert sich nichts.

`ContentType getContent()`

Diese Anfrage liefert das Inhaltsobjekt des Suchbaumes. Wenn der Suchbaum leer ist, wird `null` zurückgegeben.

`BinarySearchTree<ContentType> getLeftTree()`

Diese Anfrage liefert den linken Teilbaum des binären Suchbaumes. Der binäre Suchbaum ändert sich nicht. Wenn er leer ist, wird `null` zurückgegeben.

`BinarySearchTree<ContentType> getRightTree()`

Diese Anfrage liefert den rechten Teilbaum des Suchbaumes. Der Suchbaum ändert sich nicht. Wenn er leer ist, wird `null` zurückgegeben.



Name: _____

Das generische Interface ComparableContent<ContentType>

Das generische Interface ComparableContent muss von Klassen implementiert werden, deren Objekte in einen Suchbaum (BinarySearchTree) eingefügt werden sollen. Die Ordnungsrelation wird in diesen Klassen durch Überschreiben der drei implizit abstrakten Methoden isGreater, isEqual und isLess festgelegt.

Das Interface ComparableContent gibt folgende implizit abstrakte Methoden vor:

boolean isGreater(ContentType pComparableContent)

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation größer als das Objekt pComparableContent ist, wird true geliefert. Sonst wird false geliefert.

boolean isEqual(ContentType pComparableContent)

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation gleich dem Objekt pComparableContent ist, wird true geliefert. Sonst wird false geliefert.

boolean isLess(ContentType pComparableContent)

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation kleiner als das Objekt pComparableContent ist, wird true geliefert. Sonst wird false geliefert.

Unterlagen für die Lehrkraft

Abiturprüfung 2021

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation von kontextbezogenen Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung und Algorithmen

2. Aufgabenstellung¹

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2021 (Stand: August 2020)

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf. Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. Inhaltsfelder und inhaltliche Schwerpunkte

Daten und ihre Strukturierung

- Objekte und Klassen
 - Entwurfsdiagramme und Implementationsdiagramme
 - Lineare Strukturen
 - Lineare Liste (Klasse List)*
 - Nicht-lineare Strukturen
 - Binärer Suchbaum (Klasse BinarySearchTree)*

Algorithmen

- Analyse, Entwurf und Implementierung von Algorithmen
- Algorithmen in ausgewählten informatischen Kontexten

Formale Sprachen und Automaten

- Syntax und Semantik einer Programmiersprache
 - Java

2. Medien/Materialien

- entfällt

¹ Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

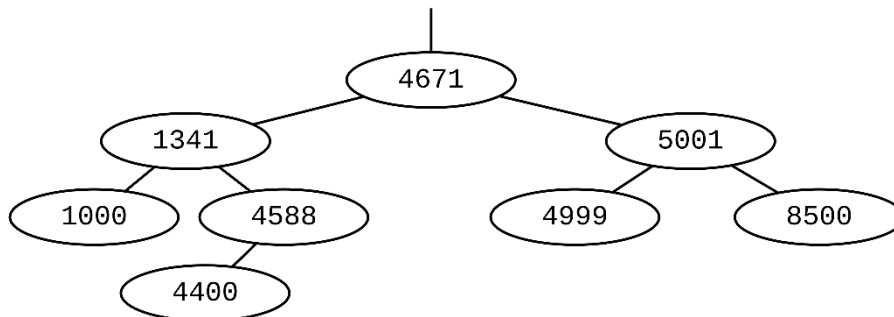
- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

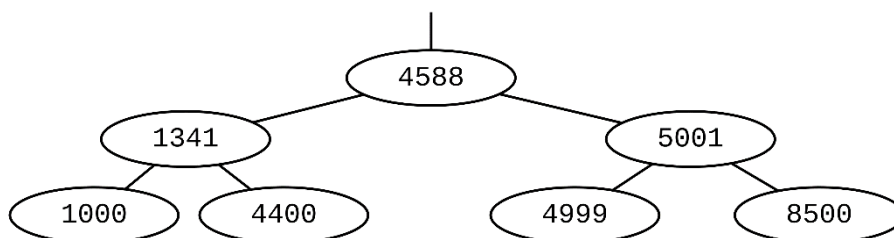
Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Der folgende Suchbaum entspricht den Anforderungen nach dem Einfügen der Suchschlüssel 4999, 4400, 1000 und 8500 in den Suchbaum.



Der folgende Suchbaum entspricht den Anforderungen nach dem Löschen des Suchschlüssels 4671 aus dem Suchbaum.



Teilaufgabe b)

Ein Objekt der Klasse `ScooterVerwaltung` verwaltet beliebig viele Objekte der Klasse `Kunde` in einem durch `Kunde` parametrisierten Objekt der Klasse `List`, welche durch das Attribut `alleKunden` referenziert wird. Zudem verwaltet ein Objekt der Klasse `ScooterVerwaltung` beliebig viele Objekte der Klasse `Scooter` in einem durch `Scooter` parametrisierten Objekt der Klasse `BinarySearchTree`, welches durch das Attribut `alleScooter` referenziert wird.

Die Klasse `Scooter` implementiert das mit `Scooter` parametrisierte Interface `ComparableContent`. Somit können Objekte der Klasse `Scooter` in einen binären Suchbaum eingefügt werden. Außerdem verwaltet ein Objekt der Klasse `Scooter` die Daten der Person, die den Scooter entliehen hat, in einem Objekt der Klasse `Kunde`.

Im Sachkontext ist eine Verwaltung der Scooter in einem binären Suchbaum deshalb sinnvoll, da die Scooter im Minutentakt ihren aktuellen Standort senden, welcher dann in den Objekten der Klasse `Scooter` aktualisiert werden muss. Es ist also eine sehr häufige Suche nach einem Scooter-Objekt nötig, weshalb der binäre Suchbaum aus Laufzeitgründen zu bevorzugen ist.

Die Verwaltung der Kundinnen und Kunden kann in Form einer Liste erfolgen, da lediglich beim Prozess des Ausleihens eine Kundin bzw. ein Kunde gesucht werden muss. Die Häufigkeit der Suche nach einer Kundin bzw. einem Kunden ist also wesentlich geringer als die Häufigkeit der Suche nach einem Scooter.

Teilaufgabe c)

Die Methode wird mit der Liste `pScooterSammlung` aufgerufen, welche die Scooter-Objekte in der Reihenfolge `[4588, 4671, 1341, 5001]` enthält. Die Methode ermittelt dann wie folgt die Rückgabe-Liste:

Schritt/Zeile	ort	pScooterSammlung	ergebnis	naechster
Initialisierung Zeile 1 bis 4	(0 0)	[4588, 4671, 1341, 5001]	[]	
Zeile 7				4588
Schleife Zeile 9 bis 16				4671
Zeile 17 bis 23	(35 -187)	[4588, 1341, 5001]	[4671]	
Zeile 7				4588
Schleife Zeile 9 bis 16				5001
Zeile 17 bis 23	(-93 -223)	[4588, 1341]	[4671, 5001]	
Zeile 7				4588
Schleife Zeile 9 bis 16				4588
Zeile 17 bis 23	(316 131)	[1341]	[4671, 5001, 4588]	
Zeile 7				1341
Zeile 21 bis 23	(-689 488)	[]	[4671, 5001, 4588, 1341]	

Zum Schluss wird die Liste `ergebnis` mit den Scooter-Objekten in der Reihenfolge `[4671, 5001, 4588, 1341]` zurückgegeben.

Die Funktionsweise der Methode entspricht einer Umsortierung der Liste `pScooterSammlung` in einer neuen Liste, welche zurückgeliefert wird. Nacheinander werden die Scooter-Objekte der Liste `pScooterSammlung` entfernt und in einer neuen Reihenfolge in die Ergebnisliste eingefügt.

Die Methode ermittelt eine Liste von Scootern, welche die gleichen Scooter enthält, wie die Liste des Parameters `pScooterSammlung`, allerdings in einer bezüglich der Problemstellung angepassten Reihenfolge. Die Reihenfolge ist dabei so gewählt, dass der erste Scooter derjenige ist, der der Firmenzentrale am nächsten liegt. Gibt es mehrere Scooter, die gleich weit entfernt liegen, so wird der erste Scooter der Liste `pScooterSammlung` gewählt. Alle weiteren Scooter sind so sortiert, dass sie jeweils immer der nächstgelegene Scooter zum vorherigen sind. Auch hier gilt, dass bei gleich weit entfernten Scootern immer derjenige zuerst gewählt wird, der in der Liste `pScooterSammlung` weiter vorne stand. Im Sachkontext wird damit also eine Besuchsreihenfolge ermittelt, welche die abzufliegende Gesamtstrecke für das allabendliche Anfliegen der Scooter ausgehend von der Firmenzentrale zu reduzieren versucht.

Teilaufgabe d)

Eine Strategie des Algorithmus lautet wie folgt:

- Durchlaufe den binären Suchbaum rekursiv und besuche jeden Knoten des Baums.
- Überprüfe innerhalb eines jeden Rekursionsschritts, ob der Baum leer ist. Ist dies der Fall, so ist die Rückgabe-Liste ebenfalls leer. Andernfalls wird das Scooter-Objekt der Wurzel des Baumes überprüft.
- Prüfe, ob dessen Distanz zur Firmenzentrale größer als der Parameter `pDistanz` ist und der Scooter ausgeliehen ist. Ist dies der Fall, so erstelle eine Liste mit diesem Scooter-Objekt als einzigem Objekt. Andernfalls erstelle eine leere Liste.
- Erstelle durch Rekursionsaufruf jeweils eine Liste für den linken Teilbaum und eine für den rechten Teilbaum.
- Erstelle die gesamte Rückgabe-Liste durch Aneinanderreihung der drei zuvor erstellten Listen und gib diese zurück.

Eine mögliche Implementierung lautet wie folgt:

```
public List<Kunde> ermittleWarnliste(int pDistanz) {
    return ermittleWarnliste(pDistanz, alleScooter);
}

private List<Kunde> ermittleWarnliste(
    int pDistanz, BinarySearchTree<Scooter> pBaum) {
    List<Kunde> ergebnis = new List<Kunde>();
    if (!pBaum.isEmpty()) {
        Scooter s = pBaum.getContent();
        if (s.gibStandort().liefereDistanz(new Standort(0,0))
            > pDistanz && s.gibAusleiher() != null) {
            ergebnis.append(s.gibAusleiher());
        }
        ergebnis.concat(ermittleWarnliste(
            pDistanz, pBaum.getLeftTree()));
        ergebnis.concat(ermittleWarnliste(
            pDistanz, pBaum.getRightTree()));
    }
    return ergebnis;
}
```

Teilaufgabe e)

Die bisherige Modellierung müsste wie folgt verändert werden:

Die Entfernung eines Scooters von der Firmenzentrale sollte eine Eigenschaft eines Objekts der Klasse `Scooter` sein, um so als Suchschlüssel verwendet zu werden. Die Klasse `Scooter` könnte also zum Beispiel um ein Attribut `entfernung` erweitert werden.

Weitere Veränderungen der Modellierung sind prinzipiell nicht nötig, lediglich die Funktionalität der von `ComparableContent` geerbten Vergleichsmethoden muss dahingehend angepasst werden, dass nun die Entfernung zur Firmenzentrale beider `Scooter`-Objekte verglichen wird.

Die Speicherung verschiedener Scooter mit gleicher Entfernung ist in dieser Modellierung problematisch, da der Suchbaum nur Objekte mit verschiedenen Suchschlüsseln aufnehmen kann. Wenn also in den von der Klasse `ComparableContent` geerbten Vergleichsmethoden nur die Werte des Attributs `entfernung` beider `Scooter`-Objekte verglichen werden, so könnten keine zwei Scooter mit gleicher Entfernung im Baum gespeichert werden.

Für entliehene Scooter ist der Modellierungsvorschlag ungeeignet, da die Scooter in Bewegung sind und ihre aktuelle Position im Minutentakt an die Software senden. Deshalb müsste das Attribut `entfernung` aller entliehenen `Scooter`-Objekte im Minutentakt verändert werden. Damit einhergehend müssten dann aber auch alle diese `Scooter`-Objekte in dem binären Suchbaum umsortiert werden.

Für nicht entliehene Scooter entfällt das vorherige Argument, allerdings ist der Modellierungsvorschlag für den dargestellten Anwendungsfall ebenfalls nicht geeignet. Dies liegt daran, dass in der Regel nicht nach einem konkreten Suchschlüssel (also einer konkreten Entfernung) gesucht wird, sondern z. B. nach „der kleinsten Entfernung“ oder nach „einer Entfernung größer als 10 000 Meter“. Für solche Suchanfragen bietet der binäre Suchbaum allerdings keine Vorteile.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK ²	ZK	DK
1	stellt den Suchbaum grafisch dar, der sich nach dem Einfügen ergibt.	4			
2	stellt den Suchbaum grafisch dar, der sich nach dem Löschen ergibt.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (7)					
	Summe Teilaufgabe a)	7			

Teilaufgabe b)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	erläutert die Beziehungen zwischen den angegebenen Klassen und dem Interface.	5			
2	begründet im Sachkontext, weshalb die Verwaltung der Scooter in Form eines binären Suchbaums effizienter ist als die Verwaltung in einer linearen Liste, wohingegen die Verwaltung der Kundinnen und Kunden in einer linearen Liste vertretbar ist.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (8)					
	Summe Teilaufgabe b)	8			

² EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	analysiert den Quelltext der Methode, indem er sie auf die Beispieldaten anwendet und die so erzeugte Rückgabe-Liste ermittelt.	5			
2	erläutert die Funktionsweise der Methode.	3			
3	erläutert im Sachkontext, welche Information die Methode liefert.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (12)					
	Summe Teilaufgabe c)	12			

Teilaufgabe d)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	entwickelt eine Strategie für einen Algorithmus der Methode.	5			
2	implementiert die Methode entsprechend der Dokumentation.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (11)					
	Summe Teilaufgabe d)	11			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	erläutert, welche Veränderungen in der Modellierung vorgenommen werden müssen.	3			
2	begründet, warum der Vorschlag problematisch ist, wenn verschiedene Scooter die gleiche Entfernung zur Firmenzentrale haben.	3			
3	beurteilt mit Bezug auf die beiden Anwendungsbeispiele, inwiefern der Modellierungsvorschlag des Praktikanten sinnvoll ist.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (12)					
	Summe Teilaufgabe e)	12			
	Summe insgesamt	50			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	50			
Punktzahl der gesamten Prüfungsleistung	150			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOST				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOST

Die Klausur wird abschließend mit der Note _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	Punkte	Erreichte Punktzahl
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0



Name: _____

Abiturprüfung 2021

Informatik, Leistungskurs

Aufgabenstellung:

Eine Firma möchte eine App entwickeln, die das Erstellen und Verwalten von mehreren Einkaufslisten erlaubt.

Jede Benutzerin und jeder Benutzer muss sich mit dem eigenen Namen, einem eindeutigen Benutzernamen und einem Passwort registrieren. Eine Einkaufsliste kann auch von mehreren Personen gemeinsam verwaltet werden.

Um das Erstellen einer Einkaufsliste zu erleichtern, hat das System bereits viele Produkte gespeichert, aus denen man dann auswählen kann. Natürlich können Benutzerinnen und Benutzer auch neue Produkte hinzufügen.

Die Firma verwaltet ihre Daten in einer relationalen Datenbank auf einem Datenbank-Server im Internet, der mit dem Benutzernamen und dem Passwort zugänglich ist. Eine Teilmodellierung dieser Datenbank ist im Folgenden dargestellt.

Abbildung 1:
Beispielansicht für eine
Einkaufsliste mit dem
Titel Wochenmarkt

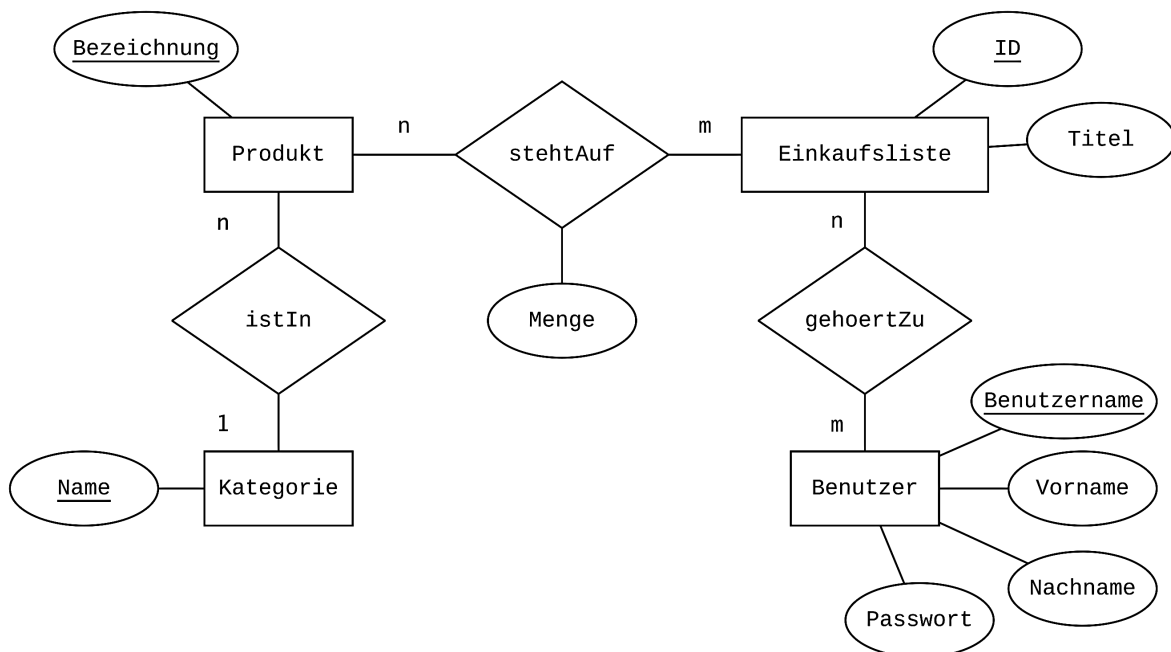


Abbildung 2: Datenbankentwurf zur Einkaufslisten-App



Name: _____

Die Modellierung der Datenbank entspricht dem folgenden Datenbankschema. Beispieldaten-sätze zu diesem Schema sind im Anhang zu finden.

Benutzer (Benutzername, Vorname, Nachname, Passwort) Einkaufsliste (<u>ID</u> , Titel) Produkt (<u>Bezeichnung</u> , ↑KategorieName) Kategorie (<u>Name</u>) stehtAuf (↑ <u>ProduktBezeichnung</u> , ↑ <u>EinkaufslisteID</u> , Menge) gehörtZu (↑ <u>EinkaufslisteID</u> , ↑ <u>Benutzername</u>)
--

Abbildung 3: Datenbankschema zur Einkaufslisten-App

a) *Erläutern Sie den Beziehungstyp stehtAuf im Sachzusammenhang.*

Begründen Sie, warum es sinnvoller ist, einen Entitätstyp Kategorie zu modellieren, statt nur ein Attribut mit dem Bezeichner Kategorie des Entitätstyps Produkt zu modellieren, wenn eine Benutzerin oder ein Benutzer neue Produkte hinzufügen kann.

(8 Punkte)

b) Gesucht sind die folgenden Informationen aus der Datenbank, die dem Schema in Abbildung 3 entspricht:

- Es werden die Bezeichnungen der Produkte gesucht, deren Bezeichnung mit der Zeichenkette 'Bio' beginnt, aufsteigend sortiert nach der Kategorie und anschließend nach der Bezeichnung.
- Um die Datenbank kleinzuhalten, werden die Produktbezeichnungen und die Kategoriennamen der Produkte gesucht, die auf keiner Einkaufsliste enthalten sind. Dabei soll jedes Produkt nur einmal aufgeführt werden.

Entwerfen sie für die obigen Anfragen jeweils eine SQL-Anweisung.

(8 Punkte)



Name: _____

c) Im Folgenden sind die SQL-Anweisungen i. und ii. gegeben:

- i. 1 SELECT Produkt.Bezeichnung, SUM(stehtAuf.Menge) AS Menge
2 FROM Produkt
3 LEFT JOIN stehtAuf
4 ON Produkt.Bezeichnung = stehtAuf.ProduktBezeichnung
5 WHERE Produkt.KategorieName = 'Obst/Gemüse'
6 GROUP BY Produkt.Bezeichnung

Diese SQL-Anweisung liefert auf Basis der Beispieldaten (siehe Anlage) das folgende Ergebnis:

Bezeichnung	Menge
Biomango	NULL
Gurke	2
Paprika	2
Salat	1

Abbildung 4: Ergebnis der SQL-Anweisung i.

- ii. 1 SELECT Produkt.Bezeichnung, SUM(stehtAuf.Menge) AS Menge
2 FROM Produkt
3 INNER JOIN stehtAuf
4 ON Produkt.Bezeichnung = stehtAuf.ProduktBezeichnung
5 WHERE Produkt.KategorieName = 'Obst/Gemüse'
6 GROUP BY Produkt.Bezeichnung
7 UNION
8 SELECT Produkt.Bezeichnung, 0 AS Menge
9 FROM Produkt
10 WHERE Produkt.Bezeichnung NOT IN (
11 SELECT stehtAuf.ProduktBezeichnung
12 FROM stehtAuf
13)
14 AND Produkt.KategorieName = 'Obst/Gemüse'

Analysieren Sie die zweite SQL-Anweisung (ii), indem Sie sie auf die in der Anlage angegebenen Beispieldaten anwenden und die Ergebnisse angeben.

Vergleichen Sie die Ergebnisse der SQL-Anweisungen und die Struktur der SQL-Anweisungen.

Hinweis: Ein SQL-Konstrukt der Form 0 AS spaltenbezeichner füllt jeden Datensatz in der Spalte spaltenbezeichner mit dem Wert 0.

(13 Punkte)



Name: _____

Für die Endgeräte der Benutzerinnen und Benutzer soll eine Software entwickelt werden. Sie lädt die Daten der angemeldeten Benutzerin bzw. des angemeldeten Benutzers aus einer Datenbank. Die App stellt die Daten für den Einkauf im Geschäft angemessen dar. Für die Software wurde folgendes Teilmodell entwickelt:

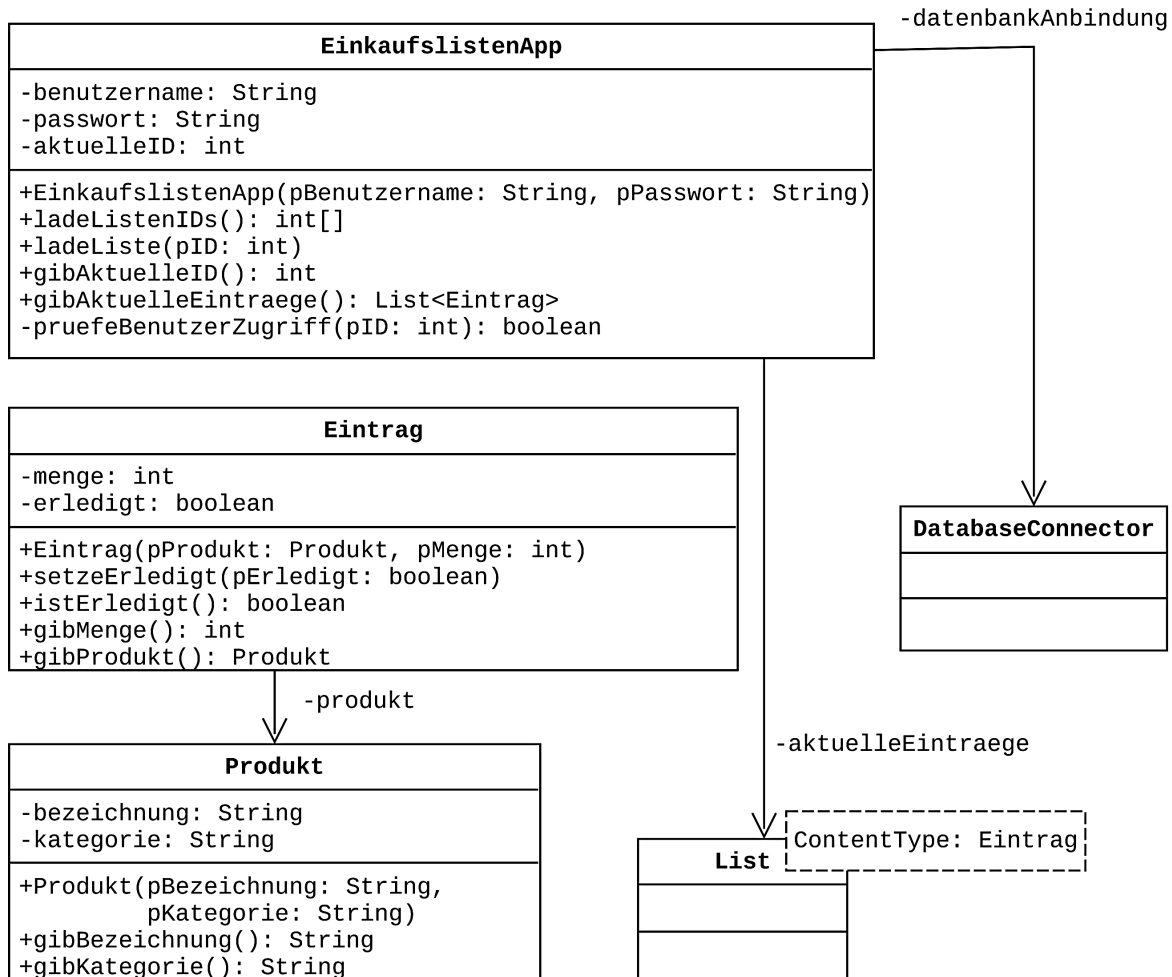


Abbildung 5: Teilmodellierung in Form eines Implementationsdiagramms

d) Erläutern Sie die in Abbildung 5 dargestellten Assoziationen im Sachkontext.

Vergleichen Sie das Implementationsdiagramm aus Abbildung 5 mit dem Entity-Relationship-Diagramm aus Abbildung 2 und erläutern Sie die Gemeinsamkeiten und die Unterschiede in Bezug auf die in der App und die in der Datenbank modellierten Daten.

Hinweis: Nutzen Sie dazu auch die Dokumentationen im Anhang.

(14 Punkte)



Name: _____

- e) Bei einer Sicherheitsüberprüfung stellt das Unternehmen fest, dass ein Benutzer alle Einkaufslisten der gesamten Datenbank heruntergeladen hat. Ein Praktikant behauptet nun, das liege daran, dass das Laden einer fremden Einkaufsliste, auf die man keinen Zugriff haben sollte, nur von der App auf dem Endgerät verhindert wird. Besser sei es, wenn man die Leserechte einer Benutzerin bzw. eines Benutzers serverseitig einschränkt.

Beurteilen Sie, welche Daten aus der modellierten Datenbank (siehe Abbildung 2 bzw. Abbildung 3) personenbezogene Daten und deshalb schützenswert sind.

Beurteilen Sie, ob die serverseitige Zugriffs-Überprüfung sinnvoller ist als die clientseitige Überprüfung.

(7 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung



Name: _____

Anlage:

Einkaufsliste	
ID	Titel
14	Salatliste
15	Biomarkt
16	Hamstern
17	Einkaufen
18	Geburtstagsfeier
19	Biomarkt

Kategorie
Name
Backwaren
Getränke
Hygieneartikel
Kühlprodukte
Obst/Gemüse
Süßspeisen
Tiefkühlprodukte

Produkt	
Bezeichnung	Kategorie
Schafskäse	Kühlprodukte
Gouda	Kühlprodukte
Hefe	Kühlprodukte
Brot	Backwaren
Baguette	Backwaren
Paprika	Obst/Gemüse
Gurke	Obst/Gemüse
Salat	Obst/Gemüse
Salamipizza	Tiefkühlprodukte
Erbsen	Tiefkühlprodukte
Schokolade	Süßspeisen
Bonbons	Süßspeisen
Limonade	Getränke
Klopapier	Hygieneartikel
Biomango	Obst/Gemüse

Benutzer			
Benutzername	Vorname	Nachname	Passwort ¹
FabBen	Ben	Lovelace	ab4bd73e2123ddef3
JaStIn	Jana	Strehl	3e2123ddef3ab4bd7
Edelweiss	Julian	Beton	ab4bd7ab4bd7eefde
Pudelski	Laura	Pudelski	bd7ab4bd7eef00e0e
Linus	Knut	Bremhorst	ef0023ddef3ab47ea

¹ Die Passwörter werden nicht im Klartext, sondern verschlüsselt in der Datenbank gespeichert.



Name: _____

gehört zu	
<u>EinkaufslisteID</u>	<u>Benutzername</u>
14	Linus
15	FabBen
16	FabBen
16	Linus
17	Linus
18	FabBen
19	JaStIn

steht auf		
<u>Produktbezeichnung</u>	<u>EinkaufslisteID</u>	<u>Menge</u>
Schafskäse	14	2
Schokolade	15	3
Salat	14	1
Paprika	14	1
Limonade	18	24
Bonbons	18	3
Brot	16	5
Gouda	15	1
Brot	18	3
Gurke	17	2
Paprika	17	1
Baguette	18	6
Salamipizza	17	3
Schafskäse	15	2
Schokolade	18	1
Erbsen	18	1
Schokolade	17	2
Hefe	17	1
Hefe	16	12
Kloppapier	16	25



Name: _____

Anhang:

Die Klasse **EinkaufslistenApp**

Ein Objekt der Klasse **EinkaufslistenApp** verwaltet eine aus der passenden Datenbank geladene Einkaufsliste.

Ausschnitt aus der Dokumentation der Klasse **EinkaufslistenApp**

EinkaufslistenApp(String pBenutzername, String pPasswort)

Ein neues Objekt der Klasse wird mit den Parametern `pBenutzername` und `pPasswort` initialisiert.

int[] ladeListenIDs()

Die Methode lädt die IDs der Einkaufslisten aus der Datenbank, die der angemeldeten Person zugeordnet sind, und gibt sie zurück.

void ladeListe(int pID)

Die Methode lädt die Einträge der mit der ID identifizierten Einkaufsliste aus der Datenbank und speichert sie. Existiert die ID nicht, ist keine Benutzerin bzw. kein Benutzer angemeldet oder gehört die Einkaufsliste mit der übergebenen ID nicht zu dem Benutzernamen, so wird keine Einkaufsliste gespeichert.

int gibAktuelleID()

Die Methode liefert die ID zu der aktuell geladenen Einkaufsliste. Ist keine Einkaufsliste geladen, wird -1 zurückgegeben.

List<Eintrag> gibAktuelleEintraege()

Die Methode liefert die Einträge der aktuellen Einkaufsliste in einem Objekt der Klasse `Liste<Eintrag>`. Ist keine aktuelle Einkaufsliste geladen, so wird `null` zurückgegeben.

Dokumentation der privaten Methode **pruefeBenutzerZugriff**

boolean pruefeBenutzerZugriff(int pID)

Die Methode prüft anhand eines Datenbankzugriffs, ob die über den Parameter identifizierte Einkaufsliste zu dem angemeldeten Benutzer gehört. In diesem Fall liefert sie den Wert `true` als Rückgabe, in allen anderen Fällen liefert sie den Wert `false` als Rückgabe.



Name: _____

Die Klasse **Eintrag**

Ein Objekt der Klasse **Eintrag** verwaltet ein aus der passenden Datenbank geladenes Produkt sowie die für die zugehörige Einkaufsliste gespeicherte Menge.

Ausschnitt aus der Dokumentation der Klasse **Eintrag**

Eintrag(Produkt pProdukt, int pMenge)

Ein neues Objekt der Klasse wird mit den Parametern pProdukt und pMenge initialisiert.

void setzeErledigt(boolean pErledigt)

Die Methode speichert, ob der Eintrag erledigt ist.

boolean istErledigt()

Die Methode liefert, ob der Eintrag erledigt ist.

int gibMenge()

Die Methode liefert die gespeicherte Menge des Eintrags.

Produkt gibProdukt()

Die Methode liefert das Produkt zu dem Eintrag.

Die Klasse **Produkt**

Ein Objekt der Klasse **Produkt** repräsentiert ein aus der passenden Datenbank geladenes Produkt.

Ausschnitt aus der Dokumentation der Klasse **Produkt**

Produkt(String pBezeichnung, String pKategorie)

Ein neues Objekt der Klasse wird mit den Parametern pBezeichnung und pKategorie initialisiert.

String gibBezeichnung()

Die Methode liefert die gespeicherte Bezeichnung des Produkts.

String gibKategorie()

Die Methode liefert die Kategorie des Produkts.



Name: _____

Die generische Klasse `List`

Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste kann durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

`List()`

Eine leere Liste wird erzeugt.

`boolean isEmpty()`

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

`boolean hasAccess()`

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

`void next()`

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

`void toFirst()`

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`void toLast()`

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.



Name: _____

ContentType getContent()

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.

void setContent(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

void append(ContentType pContent)

Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess() == false`). Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

void insert(ContentType pContent)

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pContent == null` ist, bleibt die Liste unverändert.

void concat(List<ContentType> pList)

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList == null` oder eine leere Liste ist, bleibt die Liste unverändert.

void remove()

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.



Name: _____

Die Klasse DatabaseConnector

Ein Objekt der Klasse DatabaseConnector ermöglicht die Abfrage und Manipulation einer relationalen Datenbank. Beim Erzeugen des Objekts wird eine Datenbankverbindung aufgebaut, sodass anschließend SQL-Anweisungen an diese Datenbank gerichtet werden können.

Dokumentation der Klasse DatabaseConnector

**DatabaseConnector(String pIP, int pPort, String pDatabase,
String pUsername, String pPassword)**

Ein Objekt vom Typ DatabaseConnector wird erstellt, und eine Verbindung zur Datenbank wird aufgebaut. Mit den Parametern pIP und pPort werden die IP-Adresse und die Port-Nummer übergeben, unter denen die Datenbank mit Namen pDatabase zu erreichen ist. Mit den Parametern pUsername und pPassword werden Benutzername und Passwort für die Datenbank übergeben.

void executeStatement(String pSQLStatement)

Der Auftrag schickt den im Parameter pSQLStatement enthaltenen SQL-Befehl an die Datenbank ab.

Handelt es sich bei pSQLStatement um einen SQL-Befehl, der eine Ergebnismenge liefert, so kann dieses Ergebnis anschließend mit der Methode getCurrentQueryResult abgerufen werden.

QueryResult getCurrentQueryResult()

Die Anfrage liefert das Ergebnis des letzten mit der Methode executeStatement an die Datenbank geschickten SQL-Befehls als Objekt vom Typ QueryResult zurück.

Wurde bisher kein SQL-Befehl abgeschickt oder ergab der letzte Aufruf von executeStatement keine Ergebnismenge (z. B. bei einem INSERT-Befehl oder einem Syntaxfehler), so wird null geliefert.

String getErrorMessage()

Die Anfrage liefert null oder eine Fehlermeldung, die sich jeweils auf die letzte zuvor ausgeführte Datenbankoperation bezieht.

void close()

Die Datenbankverbindung wird geschlossen.



Name: _____

Die Klasse `QueryResult`

Ein Objekt der Klasse `QueryResult` stellt die Ergebnistabelle einer Datenbankabfrage mit Hilfe der Klasse `DatabaseConnector` dar. Objekte dieser Klasse werden nur von der Klasse `DatabaseConnector` erstellt. Die Klasse verfügt über keinen öffentlichen Konstruktor.

Dokumentation der Klasse `QueryResult`

`String[][] getData()`

Die Abfrage liefert die Einträge der Ergebnistabelle als zweidimensionales Feld vom Typ `String`. Der erste Index des Feldes stellt die Zeile und der zweite die Spalte dar (d. h. `String[zeile][spalte]`).

`String[] getColumnNames()`

Die Abfrage liefert die Bezeichner der Spalten der Ergebnistabelle als Feld vom Typ `String` zurück.

`String[] getColumnTypes()`

Die Abfrage liefert die Typenbezeichnung der Spalten der Ergebnistabelle als Feld vom Typ `String` zurück. Die Bezeichnungen entsprechen den Angaben in der Datenbank.

`int getRowCount()`

Die Abfrage liefert die Anzahl der Zeilen der Ergebnistabelle als `int`.

`int getColumnCount()`

Die Abfrage liefert die Anzahl der Spalten der Ergebnistabelle als `int`.

Unterlagen für die Lehrkraft

Abiturprüfung 2021

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation kontextbezogener Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung sowie Informatik, Mensch und Gesellschaft

2. Aufgabenstellung¹

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2021 (Stand: August 2020)

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf. Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. Inhaltsfelder und inhaltliche Schwerpunkte

Daten und ihre Strukturierung

- Objekte und Klassen
 - Entwurfsdiagramme und Implementationsdiagramme
 - Lineare Strukturen
- Lineare Liste (Klasse List)*

- Datenbanken
 - Klassen DatabaseConnector, QueryResult

Formale Sprachen und Automaten

- Syntax und Semantik einer Programmiersprache
 - SQL

Informatik, Mensch und Gesellschaft

- Wirkungen der Automatisierung
 - Grundprinzipien des Datenschutzes

2. Medien/Materialien

- entfällt

¹ Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Der Beziehungstyp `stehtAuf` setzt Entitäten der Entitätsmengen `Produkt` und `Einkaufsliste` miteinander in Beziehung. Es handelt sich um einen n:m-Beziehungstyp. Dies bedeutet, ein Produkt kann auf mehreren Einkaufslisten stehen und auf einer Einkaufsliste können mehrere Produkte stehen. Für jede Einkaufsliste kann individuell durch das Attribut `Menge` festgehalten werden, welche Menge des angegebenen Produkts auf der Einkaufsliste stehen soll.

Die Modellierung eines Entitätstyps `Kategorie` stützt die Konsistenz der Bezeichnung einer Produkt-Kategorie, wenn es für eine Benutzerin bzw. einen Benutzer möglich ist, Produkte der Datenbank hinzuzufügen. Soll beispielsweise ein Produkt mit der Bezeichnung `'tiefgefrorene Seelachsfilets'` hinzugefügt werden, so könnten bei einer freien Bezeichnung der Kategorie fälschlicherweise die Zeichenketten `'Tiefkühl'`, `'Tiefkühl-Produkte'` oder `'TK-Produkte'` eingetragen werden. Dies führt zu einer Inkonsistenz bei den Kategorien. Das Modellieren eines Entitätstyps für die Kategorie eines Produkts ermöglicht vorgegebene Kategorien mit einheitlichem Bezeichner.

Teilaufgabe b)

- i.

```
SELECT Produkt.Bezeichnung
FROM Produkt
WHERE Produkt.Bezeichnung LIKE 'Bio%'
ORDER BY Produkt.KategorieName ASC, Produkt.Bezeichnung ASC
```
- ii.

```
SELECT Produkt.Bezeichnung, Produkt.KategorieName
FROM Produkt
WHERE Produkt.Bezeichnung NOT IN (
    SELECT stehtAuf.ProduktBezeichnung
    FROM stehtAuf
)
```

Teilaufgabe c)

Ergebnis der SQL-Anweisung ii.

Bezeichnung	Menge
Biomango	0
Gurke	2
Paprika	2
Salat	1

Im Ergebnis unterscheiden sich die beiden Anweisungen nur in dem Wert der Gesamtmenge für die Produkte, die auf keiner Einkaufsliste enthalten sind: In der ersten Anweisung ist hier NULL das Ergebnis, während im Ergebnis der zweiten Anweisung der Wert 0 steht.

In der ersten Anweisung werden aufgrund des verwendeten LEFT JOINs sämtliche Produkte aufgeführt, die in der Relation Produkt enthalten sind. Durch den LEFT JOIN werden diese Einträge mit den enthaltenen Produkten und den zugehörigen Mengen der Relation stehtAuf verknüpft. Durch die Gruppierung ist das Ermitteln der Gesamtmenge jedes Produkts möglich. Ist ein Produkt in der Relation stehtAuf nicht enthalten, so ist sowohl vor der Summierung als auch nach der Summierung das zugehörige Datum im Attribut Menge NULL.

In der zweiten Anweisung hingegen wird statt des LEFT JOINs zunächst ein INNER JOIN verwendet, weshalb nur die Produkte enthalten sind, die in beiden Relationen mindestens einmal enthalten sind. Durch die Gruppierung und Summierung wird hier ebenfalls die Gesamtmenge jedes Produkts ermittelt. Damit die Produkte, die auf keiner Einkaufsliste enthalten sind, auch aufgeführt werden, wird die Vereinigung (UNION) mit dem Ergebnis des zweiten Teils der Anweisung verwendet. In diesem Teil werden zunächst in der Unteranweisung alle Produkte herausgefiltert, die auf mindestens einer Einkaufsliste stehen. Von allen Produkten werden nur die übernommen, die in der Unteranweisung nicht enthalten sind. Durch die Ergänzung des Datums 0 für alle Datensätze ist die Vereinigung mit dem Ergebnis des ersten Teils möglich.

Teilaufgabe d)

Die Assoziation mit dem Bezeichner produkt modelliert den Sachverhalt, dass die Einträge in einer Einkaufsliste sich auf Produkte beziehen.

Die Assoziation aktuelleEintraege modelliert die Sammlung der Einträge zur aktuell geladenen Einkaufsliste. Als Datensammlung für Objekte der Klasse Eintrag wird ein Objekt der Klasse List verwendet.

Die Assoziation datenbankAnbindung modelliert den technischen Zugriff zur SQL-Datenbank über die vorgegebene Schnittstelle.

Vergleicht man das Datenbankschema mit dem Implementationsdiagramm, ergeben sich deutliche Unterschiede in der Verwaltung der Daten.

Während in der Datenbank Vorname, Nachname, Benutzername und Passwort aller Benutzerinnen und Benutzer im Entitätstyp `Benutzer` modelliert werden, werden in der Klasse `EinkaufslistenApp` nur der Benutzername und das Passwort der aktuell angemeldeten Person verwaltet.

In der Datenbank werden sämtliche Einkaufslisten aller Benutzerinnen und Benutzer mit dem Entitätstyp `Einkaufsliste` und den zugehörigen Attributen `ID` und `Titel` sowie die jeweils enthaltenen Produkte über den Beziehungstyp `stehtAuf` mit dem Attribut `Menge` modelliert.

Ein Objekt der Klasse `EinkaufslistenApp` speichert nur eine Auswahl dieser beschriebenen Daten aus der Datenbank: Es verwaltet die Produkte nur genau einer Einkaufsliste. Diese Produkte der aktuell geladenen Einkaufsliste werden jeweils über ein Objekt der Klasse `Eintrag` verwaltet. Dieses Objekt verwaltet die in der Datenbank gespeicherte Menge eines Produkts aus dem Beziehungstyp `stehtAuf`. Die Klasse `Eintrag` bietet zusätzlich zu den Daten aus der Datenbank für jedes Produkt die Möglichkeit der Auswahl, ob ein Produkt als „erledigt“ zu markieren ist.

Bei der Software auf den Endgeräten werden also nur die für die Benutzerin bzw. den Benutzer für die aktuell geladene Einkaufsliste notwendigen Daten verwaltet, ergänzt um eine Kennzeichnung, dass ein Produkt „erledigt“ ist.

Teilaufgabe e)

Personenbezogene Daten in der Datenbank sind sämtliche Daten, die auf eine Person bezogen sind oder auf eine Person bezogen werden können. In dem Datenbankmodell sind nur die Produkte, die Kategorien und deren Beziehung untereinander nicht personenbezogen bzw. personenbeziehbar. Alle anderen Daten, die in den Relationsschemata `stehtAuf`, `Einkaufsliste`, `Benutzer` und `gehörtZu` modelliert werden, können einem oder wenigen Benutzernamen und damit Personen zugeordnet werden.

Da die Datenbank von überall aus dem Internet zu erreichen ist, kann nicht gewährleistet werden, dass die Datenbank nicht mit einer manipulierten Einkaufslisten-App oder ganz ohne Client abgefragt wird. Wird die Zugriffsbeschränkung nur durch die App realisiert, können in dem Fall alle Daten uneingeschränkt mit einem beliebigen Benutzerkonto abgefragt werden. Eine serverseitige Zugriffsbeschränkung würde auch in diesem Fall Sicherheit gewährleisten.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK ²	ZK	DK
1	erläutert den Beziehungstyp im Sachzusammenhang.	4			
2	begründet, warum es sinnvoller ist, einen Entitätstyp Kategorie zu modellieren, statt nur ein Attribut mit dem Bezeichner Kategorie des Entitätstyps Produkt zu modellieren, wenn eine Benutzerin oder ein Benutzer neue Produkte hinzufügen kann.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (8)					
.....					
.....					
	Summe Teilaufgabe a)	8			

Teilaufgabe b)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	entwirft die erste SQL-Anweisung.	4			
2	entwirft die zweite SQL-Anweisung.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (8)					
.....					
.....					
	Summe Teilaufgabe b)	8			

² EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	analysiert die zweite SQL-Anweisung, wendet sie auf die in der Anlage angegebenen Beispieldaten an und gibt die Ergebnisse an.	5			
2	vergleicht die Ergebnisse der SQL-Anweisungen und die Struktur der SQL-Anweisungen.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (13)					
	Summe Teilaufgabe c)	13			

Teilaufgabe d)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	erläutert die im Diagramm dargestellten Assoziationen im Sachkontext.	4			
2	vergleicht das Implementationsdiagramm mit dem Entity-Relationship-Diagramm und erläutert die Gemeinsamkeiten und die Unterschiede in Bezug auf die Benutzerinnen und Benutzer.	2			
3	vergleicht das Implementationsdiagramm mit dem Entity-Relationship-Diagramm und erläutert die Gemeinsamkeiten und die Unterschiede in Bezug auf die Einkaufslisten.	4			
4	vergleicht das Implementationsdiagramm mit dem Entity-Relationship-Diagramm und erläutert die Gemeinsamkeiten und die Unterschiede in Bezug auf die Produkte.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (14)					
	Summe Teilaufgabe d)	14			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	beurteilt, welche Daten aus der modellierten Datenbank personenbezogene Daten und deshalb schützenswert sind.	3			
2	beurteilt, ob die serverseitige Zugriffs-Überprüfung sinnvoller ist als die clientseitige Überprüfung.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (7)					
	Summe Teilaufgabe e)	7			
	Summe insgesamt	50			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	50			
Punktzahl der gesamten Prüfungsleistung	150			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOST				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOST

Die Klausur wird abschließend mit der Note _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	Punkte	Erreichte Punktzahl
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0



Name: _____

Abiturprüfung 2021

Informatik, Leistungskurs

Aufgabenstellung:

Leonard ist Schüler eines Informatikkurses und möchte nach seinem Abitur ein Informatikstudium beginnen. Als seine Eltern ihn damit beauftragen, seine Spielsachen vom Dachboden zur sortieren, findet er seine Eisenbahn, mit der er als Kind viel gespielt hat. Er erinnert sich daran, dass er immer versucht hat, eine sinnvolle Schienenanlage aufzubauen, bei der seine Eisenbahn entweder im Kreis fuhr oder die Anlage durch Prellböcke beendet wurde. Leonard möchte mit seinen Kenntnissen aus der theoretischen Informatik analysieren, ob eine Schienenanlage sinnvoll aufgebaut ist. Damit seine Analyse nicht zu komplex wird, beschränkt er sich zunächst auf die folgenden Bauteile.

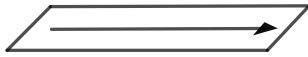
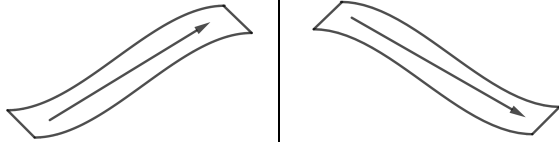
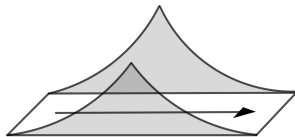
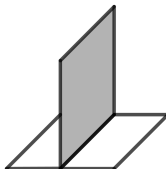
Bauteil (mit Bauteil-ID)	Skizze
gerades Gleisstück (g)	
Brückenauffahrt (1) und Brückenabfahrt (2)	
Brückenelement (b)	
Prellbock (p), d. h. Anfang oder Ende einer Schienenanlage, wenn diese keinen Kreis bildet.	

Abbildung 1: Bauteile für die Schienenanlage



Name: _____

Mithilfe eines endlichen Automaten soll geprüft werden, ob eine Schienenanlage sinnvoll aufgebaut ist.

Dazu setzt er die Bauteil-IDs zu Wörtern zusammen, welche die Schienenanlagen darstellen. So wird beispielsweise eine Schienenanlage, die aus zwei Prellböcken und einer dazwischenliegenden Brücke besteht, durch das Wort p1b2p dargestellt.

Leonard entwickelt das in Abbildung 2 dargestellte Zustandsübergangsdiagramm eines deterministischen endlichen Automaten A zum Überprüfen von Schienenanlagen:

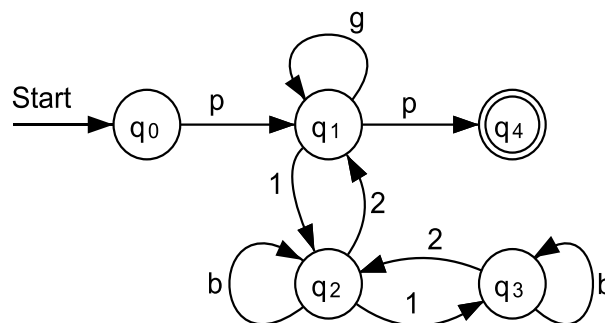


Abbildung 2: Zustandsübergangsdiagramm zum endlichen Automaten A

- a) Erläutern Sie, welche Eigenschaften die Schienenanlagen besitzen, die durch Wörter dargestellt werden, welche vom Automaten A akzeptiert werden.

Ermitteln Sie alle Wörter mit der kürzesten möglichen Länge, die jede Bauteil-ID mindestens einmal enthalten und die vom Automaten A akzeptiert werden.

Erläutern Sie die Bedeutung des Zustands q_1 im Sachzusammenhang.

(9 Punkte)

- b) Das Wort pp gehört zur Sprache L_A des Automaten A (vgl. Abbildung 2). Die zum Wort pp gehörige Schienenanlage ist aber nicht sinnvoll.

Begründen Sie im Sachzusammenhang, warum die Schienenanlage zum Wort pp nicht sinnvoll ist.

Modifizieren Sie das Zustandsübergangsdiagramm in Abbildung 2 so, dass der dazugehörige deterministische endliche Automat alle Wörter der Sprache L_A mit Ausnahme des Wortes pp akzeptiert.

(8 Punkte)



Name: _____

Im Folgenden werden keine Brücken und Prellböcke mehr verbaut, dafür aber zusätzlich 90°-Kurvenelemente für den Bau der Schienenanlagen verwendet. Leonard möchte allerdings nur Rechtskurven einbauen:

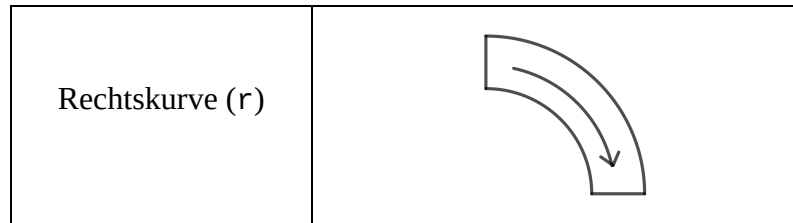


Abbildung 3: Rechtskurven

c) Die folgende Grammatik G erzeugt die Sprache L_G :

Startsymbol: S
Terminalsymbole: $\{g, r\}$
Nichtterminalsymbole: $\{S, A, B\}$
Produktionen: $\left\{ \begin{array}{l} S \rightarrow rrA, \\ A \rightarrow gBg, \\ B \rightarrow gBg \mid rr \end{array} \right\}$

Zeigen Sie, dass sich das Wort $rrggrrgg$ aus der Grammatik G ableiten lässt, das Wort $rrrr$ allerdings nicht.

Erläutern Sie im Sachzusammenhang, welche Schienenanlagen durch die Sprache L_G dargestellt werden.

Begründen Sie, warum die Grammatik G nicht regulär ist.

Entwickeln Sie einen Kellerautomaten für die Sprache L_G .

(16 Punkte)



Name: _____

- d) Leonard möchte mithilfe eines Computerprogramms prüfen, ob die vorhandenen Bauteile ausreichen, um eine bestimmte Schienenanlage zu bauen. Er möchte für diese Schienenanlagen nur Prellböcke (p), Rechtskurven (r) und gerade Gleisstücke (g) verwenden.

Für die Klasse `Schienenanlagentester` schreibt er eine Methode `genugBauteile`, die als Parameter zwei Zeichenketten, bestehend aus den Zeichen p, r und g, erhält.

Die Methode hat den folgenden Methodenkopf:

```
public boolean genugBauteile (String pAnlage, String pBauteile)
```

- Der Parameter `pAnlage` stellt das Wort der zu bauenden Schienenanlage dar.
- Der Parameter `pBauteile` ist eine Zeichenkette, in der die IDs aller vorhandenen Bauteile in unsortierter Reihenfolge einfach hintereinandergeschrieben werden.
- Die Methode muss NICHT prüfen, ob das Wort für die zu bauende Schienenanlage eine sinnvolle Schienenanlage darstellt.
- Sie soll `true` zurückgeben, wenn die vorhandenen Bauteile für die Schienenanlage ausreichen, ansonsten soll sie `false` zurückgeben.

Entwickeln Sie ein algorithmisches Verfahren für die Methode `genugBauteile` der Klasse `Schienenanlagentester`.

Implementieren Sie die Methode `genugBauteile` der Klasse `Schienenanlagentester`.

(12 Punkte)

- e) Leonard möchte mithilfe eines deterministischen endlichen Automaten prüfen, ob ein Wort zu einer Schienenanlage gehört, die ein „Quadrat“ bildet. Mit einem „Quadrat“ ist eine Schienenanlage gemeint, die aus vier Rechtskurven an den vier Ecken besteht. Zwischen den Rechtskurven sind für jede der vier Seiten gleich viele gerade Gleisstücke verbaut. Leonard möchte beliebig große „Quadrate“ zulassen.

Beurteilen Sie, ob es möglich ist, mithilfe eines endlichen Automaten zu entscheiden, ob ein Wort zu einer Schienenanlage gehört, die ein „Quadrat“ darstellt.

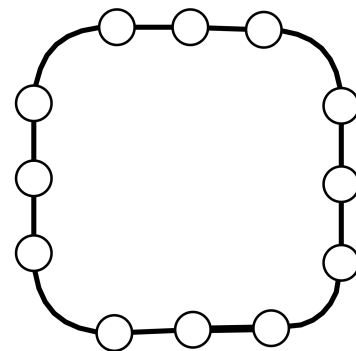


Abbildung 4: „Quadrat“ mit zwei geraden Gleisstücken an jeder Seite

(5 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung



Name: _____

Anhang

Dokumentation ausgewählter Methoden der Klasse String

char charAt(int index)

Die Anfrage gibt das Zeichen am angegebenen Index zurück. Der Index hat einen Wertebereich von 0 bis `length() - 1`. Das erste Zeichen dieser Zeichenkette ist an Index 0, das nächste an Index 1 und so weiter, wie bei Array-Indizes.

int compareTo(String anotherString)

Die Methode vergleicht zwei Zeichenketten lexikografisch. Sie liefert einen Integerwert kleiner als 0, wenn der String lexikografisch vor der übergebenen Zeichenkette `anotherString` anzuordnen ist. Sie liefert einen Integerwert größer als 0, wenn der String lexikografisch nach der übergebenen Zeichenkette `anotherString` anzuordnen ist. Sind beide Zeichenketten identisch, wird der Wert 0 zurückgegeben.

boolean equals(Object anObject)

Die Methode liefert genau dann `true`, wenn das übergebene Objekt `anObject` nicht `null` ist und wenn es vom Typ `String` ist und die gleiche Zeichenkette repräsentiert.

int indexOf(String str)

Die Methode liefert den Index des ersten Zeichens beim ersten Vorkommen der übergebenen Zeichenkette `str`. Kommt die übergebene Zeichenkette nicht vor, wird `-1` zurückgegeben.

int length()

Die Methode liefert die Länge der Zeichenkette.

boolean startsWith(String prefix)

Die Methode liefert genau dann den Wert `true`, wenn die Zeichenkette mit dem angegebenen Präfix `prefix` beginnt. Andernfalls liefert sie den Wert `false`. Wird eine leere Zeichenkette oder die identische Zeichenkette übergeben, liefert die Methode dementsprechend ebenfalls den Wert `true`.

String substring(int beginIndex)

Die Methode liefert einen Teil der Zeichenkette. Dieser Teil beginnt mit dem übergebenen Index `beginIndex` und reicht bis zum Ende der Zeichenkette.

Der Wert des Parameters `beginIndex` darf nicht größer als die Länge der Zeichenkette sein. Ist er gleich der Länge der Zeichenkette, so wird die leere Zeichenkette `""` zurückgegeben.

String substring(int beginIndex, int endIndex)

Die Methode liefert einen Teil der Zeichenkette. Dieser Teil beginnt mit dem übergebenen Index `beginIndex` und reicht bis zum Zeichen an Position `endIndex - 1`. Der Wert des Parameters `endIndex` darf nicht größer als die Länge des Strings sein. Der Wert des Parameters `beginIndex` muss größer oder gleich 0 sein und darf nicht größer als der Wert von `endIndex` sein.

Unterlagen für die Lehrkraft

Abiturprüfung 2021

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation von kontextbezogenen Problemstellungen mit Schwerpunkt auf dem Inhaltsfeld Formale Sprachen und Automaten

2. Aufgabenstellung¹

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2021 (Stand: August 2020)

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf. Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. Inhaltsfelder und inhaltliche Schwerpunkte

Daten und ihre Strukturierung

- Objekte und Klassen

Algorithmen

- Analyse, Entwurf und Implementierung von Algorithmen
- Algorithmen in ausgewählten informatischen Kontexten

Formale Sprachen und Automaten

- Syntax und Semantik einer Programmiersprache
 - Java
- Endliche Automaten
 - Deterministische endliche Automaten
 - Nichtdeterministische Kellerautomaten
- Grammatiken regulärer Sprachen
- Möglichkeiten und Grenzen von Automaten und formalen Sprachen

2. Medien/Materialien

- entfällt

¹ Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Schienenanlagen, die durch Wörter dargestellt werden, welche vom Automaten akzeptiert werden, besitzen folgende Eigenschaften:

- Die Schienenanlage beginnt und endet mit einem Prellbock. Weitere Prellböcke kommen in der Schienenanlage nicht vor.
- Nach dem ersten Prellbock können beliebig viele (auch null) gerade Gleise oder Brückenanlagen in beliebiger Reihenfolge folgen, bevor der zweite Prellbock die Schienenanlage beendet.
- Eine Brückenanlage besteht aus einer Brückenauffahrt und einer Brückenabfahrt. Zwischen Auffahrt und Abfahrt können beliebig viele (auch null) Brückenelemente oder weitere Brückenanlagen liegen. Diese weiteren Brückenanlagen bestehen aus einer Auffahrt, beliebig vielen (auch null) Brückenelementen und einer Abfahrt.

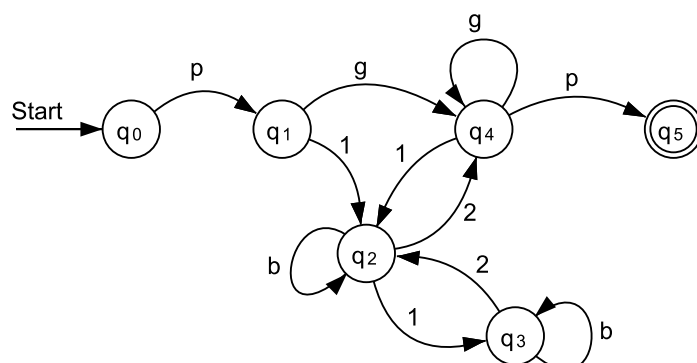
Kürzeste Wörter mit allen Bauteilen: p1b2gp oder pg1b2p

Im Zustand q_1 befindet man sich mitten in einer Schienenanlage auf der unteren Ebene. Die Schienenanlage kann entweder mit Brückenanlagen oder mit geraden Gleisstücken fortgesetzt oder durch einen Prellbock beendet werden.

Teilaufgabe b)

Auf diese Schienenanlage könnte man keine Lok und keine Wagen stellen, weil sie nur aus zwei Prellböcken besteht.

Modifiziertes Zustandsübergangsdiagramm:



Teilaufgabe c)

Ableitung des Wortes $rrgrrgg$ aus der Grammatik:

$S \rightarrow rrA$
 $\rightarrow rrgBg$
 $\rightarrow rrggBgg$
 $\rightarrow rrggrrgg$

Versuchte Ableitung des Wortes $rrrr$ aus der Grammatik:

$S \rightarrow rrA$
 $\rightarrow rr?$

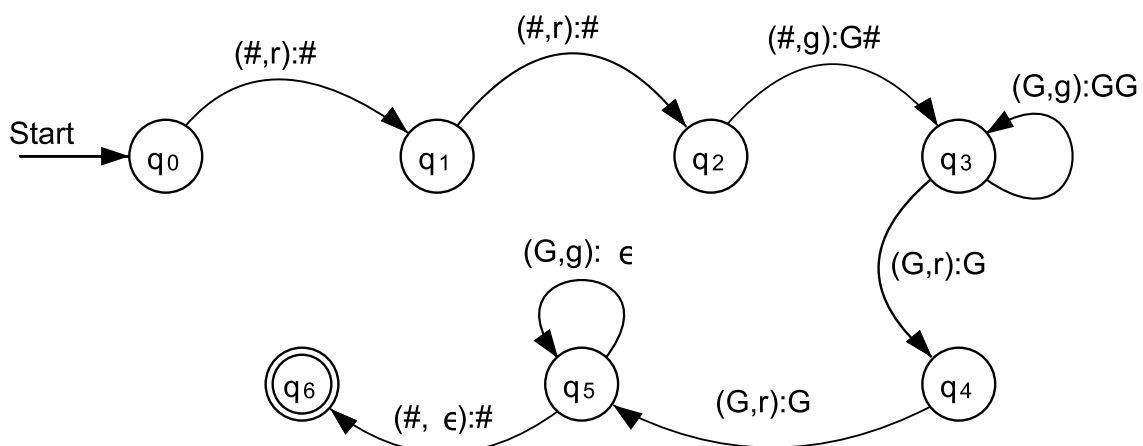
Aus dem Nichtterminal A lässt sich nur eine Zeichenfolge ableiten, die das Terminal g enthält. Da g allerdings nicht im Wort vorkommt, lässt sich das Wort $rrrr$ nicht ableiten.

Durch die Grammatik können Wörter erzeugt werden, die Schienenanlagen darstellen, die mit einem Halbkreis aus zwei Rechtskurven beginnen, dann eine Anzahl $n > 0$ gerader Gleisstücke haben, anschließend wieder einen Halbkreis aus zwei Rechtskurven, zum Schluss dann wieder dieselbe Anzahl n gerader Gleisstücke. Insgesamt haben die Schienenanlagen die Form eines Ovals, ähnlich einer Laufbahn im Stadion.

Die Grammatik G ist nicht regulär, weil z. B. auf der rechten Seite der Produktion $A \rightarrow gBg$ rechts und links vom Nichtterminal jeweils ein Terminal steht. Das widerspricht den Regeln für Produktionen rechts- oder linksregulärer Grammatiken.

Kellerautomat M für die Sprache L_G :

Zustandsmenge: $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
 Eingabealphabet: $\{r, g\}$
 Kelleralphabet: $\{G, \#\}$
 Anfangszustand: q_0
 Kellerstartsymbol: $\#$
 Menge der Endzustände: $\{q_6\}$
 Übergangsfunktion d :



Teilaufgabe d)

Algorithmisches Verfahren:

Als Hilfsmethode wird die Methode `liefereAnzahl` implementiert, die zu einem übergebenen Zeichen und einer Zeichenkette die Anzahl zurückgibt, wie oft das Zeichen in der Zeichenkette vorkommt. In dieser Methode vergleicht man in einer Zählschleife nacheinander alle Zeichen der Zeichenkette mit dem zu vergleichenden Zeichen und erhöht die Anzahl bei Gleichheit jeweils um 1.

Mithilfe dieser Hilfsmethode kann nun entschieden werden, ob die Bauteile (`pBauteile`) für den Bau der Anlage (`pAnlage`) ausreichen. Man vergleicht die Anzahl der `p` in `pAnlage` mit der Anzahl `p` in `pBauteile`. Analog die Anzahlen für die Zeichen `g` und `r`. Nur wenn für alle drei Zeichen die Anzahl in `pAnlage` maximal so groß ist wie die Anzahl in `pBauteile`, liefert die Methode `true` zurück.

Implementation des algorithmischen Verfahrens:

```
public boolean genugBauteile(String pAnlage, String pBauteile) {
    return liefereAnzahl('p', pAnlage) <= liefereAnzahl('p',
                                                            pBauteile)
        && liefereAnzahl('g', pAnlage) <= liefereAnzahl('g',
                                                            pBauteile)
        && liefereAnzahl('r', pAnlage) <= liefereAnzahl('r',
                                                            pBauteile);
}

public int liefereAnzahl(char pZeichen, String pWort) {
    int anzahl = 0;
    for (int i = 0; i < pWort.length(); i++){
        if (pWort.charAt(i) == pZeichen){
            anzahl = anzahl + 1;
        }
    }
    return anzahl;
}
```

Teilaufgabe e)

Will man einen endlichen Automaten für diese Aufgabe konstruieren, so muss dieser in den Zuständen die Anzahl der geraden Gleisstücke für die erste Seite des Quadrats speichern, damit der Automat für die zweite Seite des Quadrats nach derselben Anzahl gerader Gleisstücke die nächste Kurve erwarten kann.

Da Leonard beliebig große „Quadrate“ zulassen möchte, ist die Anzahl gerader Gleisstücke, und damit die Anzahl der Zustände nach oben unbegrenzt.

Endliche Automaten dürfen aber nur eine endliche Anzahl Zustände besitzen. Daher kann es keinen solchen endlichen Automaten geben.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK ²	ZK	DK
1	erläutert, welche Eigenschaften die Schienenanlagen besitzen, die durch Wörter dargestellt werden, welche vom Automaten A akzeptiert werden.	3			
2	ermittelt alle Wörter mit der kürzesten möglichen Länge, die jede Bauteil-ID mindestens einmal enthalten und die vom Automaten A akzeptiert werden.	2			
3	erläutert die Bedeutung des Zustands q_1 im Sachzusammenhang.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (9)					
	Summe Teilaufgabe a)	9			

Teilaufgabe b)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	begründet im Sachzusammenhang, warum die Schienenanlage zum Wort pp nicht sinnvoll ist.	2			
2	modifiziert das Zustandsübergangsdiagramm so, dass der dazugehörige deterministische Automat alle Wörter aus L_A mit Ausnahme des Wortes pp akzeptiert.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (8)					
	Summe Teilaufgabe b)	8			

² EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	zeigt, dass sich das Wort rrggrrgg aus der Grammatik ableiten lässt, das Wort rrrr allerdings nicht.	4			
2	erläutert im Sachzusammenhang, welche Schienenanlagen durch die Sprache L_G dargestellt werden.	3			
3	begründet, warum die Grammatik G nicht regulär ist.	2			
4	entwickelt einen Kellerautomaten für die Sprache L_G .	7			
Sachlich richtige Lösungsalternative zur Modelllösung: (16)					
	Summe Teilaufgabe c)	16			

Teilaufgabe d)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	entwickelt ein algorithmisches Verfahren für die Methode.	5			
2	implementiert die Methode.	7			
Sachlich richtige Lösungsalternative zur Modelllösung: (12)					
	Summe Teilaufgabe d)	12			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	maximal erreichbare Punktzahl	EK	ZK	DK
1	beurteilt, ob es möglich ist, mithilfe eines endlichen Automaten zu entscheiden, ob ein Wort zu einer Schienenanlage gehört, die ein „Quadrat“ darstellt.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (5)					
	Summe Teilaufgabe e)	5			

	Summe insgesamt	50			
--	------------------------	----	--	--	--

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	EK	ZK	DK
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	50			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	50			
Punktzahl der gesamten Prüfungsleistung	150			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOST				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOST

Die Klausur wird abschließend mit der Note _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	Punkte	Erreichte Punktzahl
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0