Bitte beachten:

Die Aufgaben wurden nur an den Schulen 183283_198493_165785_167368_167113 verwendet.

Sie dürfen nicht veröffentlicht werden!

Name: _____

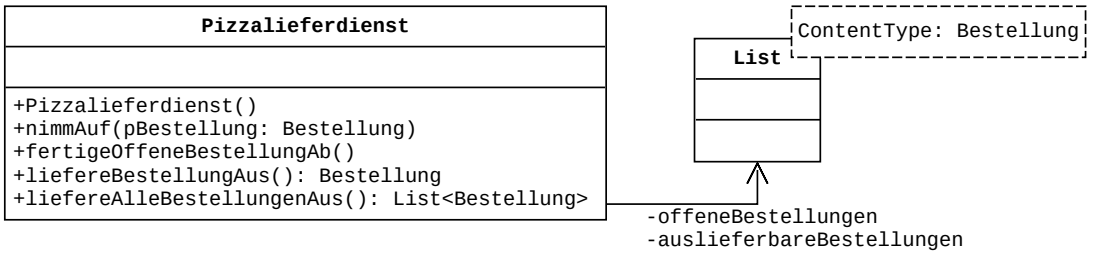
Abiturprüfung 2019

Informatik, Leistungskurs

Aufgabenstellung:

Bei dem Start-up-Unternehmen *Pizza2Call* können Kunden Pizza bestellen und sich liefern lassen. Durch ein Informatiksystem sollen Bestellungen noch effizienter verwaltet und abgearbeitet werden können.
Dafür wird jede Bestellung – nach Ermessen der Mitarbeiterin oder des Mitarbeiters – mit einer Priorität versehen und in der Liste `offeneBestellungen` verwaltet. Es stehen lediglich die Prioritäten 1 (niedrig), 2 (mittel) und 3 (hoch) zur Verfügung. Je höher die Priorität einer Bestellung ist, desto weiter vorne wird sie in die Liste eingereiht. Bei gleicher Priorität wird eine neue Bestellung hinter die bereits bestehenden Bestellungen mit dieser Priorität eingereiht.
Wenn die erste Bestellung aus der Liste `offeneBestellungen` abgefertigt bzw. zubereitet wurde, dann wird sie aus der Liste `offeneBestellungen` entfernt und an die Liste `auslieferbareBestellungen` angehängt.
In der ersten Version soll jede Bestellung nur aus jeweils einem Gericht bestehen.

Abbildung 1 zeigt einen Ausschnitt des Implementationsdiagramms des zugehörigen Informatiksystems. Eine Dokumentation relevanter Klassen finden Sie im Anhang.



Bestellung

```
-bestellungID: int
-gericht: String
-anschrift: String
-prioritaet: int
-preis: double
```

```
+Bestellung(pBestellungID: int, pGericht: String, pAnschrift: String,
           pPrioritaet: int, pPreis: double)
+gibBestellungID(): int
+gibGericht(): String
+gibAnschrift(): String
+gibPrioritaet(): int
+gibPreis(): double
+setzePrioritaet(pPrioritaet: int)
```

Abbildung 1: Ausschnitt aus dem Implementationsdiagramm

Name: _____

a) Gehen Sie davon aus, dass die Listenobjekte `offeneBestellungen` und `auslieferbareBestellungen` leer sind.

Folgende Aktionen werden nacheinander durchgeführt:

i. Eine Bestellung aufnehmen:

BestellungID	Gericht	Anschrift	Priorität	Preis in Euro
1	Pizza Salami	Schulstr. 1	2	7,00

ii. Eine Bestellung aufnehmen:

BestellungID	Gericht	Anschrift	Priorität	Preis in Euro
2	Pizza Hawaii	Schulstr. 2	3	8,00

iii. Eine Bestellung aufnehmen:

BestellungID	Gericht	Anschrift	Priorität	Preis in Euro
3	Salat	Bahnhofstr. 10	3	5,00

iv. Eine offene Bestellung abfertigen.

v. Eine Bestellung aufnehmen:

BestellungID	Gericht	Anschrift	Priorität	Preis in Euro
4	Pizzabrötchen	Schulstr. 1	1	3,00

vi. Eine Bestellung ausliefern.

vii. Eine offene Bestellung abfertigen.

viii. Eine offene Bestellung abfertigen.

ix. Alle Bestellungen ausliefern.

Dokumentieren Sie nach jeder der neun genannten Aktionen die Belegungen der Listenobjekte `offeneBestellungen` und `auslieferbareBestellungen`, *indem Sie lediglich die IDs und die Prioritäten der Bestellungen in den jeweiligen Listenobjekten angeben.*

Beschreiben Sie die Beziehungen zwischen den Klassen `Pizzalieferdienst`, `List` *und* `Bestellung`.

(8 Punkte)

Name: _____

b) Die Klasse `Pizzalieferdienst` verfügt über folgende neue Methoden:

```
 1 public void tueEtwas(List<Integer> pListe){
 2   List<Bestellung> neueListe = gibEtwas(pListe);
 3   neueListe.toFirst();
 4   while (neueListe.hasAccess()) {
 5     nimmAuf(neueListe.getContent());
 6     neueListe.next();
 7   }
 8 }
 9 public List<Bestellung> gibEtwas(List<Integer> pListe){
10   List<Bestellung> liste = new List<Bestellung>();
11   offeneBestellungen.toFirst();
12   while (offeneBestellungen.hasAccess()) {
13     Bestellung akt = offeneBestellungen.getContent();
14     pListe.toFirst();
15     boolean gefunden = false;
16     while (!gefunden && pListe.hasAccess()) {
17       if (akt.gibBestellungID() == (int)(pListe.getContent())
18           && akt.gibPrioritaet() < 3) {
19         gefunden = true;
20         akt.setzePrioritaet(akt.gibPrioritaet() + 1);
21         liste.append(akt);
22         offeneBestellungen.remove();
23       }
24       pListe.next();
25     }
26     if (!gefunden) {
27       offeneBestellungen.next();
28     }
29   }
30   return liste;
31 }
```

Gegeben ist die Liste `offeneBestellungen` mit folgenden Testdaten:

BestellungID	Gericht	Anschrift	Priorität	Preis in Euro
9	Pizza Schinken	Schulstraße 1	3	7,00
5	Pizza Funghi	Kindergartenweg 100	2	7,50
6	Pizza Hawaii	Schulstraße 3	2	8,00
3	Pizzabrötchen	Schulstraße 1	1	3,00
8	Pizza Salami	Bergweg 80	1	7,70

Abbildung 2: Testdaten für die Liste `offeneBestellungen`

Name: _____

Analysieren Sie die Methoden `tueEtwas` *und* `gibEtwas`*, indem Sie die Informationen* `BestellungID` *und* `Priorität` *der Testdaten in Abbildung 2 angeben, nachdem die Methode* `tueEtwas` *mit einer Liste bestehend aus den Zahlen* `8`, `6` *und* `9` *aufgerufen wurde.*

Erläutern Sie die Funktionsweise der Methoden und die Funktionalität der Methoden im Sachkontext.

(11 Punkte)

c) Um unnötige Fahrwege zu vermeiden, sollen die auslieferbaren Bestellungen in der Liste `auslieferbareBestellungen` nach Ihrer Anschrift aufsteigend sortiert werden.

In der Klasse `Pizzalieferdienst` wird eine neue Methode `sortiereAuslieferbareBestellungen` benötigt, die wie folgt dokumentiert ist:

> `private void sortiereAuslieferbareBestellungen()`
> Nach Methodenausführung sind die Bestellobjekte in der Liste `auslieferbareBestellungen` aufsteigend nach Ihrer Anschrift sortiert.

Entwickeln Sie eine Strategie für die Arbeitsweise der Methode `sortiereAuslieferbareBestellungen` *und stellen Sie diese in geeigneter Form dar.*

Implementieren Sie die Methode `sortiereAuslieferbareBestellungen` *entsprechend Ihrer Strategie.*

(12 Punkte)

Name: _____

d) Das Start-Up-Unternehmen *Pizza2Call* möchte die Software so verbessern, dass mit einer Bestellung auch mehrere Gerichte bestellt werden können. Außerdem sollen für spätere Werbemaßnahmen die Kundeninformationen (Vorname, Nachname und Telefonnummer) verwaltet werden.
Darüber hinaus soll für statistische Zwecke protokolliert werden, welcher Kunde welche Bestellungen getätigt hat.

In der Modellerweiterung müssen folgende Anforderungen berücksichtigt werden:

i. Mit einer Bestellung dürfen mehrere Gerichte bestellt werden.
ii. Die Daten (Vorname, Nachname und Telefonnummer) eines Kunden müssen verwaltet werden.
Alle Kunden müssen verwaltet werden.
iii. Ein neuer Kunde muss angelegt werden können.
iv. Jedem Kunden müssen alle seine Bestellungen zugeordnet werden.
v. Es sollen alle Kunden ermittelt werden, die mindestens so viele Bestellungen getätigt haben, wie dies durch einen Schwellenwert vorgegeben wird.

Modellieren Sie die oben genannten Anforderungen als Erweiterung des Implementationsdiagramms aus Abbildung 1.
Hinweis: Unveränderte Attribute, Methoden und Assoziationen aus dem Implementationsdiagramm (Abbildung 1) müssen nicht aufgeführt werden.

Erläutern Sie, wie Sie jede der genannten Anforderungen in Ihrem Implementationsdiagramm realisieren.

(12 Punkte)

e) *Nehmen Sie Stellung, inwieweit man statt der Datenstruktur* `List` *die Datenstruktur* `Queue` *für die Verwaltung der offenen Bestellungen nehmen könnte, indem Sie die Vor- und Nachteile der beiden Datenstrukturen im vorliegenden Sachzusammenhang darstellen.*

(7 Punkte)

Zugelassene Hilfsmittel:

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse Pizzalieferdienst

Ein Objekt der Klasse `Pizzalieferdienst` dient zur Verwaltung der Bestellungen.

Ausschnitt aus der Dokumentation der Klasse `Pizzalieferdienst`

Konstruktor **`Pizzalieferdienst()`**
Ein neues Pizzalieferdienstobjekt wird erzeugt. Die Listen `offeneBestellungen` und `auslieferbareBestellungen` werden initialisiert und sind leer.

Auftrag **`void nimmAuf(Bestellung pBestellung)`**
Das Objekt `pBestellung` wird anhand seiner Priorität absteigend in die Liste `offeneBestellungen` eingereiht. Bei gleicher Priorität wird das Objekt `pBestellung` hinter die bestehenden Bestellungen mit dieser Priorität eingereiht.

Auftrag **`void fertigeOffeneBestellungAb()`**
Das erste Objekt aus der Liste `offeneBestellungen` wird entfernt und an die Liste `auslieferbareBestellungen` angehängt.
Falls die Liste `offeneBestellungen` leer ist, geschieht nichts.

Anfrage **`Bestellung liefereBestellungAus()`**
Die Methode liefert das vorderste Objekt aus der Liste `auslieferbareBestellungen` zurück und entfernt dieses aus der Liste.
Falls die Liste `auslieferbareBestellungen` leer ist, wird `null` zurückgegeben.

Anfrage **`List<Bestellung> liefereAlleBestellungenAus()`**
Die Methode liefert alle Objekte aus der Liste `auslieferbareBestellungen` zurück und entfernt diese aus der Liste.
Falls die Liste `auslieferbareBestellungen` leer ist, wird eine leere Liste zurückgeliefert.

Name: _____

Die Klasse Bestellung

Ein Objekt der Klasse `Bestellung` verwaltet die Informationen einer Bestellung.

Ausschnitt aus der Dokumentation der Klasse `Bestellung`

Konstruktor `Bestellung(int pBestellungID, String pGericht, String pAnschrift, int pPrioritaet, double pPreis)`
Ein Bestellungsobjekt mit den im Parameter übergebenen Werten wird erzeugt.

Anfrage `int gibBestellungID()`
Die ID der Bestellung wird zurückgeliefert.

Anfrage `String gibGericht()`
Der Name des Gerichts wird zurückgeliefert.

Anfrage `String gibAnschrift()`
Die Lieferanschrift wird zurückgeliefert.

Anfrage `int gibPrioritaet()`
Die Priorität wird zurückgeliefert.

Anfrage `double gibPreis()`
Der Preis wird zurückgeliefert.

Auftrag `void setzePrioritaet(int pPrioritaet)`
Die Priorität wird auf den im Parameter übergebenen Wert gesetzt.

Die Klasse String

Ausschnitt aus der Dokumentation der Klasse `String`

Anfrage `compareTo(String pAnotherString) : int`
Diese Anfrage liefert den Wert `0`, wenn die Strings lexikografisch identisch sind. Sie liefert einen Wert kleiner als `0`, wenn `pAnotherString` lexikografisch größer als der String ist. Sie liefert einen Wert größer als `0`, wenn `pAnotherString` lexikografisch kleiner als der String ist.

Name: _____

Die generische Klasse List<ContentType>

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 können 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>`

Konstruktor **`List()`**

Eine leere Liste wird erzeugt. Objekte, die in dieser Liste verwaltet werden, müssen vom Typ `ContentType` sein.

Anfrage **`boolean isEmpty()`**

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

Anfrage **`boolean hasAccess()`**

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

Auftrag **`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`.

Auftrag **`void toFirst()`**

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

Auftrag **`void toLast()`**

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

Name: _____

Anfrage `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.

Auftrag `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.

Auftrag `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.

Auftrag `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.

Auftrag `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.

Auftrag `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 2019

Informatik, Leistungskurs

1. Aufgabenart

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

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2019

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, Array

   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

5. Zugelassene Hilfsmittel

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

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)

Zustand der Listen nach `i`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 1, Prio: 2 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔

Zustand der Listen nach `ii`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 2, Prio: 3 ➔ ID 1, Prio: 2 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔

Zustand der Listen nach `iii`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 2, Prio: 3 ➔ ID 3, Prio: 3 ➔ ID 1, Prio: 2 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔

Zustand der Listen nach `iv`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 3, Prio: 3 ➔ ID 1, Prio: 2 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔ ID 2, Prio: 3 ➔

Zustand der Listen nach `v`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 3, Prio: 3 ➔ ID 1, Prio: 2 ➔ ID 4, Prio: 1 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔ ID 2, Prio: 3 ➔

Zustand der Listen nach `vi`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 3, Prio: 3 ➔ ID 1, Prio: 2 ➔ ID 4, Prio: 1 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔

Zustand der Listen nach `vii`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 1, Prio: 2 ➔ ID 4, Prio: 1 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔ ID 3, Prio: 3 ➔

Zustand der Listen nach `viii`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 4, Prio: 1 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔ ID 3, Prio: 3 ➔ ID 1, Prio: 2 ➔

Zustand der Listen nach `ix`:
`offeneBestellungen`:
(Listenanfang) ➔ ID 4, Prio: 1 ➔
`auslieferbareBestellungen`:
(Listenanfang) ➔

Ein Objekt der Klasse `Pizzalieferdienst` verwaltet in der linearen Liste `offeneBestellungen` Objekte vom Typ `Bestellung`.
Ein Objekt der Klasse `Pizzalieferdienst` verwaltet in der linearen Liste `auslieferbareBestellungen` Objekte vom Typ `Bestellung`.

Teilaufgabe b)

BestellungID	Priorität
9	3
6	3
5	2
8	2
3	1

Abbildung 1: Belegung der Liste offeneBestellungen nach Methodenausführung

Die Methode `tueEtwas` bekommt eine Liste mit Integerzahlen übergeben und liefert nichts zurück.
Die Methode `gibEtwas` bekommt eine Liste mit Integerzahlen übergeben und liefert eine Liste mit Bestellungsobjekten zurück.

Die Methode `tueEtwas` verwaltet das Ergebnis der Methode `gibEtwas` in der lokalen Variable `neueListe` (vgl. Zeile 2). Diese Liste mit Bestellungsobjekten wird komplett durchlaufen. Dabei wird jedes Bestellungsobjekt mit der Methode `nimmAuf` sortiert nach Priorität in die Liste `offeneBestellungen` eingefügt (vgl. Zeilen 3 – 7).
In der Methode `gibEtwas` wird die neue leere Liste `liste` für Bestellungsobjekte deklariert und initialisiert (vgl. Zeile 10).
In einer äußeren Schleife wird die Liste `offeneBestellungen` durchlaufen (vgl. Zeilen 11 – 29). In einer inneren Schleife wird die Liste `pListe` durchlaufen (vgl. Zeilen 16 – 25). Falls es in der Liste `offeneBestellungen` eine Bestellung mit einer der im Parameter übergebenen IDs gibt und die Priorität dieser Bestellung kleiner als 3 ist (vgl. Zeilen 17 f.), dann wird diese aus der Liste `offeneBestellungen` gelöscht (vgl. Zeile 22), die Priorität um eins erhöht (vgl. Zeile 20) und an die lokale Liste `liste` angehängt (vgl. Zeile 21).

Die Methoden erhöhen bei jeder Bestellung in der Liste `offeneBestellungen`, bei der die Bestellnummer mit einer der im Parameter übergebenen Bestellnummern übereinstimmt und die Priorität kleiner als drei ist, die Priorität um eins. Dabei wird die entsprechende Bestellung aus der Liste `offeneBestellung` gelöscht und mit der um eins erhöhten Priorität neu eingefügt.

Teilaufgabe c)

Strategie:

- Eine neue anfangs leere Liste `neueListe` für Bestellungsobjekte wird erzeugt.
- Die Liste `auslieferbareBestellungen` wird einmal vollständig durchlaufen.
  - Jedes Bestellobjekt wird im Sinne von InsertionSort aufsteigend nach der Anschrift in die Liste `neueListe` eingefügt.
  - Man beginnt bei der Suche für die richtige Einfügeposition bei dem ersten Element der Liste `neueListe`. Die aktuelle Referenz wird jeweils um ein Element nach hinten verschoben, solange die Anschrift des neu einzufügenden Bestellobjekts lexikografisch größer gleich ist als die Anschrift des aktuellen Elements der Liste `neueListe`.
- Der Referenz `auslieferbareBestellungen` wird die Referenz `neueListe` zugewiesen.

Implementierung:

```
public void sortiereFertigeBestellungen() {
  List<Bestellung> neueListe = new List<Bestellung>();
  auslieferbareBestellungen.toFirst();
  while (auslieferbareBestellungen.hasAccess()) {
    Bestellung aktBestellung
        = auslieferbareBestellungen.getContent();
    auslieferbareBestellungen.remove();
    neueListe.toFirst();
    while (neueListe.hasAccess()
           && neueListe.getContent().gibAnschrift()
              .compareTo(aktBestellung.gibAnschrift()) <= 0) {
      neueListe.next();
    }
    if (neueListe.hasAccess()) {
      neueListe.insert(aktBestellung);
    } else {
      neueListe.append(aktBestellung);
    }
  }
  auslieferbareBestellungen = neueListe;
}
```

Teilaufgabe d)

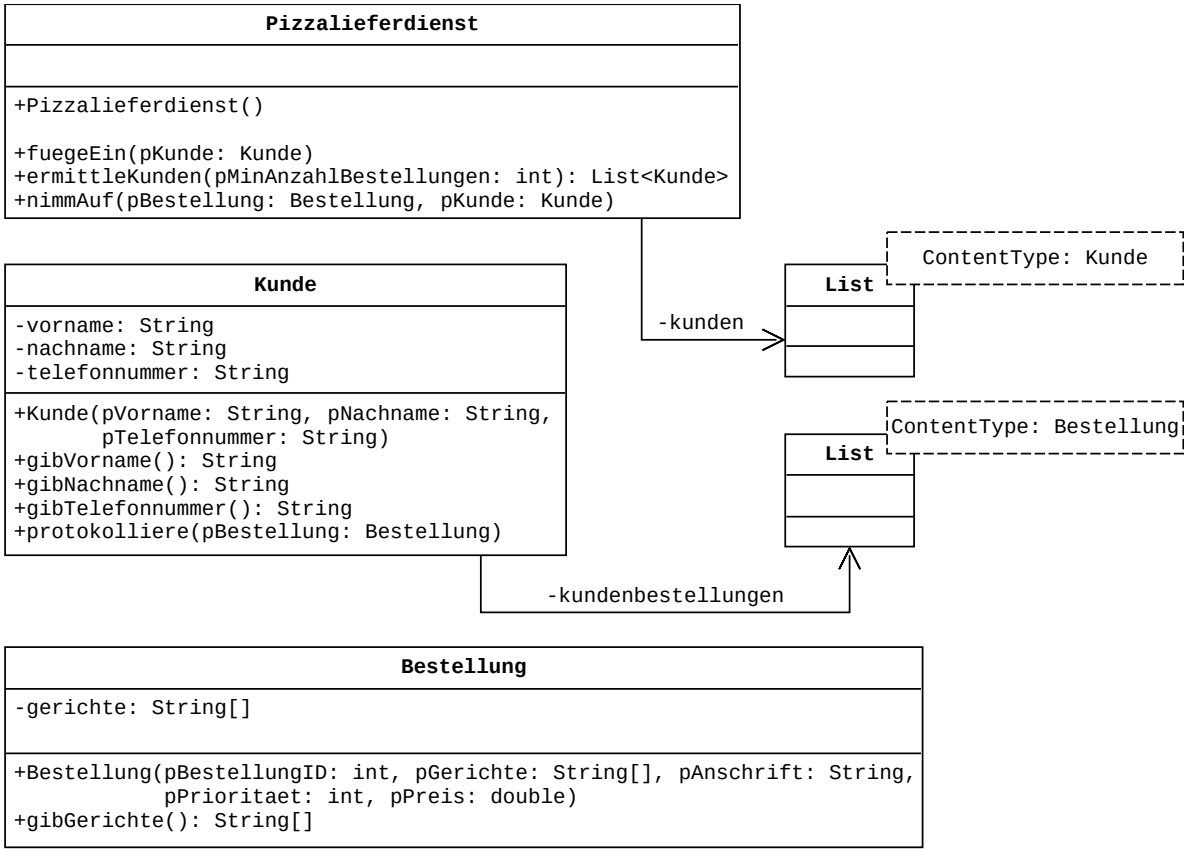


Abbildung 2: Erweiterung des Implementationsdiagramms

Zu i) In der Klasse `Bestellung` wird nicht mehr das Stringobjekt `gericht` verwaltet, sondern ein Stringarray namens `gerichte`. Der Konstruktor und die zugehörige Getter-Methode wurden angepasst.

Zu ii) Der Vorname, der Nachname und die Telefonnummer eines Kunden werden mithilfe der Klasse `Kunde` verwaltet.
Die Klasse `Pizzalieferdienst` verwaltet alle Kunden in dem Objekt `kunden` vom Typ `List`.

Zu iii) Die Klasse `Pizzalieferdienst` enthält die Methoden `fuegeEin`.

Zu iv) Die Klasse `Kunde` verwaltet die eigenen Bestellungen in dem Objekt `kundenbestellungen` vom Typ `List`.
Außerdem wird in der Klasse `Pizzalieferdienst` die Methode `nimmAuf` ersetzt. Die neue Methode enthält die zwei Parameter `pBestellung` und `pKunde`.
Zusätzlich enthält die Klasse `Kunde` die Methode `protokolliere`, um eine neue Bestellung an die Liste `kundenbestellungen` anzuhängen.

Zu v) Die Verwaltung stellt die Methode `ermittleKunden` zur Verfügung, die eine Liste mit Objekten von der Klasse `Kunde` zurückliefert. Als Parameter wird der Schwellenwert `pMinAnzahlBestellungen` übergeben.

Teilaufgabe e)

Das Listenobjekt `offeneBestellungen` stellt eine Prioritätswarteschlange dar. Die Bestellungen werden aufsteigend sortiert nach Prioritäten eingefügt. Dies erfordert die Möglichkeit zum Einfügen an jeder Stelle der linearen Datenstruktur. Dafür eignet sich insbesondere die Datenstruktur `List`. Eine vollständige Ausgabe ist mit einem einfachen Durchlauf durch die Liste möglich.

Alternativ dazu werden zwei mögliche Varianten unter Verwendung der Klasse `Queue` genannt, die nach dem FIFO-Prinzip arbeitet:

1. Alternative: Eine einzige `Queue`
Bei einer `Queue` ist ein Einfügen am Ende der Schlange effizient möglich. Ein Einfügen an einer anderen Stelle ist nicht vorgesehen, ließe sich allerdings mit zusätzlichem Aufwand dennoch realisieren. Z. B. könnte man dabei wie folgt vorgehen: Man müsste für das Finden der richtigen Einfügestelle die `Queue` sukzessive abbauen und die entfernten Objekte in einer lokalen Hilfsschlange speichern und anschließend das neue Bestellungsobjekt einfügen und die Schlangen wieder vereinen.

2. Alternative: Pro Prioritätsstufe eine `Queue`
Falls die Anzahl der Prioritätsstufen im Vorfeld festgelegt und begrenzt ist, könnte man für jede Prioritätsstufe ein `Queue`-Objekt vorsehen. Beim Einfügen wird die neue Bestellung sofort an das richtige `Queue`-Objekt hinten angehängt. Das Suchen der richtigen Einfügestelle, wie bei der `List`, würde entfallen.
Beim Auslesen bzw. Löschen einer Bestellung würde man sukzessive die `Queue`-Objekte abarbeiten und das vorderste Bestellungsobjekt aus dem `Queue`-Objekt mit der höchsten Priorität suchen bzw. löschen.
Wenn man eine oder mehrere Schlangen ausgeben möchte, müsste man die Datenstruktur sukzessive abbauen, das aktuelle Element ausgeben und anschließend die Datenstruktur wieder aufbauen, was mit einem Zusatzaufwand verbunden wäre.

Die Verwaltung der offenen Bestellungen im Sinne einer Prioritätswarteschlange mit einer Liste erscheint geeigneter als die Verwendung mehrerer `Queue`-Objekte oder eines `Queue`-Objektes. Insbesondere das problemlose Hinzufügen von weiteren Prioritätsstufen, die komfortablere Möglichkeit zum Sortieren und die einfache Ausgabe sind überzeugende Argumente für die verwendete Datenstruktur `List`.

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	dokumentiert die Belegungen der Listenobjekte `offeneBestellungen` und `auslieferbareBestellungen`.	6			
2	beschreibt die Beziehungen zwischen den Klassen `Pizzalieferdienst`, `List` und `Bestellung`.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methoden `tueEtwas` und `gibEtwas`, indem er die Informationen `BestellungID` und `Priorität` der Testdaten angibt, nachdem die Methode `tueEtwas` mit einer Liste bestehend aus den Zahlen `8`, `6` und `9` aufgerufen wurde.	3			
2	erläutert die Funktionsweise der Methoden.	6			
3	erläutert die Funktionalität der Methoden im Sachkontext.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe b)	**11**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine Strategie für die Arbeitsweise der Methode `sortiereAuslieferbareBestellungen` und stellt diese in geeigneter Form dar.	5			
2	implementiert die Methode.	7			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe c)	**12**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	modelliert die Erweiterung als Implementationsdiagramm.	7			
2	erläutert, wie jede Anforderung im Implementations-diagramm realisiert wird.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	nimmt Stellung, inwieweit man statt der Datenstruktur `List` die Datenstruktur `Queue` für die Verwaltung der offenen Bestellungen nehmen könnte, indem er die Vor- und Nachteile der beiden Datenstrukturen im vorliegenden Sachzusammenhang darstellt.	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 2019

Informatik, Leistungskurs

Aufgabenstellung:

In einem „Smart Home“ werden unterschiedliche Geräte (z. B. Beleuchtungsanlagen und Heizungssysteme) über Clients ferngesteuert. In einer ersten Version werden „smarte“ Heizkörper betrachtet. Ein Heizkörper kennt den Raum, in dem er sich befindet, sowie die Befehle `aktivieren` zum Einschalten und `deaktivieren` zum Ausschalten. Er ist im Zustand `an` oder `aus`. Abbildung 1 zeigt einen Ausschnitt des Kommunikationsprotokolls zwischen dem Server und einem Client. Die Parameterwerte enthalten keine Leerzeichen.

Client sendet an Server	Server sendet an Client

`ANMELDEN <Name> <Passwort>`	`+OK ANGEMELDET <Name>`
	`-ERR Anmeldedaten fehlerhaft`
	`-ERR Parameteranzahl fehlerhaft`

`STEUERE <GeraeteID> <Befehl>`	`+OK Befehl <Befehl> ausgefuehrt`
	`-ERR <GeraeteID> unbekannt`
	`-ERR <GeraeteID> im Zustand <Status>`
	`-ERR Parameteranzahl fehlerhaft`
	`-ERR Benutzer nicht angemeldet`

`STATUS <GeraeteID>`	`+OK <GeraeteID> im Status <Status>`
	`-ERR <GeraeteID> unbekannt`
	`-ERR Parameteranzahl fehlerhaft`
	`-ERR Benutzer nicht angemeldet`

`GERAETELISTE`	`+OK ALLE_GERAETE`
	`<GeraeteID_1> <Standort_1> <Status_1>;`
	`<GeraeteID_2> <Standort_2> <Status_2>; ...;`
	`<GeraeteID_n> <Standort_n> <Status_n>`
	`-ERR Parameteranzahl fehlerhaft`
	`-ERR Benutzer nicht angemeldet`

`ABMELDEN`	`+OK ABGEMELDET`
	Danach trennt der Server die Verbindung.
	`-ERR Parameteranzahl fehlerhaft`
	`-ERR Benutzer nicht angemeldet`

`Unbekannter Befehl`	`-ERR`

Abbildung 1: Ausschnitt aus dem Kommunikationsprotokoll

Name: _____

a) Folgende beispielhafte Kommunikation ist gegeben:
   i. Ein Hausbesitzer meldet sich beim Server mit seinem Namen `Otto` und dem Passwort `enter123` an. Dabei nutzt er zunächst ein falsches Passwort.
   ii. Beim zweiten Versuch nutzt er die richtigen Anmeldedaten.
   iii. Er lässt sich die Geräteliste anzeigen. In seinem Wohnhaus gibt es vier „smarte" Heizkörper: `Heizkoerper1` befindet sich im Schlafzimmer, `Heizkoerper2` in der Küche, `Heizkoerper3` im Wohnzimmer und `Heizkoerper4` im Kinderzimmer. Nur der Heizkörper im Wohnzimmer ist bereits eingeschaltet, alle anderen sind ausgeschaltet.
   iv. Er möchte den Heizkörper in der Küche einschalten.
   v. Anschließend möchte er den Heizkörper im Schlafzimmer ausschalten.
   vi. Danach meldet er sich vom Server ab.

   Nach ersten Tests soll das in Abbildung 1 dargestellte Kommunikationsprotokoll erweitert werden, um nicht nur einzelne Geräte, sondern alle Geräte des gleichen Typs (z. B. alle Heizkörper) gleichzeitig mit einem Befehl zu steuern.

   Stellen Sie die beispielhafte Kommunikation zwischen dem Client und dem Server nach den Vorgaben aus Abbildung 1 tabellarisch dar.

   Erweitern Sie das Kommunikationsprotokoll aus Abbildung 1 wie beschrieben unter Berücksichtigung zweier neuer Fehlerfälle.

(9 Punkte)

Es wurde ein Client entwickelt, über den sich Benutzer beim Server anmelden und die vorhandenen Heizkörper verwalten. Abbildung 2 stellt einen Ausschnitt des zugehörigen Implementationsdiagramms dar.
Die vom Server verwalteten Objekte der Klasse `Heizkoerperdatensatz` entsprechen den vom Server vorgehaltenen, konsistenten Datenabbildern der real verbauten „smarten" Heizkörper, die über eigene Clients mit dem Server kommunizieren. Ein neuer Heizkörper meldet sich bei seiner Inbetriebnahme automatisch beim Server an und informiert diesen über alle Änderungen seines Zustands, damit dieser sein Datenabbild aktualisieren kann. Umgekehrt leitet der Server Aufträge des Benutzers an den Heizkörper weiter, nachdem er sein Datenabbild entsprechend angepasst hat. Auf die Modellierung der Kommunikation zwischen dem Server und den Heizkörper-Clients wird im Rahmen dieser Aufgabe nicht näher eingegangen. Die Datenabbilder werden im weiteren Aufgabentext als Heizkörper bezeichnet.
Das Attribut `an` der Klasse `Heizkoerperdatensatz` hat den Wert `true`, wenn der Heizkörper eingeschaltet ist. Andernfalls hat das Attribut den Wert `false`.

Eine vollständige Dokumentation relevanter Klassen finden Sie im Anhang.

Name: _____

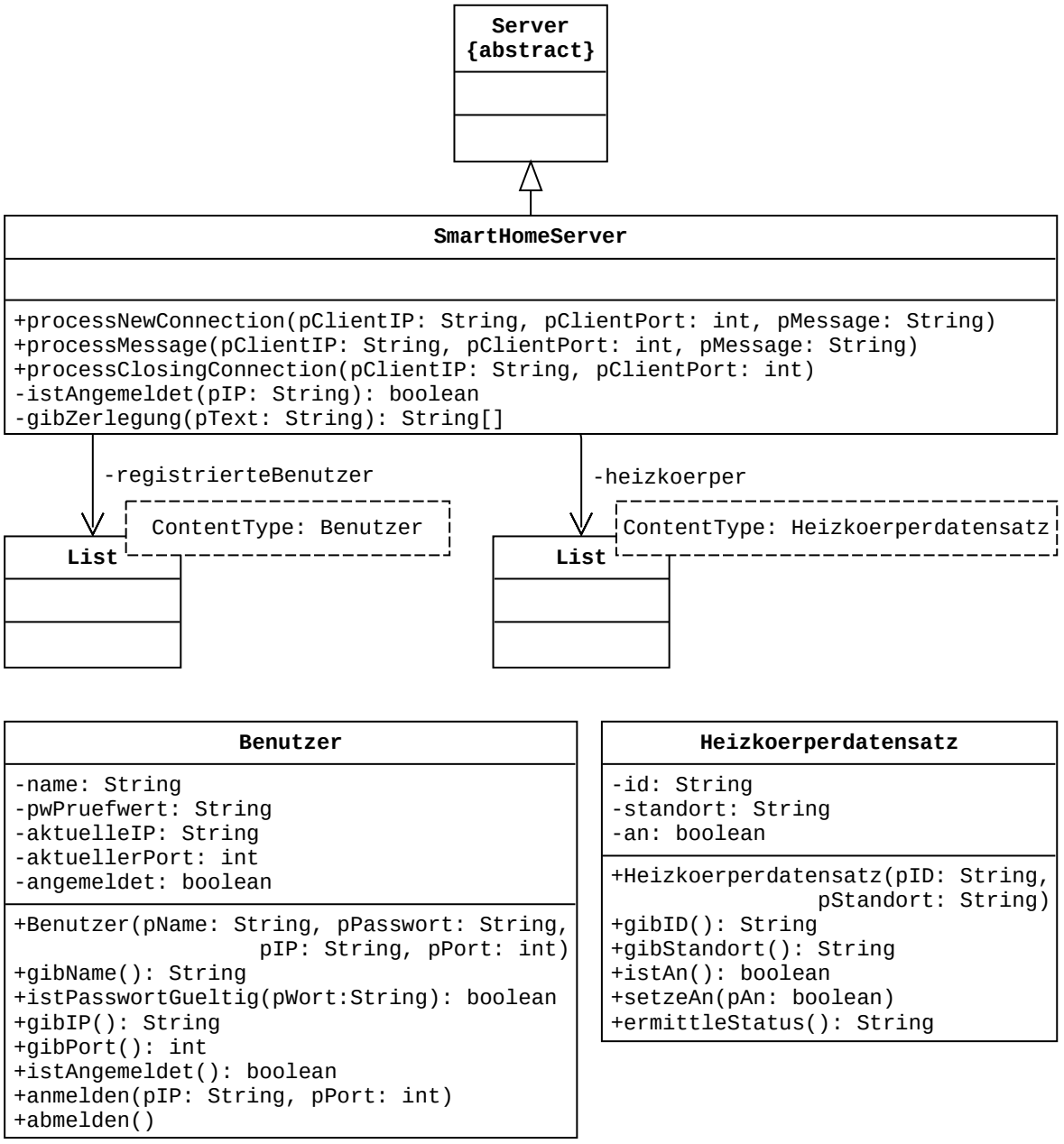


Abbildung 2: Implementationsdiagramm (Teilmodellierung)

Name: _____

b) Der folgende Anfang der Methode `processMessage` der Klasse `SmartHomeServer` ist gegeben:

```
1 public void processMessage (String pClientIP,
2                             int pClientPort,
3                             String pMessage) {
4   boolean angemeldet = istAngemeldet(pClientIP);
5   String[] nachricht = gibZerlegung(pMessage);
…   …
```

Implementieren Sie den Teil der Methode `processMessage`*, der nur den Protokollbefehl* `STATUS` *gemäß Abbildung 1 umsetzt.*

(12 Punkte)

c) Ein Benutzer möchte vor der Fahrt in den Urlaub schnell überprüfen, ob alle Heizkörper entweder eingeschaltet oder ausgeschaltet wurden. Dazu wird das bisherige Protokoll wie folgt erweitert:

Client sendet an Server	Server sendet an Client
`PRUEFE <Status>`	`+OK Alle Geraete im Status <Status>` `-ERR Nicht alle Geraete im Status <Status>`

Abbildung 3: Erweiterung des Kommunikationsprotokolls aus Abbildung 1

Die Klasse `SmartHomeServer` wird um folgende private Hilfsmethode erweitert:

```
private boolean sindAlleImZustand(String pZustand)
```

Die Methode gibt `true` zurück, wenn sich alle Heizkörper im Zustand `pZustand` befinden, andernfalls liefert sie `false` zurück.

Entwerfen Sie einen Algorithmus, der die Arbeitsweise dieser Methode realisiert.

Implementieren Sie die Methode `sindAlleImZustand` *gemäß Ihrem Algorithmus.*

(9 Punkte)

Name: _____

d) Aufgrund des großen Erfolgs der „smarten“ Heizkörper entwickelt der Hersteller diese weiter, indem er einen Sensor am Heizkörperventil anbringt, der die Temperatur im Raum messen kann. Entsprechend wird die Klasse `Heizkoerperdatensatz` aus Abbildung 2 wie in Abbildung 4 dargestellt um ein Attribut und zwei Methoden erweitert.

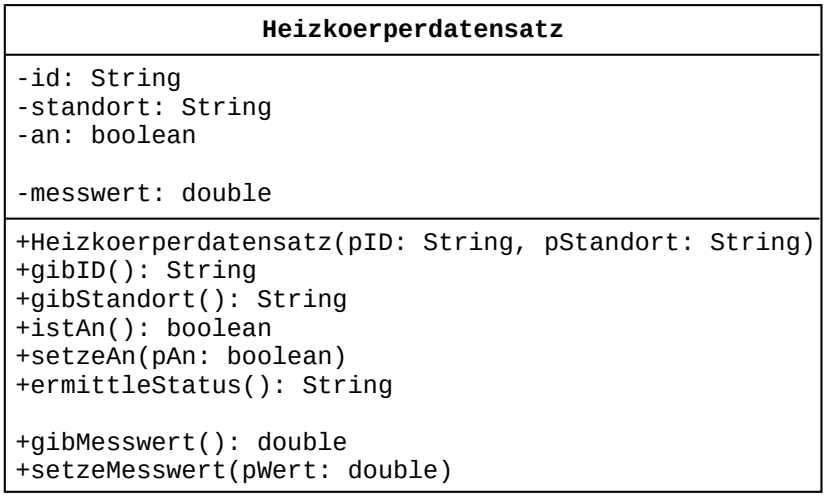


Abbildung 4: Klassendiagramm der erweiterten Klasse `Heizkoerperdatensatz`

Die Methode `gibMesswert` liefert den zuletzt gespeicherten Messwert des Heizkörpers zurück. Die Methode `setzeMesswert` setzt den gespeicherten Messwert des Heizkörpers auf `pWert`.

Name: _____

Die Klasse `SmartHomeServer` wird um folgende Methoden erweitert:

```
 1 private boolean istEtwas(double pWert) {
 2   boolean b = false;
 3   List<Heizkoerperdatensatz> liste = wasLiefereIch();
 4   if (!liste.isEmpty()) {
 5     liste.toFirst();
 6     double wert1 = liste.getContent().gibMesswert();
 7     liste.toLast();
 8     double wert2 = liste.getContent().gibMesswert();
 9     if (wert2 - wert1 > pWert) {
10       b = true;
11     }
12   }
13   return b;
14 }
15
16 private List<Heizkoerperdatensatz> wasLiefereIch() {
17   List<Heizkoerperdatensatz> h
18       = new List<Heizkoerperdatensatz>();
19   heizkoerper.toFirst();
20   while (heizkoerper.hasAccess()) {
21     Heizkoerperdatensatz aktuell = heizkoerper.getContent();
22     h.toFirst();
23     while (h.hasAccess()
24            && aktuell.gibMesswert()
25               > h.getContent().gibMesswert()) {
26       h.next();
27     }
28     if (h.hasAccess()) {
29       h.insert(aktuell);
30     } else {
31       h.append(aktuell);
32     }
33     heizkoerper.next();
34   }
35   return h;
36 }
```

Analysieren Sie die Methoden `wasLiefereIch` *und* `istEtwas` *der Klasse* `SmartHomeServer`*, indem Sie jeweils die Arbeitsweise und die Funktionalität im Sachzusammenhang erläutern.*

(10 Punkte)

Name: _____

e) Die Eigentümerin eines Wohnhauses mit mehreren Mietparteien plant, dieses in ein „Smart Home“ mit „smarten“ Heizkörpern umzuwandeln. Eine der Mietparteien äußert Bedenken bezüglich der Sicherheit und macht sich Sorgen, dass fremde Heizkörper manipuliert werden könnten. Die Eigentümerin bietet an, dem „Smart Home“ neben dem bereits realisierten Anmeldeverfahren eine individualisierte Zugriffssteuerung für die unterschiedlichen Geräte einzubauen. Dabei sollen folgende Aspekte umgesetzt werden:
   - Die individuellen Zugriffsrechte müssen in geeigneter Form gespeichert werden.
   - Es muss überprüfbar sein, ob ein Benutzer Zugriff auf einen Heizkörper hat.
   - Die Berechtigung zur Steuerung eines Heizkörpers muss einem Benutzer zugeteilt und auch wieder entzogen werden können.

   Entwickeln Sie in Anlehnung an das Implementationsdiagramm aus Abbildung 2 eine Modellierung, mit der die genannten Aspekte einer individualisierten Zugriffssteuerung realisiert werden.

   Hinweis: Unveränderte Attribute und Methoden aus dem ursprünglichen Modell aus Abbildung 2 müssen nicht dargestellt werden.

   Beschreiben Sie, wie Sie die einzelnen Aspekte in Ihrer Modellierung umsetzen.

   (10 Punkte)

Zugelassene Hilfsmittel:

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse SmartHomeServer

Diese Klasse ist Unterklasse der Klasse `Server`. Objekte dieser Klasse verwalten die am Server angemeldeten Benutzer sowie Datenabbilder aller „Smart Home“-Geräte (wie z. B. Heizkörper). Dabei stellt sie neben den geerbten Methoden unter anderem folgende Methoden zur Verfügung:

Dokumentation der Klasse `SmartHomeServer`

Auftrag **`void processClosingConnection(String pClientIP, int pClientPort, String pMessage)`**
Das Benutzer-Objekt mit der IP-Adresse `pClientIP` und der Port-Nummer `pClientPort` meldet sich ab. Dem Benutzer wird die Nachricht `"+OK ABGEMELDET"` gesendet. Die Verbindung zum Client wird getrennt.

Dokumentation privater Methoden der Klasse `SmartHomeServer`

Anfrage **`boolean istAngemeldet(String pClientIP)`**
Die Anfrage liefert den Wert `true`, wenn ein Benutzer mit der IP `pClientIP` angemeldet ist. Sonst liefert sie den Wert `false`.

Anfrage **`String[] gibZerlegung(String pText)`**
Die Anfrage zerlegt den Text `pText` anhand der darin enthaltenen Leerzeichen in einzelne Worte und liefert diese in einem Feld zurück. Das erste Wort von `pText` wird zum ersten Element des Feldes, das zweite Wort zum zweiten Element usw., die Feldlänge entspricht der Anzahl der Worte. Falls `pText` leer ist, wird ein leeres Feld zurückgegeben.

Name: _____

Die Klasse Benutzer

Objekte der Klasse `Benutzer` verwalten ein Benutzerkonto für eine Person, die in einem „Smart Home“ lebt und vom Server mit ihren Anmeldedaten (Name und Passwortprüfwert), dem Anmeldestatus und ihren aktuellen Verbindungsdaten (IP und Port) verwaltet wird.

Dokumentation der Klasse `Benutzer`

Konstruktor **`Benutzer(String pName, String pPasswort, String pIP, int pPort)`**
Die Methode erstellt ein neues Objekt der Klasse `Benutzer` mit dem eindeutigen Namen `pName`, einem aus `pPasswort` ermittelten Passwortprüfwert sowie seinen zugehörigen Verbindungsdaten – bestehend aus der IP-Adresse `pIP` und der Port-Nummer `pPort`.
Ein neuer Benutzer gilt zunächst als abgemeldet.

Anfrage **`String gibName()`**
Die Anfrage liefert den Namen des Benutzers zurück.

Anfrage **`boolean istPasswortGueltig(String pWort)`**
Die Anfrage liefert den Wert `true`, wenn der aus `pWort` ermittelte Passwortprüfwert dem Passwortprüfwert des Benutzers entspricht. Sonst liefert sie `false` zurück.

Anfrage **`String gibIP()`**
Die Anfrage liefert die IP-Adresse des Benutzers zurück.

Anfrage **`int gibPort()`**
Die Anfrage liefert die Portnummer des Benutzers zurück.

Anfrage **`boolean istAngemeldet()`**
Die Anfrage liefert den Anmeldestatus des Benutzers zurück.
Sie liefert also den Wert `true`, wenn der Benutzer angemeldet ist. Sonst liefert sie den Wert `false`.

Auftrag **`void anmelden(String pIP, int pPort)`**
Die Verbindungsdaten des Benutzers werden auf die IP `pIP` und den Port `pPort` aktualisiert und der Anmeldestatus wird auf `true` gesetzt.

Auftrag **`void abmelden()`**
Der Anmeldestatus des Benutzers wird auf `false` gesetzt.

Name: _____

Die Klasse Heizkoerperdatensatz

Objekte der Klasse `Heizkoerperdatensatz` verwalten alle Informationen eines „smarten“ Heizkörpers. Dabei handelt es sich um ein vom Server vorgehaltenes Abbild aller Daten eines realen Heizkörpers. Dieser teilt dem Server alle Änderungen seines Zustands mit und meldet sich bei diesem in regelmäßigen zeitlichen Abständen.

Dokumentation der Klasse `Heizkoerperdatensatz`

Konstruktor **`Heizkoerperdatendatz(String pID,String pStandort)`**
Die Methode erstellt ein neues Objekt der Klasse `Heizkoerperdatensatz` mit der eindeutigen Bezeichnung `pID`, das sich am Standort `pStandort` befindet.
Ein neuer Heizkörper ist im Zustand `aus`.

Anfrage **`String gibID()`**
Die Anfrage liefert die Bezeichnung des Heizkörpers zurück.

Anfrage **`String gibStandort()`**
Die Anfrage liefert den Standort des Heizkörpers zurück.

Anfrage **`boolean istAn()`**
Die Anfrage liefert den Wert `true`, wenn der Heizkörper im Zustand `an` ist. Sonst liefert sie den Wert `false`.

Auftrag **`void setzeAn(boolean pAn)`**
Der Zustand des Heizkörpers wird auf `pAn` gesetzt.

Anfrage **`String ermittleStatus()`**
Die Anfrage liefert als Information zurück, ob der Heizkörper an oder aus ist.

Name: _____

Die Klasse Server

Objekte von Unterklassen der abstrakten Klasse Server ermöglichen das Anbieten von Serverdiensten, so dass Clients Verbindungen zum Server mittels TCP/IP-Protokoll aufbauen können. Zur Vereinfachung finden Nachrichtenversand und -empfang zeilenweise statt, d. h., beim Senden einer Zeichenkette wird ein Zeilentrenner ergänzt und beim Empfang wird dieser entfernt. Verbindungsannahme, Nachrichtenempfang und Verbindungsende geschehen nebenläufig. Auf diese Ereignisse muss durch Überschreiben der entsprechenden Ereignisbehandlungsmethoden reagiert werden. Es findet nur eine rudimentäre Fehlerbehandlung statt, so dass z. B. Verbindungsabbrüche nicht zu einem Programmabbruch führen. Einmal unterbrochene oder getrennte Verbindungen können nicht reaktiviert werden.

Dokumentation der Klasse `Server`

Konstruktor **`Server(int pPort)`**
Ein Objekt vom Typ `Server` wird erstellt, das über die angegebene Portnummer einen Dienst anbietet. Clients können sich mit dem Server verbinden, so dass Daten (Zeichenketten) zu diesen gesendet und von diesen empfangen werden können. Kann der Server unter der angegebenen Portnummer keinen Dienst anbieten (z. B. weil die Portnummer bereits belegt ist), ist keine Verbindungsaufnahme zum Server und kein Datenaustausch möglich.

Anfrage **`boolean isOpen()`**
Die Anfrage liefert den Wert `true`, wenn der Server auf Port `pPort` einen Dienst anbietet. Ansonsten liefert die Methode den Wert `false`.

Anfrage **`boolean isConnectedTo(String pClientIP, int pClientPort)`**
Die Anfrage liefert den Wert `true`, wenn der Server mit dem durch `pClientIP` und `pClientPort` spezifizierten Client aktuell verbunden ist. Ansonsten liefert die Methode den Wert `false`.

Auftrag **`void send (String pClientIP, int pClientPort, String pMessage)`**
Die Nachricht `pMessage` wird – um einen Zeilentrenner erweitert – an den durch `pClientIP` und `pClientPort` spezifizierten Client gesendet. Schlägt der Versand fehl, geschieht nichts.

Auftrag **`void sendToAll(String pMessage)`**
Die Nachricht `pMessage` wird – um einen Zeilentrenner erweitert – an alle mit dem Server verbundenen Clients gesendet. Schlägt der Versand an *einen* Client fehl, wird dieser Client übersprungen.

Name: _____

Auftrag **`void closeConnection(String pClientIP, int pClientPort)`**
Die Verbindung des Servers zu dem durch `pClientIP` und `pClientPort` spezifizierten Client wird getrennt. Zuvor wird die Methode `processClosingConnection` mit IP-Adresse und Port des jeweiligen Clients aufgerufen. Ist der Server nicht mit dem in der Parameterliste spezifizierten Client verbunden, geschieht nichts.

Auftrag **`void close()`**
Alle bestehenden Verbindungen zu Clients werden getrennt und der Server kann nicht mehr verwendet werden. Ist der Server bereits vor Aufruf der Methode in diesem Zustand, geschieht nichts.

Auftrag **`void processNewConnection(String pClientIP, int pClientPort)`**
Diese Ereignisbehandlungsmethode wird aufgerufen, wenn sich ein Client mit IP-Adresse `pClientIP` und Portnummer `pClientPort` mit dem Server verbunden hat. Die Methode ist abstrakt und muss in einer Unterklasse der Klasse `Server` überschrieben werden, so dass auf den Neuaufbau der Verbindung reagiert wird. Der Aufruf der Methode erfolgt nicht synchronisiert.

Auftrag **`void processMessage(String pClientIP, int pClientPort, String pMessage)`**
Diese Ereignisbehandlungsmethode wird aufgerufen, wenn der Server die Nachricht `pMessage` von dem durch `pClientIP` und `pClientPort` spezifizierten Client empfangen hat. Der vom Client hinzugefügte Zeilentrenner wurde zuvor entfernt. Die Methode ist abstrakt und muss in einer Unterklasse der Klasse `Server` überschrieben werden, so dass auf den Empfang der Nachricht reagiert wird. Der Aufruf der Methode erfolgt nicht synchronisiert.

Auftrag **`void processClosingConnection(String pClientIP, int pClientPort)`**
Sofern der Server die Verbindung zu dem durch `pClientIP` und `pClientPort` spezifizierten Client trennt, wird diese Ereignisbehandlungsmethode aufgerufen, unmittelbar *bevor* die Verbindungstrennung tatsächlich erfolgt. Wird die Verbindung unvermittelt unterbrochen oder hat der in der Parameterliste spezifizierte Client die Verbindung zum Server unvermittelt getrennt, erfolgt der Methodenaufruf *nach* der Unterbrechung/Trennung der Verbindung. Die Methode ist abstrakt und muss in einer Unterklasse der Klasse `Server` überschrieben werden, so dass auf das Ende der Verbindung zum angegebenen Client reagiert wird. Der Aufruf der Methode erfolgt nicht synchronisiert.

Name: _____

Die generische Klasse List<ContentType>

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 können 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>`

Konstruktor **`List()`**

Eine leere Liste wird erzeugt. Objekte, die in dieser Liste verwaltet werden, müssen vom Typ `ContentType` sein.

Anfrage **`boolean isEmpty()`**

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

Anfrage **`boolean hasAccess()`**

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

Auftrag **`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`.

Auftrag **`void toFirst()`**

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

Auftrag **`void toLast()`**

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

Name: _____

Anfrage `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.

Auftrag `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.

Auftrag `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.

Auftrag `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.

Auftrag `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.

Auftrag `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 2019

Informatik, Leistungskurs

1. Aufgabenart

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

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2019

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
     - Nicht-lineare Strukturen

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

   Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - Java

   Informatiksysteme
   - Einzelrechner und Rechnernetzwerke
   - Sicherheit

2. *Medien/Materialien*
   - entfällt

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- 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)

	Client sendet an Server	Server sendet an Client
i.	`ANMELDEN Otto enter`	`-ERR Anmeldedaten fehlerhaft`
ii.	`ANMELDEN Otto enter123`	`+OK ANGEMELDET Otto`

iii.	`GERAETELISTE`	`+OK ALLE_GERAETE`
		`Heizkoerper1 Schlafzimmer aus;`
		`Heizkoerper2 Küche aus;`
		`Heizkoerper3 Wohnzimmer an;`
		`Heizkoerper4 Kinderzimmer aus`

iv.	`STEUERE Heizkoerper2 aktivieren`	`+OK Befehl aktivieren ausgefuehrt`
v.	`STEUERE Heizkoerper1 deaktivieren`	`-ERR Heizkoerper1 im Zustand aus`
vi.	`ABMELDEN`	`+OK ABGEMELDET`

Eine mögliche Erweiterung des Kommunikationsprotokolls könnte wie folgt aussehen:

Client sendet an Server	Server sendet an Client

`STEUERE <Geraetetyp> <Befehl>`	`+OK Befehl <Befehl> fuer alle Geraete vom Typ <Geraetetyp> ausgefuehrt`
	`-ERR Keine Geraete vom Typ <Geraetetyp> gefunden`
	`-ERR Befehl <Befehl> unbekannt`

Teilaufgabe b)

Eine mögliche Implementierung könnte wie folgt aussehen:

```
public void processMessage (String pClientIP, int pClientPort,
                            String pMessage) {
  // Vorgabe laut Aufgabenstellung
  boolean angemeldet = istAngemeldet(pClientIP);
  String[] nachricht = gibZerlegung(pMessage);

  String befehl = nachricht[0];
  if (befehl.equals("STATUS")) {
    // Anmeldung pruefen, ggf. Fehlermeldung senden
    if (!angemeldet) {
      send(pClientIP, pClientPort,
           "-ERR Benutzer nicht angemeldet");
    } else {
      // Anzahl der Parameter pruefen
      if (nachricht.length != 2) {
        send(pClientIP, pClientPort,
             "-ERR Parameteranzahl fehlerhaft");
      } else {
        // Heizkoerperdatensatz in Liste suchen
        String pID = nachricht[1];
        boolean gefunden = false;
        heizkoerper.toFirst();
        while (heizkoerper.hasAccess() && !gefunden) {
          if (heizkoerper.getContent().gibID().equals(pID)) {
            gefunden = true;
          } else {
            heizkoerper.next();
          }
        }
        // Statusmeldung oder Fehlermeldung senden
        String ausgabe = "";
        if (gefunden) {
          Heizkoerperdatensatz aktuell = heizkoerper.getContent();
          ausgabe = "+OK " + aktuell.gibID()
                    + " im Status " + aktuell.ermittleStatus();
        } else {
          ausgabe = "-ERR " + pID + " unbekannt";
        }
        send(pClientIP, pClientPort, ausgabe);
      }
    }
  }
}
```

Hinweis: Die Kommentare sind nicht verlangt.

Teilaufgabe c)

Die Methode `sindAlleImZustand` muss die Liste der Heizkörper durchlaufen und für jeden einzelnen Heizkörper prüfen, ob sich dieser im Zustand `pZustand` befindet. Zum Auslesen des aktuellen Zustands eines Heizkörpers dient die Methode `ermittleStatus`.
Sobald die Überprüfung erstmalig fehlschlägt, kann die Überprüfung der restlichen Heizkörper abgebrochen und als Ergebnis `false` zurückgegeben werden.
Sofern alle Heizkörper erfolgreich durchlaufen wurden (und damit alle durchgeführten Überprüfungen positiv verlaufen sind), wird als Ergebnis `true` zurückgegeben.

```
private boolean sindAlleImZustand(String pZustand) {
  boolean ergebnis = true;
  heizkoerper.toFirst();
  while (ergebnis && heizkoerper.hasAccess()) {
    Heizkoerperdatensatz aktuell = heizkoerper.getContent();
    if (!aktuell.ermittleStatus().equals(pZustand)) {
      ergebnis = false;
    }
    heizkoerper.next();
  }
  return ergebnis;
}
```

Teilaufgabe d)

In der privaten Methode `wasLiefereIch` wird eine zu Anfang leere Liste `h` deklariert und initialisiert (Zeilen 17 und 18), die am Ende (Zeile 35) zurückgegeben wird.
Die Methode durchläuft die Liste der Heizkörperdatensätze (Zeilen 19 bis 34) und speichert den jeweils aktuellen Heizkörperdatensatz in der Hilfsvariablen `aktuell` (Zeile 21). Der aktuelle Heizkörperdatensatz wird in die Liste `h` eingefügt (Zeilen 28 bis 32). Um die gewünschte Einfügestelle zu finden, durchläuft man die Liste `h` (Zeilen 22 bis 27), solange es ein aktuelles Element gibt (Zeile 23) und der Messwert des aktuell betrachteten Heizkörperdatensatzes größer als der Messwert des Elements der Liste `h` ist (Zeilen 24 und 25).

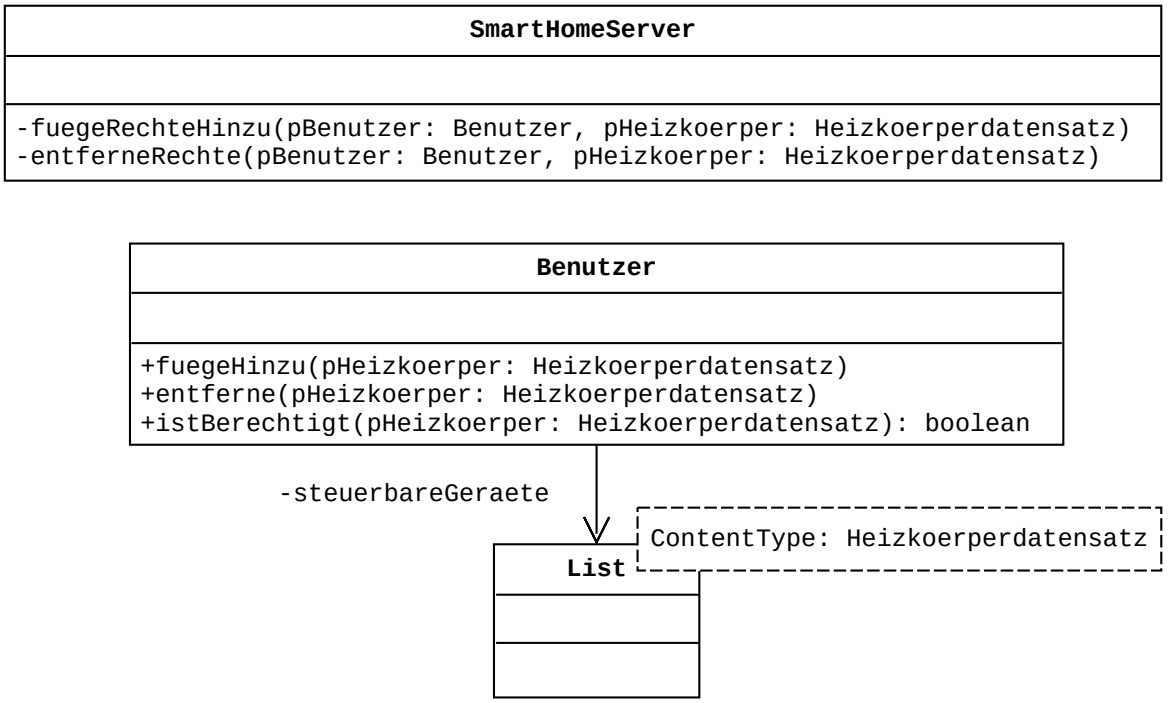
Insgesamt liefert die Methode `wasLiefereIch` eine Liste aller Heizkörperdatensätze, die aufsteigend nach den erfassten Messwerten sortiert ist.

Die private Methode `istEtwas` liefert einen Wahrheitswert zurück und erwartet als Übergabe im Parameter `pWert` eine Dezimalzahl (Zeile 1).
Zunächst wird eine boolesche Variable `b` deklariert und als `false` initialisiert (Zeile 2), die am Ende zurückgegeben wird (Zeile 13).
Das Ergebnis des Aufrufs der privaten Methode `wasLiefereIch` wird in der Liste `liste` gespeichert (Zeile 3). Sofern diese nicht leer ist (Zeile 4), werden die Messwerte des ersten (Zeilen 5 und 6) und des letzten (Zeilen 7 und 8) Elements – also der kleinste und größte Messwert – als `wert1` und `wert2` gespeichert. Sofern sich die beiden Werte um mehr als `pWert` unterscheiden, wird `b` auf true gesetzt (Zeilen 9 bis 11).

Insgesamt prüft die Methode `istEtwas`, ob die Differenz aus dem größten und kleinsten Messwert über einem Schwellenwert `pWert` liegt.

Teilaufgabe e)

Eine mögliche Modellerweiterung könnte wie folgt aussehen:



(Die Darstellung beschränkt sich auf notwendige Änderungen im Vergleich zum in Abbildung 2 gegebenen Implementationsdiagramm.)

Um den individuellen (d. h. benutzerabhängigen) Zugriff zu realisieren, verwaltet jedes Objekt der Klasse `Benutzer` eine Liste aller von ihm steuerbaren Heizkörper. Zusätzlich verfügt die Klasse `Benutzer` über Methoden, um das Zugriffsrecht auf einen Heizkörper hinzuzufügen (`fuegeHinzu`) bzw. zu entfernen (`entferne`), sowie über eine Methode zum Testen der Berechtigung (`istBerechtigt`).
Bevor ein an den Server gesendeter Befehl ausgeführt wird, muss dieser überprüfen, ob der den Befehl sendende Benutzer den Heizkörper steuern darf (z. B. über die entsprechend hinzugefügte Methode der Klasse `Benutzer`). Zusätzlich erhält der Server Methoden, um einem Objekt der Klasse `Benutzer` Rechte zur Steuerung eines Objekts der Klasse `Heizkoerperdatensatz` zu erteilen (`fuegeRechteHinzu`) oder zu entziehen (`entferneRechte`), die jeweils auf die neu hinzugefügten Methoden der Klasse `Benutzer` zurückgreifen.

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 die beispielhafte Kommunikation zwischen Client und Server geeignet tabellarisch dar.	6			
2	erweitert das Kommunikationsprotokoll unter Berücksichtigung zweier neuer Fehlerfälle um die Möglichkeit, alle Geräte des gleichen Typs gleichzeitig mit einem Befehl zu steuern.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe a)	**9**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert den Teil der Methode `processMessage`, der den Protokollbefehl `STATUS` umsetzt.	6			
2	berücksichtigt dabei den Fall, dass der Benutzer nicht angemeldet ist.	2			
3	berücksichtigt dabei den Fall, dass die Anzahl der Parameter fehlerhaft ist.	2			
4	berücksichtigt dabei den Fall, dass das Gerät nicht bekannt ist.	2			
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	entwirft einen Algorithmus, der die Arbeitsweise der Methode `sindAlleImZustand` realisiert.	4			
2	implementiert die Methode `sindAlleImZustand` gemäß Algorithmus und Dokumentation.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe c)	**9**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert die Arbeitsweise und die Funktionalität der Methode `wasLiefereIch` im Sachzusammenhang.	5			
2	erläutert die Arbeitsweise und die Funktionalität der Methode `istEtwas` im Sachzusammenhang.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt in Anlehnung an das Implementationsdiagramm aus Abbildung 2 eine Modellierung, mit der die genannten Aspekte einer individualisierten Zugriffssteuerung realisiert werden.	5			
2	beschreibt, wie die einzelnen Aspekte in der Modellierung umgesetzt sind.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe e)	**10**			

	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 2019

Informatik, Leistungskurs

Aufgabenstellung:

Ein Großteil des Handels mit Computerspielen läuft heutzutage über sogenannte Online-Vertriebsplattformen. Hat man sich bei dem entsprechenden Anbieter als Benutzerin oder Benutzer angemeldet, kann man aus einem großen Angebot von Spielen auswählen. Hat man ein Spiel gekauft, kann man seine Kopie des Spiels mit Hilfe eines Internetzugangs beliebig oft auf seinem Rechner installieren bzw. deinstallieren.
Im Folgenden soll eine relationale Datenbank für eine einfache Vertriebsplattform für Computerspiele entworfen werden. Abbildung 1 zeigt eine Teilmodellierung dieser Datenbank.

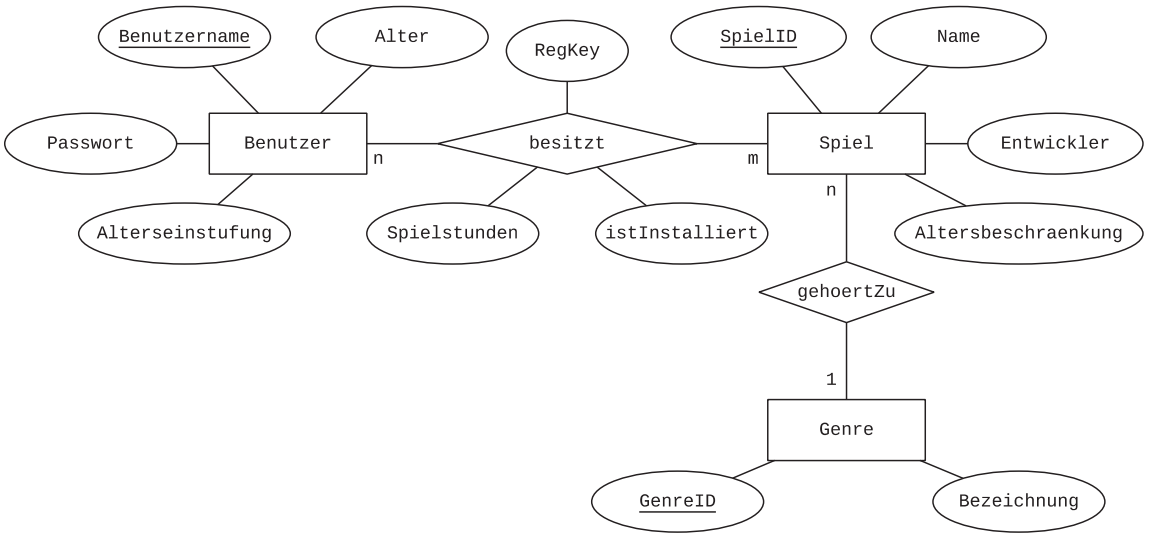


Abbildung 1: Teilmodellierung einer Vertriebsplattform für Computerspiele

Das folgende Datenbankschema setzt die Modellierung aus Abbildung 1 um:

```
Benutzer(Benutzername, Passwort, Alterseinstufung, Alter)
besitzt(↑Benutzername, ↑SpielID, RegKey, Spielstunden,
                                                istInstalliert)
Spiel(SpielID, Name, Entwickler, Altersbeschraenkung, ↑GenreID)
Genre(GenreID, Bezeichnung)
```

Abbildung 2: Datenbankschema zur Teilmodellierung aus Abbildung 1

Name: _____

a) *Erläutern Sie am Beispiel der Modellierung in Abbildung 1 und des Datenbankschemas in Abbildung 2, wie aus einem Entity-Relationship-Diagramm ein Datenbankschema entwickelt wird, und gehen Sie dabei insbesondere auf die Umsetzung der Beziehungstypen ein.*

(8 Punkte)

b) Die folgende SQL-Anweisung wird auf dem Datenbankschema in Abbildung 2 ausgeführt.

```
 1  SELECT AVG(tmp.C) AS A
 2  FROM (
 3    SELECT Genre.GenreID, COUNT(Spiel.SpielID) AS C
 4    FROM besitzt
 5      INNER JOIN Spiel
 6        ON besitzt.SpielID = Spiel.SpielID
 7      INNER JOIN Genre
 8        ON Spiel.GenreID = Genre.GenreID
 9    GROUP BY Genre.GenreID
10  ) AS tmp
```

Analysieren Sie die obige SQL-Anweisung und erläutern Sie zunächst die Unterabfrage von Zeile 3 *bis Zeile* 9 *und anschließend die gesamte SQL-Anweisung.*

Erläutern Sie, welche Information die Anweisung im Sachzusammenhang ermittelt.

Hinweis: Beispieldaten zum Datenbankschema in Abbildung 2 finden Sie in der Anlage.

(12 Punkte)

c) Die folgenden Anfragen sollen ebenfalls auf dem Datenbankschema in Abbildung 2 realisiert werden.

(i) Gesucht sind die Namen aller Benutzer, die 16 Jahre oder jünger sind.
(ii) Gesucht sind alle Benutzer, die das Spiel „SimTown“ mehr als 300 Stunden lang gespielt haben. Es sollen ihre Namen, die jeweiligen Spielstunden für dieses Spiel und der RegKey ermittelt werden.
(iii) Gesucht sind in alphabetisch aufsteigender Reihenfolge die Namen aller Spiele, die dem Benutzer mit dem Namen „Baldur80“ noch in seiner Sammlung fehlen.

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

(12 Punkte)

Name: _____

d) Für eine erweiterte Version soll die Vertriebsplattform um Social-Media-Komponenten ergänzt werden.
In einer ersten Version sollen Benutzerinnen und Benutzer anderen Benutzerinnen und Benutzern Chat-Nachrichten schicken können.
Damit man signalisieren kann, ob man gerade bereit ist, die Chat-Funktion zu nutzen, soll aus mehreren vorgegebenen Alternativen ein Status auswählt werden können, der anderen angezeigt wird (z. B. online, abwesend, beschäftigt usw.).

Modellieren Sie die erweiterte Datenbank als Entity-Relationship-Diagramm, indem Sie den Entitätstypen `Benutzer` *aus Abbildung 1 übernehmen und die erforderlichen neuen Entitätstypen und Beziehungstypen mit Attributen und Kardinalitäten ergänzen.*

Erläutern Sie, wie die zusätzlichen Anforderungen in Ihrer Modellierung umgesetzt sind.

(10 Punkte)

e) Die folgende Behauptung im Kontext der Modellierung einer relationalen Datenbank soll auf ihren Wahrheitsgehalt untersucht werden:

Einen n:m-Beziehungstypen braucht man zur Modellierung einer relationalen Datenbank gar nicht. Jeder n:m-Beziehungstyp kann durch zwei 1:n-Beziehungstypen ersetzt werden.

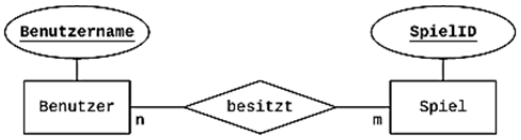


Abbildung 3

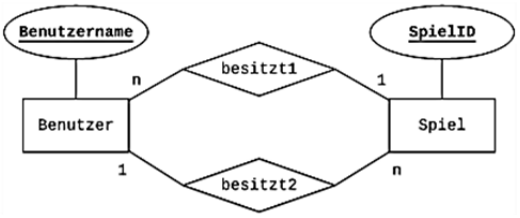


Abbildung 4

Die Modellierungen in Abbildung 3 und Abbildung 4 sind also gleichwertig und können beide mit dem folgenden Datenbankschema umgesetzt werden:

`Benutzer``(Benutzername, ↑SpielID)`
`Spiel``(SpielID, ↑Benutzername)`

Entwerfen Sie für das Relationenschema `Benutzer` *und für das Relationenschema* `Spiel` *jeweils eine Beispielrelation mit zwei Datensätzen, die zum Ausdruck bringen, dass beide Benutzer beide Spiele besitzen.*

Begründen Sie, warum die Behauptung, das obige Datenbankschema setze beide Modellierungen um, im Allgemeinen nicht zutrifft.

(8 Punkte)

Zugelassene Hilfsmittel:

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anlage:

Beispieldaten zum Datenbankschema aus Abbildung 2

Benutzer			
Benutzername	**Passwort**	**Alterseinstufung**	**Alter**
Trooper77	§k(652J!	18	22
Baldur80	#)§6dh%“6	16	17
I_will_win123	<ß§&JG51§KK.1&512!	12	12
Averroes79	5%4dK@72/§2n%	18	39

Hinweise: Passwörter werden verschlüsselt in der Datenbank abgelegt. Das Attribut `Alterseinstufung` kann die Werte `0`, `6`, `12`, `16` und `18` annehmen. Jede Person in der Relation `Benutzer` wird entsprechend ihres Alters in die höchstmögliche Alterseinstufung eingeordnet.

Spiel				
SpielID	**Name**	**Entwickler**	**Altersbeschraenkung**	**GenreID**
1	SimTown	Hellhackers & Co	6	5
2	Older Scribbles 5: Skyedge	Mayland Softworks	16	3
3	Need for Quickness 6	Electronic Crafts	12	1

besitzt				
Benutzername	**SpielID**	**RegKey**	**Spielstunden**	**istInstalliert**
Averroes79	2	GDH5-H652-HQQP-BG27	312	TRUE
Baldur80	3	K55G-HBD1-QQ2K-JH12	5	FALSE
Trooper77	1	J7GB-BGG1-GDOQ-77R4	11	TRUE
Baldur80	2	LQUF-7GD5-HQ77-HS2E	122	TRUE

Genre	
GenreID	**Bezeichnung**
1	Sportspiel
2	Ego-Shooter
3	Rollenspiel
4	Lernspiel
5	Simulation

Unterlagen für die Lehrkraft

Abiturprüfung 2019

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Abfrage relationaler Datenbanken

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2019

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
   - Datenbanken

   Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - SQL

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

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)

Um aus einem Entity-Relationship-Diagramm ein Datenbankschema zu entwickeln, wird zu jedem Entitätstyp ein Relationenschema erstellt. In diesem Relationenschema werden dieselben Attribute eingetragen wie im entsprechenden Entitätstyp. Auch die Primärschlüssel werden übernommen. Im Fall der in Abbildung 1 gegebenen Modellierung wird so mit den Entitätstypen `Benutzer`, `Spiel` und `Genre` verfahren.

Um 1:n-Beziehungstypen umzusetzen, kann der Primärschlüssel an der 1-Seite in das Relationenschema an der n-Seite als Fremdschlüssel eingetragen werden. Im gegebenen Beispiel wird für die Umsetzung des Beziehungstyps `gehoertZu` also der Primärschlüssel des Relationenschemas `Genre` als Fremdschlüssel in das Relationenschema `Spiel` aufgenommen.

Jeder n:m-Beziehungstyp wird durch ein eigenes Relationenschema realisiert. Die Attribute des Beziehungstyps werden zusammen mit den Primärschlüsseln beider beteiligter Entitätstypen aufgenommen. Diese Fremdschlüssel werden in der Regel als kombinierter Primärschlüssel der Beziehungsrelation verwendet. Im Fall der gegebenen Modellierung wird so mit dem Beziehungstyp `besitzt` verfahren.

Teilaufgabe b)

Die Unterabfrage von Zeile `3` bis Zeile `9` führt in den Zeilen `4` bis `8` zwei Verbundoperationen (`INNER JOIN`) zwischen den Relationen `besitzt`, `Spiel` und `Genre` durch. In Zeile `3` wird eine Projektion auf die IDs der Genres und die Anzahl der dazugehörigen Spielkopien durchgeführt, die mit dem Alias `C` versehen werden. Da eine Aggregatfunktion (`COUNT`) verwendet wird, wird in Zeile `9` nach den IDs der Genres gruppiert.
Die Unterabfrage ermittelt für jedes Genre seine ID und die Anzahl der Spielkopien, die im jeweiligen Genre einem Benutzer zugeordnet sind, d. h. verkauft wurden.

Die äußere SQL-Anweisung errechnet in Zeile `1` den Durchschnittswert (`AVG`) dieser von der Unterabfrage errechneten Werte `C`. Im Sachzusammenhang wird also die durchschnittliche Anzahl von Spielkopien ermittelt, die in einem Genre verkauft wurden.

Teilaufgabe c)

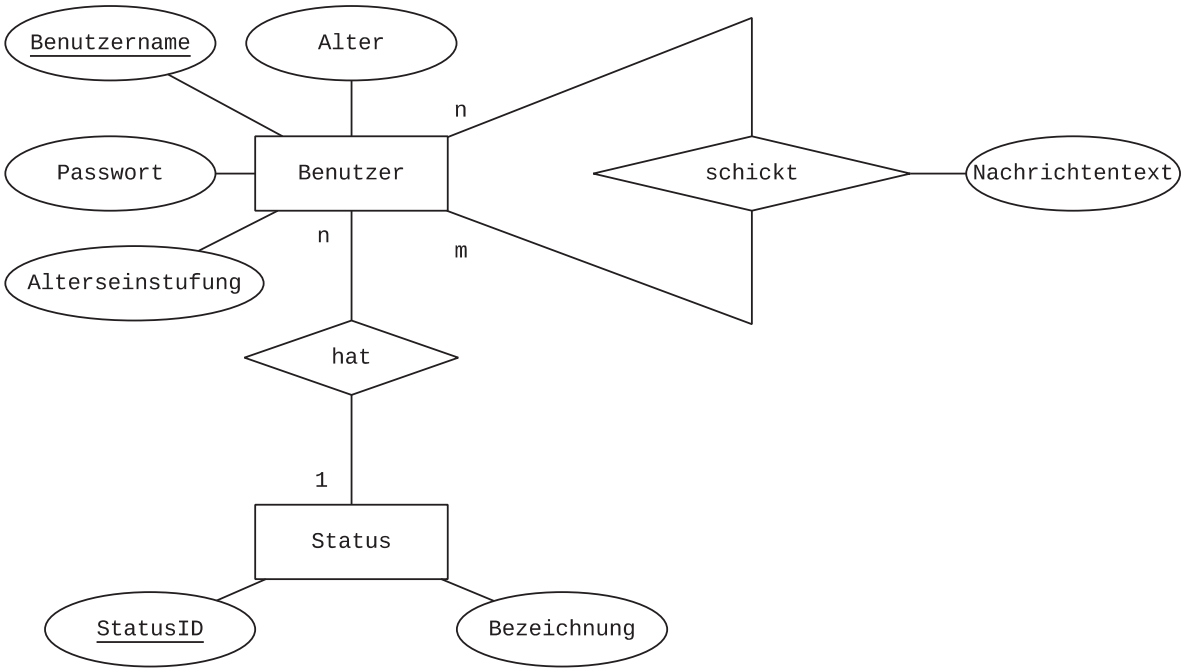
(i)
```
SELECT Benutzer.Benutzername
FROM Benutzer
WHERE Benutzer.Alter <= 16
```

(ii)
```
SELECT Benutzer.Benutzername, besitzt.RegKey, besitzt.Spielstunden
FROM Benutzer
  INNER JOIN besitzt
    ON Benutzer.Benutzername = besitzt.Benutzername
  INNER JOIN Spiel
    ON besitzt.SpielID = Spiel.SpielID
WHERE Spiel.Name = "SimTown" AND
      besitzt.Spielstunden > 300
```

(iii)
```
SELECT Spiel.Name
FROM Spiel
WHERE Spiel.SpielID NOT IN (
  SELECT besitzt.SpielID
  FROM besitzt
  WHERE besitzt.Benutzername = "Baldur80"
)
ORDER BY Spiel.Name
```

Teilaufgabe d)

Die folgende Datenbankmodellierung setzt die zusätzlichen Anforderungen um:



Um Nachrichten verschicken zu können, wird der n:m-Beziehungstyp `schickt` ergänzt. Datensätze der entsprechenden Beziehungsrelation stellen jeweils eine Nachricht zwischen zwei Benutzern dar. Daher erhält dieser Beziehungstyp auch noch den Text der Chat-Nachricht (`Nachrichtentext`) als Attribut.

Der Entitätstyp `Status` wird mit `StatusID` und einer `Bezeichnung` ergänzt. Zwischen `Benutzer` und `Status` wird mit dem Beziehungstypen `hat` ein 1:n-Beziehungstyp modelliert, da jeder Benutzer immer genau einen Status hat.

Teilaufgabe e)

Folgende Beispielrelationen bringen zum Ausdruck, dass beide Benutzer beide Spiele besitzen:

Benutzer	
Benutzername	**SpielID**
Trooper77	1
Baldur80	2

Spiel	
SpielID	**Benutzername**
1	Baldur80
2	Trooper77

Das folgende Argument zeigt, dass die Behauptung im Allgemeinen nicht zutrifft:

In dem gegebenen Datenbankschema steht jeder Eintrag eines Fremdschlüssels für eine Beziehung zwischen zwei Entitäten der Entitätstypen `Benutzer` und `Spiel`. Umfasst die Entitätsmenge `Benutzer` z. B. drei Elemente und die Entitätsmenge `Spiel` ebenfalls drei Elemente, so können auf diese Weise maximal sechs Beziehungen zwischen zwei Entitäten eingetragen werden. Sollen in diesem Fall aber alle Entitäten der Entitätsmenge `Benutzer` mit allen Entitäten der Entitätsmenge `Spiel` in Beziehung gesetzt werden, wie es bei einem n:m-Beziehungstypen möglich sein sollte, so müssten drei mal drei, d. h. neun Beziehungen eingetragen werden. Das gegebene Datenbankschema stellt also im Allgemeinen keine Umsetzung eines n:m-Beziehungstypen dar.

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 am Beispiel, wie aus einem Entity-Relationship-Diagramm ein Datenbankschema entwickelt wird und geht dabei auf die Umsetzung der Beziehungstypen ein.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert und erläutert die Unterabfrage.	6			
2	analysiert und erläutert die gesamte SQL-Anweisung.	3			
3	erläutert, welche Information die Anweisung im Sach-zusammenhang ermittelt.	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	entwirft eine SQL-Anweisung für die erste Anfrage.	3			
2	entwirft eine SQL-Anweisung für die zweite Anfrage.	4			
3	entwirft eine SQL-Anweisung für die dritte Anfrage.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe c)	**12**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	modelliert die erweiterte Datenbank als Entity-Relationship-Diagramm.	6			
2	erläutert, wie die zusätzlichen Anforderungen in der Modellierung umgesetzt sind.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft jeweils eine Beispielrelation.	4			
2	begründet, warum die Behauptung im Allgemeinen nicht zutrifft.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe e)	**8**			

	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 2019

Informatik, Leistungskurs

Aufgabenstellung:

Mit sogenannten Strichcodes ist es möglich, binäre Codes optisch darzustellen. Für eine 1 wird ein schwarzer Streifen verwendet, für eine 0 ein weißer Streifen.
Beim Lesen des Strichcodes kann es zu Lesefehlern kommen. Es wird statt einer `0` eine `1` gelesen oder umgekehrt. Eine Methode zur Erkennung der Fehler besteht darin, dass die Bitfolge, die gelesen werden soll, durch ein letztes Prüfbit ergänzt wird.

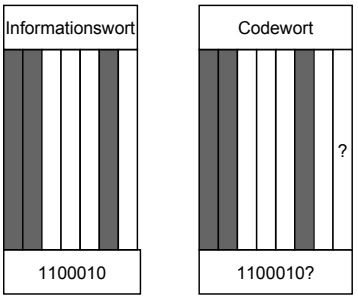


Abbildung 1:
Informationswort und Codewort

Zur Überprüfung, ob das Codewort korrekt gebildet ist, wird hier ein endlicher Automat A benutzt, der durch einen Übergangsgraphen dargestellt wird:

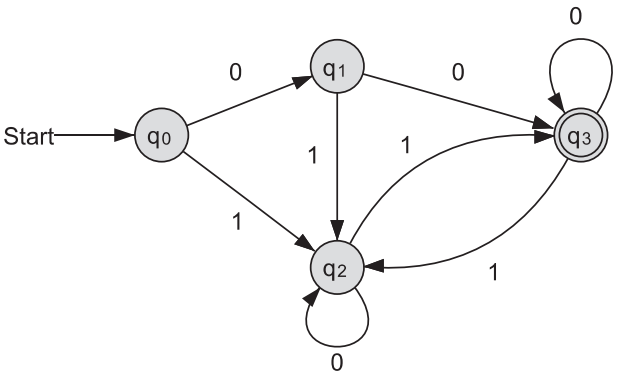


Abbildung 2: Zustandsübergangsdiagramm Automat A

a) *Zeigen Sie, dass das Codewort* `01101101001` (Informationswort und Prüfbit) *vom Automaten* `A` *akzeptiert wird.*

Erläutern Sie, wieso der Automat `A` *in der Lage ist zu prüfen, ob die Anzahl der Einsen gerade oder ungerade ist.*

Erläutern Sie einen Lesefehler, der durch die Anwendung dieses Automaten erkannt wird, und einen Lesefehler, der durch die Anwendung dieses Automaten nicht erkannt wird.

(6 Punkte)

Name: _____

b) Es werden nun nicht-leere Informationsworte gerader Länge betrachtet. Um mehr Lesefehler zu erkennen, wird jeweils nach zwei Bit des Informationswortes ein Prüfbit ergänzt. Das Bit wird so gewählt, dass die drei Bit eine ungerade Anzahl von Einsen enthalten.
Beispiel: Zur besseren Lesbarkeit werden Leerzeichen ergänzt.
Das Informationswort `100011` (`10 00 11`) führt zum Codewort `100001111` (`100 001 111`).

Entwerfen Sie einen endlichen Automaten A_1*, der nur die Codewörter akzeptiert, die nach dem oben beschriebenen Verfahren gebildet werden.*

(8 Punkte)

c) Durch die Java-Klasse `Codetester` soll überprüft werden, ob ein Lesefehler beim Strichcode für Ziffern vorliegt. Zur Prüfung wird die Bitfolge benutzt. Falls die zugehörige Bitfolge nicht die Länge sieben hat oder die zugehörige Bitfolge nicht vom Automat A (Abbildung 2) akzeptiert wird, liegt ein Lesefehler vor. Die Methode `hatLesefehler`, die nur `true` liefert, wenn ein Fehler vorliegt, hat folgenden Methodenkopf:

```
boolean hatLesefehler(String pCodewort).
```

Im Anhang sind einige Stringmethoden dokumentiert.

Implementieren Sie die Methode `hatLesefehler` *der Klasse* `Codetester`.

(10 Punkte)

d) Im Folgenden wird ein Code zur Zifferndarstellung benutzt. Jede Ziffer wird durch eine Folge aus sieben Bit definiert. Im Strichcode wird die 1 durch einen schwarzen Streifen und die 0 durch weißen Streifen dargestellt. Benachbarte gleichfarbige Streifen werden zu Balken zusammengefasst, die unterschiedliche Breite haben können. Der Stichcode für eine Ziffer besteht aus zwei schwarzen und zwei weißen Balken. Die Kennzeichnung gibt die Balkenbreiten an.
Beispiel:

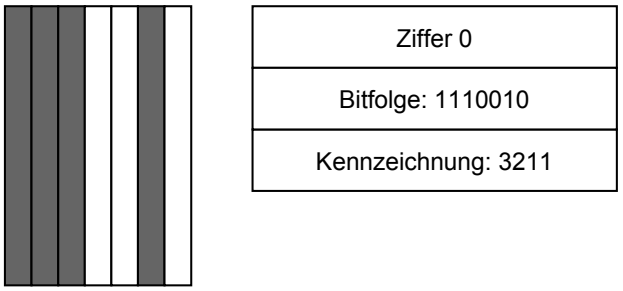


Abbildung 3: Strichcode für Ziffern

Name: _____

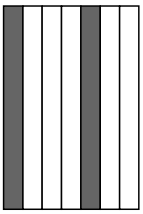
Von links nach rechts steht ein schwarzer Balken dreifacher Breite (3), ein weißer Balken doppelter Breite (2), dann ein schwarzer Balken einfacher Breite (1) und ein weißer Balken einfacher Breite (1). Damit hat der Strichcode die Kennzeichnung `3211`.

Die folgende Grammatik `G = (N,T,S,P)` erzeugt die Sprache, die die oben beschriebenen Kennzeichnungen des Codes enthält. Damit die Produktionen nicht zu umfangreich werden, werden durch die Grammatik G die Kennzeichnungen nur für einige Ziffern erzeugt.

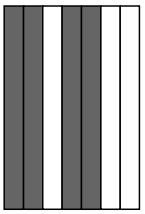
Startsymbol: `S`
Nichtterminale: `N = {S, A, B, C, D, E, X, Y, Z}`
Terminalsymbole: `T = {1, 2, 3}`

Produktionen:

```
P = {
    S → A B | A E | C D | D C,
    A → 1 Y,
    B → 1 Z,
    C → 2 X,
    D → 2 Y,
    E → 3 X,
    X → 1,
    Y → 2,
    Z → 3
    }
```



Ziffer 7
Bitfolge: 1000100
Kennzeichnung: 1312



Ziffer 2
Bitfolge: 1101100
Kennzeichnung: 2122

Abbildung 4: Strichcode Ziffer 7 und Ziffer 2

Name: _____

Zeigen Sie, dass die Kennzeichnung für die Ziffer 7 *nicht zur Sprache von* G *gehört und die Kennzeichnung der Ziffer* 2 *zur Sprache von* G *gehört.*

Entwickeln Sie eine Grammatik G_1*, deren Sprache die Sprache von* G *umfasst erweitert um die Kennzeichnung von Ziffer 7 und geben Sie dazu nur die Produktionen an, die verändert oder ergänzt werden.*

Begründen Sie, dass die Grammatik G *nicht regulär ist.*

Entwerfen Sie eine reguläre Grammatik G_2*, aus der sich genau die Wörter ableiten lassen, die sich aus der Grammatik* G *ableiten lassen.*

(16 Punkte)

e) Ein beliebig langes Informationswort besteht aus einer geraden Anzahl von Binärziffern.

Folgende Behauptung wird aufgestellt:
Wenn man das Informationswort vollständig oder blockweise in Zweierblöcken wiederholt, kann man endliche Automaten konstruieren, die alle Lesefehler beim Einlesen erkennen.

Beispiele:

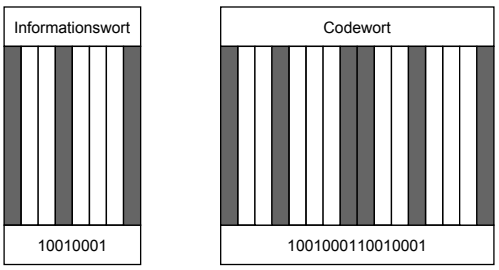


Abbildung 5: Vollständige Wiederholung

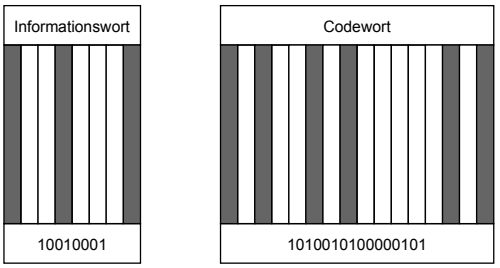


Abbildung 6: Blockweise Wiederholung

Beurteilen Sie die Behauptung.

(10 Punkte)

Zugelassene Hilfsmittel:

- Taschenrechner (grafikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Die Klasse String

Ein Objekt der Klasse String repräsentiert eine Zeichenkette.

Ausschnitt aus der Klasse Dokumentation der Klasse String

Anfrage **`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.

Anfrage **`String substring(int beginIndex)`**

Gibt eine neue Zeichenkette zurück, die eine Unter-Zeichenkette dieser Zeichenkette ist. Die Unter-Zeichenkette beginnt mit dem Zeichen, das sich am angegebenen Index befindet (beginnend bei Index `0`) und reicht bis zum Ende dieser Zeichenkette.
Beispiele:

- `"unschön".substring(2)` gibt `"schön"` zurück
- `"Leere".substring(5)` gibt `""` zurück (eine leere Zeichenkette)

Anfrage **`int length()`**

gibt die Anzahl der Zeichen in der Zeichenkette zurück.

Unterlagen für die Lehrkraft

Abiturprüfung 2019

Informatik, Leistungskurs

1. Aufgabenart

Analyse und Modellierung von kontextbezogenen Problemstellungen mit Schwerpunkt auf dem Inhaltsfeld formale Sprachen und Automaten

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2019

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*
   Formale Sprachen und Automaten
   - Endliche Automaten
     - Deterministische endliche Automaten
     - Nichtdeterministische endliche Automaten
   - Grammatiken regulärer Sprachen
   - Möglichkeiten und Grenzen von Automaten und formalen Sprachen

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Taschenrechner (graphikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

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)

01101101001

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_3 \xrightarrow{1} q_2 \xrightarrow{0} q_2 \xrightarrow{0} q_2 \xrightarrow{1} q_3$

q_3 ist ein Endzustand, das Codewort wird akzeptiert.

Das Prüfbit wird so ergänzt, dass die Anzahl der Einsen im Codewort gerade ist. Zum Zustand q_2 kommt man zum ersten Mal durch folgende Bitfolgen: 1, 01, 001.
Von q_2 kommt man mit einer 1 zum Endzustand q_3. In diesen Fällen ist also die Anzahl der Einsen gerade. Von q_3 kommt man durch 0 oder durch 11 wieder zu q_3, wobei zwischen den beiden Einsen beliebig viele Nullen stehen können. Die Anzahl der Einsen bleibt also gerade.

Da das Verändern eines einzelnen Bit zu einer falschen Parität führt, können solche Fehler beim Einlesen erkannt werden. Werden zwei Bit falsch gelesen, so ergibt sich wieder die korrekte Parität, der Fehler wird nicht erkannt.
Verallgemeinernd kann bei einer ungeraden Anzahl an Bitfehlern das fehlerhafte Einlesen erkannt werden, bei einer geraden Anzahl an Bitfehlern bleibt das fehlerhafte Einlesen unbemerkt.

Teilaufgabe b)

Endlicher deterministischer Automat A_1:

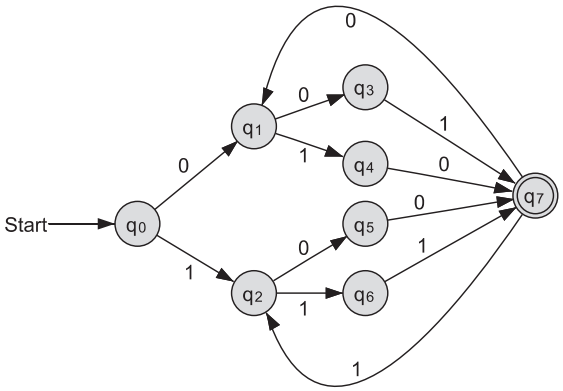
Zustandsmenge: $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

Eingabealphabet: $\{0, 1\}$

Startzustand: q_0

Menge der Endzustände: $\{q_7\}$

Zustandsübergangsdiagramm:



Teilaufgabe c)

Der Test kann wie folgt implementiert werden:

```
private boolean hatLesefehler(String pCodewort) {
  int zustand = 0;
  int index = 0;
  if (pCodewort.length() != 7) {
    zustand = - 1;
  }
  while (zustand != -1 && index < 7) {
    int zeichen = pCodewort.charAt(index);
    switch (zustand) {
      case 0: {
        switch (zeichen) {
          case '0': zustand = 1; break;
          case '1': zustand = 2; break;
        } break;
      }
      case 1: {
        switch (zeichen) {
          case '0': zustand = 3; break;
          case '1': zustand = 2; break;
        } break;
      }
      case 2: {
        switch (zeichen) {
          case '0': zustand = 2; break;
          case '1': zustand = 3; break;
        } break;
      }
      case 3: {
        switch (zeichen) {
          case '0': zustand = 3; break;
          case '1': zustand = 2; break;
        } break;
      }
      default: zustand = -1;
    }
    index++;
  }
  return zustand != 3;
}
```

Teilaufgabe d)

Die Ziffer 7 hat die Kennzeichnung 1312.
S → AB → 1YB
1312 gehört nicht zur Sprache, da es bei der Ableitung keine Alternative gibt, die zu 13B führt.

Die Ziffer 2 hat die Kennzeichnung 2122.
S → CD → 2XD → 21D → 212Y → 2122
2122 gehört zur Sprache, wie die Ableitung zeigt.

Wenn man die Produktionen von G durch die Alternative B A ergänzt, gehört die Kennzeichnung 1312 der Ziffer 7 auch zur Sprache

S → A B | A E | D C | C D | B A,

Eine reguläre Grammatik kann entweder rechtslinear oder linkslinear sein. Bei einer rechtslinearen Grammatik haben alle Produktionen auf der rechten Seite ein Terminalsymbol oder ein Terminalsymbol gefolgt von einem Nichtterminalsymbol. Bei einer linkslinearen Grammatik haben alle Produktionen auf der rechten Seite entweder ein Terminalsymbol oder ein Nichtterminalsymbol gefolgt von einem Terminalsymbol.

Die Produktion S → A B | A E | D C | C D verstößt gegen diese Regel, denn auf der rechten Seite stehen bei jeder Alternative zwei Nichtterminalsymbole.

Die folgende rechtslineare Grammatik G_2 erzeugt die gleiche Sprache wie die Grammatik G:

Startsymbol: S
Nichtterminale: N = {S, A, B, C, D, E, F, G, H}
Terminalsymbole: T = {1, 2, 3}
Produktionen: P = {
S → 1 A | 2 B
A → 2 C
B → 1 E | 2 D
C → 1 F | 3 G
D → 2 G
E → 2 H
F → 3
G → 1
H → 2
}

Teilaufgabe e)

Bei der blockweisen Wiederholung werden auch Lesefehler nicht erkannt. Wenn in dem Block an einer Position und in der Wiederholung an der gleichen Position das Bit falsch gelesen wird, wird der Fehler nicht erkannt.

In einem Informationswort kann es folgende Zweierblöcke geben: `00, 01, 10, 11`. Das zugehörige Codewort hat dann die Viererblöcke `0000, 0101, 1010` und `1111`. Durch einen endlichen Automaten kann man überprüfen, ob diese Viererblöcke in dem Codewort vorkommen. Damit kann man einen endlichen Automaten konstruieren, mit dem man die Korrektheit des Codewortes bei blockweiser Wiederholung testen kann.

Es wird keine Aussage über die Länge des Informationswortes gemacht. Bei einer vollständigen Wiederholung der Information im Codewort ist es allgemein nicht möglich, mit einem endlichen Automaten zu überprüfen, ob die beide Informationswörter im Codewort identisch sind. Für jede Bitfolge müsste ein eigener Automat konstruiert werden.

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	zeigt, dass das angegebene Wort vom Automaten akzeptiert wird.	2			
2	erläutert, wieso der Automat prüft, ob die Anzahl der Einsen gerade oder ungerade ist.	2			
3	erläutert einen Lesefehler, der durch den Automaten erkannt wird, und einen anderen Lesefehler, der nicht erkannt wird.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (6) …… ……					
	Summe Teilaufgabe a)	**6**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft einen endlichen Automaten, der nur Codewörter akzeptiert, die nach dem beschriebenen Verfahren gebildet werden.	8			
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	implementiert die Methode `hatLesefehler`.	10			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe c)	**10**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	zeigt, dass ein Wort nicht zur Sprache der Grammatik gehört und dass ein Wort zur Sprache gehört.	4			
2	entwickelt eine neue Grammatik G_1 aus der Grammatik G, die den Anforderungen genügt.	2			
3	begründet, dass die Grammatik G nicht regulär ist.	2			
4	entwirft eine reguläre Grammatik G_2 mit den geforderten Eigenschaften.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (16) …… ……					
	Summe Teilaufgabe d)	**16**			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt und formuliert für den Fall der blockweisen Wiederholung, dass auch Fehler nicht erkannt werden können.	2			
2	begründet, dass es im Fall der blockweisen Wiederholung einen endlichen Automaten gibt, der auf Korrektheit prüft.	4			
3	beurteilt und formuliert, dass es für beliebig lange Informationswörter, die vollständig wiederholt werden, keinen endlichen Automaten gibt, der in jedem Fall überprüfen kann, ob der Scanvorgang fehlerfrei ist.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe e)	**10**			

	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