Name: _____

Abiturprüfung 2012

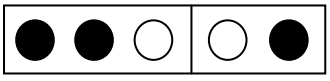
Informatik, Leistungskurs

Aufgabenstellung:

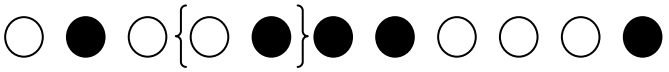
TAUSCH! ist ein Legespiel für 2 Personen. Die beiden Spieler erhalten jeweils 12 Steine (6 schwarze und 6 weiße), die sie zu Beginn in beliebiger Reihenfolge in einer Reihe vor sich hinlegen. Eine Reihe eines der beiden Spieler könnte z. B. so aussehen:

In einem Steinvorrat liegen weitere (unbegrenzt viele) Spielsteine beider Farben.

Zum Spiel gehören auch 64 Spielkarten. Die beiden Spieler bekommen jeweils drei davon, die restlichen liegen in durchmischter Reihenfolge übereinander in der Mitte im sogenannten „Kartenvorrat“. Auf jeder Spielkarte findet man eine graphisch dargestellte „Austauschregel“, z. B.:

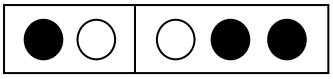


Die linke Seite zeigt die Steine an, die irgendwo in der eigenen Steinreihe so wie dargestellt nebeneinander liegen müssen. Der Spieler, der am Zug ist, darf diese Steingruppe ersetzen durch genau die Steingruppe, die auf der rechten Seite der Karte gezeigt wird. So könnte man in der oben dargestellten Reihe mit Hilfe dieser Spielkarte die Steine 4, 5 und 6 (von links gezählt) ersetzen, so dass sich die folgende neue Anordnung ergibt:

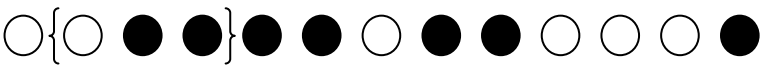


Der überflüssig gewordene schwarze Stein wandert in den Steinvorrat.

Ein Spieler darf eine solche Karte, bevor er sie benutzt, um 180 Grad drehen. Die oben dargestellte Karte erscheint dann so:



Benutzt man diese Karte für die Ausgangsreihe an den Positionen 2 und 3, ergibt sich:



Dazu musste der Spieler einen schwarzen Stein aus dem Steinvorrat nehmen.

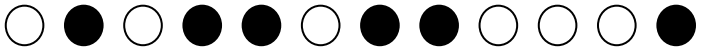
Ziel des Spiels ist es, als Erster eine Reihe beliebiger Länge zu bilden, die ausschließlich schwarze Steine enthält.

Name: _____

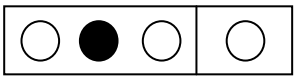
Der Spielablauf:

- Die beiden Spieler machen abwechselnd einen Zug:
  - Der Spieler, der an der Reihe ist, nimmt die oberste Karte vom Kartenvorrat, so dass er jetzt 4 Karten hat.
  - Er wählt aus diesen 4 Karten eine geeignete Karte aus, nimmt die entsprechende Ersetzung vor und legt diese Karte zurück unter den Kartenvorrat.
  - Kann er mit keiner seiner 4 Karten eine Ersetzung vornehmen, so legt er eine beliebige seiner 4 Karten zurück oben auf den Kartenvorrat (so dass der andere Spieler diese Karte als nächstes erhält).

a) Vor dem ersten Spieler liegt die Steinreihe



Er benutzt die Karte (ggf. auch in gedrehter Form):



Geben Sie drei verschiedene Endreihen an, die entstehen können.

(4 Punkte)

Im Folgenden sollen einige Aspekte dieses Spiels für den Computer modelliert werden.

b) Der oben beschriebene Kartenvorrat soll mit einer Klasse `Kartenvorrat` simuliert werden. In ihr sollten Objekte vom Typ `Spielkarte` verwaltet werden. Die Klasse `Kartenvorrat` soll folgende Methoden haben:

Konstruktor **`Kartenvorrat()`**
Ein neuer leerer Kartenvorrat wird erzeugt.

Auftrag **`void legeKarteDrauf(Spielkarte pKarte)`**
Legt die als Parameter übergebene Spielkarte oben auf den Kartenvorrat.

Auftrag **`void legeKarteDrunter (Spielkarte pKarte)`**
Legt die als Parameter übergebene Spielkarte unter den Kartenvorrat.

Geben Sie eine geeignete Datenstruktur für den Kartenvorrat an und begründen Sie Ihre Entscheidung, indem Sie Ihre Wahl gegen andere ungeeignete abgrenzen.

Implementieren Sie in dieser Klasse die beiden Methoden `legeKarteDrauf` und `legeKarteDrunter` *unter Benutzung der von Ihnen gewählten Datenstruktur.*

(16 Punkte)

Name: _____

c) Der Spieler, der am Zuge ist, muss testen, ob das Muster, das auf einer seiner Karten steht, in seiner Steinreihe vorkommt. Einen Ausschnitt aus der Dokumentation der Klassen `Stein` und `Steinreihe` finden Sie im Anhang.

Implementieren Sie in der Klasse `Steinreihe` *eine Methode* `istEnthalten` *mit dem Methodenkopf*

```
public boolean istEnthalten(Steinreihe pMuster)
```

die genau dann `true` *zurückliefert, wenn* `pMuster` *in dem aktuellen Objekt enthalten ist.*

(6 Punkte)

Name: _____

d) Hier finden Sie den Quellcode einer Methode `wasLiefereIch` aus der Klasse `Steinreihe`:

```
 1 public List wasLiefereIch(int pL) {
 2   List res = new List ();
 3   if (pL == 1) {
 4     Steinreihe reihe1 = new Steinreihe();
 5     Steinreihe reihe2 = new Steinreihe();
 6     reihe1.fuegeSteinHinzu(new Stein(true));
 7     reihe2.fuegeSteinHinzu(new Stein(false));
 8     res.append(reihe1);
 9     res.append(reihe2);
10     return res;
11   }
12   else {
13     List v = wasLiefereIch(pL-1);
14     v.toFirst();
15     while (v.hasAccess()) {
16       Steinreihe diese = (Steinreihe) v.getObject();
17       Steinreihe reihe1 = diese.kopie();
18       Steinreihe reihe2 = diese.kopie();
19       reihe1.fuegeSteinHinzu(new Stein(true));
20       reihe2.fuegeSteinHinzu(new Stein(false));
21       res.append(reihe1);
22       res.append(reihe2);
23       v.next();
24     }
25     return res;
26   }
27 }
```

Analysieren Sie den Quelltext, wenn Sie die Methode mit dem Parameter 3 aufrufen. Geben Sie an, welchen Wert die Methode zurückliefert.

Beschreiben Sie die Funktionalität der Methode im Sachzusammenhang.

(12 Punkte)

Name: _____

e) Der Spieler, der an der Reihe ist, hat in der Regel mehrere Möglichkeiten, Steine seiner Steinreihe auszutauschen, da mehrere seiner Muster passen.

Ziel ist die Entwicklung einer Methode `besteSpielkarte` in der Klasse `Steinreihe`:

Anfrage **`Spielkarte besteSpielkarte(List pKartenliste)`**
liefert aus der Liste `pKartenliste` (laut Spielregel sind das 4 Karten) diejenige Karte, die der Spieler benutzen sollte, um möglichst viele schwarze Steine in der Steinreihe zu erhalten.

Entwickeln und beschreiben Sie eine Strategie, um diese beste Spielkarte zu finden. Eine Implementation ist nicht erforderlich.

(12 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anlagen

Auszug aus der Dokumentation der Klasse `Spielkarte`

Ein Objekt dieser Klasse verwaltet eine Spielkarte. Auf einer solchen Spielkarte sind die Muster, die austauschbar sind, in Objekten der Klasse `Steinreihe` verwaltet.

Konstruktor `Spielkarte(Steinreihe pMusterLinks, Steinreihe pMusterRechts)`

Eine neue Spielkarte wird erzeugt. Sie enthält in Form von Steinreihen die beiden Muster, die die Regel zum Mustertausch repräsentieren.

Anfrage `Steinreihe gibLinks()`

liefert das linke Muster auf der Karte in Form einer Steinreihe.

Anfrage `Steinreihe gibRechts()`

liefert das rechts Muster auf der Karte in Form einer Steinreihe.

Auszug aus der Dokumentation der Klasse `Stein`

Ein Objekt dieser Klasse verwaltet einen Spielstein. Ein solcher Spielstein kann schwarz oder weiß sein.

Konstruktor `Stein(boolean pIstSchwarz)`

Ein neuer Spielstein wird erzeugt. Er ist genau dann schwarz, wenn `pIstSchwarz` den Wert `true` hat.

Anfrage `String gibFarbe()`

Die Anfrage liefert den Wert `"s"`, wenn der Stein schwarz ist, ansonsten wird `"w"` geliefert.

Name: _____

Auszug aus der Dokumentation der Klasse `Steinreihe`

Ein Objekt dieser Klasse verwaltet eine Liste von Steinen. Die Plätze, an denen Steine liegen, sind nummeriert mit den Zahlen `0,1,2,…` Ist die Anzahl der Steine einer Reihe `n`, so ist die größte Platznummer `n-1`. Es gibt u. a. folgende Methoden:

Konstruktor **`Steinreihe()`**
Eine leere Steinreihe wird erzeugt.

Anfrage **`int anzahlSteine()`**
Die Anfrage liefert die Anzahl der Steine, die in der Steinreihe verwaltet werden.

Anfrage **`Stein gibStein(int pIndex)`**
liefert den Stein, der in der Steinreihe an der Position `pIndex` liegt (`0 <= pIndex < anzahlSteine`).

Auftrag **`void fuegeSteinHinzu(Stein pStein)`**
fügt den als Parameter übergebenen Stein als letzten Stein an die Steinreihe an.

Anfrage **`Steinreihe kopie()`**
Die Methode erzeugt eine neue Steinreihe, die sich aus der gleichen Folge von schwarzen und weißen Spielsteinen zusammensetzt, wie die Original-steinreihe.

Name: _____

Die Klasse Stack

Objekte der Klasse **`Stack`** (Keller, Stapel) verwalten beliebige Objekte nach dem Last-In-First-Out-Prinzip, d. h., das zuletzt abgelegte Objekt wird als erstes wieder entnommen.

Dokumentation der Klasse Stack

Konstruktor **`Stack()`**

Ein leerer Stapel wird erzeugt.

Anfrage **`boolean isEmpty()`**

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

Auftrag **`void push(Object pObject)`**

Das Objekt `pObject` wird oben auf den Stapel gelegt. Falls `pObject` gleich `null` ist, bleibt der Stapel unverändert.

Auftrag **`void pop()`**

Das zuletzt eingefügte Objekt wird von dem Stapel entfernt. Falls der Stapel leer ist, bleibt er unverändert.

Anfrage **`Object top()`**

Die Anfrage liefert das oberste Stapelobjekt. Der Stapel bleibt unverändert. Falls der Stapel leer ist, wird `null` zurückgegeben.

Name: _____

Die Klasse Queue

Objekte der Klasse **`Queue`** (Warteschlange) verwalten beliebige Objekte nach dem First-In-First-Out-Prinzip, d. h., das zuerst abgelegte Objekt wird als erstes wieder entnommen.

Dokumentation der Klasse Queue

Konstruktor **`Queue()`**

Eine leere Schlange wird erzeugt.

Anfrage **`boolean isEmpty()`**

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

Auftrag **`void enqueue(Object pObject)`**

Das Objekt `pObject` wird an die Schlange angehängt. Falls `pObject` gleich `null` ist, bleibt die Schlange unverändert.

Auftrag **`void dequeue()`**

Das erste Objekt wird aus der Schlange entfernt. Falls die Schlange leer ist, wird sie nicht verändert.

Anfrage **`Object front()`**

Die Anfrage liefert das erste Objekt der Schlange. Die Schlange bleibt unverändert. Falls die Schlange leer ist, wird `null` zurückgegeben.

Name: _____

Die Klasse `List`

Objekte der Klasse **`List`** verwalten beliebig viele, linear angeordnete Objekte. 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 oder ein Listenobjekt an das Ende der Liste angefügt werden.

Dokumentation der Klasse `List`

Konstruktor **`List()`**
Eine leere Liste wird erzeugt.

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 **`Object getObject()`**

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 setObject(Object pObject)`**

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

Auftrag **`void append(Object pObject)`**

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

Auftrag **`void insert(Object pObject)`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`), wird ein neues Objekt 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 `pObject` 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 `pObject` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void concat(List pList)`**

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls `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 2012

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Modellierung einer Problemstellung, Entwurf und Implementation von Algorithmen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2012

1. *Inhaltliche Schwerpunkte*
   Konzepte des objektorientierten Modellierens
   - Lineare Strukturen mit den Akzenten
     - Lineare Liste
     - Schlange, Stapel

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

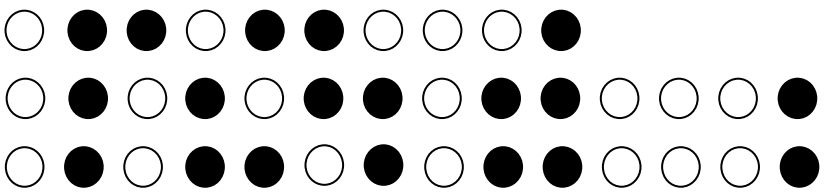
[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösungen

Teilaufgabe a)

Mögliche Endreihen neben weiteren sind:



Teilaufgabe b)

Da sowohl oben auf den Kartenvorrat als auch unter den Kartenvorrat eine Karte gelegt werden kann, scheiden Schlange und Stapel als Datenstruktur aus. Da die Größe des Kartenvorrates ständig variiert, ist auch ein Array nicht sinnvoll. Eine Liste ist hier sicherlich geeignet.

Eine mögliche Implementation könnte die folgende Form haben:

```
1 public List vorrat;

2 public void legeKarteDrauf(Spielkarte pKarte) {
3   vorrat.append(pKarte);
4 }

5 public void legeKarteDrunter(Spielkarte pKarte) {
6   vorrat.toFirst();
7   vorrat.insert(pKarte);
8 }
```

Teilaufgabe c)

Eine mögliche Lösung:

```
 1 public boolean istEnthalten(Steinreihe pMuster) {
 2   int meineAnzahl = anzahlSteine();
 3   int musterAnzahl = pMuster.anzahlSteine();
 4   boolean gefunden = false;

 5   int index = 0;
 6   while ((!gefunden) && (index <= (meineAnzahl - musterAnzahl))) {
 7     int i = 0;
 8     boolean identisch = true;
 9     while (identisch && (i < musterAnzahl)) {
10       if (pMuster.gibStein(i).istSchwarz() !=
                              this.gibStein(index + i).istSchwarz()) {
11         identisch = false;
12       }
13       i++;
14     }
15     index++;
16     gefunden = identisch;
17   }
18   return gefunden;
19 }
```

Teilaufgabe d)

Ruft man die Methode mit dem Parameterwert 3 auf, wird eine Liste mit 8 Steinreihen der Länge 3 geliefert:

`www, wws, wsw, wss, sww, sws, ssw, sss` (oder als Graphik angegeben).

Die Methode liefert eine Liste mit allen möglichen Steinreihen der Länge `pL`. Dazu werden rekursiv alle Steinreihen der Länge `pL-1` erzeugt und an alle so erzeugten Steinreihen ein schwarzer bzw. ein weißer Stein angehängt, so dass jeweils zwei neue Steinreihen erzeugt werden.

Teilaufgabe e)

Eine mögliche Lösungsstrategie:

Die als Parameter übergebene Liste von Spielkarten (nach Spielregel 4) wird quasi verdoppelt, indem auch die um 180 Grad gedrehten Karten hinzugefügt werden, so dass jetzt 8 Karten in einer Liste vorliegen. Die jeweils linken Muster dieser 8 Karten versucht man jetzt in der vorgegebenen Steinreihe durch die entsprechenden rechten Muster zu ersetzen. Das kann ggf. an keiner, einer oder an mehreren Stellen geschehen, so dass für jede der 8 Karten eine Liste von Ergebnissteinreihen entsteht. In jeder Steinreihe jeder dieser 8 Listen wird die Anzahl der schwarzen Steine bestimmt sowie das Maximum dieser Anzahlen. Die zugehörige Karte ist die gesuchte.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt (mindestens) drei verschiedene Endreihen an.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt eine geeignete Datenstruktur an.	4
2	begründet die Entscheidung.	4
3	implementiert die Methode `legeKarteDrauf`.	4
4	implementiert die Methode `legeKarteDrunter`.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	implementiert die Methode `istEnthalten`.	6
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	analysiert den Quelltext.	4
2	beschreibt die Funktionalität der Methode im Sachzusammenhang.	4
3	gibt den Rückgabewert an, wenn die Methode mit dem Parameter 3 aufgerufen wird.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe e)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	entwickelt und beschreibt eine problemgerechte Lösungsstrategie.	12
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**[2]	**ZK**	**DK**
1	gibt (mindestens) drei …	4			
	Summe Teilaufgabe a)	**4**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt eine geeignete Datenstruktur …	4			
2	begründet die Entscheidung.	4			
3	implementiert die Methode …	4			
4	implementiert die Methode …	4			
	Summe Teilaufgabe b)	**16**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Methode …	6			
	Summe Teilaufgabe c)	**6**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert den Quelltext.	4			
2	beschreibt die Funktionalität …	4			
3	gibt den Rückgabewert …	4			
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt und beschreibt …	12			
	Summe Teilaufgabe e)	**12**			

	Summe insgesamt	**50**			

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

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

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

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:

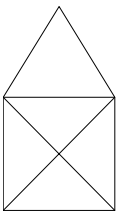
Note	**Punkte**	**Erreichte Punktzahl**
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 58
mangelhaft plus	3	57 – 49
mangelhaft	2	48 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0

Name: _____

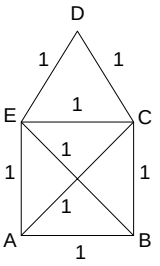
Abiturprüfung 2012

Informatik, Leistungskurs

Aufgabenstellung:



Das Haus des Nikolaus ist ein Zeichenrätsel. Ziel ist es, das spezielle Haus in einem Linienzug zu zeichnen, ohne eine der Strecken mehrfach zu durchlaufen. Begleitet wird das Zeichnen durch folgenden gesprochenen Reim: „Das ist das Haus vom Ni-ko-laus“.
Mathematisch gesehen handelt es sich um ein Problem der Graphentheorie. Von einem Knoten aus werden alle Kanten nach einer gewissen Vorschrift durchlaufen.
Im Folgenden sollen das Zeichenrätsel und ähnliche Probleme mit Hilfe von Programmen analysiert werden.
Das Haus des Nikolaus ist hier als Graph gekennzeichnet worden.
Die Knoten haben Namen. Alle Kanten erhalten das Gewicht 1.



a) *Stellen Sie den Graphen mit den angegebenen Knotenbezeichnungen und Kantengewichten als Adjazenzmatrix dar.*

Der Graph soll in der Klasse `NikoHaus` erzeugt und analysiert werden. Verwenden Sie die Graph-Klassen, die im Anhang dokumentiert sind.

Implementieren Sie den Konstruktor der Klasse `NikoHaus`, in dem der Graph (Haus des Nikolaus) erzeugt wird. (8 Punkte)

b) Gegeben ist die Methode `berechneUnbekannt` der Klasse `NikoHaus`. Die Methode ruft eine Methode `unbekannt` auf. In der Methode `unbekannt` werden Nachbarlisten gebildet. Sie können davon ausgehen, dass in den Nachbarlisten die Knoten nach Namen sortiert vorliegen.

NikoHaus
- nikoGraph: Graph
+ NikoHaus() + berechneUnbekannt(): String + gibGraph(): Graph ... - unbekannt(pKnoten: GraphNode, pWeg: String, pStufe: int, pTeilloesung: String): String

Name: _____

```
1 public String berechneUnbekannt() {
2   return unbekannt(nikoGraph.getNode("A"),"A",0,"");
3 }

4 private String unbekannt(GraphNode pKnoten, String pWeg,
                                int pStufe, String pTeilloesung) {
5   List nachbarListe = nikoGraph.getNeighbours(pKnoten);
6   nachbarListe.toFirst();
7   while (nachbarListe.hasAccess()) {
8     GraphNode nachbarKnoten =
                             (GraphNode) nachbarListe.getObject();
9     if (nikoGraph.hasEdge(pKnoten,nachbarKnoten)) {
10      if (nikoGraph.getEdgeWeight(pKnoten,nachbarKnoten) == 1) {
11        nikoGraph.removeEdge(pKnoten,nachbarKnoten);
12        nikoGraph.addEdge(pKnoten,nachbarKnoten,-1);
13        if (pStufe < 7){
14          pTeilloesung = unbekannt(nachbarKnoten,
       pWeg + nachbarKnoten.getName(), pStufe + 1, pTeilloesung);
15        }
16        else {
17          pTeilloesung = pTeilloesung + pWeg +
                                  nachbarKnoten.getName() + "\n";
18        }
19        nikoGraph.addEdge(pKnoten,nachbarKnoten,1);
20      }
21    }
22    nachbarListe.next();
23  }
24  return pTeilloesung;
25 }
```

Analysieren Sie die Methode bis bei der Abarbeitung zum ersten Mal die Zeile 17 erreicht wird, indem Sie Wertebelegungen der Parameter für jeden Methodenaufruf dokumentieren, und geben Sie die Zeichenkette an, die bis dahin erzeugt wird.

Erläutern Sie die Arbeitsweise der angegebenen Methoden.

(12 Punkte)

Name: _____

Der Mathematiker Leonard Euler (1707 – 1783), der eine Zeit lang an der Universität in Königsberg lehrte, hat u. a. auch an der Erforschung von Graphen gearbeitet. Daher sind einige Eigenschaften von Graphen mit seinem Namen verbunden.

Eulerkreis: Ein Eulerkreis ist ein Rundweg (Startknoten ist gleich Zielknoten) durch einen Graphen, der jede Kante genau einmal benutzt.

eulersch: Man nennt einen Graphen „eulersch“, wenn er einen Eulerkreis besitzt.

Leonard Euler fand in seinen Untersuchungen ein einfaches Kriterium zur Überprüfung der Eulereigenschaft eines Graphen heraus:

Ein Graph ist genau dann eulersch, wenn von jedem Knoten eine gerade Anzahl von Kanten abgeht.

c) *Erläutern Sie, dass im Graphen „Haus des Nikolaus“ kein Eulerkreis existieren kann.*

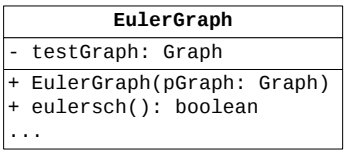
Entfernen Sie bei dem Graphen „Haus des Nikolaus“ eine oder mehrere Kanten, so dass der veränderte Graph eulersch ist.

Erweitern Sie den Graphen „Haus des Nikolaus“ um maximal einen Knoten und um weitere Kanten, so dass der erweiterte Graph eulersch ist.

Erläutern Sie Ihre Entscheidungen.

(10 Punkte)

Mit Hilfe der Klasse `EulerGraph` können u. a. Graphen untersucht werden, ob sie die Eigenschaft „eulersch“ haben.



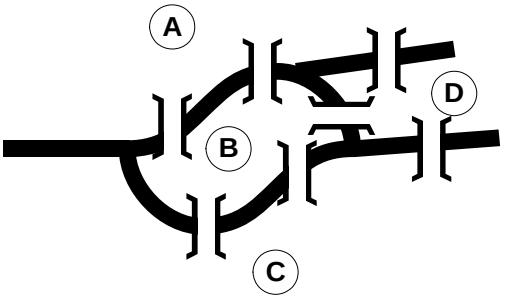
Im Text vor der Teilaufgabe c) ist ein Kriterium angegeben worden, wie man feststellen kann, ob ein Graph eulersch ist.

d) *Implementieren Sie die Methode* `eulersch` *der Klasse* `EulerGraph`, *die testet, ob der gegebene Graph eulersch ist oder nicht. Implementieren Sie die Methode und eventuell weitere notwendige Hilfsmethoden.*

(10 Punkte)

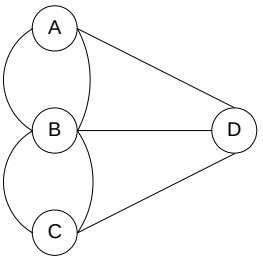
Name: _____

In Königsberg vereinen sich der Neue und der Alte Pregel so, dass sie dabei eine Insel umschließen (siehe Abbildung). Dabei entstehen 4 Stadtteile (hier mit A, B, C, D gekennzeichnet), welche durch die Flüsse getrennt werden. Diese waren zu Beginn des 18. Jahrhunderts durch sieben Brücken verbunden.



Die Professoren der Königsberger Universität wandelten damals gelegentlich über diese Brücken und einer von ihnen (LEONHARD EULER) wollte wissen, ob es einen Rundgang durch die Stadtteile gibt, bei dem jede Brücke genau einmal überquert wird. Diese Fragestellung ist unter dem Begriff „Königsberger Brückenproblem“ bekannt geworden.

Die nebenstehende Abbildung zeigt den Graphen zum Brückenproblem. Bei diesem Graphen existieren zwischen den Punkten A und B bzw. B und C jeweils zwei verschiedene Kanten. Graphen dieser Art werden auch als Multigraphen bezeichnet. Gesucht ist ein Programm, mit dem man Multigraphen der angegebenen Art prüfen kann, ob sie eulersch sind.



e) *Modellieren Sie eine Möglichkeit, diesen Multigraph mit Hilfe der dokumentierten Klassen* `Graph` *und* `GraphNode` *zu verwalten.*

Erläutern Sie grundsätzlich die Funktionsweise des Algorithmus, der entscheidet, ob der Multigraph eulersch ist.

Es soll keine Implementation angegeben werden.

(10 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anlagen

Graphen

Ein ungerichteter Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Die Kanten verbinden jeweils zwei Knoten und können ein Gewicht haben.

Die Klasse GraphNode

Objekte der Klasse **`GraphNode`** sind Knoten eines Graphen. Ein Knoten hat einen Namen und kann markiert werden.

Dokumentation der Klasse `GraphNode`

Konstruktor **`GraphNode(String pName)`**
Ein Knoten mit dem Namen `pName` wird erzeugt. Der Knoten ist nicht markiert.

Auftrag **`void mark()`**
Der Knoten wird markiert, falls er nicht markiert ist, sonst bleibt er unverändert.

Auftrag **`void unmark()`**
Die Markierung des Knotens wird entfernt, falls er markiert ist, sonst bleibt er unverändert.

Anfrage **`boolean isMarked()`**
Die Anfrage liefert den Wert `true`, wenn der Knoten markiert ist, sonst liefert sie den Wert `false`.

Anfrage **`String getName()`**
Die Anfrage liefert den Namen des Knotens.

Name: _____

Die Klasse Graph

Objekte der Klasse **`Graph`** sind ungerichtete, gewichtete Graphen. Der Graph besteht aus Knoten, die Objekte der Klasse `GraphNode` sind, und Kanten, die Knoten miteinander verbinden. Die Knoten werden über ihren Namen eindeutig identifiziert.

Dokumentation der Klasse `Graph`

Konstruktor **`Graph()`**

Ein neuer Graph wird erzeugt. Er enthält noch keine Knoten.

Anfrage **`boolean isEmpty()`**

Die Anfrage liefert `true`, wenn der Graph keine Knoten enthält, andernfalls liefert die Anfrage `false`.

Auftrag **`void addNode(GraphNode pNode)`**

Der Knoten `pNode` wird dem Graphen hinzugefügt. Falls bereits ein Knoten mit gleichem Namen im Graphen existiert, wird dieser Knoten nicht eingefügt. Falls `pNode` `null` ist, verändert sich der Graph nicht.

Anfrage **`boolean hasNode(String pName)`**

Die Anfrage liefert `true`, wenn ein Knoten mit dem Namen `pName` im Graphen existiert. Sonst wird `false` zurückgegeben.

Anfrage **`GraphNode getNode(String pName)`**

Die Anfrage liefert den Knoten mit dem Namen `pName` zurück. Falls es keinen Knoten mit dem Namen im Graphen gibt, wird `null` zurückgegeben.

Auftrag **`void removeNode(GraphNode pNode)`**

Falls `pNode` ein Knoten des Graphen ist, so werden er und alle mit ihm verbundenen Kanten aus dem Graphen entfernt. Sonst wird der Graph nicht verändert.

Auftrag **`void addEdge(GraphNode pNode1, GraphNode pNode2, double pWeight)`**

Falls eine Kante zwischen `pNode1` und `pNode2` noch nicht existiert, werden die Knoten `pNode1` und `pNode2` durch eine Kante verbunden, die das Gewicht `pWeight` hat. `pNode1` ist also Nachbarknoten von `pNode2` und umgekehrt. Falls eine Kante zwischen `pNode1` und `pNode2` bereits existiert, erhält sie das Gewicht `pWeight`. Falls einer der Knoten `pNode1` oder `pNode2` im Graphen nicht existiert oder `null` ist, verändert sich der Graph nicht.

Name: _____

Anfrage **`boolean hasEdge(GraphNode pNode1,GraphNode pNode2)`**

Die Anfrage liefert `true`, falls eine Kante zwischen `pNode1` und `pNode2` existiert, sonst liefert die Anfrage `false`.

Auftrag **`void removeEdge(GraphNode pNode1,GraphNode pNode2)`**

Falls `pNode1` und `pNode2` nicht `null` sind und eine Kante zwischen `pNode1` und `pNode2` existiert, wird die Kante gelöscht. Sonst bleibt der Graph unverändert.

Anfrage **`double getEdgeWeight(GraphNode pNode1, GraphNode pNode2)`**

Die Anfrage liefert das Gewicht der Kante zwischen `pNode1` und `pNode2`. Falls die Kante nicht existiert, wird `Double.NaN` (not a number) zurückgegeben.

Auftrag **`void resetMarks()`**

Alle Knoten des Graphen werden als unmarkiert gekennzeichnet.

Anfrage **`boolean allNodesMarked()`**

Die Anfrage liefert den Wert `true`, wenn alle Knoten des Graphen markiert sind, sonst liefert sie den Wert `false`. Wenn der Graph leer ist, wird `true` zurückgegeben.

Anfrage **`List getNodes()`**

Die Anfrage liefert eine Liste, die alle Knoten des Graphen enthält.

Anfrage **`List getNeighbours(GraphNode pNode)`**

Die Anfrage liefert eine Liste, die alle Nachbarknoten des Knotens `pNode` enthält.

Name: _____

Die Klasse `List`

Objekte der Klasse **`List`** verwalten beliebig viele, linear angeordnete Objekte. 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 oder ein Listenobjekt an das Ende der Liste angefügt werden.

Dokumentation der Klasse `List`

Konstruktor **`List()`**
Eine leere Liste wird erzeugt.

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 **`Object getObject()`**

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 setObject(Object pObject)`**

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

Auftrag **`void append(Object pObject)`**

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

Auftrag **`void insert(Object pObject)`**

Falls es ein aktuelles Objekt gibt (`hasAccess()` == `true`), wird ein neues Objekt 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 `pObject` 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 `pObject` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void concat(List pList)`**

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls `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 2012

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Modellierung einer Problemstellung, Entwurf und Implementation von Algorithmen auf Graphen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2012

1. *Inhaltliche Schwerpunkte*
   Objektorientiertes Modellieren und Implementieren von kontextbezogenen Anwendungen
   - Konzepte des objektorientierten Modellierens
     - Klasse, Objekt, Attribut, Methode, Geheimnisprinzip
     - Klassendiagramme
     - Beziehungen zwischen Klassen: (Gerichtete) Assoziation mit Multiplizitäten, Vererbung
     - Abstrakte Klassen, Polymorphie
   - Datenstrukturen
     - Ungerichteter gewichteter Graph mit den Akzenten
       (Anwendung der Standardoperationen, Breiten- und Tiefensuche, Suche des kürzesten Weges zwischen zwei Knoten: Backtracking, Dijkstra-Algorithmus)

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

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

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösung

Teilaufgabe a)

Adjazenzmatrix:

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	1
C	1	1	0	1	1
D	0	0	1	0	1
E	1	1	1	1	0

Die nicht verbundenen Kanten können auch durch ∞ markiert werden oder die Einträge können leer bleiben.

```
public NikoHaus(){
  nikoGraph = new Graph();
  nikoGraph.addNode(new GraphNode("A"));
  nikoGraph.addNode(new GraphNode("B"));
  nikoGraph.addNode(new GraphNode("C"));
  nikoGraph.addNode(new GraphNode("D"));
  nikoGraph.addNode(new GraphNode("E"));
  nikoGraph.addEdge(nikoGraph.getNode("A"),nikoGraph.getNode("B"),1);
  nikoGraph.addEdge(nikoGraph.getNode("A"),nikoGraph.getNode("C"),1);
  nikoGraph.addEdge(nikoGraph.getNode("A"),nikoGraph.getNode("E"),1);
  nikoGraph.addEdge(nikoGraph.getNode("B"),nikoGraph.getNode("C"),1);
  nikoGraph.addEdge(nikoGraph.getNode("B"),nikoGraph.getNode("E"),1);
  nikoGraph.addEdge(nikoGraph.getNode("C"),nikoGraph.getNode("D"),1);
  nikoGraph.addEdge(nikoGraph.getNode("C"),nikoGraph.getNode("E"),1);
  nikoGraph.addEdge(nikoGraph.getNode("D"),nikoGraph.getNode("E"),1);
}
```

Teilaufgabe b)

Wertebelegungen bei den Methodenaufrufen:

pKnoten (Name)	pWeg	pStufe	pTeilloesung
A	A	0	
B	AB	1	
C	ABC	2	
A	ABCA	3	
E	ABCAE	4	
B	ABCAEB	5	
C	ABCAEC	5	
D	ABCAECD	6	
E	ABCAECDE	7	ABCAECDEB

Da die Methode `unbekannt` sich selbst wieder aufruft, handelt es sich um eine rekursive Methode.

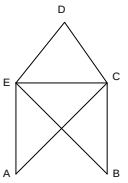
Die Methode `unbekannt` besucht per Tiefensuche alle Knoten. Der Endknoten der betrachteten Kante ist der Ausgangsknoten der nächsten betrachteten Kante. Es wird so sichergestellt, dass das Haus des Nikolaus mit einem Linienzug gezeichnet wird. Besuchte Kanten werden durch das Kantengewicht `-1` markiert, um einen doppelten Durchlauf der Kante zu verhindern. Da die Anzahl der zu untersuchenden Kanten bekannt ist, wird die Länge des Kantenzugs in dem Parameter `pStufe` mitgezählt. Falls alle Kanten gefunden worden sind, wird der bis dahin gefundene Weg in der Zeichenkette `pTeilloesung` ergänzt. Beim ersten Erreichen der Zeile 17 steht eine Lösung in der Zeichenkette `pTeilloesung`. Die Markierung der besuchten Kanten wird wieder auf `1` gesetzt. Dadurch wird erreicht, dass alle möglichen Linienzüge bestimmt werden.

Teilaufgabe c)

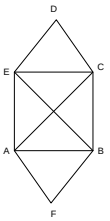
Knoten	A	B	C	D	E
Knotengrad	3	3	4	2	4

Der Graph ist nicht eulersch, da von den Knoten A und B drei Kanten abgehen. Es müssen von diesen Knoten zwei oder vier Kanten abgehen.

Wenn man die Kante von A nach B entfernt, wird der Graph eulersch. A und B haben jetzt den Knotengrad 2.



Wenn man einen Knoten F hinzufügt und die Kanten von A nach F und von B nach F wird der Graph eulersch. A und B haben den Knotengrad 4 und F den Knotengrad 2.



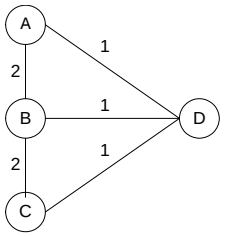
Teilaufgabe d)

```
private int knotenGrad(GraphNode pKnoten) {
  int zaehler = 0;
  List nachbarListe = testGraph.getNeighbours(pKnoten);
  nachbarListe.toFirst();
  while (nachbarListe.hasAccess()) {
    zaehler = zaehler + 1;
    nachbarListe.next();
  }
  return zaehler;
}

public boolean eulersch() {
  List knotenListe = testGraph.getNodes();
  boolean euler = true;
  knotenListe.toFirst();
  while (knotenListe.hasAccess() && euler) {
    if (knotenGrad((GraphNode)knotenListe.getObject()) %2 == 1) {
      euler = false;
    }
    else {
      knotenListe.next();
    }
  }
  return euler;
}
```

Teilaufgabe e)

Eine Möglichkeit besteht darin, die Knoten als Objekte der Klasse `GraphNode` zu verwalten. Die Knoten, die durch mindestens eine Kante verbunden sind, werden als Kanten in ein Objekt der Klasse `Graph` hinzugefügt. Als Kantengewicht wird die Anzahl der Kanten, die Knoten verbinden, angegeben.



Bei der Prüfung, ob der Graph eulersch ist, müsste für jeden Knoten der Knotengrad bestimmt werden. Man erhält ihn, indem man die Kantengewichte der Kanten zu den Nachbarknoten addiert. Falls der Knotengrad für jeden Knoten gerade ist, ist der Graph eulersch.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	stellt den Graphen als Adjazenzmatrix dar.	4
2	implementiert den Konstruktor der Klasse `NikoHaus`.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	bestimmt für jeden Methodenaufruf die Parameterwerte.	6
2	bestimmt den Wert der erzeugten Zeichenkette.	2
3	erläutert die Arbeitsweise der Methode.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	bestimmt die Knotengrade und wertet das Ergebnis aus.	4
2	bestimmt einen eulerschen Graphen mit weniger Kanten als beim vorgegebenen Graphen.	2
3	bestimmt einen eulerschen Graphen mit mehr Kanten als beim vorgegebenen Graphen.	2
4	erläutert die Entscheidungen.	2
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	implementiert die Bestimmung des Knotengrads für Knoten.	5
2	implementiert die Analyse der Knotengrade und die Entscheidung, ob der Graph eulersch ist.	5
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe e)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	modelliert eine Datenstruktur zur Darstellung der Multigraphen.	5
2	entwirft und beschreibt einen Algorithmus, der entscheidet, ob der Multigraph eulersch ist.	5
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**[2]	**ZK**	**DK**
1	stellt den Graphen …	4			
2	implementiert den Konstruktor …	4			
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	bestimmt für jeden …	6			
2	bestimmt den Wert …	2			
3	erläutert die Arbeitsweise …	4			
	Summe Teilaufgabe b)	**12**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	bestimmt die Knotengrade …	4			
2	bestimmt einen eulerschen …	2			
3	bestimmt einen eulerschen …	2			
4	erläutert die Entscheidungen.	2			
	Summe Teilaufgabe c)	**10**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Bestimmung …	5			
2	implementiert die Analyse …	5			
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	modelliert eine Datenstruktur …	5			
2	entwirft und beschreibt …	5			
	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				
Note ggf. unter Absenkung um ein bis zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

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 – 58
mangelhaft plus	3	57 – 49
mangelhaft	2	48 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0

Name: _____

Abiturprüfung 2012

Informatik, Leistungskurs

Aufgabenstellung:

Der Euro ist die gemeinsame Währung einiger Länder der Europäischen Union. Die Euro-Geldscheine weisen verschiedene Merkmale auf, mit denen einerseits Fälschungen verhindert, andererseits aber auch maschinelle Zugriffe durch Automaten möglich sind. Ein Beispiel hierfür ist ein eingearbeiteter Strichcode, der den Wert des Scheins für Automaten lesbar macht:

Abbildung 1: 50 EUR Schein, Strichcode `01 10 10 10` hervorgehoben

Hält man einen Geldschein gegen eine Lichtquelle, so erscheint in der Durchsicht rechts vom Wasserzeichen ein Strichcode aus sechs oder acht senkrechten Streifen. Jeder Streifen hat eine feste Breite. Von einem Scanner werden die Streifen gelesen und als Bits interpretiert: ein heller Streifen als „`0`“, ein dunkler als „`1`“. Dieses Bitmuster wird dann paarweise als sogenannte Manchester-Codierung interpretiert: Das Paar „`01`“ wird als `1` interpretiert, das Paar „`10`“ als `0`.

Name: _____

Manchestercode	Binärcode	Geldschein-Wert
01 10 10	**100**	5 €
01 01 10	**110**	10 €
10 10 10 10	**0000**	20 €
01 10 10 10	**1000**	50 €
01 01 10 10	**1100**	100 €
01 01 01 10	**1110**	200 €
01 01 01 01	**1111**	500 €

Mit einem endlichen Automaten lässt sich prüfen, ob der Binärcode für einen Wert des Geldscheins ein gültiger Binärcode ist.

Der folgende endliche Automat soll angeblich prüfen, ob ein ermittelter Binärcode gültig ist.

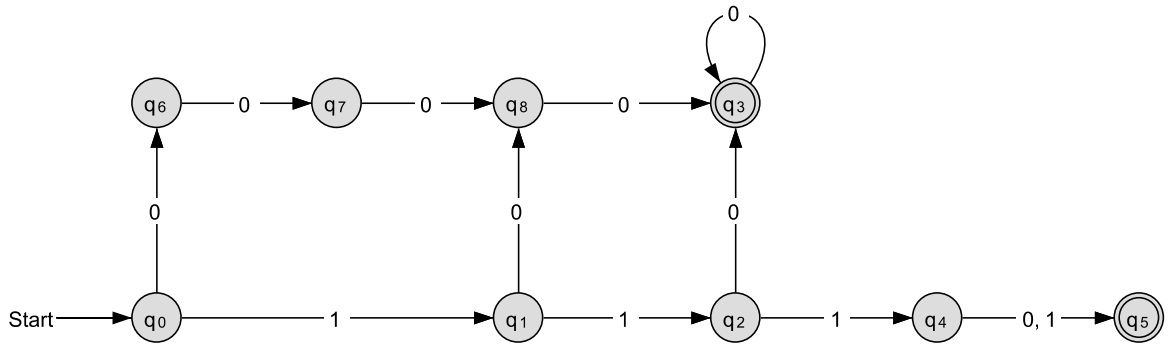


Abbildung 2: Automat 1

a) *Geben Sie für die folgenden Eingaben (in Binärcode) die Zustandsfolgen an und geben Sie an, ob die Eingaben akzeptiert werden.*

```
0000
100
1001
1110
```

Erläutern Sie, dass der endliche Automat mehr als die sieben gültigen Binärcodes akzeptiert.

Entwerfen Sie einen neuen deterministischen endlichen Automaten, der nur die gültigen Binärcodes akzeptiert.

(16 Punkte)

Name: _____

b) Statt den aus dem Manchestercode interpretierten Binärcode mit einem endlichen Automaten zu überprüfen, könnte ein endlicher Automat auch direkt den Manchestercode überprüfen. Die folgende Abbildung zeigt einen Ausschnitt aus Automat 1:

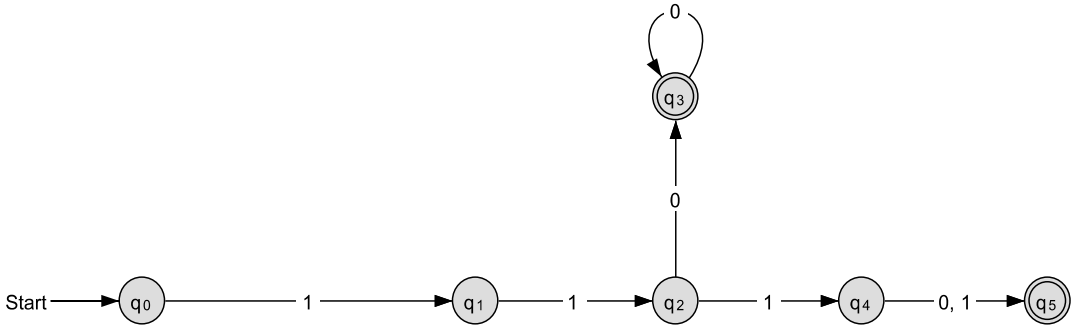


Abbildung 3: Ausschnitt aus Automat 1

Erweitern Sie den oben angegebenen Ausschnitt des endlichen Automaten für den Binärcode in einen deterministischen endlichen Automaten, der den Manchestercode für den Wert von Geldscheinen überprüft.

Erläutern Sie Ihr Vorgehen. (10 Punkte)

Die Entscheidung, welcher Wert des Geldscheins zu dem eingegebenen Binärcode gehört, ist deutlich interessanter als nur zu überprüfen, ob der Binärcode korrekt ist. Dazu ist es notwendig, die klassische Entscheidung „Eingabe akzeptiert“ oder „Eingabe nicht akzeptiert“ um eine Ausgabe zu erweitern. Die sogenannten *Moore-Automaten* ermöglichen eine solche Ausgabe:

Die Moore-Automaten unterscheiden sich von den endlichen deterministischen Automaten in folgenden Punkten:
1. Moore-Automaten haben üblicherweise keine Endzustände.
2. Jedem Zustand ist ein Ausgabezeichen zugeordnet, das bei jedem Erreichen dieses Zustands ausgegeben wird. Das Ausgabezeichen befindet sich in der Darstellung im unteren Bereich der Zustandssymbole.
   Die Menge der Ausgabezeichen bezeichnet man als das *Ausgabealphabet*.

Moore-Automaten können zur Decodierung des Binärcodes verwendet werden. Dabei wird zunächst ein Moore-Automat mit dem Eingabealphabet `{0;1}` gewählt. Die zu decodierende binäre Sequenz wird in den Automaten eingegeben; die gesamte Ausgabe ergibt sich aus der Folge der einzelnen Ausgabezeichen.

Beispiel:
Der im Folgenden dargestellte Moore-Automat hat das Ausgabealphabet
`O = {0, 2, 5, e}`, wobei `e` für eine leere Ausgabe stehen soll. Er liefert bei Eingabe der Sequenz `100` die Ausgabe `eee5`. Der Wert des Geldscheins wäre also 5 €.

Name: _____

Zu beachten ist, dass die Ausgabe des Startzustands in jedem Fall stattfindet.

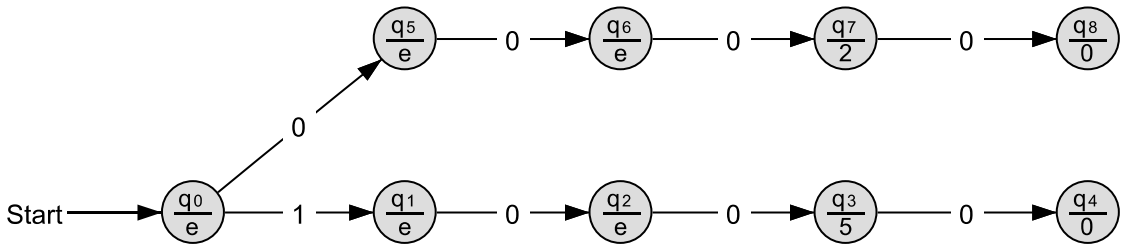


Abbildung 4: Beispiel für einen Moore-Automaten

c) *Geben Sie für die folgenden Eingaben des Binärcodes die Zustandsfolgen an und ermitteln Sie die jeweilige Ausgabe:*

```
1000
0000
```

(6 Punkte)

d) *Erweitern Sie den in Abbildung 4 dargestellten Moore-Automaten so, dass er zusätzlich die Geldscheinwerte 10 € und 100 € anhand ihres Binärcodes erkennt. Gehen Sie davon aus, dass nur korrekte Eingaben vorkommen. Das Ausgabealphabet des Automaten soll nur aus Dezimalziffern und der leeren Ausgabe* e *bestehen.*

(7 Punkte)

e) Der Automat aus Abbildung 4 kann nicht um das Erkennen der Geldscheinwerte 200 € und 500 € erweitert werden, wenn die Ziffern des Geldscheinwertes wie bisher einzeln ausgegeben werden sollen. Dies liegt daran, dass die Binärcodes dieser beiden Geldscheinwerte identisch anfangen und sich nur in der letzten Stelle unterscheiden.

Entwerfen Sie für alle Geldscheinwerte einen neuen Binärcode, der es ermöglicht, dass ein Moore-Automat den Geldscheinwert ziffernweise ausgibt.

Entwerfen Sie auch den zugehörigen Automaten.

(11 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Unterlagen für die Lehrkraft

Abiturprüfung 2012

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Aufgabenstellung aus dem Bereich endliche Automaten und formale Sprachen

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2012

1. *Inhaltliche Schwerpunkte*
   Endliche Automaten und formale Sprachen
   - Modellieren kontextbezogener Problemstellungen als deterministische endliche Automaten
   - Darstellung von deterministischen endlichen Automaten als Graph und als Tabelle

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

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

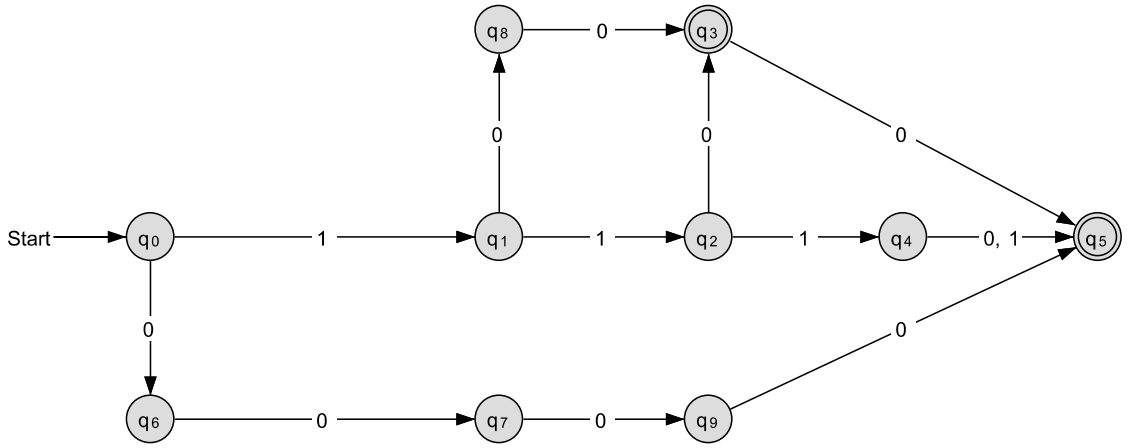
6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösung

Teilaufgabe a)

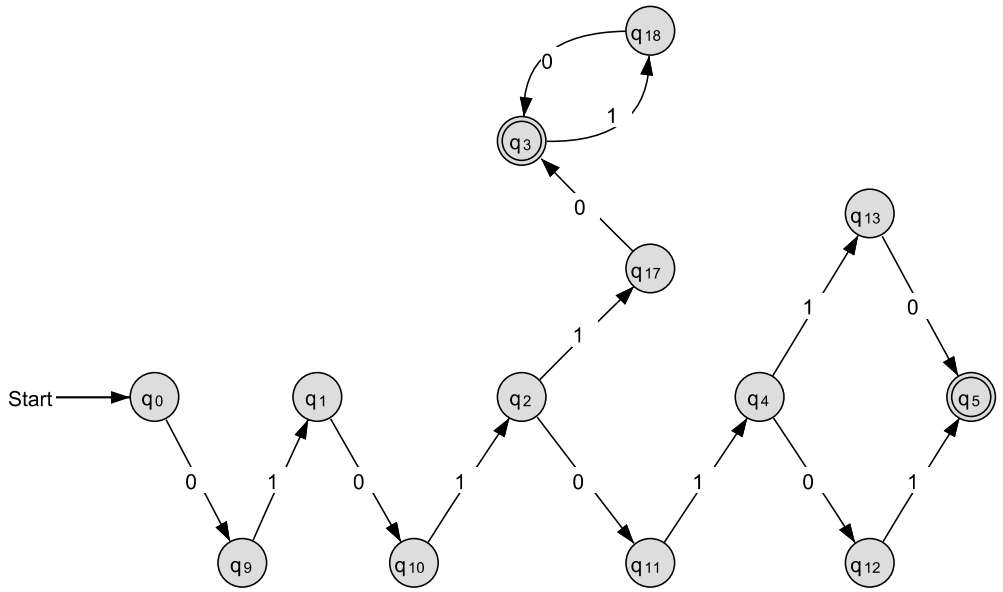
`0000`: $q_0 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8 \rightarrow q_3$	`0000` wird akzeptiert.
`100`: $q_0 \rightarrow q_1 \rightarrow q_8 \rightarrow q_3$	`100` wird akzeptiert.
`1001`: $q_0 \rightarrow q_1 \rightarrow q_8 \rightarrow q_3 \rightarrow$ nicht definiert	`1001` wird nicht akzeptiert.
`1110`: $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5$	`1110` wird akzeptiert.

Der Automat erkennt mehr als die zulässigen Binärcodes. Dies ergibt sich schon durch die Schleife im Endzustand q_3. Sie sorgt dafür, dass unendlich viele Eingaben akzeptiert werden, obwohl es nur sieben gültige Binärcodes gibt.



Teilaufgabe b)

`Z = {q0, q1, q2, q3, q4, q5, q9, q10, q11, q12, q13, q17, q18}`
`E = {q3, q5}`



Die grundsätzliche Idee zur Überführung des Automaten besteht darin die Übergänge zu ändern. Dazu muss bei jedem Übergang ein zusätzlicher Zustand hinzugefügt werden. Jeder Übergang mit der Eingabe `1` wird nun in zwei Übergänge übersetzt, zuerst die Eingabe `0` zum neuen Zustand, dann die Eingabe `1` zum ursprünglich folgenden Zustand. Die Übergänge bei der Eingabe `0` sind genau umkehrt, zuerst die Eingabe `1`, dann die Eingabe `0`.

Dieses Verfahren muss auch für die Schleife bei Zustand q_3 ebenfalls angewendet und für den Übergang von Zustand q_4 nach q_5 doppelt angewendet werden.

Teilaufgabe c)

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$ Ausgabe: `eee50` bzw. `50`

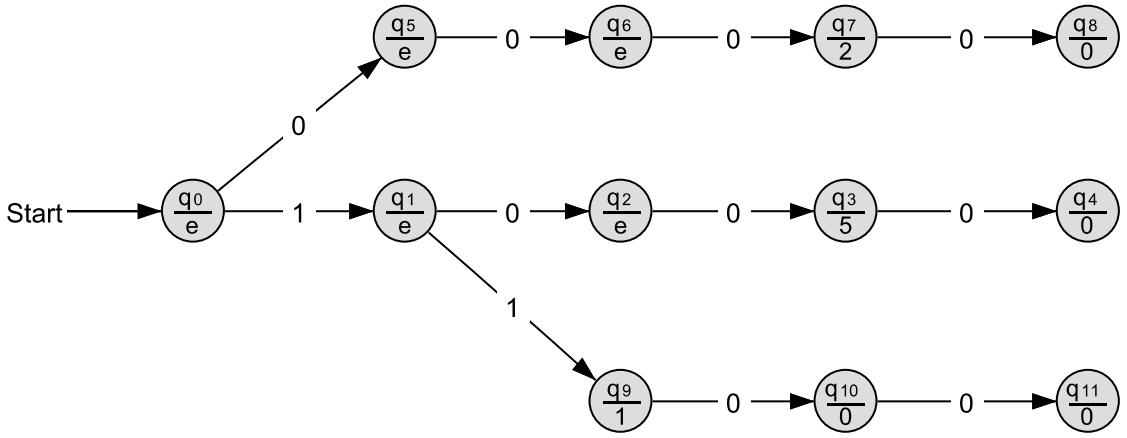
$q_0 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8$ Ausgabe: `eee20` bzw. `20`

Teilaufgabe d)

Ausgabealphabet: `O = {0, 1, 2, 5, e}`

Zustandsmenge: `Z = {`q_0`, ... `q_{11}`}`

Übergangsgraph:

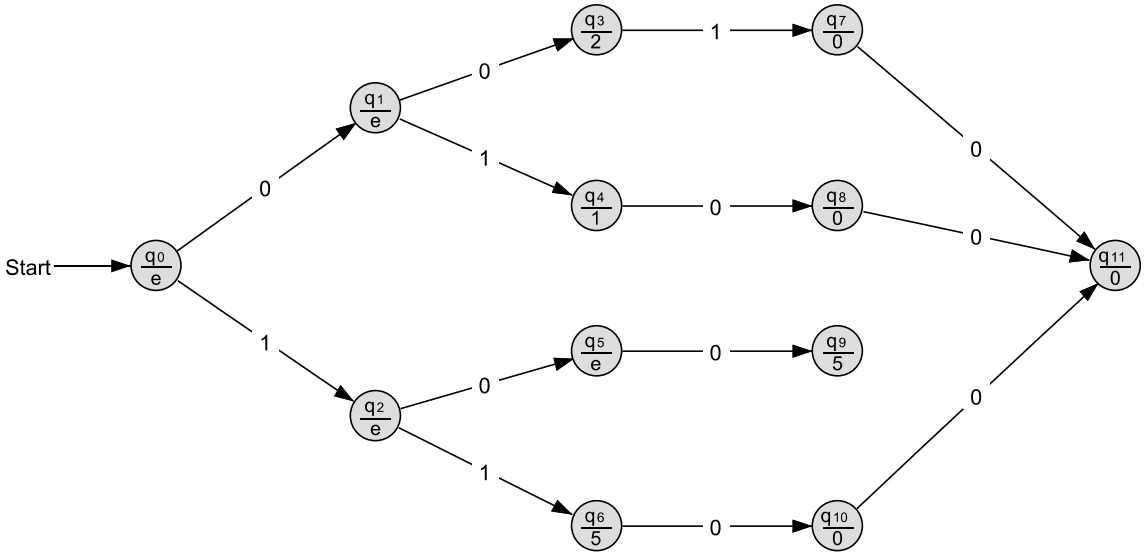


Teilaufgabe e)

Eine mögliche Lösung könnte wie folgt aussehen: Sie berücksichtigt die ursprünglich festgelegte (aber in der Aufgabenstellung nicht geforderte) Anzahl von sechs oder acht senkrechten Streifen.

Binärcode	Geldschein-Wert
100	5 €
010	10 €
001	20 €
110	50 €
0100	100 €
0010	200 €
1100	500 €

Die Ausgabe des Automaten endet mit der Verarbeitung des letzten Zeichens der Eingabe. Aus Vereinfachung wurde ein Fehlerzustand bei einer fehlerhaften Eingabe – einschließlich einer Fehlermeldung als Ausgabe – weggelassen.



6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die Zustandsfolgen an.	4
2	gibt an, ob die Eingaben akzeptiert werden.	2
3	erläutert, dass der Automat mehr als die zulässigen Eingaben akzeptiert.	3
4	entwirft den neuen Automaten.	7
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die erweiterte Menge der Zustände und die Menge der Endzustände an.	2
2	erweitert den Übergangsgraphen.	5
3	erläutert sein Vorgehen.	3
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die Zustandsfolgen an.	3
2	ermittelt die Ausgaben.	3
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	erweitert die Zustandsmenge und das Ausgabealphabet.	2
2	erweitert den Übergangsgraph.	5
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe e)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	entwirft den Binärcode.	6
2	entwickelt den zugehörigen Automaten	5
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	gibt die Zustandsfolgen …	4			
2	gibt an, ob …	2			
3	erläutert, dass der …	3			
4	entwirft den neuen …	7			
	Summe Teilaufgabe a)	**16**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt die erweiterte …	2			
2	erweitert den Übergangsgraphen.	5			
3	erläutert sein Vorgehen.	3			
	Summe Teilaufgabe b)	**10**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt die Zustandsfolgen …	3			
2	ermittelt die Ausgaben.	3			
	Summe Teilaufgabe c)	**6**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erweitert die Zustandsmenge …	2			
2	erweitert den Übergangsgraph.	5			
	Summe Teilaufgabe d)	**7**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft den Binärcode.	6			
2	entwickelt den zugehörigen …	5			
	Summe Teilaufgabe e)	**11**			

	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				
Note ggf. unter Absenkung um ein bis zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

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 – 58
mangelhaft plus	3	57 – 49
mangelhaft	2	48 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0

Name: _____

Abiturprüfung 2012

Informatik, Leistungskurs

Aufgabenstellung:

Der Literaturkurs eines Gymnasiums führt in der Aula ein Theaterstück auf.

Die Eintrittskarten können über ein Netzwerk reserviert werden. Die Nutzer des Systems sind mit ihren entsprechenden Anmeldedaten (Name und Passwort) im System in der so genannten Anwenderverwaltung hinterlegt.

Eine Reservierung kann durch Stornierung aufgehoben werden.

Im Folgenden ist ein Ausschnitt aus dem Protokoll der Netzwerkanwendung dargestellt.

Client an Server	Server an Client
`LOGIN:<anmeldename>:<kennwort>` Ein Anwender meldet sich an.	`OK` oder `ERROR:<meldung>` Der Server meldet, ob die Anmeldung erfolgreich war.
`ANZREIHEN` Es wird die Anzahl der Sitzreihen angefordert.	`REIHEN:<anzreihen>` Der Server liefert die Anzahl der Sitzreihen zurück.
`RESERV:<reihe>:<sitz>` Der Anwender will den Platz mit der entsprechenden Sitz-Nr. in der entsprechenden Reihe reservieren.	`OK` oder `ERROR:<meldung>` Der Server meldet, ob die Operation erfolgreich durchgeführt wurde.
`STORNO:<reihe>:<sitz>` Der Anwender will die Reservierung für den Platz mit der entsprechenden Sitz-Nr. stornieren.	`OK` oder `ERROR:<meldung>` Der Server meldet, ob die Operation erfolgreich durchgeführt wurde.
`LOGOUT` Der Anwender meldet sich ab.	`BYE` Der Server bestätigt die Abmeldung.

Name: _____

`REIHE:<reihenNr>`
Der Anwender erfragt die Belegung einer Sitzreihe.
	`BELEG:<belegung>`
	Es wird die Belegung der angefragten Sitzreihe zurückgegeben. Dabei stehen in `<belegung>` die Belegungen der Plätze der Reihe von links nach rechts, durch Doppelpunktzeichen getrennt.
	Beispiel:
	`"BELEG:Peter: :Fritz:Fritz: : : "`

`ANZSITZE`	`SITZE:<anzSitze>` Es wird die Anzahl der Sitze in jeder Reihe zurückgegeben.

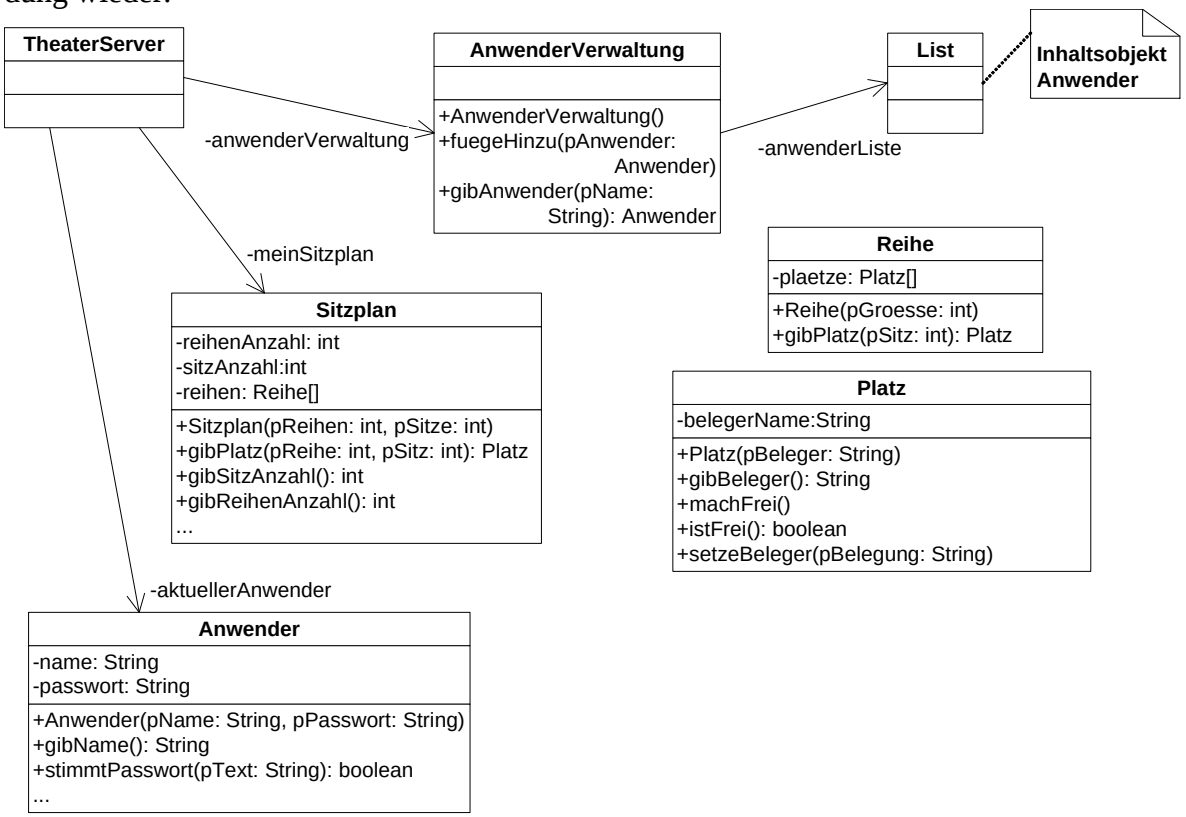
a) *Geben Sie auf der Grundlage des obigen Protokolls die zwischen Client und Server ausgetauschten Nachrichten in folgenden Situationen an:*

- Der registrierte Nutzer Hubert Meier meldet sich mit den korrekten Anmeldedaten Anmeldename `hubert.meier` und Passwort `hubert0815` an.
- Hubert Meier reserviert den (bis dahin freien) Platz 7 in Reihe 4.
- Hubert Meier möchte den Platz 8 in Reihe 4 reservieren, der jedoch schon belegt ist.
- Hubert Meier storniert seine Reservierung von Platz 7 in Reihe 4.
- Hubert Meier meldet sich ab.

(6 Punkte)

Name: _____

Das folgende Diagramm gibt die wesentlichen Aspekte eines Modells der Netzwerkanwendung wieder.



b) Pro Anwender sollen maximal 10 Karten reserviert werden können.

Erläutern Sie, wie dies realisiert werden kann, indem Sie gegebenenfalls neue Attribute und Methoden und deren Funktionsweise beschreiben. *(6 Punkte)*

c) In der ursprünglichen Planung waren die Preise der Karten einheitlich. Da der Kurs das Stück nun auch in der Stadthalle aufführen soll, in der die Bühne von bestimmten Plätzen aus schlecht einzusehen ist, soll der Kartenpreis für diese Plätze niedriger sein.

Erweitern Sie das gegebene Protokoll und Modell so, dass sich ein Anwender über den Preis eines bestimmten Platzes informieren kann.

In der Stadthalle sind die Plätze im Halbrund angeordnet, so dass nicht jede Reihe dieselbe Anzahl an Plätzen besitzt.

Erläutern Sie, wie das gegebene Modell auch bei einer Aufführung in der Stadthalle verwendet werden kann, machen Sie Vorschläge zur Veränderung des Modells.
(10 Punkte)

Name: _____

d) Bisher werden die Namen der Anwender, die einen Platz reserviert haben, im Klartext übertragen. Die Belegung einer Sitzreihe soll nun anonymisiert an den Client übertragen werden. Dazu werden in der entsprechenden Zeichenkette die Namen der Benutzer, die nicht dem angemeldeten Anwender entsprechen, durch ein `B` (für besetzt) ersetzt.

Implementieren Sie eine entsprechende Methode der Klasse `TheaterServer` *mit dem Methodenkopf*

`String gibReiheAnonym(int pReihe).` (12 Punkte)

e) Um die Sicherheit zu erhöhen, sollen Benutzername und Passwort nicht im Klartext übertragen werden. Dazu wurde eine Klasse `Crypt` mit der folgenden Methode entwickelt.

```
 1 public String chiffriere(int pSpaltenZahl, String pKlartext){
 2   String chiffre = "";
 3   int i = 0;
 4   int textLaenge = pKlartext.length();
 5   int zeilenZahl = textLaenge / pSpaltenZahl;

 6   char[][] zeichenMatrix = new char[zeilenZahl][pSpaltenZahl];

 7   int z = 0;
 8   int s = 0;

 9   while(i < textLaenge) {
10     zeichenMatrix[z][s] = pKlartext.charAt(i);
11     if(s < pSpaltenZahl - 1)
12       s = s + 1;
13     else {
14       z = z + 1;
15       s = 0;
16     }
17     i = i + 1;
18   }
19     for(s = 0; s < pSpaltenZahl; s++) {
20        for(z = 0; z < zeilenZahl; z++) {
21             chiffre += zeichenMatrix[z][s];
22        }
23     }
24     return chiffre;
25 }
```

Name: _____

Analysieren Sie die Methode, indem Sie die in Zeile 24 zurückgegebene Zeichenkette für den folgenden Methodenaufruf ermitteln:

```
chiffriere(5, "informatik")
```

und erläutern Sie die Funktionsweise des Verfahrens im Sachzusammenhang.

Erläutern Sie, welches Problem bei folgendem Methodenaufruf auftritt:

```
chiffriere(4, "informatik")
```

und schlagen Sie eine Strategie zur Vermeidung vor.

Beurteilen Sie, ob die Methode `chiffriere` *auch zum Dechiffrieren verwendet werden kann.*

(16 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anlagen

Auszug aus der Dokumentation der Klasse `Anwender`

Konstruktor `Anwender (String pName, String pPasswort)`
Ein Anwender mit Name `pName` und Passwort `pPasswort` wird erzeugt.

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

Anfrage `boolean stimmtPasswort(String pText)`
Die Anfrage liefert zurück, ob das übergebene Passwort richtig ist.

Auszug aus der Dokumentation der Klasse `AnwenderVerwaltung`

Konstruktor `AnwenderVerwaltung()`
Eine leere Anwenderverwaltung wird erzeugt.

Anfrage `Anwender gibAnwender(String pName)`
Die Anfrage liefert den Anwender mit dem Namen `pName` zurück. Existiert kein entsprechender Anwender, so wird `null` zurückgegeben.

Auftrag `void fuegeHinzu(Anwender pAnwender)`
Fügt den Anwender `pAnwender` hinzu.

Auszug aus der Dokumentation der Klasse `Reihe`

Konstruktor `Reihe (int pGroesse)`
Eine Reihe mit einer Anzahl von `pGroesse` Sitzen.

Anfrage `Platz gibPlatz(int pSitz)`
Liefert den Platz mit Sitznummer `pSitz` zurück.

Name: _____

Auszug aus der Dokumentation der Klasse `Platz`

Konstruktor `Platz (String pBeleger)`
Ein Platz mit Belegername `pBeleger` wird erzeugt.

Anfrage `String gibBeleger()`
Die Anfrage liefert den Namen des Anwenders, für den der Platz reserviert ist, zurück. Existiert kein entsprechender Beleger, so wird `null` zurückgegeben.

Auftrag `void setzeBeleger(String pBelegung)`
Setzt den Belegernamen auf den übergebenen Wert `pBelegung`.

Auftrag `void machFrei()`
Gibt die Belegung des Platzes frei. Der Belegername ist jetzt `" "`.

Anfrage `boolean istFrei ()`
Die Anfrage liefert zurück, ob der Platz belegt ist.

Auszug aus der Dokumentation der Klasse `Sitzplan`

Konstruktor `Sitzplan (int pReihen, int pSitze)`
Ein Sitzplan mit `pReihen` Reihen und `pSitze` Sitzen pro Reihe wird erzeugt. Die Nummerierung beginnt jeweils bei 0 und geht bis `pReihen-1` bzw. `pSitze-1`.

Anfrage `int gibReihenAnzahl()`
Die Anfrage liefert die Anzahl der Reihen zurück.

Anfrage `int gibSitzAnzahl()`
Die Anfrage liefert die Anzahl der Sitze pro Reihe zurück.

Anfrage `Platz gibPlatz(int pReihe, int pSitz)`
Die Anfrage liefert den Platz mit Sitznummer `pSitz` in Reihe `pReihe` zurück.

Name: _____

Die Klasse `List`

Objekte der Klasse **`List`** verwalten beliebig viele, linear angeordnete Objekte. 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 oder ein Listenobjekt an das Ende der Liste angefügt werden.

Dokumentation der Klasse `List`

Konstruktor **`List()`**
Eine leere Liste wird erzeugt.

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 **`Object getObject()`**

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 setObject(Object pObject)`**

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

Auftrag **`void append(Object pObject)`**

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

Auftrag **`void insert(Object pObject)`**

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt 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 `pObject` 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 `pObject` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void concat(List pList)`**

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls `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 2012

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Modellierung einer Problemstellung, Entwurf und Implementation von Netzwerkanwendungen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2012

1. *Inhaltliche Schwerpunkte*
   Modellieren und Implementieren kontextbezogener Problemstellungen als Netzwerkanwendungen
   - Netzwerkprotokolle
   - Client-Server-Anwendungen
   - Kryptografie
     - Symmetrische Verschlüsselungsverfahren (Caesar, Vigenère)

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

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

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösung

Teilaufgabe a)

Client an Server	Server an Client
`LOGIN:hubert.meier:hubert0815`	`OK`
`RESERV:4:7`	`OK`
`RESERV:4:8`	`ERROR:BESETZT`
`STORNO:4:7`	`OK`
`LOGOUT`	`BYE`

Der Text der Fehlermeldung kann natürlich abweichen.

Teilaufgabe b)

Mögliche Lösung:

Die Klasse `Anwender` wird um ein Attribut `anzahlReservierungen` vom Typ `int` erweitert, welches bei Reservierungen erhöht, bei Stornierungen heruntergesetzt und abgefragt werden kann.

Der Server prüft bei jedem Reservierungswunsch, wie viele Reservierungen der entsprechende Anwender bereits getätigt hat und verhindert ggf., dass dieser Anwender zu viele Plätze reserviert.

Teilaufgabe c)

Hier sind unterschiedliche Lösungen zu erwarten, eine mögliche Lösung lautet wie folgt:

Client an Server	Server an Client

`PREIS:<reihe>:<sitz>`	`KOSTEN:<preis>`
	`oder`
	`ERROR:<meldung>`

Das Protokoll wird um einen Befehl PREIS mit Parametern für die Reihe und den Sitz erweitert. Existiert der entsprechende Platz, so wird der Preis zurückgeliefert, andernfalls eine Fehlermeldung.

Die Klasse `Platz` wird um ein Attribut für den Preis mit entsprechenden setze- und gib-Methoden erweitert.

Um das Modell weiter verwenden zu können, müsste die maximale Sitzanzahl in einer Reihe ermittelt werden, um einen Sitzplan hinreichender Größe erzeugen zu können. Hierdurch entsteht das Problem, dass es nun in manchen Reihen mehr Plätze im Sitzplan gibt als tat-

sächlich im Zuschauerraum vorhanden. Daher müssten beispielsweise die nicht existierenden Plätze durch einen bestimmten Belegungsnamen gekennzeichnet werden. Besser wäre aber eine Dynamisierung der Datenstrukturen im Sitzplan. Er könnte beispielsweise als Liste von Listen verwaltet werden.

Teilaufgabe d)

```
public String gibReiheAnonym(int pReihe) {
  int sitzAnzahl = meinSitzplan.gibSitzAnzahl();
  int reihenAnzahl = meinSitzplan.gibReihenAnzahl();

  String anonym = "BELEG";

  if(pReihe < 0 || (reihenAnzahl-pReihe) < 1) {
    return "ERROR: UNGUELTIGE REIHENANZAHL";
  }
  else
  {
    for(int s = 0; s < sitzAnzahl; s++) {
      Platz platz = meinSitzplan.gibPlatz(pReihe, s);
      if(!platz.gibBeleger().equals(aktuellerAnwender.gibName())) {
        if(platz.istFrei()) {
          anonym = anonym + ": ";
        }
        else
          anonym = anonym + ":B";
      }
      else
        anonym = anonym + ":" + aktuellerAnwender.gibName();
    }

    return anonym;
  }
}
```

Teilaufgabe e)

Es wird die Zeichenkette "`imnaftoirk`" zurückgeliefert.

Der Klartext wird zeilenweise in eine Matrix mit der gegebenen Spaltenanzahl übertragen. Anschließend wird diese Matrix spaltenweise ausgelesen. Die dabei entstehende Chiffre ist dabei eine Permutation des Klartextes. Es handelt sich um das so genannte Skytale-Verfahren.

Ist die Länge des Chiffretextes nicht durch die Spaltenzahl teilbar, ist die in der Methode angelegte Matrix nicht ausreichend groß und es kommt zu einer Speicher-Zugriffsverletzung.

Die gegebene Methode lässt sich nur dann zum Dechiffrieren verwenden, wenn als Spaltenzahl die Zeilenzahl der Matrix, die zum Chiffrieren verwendet wurde, übergeben wird.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die Nachrichten an.	6
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	erläutert, wie die Anforderungen im gegebenen Modell realisiert werden können, bzw. erläutert geeignete Erweiterungen oder Änderungen.	6
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	erweitert das Protokoll in gewünschter Weise.	4
2	erläutert, wie das Modell angewendet werden kann, und macht Vorschläge zur Verbesserung.	6
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	bestimmt bei der Implementation die Reihen- und Sitzanzahl.	2
2	prüft, ob die Reihennummer gültig ist, gibt sonst eine geeignete Fehlermeldung.	4
3	implementiert, dass die Sitze der korrekten Reihe durchlaufen werden.	4
4	prüft bei der Implementation die Belegung und anonymisiert ggf. die Namen.	2
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe e)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	analysiert die Methode und gibt die korrekte Chiffre an.	4
2	erläutert die Funktionsweise im Sachzusammenhang.	4
3	erläutert das auftretende Problem.	4
4	erläutert, wie die Methode zum Dechiffrieren verwendet werden kann.	4
Der gewählte Lösungsansatz und -weg muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	gibt die Nachrichten …	6			
	Summe Teilaufgabe a)	**6**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert, wie die …	6			
	Summe Teilaufgabe c)	**6**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erweitert das Protokoll …	4			
2	erläutert, wie das …	6			
	Summe Teilaufgabe d)	**10**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	bestimmt bei der …	2			
2	prüft, ob die …	4			
3	implementiert, dass die …	4			
4	prüft bei der …	2			
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methode …	4			
2	erläutert die Funktionsweise …	4			
3	erläutert das auftretende …	4			
4	erläutert, wie die …	4			
	Summe Teilaufgabe e)	**16**			

	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				
Note ggf. unter Absenkung um ein bis zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

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 – 58
mangelhaft plus	3	57 – 49
mangelhaft	2	48 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0

Name: _____

Abiturprüfung 2012

Informatik, Leistungskurs

Aufgabenstellung:

Bei der Firma „*Rent & Drive*“ handelt es sich um eine erfolgreiche Autovermietung mit einem interessanten Geschäftskonzept:

Das Unternehmen verfügt über mehrere Verleihstationen, in denen Kunden Autos ausleihen können. Diese Autos gehören jedoch nicht der Firma „*Rent & Drive*“. Vielmehr stellen nahegelegene Autohäuser ihre Ausstellungsfahrzeuge zur Verfügung. So kann das Unternehmen Autos verleihen, ohne sie selbst zu besitzen und somit preiswerte Angebote machen.

Da dieses Geschäftskonzept eine etwas kompliziertere Logistik erfordert, hat die Geschäftsleitung beschlossen, die folgenden Informationen in einer Datenbank zu verwalten.

Zu jeder Verleihstation und jedem Autohaus werden ein Name und die Standortstadt in die Datenbank eingetragen. Autohäuser können nicht beliebig viele Autos an Verleihstationen abgeben. Die maximale Anzahl von verfügbaren Autos wird für jedes Autohaus gespeichert. Die Verleihstationen werden von mehreren Autohäusern beliefert, wobei ein Autohaus natürlich auch Verträge mit mehreren Verleihstationen haben kann. Darüber hinaus werden an einer Verleihstation verschiedene Automarken angeboten. Diese Automarken haben jeweils einen Namen und eine Schulnote in der Pannenstatistik. Da die Autohäuser Filialen von Autoherstellern sind, führen sie natürlich nur die eigene Automarke. Des Weiteren sollte gespeichert werden, wie viele Autos einer bestimmten Marke eine Station zur Verfügung hat. Einzelne Fahrzeuge sollen in der Datenbank hingegen nicht gespeichert werden.

a) *Entwerfen Sie zu der hier gegebenen Beschreibung ein Entity-Relationship-Modell und erläutern Sie die Beziehungen zwischen den Entitäten.*

   Geben Sie anschließend das entsprechende Datenbankschema an und erläutern Sie, wie die verschiedenen Beziehungen aus Ihrem Modell darin umgesetzt sind.

   (12 Punkte)

Während für die Geschäftsleitung die oben beschriebene Datenbank ausreicht, ist es für die einzelne Verleihstation wichtig, einen Überblick über die einzelnen Fahrzeuge, Kunden und Vermietungsvorgänge zu behalten. Sie verwenden daher eine jeweils eigene Datenbank, die aus den folgenden Tabellen besteht.

Name: _____

Tabelle: Kunde	
ID	Zahl
Vorname	Text
Nachname	Text
Adresse	Text
Führerscheinklasse	Text

Tabelle: Fahrzeug	
ID	Zahl
Marke	Text
Typcode	Zahl
Navigation	Wahrheitswert
Kilometerstand	Zahl
PreisProTag	Zahl

Tabelle: mietet	
Monat	Zahl
Jahr	Zahl
↑IDKunde	Zahl
↑IDFahrzeug	Zahl
AnzahlTage	Zahl
Sportwagen	Wahrheitswert

Zu jedem Kunden werden sein Vor- und Nachname, die Adresse und seine Führerscheinklasse gespeichert. Fahrzeuge sind mit Marke, Typcode, Kilometerstand, Leihpreis pro Tag und der Information, ob sie über ein Navigationssystem verfügen, erfasst. Mit Hilfe des Typcodes kann z. B. eine Werkstatt die exakte Baureihe des Fahrzeugs ermitteln. Fahrzeuge unterschiedlicher Marke können niemals den gleichen Typcode haben.

Mietet ein Kunde ein Fahrzeug, so werden der Monat und das Jahr des Mietvorgangs sowie die Anzahl der Tage, für die das Auto vermietet wird, gespeichert. Des Weiteren wird vermerkt, ob hier ein Sportwagen vermietet wurde, da in dem Fall eine besondere Versicherung abzuschließen ist.

b) Im Folgenden sollen aus den oben gegebenen Tabellen Informationen ermittelt werden.

Entwerfen Sie jeweils eine geeignete SQL-Anweisung, …

- *die alle Marken ausgibt und wie viele Autos je Marke im System erfasst sind.*
- *die ausgibt, welche Fahrzeuge (ID, Marke, Typcode) von „Marie Schneider“ gemietet wurden. Die Liste soll nach der Anzahl der gemieteten Tage absteigend sortiert sein.*
- *die alle Fahrzeuge mit ID, Marke und Typcode ausgibt, die im Jahr 2012 noch nicht vermietet wurden.*

(10 Punkte)

Name: _____

c) Des Weiteren ist die folgende SQL-Anweisung gegeben, die ebenfalls auf den oben gegebenen Tabellen ausgeführt wird.

```
1 SELECT Fahrzeug.ID, Fahrzeug.Marke, Fahrzeug.Typcode,
                    (NeueTabelle.Anzahl * Fahrzeug.PreisProTag) AS Wert
2 FROM Fahrzeug LEFT JOIN (
3    SELECT mietet.IDFahrzeug AS Auto, SUM(mietet.AnzahlTage) AS Anzahl
4    FROM mietet
5    WHERE mietet.Jahr > 2009
6    GROUP BY mietet.IDFahrzeug
7 )  AS NeueTabelle ON Fahrzeug.ID = NeueTabelle.Auto
8 ORDER BY (NeueTabelle.Anzahl * Fahrzeug.PreisProTag) DESC;
```

Erläutern Sie den Aufbau dieser SQL-Anweisung und beschreiben Sie das Ergebnis.
(10 Punkte)

d) Das Design der oben gegebenen Datenbank soll nun auf seine Qualität hin überprüft werden.

Begründen Sie, dass die Datenbank keiner der ersten drei Normalformen entspricht.

Entwickeln Sie anschließend ein überarbeitetes Datenbankschema, welches sich in der dritten Normalform befindet. (10 Punkte)

e) Um den Service für seine Kunden zu verbessern, hat die Verleihstation „*Bodensee-Nord*“ der Firma „*Rent & Drive*“ vor, ihre Datenbank im Internet zur Verfügung zu stellen. Mit Hilfe einer entsprechenden Internetseite kann so jeder beliebig Daten daraus abfragen und z. B. nachsehen, ob das Wunschauto noch verfügbar ist.

Da man den Datenschutz bei „*Rent & Drive*“ sehr ernst nimmt, wird natürlich mit einer stets aktuellen Kopie der eigentlichen Datenbank gearbeitet. In dieser Kopie sind die Attributeinträge `Vorname` und `Nachname` aus der Tabelle `Kunde` gelöscht worden.

Nehmen Sie unter Berücksichtigung von Datenschutzaspekten Stellung zu diesem Vorhaben. (8 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Unterlagen für die Lehrkraft

Abiturprüfung 2012

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Modellieren von Datenbanken mit dem Entity-Relationship Modell, Normalisierung, SQL
Syntaxvariante	

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2012

1. *Inhaltliche Schwerpunkte*
   Relationale Datenbanken
   - Modellieren kontextbezogener Problemstellungen als Datenbanken mit dem Entity-Relationship Modell
   - Normalisierung: Überführung einer Datenbank in die 1. bis 3. Normalform
   - Relationenalgebra (Selektion, Projektion, Join)
   - SQL-Abfragen über eine und mehrere verknüpfte Tabellen

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

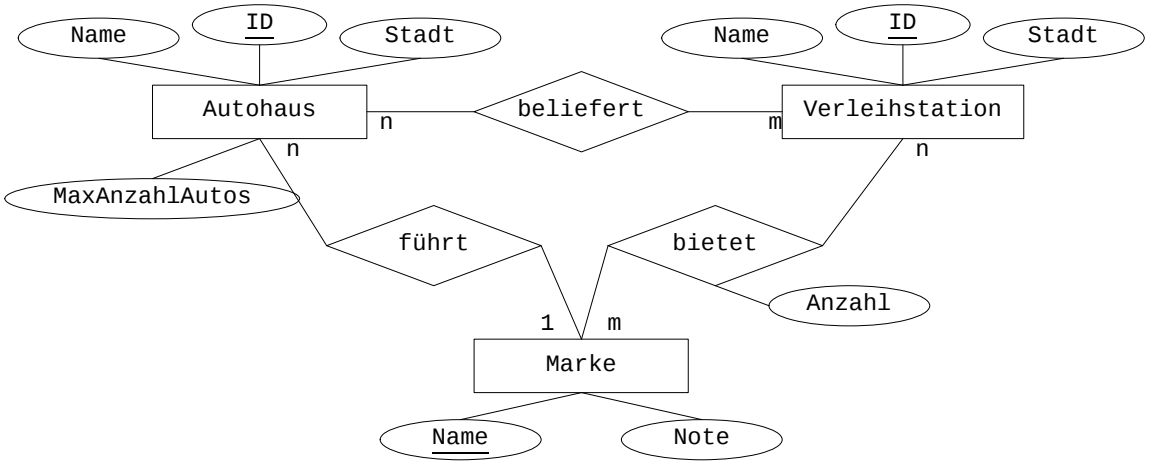
[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösung

Teilaufgabe a)

Folgendes Entity-Relationship-Modell entspricht der vorgegebenen Beschreibung:



Beziehung `beliefert`: Ein Autohaus beliefert mehrere Verleihstationen und eine Verleihstation wird von mehreren Autohäusern beliefert.

Beziehung `führt`: Ein Autohaus führt genau eine Marke. Eine Marke wird von mehreren Autohäusern geführt.

Beziehung `bietet`: Eine Verleihstation bietet mehrere Automarken an. Eine Automarke wird von mehreren Verleihstationen angeboten. In der Relation wird gespeichert, um wie viele Autos es sich jeweils handelt.

Das folgende Datenbankschema setzt das Entity-Relationship-Modell um:

Autohaus (ID, Name, Stadt, MaxAutoAnzahl, ↑NameMarke)

Verleihstation (ID, Name, Stadt)

Marke (Name, Note)

beliefert (↑IDAutohaus, ↑IDVerleihstation)

bietet (↑NameMarke, ↑IDVerleihstation, Anzahl)

Die n:1-Beziehung `führt` wird durch einen Fremdschlüsseleintrag in der Tabelle `Autohaus` realisiert. Für die n:m-Beziehungen `beliefert` und `bietet` wird jeweils eine Verknüpfungstabelle angelegt.

Teilaufgabe b)

Alle Marken und wie viele Autos je Marke im System erfasst sind:

```
1   SELECT Fahrzeug.Marke, COUNT(Fahrzeug.Marke)
2   FROM Fahrzeug
3   GROUP BY Fahrzeug.Marke;
```

Fahrzeuge (ID, Marke, Typcode) die von „Marie Schneider“ gemietet wurden:

```
1 SELECT Fahrzeug.ID, Fahrzeug.Marke, Fahrzeug.Typcode
2 FROM Kunde, mietet, Fahrzeug
3 WHERE Kunde.Vorname = "Marie" And
4       Kunde.Nachname = "Schneider" And
5       Kunde.ID = mietet.IDKunde And
6       mietet.IDFahrzeug = Fahrzeug.ID
7 ORDER BY mietet.AnzahlTage DESC;
```

Alle Fahrzeuge (ID, Marke, Typcode) die im Jahr 2012 noch nicht vermietet wurden:

```
1 SELECT Fahrzeug.ID, Fahrzeug.Marke, Fahrzeug.Typcode
2 FROM Fahrzeug
3 WHERE Fahrzeug.ID NOT IN (
4    SELECT Fahrzeug.ID
5    FROM Fahrzeug,mietet
6    WHERE Fahrzeug.ID = mietet.IDFahrzeug AND mietet.Jahr=2012
7 );
```

Teilaufgabe c)

Die vorliegende SELECT-Anweisung arbeitet mit einer Unterabfrage, die zunächst betrachtet wird.

Die Unterabfrage erstellt eine Ergebnistabelle, die aus zwei Spalten besteht. In der ersten Spalte stehen aus der Tabelle `mietet` die Fahrzeug-IDs aller Autos, die nach 2009 ausgeliehen wurden. Diese Einschränkung wird mit Hilfe der WHERE-Klausel der Unterabfrage realisiert. Gleiche Einträge werden mit GROUP BY zusammengefasst. Zu jeder Gruppe wird in der zweiten Spalte die Summe aller Miettage nach 2009 eingetragen. Die Unterabfrage liefert also die Information, welche Fahrzeuge nach 2009 wie viele Tage verliehen waren.

Der äußere SELECT-Befehl verknüpft die Tabelle `Fahrzeug` mit der Ergebnistabelle der Unterabfrage mittels eines LEFT JOIN. Er erzeugt eine Ergebnistabelle mit vier Spalten. In den ersten drei Spalten stehen alle Fahrzeuge aus der Tabelle `Fahrzeug` mit `ID`, `Marke` und `Typcode`. In der vierten Spalte wird die jeweilige Anzahl der Miettage, die in der Unterabfrage ermittelt wurde, mit dem dazugehörigen Preis pro Tag aus der Fahrzeugtabelle multipliziert.

Die SQL-Anweisung liefert also für jedes Auto, wie viel Geld es für die Firma nach 2009 erwirtschaftet hat. Da die Verknüpfung mittels eines LEFT JOIN erfolgt, wird die linke Seite der Verknüpfung, d. h. die Tabelle `Fahrzeug`, vollständig ausgegeben, auch wenn es in der Ergebnistabelle der Unterabfrage keinen entsprechenden Eintrag gibt. Auf diese Weise werden auch Fahrzeuge im Ergebnis aufgeführt, die nach 2009 noch gar nicht verliehen wurden.

Durch die ORDER BY-Klausel wird die Ergebnistabelle nach dem erwirtschafteten Betrag absteigend sortiert.

Anmerkung: Auch weniger umfangreiche Bearbeitungen dieser Teilaufgabe sind möglich.

Teilaufgabe d)

Alle drei Normalformen sind in dem Datenbankdesign verletzt.

1. Normalform:
Das Attribut `Adresse` der Tabelle `Kunde` ist nicht atomar. Für die Adresse müssen zwei neue Tabellen angelegt werden: die Tabelle `Adresse` und die Tabelle `Ort`. Wird nur eine Tabelle neu angelegt, wird die dritte Normalform durch die Änderung verletzt.

2. Normalform:
Das Attribut `Sportwagen` der Verknüpfungstabelle `mietet` ist nur von `IDFahrzeug` und somit nur von einem Teil des Primärschlüssels abhängig. Das Attribut muss in die Tabelle `Fahrzeug` verschoben werden.

3. Normalform:
Die Attribute `Marke` und `Typcode` der Tabelle `Fahrzeug` sind funktional voneinander abhängig. Sie müssen in die neue Tabelle `Typ` ausgelagert werden.

Das korrigierte Datenbankschema könnte wie folgt aussehen:

Kunde (ID, Vorname, Nachname, Führerschein, ↑IDAdresse)
Adresse (ID, Strasse, Hausnummer; ↑PLZ)
Ort (PLZ, Ortsname)

mietet (Monat, Jahr, ↑IDKunde, ↑IDFahrzeug, AnzahlTage)

Fahrzeug (ID, ↑Typcode, Navigation, Kilometerstand, PreisProTag)
Typ (Typcode, Marke, Sportwagen)

Teilaufgabe e)

Das Vorhaben der Firma ist unter anderem aus folgenden Gründen trotz Entfernung der Namen aus der Datenbank zweifelhaft:

1. Die Datenbank könnte zur statistischen Auswertung verwendet werden. So könnte z. B. ermittelt werden, welche Marken in einer Stadt am meisten vermietet wurden. Hat der Kunde einer Nutzung seiner Adressdaten zu diesem Zweck nicht zugestimmt, handelt es sich um einen Verstoß gegen die informationelle Selbstbestimmung.
2. Die abrufbaren Daten sind zwar nicht mehr personenbezogen, sie sind aber unter Umständen personenbeziehbar. Kennt man den Wohnort einer Person, so kann man mit der Datenbank ermitteln, welche Fahrzeuge wann, wie oft, wie lange und zu welchem Preis im Haushalt dieser Person gemietet wurden. Dabei handelt es sich um eine datenschutzrelevante Information.

Anmerkung: Abhängig von der unterrichtlichen Schwerpunktsetzung sind hier auch andere Argumentationen möglich.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	entwirft das Entity-Relationship Modell.	4
2	erläutert die Beziehungen zwischen den Entitäten.	2
3	gibt das Datenbankschema an.	4
4	erläutert die Umsetzung der Beziehungen im Datenbankschema.	2
Der gewählte Lösungsansatz und – weg muss nicht identisch mit der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	entwirft den SQL-Befehl für alle Marken und die Anzahl der Autos pro Marke.	3
2	entwirft den SQL-Befehl für alle Fahrzeuge von „Marie Schneider“.	3
3	entwirft den SQL-Befehl für alle Fahrzeuge, die 2012 noch nicht vermietet wurden.	4
Der gewählte Lösungsansatz und – weg muss nicht identisch mit der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	erläutert den Aufbau der SQL-Anweisung.	6
2	beschreibt das Ergebnis der SQL-Anweisung.	4
Der gewählte Lösungsansatz und – weg muss nicht identisch mit der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	begründet, warum die Datenbank nicht in der ersten Normalform ist.	2
2	begründet, warum die Datenbank nicht in der zweiten Normalform ist.	2
3	begründet, warum die Datenbank nicht in der dritten Normalform ist.	3
4	entwickelt ein Datenbankschema in dritter Normalform.	3
Der gewählte Lösungsansatz und – weg muss nicht identisch mit der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

Teilaufgabe e)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	beurteilt das Vorhaben unter Berücksichtigung von Datenschutzaspekten.	3
2	begründet sinnvoll seine Beurteilung.	5
Der gewählte Lösungsansatz und – weg muss nicht identisch mit der Modelllösung sein. Sachlich richtige Alternativen werden an dieser Stelle mit entsprechender Punktzahl bewertet.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	entwirft das Entity-Relationship-Modell.	4			
2	erläutert die Beziehungen …	2			
3	gibt das Datenbankschema …	4			
4	erläutert die Umsetzung …	2			
	Summe Teilaufgabe a)	**12**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft den SQL-Befehl …	3			
2	entwirft den SQL-Befehl …	3			
3	entwirft den SQL-Befehl …	4			
	Summe Teilaufgabe b)	**10**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert den Aufbau …	6			
2	beschreibt das Ergebnis …	4			
	Summe Teilaufgabe c)	**10**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet, warum die …	2			
2	begründet, warum die …	2			
3	begründet, warum die …	3			
4	entwickelt ein Datenbankschema …	3			
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt das Vorhaben …	3			
2	begründet sinnvoll seine …	5			
	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				
Note ggf. unter Absenkung um ein bis zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

ggf. arithmetisches Mittel der Punktsummen aus EK und ZK: _____

ggf. arithmetisches Mittel der Notenurteile aus EK und ZK: _____

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 – 58
mangelhaft plus	3	57 – 49
mangelhaft	2	48 – 40
mangelhaft minus	1	39 – 30
ungenügend	0	29 – 0