Name: _____

Abiturprüfung 2013

Informatik, Leistungskurs

Aufgabenstellung:

An der „*Grundschule am Zuckerberg*“ möchte man ein computergestütztes „Soziales Netzwerk“ installieren. Im Folgenden sollen einige Aspekte dieses Netzwerks modelliert werden.

In einem ersten, sehr einfachen Modell wird jedes Kind, das in einem sozialen Netzwerk angemeldet ist, unter seinem Vor- und Nachnamen verwaltet. Wir nehmen dabei an, dass es an der Schule keine zwei Kinder mit gleichen Vor- und Nachnamen gibt. Jedes Kind kann jederzeit andere Kinder einseitig als Freund angeben. Dabei muss in dieser ersten Version das als Freund angegebene Kind nicht zustimmen.

Die Schule verwaltet dann eine Liste aller Kinder, die an dem Netzwerk teilnehmen.

Auszüge aus den Dokumentationen der Klassen `Kind` und `Schule` befinden sich im Anhang.

a) In den Objekten der Klasse `Kind` werden die Freunde eines Kindes in einer Liste unter dem Namen `freunde` (Klasse `List`) verwaltet. In den Objekten der Klasse `Schule` werden alle Kinder (Klasse `Kind`) in einer Liste unter dem Namen `mitglieder` (Klasse `List`) verwaltet.

Überführen Sie die Dokumentationen der Klassen `Kind` *und* `Schule` *in ein Implementationsdiagramm.*

(6 Punkte)

Name: _____

b) Hier sehen Sie einen Ausschnitt aus dem Quelltext der Klasse `Kind`:

```
1 public class Kind {

2   private String vorname;
3   private String nachname;
4   private List freunde;
    ...
5   public void fuegeFreundHinzu(Kind pKind) {
6     freunde.append(pKind);
7   }
    ...
8 }
```

Mit der so implementierten Methode `fuegeFreundHinzu` ist es möglich, ein Kind mehrfach zu der Freundesliste hinzuzufügen. Auch sich selber kann man als Freund wählen. Weiterhin sind die Freunde nicht alphabetisch (nach Nachnamen und Vornamen aufsteigend) in der Liste verwaltet.

Erweitern Sie die Implementation der Methode `fuegeFreundHinzu` *so, dass*

- *Kinder nicht mehrfach als Freund hinzugefügt werden können,*
- *ein Kind sich nicht selber als Freund hinzufügen kann,*
- *die Freunde alphabetisch (nach Nachnamen und Vornamen aufsteigend) in die Liste eingefügt werden.*

(12 Punkte)

Name: _____

c) In der Klasse `Schule` ist eine Methode `wasLiefereIch` implementiert:

```
 1 public int wasLiefereIch(Kind pKind) {
 2   int xyz = 0;
 3   mitglieder.toFirst();
 4   while (mitglieder.hasAccess()) {
 5     Kind dieses = (Kind) mitglieder.getObject() ;
 6     List freunde = dieses.gibFreunde();
 7     freunde.toFirst();
 8     while (freunde.hasAccess()) {
 9       Kind jenes = (Kind) freunde.getObject();
10       if (pKind.gleicht(jenes)) {
11         xyz = xyz + 1;
12       }
13       freunde.next();
14     }
15     mitglieder.next();
16   }
17   return xyz;
18 }
```

Analysieren Sie die Methode und beschreiben Sie deren Funktionalität im Sachzusammenhang anhand eines Beispiels.

Geben Sie insbesondere an, welche Aufgabe die Schleife in den Zeilen 8 bis 14 und welche Bedeutung die Variable `xyz` *hat.*

(10 Punkte)

Name: _____

d) In der Klasse `Schule` muss es möglich sein, ein Kind zu löschen, z. B. wenn es die Schule verlässt.

Implementieren Sie in der Klasse `Kind` *die Methode* `loescheFreund` *mit dem Methodenkopf*

```
public void loescheFreund(Kind pKind)
```

die das als Parameter übergebene Kind aus der Liste der Freunde löscht.

Implementieren Sie dann in der Klasse `Schule` *die Methode* `loescheKind` *mit dem Methodenkopf*

```
public void loescheKind(Kind pKind)
```

die das als Parameter übergebene Kind aus der Mitgliederliste der Schule sowie mit der oben entwickelten Methode `loescheFreund` *aus den Freundeslisten aller Kinder löscht.*

(12 Punkte)

e) In dem bisher beschriebenen Netzwerk kann ein Kind A ein anderes Kind B zum Freund wählen, ohne dass B zustimmt. Dieses soll so abgeändert werden, dass ein Kind A den Wunsch äußern kann, dass es Freund eines Kindes B werden kann. Kind A wird aber erst als Freund registriert, wenn Kind B die Freundschaft bestätigt. Für jedes Kind können beliebig viele Freundschaftswünsche vorliegen.

Erweitern Sie das Netzwerk so, dass die oben beschriebene Funktionalität erreicht werden kann.

Geben Sie dazu die Änderungen in den Klassen an, es dürfen auch Klassen hinzugefügt werden. Dokumentieren Sie neue bzw. veränderte Methoden.

(10 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anhang

Auszug aus der Dokumentation der Klasse Kind

Ein Objekt dieser Klasse verwaltet ein Kind.

Konstruktor `Kind(String pName, String pVorname)`
Ein neues Objekt der Klasse `Kind` mit den angegebenen Werten wird erstellt. Das Kind hat (noch) keinen Freund.

Anfrage `String gibVorname()`
liefert den Vornamen des Kindes.

Anfrage `String gibNachname()`
liefert den Nachnamen des Kindes.

Auftrag `void fuegeFreundHinzu(Kind pKind)`
`pKind` wird an die Liste der Freunde dieses Kindes angehängt.

Auftrag `void loescheFreund(Kind pKind)`
siehe Aufgabenteil d).

Anfrage `List gibFreunde()`
liefert einen Verweis auf die Liste (Objekt der Klasse `List`), in der die Freunde dieses Kindes verwaltet werden.

Anfrage `boolean gleicht(Kind pKind)`
liefert genau dann `true`, wenn Vor- und Nachname dieses Kindes mit denen von `pKind` übereinstimmen.

Auszug aus der Dokumentation der Klasse Schule

Ein Objekt dieser Klasse verwaltet die Schüler der Schule, die an dem sozialen Netzwerk teilnehmen.

Konstruktor `Schule()`
Ein neues Objekt vom Typ `Schule` wird erstellt. Die Liste der verwalteten Kinder (im Folgenden Mitgliederliste genannt) ist leer.

Auftrag `void kindHinzu(Kind pKind)`
Das als Parameter übergebene Objekt `pKind` wird der Mitgliederliste hinzugefügt, sofern es noch nicht in der Mitgliederliste enthalten ist.

Anfrage `boolean istEnthalten(Kind pKind)`
liefert genau dann `true`, wenn `pKind` in der Mitgliederliste enthalten ist.

Auftrag `void loescheKind(Kind pKind)`
siehe Aufgabenteil d)

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 2013

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 2013

1. *Inhaltliche Schwerpunkte*
   Objektorientiertes Modellieren und Implementieren von kontextbezogenen Anwendungen
   - Konzepte des objektorientierten Modellierens
   - Datenstrukturen
     - Lineare Strukturen mit den Akzenten
       Lineare Liste

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

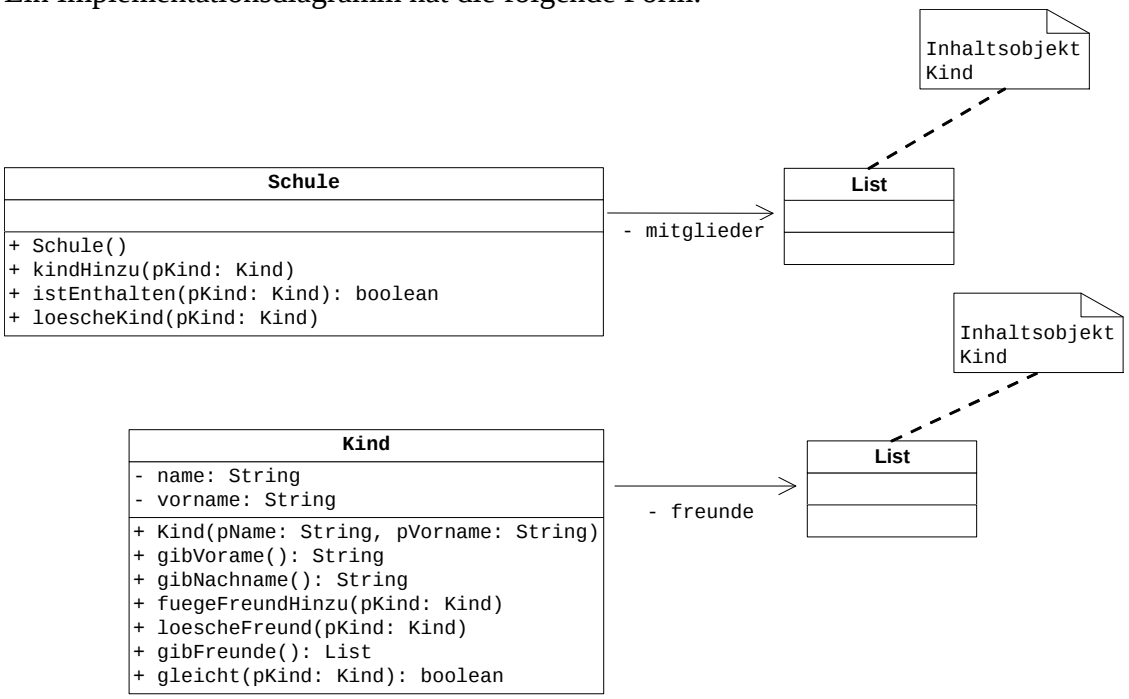
[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösungen

Teilaufgabe a)

Ein Implementationsdiagramm hat die folgende Form:



Nicht notwendig ist evtl. die zweifache Angabe des Klassendiagramms für die Klasse `List` mit dem zugehörigen Inhaltsobjekt `Kind`.

Teilaufgabe b)

Es bietet sich an, Hilfsmethoden zu implementieren, um den Programmcode übersichtlicher zu gestalten. Dann könnte eine Implementation folgendermaßen erfolgen:

```
private void alphabetischHinzu(Kind pKind) {
  freunde.toFirst();
  boolean hinzugefuegt = false;
  while (!hinzugefuegt && freunde.hasAccess()) {
    Kind dieses = (Kind)freunde.getObject();
    String nn1 = pKind.gibNachname();
    String nn2 = dieses.gibNachname();
    String vn1 = pKind.gibVorname();
    String vn2 = dieses.gibVorname();

    if ((nn1.compareTo(nn2) < 0) ||
         nn1.compareTo(nn2) == 0 && vn1.compareTo(vn2) < 0) {
      freunde.insert(pKind);
      hinzugefuegt = true;
    }
    freunde.next();
  }
  if (!hinzugefuegt) {
    freunde.append(pKind);
  }
}

private boolean istFreund (Kind pKind) {
  freunde.toFirst();
    while (freunde.hasAccess()) {
      Kind dies = (Kind)freunde.getObject();
      if (dies.gleicht(pKind)) {
        return true;
      }
      freunde.next();
  }
  return false;
}

public void fuegeFreundHinzu (Kind pKind) {
  if (!istFreund (pKind) && !this.gleicht(pKind)) {
    alphabetischHinzu(pKind);
  }
}
```

Teilaufgabe c)

Die Methode findet heraus, wie viele Kinder dieses Kind als Freund haben. Die innere Schleife durchläuft dabei die Liste der Freunde eines (in der äußeren Schleife festgelegten) Kindes, und inkrementiert den Wert der Variablen `xyz`, wenn es als Freund gewählt wurde.

Die Variable `xyz` verwaltet also die Zahl der Kinder, die das als Parameter übergebene Kind als Freund haben.

Die Methode könnte also dazu dienen, die Beliebtheit eines Kindes zu erforschen. Hierzu sollte ein passendes Beispiel angegeben werden.

Teilaufgabe d)

In der Klasse `Kind` kann die Methode `loescheFreund` folgendermaßen implementiert werden:

```
public void loescheFreund(Kind pKind) {
  freunde.toFirst();
  boolean geloescht = false;
  while (!geloescht && freunde.hasAccess()) {
    Kind jenes = (Kind)freunde.getObject();
    if (jenes.gleicht(pKind)) {
      freunde.remove();
      geloescht = true;
    }
    else {
      freunde.next();
    }
  }
}
```

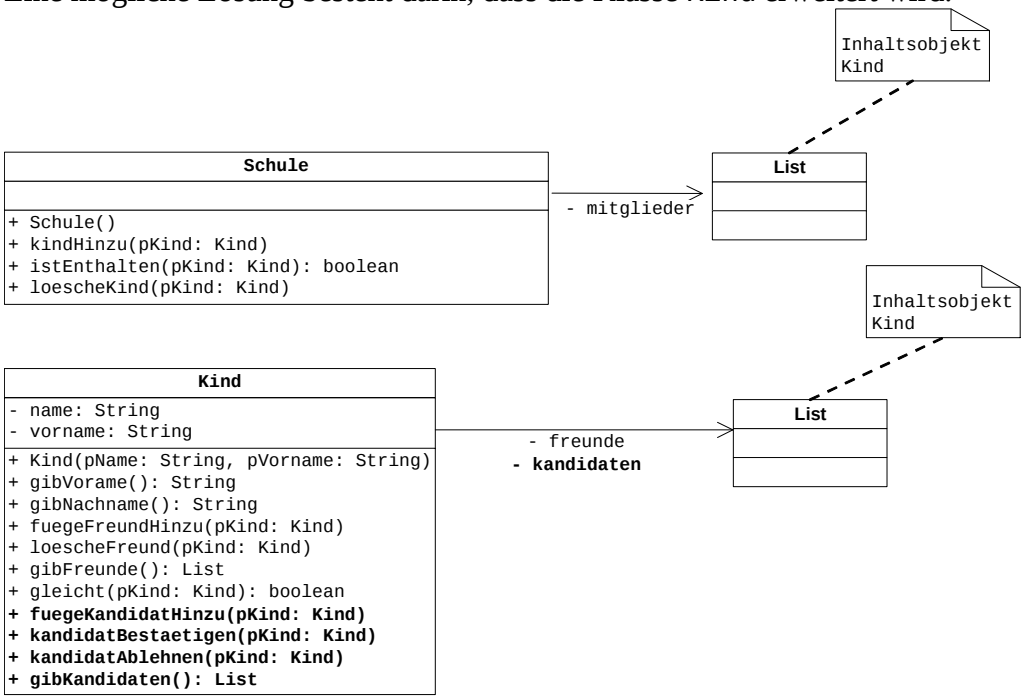
Dann kann die Methode `loescheKind` in der Klasse `Schule` erfolgen:

```
public void loescheKind(Kind pKind) {
  mitglieder.toFirst();
  while (mitglieder.hasAccess()) {
    Kind dieses = (Kind)mitglieder.getObject();
    dieses.loescheFreund(pKind);
    mitglieder.next();
  }

  boolean geloescht = false;
  mitglieder.toFirst();
  while (!geloescht && mitglieder.hasAccess()) {
    Kind jenes = (Kind)mitglieder.getObject();
    if (pKind.gleicht(jenes)) {
      mitglieder.remove();
      geloescht = true;
    } else
      mitglieder.next();
  }
}
```

Teilaufgabe e)

Eine mögliche Lösung besteht darin, dass die Klasse `Kind` erweitert wird.



Die Ergänzungen sind im Implementationsdiagramm durch Fettdruck gekennzeichnet.

Ein Kind, das Freund eines anderen Kindes werden will, wird als Kandidat bezeichnet.

Die Klasse `Kind` erhält zusätzlich eine Liste, in der die Kandidaten, die Freunde werden wollen, verwaltet werden.

Dokumentation der neuen Methoden:

Auftrag **`void fuegeKandidatHinzu(Kind pKind)`**
`pKind` wird in die Liste der Kandidaten eingefügt.

Auftrag **`void kandidatBestaetigen(Kind pKind)`**
Die Freundschaft wird bestätigt. `pKind` wird aus der Kandidatenliste gelöscht und in die Freundesliste eingetragen.

Auftrag **`void KandidatAblehnen(Kind pKind)`**
Die Freundschaft wird nicht bestätigt. `pKind` wird aus der Kandidatenliste gelöscht.

Anfrage **`List gibKandidaten()`**
Es wird eine Liste aller Kandidaten zurückgeliefert, die als Inhaltobjekte Objekte der Klasse `Kind` hat.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	überführt die Klasse `Kind` in den zugehörigen Diagrammteil.	2
2	überführt die Klasse `Schule` in den zugehörigen Diagrammteil.	2
3	fügt die Klasse `List` dem Diagramm hinzu.	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 b)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	erweitert den Code so, dass ein Kind sich nicht selber als Freund wählen kann.	2
2	erweitert den Code um einen Programmteil, der prüft, ob ein Kind bereits als Freund eingetragen ist.	3
3	erweitert den Code so, dass ein bereits als Freund eingetragenes Kind nicht erneut eingetragen wird.	2
4	implementiert eine Schleife, in der das Kind alphabetisch eingefügt wird.	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 c)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die Bedeutung der Schleife in den Zeilen 8 bis 14 an.	4
2	gibt die Bedeutung der Variablen `xyz` an.	2
3	analysiert die Methode, indem die Funktionalität im Sachzusammenhang angegeben 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 d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	implementiert in der Klasse `Kind` die Methode `loescheFreund`.	4
2	implementiert in der Klasse `Schule` in der Methode `loescheKind` eine Schleife, die das angegebene Kind aus der Freundesliste aller Kinder entfernt.	4
3	implementiert in der Klasse `Schule` in der Methode `loescheKind` den Programmteil, mit dem das angegebene Kind aus der Mitgliederliste entfernt 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	gibt die Erweiterungen an, die die Funktionalität ermöglichen.	2
2	gibt die Änderungen in den Klassen an.	4
3	dokumentiert die neuen bzw. veränderten Methoden.	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	überführt die Klasse …	2			
2	überführt die Klasse …	2			
3	fügt die Klasse …	2			
	Summe Teilaufgabe a)	**6**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erweitert den Code …	2			
2	erweitert den Code …	3			
3	erweitert den Code …	2			
4	implementiert eine Schleife …	5			
	Summe Teilaufgabe b)	**12**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt die Bedeutung …	4			
2	gibt die Bedeutung …	2			
3	analysiert die Methode …	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	implementiert in der …	4			
2	implementiert in der …	4			
3	implementiert in der …	4			
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt die Erweiterungen …	2			
2	gibt die Änderungen …	4			
3	dokumentiert die neuen …	4			
	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 2013

Informatik, Leistungskurs

Aufgabenstellung:

In einem ländlichen Neubaugebiet müssen ausgehend von einer Vermittlungsstelle (V) mehrere Haushalte mit einem DSL-Anschluss versorgt werden. Die sogenannte „letzte Meile“ verlegt ein bekanntes Telekommunikationsunternehmen.
Mit „letzte Meile“ wird umgangssprachlich der letzte Abschnitt einer Leitung von der Vermittlungsstelle zu einem Teilnehmer bezeichnet; ihre offizielle Bezeichnung lautet „Teilnehmeranschlussleitung“ (TAL). Im Bereich der Kommunikation spielt die „letzte Meile“ bei den Übertragungsraten im Internet eine wichtige Rolle. Die Leitungslänge bis zur Vermittlungsstelle bestimmt die Übertragungsrate.
Zur Vereinfachung beschränkt sich die Aufgabe auf 6 Einfamilienhäuser. Der folgende Graph gibt die Entfernungen in Metern an.

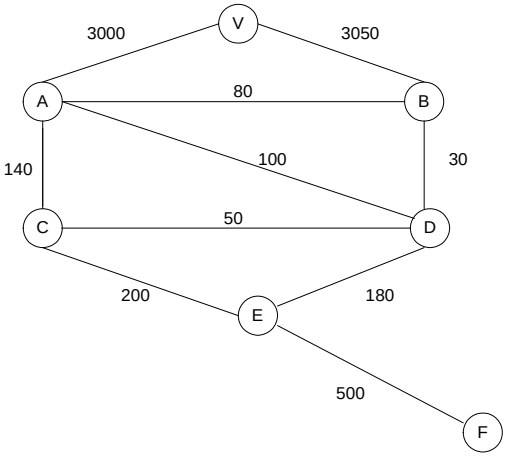


Abbildung 1

Name: _____

a) Der Graph soll mit Hilfe der Klassen `Graph` und `GraphNode` modelliert werden. Die Klasse `DSLGraph` hat u. a. einen Konstruktor, der den Beispielgraph zu Testzwecken implementiert.

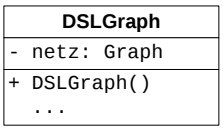


Abbildung 2

Geben Sie eine Implementation des Konstruktors der Klasse `DSLGraph` *an, der den Teilgraph mit den Knoten* `V, A, B` *und den Kanten zwischen den Knoten* `V, A, B` *erzeugt.*

Es sollen die Informationen, die zu einer Kante gehören, in Objekten einer Klasse `Kante` zusätzlich verwaltet werden. Die Dokumentation der Klasse `Kante` ist im Anhang.

Implementieren Sie die Klasse `Kante`.

(8 Punkte)

b) Die Einfamilienhäuser sollen mit der bestmöglichen Übertragungsrate von der Vermittlungsstelle versorgt werden. Ausschlaggebend für die Übertragungsrate ist die Leitungslänge von der Vermittlungsstelle bis zum Haushalt.

Erläutern Sie beispielhaft ein Verfahren zur Bestimmung der Länge der kürzesten Leitung von der Vermittlungsstelle V zum Haus F (siehe Abbildung 1) und wenden Sie es an.

(14 Punkte)

c) Ausgehend von einem Haus soll das „nächstgelegene“ Nachbarhaus bestimmt werden. Es kommen aber nur bestimmte Häuser in der Umgebung in Frage. Die Knoten, die diese Häuser repräsentieren, sind nicht markiert.

Die Methode `minUnmarkiert` bestimmt den unmarkierten Nachbarknoten des markierten Knotens `pNode`, der die kürzeste Entfernung zu `pNode` hat. Falls kein unmarkierter Knoten existiert, soll `null` zurückgegeben werden.

Hinweis: Die Konstante `Double.MAX_VALUE` gibt den größten Wert des Datentyps `double` an.

Implementieren Sie die Methode `minUnmarkiert` *der Klasse* `DSLGraph` *mit der Kopfzeile*

```
public GraphNode minUnmarkiert(GraphNode pNode) .
```

(10 Punkte)

Name: _____

d) Gegeben sind die Implementationen der Methoden `minKante` und `gibNetz` der Klasse `DSLGraph`. Die Methoden benutzen das Ergebnis der Methode `minUnmarkiert`. Die Funktionalität dieser Methode ist in c) beschrieben. Die Dokumentation der Klasse `Kante` ist im Anhang.

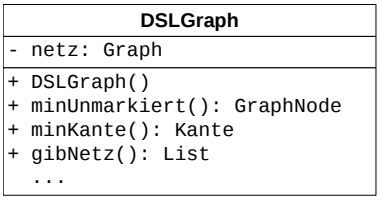


Abbildung 3

Das Telekommunikationsunternehmen entwirft mit Hilfe der folgenden Algorithmen schrittweise das Versorgungsnetz.

```
 1   public Kante minKante() {
 2     GraphNode anfang = null;
 3     GraphNode ende = null;
 4     double best = Double.MAX_VALUE;
 5     List knoten = netz.getNodes();
 6     knoten.toFirst();
 7     while (knoten.hasAccess()) {
 8       GraphNode lauf = (GraphNode) knoten.getObject();
 9       GraphNode min = minUnmarkiert(lauf);
10       if (min != null) {
11         double aktWert = netz.getEdgeWeight(lauf,min);
12         if (lauf.isMarked() &&  aktWert <= best) {
13           anfang = lauf;
14           ende = min;
15           best = netz.getEdgeWeight(anfang,ende);
16         }
17       }
18       knoten.next();
19     }
20     if (anfang == null || ende == null)
21       return null;
22     else
23       return new Kante(anfang,ende,best);
24   }
```

Name: _____

```
25   public List gibNetz() {
26     netz.resetMarks();
27     List vNetz = new List();
28     GraphNode start = netz.getNode("V");
29     start.mark();
30     while (!netz.allNodesMarked()) {
31       Kante mKante = minKante();
32       if (mKante != null) {
33         GraphNode knoten = mKante.gibKnoten2();
34         knoten.mark();
35         vNetz.append(mKante);
36       }
37     }
38     return vNetz;
39   }
```

Die Konstante `Double.MAX_VALUE` gibt den größten Wert des Datentyps `double` an.

Analysieren Sie die Methoden `minKante` *und* `gibNetz` *am Beispiel Abbildung 1, indem Sie die Kanten bestimmen, die durch die Methode* `gibNetz` *in die Liste eingetragen werden.*

Ermitteln Sie die Lösungsstrategie der Methode `gibNetz`*, die die Methoden* `minKante`*,* `minUnmarkiert` *benutzt.*

Beurteilen Sie die Strategie unter dem Gesichtspunkt einer Versorgung des Wohngebietes mit schnellen DSL-Anschlüssen.

(18 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anhang

Dokumentation der Klasse `Kante`

Objekte der Klasse `Kante` verwalten die Daten, die zu einer Kante gehören.

Konstruktor **`Kante(GraphNode pKnoten1, GraphNode pKnoten2, double pGewicht)`**

Eine Kante wird erzeugt, die Daten `pKnoten1`, `pKnoten2`, `pGewicht` werden gespeichert.

Anfrage **`GraphNode gibKnoten1()`**

Der Knoten, der in der Parameterliste des Konstruktors als erster Parameter angegeben worden ist, wird zurückgegeben.

Anfrage **`GraphNode gibKnoten2()`**

Der Knoten, der in der Parameterliste des Konstruktors als zweiter Parameter angegeben worden ist, wird zurückgegeben.

Anfrage **`double gibGewicht()`**

Die Anfrage liefert das Gewicht der Kante.

Anfrage **`GraphNode gibNachbarknoten(GraphNode pKnoten)`**

Die Anfrage liefert den weiteren Knoten ungleich `pKnoten` der Kante. Falls `pKnoten` nicht zur Kante gehört, wird `null` zurückgegeben.

Name: _____

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 2013

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Objektorientiertes Modellieren und Implementieren von kontextbezogenen Anwendungen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2013

1. *Inhaltliche Schwerpunkte*
   Objektorientiertes Modellieren und Implementieren von kontextbezogenen Anwendungen
   - 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)

Implementation des Konstruktors:

```
public DSLGraph() {
  netz = new Graph();
  netz.addNode(new GraphNode("V"));
  netz.addNode(new GraphNode("A"));
  netz.addNode(new GraphNode("B"));
  netz.addEdge(netz.getNode("V"),
               netz.getNode("A"),3000);
  netz.addEdge(netz.getNode("V"),
               netz.getNode("B"),3050);
  netz.addEdge(netz.getNode("A"),
               netz.getNode("B"),80);
}
```

Implementation der Klasse `Kante`:

```
public class Kante {
  private GraphNode knoten1;
  private GraphNode knoten2;
  private double gewicht;

  public Kante(GraphNode pKnoten1, GraphNode pKnoten2,
               double pGewicht) {
    knoten1 = pKnoten1;
    knoten2 = pKnoten2;
    gewicht = pGewicht;
  }

  public GraphNode gibKnoten1() {
    return knoten1;
  }

  public GraphNode gibKnoten2() {
    return knoten2;
  }

  public double gibGewicht() {
    return gewicht;
  }
```

```
  public GraphNode gibNachbarknoten(GraphNode pKnoten) {
    if (pKnoten == knoten1) {
      return knoten2;
    } else {
      if (pKnoten == knoten2) {
        return knoten1;
      } else {
        return null;
      }
    }
  }

}
```

Teilaufgabe b)

Die Verbindungsgeschwindigkeit ist von der Entfernung des Hauses von der Vermittlungsstelle abhängig. Mit Hilfe des Dijkstra-Verfahrens kann der kürzeste Weg von der Vermittlungsstelle zum Haus ermittelt werden.

Mit Hilfe des Dijkstra-Algorithmus werden, ausgehend von einem Startknoten, die kürzesten Wege zu allen anderen Knoten des Graphen gesucht. Eine mögliche Formulierung des Dijkstra-Algorithmus:

Für jeden Knoten müssen drei Informationen verwaltet werden:
- besucht oder noch nicht besucht
- aktuelle Distanz zum Startknoten
- Vorgängerknoten auf dem günstigsten Weg

Folgende Schritte werden durchgeführt:
Vorbereitung
- Der Startknoten erhält das Pfadgewicht 0.
- Alle anderen Knoten erhalten als Anfangswert das Pfadgewicht „unendlich“, d. h., ein möglichst hoher Wert wird gesetzt.
- Der Startknoten mit Pfadgewicht wird in eine Prioritätenwarteschlange eingetragen.

Solange die Prioritätenwarteschlange nicht leer ist und der Zielknoten nicht aus der Prioritätenschlange entnommen ist, sind folgende Schritte durchzuführen:
- Entnahme des ersten Knotens aus der Prioritätenwarteschlange
- Jeder nicht besuchte Nachbarknoten mit Pfadgewicht wird eingetragen und der Bearbeitungsvorgänger aktualisiert, falls sich ein kleineres Pfadgewicht ergibt.

Gesucht wird der Weg von V nach F
Für jeden Knoten des Graphen wird ein Distanzwert d (Distanz zum Startknoten) und ein Verweis auf den Vorgängerknoten abgespeichert.

Zu Beginn des Verfahrens hat der Startknoten den Distanzwert 0, alle anderen Knoten den Wert unendlich.

Start: `V(0,null)`, `V` gilt als besucht.

`V(0,null)`

1. Schritt:
Alle nicht besuchten Nachbarknoten von `V`:
`A(3000,V)`, `B(3050,V)`

`A(3000,V)`, `B(3050,V)`

2. Schritt:
Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `A(3000,V)`, `A` gilt als besucht.

Alle nicht besuchten Nachbarknoten von `A`:
`B(3080,A)` keine Verbesserung
`C(3140,A)`
`D(3100,A)`,

`B(3050,V), D(3100,A), C(3140,A)`

3. Schritt:
Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `B(3050,V)`, `B` gilt als besucht.

Alle nicht besuchten Nachbarknoten von `B`:
`D(3080,B)` wird angepasst

`D(3080,B), C(3140,A)`

4. Schritt:
Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `D(3080,B)`, `D` gilt als besucht.

Alle nicht besuchten Nachbarknoten von `D`:
`E(3260,D)`

`C(3130,D)` wird angepasst.

`C(3130,D)`, `E(3260,D)`

5. Schritt:
Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `C(3130,D)`, `C` gilt als besucht.
`E(3330,C)` keine Verbesserung

`E(3260,D)`

6. Schritt:
Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `E(3260,D)`, `E` gilt als besucht.

Alle nicht besuchten Nachbarknoten von `E`:
`F(3760,E)`

`F(3760,E)`

Es wird mit dem Knoten mit minimaler Bewertung weitergearbeitet, in diesem Fall `F(3760,E)`. Der Zielknoten `F` ist erreicht.
Mit Hilfe der gespeicherten Vorgängerknoten kann der Weg bestimmt werden. `V,B,D, E, F` ist der gesuchte Weg in umgekehrter Reihenfolge. Die Entfernung beträgt 3760 m.

Der kürzeste Weg kann ebenso mit Hilfe des Backtracking-Verfahrens bestimmt werden.

Teilaufgabe c)

Implementierung der Methode `minUnmarkiert`:

```
public GraphNode minUnmarkiert(GraphNode pNode) {
  List nachbarn = netz.getNeighbours(pNode);
  GraphNode minKnoten = null;
  double minWert = Double.MAX_VALUE;
  nachbarn.toFirst();
  while (nachbarn.hasAccess()) {
    GraphNode aktuell = (GraphNode) nachbarn.getObject();
    double aktWert = netz.getEdgeWeight(pNode,aktuell);
    if (!aktuell.isMarked() &&  aktWert <= minWert) {
      minKnoten = aktuell;
      minWert = netz.getEdgeWeight(pNode,aktuell);
    }
    nachbarn.next();
  }
  return minKnoten;
}
```

Teilaufgabe d)

Wählt man als Startknoten V, so erhält die Liste die folgenden Kanten:
V - A 3000
A - B 80
B - D 30
D - C 50
D - E 180
E - F 500

Der Graph ist ein Teilgraph des gegebenen Graphen, der alle Knoten miteinander verbindet.

Beschreibung des Algorithmus:
Man startet mit einen Knoten und markiert ihn. Alle anderen Knoten sind zu Beginn unmarkiert.
Man wählt die Kante mit minimalem Gewicht, von der ein Knoten der Kante markiert und der andere Knoten unmarkiert ist.

Der unmarkierte Knoten wird markiert.

Das Verfahren wird so lange durchgeführt, bis alle Knoten markiert sind.

Durch den Algorithmus wird ein Graph bestimmt, der alle Knoten miteinander verbindet und in dem die Summe aller Kantenlängen geringer ist als beim gegebenen Graphen. Das Versorgungsnetz ist kleiner und für den Netzbetreiber dadurch kostengünstiger als das ursprüngliche Netz. Für eine DSL-Versorgung kann es aber ungünstiger werden, da die Wege zwischen zwei Knoten länger werden könnten. Zum Beispiel ist die kürzeste Weglänge von V nach F im ursprünglichen Netz 3760 Meter (vgl. Teilaufgabe b) und in dem verkleinerten Netz 3790 Meter. Dadurch kann sich im schlimmsten Fall der DSL-Standard für einzelne Teilnehmer verschlechtern, denn die Leitungslänge bis zur Vermittlungsstelle bestimmt den DSL-Standard.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt eine Implementation des Konstruktors an.	2
2	gibt eine Implementation der Klasse `Kante` an.	6
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	bestimmt ein passendes Verfahren zur Ermittlung des kürzesten Wegs.	2
2	erläutert das gewählte Verfahren.	4
3	ermittelt schrittweise den kürzesten Weg.	8
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	implementiert die Schleife.	3
2	implementiert die Abfrage in der Schleife.	4
3	implementiert die Methode formal richtig.	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 d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	analysiert die Methoden und bestimmt dazu Liste der Kanten.	8
2	ermittelt und beschreibt die Lösungsstrategie.	6
3	beurteilt die Strategie auf den Kontext bezogen.	4
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	gibt eine Implementation …	2			
2	gibt eine Implementation …	6			
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	bestimmt ein passendes …	2			
2	erläutert das gewählte …	4			
3	ermittelt schrittweise den …	8			
	Summe Teilaufgabe b)	**14**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Schleife.	3			
2	implementiert die Abfrage …	4			
3	implementiert die Methode …	3			
	Summe Teilaufgabe c)	**10**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methoden …	8			
2	ermittelt und beschreibt …	6			
3	beurteilt die Strategie …	4			
	Summe Teilaufgabe d)	**18**			

	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 2013

Informatik, Leistungskurs

Aufgabenstellung:

Als es für Computer noch keine graphischen Oberflächen gab, die z. B. mit einer Maus zu steuern sind, konnten Computer ausschließlich über die Tastatur bedient werden. Das Betriebssystem stellte dabei eine große Vielfalt von Befehlen bereit, die über die Tastatur eingegeben werden mussten. Diese Befehle existieren in den aktuellen Betriebssystemen auch heute noch. Ähnlich wie bei der Programmierung müssen diese Befehle nach einem bestimmten Schema eingegeben werden. Im Folgenden soll exemplarisch eine vereinfachte Version des Befehls `dir` behandelt werden. Er listet Dateien und Unterverzeichnisse eines Verzeichnisses auf.

Der Befehl `dir` hat sogenannte *Optionen* und *Parameter*.

Sprache der Optionen und Parameter des Befehls `dir`

Beispiel:

```
-A-L/media
```

Jede Option beginnt immer mit einem Minuszeichen „-“. Es folgt genau ein Buchstabe. Optionen werden immer vor den Parametern angegeben.

Für den Befehl `dir` existieren die Optionen `A` (alle Dateien und Verzeichnisse auflisten, auch versteckte) und `L` (für die Darstellung einer Datei oder eines Verzeichnisses pro Zeile). Die Optionen dürfen beliebig oft und in beliebiger Reihenfolge angegeben oder weggelassen werden.

Als Parameter wird entweder nur der *Name des Verzeichnisses* oder ein *Verzeichnispfad* angegeben, dessen Inhalt aufgelistet werden soll. Der Parameter muss angegeben werden. Im obigen Beispiel ist der Teil `/media` nach den Optionen `-A` und `-L` ein Verzeichnispfad.

Name: _____

Der Name eines Verzeichnisses besteht aus (mindestens einem) Buchstaben von a bis z. Der Verzeichnispfad beginnt mit einem Schrägstrich „/“ oder einem Verzeichnisnamen und wird mit den Namen beliebig vieler Verzeichnisse fortgesetzt. Die Verzeichnisnamen werden ebenfalls durch einen Schrägstrich voneinander getrennt. Der Schrägstrich darf nicht zweimal hintereinander vorkommen. Er darf aber am Ende eines Verzeichnispfads stehen. Das oberste Verzeichnis wird durch den Schrägstrich „/“ angegeben.

Beispiele für Verzeichnispfade:

```
home/ich/musik/
home/ich
/etc/sshd
/var/log/
/
```

Werden solche Optionen und Parameter eingegeben, muss überprüft werden, ob sie syntaktisch korrekt sind. Diese Überprüfung kann durch einen deterministischen endlichen Automaten dargestellt werden.

Der folgende Automat überprüft angeblich die Eingabe *nur für Optionen und Parameter* des Befehls `dir`.

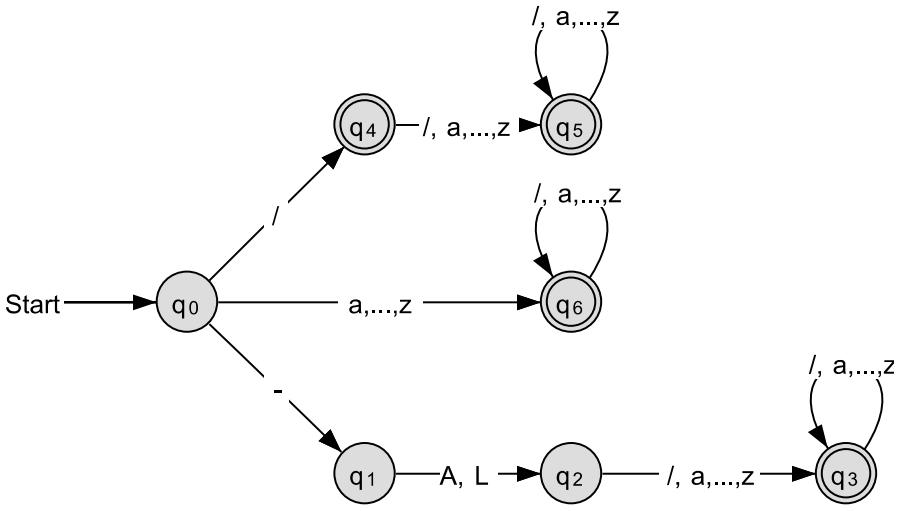


Abbildung 1: Automat 1

Nicht dargestellte Übergänge führen in einen Fehlerzustand, der im Graphen und in der Tabelle nicht dargestellt wird.

Name: _____

Der in Abbildung 1 dargestellte Automat 1 erfüllt zwei der für den Befehl `dir` festgelegten Bedingungen der Sprache der Optionen und Parameter nicht.

a) *Geben Sie für die folgenden Eingaben die Zustandsfolgen von Automat 1 an.*

Geben Sie an, ob die Eingaben akzeptiert werden.

Erläutern Sie gegebenenfalls anhand Ihrer Ergebnisse, inwiefern diese mit den Bedingungen der oben dargestellten Sprache der Optionen und Parameter nicht übereinstimmen.

```
–Amusik
/var//inf/
–L/
–A–L/
```

Überführen Sie anschließend den Automaten in einen neuen deterministischen endlichen Automaten, der die beschriebenen Bedingungen für den Befehl `dir` *erfüllt.*

(17 Punkte)

b) *Entwerfen Sie eine reguläre Grammatik für die oben (Seite 1) dargestellte Sprache der Optionen und Parameter.*

Beschreiben Sie Ihr Vorgehen.

(16 Punkte)

Name: _____

c) Der in Abbildung 1 dargestellte Automat 1 enthält einige überflüssige Zustände. Der folgende Automat wurde entworfen, um diesen Automaten zu vereinfachen.

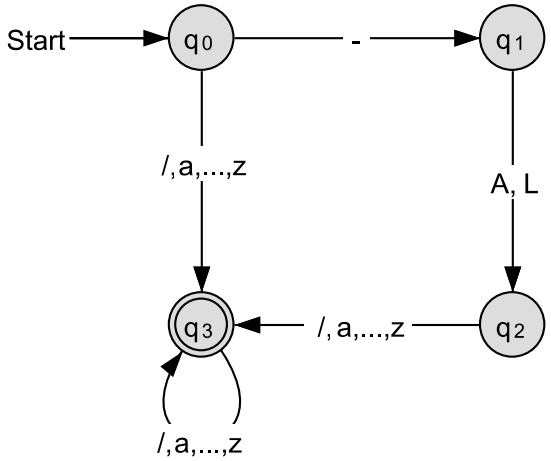


Abbildung 2: Automat 2

Nicht dargestellte Übergänge führen in einen Fehlerzustand, der im Graphen und in der Tabelle nicht dargestellt wird.

Begründen Sie, dass beide Automaten die gleiche Sprache erkennen.

(9 Punkte)

Name: _____

d) Eine formale Möglichkeit, die erkannten Sprachen zweier Automaten zu vergleichen, bieten die regulären Ausdrücke. Einen regulären Ausdruck kann man als eine formelhafte präzise Beschreibung einer regulären Sprache verstehen – ähnlich zu einem mathematischen Term.

Reguläre Ausdrücke sind gemäß der folgenden vier Regeln aufgebaut:

1. Jedes einzelne Eingabezeichen des Eingabealphabets ist ein regulärer Ausdruck. Außerdem ist auch das leere Wort ε ein regulärer Ausdruck. Das leere Wort ist ein Symbol für ein Zeichen, das wegfallen kann (siehe auch Beispiel 3).

   Beispiele: a ist ein regulärer Ausdruck, genauso wie / oder ε.

2. Die Aneinanderreihung zweier regulärer Ausdrücke ist ein regulärer Ausdruck. Diese Regel ist die Basis, um aus mehreren Zeichen des Alphabets Wörter zu erzeugen:

   Schreibweise: αβ oder gegebenenfalls: (αβ)

   Beispiele: Die Zeichen a und b sind reguläre Ausdrücke. Dann ist auch das Wort ab ein regulärer Ausdruck.
   Sind die Worte ab und gt reguläre Ausrücke, dann ist auch abgt ein regulärer Ausdruck.

3. Die Auswahl aus zwei regulären Ausdrücken ist ein regulärer Ausdruck. Diese Regel bedeutet, dass entweder der erste Ausdruck oder der zweite Ausdruck zum Erzeugen eines Wortes gewählt werden kann: Entweder α oder β. Beide Ausdrücke zu wählen ist nicht möglich.

   Schreibweise: α|β oder gegebenenfalls (α|β)

   Beispiele:
   (a|abb) beschreibt die Sprache, die aus den beiden Worten a und abb besteht.
   a(**a**|**b**|**ε**)c beschreibt die Sprache, die aus den Worten a**a**c, a**b**c und ac besteht.
   Verändert wird bei diesem Beispiel also nur das mittlere Zeichen: Es ist entweder a oder b oder es fällt wegen des leeren Worts ε weg.

Name: _____

4. Die beliebige Wiederholung eines regulären Ausdrucks ist ein regulärer Ausdruck. Diese Regel sorgt dafür, dass ein regulärer Ausdruck unendlich viele Worte beschreiben kann. Dabei tritt der reguläre Ausdruck beliebig oft – eventuell auch keinmal – hintereinander auf.

   Schreibweise: `α*` oder gegebenenfalls `(α)*`

   Beispiele: `a*` beschreibt die Sprache, die alle Wörter enthält, die aus dem Zeichen `a` bestehen, und das leere Wort. Die Sprache aller Wörter aus den Zeichen `a` und `b` (und dem leeren Wort) wird durch den regulären Ausdruck `(a|b)*` beschrieben. Der reguläre Ausdruck `ab(ab)*` beschreibt die Sprache, die immer abwechselnd die Buchstaben `ab` enthält, also `ab`, `abab`, `abababab`, `ababababab`, …

Begründen Sie, dass die Sprache des in Abbildung 2 dargestellten Automaten mit dem regulären Ausdruck

`(ε|-(A|L)) (/|a|…|z)(/|a|…|z)*`

beschrieben werden kann.

(8 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Unterlagen für die Lehrkraft

Abiturprüfung 2013

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 2013

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
- Formale Sprachen: Reguläre Sprachen und ihre Grammatiken

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

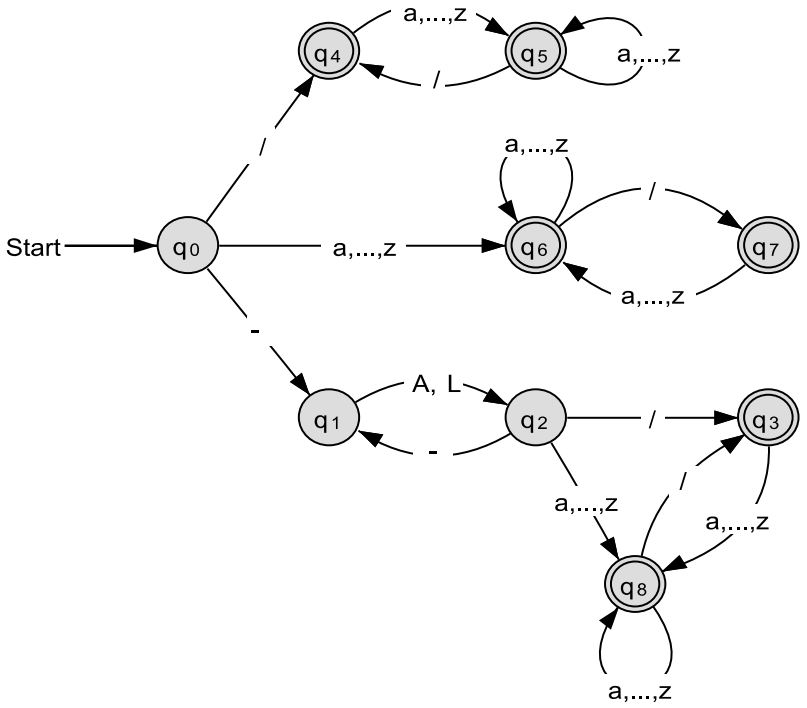
Teilaufgabe a)

`-Amusik`
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3 \rightarrow q_3 \rightarrow q_3 \rightarrow q_3$
Das Wort wird korrekterweise akzeptiert.

`/var//inf/`
$q_0 \rightarrow q_4 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5 \rightarrow q_5$
Das Wort wird fälschlicherweise akzeptiert. Hier wird der Bedingung nicht entsprochen, dass der Schrägstrich nicht zweimal direkt hintereinander auftreten darf. Solche Worte sollte der Automat nicht akzeptieren.

`-L/`
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$
Das Wort wird korrekterweise akzeptiert.

`-A-L/`
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow$ `Abbruch`
Das Wort wird fälschlicherweise nicht akzeptiert. Laut Vorgabe dürfen beliebig viele Optionen hintereinander aufgeführt werden, wobei jeder Buchstabe als Bezeichner einer Option als Präfix ein Minuszeichen erhält. Bei dem gegebenen Wort sind die beiden für `dir` zulässigen Optionen `A` und `L`, jeweils mit führendem Minuszeichen, hintereinander an der korrekten Position angegeben. Trotzdem gelangt der Automat in einen nicht definierten Zustand.



Dieser als Lösung angegebene Automat verwendet die ursprünglichen Zustände mit minimalen Änderungen. Ein vollkommen neuer Automat ist ebenso als korrekte Lösung zu werten, sofern er den Vorgaben entspricht.

Teilaufgabe b)

Terminale T = {a, … , z, /, –,A, L}
Nichtterminale N = {S, O, V, D}
Startsymbol: S

Produktionen (mögliche Lösung mit ε-Produktionen):
{S → -O | /D | aV | … | zV,
 O → AS | LS,
 D → aV | … | zV | ε,
 V → /D | aV | … | zV | ε }

Produktionen (mögliche Lösung ohne ε-Produktionen):
{S → / | a | … | z | -O | /D | aV | … | zV,
 O → AS | LS,
 D → a | … | z | aV | … | zV,
 V → / | a | … | z | /D | aV | … | zV }

Zunächst müssen die Optionen erzeugt werden. Da beliebig viele Angaben der beiden Optionen A und L mit führenden Minuszeichen angegeben werden können, muss vom Startzustand aus ein Kreis angelegt werden, der jeweils eine der beiden Optionen anlegen kann. Da Optionen auch entfallen können, muss der Kreislauf wieder zum Startsymbol führen. Da nach den Optionen auf jeden Fall ein Parameter folgen muss, gibt es in diesen Regeln stets ein Nichtterminal.

Beim Parameter ist zu beachten, dass ein Terminalzeichen nach dem Startsymbol für ein gültiges Wort ausreicht, so dass hier auch Regeln ohne Nichtterminale eingefügt werden müssen.
Nach dem Startsymbol ist eine fast beliebig lange Verkettung der Terminalen a bis z und / zu erzeugen. Zu beachten ist, dass der Schrägstrich nicht zweimal hintereinander vorkommen darf, weshalb ein weiteres Nichtterminal D eingeführt werden muss, welchem die gleichen Produktionen von V, ohne die mit einem Schrägstrich, zugeordnet sind.

Teilaufgabe c)

Ein formaler Beweis ist nicht gefordert. Eine verständliche und vollständige Begründung ist ausreichend. Hierzu können vielfältige Ansätze gewählt werden.

Die Übergänge zwischen den Zuständen q_0 über q_1 und q_2 bis q_3 sind identisch geblieben. Außerdem ist q_0 weiterhin der Startzustand und q_0 bis q_2 sind weiterhin keine Endzustände. Dieser Teil des Automaten wurde nicht verändert.

Änderung gibt es nur bei den Übergängen von q_0 zu q_3 durch den Schrägstrich und die Buchstaben. Der Automat 1 geht sofort jeweils in einen akzeptierenden Teil über, der beliebige weitere Kombinationen aus dem Schrägstrich und den Buchstaben akzeptiert. Bei diesem Automaten ist dieser Bereich nur in drei Unterbereiche mit den Zuständen q_4 und q_5, dem Zustand q_6 sowie dem Zustand q_3 aufgeteilt.
Der Automat 2 vereinigt diesen Bereich auf einen Zustand, der ebenfalls alle beliebigen Kombinationen aus Buchstaben und dem Schrägstrich akzeptiert. Der Automat 2 erkennt also die gleiche Sprache wie der Automat 1.

Teilaufgabe d)

Der erste Teil `(ε|-(A|L))` des Ausdrucks bezieht sich auf die Optionen des Befehls `dir`. Entweder wird keine Option – also sozusagen das leere Wort `ε` – eingegeben und der Automat bleibt zunächst in Zustand q_0, oder es wird eine Option eingegeben. Die Eingabe dieser Option beginnt zwingend mit einem `-`, das dem Übergang zu q_1 entspricht. Gefolgt wird es entweder von einem `A` oder von einem `L`, wie sich sowohl am Übergang von q_1 zu q_2 als auch dem regulären Ausdruck `(A|L)` entnehmen lässt.

Der zweite Teil `(/|a|…|z)(/|a|…|z)*` des Ausdrucks bezieht sich auf den Parameter des Befehls `dir`. Der Parameter muss wegen des Ausdrucks `(/|a|…|z)` aus mindestens einem der angegebenen Zeichen bestehen. Sie finden sich auch sowohl bei den Übergängen von q_0 zu q_3 als auch von q_2 zu q_3. Nun befindet sich der Automat in einem Endzustand. Die beliebig oft wiederholte Reihung der angegebenen Zeichen durch `(/|a|…|z)*` lässt sich in der Schleife im Endzustand q_3 erkennen. Auch die Auslassung dieser Zeichen, was beim *-Operator möglich ist, widerspricht nicht der Darstellung im Automaten, da der Endzustand bereits durch die vorhergehende Eingabe erreicht ist.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	gibt die Zustandsfolgen an.	4
2	gibt an, ob die Wörter akzeptiert werden.	2
3	erläutert für das zweite und vierte Wort den Widerspruch zu den Bedingungen.	4
4	überführt den 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	entwirft die Menge der Nichtterminale.	2
2	entwirft die Menge der Terminale.	2
3	entwirft die Menge der Produktionen.	6
4	beschreibt das Vorgehen.	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	begründet die Entsprechung des Startzustandes und der Endzustände.	3
2	begründet die Entsprechung der Übergänge.	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	begründet den ersten Teil `(ε|-(A|L))` des Ausdrucks.	4
2	begründet den zweiten Teil `(/|a|…|z)(/|a|…|z)*` des Ausdrucks.	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 Zustandsfolgen …	4			
2	gibt an, ob …	2			
3	erläutert für das …	4			
4	überführt den Automaten.	7			
	Summe Teilaufgabe a)	**17**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft die Menge …	2			
2	entwirft die Menge …	2			
3	entwirft die Menge …	6			
4	beschreibt das Vorgehen.	6			
	Summe Teilaufgabe b)	**16**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet die Entsprechung …	3			
2	begründet die Entsprechung …	6			
	Summe Teilaufgabe c)	**9**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet den ersten …	4			
2	begründet den zweiten …	4			
	Summe Teilaufgabe d)	**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

Name: _____

Abiturprüfung 2013

Informatik, Leistungskurs

Aufgabenstellung:

Bei einer Quizshow stellt der Showmaster der Kandidatin bzw. dem Kandidaten eine Frage und gibt ihm oder ihr 4 mögliche Antworten (mit a, b, c und d bezeichnet). Nur eine dieser Antworten ist richtig. Die Kandidatin bzw. der Kandidat entscheidet sich für eine der Antworten. Der Showmaster entscheidet dann, ob die Antwort richtig war oder nicht. Varianten dieses Spiels sollen als Netzwerkspiel implementiert werden.

Beispiel:

Frage: Was baute Konrad Zuse?

Antwortmöglichkeiten:
a) Kaffeemaschine
b) Telefon
c) Computer
d) Auto

Richtig ist Antwort c.

In einer ersten Version arbeitet der Quiz-Server nach folgendem Protokoll:

Tabelle 1:

Client an Server	**Server an Client**
`Baut Verbindung auf`	`+OK Herzlich willkommen bei unserem Spiel` `<sendet zufällig gewählte Frage mit vier Antwortmöglichkeiten>`

`ANTWORT <a, b, c oder d>`	`+OK Richtige Antwort`
	`<sendet zufällig gewählte Frage mit vier Antwortmöglichkeiten>`
	`oder`
	`+OK Falsche Antwort`
	`<sendet zufällig gewählte Frage mit vier Antwortmöglichkeiten>`

`ENDE`	`+OK Vielen Dank für die Teilnahme an unserem Spiel`
	`gespielte Fragen: <Anzahl>`
	`davon richtig beantwortet:<Anzahl>`
	`<beendet Verbindung>`

Name: _____

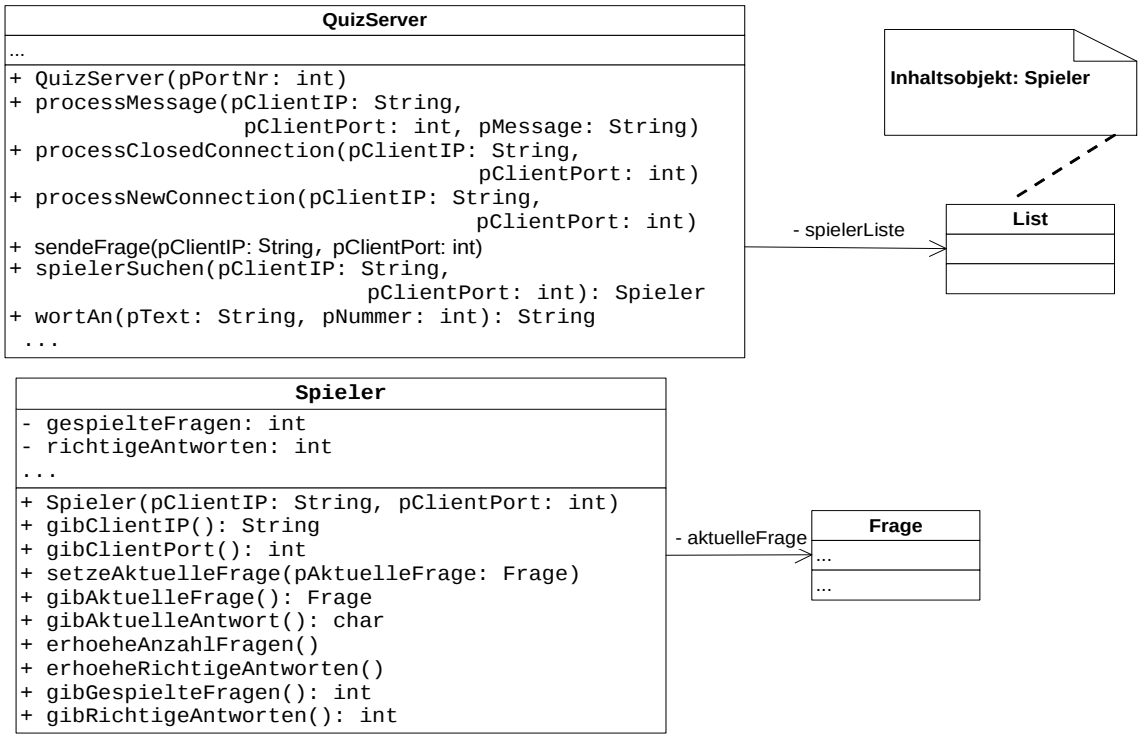
a) *Stellen Sie die Kommunikation zwischen Client und Server nach dem Protokoll aus Tabelle 1 in folgendem Fall dar: Der Client meldet sich an, erhält zufällig die Frage aus der Aufgabenstellung (Beispiel Seite 1), antwortet falsch, erhält zufällig wieder dieselbe Frage, antwortet richtig und meldet sich dann wieder ab.*

Beschreiben Sie mögliche Eingabefehler und erweitern sie das Protokoll um eine Fehlerbehandlung für diese.

(10 Punkte)

Zur Implementierung des Spiels werden die Klassen `QuizServer`, `Spieler` und `Frage` zur Verfügung gestellt. Die Klasse `QuizServer` implementiert einen Server und verwaltet die Spieler und die Fragen jeweils in einer Liste. In der Klasse `Spieler` werden sowohl die Anzahl der gespielten Fragen, als auch die Anzahl der richtigen Antworten verwaltet. Hier sehen Sie einen Ausschnitt des Implementationsdiagramms:

Diagramm 1:



Für die Aufgabe sind nur die Klassen `QuizServer` und `Spieler` von Bedeutung.
Sie finden Dokumentationen der für die Aufgabe wichtigen Teile im Anhang.

b) *Begründen Sie, dass die aktuelle Frage des jeweiligen Spielers (oder ein Verweis auf diese) zusammen mit dem Spieler gespeichert werden muss, wenn die Spieler individuelle Fragen erhalten sollen.*

(2 Punkte)

Name: _____

c) *Implementieren Sie die Methode* `processMessage` *der Klasse* `QuizServer` *gemäß dem in Tabelle 1 auf Seite 1 angegebenen Protokoll.*

Berücksichtigen Sie dabei die Dokumentation der Klasse `QuizServer` im Anhang. Eine Fehlerbehandlung ist nicht notwendig.

(12 Punkte)

d) Es soll in Zukunft sogenannten Administratoren des Spiels erlaubt sein, Fragen zu ergänzen. Daher muss man sich mit einem Namen und Passwort anmelden. Das Ergänzen von Fragen ist dann möglich, wenn man sich mit einem Namen angemeldet hat, der als Administrator geführt wird. Die Passwörter sollen verschlüsselt übergeben werden.

Ein neuer Kollege schlägt Ihnen eine Methode zur Verschlüsselung vor, die im Folgenden exemplarisch zur Verschlüsselung von Großbuchstaben implementiert ist:

```
 1  public String verschluesseln(String pText) {
 2    int j = 1;
 3    String hilf = "";
 4    String eingabe = pText + " ";
 5    String ausgabe = "";
 6    for (int i = 0; i < eingabe.length( ); i++){
 7      if (eingabe.charAt(i) == ' '){
 8        ausgabe = ausgabe + ' ';
 9        for (j= 0; j < hilf.length(); j++){
10          ausgabe = ausgabe + hilf.charAt(hilf.length()-j-1);
11        }
12        hilf = "";
13        j = 1;
14      }
15      else {
16        if ((int)(eingabe.charAt(i))+j > 90){
17          hilf = hilf + (char)((int)(eingabe.charAt(i))+j-26);
18        }
19        else {
20          hilf = hilf + (char)((int)(eingabe.charAt(i))+j);
21        }
22        j = j+1;
23      }
24    }
25    return ausgabe;
26  }
```

Name: _____

Ermitteln Sie den Rückgabewert der Methode `verschluesseln`*, wenn sie mit dem String* ‘`HEUTE IST ABI`‘ *aufgerufen wird und erläutern Sie anschließend die Funktionsweise des Verschlüsselungsverfahrens.*

Geben Sie an, ob es sich um ein mono- oder polyalphabethisches Verfahren handelt und beurteilen Sie die Sicherheit des Verfahrens.

Hinweis: Beachten Sie, dass die Großbuchstaben die ASCII-Codierungen 65 bis 90 haben. Eine Tabelle ist in Anlage 2.

(16 Punkte)

e) Zur Verschlüsselung wird in diesem Aufgabenteil das Verfahren von Vigenère verwendet. In der Anlage 1 finden Sie das Vigenère-Quadrat. Der Schüler ALI hat seinen Namen als Passwort verwendet. Es wurde mit einem Wort der Länge 3 nach dem Verfahren von Vigenère verschlüsselt. Als Chiffre erhielt er GHJ.

Bestimmen Sie den Schlüssel und bestimmen Sie anschließend einen Schlüssel, der aus dem Klartext `ABI` *ebenfalls das Chiffre* `GHJ` *erzeugt.*

Beurteilen Sie Sicherheit des Vigenère-Verfahrens, wenn der Schlüssel genauso lang ist wie der Text.

(10 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anlage: Das Vigenère-Quadrat:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Anlage 2: Ausschnitt aus der ASCII-Tabelle:

65	66	67	68	69	70	71	72	73	74	75	76	77
A	B	C	D	E	F	G	H	I	J	K	L	M

78	79	80	81	82	83	84	85	86	87	88	89	90
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Name: _____

Anhang

Die Klasse QuizServer

Die Klasse `QuizServer` implementiert einen Quiz-Server.

Konstruktor **`QuizServer(int pPortNr)`**

Nach dem Aufruf dieses Konstruktors bietet der Quiz-Server seinen Dienst über die Portnummer `pPortNr` an. Clients können sich nun mit dem Server verbinden. Eine Liste mit Fragen und eine (leere) Spielerliste wurden erzeugt.

Auftrag **`void processMessage(String pClientIP, int pClientPort, String pMessage)`**

Der Client mit der angegebenen IP und der angegebenen Portnummer hat dem Server eine Nachricht gesendet. Dieser reagiert gemäß dem Protokoll in der Aufgabenstellung.

Auftrag **`void processClosedConnection(String pClientIP, int pClientPort)`**

Der Client mit der IP-Adresse `pClientIP` und der Portnummer `pClientPort` wird gemäß Protokoll verabschiedet und aus der Spielerliste gelöscht.

Auftrag **`void processNewConnection(String pClientIP, int pClientPort)`**

Der Client mit der IP-Adresse `pClientIP` und der Portnummer `pClientPort` hat eine Verbindung zum Server aufgebaut. Der Server begrüßt den neuen Teilnehmer, nimmt ihn in die Spielerliste auf, bestimmt die erste Frage und sendet sie.

Auftrag **`void sendeFrage(String pClientIP, int pClientPort)`**

Eine Frage wird zufällig aus der Liste ausgewählt und an den Client mit der angegebenen IP-Adresse und der angegebenen Portnummer gesendet.

Anfrage **`Spieler spielerSuchen(String pClientIP, int pClientPort)`**

Diese Anfrage liefert den Spieler mit der IP-Adresse `pClientIP` und der Portnummer `pClientPort`. Gibt es keinen solchen Spieler, so liefert die Methode `null`.

Anfrage **`String wortAn(String pText, int pNummer)`**

Diese Anfrage liefert das Wort mit der angegebenen Wortnummer in dem angegebenen Text. Die Wörter sind dabei beginnend mit 1 nummeriert und durch Leerzeichen voneinander getrennt. Falls kein Wort an der angegebenen Wortnummer existiert, wird `null` zurückgegeben.

Name: _____

Die Klasse Spieler

In der Klasse **`Spieler`** wird die IP-Adresse, die Portnummer und die aktuelle Frage des Spielers gespeichert.

Konstruktor **`Spieler (String pClientIP, int pClientPort)`**
Nach dem Aufruf dieses Konstruktors wurden die in den Parametern übergebenen Daten in entsprechenden Zustandsvariablen gespeichert. Die aktuelle Frage des Spielers ist `null` .

Anfrage **`String gibClientIP()`**
Die IP-Adresse des Spielers wird zurückgegeben.

Anfrage **`int gibClientPort()`**
Die Portnummer des Spielers wird zurückgegeben.

Auftrag **`void setzeAktuelleFrage(Frage pAktuelleFrage)`**
Die aktuelle Frage des Spielers wird neu gesetzt.

Anfrage **`Frage gibAktuelleFrage()`**
Die aktuelle Frage des Spielers wird zurückgeliefert. Gibt es keine aktuelle Frage, so ist der Rückgabewert `null`.

Anfrage **`char gibAktuelleAntwort()`**
Die Nummer (als Buchstabe) der richtigen Antwort auf die aktuelle Frage des Spielers wird zurückgegeben.

Auftrag **`void erhoeheAnzahlFragen()`**
Die Anzahl der vom Spieler gespielten Fragen wird um eins erhöht.

Auftrag **`void erhoeheRichtigeAntworten()`**
Die Anzahl der vom Spieler richtig beantworteten Fragen wird um eins erhöht.

Anfrage **`int gibGespielteFragen()`**
Die Anzahl der vom Spieler gespielten Fragen wird zurückgegeben.

Anfrage **`int gibRichtigeAntworten()`**
Die Anzahl der vom Spieler richtig beantworteten Fragen wird zurückgegeben.

Name: _____

Die Klasse Server

Über die Klasse **`Server`** ist es möglich, eigene Serverdienste anzubieten, so dass Clients Verbindungen gemäß dem TCP/IP-Protokoll hierzu aufbauen können. Nachrichten werden grundsätzlich zeilenweise verarbeitet, d. h., beim Senden einer Zeichenkette wird ein Zeilentrenner ergänzt und beim Empfangen wird er entfernt.

Verbindungsaufbau, Nachrichtenempfang und Verbindungsende geschehen nebenläufig. Durch Überschreiben der entsprechenden Methoden kann der Server auf diese Ereignisse reagieren.

Eine Fehlerbehandlung ist in dieser Klasse aus Gründen der Vereinfachung nicht vorgesehen.

Dokumentation der Klasse Server

Konstruktor **`Server(int pPortNr)`**

Nach dem Aufruf dieses Konstruktors bietet ein Server seinen Dienst über die angegebene Portnummer an. Clients können sich nun mit dem Server verbinden.

Auftrag **`void closeConnection(String pClientIP, int pClientPort)`**

Unter der Voraussetzung, dass eine Verbindung mit dem angegebenen Client existiert, wird diese beendet. Der Server sendet sich die Nachricht `processClosedConnection`.

Auftrag **`void processClosedConnection(String pClientIP, int pClientPort)`**

Diese Methode ohne Anweisungen wird aufgerufen, bevor der Server die Verbindung zu dem in der Parameterliste spezifizierten Client schließt. Durch das Überschreiben in Unterklassen kann auf die Schließung der Verbindung zum angegebenen Client reagiert werden.

Auftrag **`void processMessage(String pClientIP, int pClientPort, String pMessage)`**

Der Client mit der angegebenen IP und der angegebenen Portnummer hat dem Server eine Nachricht gesendet. Dieser ruft daraufhin diese Methode ohne Anweisungen auf. Durch das Überschreiben in Unterklassen kann auf diese Nachricht des angegebenen Client reagiert werden.

Name: _____

Auftrag `void processNewConnection(String pClientIP, int pClientPort)`

Der Client mit der angegebenen IP-Adresse und der angegebenen Portnummer hat eine Verbindung zum Server aufgebaut. Der Server hat daraufhin diese Methode aufgerufen, die in dieser Klasse keine Anweisungen enthält. Durch das Überschreiben in Unterklassen kann auf diesen Neuaufbau einer Verbindung von dem angegebenen Client zum Server reagiert werden.

Auftrag `void send(String pClientIP, int pClientPort, String pMessage)`

Wenn eine Verbindung zum angegebenen Client besteht, dann wird diesem Client die angegebene Nachricht – um einen Zeilentrenner erweitert – gesendet.

Auftrag `void sendToAll(String pMessage)`

Die angegebene Nachricht wird – um einen Zeilentrenner erweitert – an alle verbundenen Clients gesendet.

Auftrag `void close()`

Alle bestehenden Verbindungen werden getrennt. Der Server kann nicht mehr verwendet werden.

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 2013

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Aufgabenstellungen aus dem Bereich Client-Server-Strukturen
Syntaxvariante	Java

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2013

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

Die Kommunikation in dem Beispiel:

Client meldet sich an.
Server: +OK Herzlich willkommen bei unserem Spiel.
Server: **Frage**: Was baute Konrad Zuse?
Server: **Antwortmöglichkeiten**: a) Kaffeemaschine; b) Telefon; c) Computer; d) Auto;
Client: ANTWORT b
Server: +OK Falsche Antwort
Server: **Frage**: Was baute Konrad Zuse?
Server: **Antwortmöglichkeiten**: a) Kaffeemaschine; b) Telefon; c) Computer; d) Auto;
Client: ANTWORT c
Server: +OK Richtige Antwort
<Server sendet weitere Frage mit Antwortmöglichkeiten>
Client: ENDE
Server: +OK Vielen Dank für die Teilnahme an unserem Spiel.
Server: gespielte Fragen: 2
Server: davon richtig beantwortet: 1
Server beendet Verbindung.

Da in der Aufgabe keine Vorgaben gemacht sind, wird auch eine andere Darstellung der vom Server gesendeten Frage und der möglichen Antworten als richtig gewertet.

Die Fehlerbehandlung könnte wie folgt aussehen:

Client an Server	Server an Client
<Jede nicht im Protokoll aufgeführte Anweisung>	-ERR Fehler: Anweisung nicht im Protokoll.
`ANTWORT <Andere Antwort als a, b, c oder d>`	-ERR Fehler: Es sind nur die Antworten a, b, c oder d möglich.

Teilaufgabe b)

Das Speichern der Frage zusammen mit dem zugehörigen Spieler ist nötig, da man von dem Spieler auf die zugehörige Frage schließen können muss, wenn sich mehrere Spieler anmelden.

Teilaufgabe c)

Eine mögliche Lösung:

```
public void processMessage(String pClientIP, int pClientPort,
                                                    String pMessage){
  String anweisung = wortAn(pMessage, 1);
  Spieler derSpieler = spielerSuchen(pClientIP, pClientPort);
  if (anweisung.equals("ENDE")){
    closeConnection(pClientIP, pClientPort);
  }
  else if (anweisung.equals("ANTWORT")) {
    derSpieler.erhoeheAnzahlFragen();
    if (pMessage.charAt(pMessage.length()-1) ==
                                derSpieler.gibAktuelleAntwort()){
      send(pClientIP, pClientPort, "+OK Richtige Antwort");
      derSpieler.erhoeheRichtigeAntworten();
    }
    else {
      send(pClientIP, pClientPort, "+OK Falsche Antwort");
    }
    sendeFrage(pClientIP, pClientPort);
  }
}
```

Es sind hier viele andere Lösungen möglich. In dieser Lösung wird davon ausgegangen, dass die Reaktion auf die Protokollanweisung `ENDE`, wie in der Dokumentation der Aufgabe vorgesehen, in der Methode `processClosedConnection` erfolgt. Wenn der Kandidat dies in `processMessage` durchführt, ist das als richtig zu werten.

Teilaufgabe d)

Die Methode liefert den String `“ JXXGI WUJ LDB“`. Das Verfahren ist folgendes: Die Leerzeichen bleiben erhalten. Die Wörter zwischen den Leerzeichen werden wie folgt verschlüsselt: Der erste Buchstabe im Wort wird um 1 im Alphabet verschoben, der zweite um zwei, der dritte um drei usw. Anschließend findet noch eine Transposition statt, bei der die Wörter zwischen den Leerzeichen rückwärts ausgegeben werden.

Das Verfahren ist also polyalphabetisch. Die Sicherheit ist dennoch sehr gering, da es keinen Schlüssel gibt, sondern die Sicherheit nur von der Geheimhaltung des Verfahrens abhängt. Zusätzlich kann das Verfahren sehr schnell entschlüsselt werden, da Wortlängen erhalten bleiben. Ein Angriff mit gegebenem Klartext genügt, um das Verfahren zu verstehen.

(Es kann auch argumentiert werden, dass dieses Verfahren im Sachzusammenhang ungeeignet ist, da Passwörter meist kurz sind und nur aus einem Wort bestehen.)

Teilaufgabe e)

Der Schlüssel `GWB` verschlüsselt `ALI` zu `GHJ`.

Der Schlüssel `GGB` verschlüsselt `ABI` zu `GHJ`.

Zur Sicherheit: Die Sicherheit des Verfahrens hängt, wenn der Schlüssel genauso lang ist, wie der Text, nur noch von der Geheimhaltung des Schlüssels ab, da man mit anderen Schlüsseln jeden Klartext derselben Länge erzeugen kann, wie man an dem Beispiel sieht.

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	stellt die Kommunikation zwischen Client und Server richtig dar.	6
2	beschreibt zwei mögliche Eingabefehler und erweitert das Protokoll entsprechend.	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	begründet, dass die aktuelle Frage des jeweiligen Spielers zusammen mit dem Spieler gespeichert werden muss.	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 c)

	Anforderungen	**maximal erreichbare Punktzahl (AFB)**
	Der Prüfling	
1	implementiert die Fallunterscheidung nach den Anweisungen im Protokoll.	4
2	implementiert die Fallunterscheidung nach richtiger und falscher Antwort.	4
3	implementiert das Erhöhen der gespeicherten Werte geeignet.	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 d)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	ermittelt den Rückgabewert der Methode `verschluesseln`, wenn sie mit dem String ‘HEUTE IST ABI‘ aufgerufen wird.	6
2	erläutert die Funktionsweise des Verschlüsselungsverfahrens.	6
3	gibt an, ob es sich um ein mono- oder polyalphabethisches Verfahren handelt.	2
4	beurteilt die Sicherheit des Verfahrens.	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 (AFB)**
	Der Prüfling	
1	bestimmt den Schlüssel.	3
2	bestimmt einen Schlüssel, der aus dem Klartext ABI ebenfalls die Chiffre GHJ erzeugt.	3
3	beurteilt die Sicherheit des Vigenère-Verfahrens, wenn der Schlüssel genauso lang ist wie der Text.	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	stellt die Kommunikation …	6			
2	beschreibt zwei mögliche …	4			
	Summe Teilaufgabe a)	**10**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet, dass die …	2			
	Summe Teilaufgabe b)	**2**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Fallunterscheidung …	4			
2	implementiert die Fallunterscheidung …	4			
3	implementiert das Erhöhen …	4			
	Summe Teilaufgabe c)	**12**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt den Rückgabewert …	6			
2	erläutert die Funktionsweise …	6			
3	gibt an, ob …	2			
4	beurteilt die Sicherheit …	2			
	Summe Teilaufgabe d)	**16**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	bestimmt den Schlüssel.	3			
2	bestimmt einen Schlüssel …	3			
3	beurteilt die Sicherheit …	4			
	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 2013

Informatik, Leistungskurs

Aufgabenstellung:

Die Abiturienten der verschiedenen Schulen eines Landkreises planen ein gemeinsames soziales Netzwerk für Abiturjahrgänge, das auch nach dem Abitur den Kontakt zwischen den Mitgliedern des Netzwerkes aufrecht halten soll.

Das Netzwerk soll mit Hilfe einer Datenbank verwaltet werden.

a) Für einen ersten Entwurf wurde der folgende Anforderungskatalog an die Datenbank aufgestellt:

- Von den einzelnen Schulen werden der Name und der Ort der Schule gespeichert.
- Von den Mitgliedern werden der Name und die von ihnen besuchte Schule gespeichert. Außerdem wird das Jahr, in dem sie ihr Abitur machen, festgehalten.
- Bei der Registrierung werden für jedes Mitglied ein Anmeldename und ein Passwort festgelegt.
- Mitglieder des Netzwerks können mit anderen Mitgliedern befreundet sein. Diese Freundschaft soll gespeichert werden.
- Jeder Teilnehmer kann Nachrichten verfassen. Andere Teilnehmer können von einer Nachricht festhalten, dass sie ihnen gefällt.

Modellieren Sie die zu erstellende Datenbank im ER-Modell. Geben Sie dazu das ER-Diagramm an, aus dem die Entitätstypen mit den zugehörigen Attributen und ihre Beziehungstypen einschließlich der Kardinalitäten hervorgehen.

Übersetzen Sie das Modell in entsprechende Relationenschemata.

(10 Punkte)

Name: _____

Nach gründlicher Überarbeitung des ersten Entwurfs wurden die Möglichkeiten im Umgang mit Nachrichten erweitert: Es soll nun möglich sein, eine Nachricht mit einer Note von 1 bis 6 zu bewerten. Zur Sicherheit muss jedes Mitglied hier eine eindeutige Mailadresse angeben. Außerdem kann man zu einer Nachricht Kommentare verfassen. Im Folgenden sind die für die Verwaltung von Nachrichten wichtigen Relationen angegeben.

```
Mitglied (idMitglied, Name, Schulname, Schulort,
                                  Abijahrgang, Anmeldename, Passwort)
Nachricht (idNachricht, ↑idMitglied, NachrichtText)
bewertet (↑idMitglied, ↑idNachricht, MailAdresse, Note)
Kommentar (idKommentar, ↑idNachricht, ↑idMitglied, KommentarText)
```

b) In Anlage 1 sind Beispieltabellen zu den obigen Relationen und ein Hinweis zu dem `UNION ALL`-Befehl angegeben.

Wenden Sie die folgenden SQL-Anweisungen auf die in der Anlage 1 angegebenen Datensätze an und notieren Sie das Ergebnis der Abfragen in einer Tabelle.

Erläutern Sie die SQL-Anweisungen.

1.
```
SELECT Mitglied.Name
FROM Mitglied
WHERE Abijahrgang = 2013
```

2.
```
SELECT Mitglied.Name, Kommentar.idNachricht
FROM Mitglied, Kommentar, bewertet
WHERE Kommentar.idNachricht = bewertet.idNachricht
      AND Mitglied.idMitglied = Kommentar.idMitglied
      AND Kommentar.idMitglied = bewertet.idMitglied
```

3.
```
SELECT Mitglied.idMitglied, SUM(tmp.anzahl) AS gesamt
FROM Mitglied,
  (SELECT Mitglied.idMitglied,
          COUNT(Nachricht.idMitglied) AS anzahl
   FROM Mitglied, Nachricht
   WHERE Mitglied.idMitglied = Nachricht.idMitglied
   GROUP BY Mitglied.idMitglied
   UNION ALL
   SELECT Mitglied.idMitglied,
          COUNT(Kommentar.idMitglied) AS anzahl
   FROM Mitglied, Kommentar
   WHERE Mitglied.idMitglied = Kommentar.idMitglied
   GROUP BY Mitglied.idMitglied) AS tmp
WHERE Mitglied.idMitglied = tmp.idMitglied
GROUP BY Mitglied.idMitglied
ORDER BY gesamt DESC
```

(12 Punkte)

Name: _____

c) *Entwickeln Sie die zugehörigen SQL-Anweisungen, um folgende Abfragen durchzuführen:*

1. Es soll eine Tabelle aller Nachrichtentexte von Lea Braun, die die Mitgliedernummer 3 hat, erstellt werden.
2. Es soll eine Tabelle aller Mitgliedsnummern der Mitglieder erstellt werden, die noch keine Nachrichten verfasst haben.
3. Als Gesamtbewertung einer Nachricht wird der Durchschnitt aller Bewertungen genommen. Es soll eine Tabelle erstellt werden, in der alle Nachrichten zusammen mit ihrer Gesamtbewertung aufgelistet sind, die aufsteigend nach der Bewertung sortiert sein soll. Dabei sollen nur die Bewertungen berücksichtigt werden, die einer der gängigen sechs Schulnoten entsprechen.

(16 Punkte)

d) Durch die Normalisierung von Datenbanken werden Redundanzen und Anomalien vermieden.

Geben Sie die Bedingungen für die 1., 2. und 3. Normalform an.

Erläutern Sie für die vor Aufgabenteil b) gegebenen Relationenschemata, welche der Bedingungen für die einzelnen Normalformen verletzt werden. Berücksichtigen Sie ggf. die in Anlage 2 aufgeführten Abhängigkeiten.

Überführen Sie die gegebenen Relationenschemata schrittweise in die 3. Normalform und begründen Sie Ihr Vorgehen.

(12 Punkte)

Zugelassene Hilfsmittel:

- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

Name: _____

Anlage 1: Beispieltabellen zu Aufgabenteil b)

Mitglied

idMitglied	Name	Schulname	Schulort	Abijahrgang	Anmeldename	Passwort
1	Dirk Meier	Konrad-Zuse-Gymnasium	Aburg	2013	dirk	bca!314
2	Thinh Phu	Carl-Petri-Gymnasium	Bedorf	2014	thinh	cba?159
3	Lea Braun	Alan-Turing-Gesamtschule	Cefurt	2013	lea	bac$265
4	Gamze Ergin	Alan-Turing-Gesamtschule	Cefurt	2013	gamze	cab&358
5	Arndt Schmidt	von-Neumann-Gesamtschule	Aburg	2014	andy	acb(979

Nachricht

idNachricht	↑idMitglied	NachrichtText
1	3	Bald habe ich mein Abi!
2	4	Schule ist doof.
3	3	Ihr seid alle toll!
4	1	Wer von euch kommt zur Party?

bewertet

↑idMitglied	↑idNachricht	MailAdresse	Note
1	2	dmeier@mail.de	5
1	1	dmeier@mail.de	3
3	1	lea@mail.de	1
4	1	ge13@mail.de	7

Kommentar

idKommentar	↑idNachricht	↑idMitglied	KommentarText
1	2	3	Bitte hier nicht lästern!
2	2	1	Das finde ich gar nicht.

Hinweis zu `UNION ALL`: Die Ergebnismenge einer `UNION`-Operation enthält alle Zeilen beider Abfragen. Bei `UNION ALL` werden vollständig identische Zeilen nicht entfernt.

Name: _____

Anlage 2: Abhängigkeiten zu Aufgabenteil d)

Die Abhängigkeiten geben an, ob Attribute voneinander abhängig sind. So wird durch die Beziehung `Schulname → Schulort` festgelegt, dass der Ort einer Schule vom Namen der Schule abhängt.

Im Aufgabenteil d) werden die folgenden Abhängigkeiten angenommen:

```
Schulname → Schulort

idMitglied → Nachname, Vorname, Mailadresse, Schulname,
                       Schulort, Abijahrgang, Anmeldename, Passwort

idNachricht → NachrichtText

idKommentar → KommentarText
```

Unterlagen für die Lehrkraft

Abiturprüfung 2013

Informatik, Leistungskurs

1. Aufgabenart

Aufgabenart	Aufgabenstellungen aus dem Bereich Relationale Datenbanken
Syntaxvariante	–

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

- entfällt

4. Bezüge zu den Vorgaben 2013

1. *Inhaltliche Schwerpunkte*
   Relationale Datenbanken
   - Normalisierung: Überführung einer Datenbank in die 1. bis 3. Normalform
   - Relationenalgebra (Selektion, Projektion, Join)
   - SQL-Abfragen über eine und mehrere verknüpfte Tabellen
   - Datenschutzaspekte

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

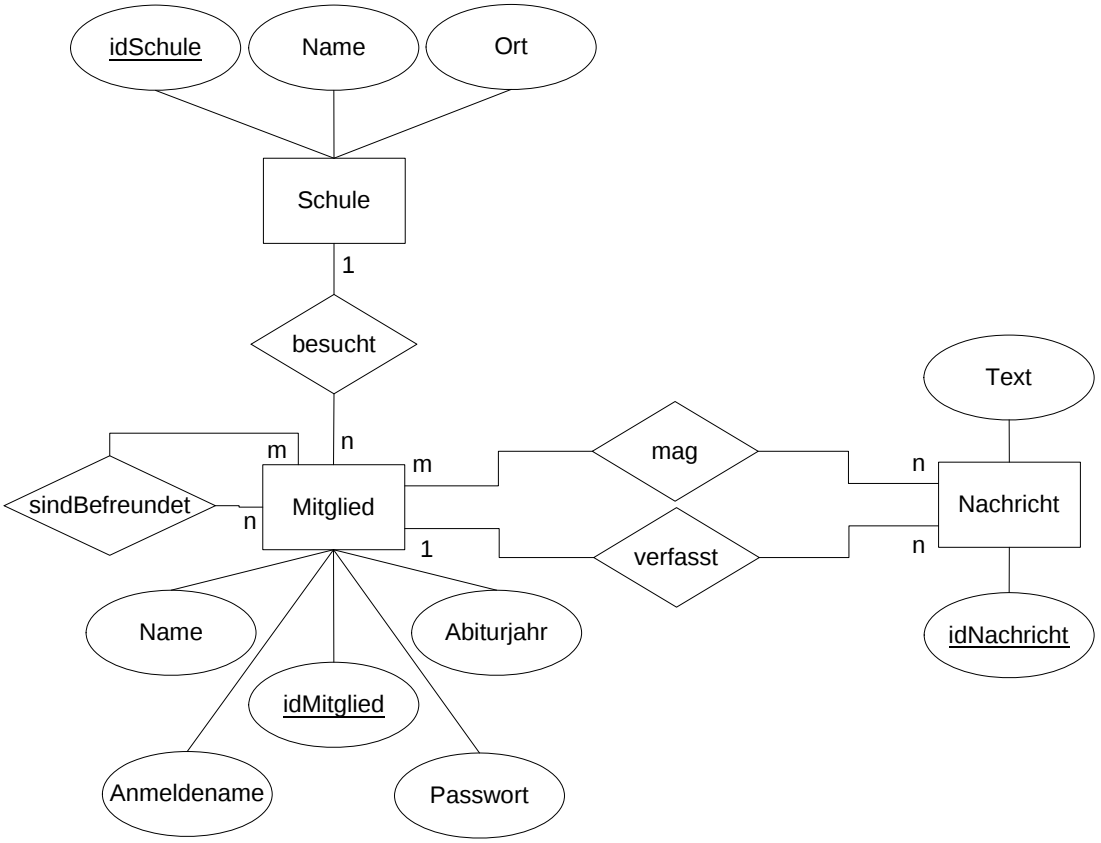
- Wörterbuch zur deutschen Rechtschreibung
- Taschenrechner

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

6. Vorgaben für die Bewertung der Schülerleistungen

6.1 Modelllösung

Teilaufgabe a)




```
Mitglied (idMitglied, Name, Abiturjahr,
                                  Anmeldename, Passwort, ↑idSchule)
sindBefreundet (↑idMitglied, ↑idMitglied)
Schule (idSchule, Name, Ort)
Nachricht (idNachricht, ↑idMitglied, Text)
mag (↑idMitglied, ↑idNachricht)
```

Teilaufgabe b)

1.

Name
Dirk Meier
Lea Braun
Gamze Ergin

Es werden alle Schüler des Abiturjahrgangs 2013 selektiert und anschließend auf den Namen projiziert.

Es wird eine Liste aller Mitglieder, die im Jahr 2013 ihr Abitur ablegen, erstellt.

2.

Mitglied.Name	Kommentar.idNachricht
Dirk Meier	2

Zunächst werden in den Tabellen `Mitglied`, `Kommentar` und `bewertet` alle Datensätze miteinander verbunden und selektiert, bei denen der Kommentar und die Bewertung sich auf die gleiche Nachricht beziehen und bei denen der Kommentar und die Bewertung von dem gleichen Mitglied stammen. Anschließend wird auf den Namen des Mitgliedes und die Nummer der Nachricht projiziert.

Es wird eine Liste aller Mitglieder, die eine Nachricht sowohl bewertet als auch kommentiert haben, zusammen mit der Nummer der Nachricht erstellt.

3.

Mitglied.idMitglied	gesamt
3	3
1	2
4	1

Die Abfrage enthält zwei innere Abfragen, die durch `UNION` verbunden sind. Beide Abfragen liefern als Ergebnis Mitgliedsnummern zusammen mit einer Anzahl. In der ersten Abfrage ist es die Anzahl der Nachrichten, die das Mitglied verfasst hat, in der zweiten die Anzahl der Kommentare.

Da die Anzahl der Nachrichten mit der Anzahl der Kommentare übereinstimmen könnte, muss zum Befehl `UNION` noch der Befehl `ALL` hinzugefügt werden, damit die dann gleichen Zeilen auch in der Ergebnistabelle doppelt auftauchen.

In der äußeren Selektion werden die Anzahlen, gruppiert nach der Mitgliedsnummer, addiert.

Es wird eine Tabelle erstellt, in der alle Mitglieder zusammen mit der Anzahl ihrer Beiträge – Nachrichten oder Kommentare – aufgeführt sind, die nach der Anzahl der Beiträge absteigend sortiert ist.

Teilaufgabe c)

1.

```
SELECT Nachricht.NachrichtText
FROM Nachricht
WHERE Nachricht.idMitglied = 3
```

2.

```
SELECT mitglied.idMitglied
FROM mitglied
WHERE mitglied.idMitglied NOT IN (SELECT nachricht.idMitglied
                                  FROM nachricht)
```

3.

```
SELECT bewertet.idNachricht,
 SUM(bewertet.Note) / COUNT(bewertet.idMitglied) AS Gesamtbewertung
FROM bewertet
WHERE bewertet.Note IN (1, 2, 3, 4, 5, 6)
GROUP BY bewertet.idNachricht
ORDER BY Gesamtbewertung
```

Teilaufgabe d)

1. Normalform

Eine Tabelle liegt in der ersten Normalform (1NF) vor, wenn jeder Attributwert eine atomare, nicht weiter zerlegbare Dateneinheit ist. Eine Tabelle ist nicht in der 1NF, wenn Attribute mehrfach oder komplex in einer Spalte auftreten.

Das Attribut `Name` ist nicht atomar, denn es lässt sich in die Dateneinheiten `Nachname` und `Vorname` zerlegen.

Um das Relationenschema in 1NF zu überführen, wird das Attribut `Name` durch die oben genannten Dateneinheiten ersetzt.

Die Überführung in die erste Normalform lautet deshalb wie folgt:

```
Mitglied (idMitglied, Nachname, Vorname, Schulname, Schulort,
                                Abijahrgang, Anmeldename, Passwort)
Nachricht (idNachricht, ↑idMitglied, NachrichtText)
bewertet (↑idMitglied, ↑idNachricht, MailAdresse, Note)
Kommentar (idKommentar, ↑idNachricht, ↑idMitglied, KommentarText)
```

2. Normalform

Eine Tabelle liegt in der zweiten Normalform (2NF) vor, wenn sie in der 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig vom Primärschlüssel ist. Eine Tabelle ist nicht in der 2NF, wenn Attribute von einem Teil des Schlüssels eindeutig identifiziert werden.

In dem Relationenschema bewertet hängt das Attribut `MailAdresse` nur vom Teilschlüssel `idMitglied` ab.

Um das Relationenschema in 2NF zu überführen, wird das Attribut `Mailadresse` in das Relationenschema `Mitglied` eingefügt.

Die Überführung in die zweite Normalform lautet wie folgt:

```
Mitglied (idMitglied, Nachname, Vorname, MailAdresse,
          Schulname, Schulort, Abijahrgang, Anmeldename, Passwort)
Nachricht (idNachricht, ↑idMitglied, NachrichtText)
bewertet (↑idMitglied, ↑idNachricht, Note)
Kommentar (idKommentar, ↑idNachricht, ↑idMitglied, KommentarText)
```

3. Normalform

Eine Tabelle liegt in der dritten Normalform (3NF) vor, wenn sie sich in der 2NF befindet und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Primärschlüssel ist. Eine Tabelle ist nicht in der 3NF, wenn Attribute von anderen Nicht-Schlüsselattributen identifiziert werden.

In der Relation `Mitglied` hängt (nach Anlage 2) das Attribut `Schulort` vom Nichtschlüsselattribut `Schulname` ab.

Um die Relationenschemata in 3NF zu überführen, wird das neue Relationenschema `Schule` eingeführt. Die durch die neuen Relationenschemata aufgefangenen Attribute werden im Relationenschema `Mitglied` durch den entsprechenden Fremdschlüssel ersetzt.

Die Überführung in die dritte Normalform lautet wie folgt:

```
Schule (idSchule, Schulname, Schulort)
Mitglied (idMitglied, Nachname, Vorname, MailAdresse,
                      idSchule, Abijahrgang, Anmeldename, Passwort)
Nachricht (idNachricht, ↑idMitglied, NachrichtText)
bewertet (↑idMitglied, ↑idNachricht, Note)
Kommentar (idKommentar, ↑idNachricht, ↑idMitglied, KommentarText)
```

6.2 Teilleistungen – Kriterien

Teilaufgabe a)

	Anforderungen	**maximal erreichbare Punktzahl**
	Der Prüfling	
1	stellt die Entitäten und Beziehungen in einem ER-Diagramm dar.	4
2	bestimmt die Kardinalitäten.	2
3	übersetzt das Modell in entsprechende Relationenschemata.	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	wendet die erste SQL-Anweisung auf die Daten an, notiert das Ergebnis und erläutert die Anweisung.	2
2	wendet die zweite SQL-Anweisung auf die Daten an, notiert das Ergebnis und erläutert die Anweisung.	4
3	wendet die dritte SQL-Anweisung auf die Daten an, notiert das Ergebnis und erläutert die Anweisung..	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	entwickelt eine SQL-Anweisung zur ersten Abfrage.	4
2	entwickelt eine SQL-Anweisung zur zweiten Abfrage.	6
3	entwickelt eine SQL-Anweisung zur dritten Abfrage.	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	gibt die Bedingungen für die erste, zweite und dritte Normalform an.	6
2	erläutert, dass die Bedingungen für die Normalformen verletzt sind.	3
3	überführt das Relationenschema in die 3. Normalform und begründet das 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.		

7. Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	stellt die Entitäten …	4			
2	bestimmt die Kardinalitäten.	2			
3	übersetzt das Modell …	4			
	Summe Teilaufgabe a)	**10**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	wendet die erste …	2			
2	wendet die zweite …	4			
3	wendet die dritte …	6			
	Summe Teilaufgabe b)	**12**			

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine SQL-Anweisung …	4			
2	entwickelt eine SQL-Anweisung …	6			
3	entwickelt eine SQL-Anweisung …	6			
	Summe Teilaufgabe c)	**16**			

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

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt die Bedingungen …	6			
2	erläutert, dass die …	3			
3	überführt das Relationenschema …	3			
	Summe Teilaufgabe d)	**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