Bitte beachten:

Die Aufgaben wurden nur an den Schulen
164458_167605_169110_194232_164872_
164471
verwendet.

Sie dürfen nicht veröffentlicht werden!

Name: _____

Abiturprüfung 2017

Informatik, Leistungskurs

Aufgabenstellung:

Das Verwalten von Terminen ist eine Aufgabe, die oft mit Software gelöst wird. Im Folgenden soll eine vereinfachte Variante betrachtet werden, um grundsätzliche Elemente einer solchen Software zu untersuchen.

Die mit einer geeigneten Software gespeicherten Termine können folgende Daten verwalten: Zu jedem Termin gehört zwingend ein Startzeitpunkt, eine Dauer und eine Bezeichnung. Außerdem hat jeder Termin immer eine eindeutige Identifizierungsnummer, abgekürzt „UID“. Die UID eines Termins kann nicht geändert werden. Weitere Daten, die angegeben werden können, sind der Ort, an dem der Termin stattfindet und eine Kategorie, die ebenfalls in Textform angegeben wird.

Die Klasse `TerminVerwaltung` speichert die verwalteten Termine, wobei keine Reihenfolge vorgegeben ist. Dabei soll ein Objekt dieser Klasse garantieren, dass jede gespeicherte UID tatsächlich eindeutig ist. Es können Termine hinzugefügt, gelöscht oder gesucht werden. Zur Verwaltung der Termine wird eine Liste (Datentyp `List`) verwendet.

Terminliste

```
10.05.2017, 09:00   Informatik GK
Dauer: 180 Minuten  UID: abitur:nrw:5B12-0
Kategorie: Prüfung  Raum: A530

03.05.2017, 09:00   Mathematik GK
Dauer: 180 Minuten  UID: abitur:nrw:EEF1-2
Kategorie: Prüfung  Raum: C122

10.05.2017, 09:01   Informatik LK
Dauer: 255 Minuten  UID: abitur:nrw:51BC-1
Kategorie: Prüfung  Raum: C105

03.05.2017, 09:00   Mathematik LK
Dauer: 180 Minuten  UID: abitur:nrw:AA01-9
Kategorie: Prüfung  Raum: A530

09.05.2017, 09:00   Russisch GK
Dauer: 180 Minuten  UID: abitur:nrw:5B12-13
Kategorie: Prüfung  Raum: C122
```

Abbildung 1: Liste mit Terminen

Name: _____

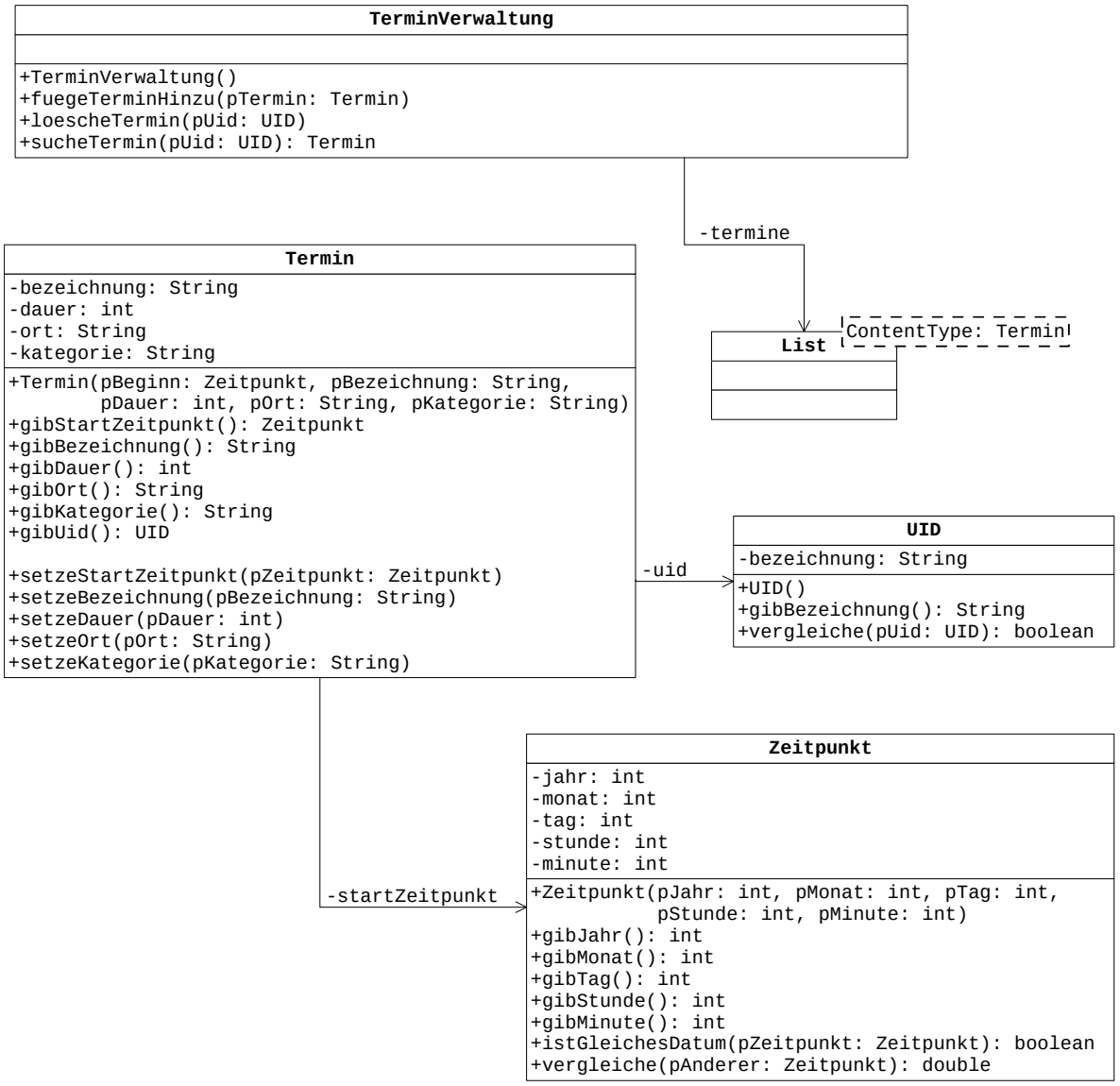


Abbildung 2: Implementationsdiagramm

Name: _____

a) Da die Liste der Termine unsortiert ist, können neue Termine einfach an die Liste angehängt werden. Um einen neuen Termin hinzufügen zu können, ist es allerdings notwendig zu überprüfen, ob die UID des neuen Termins noch nicht in der Terminliste existiert. Existiert die UID bereits, wird der Termin nicht hinzugefügt.

Die Methode `fuegeTerminHinzu` der Klasse `TerminVerwaltung` soll dies leisten. Sie hat den folgenden Methodenkopf

```
public void fuegeTerminHinzu(Termin pTermin)
```

Entwickeln Sie einen Algorithmus für das Hinzufügen eines Termins unter den genannten Bedingungen als Struktogramm.

Implementieren Sie die Methode `fuegeTerminHinzu`.

(14 Punkte)

b) Die folgende Methode `wasMacheIch` der Klasse `TerminVerwaltung` ist eine nicht dokumentierte Methode im Quellcode:

```
 1 public List<Termin> wasMacheIch(Zeitpunkt pDatum) {
 2   List<Termin> ergebnis = new List<Termin>();
 3   termine.toFirst();
 4   while (termine.hasAccess()) {
 5     Zeitpunkt neuerZeitpunkt
           = termine.getContent().gibStartZeitpunkt();
 6     if (neuerZeitpunkt.istGleichesDatum(pDatum)) {
 7       ergebnis.toFirst();
 8       while (ergebnis.hasAccess()
                 && ergebnis.getContent().gibStartZeitpunkt()
                     .vergleiche(neuerZeitpunkt) < 0.0) {
 9         ergebnis.next();
10       }
11       if (ergebnis.hasAccess()) {
12         ergebnis.insert(termine.getContent());
13       } else {
14         ergebnis.append(termine.getContent());
15       }
16     }
17     termine.next();
18   }
19   return ergebnis;
20 }
```

Name: _____

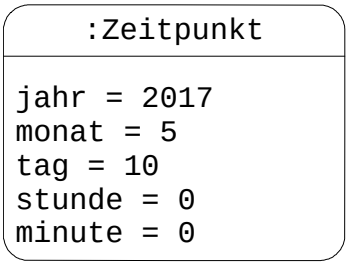


Abbildung 3: Darstellung des als Parameter übergebenen Zeitpunkts

Analysieren Sie die Methode, indem Sie sie auf die Beispieldaten aus Abbildung 1 und Abbildung 3 und anwenden. Dokumentieren Sie dazu immer nach Ausführen von Zeile 3 und Zeile 17 den Zustand der Listen `termine` *und* `ergebnis`*.*

Erläutern Sie im Sachzusammenhang, was die Methode leistet.

(10 Punkte)

Für den Benutzer ist es wichtig, eine Auswahl von Terminen anhand von Kriterien zu ermöglichen. Beispielsweise sollen folgende Termine gesucht werden: alle Termine der Kategorie `"Besprechung"`, alle Termine in Raum `"A530"` oder Termine der Kategorie `"Prüfung"`, die die Bezeichnung `"Klausur"` haben.

Um Termine so zu filtern, wird folgendes Konzept genutzt: Die verwalteten Termine werden mit einem Musterobjekt verglichen. Das Musterobjekt hat die gleichen Attribute wie ein Termin, besitzt aber keine UID. Wenn die Attribute eines gespeicherten Termins gleich denen des Musterobjekts sind, dann ist der Termin Teil des Ergebnisses.

Referenzieren Attribute des Musterobjekts `null` oder besitzen sie einen vereinbarten Wert – zum Beispiel `-1` beim Attribut `dauer` – so werden diese Attribute ignoriert. Die Gleichheit der Attribute wird also nur bei den gesetzten Attributen geprüft.

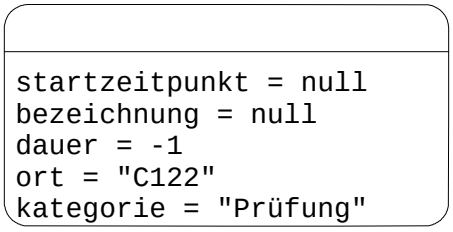


Abbildung 4: Beispiel für ein Musterobjekt zum Filtern nach Ort und Kategorie

Name: _____

c) *Wenden Sie dieses Filterkonzept auf die in Abbildung 1 und Abbildung 4 dargestellten Daten an und ermitteln Sie so eine Ergebnisliste.*

Modifizieren Sie das Implementationsdiagramm aus Abbildung 2, um dieses Filterkonzept implementieren zu können.

Erläutern Sie die Veränderungen an bestehenden Methoden und dokumentieren Sie die ergänzten Methoden.

Hinweis:
Nutzen Sie für die Modifikation des Implementationsdiagramms die Vorlage in der Anlage.

(13 Punkte)

d) Oft möchte man bei einer Terminverwaltung anders filtern, da Termine für einen bestimmten Zeitraum gesucht sind. Beispielsweise können alle Termine gesucht sein, die in der nächsten Woche beginnen, oder man möchte alle Vormittagstermine kennen, die zwischen 9:00 Uhr und 12:00 Uhr beginnen.

Begründen Sie, warum es mit dem gegebenen Filtersystem nicht möglich ist, eine solche Auswahl von Terminen zu filtern.

Erläutern Sie, wie das Filtermodell geändert werden kann, um derartige Filterungen zu ermöglichen.

(7 Punkte)

e) Ein Schüler behauptet, dass in der Implementation ein Fehler existiert: Die Software könne nicht garantieren, dass jede UID jederzeit eindeutig sei: die UID könne geändert werden. Dabei verweist er auf die Implementation der Methode `gibUid` der Klasse `Termin`: Man könne dadurch auf das private Objekt `uid` Zugriff erlangen.

```
1  public UID gibUid() {
2    return uid;
3  }
```

Beurteilen Sie die Behauptung des Schülers.

(6 Punkte)

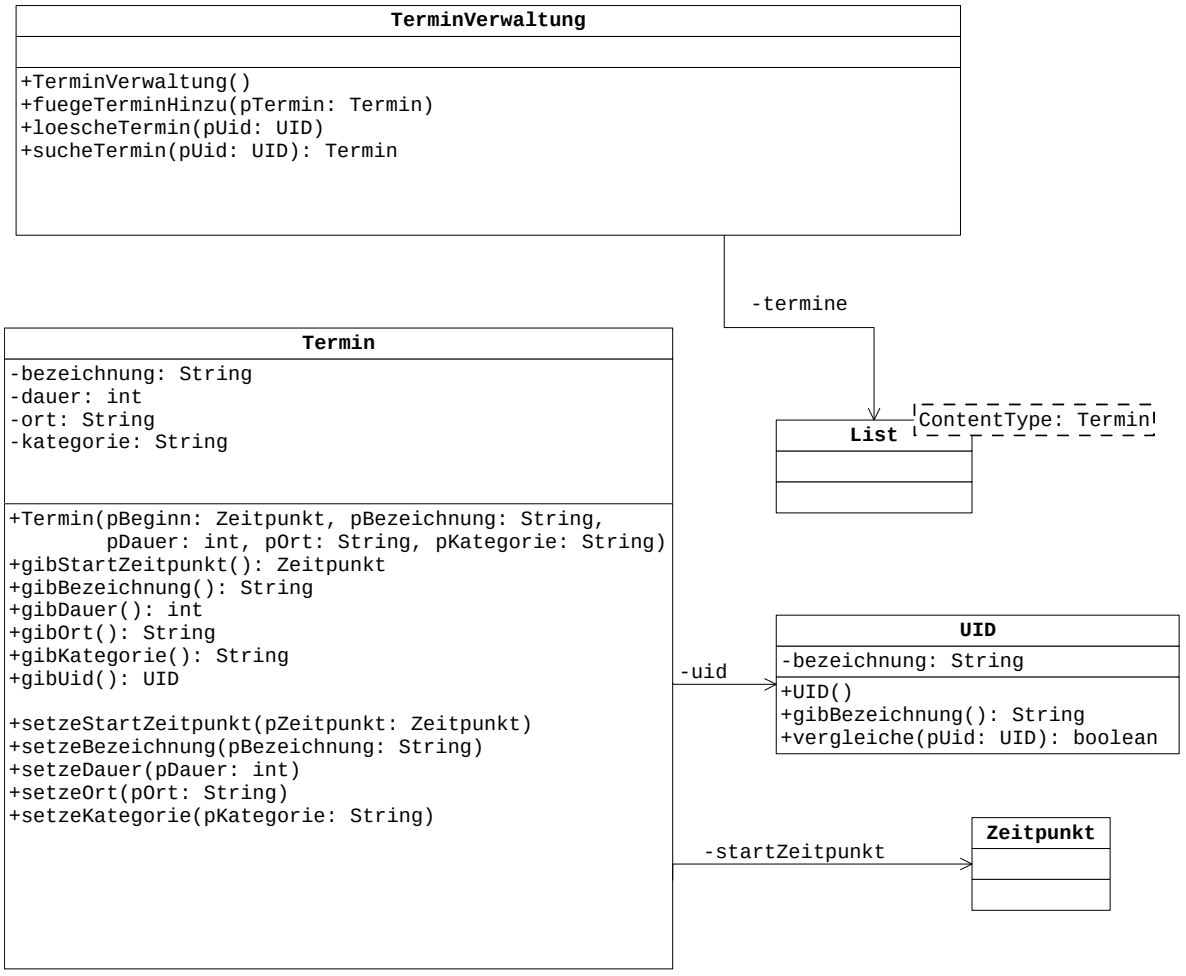
Zugelassene Hilfsmittel:

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

Name: _____

Anlage zu Aufgabenteil c)

Vorlage für die Modifikation des Implementationsdiagramms.



Name: _____

Ausschnitt aus der Dokumentation der Klasse `TerminVerwaltung`

Diese Klasse verwaltet beliebig viele Termine. Die Termine werden anhand ihrer UID unterschieden. Die UID jedes verwalteten Termins muss eindeutig sein.

Konstruktor **`TerminVerwaltung()`**
Nach dem Aufruf des Konstruktors ist ein Objekt der Klasse erzeugt.

Auftrag **`void fuegeTerminHinzu(Termin pTermin)`**
Die Methode fügt einen Termin in die Terminverwaltung ein, sofern die UID des Termins noch nicht existiert.

Auftrag **`void loescheTermin(UID pUid)`**
Die Methode löscht den Termin mit der übergebenen UID aus der Terminverwaltung, sofern der Termin existiert. Referenziert `pUid null`, wird keine Veränderung vorgenommen.

Anfrage **`Termin sucheTermin(UID pUid)`**
Die Methode gibt das Objekt mit der als Parameter übergebenen UID zurück, sofern ein Termin mit dieser UID in der Terminverwaltung existiert. Andernfalls wird `null` zurückgegeben. Referenziert `pUid null`, wird `null` zurückgegeben.

Ausschnitt aus der Dokumentation der Klasse `UID`

Objekte dieser Klasse repräsentieren eine eindeutige Identifikationsnummer.

Konstruktor **`UID()`**
Nach dem Aufruf des Konstruktors existiert eine eindeutige UID.

Anfrage **`String gibBezeichnung()`**
Die Methode gibt die Bezeichnung der UID zurück.

Anfrage **`boolean vergleiche(UID pUid)`**
Die Methode gibt den Wert `true` zurück, wenn die übergebene UID mit dem Objekt identisch ist, andernfalls wird `false` zurückgegeben. Referenziert `pUid null`, wird `false` zurückgegeben.

Name: _____

Ausschnitt aus der Dokumentation der Klasse `Termin`

Diese Klasse repräsentiert einen Termin.

Konstruktor **`Termin(Zeitpunkt pBeginn, String pBezeichnung, int pDauer, String pOrt, String pKategorie)`**
Nach dem Aufruf des Konstruktors sind die übergebenen Werte der Parameter und die übergebenen Referenzen der Parameter gespeichert. Das Objekt hat eine UID.

Anfrage **`Zeitpunkt gibStartZeitpunkt()`**
Die Methode gibt eine Referenz auf den Startzeitpunkt des Termins zurück.

Anfrage **`UID gibUid()`**
Die Methode gibt die UID des Termins zurück.

Anfrage **`String gibBezeichnung()`**
Die Methode gibt die Bezeichnung des Termins zurück.

Anfrage **`int gibDauer()`**
Die Methode gibt die Dauer des Termins zurück.

Anfrage **`String gibOrt()`**
Die Methode gibt den Ort des Termins zurück.

Anfrage **`String gibKategorie()`**
Die Methode gibt die Kategorie des Termins zurück.

Auftrag **`void setzeStartZeitpunkt(Zeitpunkt pStartZeitpunkt)`**
Referenziert `pStartZeitpunkt` nicht `null`, wird die übergebene Referenz als Startzeitpunkt gespeichert.

Auftrag **`void setzeBezeichnung(String pBezeichnung)`**
Referenziert `pBezeichnung` nicht `null`, wird die übergebene Referenz als Bezeichnung gespeichert.

Auftrag **`void setzeDauer(int pDauer)`**
Sofern der übergebene Wert nicht negativ ist, wird er als Dauer gespeichert.

Auftrag **`void setzeOrt(String pOrt)`**
Referenziert `pOrt` nicht `null`, wird die übergebene Referenz als Ort gespeichert.

Auftrag **`void setzeKategorie(String pKategorie)`**
Referenziert `pKategorie` nicht `null`, wird die übergebene Referenz als Kategorie gespeichert.

Name: _____

Ausschnitt aus der Dokumentation der Klasse `Zeitpunkt`

Objekte dieser Klasse repräsentieren einen Zeitpunkt, der einerseits durch das Kalenderdatum, andererseits durch eine Uhrzeit festgelegt ist, wobei die Sekunden vernachlässigt werden.

Konstruktor **`Zeitpunkt(int pJahr, int pMonat, int pTag, int pStunde, int pMinute)`**

Nach dem Aufruf des Konstruktors sind die übergebenen Werte der Parameter gespeichert.

Anfrage **`int gibJahr()`**

Die Methode gibt das Jahr des Zeitpunkts zurück.

Anfrage **`int gibMonat()`**

Die Methode gibt den Monat des Zeitpunkts zurück.

Anfrage **`int gibTag()`**

Die Methode gibt den Tag des Zeitpunkts zurück.

Anfrage **`int gibStunde()`**

Die Methode gibt die Stunde des Zeitpunkts zurück.

Anfrage **`int gibMinute()`**

Die Methode gibt die Minute des Zeitpunkts zurück.

Anfrage **`boolean istGleichesDatum(Zeitpunkt pZeitpunkt)`**

Die Methode liefert als Ergebnis `true`, wenn bei beiden Zeitpunkte am gleichen Tag liegen: Das Jahr, der Monat und der Tag sind identisch. Andernfalls liefert die Methode `false`.

Anfrage **`double vergleiche(Zeitpunkt pAnderer)`**

Die Methode liefert als Ergebnis einen negativen Wert, wenn das Objekt vor dem als Parameter übergebenen Zeitpunkt liegt. Sie liefert den Wert `0`, wenn beide Zeitpunkte identisch sind, also das Datum und die Uhrzeit übereinstimmen. Andernfalls liefert die Methode einen positiven Wert.

Name: _____

Dokumentationen der verwendeten Klassen

Die generische Klasse List<ContentType>

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

Dokumentation der Klasse `List<ContentType>`

Konstruktor **`List()`**
Eine leere Liste wird erzeugt. Objekte, die in dieser Liste verwaltet werden, müssen vom Typ `ContentType` sein.

Anfrage **`boolean isEmpty()`**
Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

Anfrage **`boolean hasAccess()`**
Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

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

Auftrag **`void toFirst()`**
Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag **`void toLast()`**
Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Name: _____

Anfrage **`ContentType getContent()`**

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

Auftrag **`void setContent(ContentType pContent)`**

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

Auftrag **`void append(ContentType pContent)`**

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

Auftrag **`void insert(ContentType pContent)`**

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

Auftrag **`void concat(List<ContentType> pList)`**

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

Auftrag **`void remove()`**

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

Unterlagen für die Lehrkraft

Abiturprüfung 2017

Informatik, Leistungskurs

1. Aufgabenart

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

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2017

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Objekte und Klassen
     - Entwurfsdiagramme und Implementationsdiagramme
     - Lineare Strukturen
       Lineare Liste (Klasse `List`*)*

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

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

2. *Medien/Materialien*
   - entfällt

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

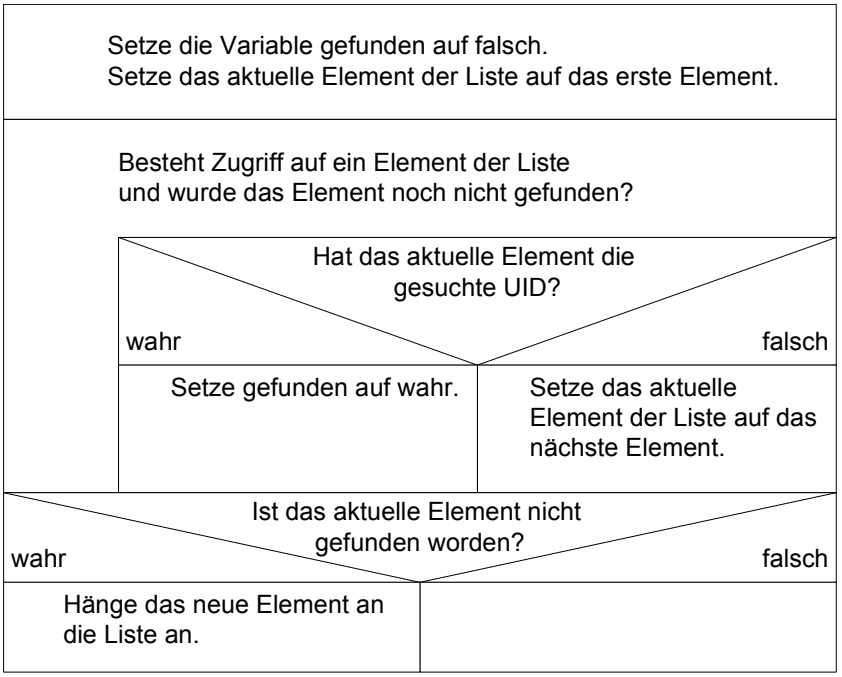
5. Zugelassene Hilfsmittel

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

6. Modelllösungen

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

Teilaufgabe a)



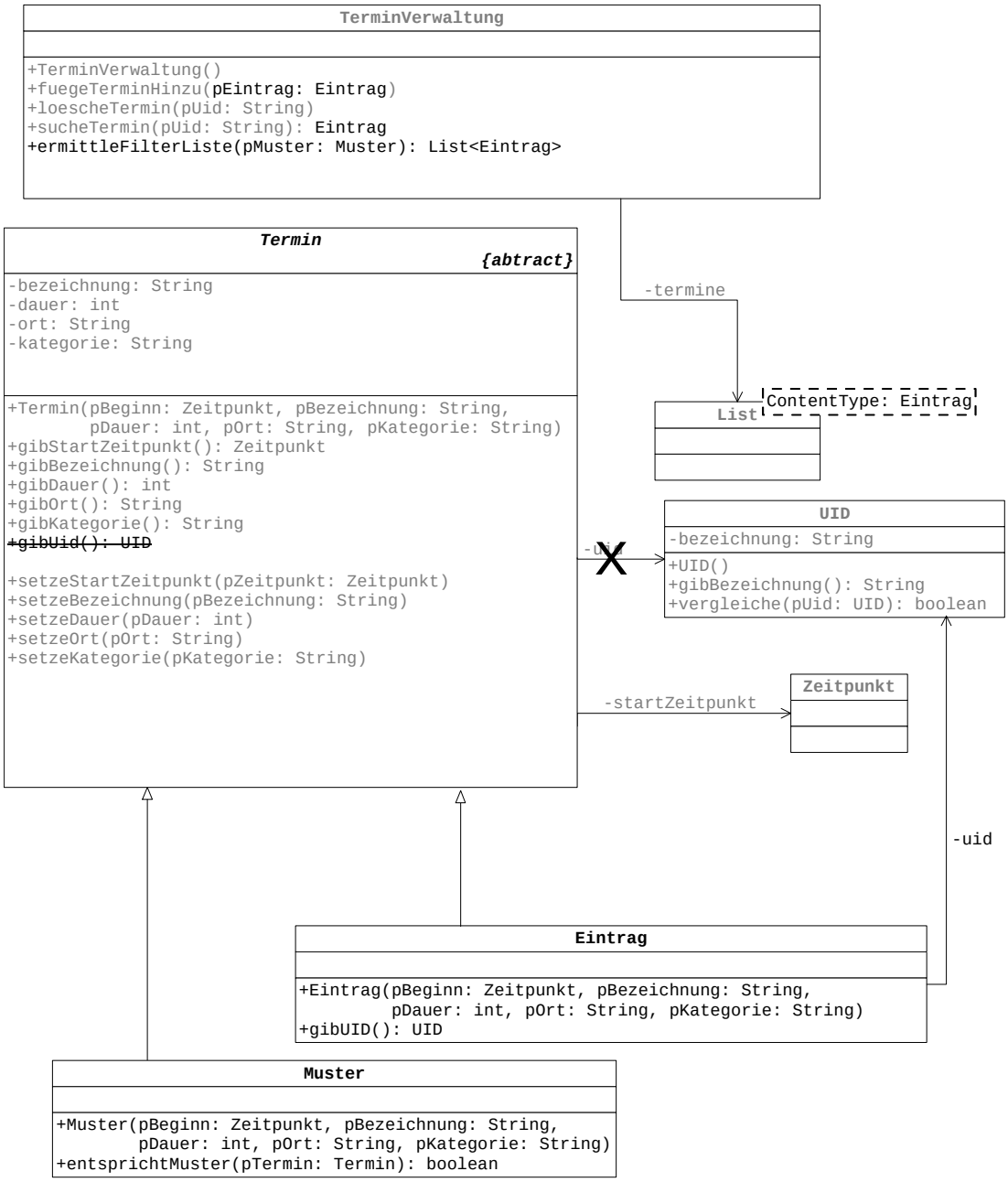

```
public void fuegeTerminHinzu(Termin pTermin) {
  boolean gefunden = false;
  termine.toFirst();
  while (termine.hasAccess() && !gefunden) {
    if (termine.getContent().gibUid().vergleiche(pTermin.gibUid())) {
      gefunden = true;
    } else {
      termine.next();
    }
  }
  if (!gefunden) {
    termine.append(pTermin);
  }
}
```

Teilaufgabe b)

Die Methode filtert die Liste der Termine dahingehend, dass alle Termine zurückgegeben werden, die an einem bestimmten Datum liegen. Dieses Filterdatum wird durch das als Parameter übergebene Objekt festgelegt. Die Uhrzeit ist für die Auswahl nicht relevant. Die Ergebnisliste ist aufsteigend nach Uhrzeit geordnet.

Teilaufgabe c)

Relevant sind für das Filtern sind nur die beiden Attribute `ort` und `kategorie`, da bei den anderen Attributen `null` referenziert wird bzw. das Attribut `dauer` den Wert `-1` hat. Für jedes Objekt in der Terminliste müssen also die Attribute `ort` und `kategorie` auf Gleichheit mit den Beispieldaten geprüft werden.
Dies trifft nur auf die Objekte mit den UIDs `abitur:nrw:EEF1-2` und `abitur:nrw:5B12-`13 zu, die dann beide in einer Ergebnisliste zu finden wären.



Erläuterung der Änderungen:

Die Methoden der Klasse `TerminVerwaltung` werden dahingehend geändert, dass statt der Klasse `Termin` die Unterklasse `Eintrag` verwendet wird. Dies ergibt sich daraus, dass die Klasse `Termin` abstrakt sein soll und die Unterklasse `Eintrag` die Verwaltung des Attributs `uid` übernimmt.

Bei der Klasse `Termin` ändert sich bezüglich des Konstruktors, dass keine UID erzeugt wird.

Dokumentation der ergänzten Methoden:

Die Klasse `Terminverwaltung`

Anfrage **`List<Eintrag> ermittleFilterListe(Muster pMuster)`**

Die Methode ermittelt eine Liste von Terminen nach dem beschriebenen Filterverfahren: Ein Termin aus der Terminliste ist in der Ergebnisliste enthalten, wenn alle Attribute mit den Attributen des übergebenen Objekts übereinstimmen, sofern die Attribute des übergebenen Objekts nicht `null` referenzieren bzw. den Wert `-1` haben. Die Objekte in der Ergebnisliste befinden sich in der gleichen Reihenfolge wie in der Terminliste. Wird ein Objekt der Klasse `Termin` übergeben, dessen UID nicht `null` referenziert, wird eine leere Liste zurückgegeben.

Die Klasse `Eintrag`

Konstruktor
```
Eintrag(Zeitpunkt pBeginn, String pBezeichnung,
        int pDauer, String pOrt, String pKategorie)
```

Nach dem Aufruf des Konstruktors sind die übergebenen Werte der Parameter und die übergebenen Referenzen der Parameter gespeichert. Das Objekt hat eine eindeutige UID. Referenzieren die Parameter `pBeginn`, `pBezeichnung`, `pOrt` oder `pKategorie` `null`, so wird stattdessen ein leerer String gespeichert. Ist der Wert von `pDauer` kleiner als `0`, so wird `0` gespeichert.

Anfrage **`UID gibUID()`**

Die Methode gibt die UID des Eintrags zurück.

Die Klasse `Muster`

Konstruktor
```
Muster(Zeitpunkt pBeginn, String pBezeichnung,
       int pDauer, String pOrt, String pKategorie)
```

Nach dem Aufruf des Konstruktors sind die übergebenen Werte der Parameter und die übergebenen Referenzen der Parameter gespeichert.

Anfrage **`boolean entsprichtMuster(Termin pTermin)`**

Die Methode prüft, ob das als Parameter übergebene Objekt der Klasse `Eintrag` dem vorgegebenen Muster entspricht: Abgesehen von der UID des übergebenen Eintrags werden die Ausprägungen aller geerbten Attribute auf Gleichheit mit denen im Muster geprüft: Sind diese identisch, wird `true` zurückgegeben, andernfalls `false`.

Teilaufgabe d)

Eine solche Auswahl von Terminen ist mit dem beschriebenen Filtersystem nicht möglich, da jedes Terminobjekt für ein Attribut nur ein Datum speichern kann, nicht mehrere. Damit kann bei einem Vergleich zwischen dem Filterobjekt und den gespeicherten Terminen nur auf Gleichheit oder Ungleichheit geprüft werden. Es muss also eine Möglichkeit geboten werden, auf Gleichheit zu mehreren Daten bzw. Zugehörigkeit zu einem Intervall zu prüfen.

Eine Möglichkeit hierfür wäre, eine neue Klasse `FilterTermin` zu entwerfen und ein Objekt dieser Klasse der Methode `ermittleFilterListe` als Parameter zu übergeben. Der Unterschied der Klasse `FilterTermin` zur Klasse `Termin` liegt darin, dass für jedes Attribut (abgesehen von der UID) in der Klasse `FilterTermin` ein Ausdruck mit Platzhaltern angelegt wird – dieser kann durch eine Zeichenkette oder eine eigene Klasse repräsentiert werden. Ist die Ausprägung eines Attributs eines Objekts der Klasse `Termin` nun durch den entsprechenden Ausdruck darstellbar, wird der Termin in die Ergebnisliste aufgenommen, andernfalls nicht.

Teilaufgabe e)

Die Behauptung des Schülers ist nicht korrekt. Zwar ist die Begründung richtig, dass durch die Rückgabe der Zugriff auf das existierende Objekt erlangt werden kann, da keine Kopie des Objekts sondern eine Referenz auf das Objekt zurückgegeben wird.
Allerdings ist durch Kapselung gewährleistet, dass Attribute des Objekts nicht geändert werden können.

Hinweis: Auch durch die Anfrage der Bezeichnung der UID kann die Bezeichnung nicht geändert werden, da Objekte der Klasse `String` nicht verändert werden können.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	stellt den entwickelten Algorithmus als Struktogramm dar.	7			
2	implementiert das Durchsuchen der Liste.	4			
3	implementiert das Einfügen.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (14) …… ……					
	Summe Teilaufgabe a)	**14**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	dokumentiert den Durchlauf durch die Liste `termine`.	4			
2	dokumentiert den Durchlauf durch die Liste `ergebnis`.	4			
3	erläutert im Sachzusammenhang, was die Methode leistet.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe b)	**10**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt die Ergebnisliste.	2			
2	modifiziert das Diagramm.	4			
3	erläutert die Veränderungen bei bestehenden Methoden.	3			
4	dokumentiert die ergänzten Methoden.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (13) …… ……					
	Summe Teilaufgabe c)	**13**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet, dass eine solche Filterung nicht möglich ist.	3			
2	erläutert eine mögliche Veränderung des Filtermodells.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe d)	**7**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt, dass der Zugriff auf das Objekt tatsächlich möglich ist.	3			
2	beurteilt, dass eine Veränderung der Attribute nicht möglich ist.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (6) …… ……					
	Summe Teilaufgabe e)	**6**			

	Summe insgesamt	**50**			

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

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

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

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

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

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

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

Name: _____

Abiturprüfung 2017

Informatik, Leistungskurs

Aufgabenstellung:

So genannte Cloud-Dienste synchronisieren durch regelmäßige Kopier- und Löschaktionen Dateiverzeichnisse, die sich auf einem Computer, Tablet, Smartphone o. ä. befinden, mit Dateiverzeichnissen auf Servern. Sie dienen vornehmlich zur Ablage von Sicherungskopien und dazu, die im Netz befindlichen Dateien auf unterschiedlichen Geräten abrufen und dort verarbeiten zu können.

In dieser Aufgabe wird ein vereinfachtes Szenario betrachtet, in dem nur genau zwei Geräte an der Cloud-Anwendung beteiligt sind: das lokale Gerät, auf dem die Originaldateien bearbeitet werden und der Cloud-Server, der die Sicherungskopien verwaltet. Die auf dem Server abgelegten Dateien werden zwischen zwei Synchronisationen nicht verändert. Daher beschränkt sich der Synchronisationsvorgang darauf, dass der Client eine Kopie seines Arbeitsverzeichnisses auf dem Server erzeugt (sogenanntes Spiegeln). Um die Netzlast so gering wie möglich zu halten, sollen Dateien, die seit der letzten Synchronisation nicht verändert wurden, nicht noch einmal übertragen werden. Der Client hat keine Informationen über den letzten Synchronisationsvorgang. Daher müssen alle zur Synchronisation notwendigen Informationen **bei jeder Synchronisation** beim Server abgefragt werden.

Das Arbeitsverzeichnis auf dem lokalen Gerät wird **lokales Verzeichnis**, dasjenige auf dem Server **entferntes Verzeichnis** genannt.

Zunächst kann das lokale Arbeitsverzeichnis lediglich Dateien, nicht aber weitere (Unter-) Verzeichnisse (Ordner) enthalten. Auf der Grundlage dieser Vorgaben wurde das Protokoll, das in Tabelle 1 im Ausschnitt wiedergegeben ist, entwickelt.

Name: _____

Client an Server	Server an Client / Kommentar
Verbindung wird aufgenommen	`CloudServer ok`
`LADEHOCH <Dateiname> <Zeitstempel> <Daten>`	Eine Datei wird auf den Server hochgeladen. Die durch die Platzhalter `<Dateiname>`, `<Zeitstempel>` und `<Daten>` angegebenen Parameter sind jeweils Zeichenketten und werden durch Leerzeichen voneinander getrennt. Als Parameter `<Dateiname>` wird im gesamten Protokoll der Name der Datei innerhalb des lokalen Verzeichnisses (ohne Pfadangabe) eingesetzt. Die Datei wird auf dem Server mit dem durch `<Zeitstempel>` angegebenen Zeitstempel gespeichert. Ein Zeitstempel ist eine Zeichenkette im Format `JJJJ-MM-TT-HH:MM:SS`, der den sekundengenauen Zeitpunkt des Beginns des Synchronisationsprozesses codiert. Es folgen die Bytes der Datei in Form einer Zeichenkette aus Hexadezimalzeichen `<Daten>`. Falls eine gleichnamige Datei schon auf dem Server vorhanden ist, wird diese überschrieben. Mögliche Antworten des Servers: `+OK` im Erfolgsfall oder `–ERR` sonst.
`LOESCHE <Dateiname>`	Die Datei mit dem Namen `<Dateiname>` wird aus dem entfernten Verzeichnis gelöscht. Mögliche Antworten des Servers: `+OK` im Erfolgsfall oder `–ERR` sonst.
`GIBDATEILISTE`	`DATEILISTE:<Dateiname1>:<Dateiname2>:…` Der Server antwortet mit einer Auflistung der Dateinamen im entfernten Verzeichnis in Form einer Zeichenkette. In dieser sind die Dateinamen jeweils durch einen Doppelpunkt („:“) getrennt. Ist das entfernte Verzeichnis leer, so ist der Inhalt dieser Zeichenkette lediglich `DATEILISTE:`
`GIBZEITSTEMPELVONDATEI <Dateiname>`	Rückgabe ist der Zeitstempel der angegebenen Datei im Format `ZEITSTEMPEL <Dateiname> <Zeitstempel>`. Der Zeitstempel gibt den Zeitpunkt an, zu dem die Datei das letzte Mal synchronisiert wurde. Falls die Datei im entfernten Verzeichnis unbekannt ist, wird mit `-ERR` geantwortet.
`LOGOUT`	`Verbindung beendet`

Tabelle 1: Protokoll

Name: _____

a) Im **lokalen** Verzeichnis befinden sich drei Dateien mit den folgenden Namen:

- `Foto1.jpg`
- `Text1.odt`
- `Film1.mp4`

Das **entfernte** Verzeichnis ist leer.

Es wird eine Synchronisation mit dem Zeitstempel `2017-03-01-10:30:00` (d. h. am 1. März 2017 um 10:30 Uhr und 0 Sekunden) durchgeführt.

Danach werden im lokalen Verzeichnis vom Benutzer folgende Änderungen vorgenommen:

- Datei `Foto1.jpg` wird gelöscht.
- Datei `Text1.odt` wird verändert.
- Eine neue Datei `Foto2.jpg` wird erstellt.

Mit dem Zeitstempel `2017-03-01-11:00:00` (d. h. am 1. März 2017 um 11:00 Uhr und 0 Sekunden) wird nun eine zweite Synchronisation durchgeführt.

Stellen Sie den Ablauf beider Synchronisationen unter Verwendung des Protokolls aus Tabelle 1 im Sachzusammenhang dar, indem Sie die zwischen Client und Server ausgetauschten Nachrichten angeben.

Erläutern Sie wesentliche Unterschiede der zweiten Synchronisation im Vergleich zur ersten.

Hinweis: Da die Datenbytes der einzelnen Dateien nicht bekannt sind, sollen diese im Protokoll durch drei Auslassungspunkte (…) angegeben werden.

(10 Punkte)

Name: _____

Es soll ein Informatiksystem entwickelt werden, welches das Spiegeln realer Dateiverzeichnisse ermöglicht, d. h. neben Dateien existieren auch Ordner (insbesondere sind auch Ordner in Ordnern – in beliebig tiefen Verschachtelungen – möglich). Innerhalb eines Ordners sind auf derselben Ebene mehrere gleichnamige Einträge nicht zugelassen (etwa eine Datei und ein Ordner mit gleichem Bezeichner).
Die Implementation der Synchronisation soll auf der Clientseite gemäß dem (vereinfachten) Implementationsdiagramm erfolgen, das in Abbildung 1 dargestellt ist. Die Dokumentation der Klassen finden Sie im Anhang.

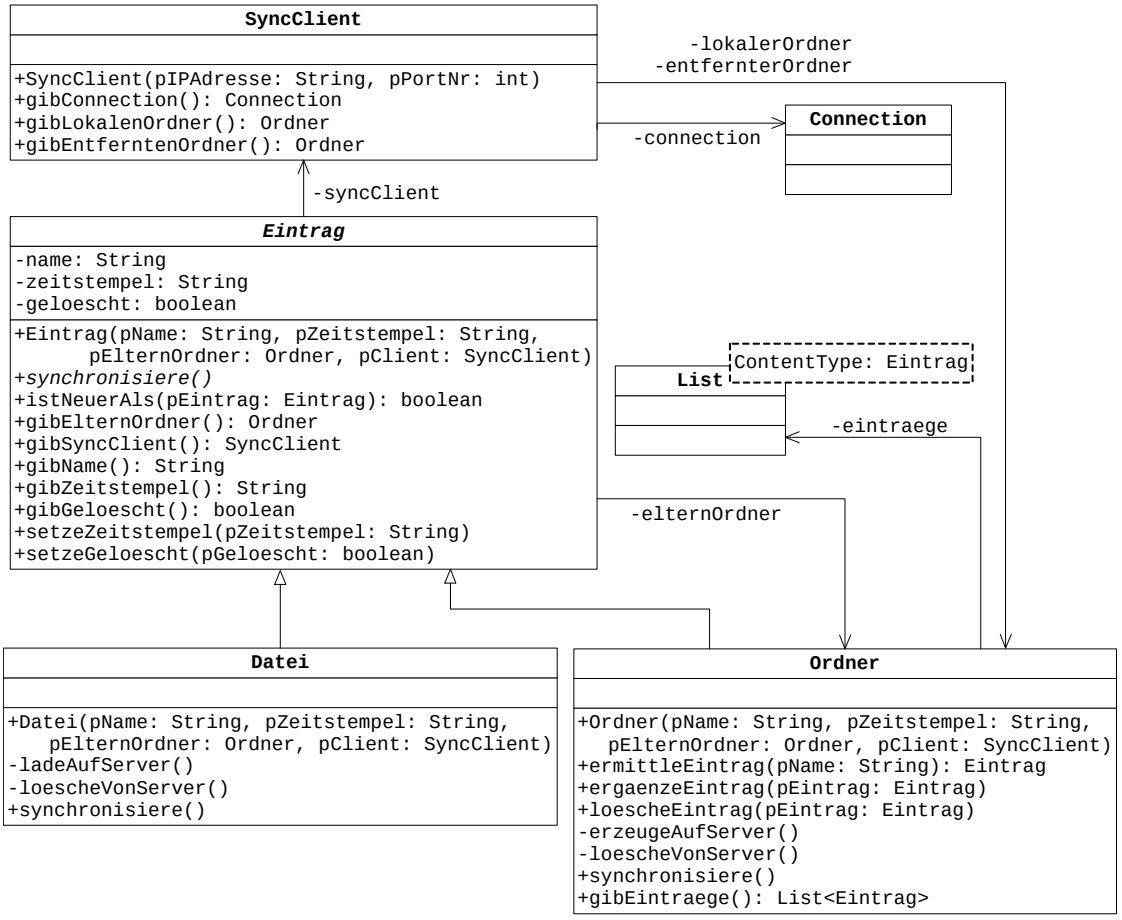


Abbildung 1: Teilmodellierung in Form eines Implementationsdiagramms

Name: _____

b) *Analysieren Sie die durch das Implementationsdiagramm in Abbildung 1 gegebene Modellierung, indem Sie die Bedeutung der Klassen und deren Beziehungen untereinander im Sachzusammenhang erläutern.*

Erläutern Sie dabei insbesondere, wie die rekursive Struktur von Dateiverzeichnissen modelliert wird.

Bestimmen Sie im Sachzusammenhang Besonderheiten, die für das Wurzelverzeichnis (also das zu spiegelnde lokale Arbeitsverzeichnis selbst) gelten.

Begründen Sie, warum die Methode `synchronisiere` *der Klasse* `Eintrag` *als* `abstract` *modelliert wird.*

(12 Punkte)

c) Die Methode `ermittleEintrag` der Klasse `Ordner` hat den folgenden Methodenkopf:

```
public Eintrag ermittleEintrag(String pName)
```

Implementieren Sie die Methode `ermittleEintrag` *der Klasse* `Ordner` *mit der im Anhang beschriebenen Funktionalität.*

Erläutern Sie die Funktionsweise Ihrer Implementation.

(6 Punkte)

d) Die Methode `synchronisiere` der Klasse `Eintrag` ist abstrakt und wird in den Unterklassen `Datei` und `Ordner` realisiert, wobei folgendes gilt:

Vor dem Aufruf der Methode ist sichergestellt, dass die Methoden `gibLokalenOrdner` und `gibEntferntenOrdner` der Klasse `SyncClient` den jeweils aktuellen Zustand des lokalen und entfernten Verzeichnisses zurückgeben. Wurde im lokalen Verzeichnis seit der letzten Synchronisation eine Datei gelöscht, ist das zugehörige `Eintrag`-Objekt immer noch vorhanden, aber der Wert seines Attributs `geloescht` auf `true` gesetzt. In allen anderen Fällen hat es den Wert `false`.

Die Methode `synchronisiere` hat den folgenden Methodenkopf:

```
public void synchronisiere()
```

Analysieren Sie die verschiedenen Situationen, die bei der Synchronisation von `Datei`- *und* `Ordner`-*Objekten auftreten können, indem Sie die notwendigen Operationen darstellen, die sich jeweils daraus ergeben.*

Implementieren Sie die Methode `synchronisiere` *der Klasse* `Datei`.

(12 Punkte)

Name: _____

e) Die Methode `gibAus` dient dazu, die Verzeichnisstruktur in Textform auszugeben. Die Klasse `Eintrag` wird um die öffentliche, abstrakte Methode `gibAus` erweitert, die in den Klassen `Datei` und `Ordner` folgendermaßen realisiert wird:

Implementation in der Klasse `Datei`:

```
1   public String gibAus() {
2     return gibName() + ":";
3   }
```

Implementation in der Klasse `Ordner`:

```
1    public String gibAus() {
2      String s = "<:" + gibName() + ":";
3      gibEintraege().toFirst();
4      while (gibEintraege().hasAccess()) {
5        Eintrag eintrag = gibEintraege().getContent();
6        s = s + eintrag.gibAus();
7        gibEintraege().next();
8      }
9      s = s + ">:";
10     return s;
11   }
```

Folgende Ordnerstruktur befindet sich nun im lokalen Arbeitsverzeichnis:

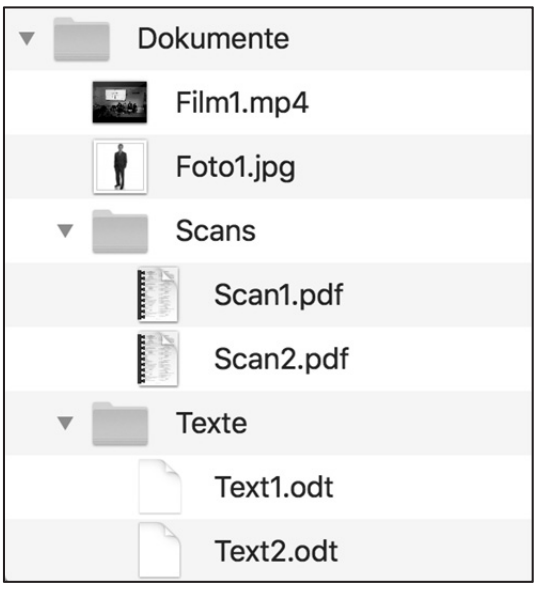


Abbildung 2: Ordner „Dokumente“ mit Einträgen

Name: _____

Ermitteln Sie die Zeichenkette, die durch `gibAus` *erzeugt wird, indem Sie diese Methode für ein dem Ordner „Dokumente“ aus Abbildung 2 entsprechendes* `Ordner`*-Objekt anwenden.*

Erläutern Sie allgemein, auf welche Weise die erzeugte Zeichenkette eine rekursive Ordnerstruktur beschreibt.

Erläutern Sie, wie der Algorithmus von `gibAus` *den vollständigen Ordnerinhalt in eine lineare Darstellung umwandelt, indem Sie Bezüge zu den Quelltexten herstellen.*

(10 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Dokumentationen der verwendeten Klassen

Alle in diesem Abschnitt nicht weiter dokumentierten **`gib`**...- bzw. **`setze`**...-Methoden entsprechen den üblichen Funktionsweisen. Nicht weiter dokumentierte Konstruktoren rufen lediglich den Konstruktor der Oberklasse auf.

Die abstrakte Klasse `Eintrag`

Konstruktor **`Eintrag(String pName, String pZeitstempel, Ordner pElternOrdner, SyncClient pClient)`**

Ein Eintrag wird erzeugt. Erläuterung der Parameter:

- `pName` gibt den Namen des Eintrags an.
- `pZeitstempel` gibt das Datum der letzten Synchronisation des Eintrags als String in der Form `JJJJ-MM-TT-HH:MM:SS` an.
- `pElternOrdner` referenziert das `Ordner`-Objekt, zu dessen Inhaltsliste dieser Eintrag gehört.
- `pClient` referenziert das `SyncClient`-Objekt, mit dessen Hilfe sich der Eintrag ins entfernte Verzeichnis auf dem Server hochladen bzw. sich aus ihm löschen kann. Der Parameter hat nur bei Verwendung dieses Eintrags im lokalen Verzeichnis einen gültigen Wert. Wird dieses `Eintrag`-Objekt zur Beschreibung eines Eintrags im entfernten Verzeichnis benutzt, hat er den Wert `null`.

Anfrage **`boolean istNeuerAls(Eintrag pEintrag)`**

Falls dieser Eintrag zu einem späteren Zeitpunkt synchronisiert wurde als der durch `pEintrag` übergebene Eintrag, wird `true` zurückgegeben, sonst `false`.

Abstrakter Auftrag **`void synchronisiere()`**

Dieses `Eintrag`-Objekt synchronisiert sich mit dem entfernten Verzeichnis, d. h., nach der Ausführung des Auftrags ist der zugehörige Eintrag im entfernten Verzeichnis auf dem gleichen Stand wie dieser Eintrag und das Datenmodell ist auf dem aktuellen Stand.

Name: _____

Die Klasse `Datei`

Auftrag **`void ladeAufServer()`**

Die durch dieses `Datei`-Objekt beschriebene Datei im lokalen Verzeichnis wird zum entfernten Verzeichnis hochgeladen. Ist eine gleichnamige Datei im entfernten Verzeichnis schon vorhanden, wird diese überschrieben. Die Zeitstempel aller übergeordneten Ordner dieser Datei im lokalen und entfernten Verzeichnis werden auf den Zeitstempel dieses `Datei`-Objekts gesetzt. Eine Aktualisierung im Datenmodell wird nicht vorgenommen.

Auftrag **`void loescheVonServer()`**

Die durch dieses `Datei`-Objekt beschriebene Datei wird aus dem entfernten Verzeichnis gelöscht. Die Zeitstempel aller übergeordneten Ordner der durch dieses `Datei`-Objekt beschriebenen Datei im entfernten Verzeichnis werden auf den Zeitpunkt gesetzt, an dem die Löschung erfolgt. Eine Aktualisierung im Datenmodell wird nicht vorgenommen.

Die Klasse `Ordner`

Konstruktor **`Ordner(String pName, String pZeitstempel, Ordner pElternOrdner, SyncClient pClient)`**

Der Konstruktor ruft den Konstruktor der Oberklasse auf und legt eine leere Eintragsliste an.

Anfrage **`Eintrag ermittleEintrag(String pName)`**

Wenn in der Liste der Einträge ein `Eintrag`-Objekt existiert, das den gleichen Dateinamen hat wie durch `pName` übergeben, wird dieser `Eintrag` zurückgegeben, sonst `null`.

Auftrag **`void ergaenzeEintrag(Eintrag pEintrag)`**

Das übergebene `Eintrag`-Objekt wird der Liste der Einträge dieses `Ordners` in alphabetischer Sortierung hinzugefügt. Ist ein gleichnamiger Eintrag schon vorhanden, geschieht nichts.

Auftrag **`void loescheEintrag(Eintrag pEintrag)`**

Der durch `pEintrag` referenzierte Eintrag wird aus der Liste der Einträge dieses `Ordner`-Objekts gelöscht. Falls er nicht gefunden werden kann, geschieht nichts.

Name: _____

Auftrag **`void erzeugeAufServer()`**

Im aktuellen entfernten Verzeichnis wird ein Ordner mit dem gleichen Namen, wie durch dieses Ordner-Objekt vorgegeben, angelegt. Ist ein gleichnamiger Ordner dort schon vorhanden, geschieht nichts. Die Zeitstempel aller übergeordneten Ordner des durch dieses `Ordner`-Objekt beschriebenen Ordners im aktuellen lokalen und entfernten Verzeichnis werden auf den Zeitstempel dieses `Ordner`-Objekts gesetzt. Eine Aktualisierung im Datenmodell wird nicht vorgenommen.

Auftrag **`void loescheVonServer()`**

Der Ordner im entfernten Verzeichnis, der den gleichen Namen hat wie durch dieses `Ordner`-Objekt angegeben, wird mitsamt seinem Inhalt gelöscht. Die Zeitstempel aller übergeordneten Ordner des durch dieses `Ordner`-Objekt beschriebenen Ordners im aktuellen lokalen und entfernten Verzeichnis werden auf den Zeitpunkt gesetzt, an dem die Löschung erfolgt. Eine Aktualisierung im Datenmodell wird nicht vorgenommen.

Die Klasse `SyncClient`

Konstruktor **`SyncClient(String pIPAdresse, int pPortNr)`**

Ein Objekt der Klasse `SyncClient` wird erzeugt und die Verbindung zum Cloud-Server über IP-Adresse `pIPAdresse` und Port `pPortNr` hergestellt.

Name: _____

Die generische Klasse List<ContentType>

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

Dokumentation der Klasse `List<ContentType>`

Konstruktor **`List()`**

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

Anfrage **`boolean isEmpty()`**

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

Anfrage **`boolean hasAccess()`**

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

Auftrag **`void next()`**

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

Auftrag **`void toFirst()`**

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

Auftrag **`void toLast()`**

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

Name: _____

Anfrage **`ContentType getContent()`**

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

Auftrag **`void setContent(ContentType pContent)`**

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

Auftrag **`void append(ContentType pContent)`**

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

Auftrag **`void insert(ContentType pContent)`**

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

Auftrag **`void concat(List<ContentType> pList)`**

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

Auftrag **`void remove()`**

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

Unterlagen für die Lehrkraft

Abiturprüfung 2017

Informatik, Leistungskurs

1. Aufgabenart

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

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2017

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Objekte und Klassen
     - Entwurfsdiagramme und Implementationsdiagramme
     - Lineare Strukturen
       Lineare Liste (Klasse `List`*)*

   Algorithmen
   - Analyse, Entwurf und Implementierung von Algorithmen

   Informatiksysteme
   - Einzelrechner und Rechnernetzwerke

2. *Medien/Materialien*
   - entfällt

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

5. Zugelassene Hilfsmittel

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

6. Modelllösungen

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

Teilaufgabe a)

Die erste Synchronisation bei leerem entfernten Verzeichnis hätte folgenden Ablauf:

	Client an Server	**Server an Client**
1	Verbindungsaufnahme	`CloudServer ok`
2	`GIBDATEILISTE`	`DATEILISTE:`
3	`LADEHOCH␣Foto1.jpg␣2017-03-01-10:30:00␣…`	`+OK`
4	`LADEHOCH␣Text1.odt␣2017-03-01-10:30:00␣…`	`+OK`
5	`LADEHOCH␣Film1.mp4␣2017-03-01-10:30:00␣…`	`+OK`
6	`LOGOUT`	`Verbindung beendet`

Anmerkung für die korrigierende Lehrkraft:
Zur besseren Erkennbarkeit sind hier die obligatorischen Leerzeichen zur Parametertrennung im Befehl `LADEHOCH` durch das „Open Box“-Sonderzeichen (␣) dargestellt.

Da die vom Server übertragene Dateiliste leer ist, müssen alle lokal vorhandenen Dateien auf jeden Fall übertragen werden. Der Zeitstempel ergibt sich aus der Aufgabenstellung; die Angabe konkreter Datenbytes ist nicht möglich.

Der Aufruf von `GIBDATEILISTE` ist auch im speziellen Fall des leeren entfernten Verzeichnisses notwendig, da die Clientseite keine Kenntnis über dessen Inhalt hat und daher nicht von einem leeren Verzeichnis ausgehen darf. Mit Blick auf eine allgemeine Lösung ist das Ermitteln des Inhalts des entfernten Verzeichnisses zu Beginn der Synchronisation notwendig.

Nach den beschriebenen Veränderungen am lokalen Verzeichnis läuft die nächste Synchronisation folgendermaßen ab:

	Client an Server	**Server an Client**
1	Verbindungsaufnahme	`CloudServer ok`
2	`GIBDATEILISTE`	`DATEILISTE:Foto1.jpg:` `Text1.odt:Film1.mp4`
3	`LOESCHE␣Foto1.jpg`	`+OK`
4	`GIBZEITSTEMPELVONDATEI␣Text1.odt`	`ZEITSTEMPEL␣Text1.odt␣` `2017-03-01-10:30:00`
5	`LADEHOCH␣Text1.odt␣` `2017-03-01-11:00:00␣...`	`+OK`
6	`GIBZEITSTEMPELVONDATEI␣Film1.mp4`	`ZEITSTEMPEL␣Film1.mp4␣` `2017-03-01-10:30:00`
7	`LADEHOCH␣Foto2.jpg␣` `2017-03-01-11:00:00␣...`	`+OK`
8	`LOGOUT`	`Verbindung beendet`

Im Unterschied zur ersten Synchronisation ist die vom Server zu Beginn dieser Synchronisation angeforderte Dateiliste nicht leer (Zeile 2). Daher muss das jeweilige Datum der letzten Synchronisation (Zeitstempel) der Dateien in der Liste ggf. angefordert und ausgewertet werden.

- Zeile 3: Die lokal bereits gelöschte Datei `Foto1.jpg` muss in jedem Fall aus dem entfernten Verzeichnis gelöscht werden, daher ist das Abfragen ihres Zeitstempels überflüssig.
- Zeile 4/5: Die Datei `Text1.odt` ist im lokalen und entfernten Verzeichnis vorhanden. Ihr Zeitstempel muss ausgewertet werden, um festzustellen, ob die Datei im lokalen Verzeichnis neueren Datums ist als die im entfernten. Die Datei auf dem Server ist älter ist als die lokale, daher muss `Text1.odt` erneut, mit neuem Zeitstempel, hochgeladen werden.
- Zeile 6: Auch für die Datei `Film1.mp4` muss der Zeitstempel ausgewertet werden; der Vergleich mit der lokalen Version zeigt aber, dass die Datei nicht verändert wurde und daher keine weitere Operation erfolgen darf.
- Zeile 7: Die Datei `Foto2.jpg` taucht nicht in der in Zeile 2 übertragenen Dateiliste auf. Daher muss sie auf jeden Fall hochgeladen werden.

Teilaufgabe b)

Zentrale Klassen in diesem Modell sind `Eintrag`, `Ordner`, `Datei` und `SyncClient`.
Die Modellierung beschreibt ein Datenmodell, das den Zustand der realen Verzeichnisse auf dem Client und dem Server beschreibt.
Ein Objekt der Klasse `Eintrag` repräsentiert im Datenmodell abstrakt einen Eintrag in einem Dateiverzeichnis. Ein solcher Eintrag kann konkret entweder eine Datei oder ein Ordner sein, was durch die konkreten Unterklassen `Datei` und `Ordner` modelliert wird. Die abstrakte Oberklasse `Eintrag` besitzt die Methoden und Attribute, die zum Beschreiben und Vergleichen von `Eintrag`-Objekten notwendig sind, etwa für Name und Zeitstempel.
Jedes Objekt der Klasse `Eintrag` besitzt über eine Assoziation `elternOrdner` Zugriff auf dasjenige `Ordner`-Objekt, zu dessen Inhalt es gehört. Zur Verwaltung dieses Inhalts besitzt jedes Objekt der Klasse `Ordner` eine generische lineare Liste `eintraege` aus `Eintrag`-Objekten.
Die Modellierung impliziert, dass das lokale und das entfernte Verzeichnis als `Ordner`-Objekte repräsentiert werden; da diese aber - als Wurzelverzeichnisse - keinen übergeordneten Ordner besitzen, darf in diesem (und nur diesem) Fall `elternOrdner` nicht auf ein weiteres `Ordner`-Objekt verweisen und muss mithin den Wert `null` haben. Da jedes Objekt der Klasse `Eintrag` auch ein `Ordner`-Objekt sein kann, wird so die rekursive Struktur realer Dateiverzeichnisse modelliert (Ordner als Inhalte von Ordnern sind möglich).
Die Klassen `Datei` und `Ordner` sind konkrete Unterklassen der Klasse `Eintrag`, die die Methode `synchronisiere` implementieren. Diese Methode muss innerhalb von `Eintrag` abstrakt bleiben, da das Synchronisieren von `Datei`- und `Ordner`-Objekten unterschiedliche Verfahren sind: Im Fall von Objekten der Klasse `Datei` ist das Hochladen oder Löschen genau einer Datei ausreichend, im Fall von Objekten der Klasse `Ordner` muss die rekursive Struktur in einer Traversierung abgearbeitet werden.
Die Klasse `SyncClient` repräsentiert in diesem Modell ein Objekt, das über seine `Connection`-Assoziation die Kommunikation mit dem Server durchführen kann, sowie die aktuellen Inhalte des Datenmodells des lokalen und entfernten Verzeichnisses als `Ordner`-Objekte angeben kann. Da alle `Eintrag`-Objekte mittels ihrer Assoziation `syncClient` Zugriff auf ein `SyncClient`-Objekt haben, haben sie so mittelbar auch Zugriff auf das Datenmodell des lokalen und entfernten Verzeichnisses sowie auf Sende- und Empfangsmethoden. Dies ist nötig, um die Methode `synchronisiere` in den Klassen `Datei` und `Ordner` konkret implementieren zu können.

Teilaufgabe c)

Die zu implementierende Methode muss die Inhaltsliste eines Objekts der Klasse `Ordner` auf `Eintrag`-Objekte hin durchsuchen, die den gleichen Namen haben, wie durch den Parameter übergeben.

Eine mögliche Implementation der Methode lautet:

```
 1 public Eintrag ermittleEintrag(String pName) {
 2   Eintrag gefunden = null;
 3   gibEintraege().toFirst();
 4   while (gibEintraege().hasAccess() && gefunden==null) {
 5     Eintrag aktEintrag = gibEintraege().getContent();
 6     if (aktEintrag.gibName().equals(pName)) {
 7       gefunden = aktEintrag;
 8     } else {
 9       gibEintraege().next();
10     }
11   }
12   return gefunden;
13 }
```

Die lokale Variable `gefunden` wird dazu genutzt, das ggf. gefundene `Eintrag`-Objekt zu referenzieren und wird daher zunächst auf `null` gesetzt (Z. 2). Die Liste der Einträge wird auf den Anfang zurückgesetzt und durchlaufen, bis entweder ihr Ende erreicht ist oder ein `Eintrag`-Objekt des gesuchten Namens gefunden wurde (was dadurch angezeigt wird, dass die Variable `gefunden` nicht mehr den Wert `null` hat) (Z. 4). In der Schleife wird der jeweils in der Liste besuchte Eintrag namensmäßig mit dem Wert des Parameters verglichen und abhängig vom Ergebnis entweder der Wert von `gefunden` auf das aktuelle Element gesetzt oder mit dem Durchlaufen fortgefahren (Z. 5 – 10).

Teilaufgabe d)

Bei der Synchronisation jedes Objektes der Klasse `Eintrag` aus dem lokalen Verzeichnis mit dem entfernten Verzeichnis können folgende Situationen vorkommen:

1) Das betrachtete `Eintrag`-Objekt wurde nach der letzten Synchronisation neu erzeugt und befindet sich daher noch nicht in der Eintragsliste des dem entfernten Verzeichnis entsprechenden `Ordner`-Objekts. In diesem Fall muss hochgeladen werden und die Inhaltsliste des dem entfernten Verzeichnis entsprechenden `Ordner`-Objekts um das neue `Eintrag`-Objekt erweitert werden.
   - Im Falle eine `Datei`-Objekts besteht das Hochladen lediglich aus dem Senden einer Datei.
   - Im Falle eines `Ordner`-Objekts muss der Ordner im entfernten Verzeichnis erzeugt und sein kompletter Inhalt übertragen werden. Dies bedeutet insbesondere, dass das Hochladen eines `Ordner`-Objekts rekursiv vonstattengehen muss, da er sowohl `Datei`-Objekte als auch weitere `Ordner`-Objekte beinhalten kann.

2) Das `Eintrag`-Objekt befindet sich bereits im entfernten Verzeichnis. In diesem Fall muss überprüft werden, ob die lokale Variante einen neueren Zeitstempel trägt als die entfernte:
   - Falls das lokale `Eintrag`-Objekt einen späteren Zeitstempel trägt als das entfernte, muss der lokale Eintrag hochgeladen werden.
     - Im Falle eines `Datei`-Objekts genügt zur Aktualisierung das Hochladen der Datei auf den Server.
     - Im Falle eines `Ordner`-Objekts muss für alle seine Inhaltsobjekte rekursiv eine Synchronisation der hier beschriebenen Art angestoßen werden. Da der Client anhand der unterschiedlichen Zeitstempel lediglich feststellen kann, *dass* sich in dem betrachteten Ordner etwas verändert hat, nicht aber *was*, kann für jedes `Eintrag`-Objekt, das sich in seiner Inhaltsliste oder in einer der Inhaltslisten enthaltener Ordner befindet, einer der in dieser Aufgabe untersuchten 3 Fälle auftreten. Es muss also rekursiv für jedes `Eintrag`-Objekt der Inhaltsliste überprüft und unterschieden werden, ob es auf dem Server unbekannt ist und daher hochgeladen werden muss, ob es auf dem Server bekannt ist und daher nach einem Zeitstempel-Vergleich ggf. erneut hochgeladen werden muss oder ob es auf dem Client als gelöscht markiert ist und daher auch vom Server gelöscht werden muss.
   - Falls das lokale `Eintrag`-Objekt den gleichen Zeitstempel trägt wie das entfernte, ist keine weitere Operation nötig.

   Anmerkung für die korrigierende Lehrkraft:
   Der Fall, dass sich auf dem Server eine neuere Variante des Eintrags befindet, wurde durch die in der Einleitung genannten vereinfachenden Voraussetzungen ausgeschlossen und muss daher vom Prüfling nicht berücksichtigt werden.

3) Das `Eintrag`-Objekt im lokalen Verzeichnis wurde seit der letzten Synchronisation gelöscht, was durch das gesetzte Flag `geloescht` angezeigt wird. In diesem Fall muss sein Pendant auf dem Server ebenfalls gelöscht werden und das `Eintrag`-Objekt aus dem lokalen Verzeichnis endgültig entfernt werden.

   Anmerkung für die korrigierende Lehrkraft:
   In diesem Fall ist eine Unterscheidung zwischen Datei und Ordner zwar prinzipiell nicht erforderlich, gleichwohl erfordern reale Dateiübertragungsprotokolle in der Regel, dass Ordner vor dem Löschen geleert sein müssen, was wiederum zuvor ein rekursives Löschen der Ordnerinhalte notwendig macht. Entsprechende Lösungen sind als richtig zu werten.

In allen 3 Fällen muss nach der Synchronisationsoperation das clientseitige Datenmodell auf den aktuellen Stand gebracht werden (vgl. Spezifikation der Methode `synchronisiere`).

Eine mögliche Java-Implementation der Methode `synchronisiere` der Klasse `Datei`:

```
1   public void synchronisiere() {
2     Ordner entfernterOrdner =
                              gibSyncClient().gibEntferntenOrdner();
3     Eintrag eintragAufServer =
                        entfernterOrdner.ermittleEintrag(gibName());

4     if (gibGeloescht()) {
5       loescheVonServer();
6       gibElternOrdner().loescheEintrag(this);
7       entfernterOrdner.loescheEintrag(eintragAufServer);
8     } else {
9       if (eintragAufServer != null) {
10        if (istNeuerAls(eintragAufServer)) {
11          ladeAufServer();
12          entfernterOrdner.loescheEintrag(eintragAufServer);
13          entfernterOrdner.ergaenzeEintrag(eintragAufServer);
14        }
15      } else {
16        ladeAufServer();
17       entfernterOrdner.ergaenzeEintrag(this);
18      }
19    }
20  }
```

Anmerkungen für die korrigierende Lehrkraft:

- Die hier vorgestellte mögliche Implementation geht in Z. 2/3 davon aus, dass die Methode `ermittleEintrag` in diesem Fall tatsächlich ein `Datei`- (und kein `Ordner`-) Objekt gleichen Namens findet oder `null` zurückgibt. Im Rahmen der Aufgabenstellung ist dies eine zulässige Annahme, da der Fall, dass beispielsweise auf dem Server ein `Ordner`- (und kein `Datei`-) Objekt gleichen Namens gefunden wird, dadurch ausgeschlossen wird, dass der Inhalt des Servers ausschließlich von genau einem Client verwaltet wird.
- In Ermangelung einer Methode zum Aktualisieren eines schon vorhandenen `Eintrag`-Objektes in der Inhaltsliste von `Ordner`-Objekten wird in Zeile 12/13 das zu synchronisierende `Datei`-Objekt erst aus dieser Liste gelöscht und dann wieder neu hinzugefügt, damit es mit aktuellem Zeitstempel in der Liste geführt wird.

Teilaufgabe e)

Die Rückgabe der Anfrage lautet

```
<:Dokumente:Film1.mp4:Foto1.jpg:<:Scans:Scan1.pdf:Scan2.pdf:>
:<:Texte:Text1.odt:Text2.odt:>:>:
```

Die Zeichenkette ist aus Tokens zusammengesetzt, die durch das Zeichen „`:`“ getrennt sind. Ist ein Token gleich dem Zeichen „`<`“, bedeutet dies, dass es sich beim auf ihn folgenden Token um einen Ordnernamen handelt. Alle folgenden Tokens bilden die Namen der in diesem Ordner enthaltenen Einträge, bis das Token „`>`“ das Ende der Auflistung des Ordnerinhalts markiert. Die Tokens „`<`“ und „`>`“ dienen also als „Klammerpaar“ für die Auflistung eines Ordnerinhalts und werden im Falle verschachtelter Ordner ebenfalls verschachtelt.
Der Algorithmus erzeugt durch eine Tiefentraversierung des Ordnerinhalts eine Zeichenkette, die die Namen und Hierarchieinformationen des Ordners in beliebiger Verschachtelungstiefe zurückliefert und damit den gesamten Ordnerinhalt serialisiert.
Für Objekte der Klasse `Datei` liefert die Methode lediglich den Dateinamen mit einem angehängten „`:`“-Zeichen.
In der Implementation für Objekte der Klasse `Ordner` wird zunächst in Zeile 2 das jeden Ordner einleitende Token „`<`“ und die Angabe des Ordnernamens samt Trennzeichen erzeugt. Im Anschluss wird die Inhaltsliste durchlaufen (Zeile 3 – 8) und für jeden darin enthaltenen `Eintrag` durch `gibAus` ein neues Token oder eine Gruppe von Tokens erzeugt, je nachdem, ob das als Laufmarke benutzte Objekt `eintrag` gerade aus der Klasse `Datei` oder `Ordner` ist. Das ggf. rekursive Herabsteigen in den Verzeichnisgraph geschieht dabei implizit in Zeile 6.
Das Abschlusstoken „`>`“ für den Ordner wird samt Trennzeichen „`:`“ in Zeile 9 angehängt.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	stellt die erste Kommunikation gemäß dem Protokoll dar.	3			
2	stellt die zweite Kommunikation gemäß dem Protokoll dar.	3			
3	erläutert wesentliche Unterschiede der zweiten Synchronisation im Vergleich zur ersten Synchronisation.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe a)	**10**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert das Modell, indem Bedeutung und Beziehungen von `Eintrag`, `Datei`, `Ordner` und `SyncClient` erläutert werden.	6			
2	erläutert, wie die rekursive Struktur von Dateiverzeichnissen modelliert wird.	2			
3	bestimmt Besonderheiten des Wurzelverzeichnisses.	2			
4	begründet, warum die Methode `synchronisiere` abstrakt ist.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe b)	**12**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die Methode.	4			
2	erläutert die Funktionsweise der Implementation.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (6) …… ……					
	Summe Teilaufgabe c)	**6**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert den Fall „neuer Eintrag“, indem er die notwendigen Operationen darstellt.	2			
2	analysiert den Fall „bekannter Eintrag“, indem er die notwendigen Operationen darstellt.	3			
3	analysiert den Fall „Eintrag lokal gelöscht“, indem er die notwendigen Operationen darstellt.	2			
4	implementiert die Methode.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt die Rückgabe.	3			
2	erläutert, auf welche Weise die zurückgegebene Zeichenkette eine rekursive Ordnerstruktur beschreibt.	3			
3	erläutert, wie der Algorithmus den Ordnerinhalt in eine lineare Darstellung umwandelt und stellt Bezüge zu den Quelltexten her.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe e)	**10**			

	Summe insgesamt	**50**			

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

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

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

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

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

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

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

Name: _____

Abiturprüfung 2017

Informatik, Leistungskurs

Aufgabenstellung:

Am Edgar-F.-Codd-Gymnasium soll als Angebot im Rahmen der individuellen Förderung die Teilnahme an Wettbewerben unterstützt und begleitet werden. Ein Wettbewerb ist eine Veranstaltung, bei der die Teilnehmerinnen und Teilnehmer gegeneinander antreten, um ihre Leistungen miteinander zu vergleichen, und bei der es für die besten Preise gibt. Wettbewerbe haben eindeutige Bezeichnungen und die Teilnahmebedingungen sind häufig auf einer Website beschrieben. Einige Wettbewerbe werden in unterschiedlichen Runden (Schulrunde, Regionalrunde etc.) ausgetragen.

Der Informatikkurs der Q1 wird damit beauftragt, eine Datenbanklösung für die Erfassung der relevanten Daten zu entwickeln. Der Kurs macht folgenden Vorschlag:

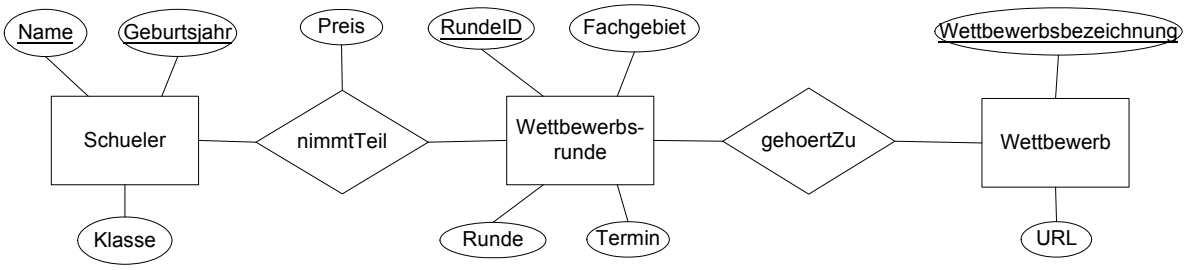


Abbildung 1: ER-Diagramm

```
Schueler(Name, Geburtsjahr, Klasse, ↑RundeID, Preis)
Wettbewerbsrunde(RundeID, Fachgebiet, Runde, Termin)
Wettbewerb(Wettbewerbsbezeichnung, URL)
gehoertZu(↑RundeID, ↑Wettbewerbsbezeichnung)
```

Abbildung 2: Datenbankschema

Eine Beispieldatenbank befindet sich im Anhang.

Dieses Schema soll in den nächsten Teilaufgaben auf Praxistauglichkeit getestet werden.

Name: _____

a) *Erläutern Sie die Informationen, die über Heike Heigel in der Beispieldatenbank stehen, im Sachzusammenhang.*

(4 Punkte)

b) Um die Beispieldatenbank zu testen werden, folgende Anfragen ausprobiert:

(i)
```
1 SELECT Schueler.Name
2 FROM Schueler, Wettbewerbsrunde
3 WHERE Schueler.RundeID = WettbewerbsRunde.RundeID
  AND WettbewerbsRunde.Fachgebiet = "Mathematik"
```

(ii)
```
1 SELECT Wettbewerb.Wettbewerbsbezeichnung,
                                          COUNT(*) AS Anzahl
2 FROM (Wettbewerb
3   JOIN gehoertZu
      ON Wettbewerb.Wettbewerbsbezeichnung =
                          gehoertZu.Wettbewerbsbezeichnung)
4   JOIN Schueler
      ON gehoertZu.RundeID = Schueler.RundeID
5 GROUP BY Wettbewerb.Wettbewerbsbezeichnung
```

(iii)
```
1 SELECT Wettbewerb.Wettbewerbsbezeichnung,
                                       Wettbewerbsrunde.Termin
2 FROM (Wettbewerb
3   JOIN gehoertZu
      ON gehoertZu.Wettbewerbsbezeichnung =
                         Wettbewerb.Wettbewerbsbezeichnung)
4   JOIN Wettbewerbsrunde
      ON gehoertZu.RundeID = Wettbewerbsrunde.RundeID
5 ORDER BY Wettbewerbsrunde.Termin
```

Anmerkung: In einigen Datenbanksystemen muss das Schlüsselwort `JOIN` durch `INNER JOIN` ersetzt werden.

Analysieren Sie die SQL-Anweisungen und bestimmen Sie die Informationen, die in der Datenbank gesucht werden.

(15 Punkte)

Name: _____

c) Die folgenden Anfragen sollen in SQL umgesetzt werden:

(i) Es sollen alle Oberstufenschüler (Klasse EF, Q1, Q2) ermittelt werden, die an einem Wettbewerb teilgenommen haben.
(ii) Gesucht sind die Namen der Schülerinnen oder Schüler sowie die Termine, an denen die Schülerin oder der Schüler einen 1. Platz belegt hat. Zusätzlich soll das Fachgebiet mit ausgegeben und das Ergebnis nach diesen sortiert werden.
(iii) Es sollen die Wettbewerbsbezeichnungen der Wettbewerbe ermittelt werden, an denen noch kein Schüler der Schule teilgenommen hat.

Entwickeln Sie für diese drei genannten Suchaufträge geeignete SQL-Anweisungen, mit deren Hilfe die geforderten Informationen verfügbar werden.

(15 Punkte)

d) Heike Heigel hat am 12.04.2017 am Landeswettbewerb Jugend forscht im Fach Technik den 1. Platz erreicht. Diese Information soll in die Beispieldatenbank aufgenommen werden.

Erläutern Sie, inwieweit das vorliegende Datenbankschema geeignet ist, diese Information aufzunehmen.

(4 Punkte)

e) Die Arbeit mit dem Entwurf und die Umsetzung im Datenbankschema zeigen Schwachstellen. Daher soll der Entwurf und die Umsetzung in das Datenbankschema grundsätzlich überarbeitet werden.

Zusätzlich erhält jede Schülerin und jeder Schüler für die Teilnahme an einer Wettbewerbsveranstaltung eine Lehrperson als Betreuer. Die Betreuer sollen ebenfalls in der Datenbank erfasst werden.

Das neue Datenbankschema soll außerdem so aufgebaut sein, dass es der dritten Normalform genügt.

Beurteilen Sie die Umsetzung des bisherigen Entwurfs in das Datenbankschema.

Entwickeln Sie ein ER-Modell, das die bisherigen Probleme korrigiert und die Betreuer der Wettbewerbe mit berücksichtigt. Geben Sie die Kardinalitäten an und erläutern Sie Ihre Entscheidungen.

Überführen Sie das neue ER-Modell in ein Datenbankschema, das in dritter Normalform ist, und erläutern Sie Ihre Entscheidungen.

(12 Punkte)

Name: _____

Zugelassene Hilfsmittel:

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

Anlage: Auszug aus der Beispieldatenbank

Schueler				
Name	**Geburtsjahr**	**Klasse**	**RundeID**	**Preis**
Eva Hansen	2004	08C	1	Teilnahme
Friedrich Meister	2004	08A	3	Platz 2
Heike Heigel	2002	EF	1	Platz 1
Thomas Meyer	1999	Q2	2	Platz 1

Wettbewerbsrunde			
RundeID	**Fachgebiet**	**Runde**	**Termin**
1	Technik	regional	24.02.2017
2	Informatik	bundesweit	15.11.2016
3	Mathematik	lokal	07.10.2016
4	Mathematik	lokal	24.03.2017

Wettbewerb	
Wettbewerbsbezeichnung	**URL**
Jugend forscht	www.jugend-forscht.de
Informatik-Biber	informatik-biber.de
Mathematik Olympiade	www.mathematik-olympiaden.de
Känguru Wettbewerb	www.mathe-kaenguru.de
Bundeswettbewerb Informatik	www.bwinf.de

gehoertZu	
RundeID	**Wettbewerbsbezeichnung**
1	Jugend forscht
2	Informatik-Biber
3	Mathematik Olympiade
4	Känguru Wettbewerb

Unterlagen für die Lehrkraft

Abiturprüfung 2017

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Abfrage relationaler Datenbanken

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2017

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Datenbanken

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

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

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

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

6. Modelllösungen

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

Teilaufgabe a)

Heike Heigel wurde im Jahr 2002 geboren und besucht die Klasse EF. Sie hat am 24.02.2017 am Regionalwettbewerb Jugend forscht im Fachgebiert Technik teilgenommen und den 1. Platz erreicht.

Teilaufgabe b)

(i) Die Abfrage sucht die Namen aller Schüler, die an einer Wettbewerbsrunde aus dem Fachgebiet Mathematik teilgenommen haben.
Zunächst werden die Tabellen `Schueler` und `Wettbewerbsrunde` miteinander kombiniert (kartesisches Produkt). Anschließend werden die Zeilen gesucht, in denen die `RundeID` der Ausgangstabellen übereinstimmt und das zugehörige Fachgebiet der Mathematik entspricht.

(ii) Die Abfrage ermittelt, wieviele Schülerinnen und Schüler an welchen Wettbewerben teilgenommen haben.
Die Tabellen `Wettbewerb`, `gehoertZu` und `Schueler` werden mit Hilfe geeigneter Attribute durch `JOIN` verbunden. Die Ergebnistabelle wird nach Wettbewerbsbezeichnungen gruppiert. Anschließend wird die Anzahl der Elemente jeder Gruppe (also die Anzahl der Schülerinnen und Schüler / Teilnehmerinnen und Teilnehmer) ermittelt und gemeinsam mit der Wettbewerbsbezeichnung ausgegeben.

(iii) Die Abfrage gibt die Wettbewerbsbezeichnungen aller Wettbewerbe zusammen mit ihren jeweiligen Terminen (bzw. mit den Terminen einer Wettbewerbsrunde dieses Wettbewerbs) aus.
Zunächst werden die Tabellen `Wettbewerb`, `gehoertZu` und `Wettbewerbsrunde` mit Hilfe geeigneter Attribute durch `JOIN` verbunden. Die Ergebnistabelle wird nach den Terminen sortiert und die Termine mit den zugehörigen Wettbewerbsbezeichnungen ausgegeben.

Teilaufgabe c)

Anmerkung für die korrigierende Lehrkraft:
In einigen Datenbanksystemen muss das Schlüsselwort `JOIN` durch `INNER JOIN` ersetzt werden.

(i)
```
SELECT Schueler.Name, Schueler.Klasse
FROM Schueler
WHERE Schueler.Klasse = "Q1" OR Schueler.Klasse = "Q2"
                                        OR Schueler.Klasse = "EF"
```

(ii)
```
SELECT Schueler.Name, Wettbewerbsrunde.Fachgebiet,
                                          Wettbewerbsrunde.Termin
FROM Wettbewerbsrunde JOIN Schueler
  ON Wettbewerbsrunde.RundeID = Schueler.RundeID
    WHERE Preis LIKE "Platz 1"
    ORDER BY WettbewerbsRunde.Fachgebiet
```

(iii)
```
SELECT Wettbewerb.Wettbewerbsbezeichnung
FROM Wettbewerb
WHERE Wettbewerb.Wettbewerbsbezeichnung NOT IN
(SELECT DISTINCT gehoertZu.Wettbewerbsbezeichnung
 FROM Schueler JOIN gehoertZu
    ON gehoertZu.RundeID = Schueler.RundeID)
```

Teilaufgabe d)

Das Einfügen dieser Information ist aufgrund der Primärschlüsseleigenschaft von `Name` und `Geburtsjahr` in der Tabelle `Schueler` nicht möglich, da Heike Heigel (2002) darin bereits existiert.

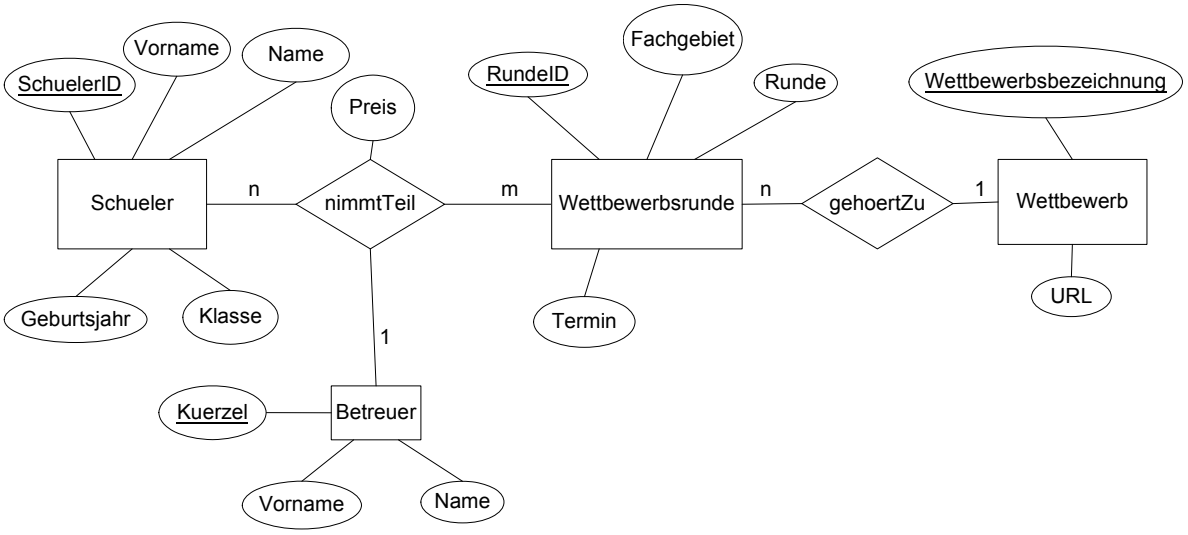
Die Information kann nur eingefügt werden, indem die bisherige Information überschrieben wird.

Teilaufgabe e)

Umsetzung des bisherigen Entwurfs:

Die `1:n`-Beziehung, die zwischen den Entitätstypen `Wettbewerbsrunde` und `Wettbewerb` besteht, ist unpassend umgesetzt worden. Die zusätzliche Relation `gehoertZu` ist überflüssig. Der Schlüssel `Wettbewerbsbezeichnung` kann als Fremdschlüssel in die Relation `Wettbewerbsrunde` aufgenommen werden.

Die n:m-Beziehung, die zwischen den Entitätstypen Schueler und Wettbewerbsrunde besteht, ist falsch umgesetzt worden. Ein Schüler kann an mehreren Veranstaltungen teilnehmen und an einer Veranstaltung können mehrere Schüler teilnehmen. Diese Beziehung erfordert eine Relation nimmtTeil.
Der Primärschlüssel Name und Geburtsjahr ist ungeeignet. An einer Schule können mehrere Schüler mit gleichem Namen und Geburtsjahr existieren. Man kann dieses Problem lösen, indem man beim Entitätstyp Schueler ein Attribut SchuelerID ergänzt und dieses Attribut zum Primärschlüssel wählt. Daraus ergibt sich, dass es keine transitive Abhängigkeit von Nichtschlüsselattributen gibt.



Datenbankschema

```
Wettbewerb(Wettbewerbsbezeichnung, URL)
Wettbewerbsrunde(RundeID, Fachgebiet, Runde, Termin,
                                        ↑Wettbewerbsbezeichnung)
Schueler(SchuelerID, Name, Vorname, Geburtsjahr, Klasse)
Betreuer(Kuerzel, Name, Vorname)
nimmtTeil(↑RundeID, ↑SchuelerID, Preis, ↑Kuerzel)
```

Durch die Aufteilung des Namens in Vorname und Name sind alle Attribute atomar. Damit sind die Bedingungen für die erste Normalform erfüllt.
Die zweite Normalform ist erfüllt, da alle Entitätstypen Primärschlüssel haben, die nur aus einem Attribut bestehen. In der Relation nimmtTeil hängen die Nichtschlüsselattribute nicht von einem Teilschlüssel ab.
Die dritte Normalform ist erfüllt, da keine transitiven Abhängigkeiten zwischen den Nichtschlüsselattributen existieren.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	erläutert die Informationen, die in der Datenbank stehen, im Sachzusammenhang.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (4) …… ……					
	Summe Teilaufgabe a)	**4**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die erste SQL-Anweisung und bestimmt die Informationen, die in der Datenbank gesucht werden.	5			
2	analysiert die zweite SQL-Anweisung und bestimmt die Informationen, die in der Datenbank gesucht werden.	5			
3	analysiert die dritte SQL-Anweisung und bestimmt die Informationen, die in der Datenbank gesucht werden.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (15) …… ……					
	Summe Teilaufgabe b)	**15**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt für den ersten Suchauftrag eine geeignete SQL-Anweisung.	5			
2	entwickelt für den zweiten Suchauftrag eine geeignete SQL-Anweisung.	5			
3	entwickelt für den dritten Suchauftrag eine geeignete SQL-Anweisung.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (15) …… ……					
	Summe Teilaufgabe c)	**15**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert, dass die Information nicht in das Datenbankschema aufgenommen werden kann.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (4) …… ……					
	Summe Teilaufgabe d)	**4**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt die Umsetzung des bisherigen Entwurfs in das Datenbankschema.	4			
2	entwickelt ein erweitertes ER-Modell und erläutert die Entscheidungen.	4			
3	überführt das neue ER-Modell in ein Datenbankschema, das in dritter Normalform ist und erläutert die Entscheidungen.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe e)	**12**			

	Summe insgesamt	**50**			

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

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

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

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

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

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

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

Name: _____

Abiturprüfung 2017

Informatik, Leistungskurs

Aufgabenstellung:

Seit einiger Zeit bieten Fernsehsender ihren Zuschauern an, bereits gesendete Beiträge nachträglich über das Internet anzuschauen. Zu jeder Sendung werden u. a. Angaben zur ursprünglichen Sendezeit abgespeichert.

Eine Servicekraft soll die Anfangs- und Endzeiten (Uhrzeiten) der Sendungen eingeben. Als Trennzeichen zwischen Stunden und Minuten wird der Doppelpunkt vereinbart. Die erste mögliche Zeitangabe eines Tages ist 00:00 Uhr, die letzte 23:59 Uhr. Da insgesamt sehr viele Sendungen zu erfassen sind, sollen Abkürzungen möglich sein: Nullen an der Zehnerstelle bei einstelligen Stunden- und Minutenangaben können weggelassen werden, d. h. neben „05:30“ bzw. „15:05“ sollen also auch die Eingaben „5:30“ bzw. „15:5“ zulässig sein.

Zur Überprüfung, ob es sich um eine syntaktisch zulässige Eingabe handelt, soll ein deterministischer endlicher Automat verwendet werden. Abbildung 1 zeigt den Übergangsgraphen des Automaten. Nicht dargestellte Übergänge führen in einen Fehlerzustand q_F.

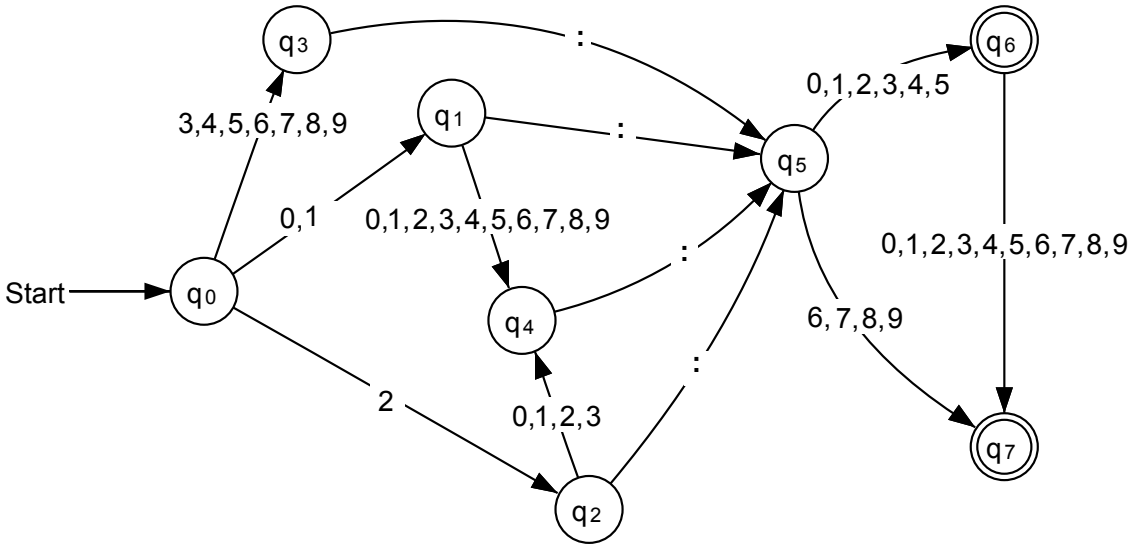


Abbildung 1: Übergangsgraph zum Überprüfen einer Zeitangabe auf Gültigkeit

Name: _____

a) Das Eingabealphabet des deterministischen endlichen Automaten zum Übergangsgraphen aus Abbildung 1 bestehe aus allen zehn Ziffern und dem Doppelpunktzeichen.

Geben Sie die Zustandsfolge des Automaten zum Übergangsgraphen in Abbildung 1 bei Eingabe der folgenden drei Wörter an:

`1:5` `12:73` `:41`

Erläutern Sie im Sachzusammenhang, warum die drei Wörter akzeptiert bzw. nicht akzeptiert werden.

Erläutern Sie die Bedeutung der Zustände q_4 *und* q_6 *im Sachzusammenhang.*

(12 Punkte)

b) Der abgebildete Automat enthält mehr Zustände, als erforderlich wären. Zwei der Zustände können zu einem „zusammengelegt“ werden, ohne die akzeptierte Sprache des Automaten zum Übergangsgraphen zu ändern.

Ermitteln Sie die beiden Zustände, die zusammengelegt werden können.

Stellen Sie dar, welche Änderungen nötig sind, um den überzähligen Zustand im Übergangsgraphen zu entfernen, ohne die akzeptierte Sprache zum Automaten zu ändern.

Hinweis: Sie können zur Darstellung der Änderungen den Übergangsgraphen in der Anlage verwenden.

(9 Punkte)

c) Mit der „Uhrzeit-Sprache“ bezeichnen wir die Menge aller Wörter, die syntaktisch korrekten Uhrzeiten entsprechen, die nach obigem Schema gebildet werden.

Entwickeln Sie eine linkslineare Grammatik, mit der die Wörter der „Uhrzeit-Sprache“ erzeugt werden können.

(7 Punkte)

Name: _____

d) Eine „vollständige Uhrzeitangabe“ bezeichnet im Folgenden eine Zeichenkette mit fünf Zeichen, wobei die ersten beiden Zeichen Ziffern sind, die eine Zahl zwischen 00 und 23 (als Stundenzahl) darstellen, danach als Trennzeichen der Doppelpunkt folgt und die letzten beiden Zeichen Ziffern sind, die eine Zahl zwischen 00 und 59 (als Minutenangabe) darstellen.

Die Methode `pruefeFormat` der im Anhang dokumentierten Klasse `Zeitformatcheck` mit dem Methodenkopf

```
public boolean pruefeFormat(String pWort)
```

soll prüfen, ob ein String eine vollständige Uhrzeitangabe darstellt.

Implementieren Sie die Methode `pruefeFormat`*, die* `true` *genau dann zurückgibt, wenn* `pWort` *eine „vollständige Uhrzeitangabe“ ist.*

(9 Punkte)

e) Gegeben ist die folgende Grammatik G, die auch Uhrzeiten erzeugt:

Startsymbol: `S`

Menge der Terminalsymbole: `T = {:, 0, 1, 2, 3, 4, 5}`

Menge der Nichtterminalsymbole: `N = {S, A, B, D}`

Menge der Produktionsregeln:

```
P = {
      S → 0A0 | 1A1 | 2B2,
      A → 0D0 | 1D1 | 2D2 | 3D3 | 4D4 | 5D5,
      B → 0D0 | 1D1 | 2D2 | 3D3,
      D → :
    }
```

Begründen Sie, warum die Grammatik G nicht regulär ist.

Ermitteln Sie die von der Grammatik G erzeugte Sprache.

Beurteilen Sie die Behauptung, dass es einen deterministischen endlichen Automaten mit maximal 4 Zuständen und dem Eingabealphabet `{:, 0, 1, 2, 3, 4, 5}` *gibt, der die von der Grammatik G erzeugte Sprache erkennt.*

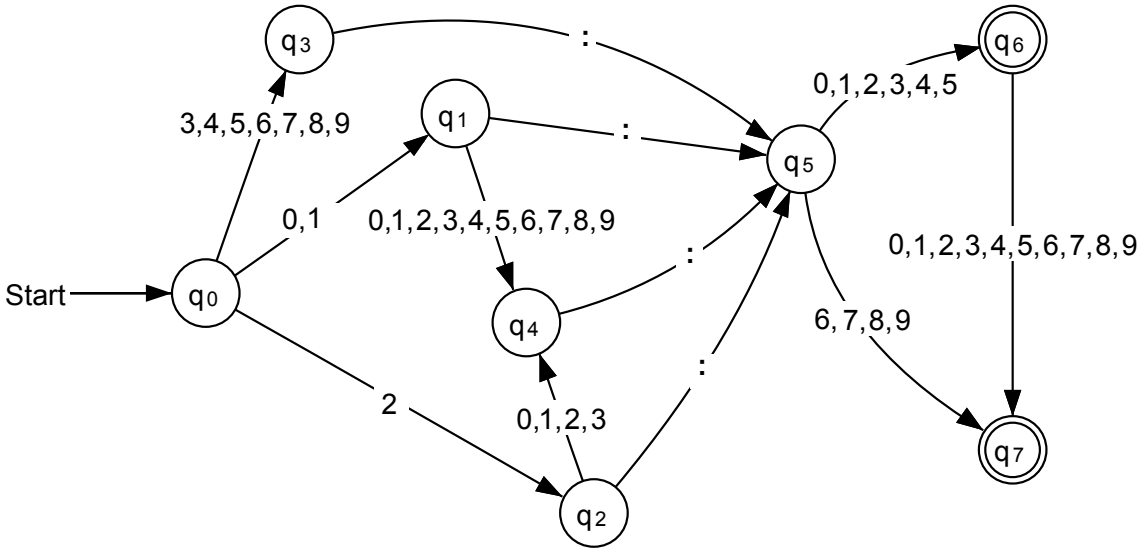
(13 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anlage: Übergangsgraph (zur Bearbeitung von Teilaufgabe b)



Name: _____

Anhang

Auszug aus der Dokumentation der Klasse `Zeitformatcheck`

Konstruktor **`Zeitformatcheck()`**

Anfrage **`boolean istZahl(String pWort)`**
Die Anfrage liefert den Wert `true`, wenn die Zeichenkette `pWort` eine ein- oder zweistellige Zahl darstellt, sonst wird `false` zurückgegeben. Die Methode darf für die Lösung der Teilaufgabe d) verwendet werden.

Anfrage **`boolean pruefeFormat(String pWort)`**
Die Anfrage liefert den Wert `true`, wenn die Zeichenkette `pWort` eine vollständige Uhrzeitangabe darstellt, sonst wird `false` zurückgegeben. Diese Methode soll in Teilaufgabe d) implementiert werden.

Dokumentation ausgewählter Methoden der Klasse `String`

Anfrage **`int length()`**
Diese Anfrage liefert die Länge des Strings.

Anfrage **`String substring(int pBegin, int pEnd)`**
Diese Anfrage liefert den Teilstring beginnend mit der übergebenen Position `pBegin` bis eine Position vor `pEnd`.

Dokumentation ausgewählter Methoden der Wrapperklasse `Integer`

Anfrage **`int parseInt(String s)`**
Die Anfrage liefert den Zahlenwert des Strings `s`, wenn `s` eine Ganzzahl darstellt.

Unterlagen für die Lehrkraft

Abiturprüfung 2017

Informatik, Leistungskurs

1. Aufgabenart

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

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2017

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - Java
   - Endliche Automaten und Kellerautomaten
   - Grammatiken regulärer und kontextfreier Sprachen
   - Scanner, Parser, und Interpreter für eine reguläre Sprache
   - Möglichkeiten und Grenzen von Automaten und formalen Sprachen

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

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

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

6. Modelllösungen

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

Teilaufgabe a)

Eingabefolge `1:5`

Zustandsfolge: $q_0 \rightarrow q_1 \rightarrow q_5 \rightarrow q_6$. Die Eingabe wird akzeptiert.

Die Eingabe entspricht der Uhrzeit 01:05 Uhr und ist eine abgekürzte gültige Eingabe.

Führende Nullen bei Stunden- und Minutenangaben dürfen weggelassen werden.

Eingabefolge `12:73`

Zustandsfolge: $q_0 \rightarrow q_1 \rightarrow q_4 \rightarrow q_5 \rightarrow q_7 \rightarrow q_F$. Die Eingabe wird nicht akzeptiert.

Da eine Stunde nur 60 Minuten hat, ist die Minutenangabe 73 ungültig.

Eingabefolge `:41`

Zustandsfolge: $q_0 \rightarrow q_F$. Die Eingabe wird nicht akzeptiert.

Die Stundenzahl ist ganz weggelassen worden. Es ist aber vereinbart worden, dass nur die Zehnerstellen weggelassen werden dürfen, die Einerstellen müssen angegeben werden.

Bedeutung des Zustands q_4:

Die bisher erfolgte Eingabe bildet eine Zahl für die Stunde (da noch kein Doppelpunkt gelesen wurde), wobei bislang zwei Ziffern eingelesen wurden: entweder erst eine `2` mit anschließender Ziffer `0` bis `3` (Stundenzahlen 20 bis 23) oder erst eine `0` oder `1` mit anschließender beliebiger Ziffer (Stundenzahlen 00 bis 19). Jetzt wird zwingend ein Doppelpunkt erwartet.

Bedeutung des Zustands q_6:

Bisher wurde (nach der Zahl für die Stunde und dem „:“-Trennzeichen) eine der Ziffern von `0` bis `5` eingelesen. Da diese einzelne Ziffer als verkürzte Schreibweise für eine Minutenangabe gelten kann, handelt es sich bereits um einen akzeptierenden Zustand. Auch jede folgende Ziffer kann noch akzeptiert werden. Dann wird die zuletzt gelesene Ziffer als Zehnerziffer der Minutenangabe interpretiert.

Teilaufgabe b)

„Zusammenlegbare Zustände“ erkennt man daran, dass man beim Lesen desselben Zeichens jeweils in denselben Folgezustand gelangt.

Vom Zustand q_3 und q_4 gelangt man beim Lesen der Eingabezeichen `0` bis `9` zum Zustand q_F und beim Lesen des Doppelpunkts zum Zustand q_5.

Die Zustände q_3 und q_4 können also zu einem einzigen Zustand verschmolzen werden.

Beispiel für notwendige Änderungen: Die Übergänge von q_0 zu q_3 werden als Übergänge von q_0 zum Zustand q_4 eingetragen. Der Zustand q_3 kann dann entfernt werden.

Teilaufgabe c)

Eine linkslineare Grammatik zur „Uhrzeit-Sprache“ ist zum Beispiel die folgende:

Startsymbol: `S;` `T = {:, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`

```
N = {S, A, B, C, D, E}
P = {
  S → A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 |
                  B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9,
  A → B0 | B1 | B2 | B3 | B4 | B5,
  B → C:,
  C → D0 | D1 | D2 | D3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
                E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 ,
  D → 2,
  E → 0 | 1
}
```

Teilaufgabe d)

```
public boolean pruefeFormat (String pWort) {
  boolean formatKorrekt = false;
  if (pWort.length() == 5) { // Uhrzeiten immer 5-stellig
    // String aufteilen
    String stdString = pWort.substring(0, 2);
    String trenner = pWort.substring(2, 3);
    String minString = pWort.substring (3, 5);
    // Sind Stunde und Minute Zahlen?
    if (istZahl(stdString) && istZahl(minString)) {
      int stdInt = Integer.parseInt(stdString);
      int minInt = Integer.parseInt(minString);
      // Werte der Stringteile pruefen
      if (stdInt >= 0 && stdInt < 24
        && trenner.equals(":")
        && minInt >= 0 && minInt < 60) {
          formatKorrekt = true;
      }
    }
  }
  return formatKorrekt;
}
```

Teilaufgabe e)

Die Grammatik ist nicht regulär, da z. B. die Produktion `S → 0A0` nicht regulär ist.

Die Grammatik erzeugt Wörter, die Uhrzeiten im genannten Format (zwei Stundenziffern, dann Doppelpunkt, dann zwei Minutenziffern) entsprechen, bei denen die Minuten rückwärts gelesen die Stundenzahl ergeben („Palindrome“).

Man muss zu dem Schluss kommen, dass es keinen solchen Automaten gibt.

Begründung: Der Automat benötigt außer dem Startzustand 3 Zustände, um das erste Zeichen zu speichern, denn das erste und letzte Zeichen jedes Wortes der Sprache müssen entweder beide `0`, beide `1` oder beide `2` sein. Das sind schon 4 Zustände. Weitere Zustände werden benötigt, um den Wert der zweiten Stelle zu speichern, denn das zweite Zeichen muss identisch zum vierten Zeichen sein. Das zeigt, dass ein deterministischer endlicher Automat zum Erkennen der Sprache mehr als 4 Zustände braucht.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	gibt die Zustandsfolge für alle drei Eingaben an.	3			
2	erläutert im Sachzusammenhang für jedes der drei Wörter, ob das Wort akzeptiert wird.	3			
3	erläutert die Bedeutung des Zustands q_4 im Sachzusammenhang.	3			
4	erläutert die Bedeutung des Zustands q_6 im Sachzusammenhang.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe a)	**12**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt die beiden Zustände.	7			
2	stellt die nötigen Änderungen dar.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe b)	**9**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine linkslineare Grammatik, mit der die Wörter der „Uhrzeit-Sprache“ erzeugt werden können.	7			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe c)	**7**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	implementiert die geforderte Methode.	9			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe d)	**9**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet, warum die Grammatik G nicht regulär ist.	2			
2	ermittelt die von der Grammatik G erzeugte Sprache.	5			
3	beurteilt die genannte Behauptung.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (13) …… ……					
	Summe Teilaufgabe e)	**13**			

	Summe insgesamt	**50**			

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

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

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

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

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

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

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