Name: _____

Abiturprüfung 2020

Informatik, Leistungskurs

Aufgabenstellung:

Ein Eisenbahnunternehmen möchte mit einer Software die Sitzplatzreservierungen für die Waggons von Zügen verwalten.

Die Sitzplätze eines Waggons sind fortlaufend nummeriert, beginnend mit der Nummer 0. Alle Sitzplätze sind in Vierer-Sitzgruppen angeordnet; jeweils vier Plätze mit fortlaufenden Platznummern gehören zu einer Sitzgruppe.

In Abbildung 1 wird ein Ausschnitt eines Zuges mit zwei Waggons dargestellt. „Waggon 15“ hat 16 Sitzplätze, „Waggon 21“ hat 24 Sitzplätze. Sitzgruppen sind mit einem Rahmen dargestellt: z. B. die Plätze 12, 13, 14 und 15 bilden in beiden Waggons jeweils eine Sitzgruppe. Reservierte Plätze sind grau markiert.

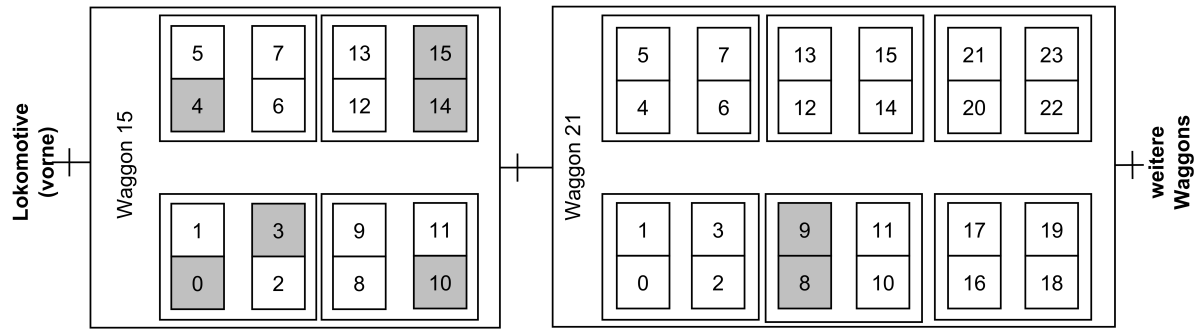


Abbildung 1: Zwei Waggons eines Beispielzuges mit Sitzplatznummern

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

Name: _____

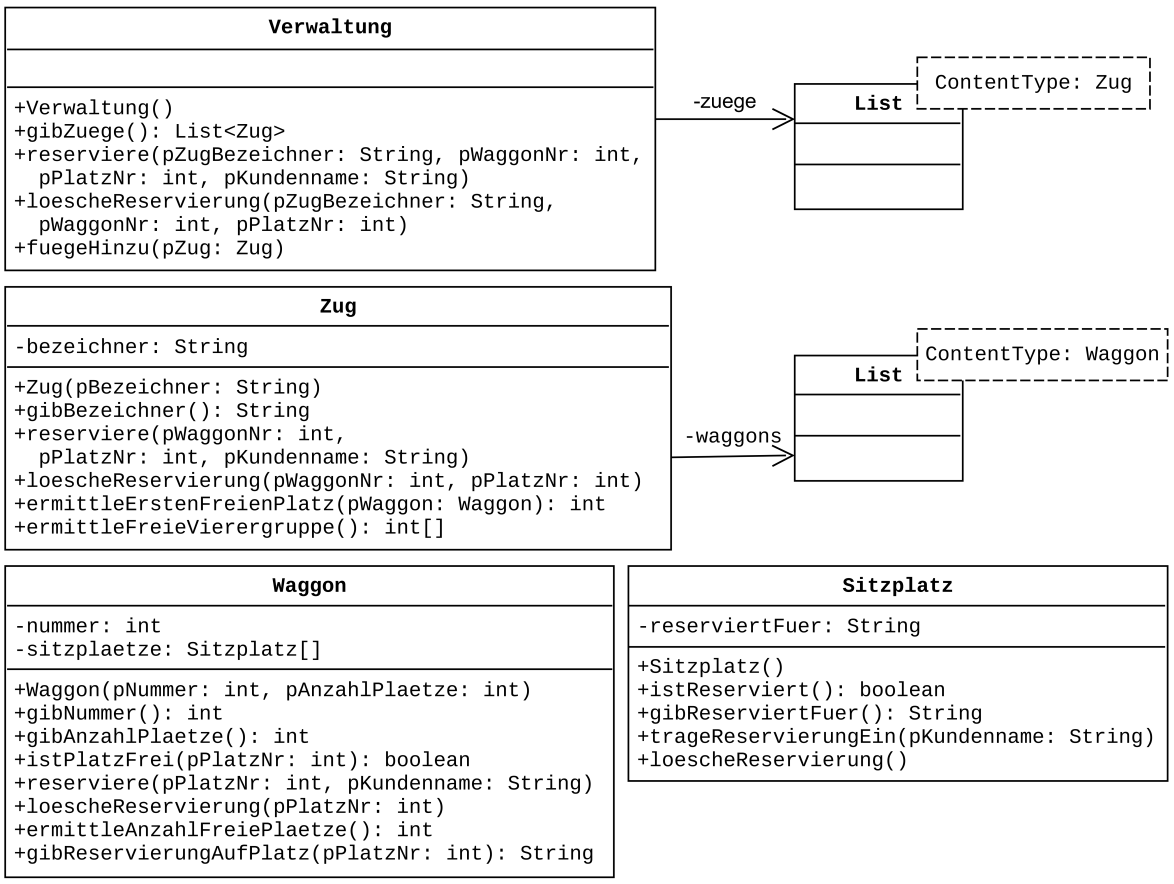


Abbildung 2: Teilmodellierung als Implementationsdiagramm

a) *Erläutern Sie mit Bezug auf Abbildung 2 die Beziehungen zwischen den Klassen* `Verwaltung`, `Zug`, `Waggon` *und* `Sitzplatz`.

Erläutern Sie im Sachkontext, welche Argumente dafür sprechen, die Datensammlung für die Züge als lineare Liste und die Datensammlung für die Sitzplätze als Feld (Array) zu realisieren.

(7 Punkte)

In einem Objekt der Klasse `Waggon` hat das Feld `sitzplaetze` so viele Elemente wie durch den Parameter des Konstruktors `pAnzahlPlaetze` angegeben wird. Die Sitzplatznummer entspricht dem Index im Feld `sitzplaetze`. Wenn es eine Reservierung für einen Sitzplatz gibt, so wird der Kundenname im entsprechenden Objekt der Klasse `Sitzplatz` eingetragen.

Name: _____

b) Die Klasse Zug erhält zu Optimierungszwecken eine zusätzliche Methode:

```
 1 public void wasMacheIch(Waggon pWaggon) {
 2   waggons.toFirst();
 3   for (int j = 0; j < pWaggon.gibAnzahlPlaetze(); j++) {
 4     if (!pWaggon.istPlatzFrei(j)){
 5       while (waggons.hasAccess()
 6           && (pWaggon == waggons.getContent()
 7               || waggons.getContent().
 8                   ermittleAnzahlFreiePlaetze() == 0)) {
 9         waggons.next();
10       }
11       if (waggons.hasAccess()) {
12         Waggon aktW = waggons.getContent();
13         int i = ermittleErstenFreienPlatz(aktW);
14         aktW.reserviere(i,
15             pWaggon.gibReservierungAufPlatz(j));
16         pWaggon.loescheReservierung(j);
17       }
18     }
19   }
20 }
```

Ermitteln Sie im Sachzusammenhang, inwieweit sich der in Abbildung 1 dargestellte Zug durch den Aufruf der Methode `wasMacheIch` *mit dem in der Abbildung 1 als „Waggon 21“ dargestellten Waggonobjekt als Parameter verändert.*

Erläutern Sie die Funktionsweise der Methode `wasMacheIch` *für die Zeilen 5 bis 10 und für die Zeilen 11 bis 17.*

Erläutern Sie die Aufgabe der Methode `wasMacheIch` *im Sachzusammenhang.*

(12 Punkte)

Name: _____

c) Um Gruppen von Reisenden zu ermöglichen, nahe beieinander zu sitzen, soll es möglich sein, Vierer-Sitzgruppen zu reservieren. Die Methode `ermittleFreieVierergruppe` der Klasse `Zug` soll einen Waggon des Zuges suchen, in dem es eine Vierer-Sitzgruppe gibt, bei der alle vier Plätze noch nicht reserviert sind. Als Rückgabe soll sie ein Feld (Array) aus zwei Zahlen liefern: die Nummer des gefundenen Waggons und die kleinste Platznummer in der freien Vierer-Sitzgruppe.

Für den Zug in Abbildung 1 würde im „Waggon 15“ keine freie Vierer-Sitzgruppe gefunden. Im „Waggon 21“ würde die Vierer-Sitzgruppe mit den Plätzen Nr. 0 bis 3 gefunden und somit die beiden Zahlen 21 (für Waggon 21) und 0 (für die kleinste Platznummer der Vierer-Sitzgruppe) im Feld `[21, 0]` zurückgegeben.

Wenn es in keinem der Waggons eine freie Vierer-Sitzgruppe gibt, so soll die Methode zweimal den Wert `-1` im Feld `[-1, -1]` zurückgeben.

Entwickeln und erläutern Sie ein algorithmisches Verfahren für die Methode.

Implementieren Sie die Methode mit dem folgenden Methodenkopf:

```
public int[] ermittleFreieVierergruppe()
```

(15 Punkte)

d) Das Modell soll um folgende Anforderungen erweitert werden:

> In Zukunft soll die Verwaltung nicht nur Reservierungen vornehmen, sondern auch noch Kunden verwalten können. In der bisherigen Modellierung wurden die Kunden nur über die Namen identifiziert. Nun sollen zu den Kunden nicht nur die Namen, sondern auch die Vornamen und eine Kundennummer gespeichert werden. Die Verwaltung soll Zugriff auf alle Kundendaten haben, um den Kundenstamm pflegen zu können. Die Verwaltung soll alle Kundendaten auf Anfrage zurückliefern können.

Modellieren Sie die oben genannten Anforderungen als Erweiterung der Modellierung aus Abbildung 2 und stellen Sie dabei nur die Veränderungen und Erweiterungen in Form eines Implementationsdiagramms dar.

Erläutern Sie zu allen Klassen, ob und welche Veränderungen durchgeführt werden müssen.

(12 Punkte)

Name: _____

e) Der Praktikant Thomas schlägt eine alternative Modellierung vor:

> „Jedes Sitzplatzobjekt hat eine Referenz auf sein Waggonobjekt, und jedes Waggonobjekt hat eine Referenz auf sein Zugobjekt, die Referenz auf das Feld aus Sitzplätzen entfällt. Dafür verwaltet die Klasse `Verwaltung` außer der Zugliste auch alle Sitzplatzobjekte in einer Liste."

Das findet die Praktikantin Johanna nicht sehr praktisch. Sie behauptet, dass man dann deutlich mehr Arbeitsschritte benötigt, um z. B. die Anzahl der nicht reservierten Plätze in einem bestimmten Waggon zu ermitteln.

Beurteilen Sie den alternativen Modellierungsvorschlag von Thomas im Hinblick auf Johannas Behauptung.

(4 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse `Sitzplatz`

Objekte der Klasse **`Sitzplatz`** verwalten je einen Sitzplatz in einem Zugwaggon. Sitzplätze können z. B. reserviert werden.

Auszug aus der Dokumentation der Klasse `Sitzplatz`

Konstruktor **`Sitzplatz()`**
Ein Sitzplatzobjekt wird initialisiert. Das Sitzplatzobjekt ist zunächst nicht reserviert.

Anfrage **`boolean istReserviert()`**
Die Anfrage liefert `true`, wenn der Sitzplatz reserviert ist, ansonsten liefert sie `false`.

Anfrage **`String gibReserviertFuer()`**
Die Anfrage liefert den Namen der Person, die diesen Sitzplatz reserviert hat. Gibt es keine Reservierung, wird `null` zurückgegeben.

Auftrag **`void trageReservierungEin(String pKundenname)`**
Der Auftrag bewirkt, dass dieser Sitzplatz für die Person mit dem Namen `pKundenname` reserviert ist. Ist bereits eine Reservierung eingetragen, so passiert nichts.

Auftrag **`void loescheReservierung()`**
Der Auftrag löscht eine ggf. vorhandene Reservierung für diesen Sitzplatz.

Name: _____

Die Klasse `Waggon`

Objekte der Klasse **`Waggon`** verwalten die Sitzplätze eines Waggons. Waggons haben immer eine durch vier teilbare Anzahl an Sitzplätzen. Die Sitzplätze sind beginnend mit 0 nummeriert.

Auszug aus der Dokumentation der Klasse `Waggon`

Konstruktor **`Waggon(int pNummer, int pAnzahlPlaetze)`**
Ein Waggonobjekt mit der Nummer `pNummer` und mit `pAnzahlPlaetze` Sitzplätzen wird initialisiert. Zu Anfang ist kein Sitzplatz reserviert.

Anfrage **`int gibNummer()`**
Die Anfrage liefert die Nummer des Waggons.

Anfrage **`int gibAnzahlPlaetze()`**
Die Anfrage liefert die Anzahl der Sitzplätze des Waggons.

Anfrage **`boolean istPlatzFrei(int pPlatzNr)`**
Die Anfrage liefert `true`, wenn es den Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt und er nicht reserviert ist, ansonsten liefert sie `false`.

Auftrag **`void reserviere(int pPlatzNr, String pKundenname)`**
Der Auftrag trägt eine Reservierung für die Person mit dem Namen `pKundenname` auf den Sitzplatz mit der Nummer `pPlatzNr` ein. Wenn es die gewünschte Sitzplatznummer im Waggon nicht gibt oder der Platz bereits reserviert ist, so passiert nichts.

Auftrag **`void loescheReservierung(int pPlatzNr)`**
Der Auftrag löscht die Reservierung auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es in diesem Waggon keinen Platz mit der Platznummer `pPlatzNr` gibt, so passiert nichts.

Anfrage **`int ermittleAnzahlFreiePlaetze()`**
Die Anfrage liefert die Anzahl der Sitzplätze ohne Reservierung.

Anfrage **`String gibReservierungAufPlatz(int pPlatzNr)`**
Die Anfrage liefert den Kundennamen, der auf dem Platz mit der Nummer `pPlatzNr` eingetragen ist. Wenn es im diesem Waggon keinen Platz mit der Platznummer `pPlatzNr` gibt oder der Platz nicht reserviert ist, wird `null` zurückgegeben.

Name: _____

Die Klasse Zug

Ein Objekt der Klasse **`Zug`** verwaltet seine Waggons.

Auszug aus der Dokumentation der Klasse Zug

Konstruktor **`Zug(String pBezeichner)`**
Ein Zugobjekt mit der Bezeichnung `pBezeichner` wird initialisiert.
Das Zugobjekt erzeugt und verwaltet eine Liste von Waggons.

Anfrage **`String gibBezeichner()`**
Die Anfrage liefert die Bezeichnung des Zugs.

Auftrag **`void reserviere(int pWaggonNr, int pPlatzNr, String pKundenname)`**
Der Auftrag trägt eine Reservierung für den Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr` für die Person mit dem Namen `pKundenname` ein. Wenn es im Zug keinen Waggon mit der angegebenen Waggonnummer gibt oder die Platznummer im Waggon nicht vorhanden ist oder der Platz bereits reserviert war, so passiert nichts.

Auftrag **`void loescheReservierung(int pWaggonNr, int pPlatzNr)`**
Der Auftrag löscht die Reservierung im Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es im Zug keinen Waggon mit der angegebenen Waggonnummer gibt oder die Platznummer im Waggon nicht vorhanden ist, so passiert nichts.

Anfrage **`int ermittleErstenFreienPlatz(Waggon pWaggon)`**
Die Anfrage ermittelt die Platznummer des ersten freien Sitzplatzes im Waggon `pWaggon` und gibt sie zurück. Gibt es keinen freien Sitzplatz, so wird `-1` zurückgegeben.

Anfrage **`int[] ermittleFreieVierergruppe()`**
Die Anfrage ist in Teilaufgabe c) zu implementieren.

Name: _____

Die Klasse `Verwaltung`

Ein Objekt der Klasse **`Verwaltung`** verwaltet Züge.

Auszug aus der Dokumentation der Klasse `Verwaltung`

Konstruktor **`Verwaltung()`**
Ein Verwaltungsobjekt wird initialisiert.

Anfrage **`List<Zug> gibZuege()`**
Die Anfrage liefert eine Liste aller Züge.

Auftrag
```
void reserviere(String pZugBezeichner,
      int pWaggonNr, int pPlatzNr, String pKundenname)
```
Der Auftrag trägt eine Reservierung für die Person mit dem Namen `pKundenname` auf den Sitzplatz mit der Nummer `pPlatzNr` im Waggon mit der Nummer `pWaggonNr` im Zug mit der Bezeichnung `pZugBezeichner` ein. Wenn es den Zug mit der genannten Bezeichnung nicht gibt oder der Zug keinen Waggon mit der Nummer `pWaggonNr` besitzt oder es keinen Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt oder der Platz bereits reserviert war, so passiert nichts.

Auftrag
```
void loescheReservierung(String pZugBezeichner,
                         int pWaggonNr, int pPlatzNr)
```
Der Auftrag löscht die Reservierung im Zug mit der Bezeichnung `pZugBezeichner` im Waggon mit der Nummer `pWaggonNr` auf dem Sitzplatz mit der Nummer `pPlatzNr`. Wenn es den Zug mit der genannten Bezeichnung nicht gibt oder der Zug keinen Waggon mit der Nummer `pWaggonNr` besitzt oder es keinen Sitzplatz mit der Nummer `pPlatzNr` im Waggon gibt, so passiert nichts.

Auftrag **`void fuegeHinzu(Zug pZug)`**
Der Auftrag hängt den Zug `pZug` hinten an die Liste der Züge an.

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 2020

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 2020

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

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

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

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

2. *Medien/Materialien*
   - entfällt

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

5. Zugelassene Hilfsmittel

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

6. Modelllösungen

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

Teilaufgabe a)

Ein Objekt der Klasse `Verwaltung` verwaltet seine Züge in einer linearen Liste mit Objekten der Klasse `Zug`. Diese Züge verwalten ihre Waggons jeweils in einer weiteren linearen Liste, welche Objekte vom Typ `Waggon` enthält. Jedes Objekt der Klasse `Waggon` hat ein Feld von Objekten der Klasse `Sitzplatz`, in dem alle Sitzplatzobjekte gespeichert werden, die zu Sitzplätzen dieses Waggons gehören.

Für die Verwaltung der Züge wird eine Liste `zuege` gewählt, weil jederzeit Züge hinzugefügt werden können, die Anzahl der Züge unbekannt ist und die Datensammlung daher dynamisch sein sollte. Die Verwaltung der Sitzplätze in der Klasse `Waggon` ist hingegen als Feld modelliert, weil sich die Anzahl der Sitzplätze eines Waggons zur Laufzeit nicht ändert und man auf die Sitzplätze über die Sitzplatznummern direkt per Index zugreifen kann.

Teilaufgabe b)

Der Sitzplatz Nummer 1 im „Waggon 15“ wird reserviert mit dem Kundennamen, der zuvor im „Waggon 21“ auf Platz Nummer 8 als Reservierung eingetragen war. Der Sitzplatz Nummer 2 im „Waggon 15“ wird reserviert mit dem Kundennamen, der zuvor im „Waggon 21“ auf Platz Nummer 9 eingetragen war. Die Plätze 8 und 9 im „Waggon 21“ sind nach dem Methodenaufruf wieder frei.

Zeilen 5 bis 10: In der Waggonliste des Zuges wird nach einem Waggon gesucht, der nicht der Waggon `pWaggon` ist und der noch freie Plätze hat.

Zeilen 11 bis 17: Wenn ein solcher Waggon gefunden werden konnte, wird die Platznummer des ersten freien Sitzplatzes dieses Waggons in der lokalen Variablen gespeichert (Zeile 13). Der Kundenname, der in `pWaggon` als Reservierung eingetragen war, wird auf dem zuvor gespeicherten Sitzplatz eingetragen (Zeilen 14 - 15). Die Reservierung in `pWaggon` wird anschließend gelöscht (Zeile 16).

Aufgabe der Methode `wasMacheIch`: Die Methode bucht so viele Reservierungen wie möglich aus dem Waggon `pWaggon` auf andere Waggons dieses Zuges um. Dabei werden der Reihe nach alle freien Plätze in den Waggons des Zuges von vorne bis hinten gefüllt.

Teilaufgabe c)

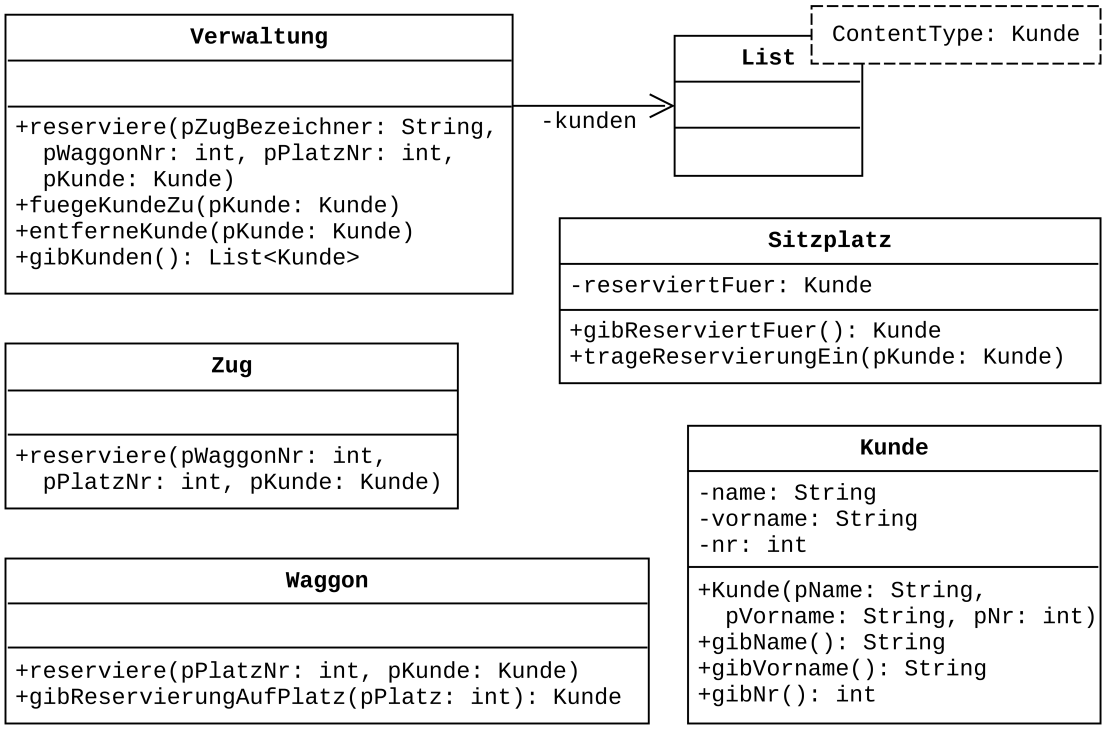
Ein algorithmisches Verfahren:
Man muss die Waggonliste des Zuges systematisch durchsuchen. Für jeden Waggon muss man, mit dem Sitzplatz `0` beginnend, in einer Schleife jeden vierten Sitzplatz untersuchen. Für jeden der untersuchten Sitzplätze ist zu prüfen, ob dieser Sitzplatz und die drei folgenden Sitzplätze nicht reserviert sind. Sind alle vier Sitzplätze noch frei, so hat man eine freie Vierergruppe gefunden und speichert im Rückgabefeld die Waggonnummer und die Nummer des ersten Sitzplatzes der Vierergruppe. Ansonsten muss man – mit einem Viererschritt – den ersten Sitzplatz der nächsten Vierer-Sitzgruppe analog untersuchen. Sobald die Sitzplatznummern der zu untersuchenden Sitzplätze größer sind als die Platzanzahl im Waggon, kann man die Suche abbrechen. Dann gibt es keine freie Vierer-Sitzgruppe in diesem Waggon, und man muss den nächsten Waggon der Waggonliste analog untersuchen. Wenn in der ganzen Waggonliste keine freie Vierer-Sitzgruppe gefunden wurde, so wird das mit dem Inhalt `[-1, -1]` initialisierte Feld zurückgegeben. Ansonsten wird das mit der Waggonnummer und der Sitzplatznummer der gefundenen Vierer-Sitzgruppe gefüllte Feld zurückgegeben.

Implementierte Methode:

```
public int[] ermittleFreieVierergruppe() {
  boolean gefunden = false;
  int[] rueckgabe = new int[2];
  rueckgabe[0] = -1;
  rueckgabe[1] = -1;
  waggons.toFirst();
  while (waggons.hasAccess() && !gefunden) {
    int platznummer = 0;
    Waggon aktWaggon = waggons.getContent();
    while (platznummer <= aktWaggon.gibAnzahlPlaetze() - 4
           && !gefunden) {
      if (aktWaggon.istPlatzFrei(platznummer)
          && aktWaggon.istPlatzFrei(platznummer + 1)
          && aktWaggon.istPlatzFrei(platznummer + 2)
          && aktWaggon.istPlatzFrei(platznummer + 3)) {
        gefunden = true;
        rueckgabe[0] = aktWaggon.gibNummer();
        rueckgabe[1] = platznummer;
      }
      platznummer = platznummer + 4;
    }
    waggons.next();
  }
  return rueckgabe;
}
```

Teilaufgabe d)

Nur die Änderungen sind im folgenden Implementationsdiagramm notiert:



Für die Erweiterung muss man eine Klasse `Kunde` modellieren mit Attributen vom Typ `String` für den Namen und den Vornamen und einem ganzzahligen Attribut für die Kundennummer. Die Klasse erhält einen Konstruktor mit den beiden Zeichenketten für Vor- und Nachname und einer ganzzahligen Nummer als Parametern zum Initialisieren eines neuen Kundenobjektes.

Die Reservierungen auf den Sitzplätzen werden mit Kundenobjektreferenzen eingetragen und verwaltet (statt mit den Kundennamen).

In den Klassen `Waggon`, `Zug` und `Verwaltung` sollen Reservierungen vorgenommen werden: Die Parameterlisten für die entsprechenden Methoden `reserviere` müssen dafür ebenfalls von Kundennamen zu Kundenobjektreferenzen verändert werden.

In der Klasse `Waggon` muss auch die Anfrage `gibReservierungAufPlatz` so geändert werden, dass statt einer Zeichenkette eine Objektreferenz zurückgegeben wird.

Die Klasse `Verwaltung` soll laut Anforderung alle Kunden verwalten können. Da der Kundenstamm sich zur Laufzeit ändern kann, wird dafür die dynamische Datenstruktur der linearen Liste gewählt. Die Klasse `Verwaltung` erhält zusätzlich eine Methode, um eine Liste aller Kunden zurückzugeben.

Teilaufgabe e)

Bei der alternativen Modellierung muss man alle Sitzplätze aller Züge, also die gesamte Sitzplatzliste der Verwaltung, durchsuchen, um die Sitzplätze herauszufiltern, die im gewünschten Waggon liegen und die keine Reservierung haben.

Da die Summe der Sitzplätze in allen Waggons größer ist als die Anzahl der Sitzplätze in einem Waggon, ist der Aufwand bei der alternativen Modellierung größer. Daher ist die alternative Modellierung im Hinblick auf das genannte Problem abzulehnen.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	erläutert die Beziehungen zwischen den Klassen.	3			
2	erläutert im Sachkontext, welche Argumente dafür sprechen, die Datensammlung für die Züge als Liste und die Datensammlung für die Sitzplätze als Feld zu realisieren.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe a)	**7**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	ermittelt im Sachzusammenhang, inwieweit sich der dargestellte Zug durch den Aufruf der Methode mit dem in Abbildung 1 als „Waggon 21“ dargestellten Waggonobjekt als Parameter verändert.	3			
2	erläutert die Funktionsweise für die Zeilen 5 bis 10 und für die Zeilen 11 bis 17.	6			
3	erläutert die Aufgabe der Methode im Sachzusammenhang.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe b)	**12**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt und erläutert ein algorithmisches Verfahren für die Methode.	6			
2	implementiert die Methode.	9			
Sachlich richtige Lösungsalternative zur Modelllösung: (15) …… ……					
	Summe Teilaufgabe c)	**15**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	modelliert die genannten Anforderungen als Erweiterung der Modellierung und stellt nur die Veränderungen und Erweiterungen als Implementationsdiagramm dar.	6			
2	erläutert zu allen Klassen, ob und welche Veränderungen durchgeführt werden müssen.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt den alternativen Modellierungsvorschlag im Hinblick auf die Behauptung.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (4) …… ……					
	Summe Teilaufgabe e)	**4**			

	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 2020

Informatik, Leistungskurs

Aufgabenstellung:

Informatikstudentin Charlotte Jabelonski arbeitet regelmäßig als „DJ Charlski“. Für DJs ist es eine Herausforderung, die einzelnen Musikstücke aus ihrer Sammlung jeden Abend neu so anzuordnen, dass sie passend aufeinander folgend abgespielt werden.

DJ Charlski möchte eine Software für DJs entwickeln, die dabei hilft, schnell passende Nachfolgestücke zu finden. Dazu werden alle Musikstücke in einem Graphen gespeichert. Eine Kante des Graphen entspricht einem direkten *Übergang* von einem zu einem anderen Musikstück und bestimmt durch ihr Kantengewicht die *Güte* dieses Übergangs.

- Güte 3: optimaler Übergang
- Güte 2: guter Übergang
- Güte 1: schlechter Übergang (d. h. nur in Ausnahmefällen zu benutzen)

Wenn Musikstücke gar nicht durch eine Kante verbunden sind, dürfen sie nie direkt hintereinander gespielt werden. Zusätzlich gilt, dass jedes Musikstück an einem Abend nur einmal gespielt werden darf.

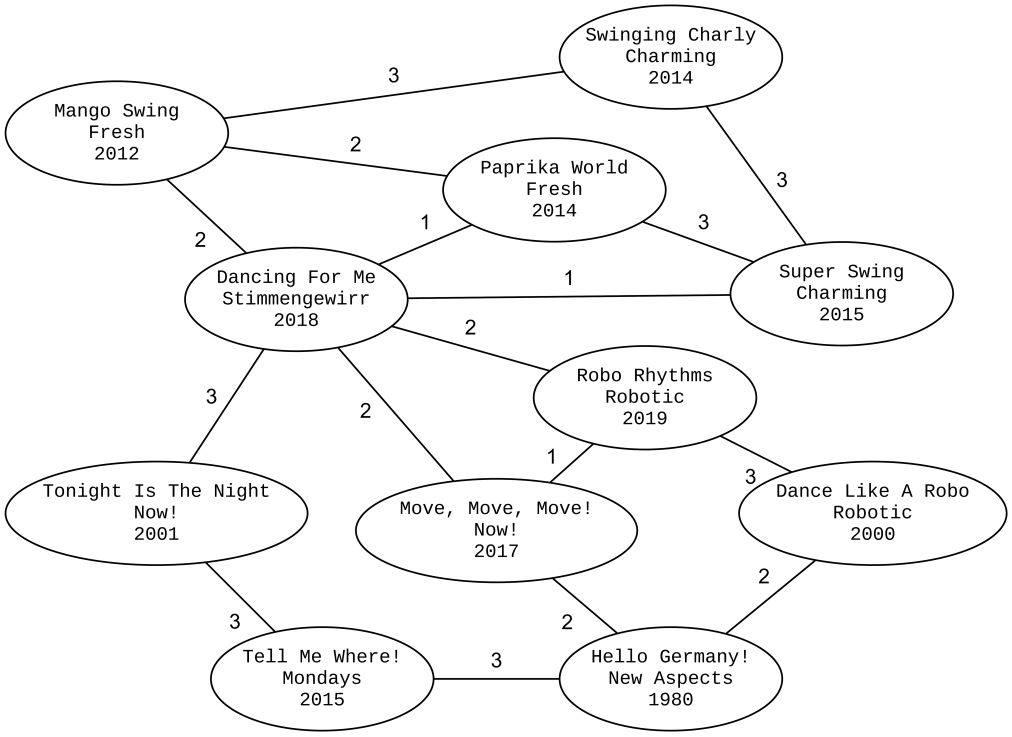


Abbildung 1: Musiksammlung als Graph

Name: _____

Abbildung 1 zeigt die Musiksammlung von DJ Charlski als Graph. Die Knoten des Graphen entsprechen jeweils einem Musikstück der Musiksammlung und beinhalten von oben nach unten den Titel, den Interpreten und das Erscheinungsjahr des Musikstücks. Aus Gründen der Vereinfachung kann davon ausgegangen werden, dass jedes Musikstück im Graphen durch seinen Titel eindeutig identifizierbar ist.

Mithilfe des Graphen kann DJ Charlski nun auch *Übergangspfade* zwischen zwei beliebigen Musikstücken herausfinden. Ein Übergangspfad zwischen zwei Musikstücken entspricht dabei einem Pfad innerhalb des Graphen, der die beiden Musikstücke miteinander verbindet. Dabei bestimmt das kleinste Kantengewicht des Pfades die Güte des Übergangspfades. Besteht ein Pfad z. B. aus drei Kanten mit den Gewichten 2, 3 und 3, so handelt es sich um einen guten Übergangspfad mit der Güte 2.

a) DJ Charlski sucht nach einem optimalen Übergangspfad (Güte 3) vom Titel `"Mango Swing"` zu `"Paprika World"`, wobei jeder Titel an einem Abend höchstens einmal abgespielt werden darf.

Geben Sie einen optimalen Übergangspfad (Güte 3) zwischen den Musikstücken mit den Titeln `"Mango Swing"` *und* `"Paprika World"` *an.*

Erläutern Sie im Sachzusammenhang, welche Auswirkungen es hat, wenn der Titel `"Dancing For Me"` *an einem Abend als erstes abgespielt wird.*

(7 Punkte)

Zur Modellierung eines Prototyps ihrer DJ-Software hat DJ Charlski ein Implementationsdiagramm erstellt, von dem ein Ausschnitt in Abbildung 2 dargestellt ist.

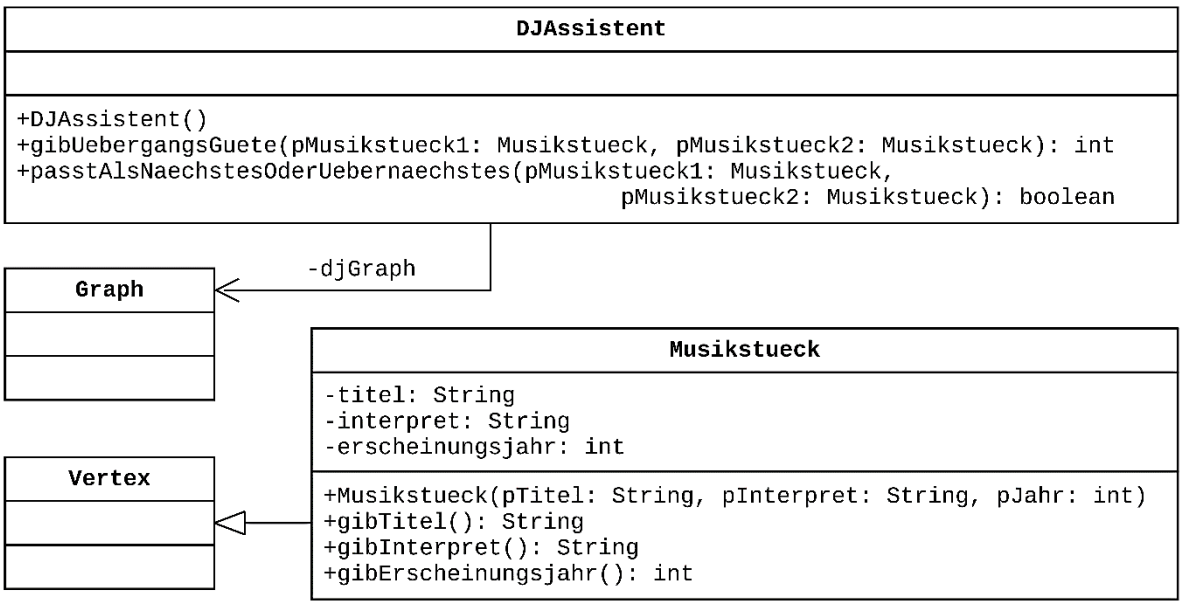


Abbildung 2: Teilmodellierung der DJ-Software in Form eines Implementationsdiagramms

Name: _____

b) *Erläutern Sie die Beziehungen zwischen den Klassen* `DJAssistent`, `Musikstueck`, `Graph` *und* `Vertex`.

Begründen Sie, dass es nicht sinnvoll ist, die Klasse `DJAssistent` *als Unterklasse der Klasse* `Graph` *zu modellieren.*

(8 Punkte)

c) DJ Charlski wird bei ihrer Arbeit häufig gefragt, ob sie als nächstes oder übernächstes ein bestimmtes Musikstück spielen kann. Wenn es einen mindestens guten Übergangspfad (Güte 2 oder 3) vom aktuellen zum gewünschten Musikstück gibt, kommt sie diesen Wünschen gerne nach.

Um solche Fragen schneller beantworten zu können, hat DJ Charlski bereits eine Methode `gibUebergangsGuete` implementiert, welche die Güte des direkten Übergangs zwischen zwei `Musikstueck`-Objekten als Wert zwischen 1 und 3 zurückgibt. Außerdem plant sie eine weitere Methode `passtAlsNaechstesOderUebernaechstes` zu implementieren. Die Dokumentationen beider Methoden befinden sich im Anhang.

Im Folgenden referenziere die Variable `musikstueck1` das `Musikstueck`-Objekt zum Titel `"Dancing For Me"` und die Variable `musikstueck2` das Objekt zum Titel `"Paprika World"` aus dem in Abbildung 1 dargestellten Beispielgraphen.

Geben Sie an, welche Werte folgende Methodenaufrufe für den in Abbildung 1 gegebenen Beispielgraphen zurückgeben:

```
gibUebergangsGuete(musikstueck1, musikstueck2)
```

```
passtAlsNaechstesOderUebernaechstes(musikstueck1, musikstueck2)
```

Implementieren Sie nur die Methode `passtAlsNaechstesOderUebernaechstes`.

Hinweis: Die Methode `gibUebergangsGuete` darf genutzt werden, sie muss aber nicht implementiert werden.

(11 Punkte)

Name: _____

d) DJ Charlski hat die Methode `wasLiefereIch` der Klasse `DJAssistent` implementiert:

```
 1 public boolean wasLiefereIch(Musikstueck pMusikstueck1,
 2                               Musikstueck pMusikstueck2) {
 3   Queue<Vertex> schlange = new Queue<Vertex>();
 4   djGraph.setAllVertexMarks(false);
 5   pMusikstueck1.setMark(true);
 6   schlange.enqueue(pMusikstueck1);
 7   while (!schlange.isEmpty()
 8                       && schlange.front() != pMusikstueck2) {
 9     Vertex a = schlange.front();
10     List<Vertex> liste = djGraph.getNeighbours(a);
11     liste.toFirst();
12     while (liste.hasAccess()) {
13       Vertex b = liste.getContent();
14       Edge e = djGraph.getEdge(a, b);
15       if (e.getWeight() == 3 && !b.isMarked()) {
16         b.setMark(true);
17         schlange.enqueue(b);
18       }
19       liste.next();
20     }
21     schlange.dequeue();
22   }
23   return (!schlange.isEmpty()
24                        && schlange.front() == pMusikstueck2);
25 }
```

Die Methode `wasLiefereIch` werde mit dem in Abbildung 1 dargestellten Graphen und mit den Parametern `musikstueck1` und `musikstueck2` aufgerufen. Dabei referenziere `musikstueck1` das `Musikstueck`-Objekt zum Titel `"Dancing For Me"` und `musikstueck2` das Objekt zum Titel `"Hello Germany!"` aus dem in Abbildung 1 dargestellten Beispielgraphen.

Analysieren Sie die Methode `wasLiefereIch` *und erläutern Sie den Quelltext unter Bezugnahme auf die Veränderungen der Schlange bei Anwendung der Methode auf die Beispieldaten.*

Ermitteln Sie den Rückgabewert der Methode bei Anwendung auf die Beispieldaten.

Erläutern Sie die Bedeutung der Knotenmarkierungen (Quelltextzeilen 5, 15 und 16) im Hinblick auf die Funktionsweise des Algorithmus.

Erläutern Sie die Funktionalität der Methode `wasLiefereIch` *im Sachzusammenhang.*

(12 Punkte)

Name: _____

e) DJ Charlski stellt die Software einem Kollegen vor. Dieser ist begeistert, allerdings merkt er an, dass die Verwaltung der Daten bei einem sehr großen Datenbestand eventuell umständlich wäre. Er hat gehört, dass sich dafür die Verwendung einer Datenbank besser eignet.

Entwickeln Sie eine Datenbankmodellierung in Form eines Entity-Relationship-Diagramms, welche die Speicherung aller Musikstücke und deren Beziehungen zueinander ermöglicht.

Erläutern Sie, wie Ihre Modellierung die Speicherung aller Musikstücke und deren Beziehungen zueinander ermöglicht.

Entwickeln Sie eine Idee, wie eine Datenbankabfrage aussehen könnte, die auf Grundlage Ihrer Datenbankmodellierung dieselbe Information wie die Methode `passtAlsNaechstesOderUebernaechstes` *ermittelt.*

Hinweis: Die Datenbankabfrage selbst muss nicht angegeben werden.

(12 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anhang

Die Klasse `Musikstueck`

Diese Klasse ist eine Unterklasse der Klasse `Vertex` (siehe weitere Dokumentationen) und stellt zusätzlich zu den geerbten Methoden die folgenden Methoden zur Verfügung:

Auszug aus der Dokumentation der Klasse `Musikstueck`

Konstruktor **`Musikstueck(String pTitel, String pInterpret, int pJahr)`**
Ein Musikstück mit dem Titel `pTitel`, dem Interpreten `pInterpret` und dem Erscheinungsjahr `pJahr` wird initialisiert. Da der Titel eines Musikstücks dieses eindeutig identifiziert, wird der ID-Eintrag der Oberklasse `Vertex` auf `pTitel` gesetzt.

Anfrage **`String gibTitel()`**
Die Methode gibt den eindeutigen Titel des Musikstücks zurück.

Anfrage **`String gibInterpret()`**
Die Methode gibt den Interpreten des Musikstücks zurück.

Anfrage **`int gibErscheinungsjahr()`**
Die Methode gibt das Erscheinungsjahr des Musikstücks zurück.

Name: _____

Die Klasse `DJAssistent`

Objekte dieser Klasse verwalten einen Graphen mit Objekten der Klasse `Musikstueck` und ihren Beziehungen untereinander. Die Klasse stellt unter anderem folgende Methoden zur Verfügung:

Auszug aus der Dokumentation der Klasse `DJAssistent`

Konstruktor **`DJAssistent()`**
Ein Objekt der Klasse `DJAssistent` wird initialisiert.

Anfrage **`int gibUebergangsGuete(Musikstueck pMusikstueck1, Musikstueck pMusikstueck2)`**
Die Methode gibt die Güte des direkten Übergangs der beiden `Musikstueck`-Objekte `pMusikstueck1` und `pMusikstueck2` als Wert zwischen 1 und 3 zurück (vgl. Aufgabenstellung). Gibt es zwischen beiden Musikstücken gar keinen Übergang, wird 0 geliefert.

Anfrage **`boolean passtAlsNaechstesOderUebernaechstes(Musikstueck pMusikstueck1, Musikstueck pMusikstueck2)`**
Die Methode überprüft, ob es gut ist, ein Musikstück direkt nach einem anderen oder als übernächstes abzuspielen.
Dazu überprüft sie, ob es einen mindestens guten Übergangspfad, d. h. Güte 2 oder 3, gibt, welcher die beiden `Musikstueck`-Objekte `pMusikstueck1` und `pMusikstueck2` mit einer oder zwei Kanten verbindet (vgl. Aufgabenstellung).
Die Methode gibt `true` zurück, falls es einen solchen Übergangspfad gibt, andernfalls gibt sie `false` zurück.

Name: _____

Die Klasse Graph

Die Klasse `Graph` stellt einen ungerichteten, kantengewichteten Graphen dar. Es können Knoten- und Kantenobjekte hinzugefügt und entfernt, flache Kopien der Knoten- und Kantenlisten des Graphen angefragt und Markierungen von Knoten und Kanten gesetzt und überprüft werden. Des Weiteren kann eine Liste der Nachbarn eines bestimmten Knoten, eine Liste der inzidenten Kanten eines bestimmten Knoten und die Kante von einem bestimmten Knoten zu einem anderen Knoten angefragt werden. Abgesehen davon kann abgefragt werden, welches Knotenobjekt zu einer bestimmten ID gehört und ob der Graph leer ist.

Dokumentation der Klasse `Graph`

Konstruktor **`Graph()`**

Ein Objekt vom Typ `Graph` wird erstellt. Der von diesem Objekt repräsentierte Graph ist leer.

Auftrag **`void addVertex(Vertex pVertex)`**

Der Auftrag fügt den Knoten `pVertex` vom Typ `Vertex` in den Graphen ein, sofern es noch keinen Knoten mit demselben ID-Eintrag wie `pVertex` im Graphen gibt und `pVertex` eine ID ungleich `null` hat. Ansonsten passiert nichts.

Auftrag **`void addEdge(Edge pEdge)`**

Der Auftrag fügt die Kante `pEdge` in den Graphen ein, sofern beide durch die Kante verbundenen Knoten im Graphen enthalten sind, nicht identisch sind und noch keine Kante zwischen den beiden Knoten existiert. Ansonsten passiert nichts.

Auftrag **`void removeVertex(Vertex pVertex)`**

Der Auftrag entfernt den Knoten `pVertex` aus dem Graphen und löscht alle Kanten, die mit ihm inzident sind. Ist der Knoten `pVertex` nicht im Graphen enthalten, passiert nichts.

Auftrag **`void removeEdge(Edge pEdge)`**

Der Auftrag entfernt die Kante `pEdge` aus dem Graphen. Ist die Kante `pEdge` nicht im Graphen enthalten, passiert nichts.

Anfrage **`Vertex getVertex(String pID)`**

Die Anfrage liefert das Knotenobjekt mit `pID` als ID. Ist ein solches Knotenobjekt nicht im Graphen enthalten, wird `null` zurückgeliefert.

Name: _____

Anfrage **`List<Vertex> getVertices()`**

Die Anfrage liefert eine neue Liste aller Knotenobjekte vom Typ `List<Vertex>`. Enthält der Graph keine Knotenobjekte, so wird eine leere Liste zurückgeliefert.

Anfrage **`List<Vertex> getNeighbours(Vertex pVertex)`**

Die Anfrage liefert alle Nachbarn des Knotens `pVertex` als neue Liste vom Typ `List<Vertex>`. Hat der Knoten `pVertex` keine Nachbarn in diesem Graphen oder ist gar nicht in diesem Graphen enthalten, so wird eine leere Liste zurückgeliefert.

Anfrage **`List<Edge> getEdges()`**

Die Anfrage liefert eine neue Liste aller Kantenobjekte vom Typ `List<Edge>`. Enthält der Graph keine Kantenobjekte, so wird eine leere Liste zurückgeliefert.

Anfrage **`List<Edge> getEdges(Vertex pVertex)`**

Die Anfrage liefert eine neue Liste aller inzidenten Kanten zum Knoten `pVertex`. Hat der Knoten `pVertex` keine inzidenten Kanten in diesem Graphen oder ist gar nicht in diesem Graphen enthalten, so wird eine leere Liste zurückgeliefert.

Anfrage **`Edge getEdge(Vertex pVertex, Vertex pAnotherVertex)`**

Die Anfrage liefert die Kante, welche die Knoten `pVertex` und `pAnotherVertex` verbindet, als Objekt vom Typ `Edge`. Ist der Knoten `pVertex` oder der Knoten `pAnotherVertex` nicht im Graphen enthalten oder gibt es keine Kante, die beide Knoten verbindet, so wird `null` zurückgeliefert.

Auftrag **`void setAllVertexMarks(boolean pMark)`**

Der Auftrag setzt die Markierungen aller Knoten des Graphen auf den Wert `pMark`.

Anfrage **`boolean allVerticesMarked()`**

Die Anfrage liefert `true`, wenn die Markierungen aller Knoten des Graphen den Wert `true` haben, ansonsten `false`.

Auftrag **`void setAllEdgeMarks(boolean pMark)`**

Der Auftrag setzt die Markierungen aller Kanten des Graphen auf den Wert `pMark`.

Name: _____

Anfrage `boolean allEdgesMarked()`
Die Anfrage liefert `true`, wenn die Markierungen aller Kanten des Graphen den Wert `true` haben, ansonsten `false`.

Anfrage `boolean isEmpty()`
Die Anfrage liefert `true`, wenn der Graph keine Knoten enthält, ansonsten `false`.

Die Klasse `Vertex`

Die Klasse `Vertex` stellt einen einzelnen Knoten eines Graphen dar. Jedes Objekt dieser Klasse verfügt über eine im Graphen eindeutige ID als String und kann diese ID zurückliefern. Darüber hinaus kann eine Markierung gesetzt und abgefragt werden.

Dokumentation der Klasse `Vertex`

Konstruktor `Vertex(String pID)`
Ein neues Objekt vom Typ `Vertex` ohne Inhaltsobjekt wird erstellt. Der von diesem Objekt repräsentierte Knoten ist noch in keinen Graphen eingefügt. Seine Markierung hat den Wert `false`.

Anfrage `String getID()`
Die Anfrage liefert die `ID` des Knotens als `String`.

Auftrag `void setMark(boolean pMark)`
Der Auftrag setzt die Markierung des Knotens auf den Wert `pMark`.

Anfrage `boolean isMarked()`
Die Anfrage liefert `true`, wenn die Markierung des Knotens den Wert `true` hat, ansonsten `false`.

Name: _____

Die Klasse Edge

Die Klasse Edge stellt eine einzelne, ungerichtete Kante eines Graphen dar. Beim Erstellen werden die beiden durch sie zu verbindenden Knotenobjekte und eine Gewichtung als `double` übergeben. Beide Knotenobjekte können abgefragt werden. Des Weiteren können die Gewichtung und eine Markierung gesetzt und abgefragt werden.

Dokumentation der Klasse Edge

Konstruktor **`Edge(Vertex pVertex, Vertex pAnotherVertex, double pWeight)`**

Ein neues Objekt vom Typ `Edge` wird erstellt. Die von diesem Objekt repräsentierte Kante verbindet die Knoten `pVertex` und `pAnotherVertex` mit der Gewichtung `pWeight` und ist noch in keinen Graphen eingefügt. Ihre Markierung hat den Wert `false`.

Auftrag **`void setWeight(double pWeight)`**

Der Auftrag setzt das Gewicht der Kante auf den Wert `pWeight`.

Anfrage **`double getWeight()`**

Die Anfrage liefert das Gewicht der Kante als `double`.

Anfrage **`Vertex[] getVertices()`**

Die Anfrage gibt die beiden Knoten, die durch die Kante verbunden werden, als neues Feld vom Typ `Vertex` zurück. Das Feld hat genau zwei Einträge mit den Indexwerten `0` und `1`.

Auftrag **`void setMark(boolean pMark)`**

Der Auftrag setzt die Markierung der Kante auf den Wert `pMark`.

Auftrag **`void setWeight(double pWeight)`**

Der Auftrag setzt das Gewicht der Kante auf den Wert `pWeight`.

Anfrage **`boolean isMarked()`**

Die Anfrage liefert `true`, wenn die Markierung der Kante den Wert `true` hat, ansonsten `false`.

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.

Name: _____

Die generische Klasse Queue<ContentType>

Objekte der generischen Klasse **`Queue`** (Warteschlange) verwalten beliebige Objekte vom Typ **`ContentType`** nach dem First-In-First-Out-Prinzip, d. h., das zuerst abgelegte Objekt wird als erstes wieder entnommen. Alle Methoden haben eine konstante Laufzeit, unabhängig von der Anzahl der verwalteten Objekte.

Dokumentation der generischen Klasse `Queue<ContentType>`

Konstruktor **`Queue()`**

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

Anfrage **`boolean isEmpty()`**

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

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

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

Auftrag **`void dequeue()`**

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

Anfrage **`ContentType front()`**

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

Unterlagen für die Lehrkraft

Abiturprüfung 2020

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation von kontextbezogenen 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 2020

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Objekte und Klassen
     - Entwurfsdiagramme und Implementationsdiagramme
     - Lineare Strukturen
       Schlange (Klasse `Queue`*)*
       Lineare Liste (Klasse `List`*)*
     - Nicht-lineare Strukturen
       Graphen (Klasse `Graph`, `Vertex`, `Edge`*)*
   - Datenbanken

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

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

2. *Medien/Materialien*
   - entfällt

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

5. Zugelassene Hilfsmittel

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

6. Modelllösungen

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

Teilaufgabe a)

Zwischen den genannten Stücken sollten die Musikstücke mit den Titeln `"Swinging Charly"` und `"Super Swing"` gespielt werden.

`"Dancing For Me"` bildet sozusagen eine Engstelle in dem Graphen. Wenn man mit diesem Musikstück anfängt, muss man sich entscheiden, ob man ausschließlich Musikstücke aus dem oberen Teilgraphen oder ausschließlich solche aus dem unteren Teilgraphen abspielen möchte. Würde man bei diesem Startstück beide Teilgraphen besuchen wollen, müsste das Startstück doppelt gespielt werden. Dies soll gemäß der Aufgabenstellung jedoch vermieden werden.

Teilaufgabe b)

Die Beziehung zwischen den Klassen `Musikstueck` und `Vertex` ist eine Vererbung, d. h., die Klasse `Musikstueck` besitzt zusätzlich zu ihren spezifischen Attributen und Methoden alle Attribute und Methoden der Klasse `Vertex`. Somit können Objekte der Klasse `Musikstueck` im Graphen als spezielle Knoten verwaltet werden.

Die Klasse `DJAssistent` verwaltet im Attribut `djGraph` ein Objekt der Klasse `Graph`. In diesem Graphen werden die einzelnen Musikstücke, samt ihren Beziehungen untereinander, verwaltet.

Aus Entwurfssicht ist ein Objekt der Klasse `DJAssistent` kein Graph, sondern verwaltet lediglich die Objekte der Klasse `Musikstueck` in einem Objekt der Klasse `Graph`. Würde die Klasse `DJAssistent` als Unterklasse der Klasse `Graph` modelliert, so würden sämtliche Methoden der Klasse `Graph` nach außen sichtbar sein und man könnte von außen die intern gespeicherten Daten kontextwidrig manipulieren. Dies widerspricht dem Grundsatz der Datenkapselung in einer objektorientierten Modellierung.

Teilaufgabe c)

`gibUebergangsGuete(musikstueck1, musikstueck2)` gibt den Wert `1` zurück.

`passtAlsNaechstesOderUebernaechstes(musikstueck1, musikstueck2)` gibt den Wert `true` zurück.

Eine Implementierung der Methode könnte wie folgt aussehen.

```
public boolean passtAlsNaechstesOderUebernaechstes(
            Musikstueck pMusikstueck1, Musikstueck pMusikstueck2) {
  boolean passt = false;
  if (gibUebergangsGuete(pMusikstueck1, pMusikstueck2) >= 2) {
    passt = true;
  } else {
    List<Vertex> nachbarn = djGraph.getNeighbours(pMusikstueck1);
    nachbarn.toFirst();
    while (nachbarn.hasAccess() && !passt) {
      Musikstueck aktuell = (Musikstueck) nachbarn.getContent();
      if (gibUebergangsGuete(pMusikstueck1, aktuell) >= 2
              && gibUebergangsGuete(aktuell, pMusikstueck2) >= 2) {
        passt = true;
      }
      nachbarn.next();
    }
  }
  return passt;
}
```

Teilaufgabe d)

Die Warteschlange wird am Anfang mit dem Objekt `pMusikstueck1 (Musikstueck`-Objekt zum Titel `"Dancing For Me")` initialisiert und der zugehörige Knoten wird markiert (Zeile 5 und 6).

Die Schleife in den Zeilen 7 bis 22 wird nun solange ausgeführt, bis die Warteschlange leer ist (und deshalb der Zielknoten nicht erreicht wurde) oder bis das erste Element der Warteschlange dem Zielknoten (hier also dem `Musikstueck`-Objekt `pMusikstueck2` zu dem Musiktitel `"Hello Germany!"`) entspricht.

Innerhalb der Schleife werden alle unmarkierten 3er-Nachbarn (d. h. über eine Kante mit dem Kantengewicht 3 verbundene Knoten) des ersten Knotens der Warteschlange gesucht, markiert und hinten in die Warteschlange eingereiht (Zeilen 9 bis 20). Dies ist im ersten Durchlauf nur das `Musikstueck`-Objekt zum Titel `"Tonight Is The Night"`.

Am Ende der Schleife wird dann der vorderste Knoten aus der Warteschlange entfernt (Zeile 21), sodass nun vom `Musikstueck`-Objekt zum Titel `"Tonight Is The Night"` weitergesucht wird.

Von diesem Musikstück ausgehend findet der Algorithmus das Nachbarmusikstück zum Titel `"Tell Me Where!"`, welches in die Warteschlange eingereiht und markiert wird. Weitere unmarkierte 3er-Nachbarn gibt es nicht, weshalb das Musikstück zum Titel `"Tonight Is The Night"` wieder aus der Warteschlange entfernt wird.

Ausgehend vom Musikstück zum Titel `"Tell Me Where!"` findet der Algorithmus das Nachbarmusikstück mit dem Titel `"Hello Germany!"`, welches in die Warteschlange eingereiht und markiert wird. Weitere unmarkierte 3er-Nachbarn gibt es nicht, weshalb das Musikstück mit dem Titel `"Tell Me Where!"` wieder aus der Warteschlange entfernt wird.

Die Schleifenbedingung in Zeile 8 ist nun nicht mehr erfüllt, da der erste Knoten der Warteschlange dem Zielknoten `pMusikstueck2` entspricht. Deshalb geht der Algorithmus zur Rückgabe (Zeilen 23 und 24) über, wo festgestellt wird, dass die Schlange nicht leer ist und der erste Knoten der Warteschlange dem Zielknoten `pMusikstueck2` entspricht.

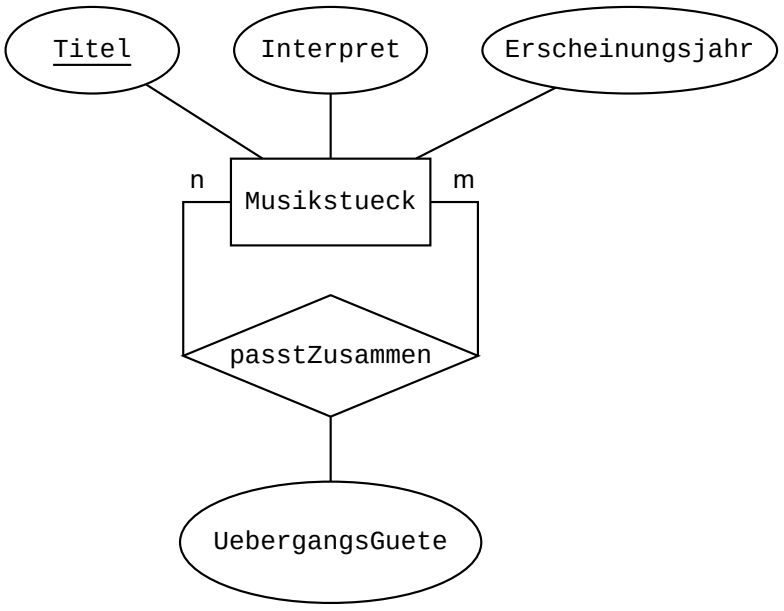
Die Methode gibt den Wert `true` zurück.

Im Hinblick auf die Funktionsweise des Algorithmus bedeutet eine Markierung, dass ein markierter Knoten bei der Suche nach einem Weg von `pMusikstueck1` nach `pMusikstueck2` bereits besucht wurde (Markierung in Zeilen 5 bzw. 16) und deshalb nicht ein weiteres Mal besucht werden darf (Bedingung in Zeile 15).

Im Sachzusammenhang überprüft die Methode, ob es einen optimalen Übergangspfad (Güte 3) zwischen zwei Musikstücken gibt.

Teilaufgabe e)

Eine mögliche Datenbankmodellierung ist die folgende:



Die im Graph gespeicherten Musikstücke entsprechen Entitäten des Entitätstyps `Musikstueck`, dessen Attribute exakt den Attributen der Klasse `Musikstueck` und damit den im Graphen gespeicherten Informationen entsprechen.

Der Beziehungstyp `passtZusammen` modelliert die Kanten, über welche jeweils zwei Objekte der Klasse `Musikstueck` im Graphen verbunden sind, wobei das Kantengewicht als Attribut `UebergangsGuete` der Beziehungsrelation modelliert wurde.

Für die Modellierung des Beziehungstyps `passtZusammen` zwischen zwei Musikstücken wurde ein `m:n`-Beziehungstyp gewählt. So kann ein Musikstück mit beliebig vielen anderen Musikstücken in Beziehung stehen, aber es können auch beliebig viele andere Musikstücke mit einem Musikstück in Beziehung stehen. Dadurch ist gewährleistet, dass jede Kante des Graphen zweimal in der Datenbank gespeichert werden kann, was der Idee des ungerichteten Graphen entspricht.

Insgesamt ermöglicht diese Modellierung somit die Speicherung aller Musikstücke und deren Beziehungen zueinander.

Die Realisierung der Aufgabe zur Methode `passtAlsNaechstesOderUebernaechstes` kann in zwei Schritten erfolgen.

Die Überprüfung, ob zwei Musikstücke durch einen mindestens guten Übergang verbunden sind, erfolgt dadurch, dass die Relation `Musikstueck` über die Relation `passtZusammen` mit sich selbst verbunden wird (`join`). Aus der Ergebnisrelation muss nun noch entsprechend der Bedingung `UebergangsGuete >= 2` selektiert werden. Wenn diese Selektion mindestens einen Datensatz liefert, so ist entschieden, dass die beiden Musikstücke einen mindestens guten Übergang bilden.

Die Überprüfung, ob die beiden Musikstücke als übernächstes gut oder sogar optimal hintereinander passen, erfolgt über eine doppelte Verknüpfung `Musikstueck - passtZusammen - Musikstueck - passtZusammen - Musikstueck`. Aus der Ergebnisrelation muss so wie oben entsprechend der Bedingung `UebergangsGuete >= 2` selektiert werden.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	gibt einen optimalen Übergangspfad zwischen den genannten Musikstücken an.	3			
2	erläutert im Sachzusammenhang, welche Auswirkungen es hat, wenn der Titel `"Dancing For Me"` an einem Abend als erstes abgespielt wird.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe a)	**7**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert die Beziehungen zwischen den Klassen `DJAssistent`, `Musikstueck`, `Graph` und `Vertex`.	4			
2	begründet, dass es nicht sinnvoll ist, die Klasse `DJAssistent` als Unterklasse der Klasse `Graph` zu modellieren.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe b)	**8**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	gibt an, welche Werte die Methodenaufrufe für den Beispielgraphen zurückgeben.	2			
2	implementiert die Methode `passtAlsNaechstesOderUebernaechstes`.	9			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe c)	**11**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methode und erläutert den Quelltext unter Bezugnahme auf die Veränderungen der Schlange bei Anwendung auf die Beispieldaten.	6			
2	ermittelt den Rückgabewert der Methode bei Anwendung auf die Beispieldaten.	2			
3	erläutert die Bedeutung der Knotenmarkierungen im Hinblick auf die Funktionsweise des Algorithmus.	2			
4	erläutert die Funktionalität der Methode im Sachzusammenhang.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe d)	**12**			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine Datenbankmodellierung in Form eines Entity-Relationship-Diagramms, welche die Speicherung aller Musikstücke und deren Beziehungen zueinander ermöglicht.	4			
2	erläutert, wie die Modellierung die Speicherung aller Musikstücke und deren Beziehungen zueinander ermöglicht.	4			
3	entwickelt eine Idee, wie eine Datenbankabfrage aussehen könnte, welche auf Grundlage der Datenbankmodellierung dieselbe Information wie die Methode `passtAlsNaechstesOderUebernaechstes` ermittelt.	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 2020

Informatik, Leistungskurs

Aufgabenstellung:

Die Firma *Serienflix* bietet Kundinnen und Kunden die Möglichkeit, (TV-)Serien zu streamen, d. h. über das Internet abzurufen. Eine Serie umfasst mehrere Episoden in mehreren Staffeln. Es wird erfasst, welche Kundin bzw. welcher Kunde sich welche Episode angesehen hat. Außerdem besteht die Möglichkeit für eine Kundin bzw. einen Kunden eine angesehene Episode zu bewerten.

Zur Verwaltung der Daten möchte das Unternehmen eine relationale Datenbank aufbauen, die der folgenden Teilmodellierung entspricht:

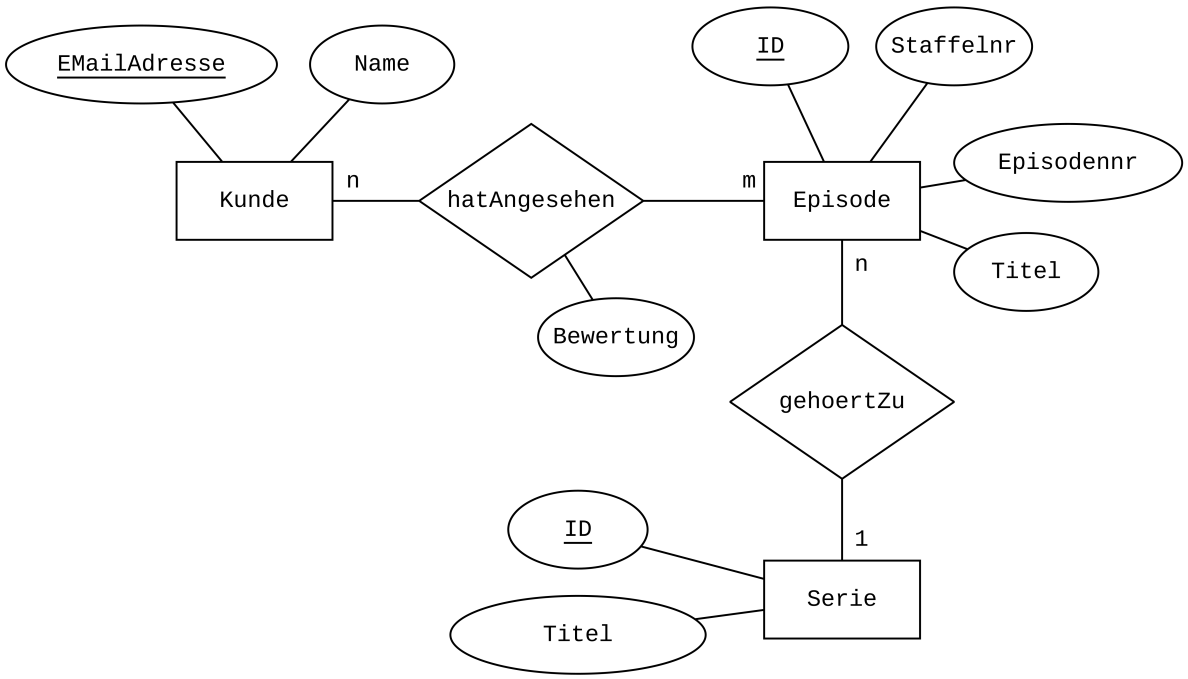


Abbildung 1: Teilmodellierung der Datenbank

Name: _____

Im Folgenden soll mit dem in Abbildung 2 gegebenen Datenbankschema gearbeitet werden. Es stellt einen Ausschnitt der Gesamtdatenbank dar. Beispieldaten zu diesem Datenbankschema sind in der Anlage zu finden.

Kunde(EMailAdresse, Name)
hatAngesehen(↑EMailAdresse, ↑EpisodenID, Bewertung)
Episode(ID, Titel, Episodennr, Staffelnr, ↑SerienID)
Serie(ID, Titel)

Abbildung 2: Datenbankschema eines Ausschnittes der Datenbank

a) *Beschreiben Sie die in Abbildung 1 als Entity-Relationship-Diagramm dargestellte Teilmodellierung.*

Begründen Sie die im Entity-Relationship-Modell gewählten Kardinalitäten für diesen Sachkontext.

Erläutern Sie, wie der `n:m`*-Beziehungstyp und der* `1:n`*-Beziehungstyp aus dem Entity-Relationship-Modell in diesem Datenbankschema umgesetzt wurden.*

(9 Punkte)

b) Aus der Datenbank sollen folgende Informationen abgefragt werden:

i. Gesucht sind die Serientitel aufsteigend sortiert, die das Wort `Spiel` beinhalten.

ii. Gesucht sind die Serientitel mit der jeweils zugehörigen Durchschnittsbewertung. Das Ergebnis soll nach der Durchschnittsbewertung der Serie absteigend sortiert sein. Serien, bei denen noch keine Episode angesehen wurde, müssen nicht berücksichtigt werden.

iii. Gesucht sind alle Namen der Kundinnen und Kunden – auch derjenigen, die noch gar keine Episode angesehen haben – mit der jeweiligen Anzahl der abgegebenen Bewertungen. Das Ergebnis soll absteigend nach der Anzahl der Bewertungen und dann aufsteigend nach den Kundennamen sortiert sein.

Entwerfen Sie für die obigen Anfragen i, ii und iii jeweils eine SQL-Anweisung.

(11 Punkte)

Name: _____

c) Folgende SQL-Anweisung ist gegeben:

```
 1  SELECT * FROM (
 2    SELECT Serie.Titel, COUNT(*) AS x
 3    FROM Serie
 4      INNER JOIN (
 5          SELECT DISTINCT Episode.SerienID, Episode.Staffelnr
 6          FROM Episode
 7        ) AS Temp
 8        ON Serie.ID = Temp.SerienID
 9    GROUP BY Temp.SerienID
10    ) AS Abfrage
11  WHERE Abfrage.x > 1
12  ORDER BY Abfrage.Titel ASC
```

Analysieren und erläutern Sie zunächst die Zeilen 5 bis 7, dann die Zeilen 2 bis 10 und anschließend die gesamte SQL-Anweisung.

Erläutern Sie im Sachzusammenhang, welche Information die gesamte SQL-Anweisung ermittelt.

(9 Punkte)

d) Die Datenbank soll um folgende Aspekte erweitert werden:

i. Für jede Episode soll verwaltet werden, in welchen Sprachen die Episode angeboten wird. Außerdem soll verwaltet werden, was die Originalsprache einer Episode ist.

ii. Um gezielt nach Angeboten mit dem Lieblingsschauspieler oder der Lieblingsschauspielerin suchen zu können, sollen alle Schauspielerinnen und Schauspieler mit ihren Vornamen und Nachnamen verwaltet werden. Außerdem soll verwaltet werden, wer in welcher Episode mitspielt.

Ein Praktikant hat für einen Teil des ersten Aspekts einen Vorschlag entwickelt:

Episode(ID, Titel, Episodennr, Staffelnr, ↑SerienID,
Sprache1, Sprache2, Sprache3, Sprache4, Sprache5)

Für jede Episode kann unter `Sprache1` bis `Sprache5` eingetragen werden, in welchen Sprachen diese Episode angeboten wird.

Name: _____

Begründen Sie, warum der Vorschlag des Praktikanten ungünstig ist.

Modellieren Sie die Erweiterung für das Entity-Relationship-Diagramm aus Abbildung 1 so, dass der neue Datenbankentwurf zusätzlich die beschriebenen Aspekte (vgl. i und ii) enthält.

Erläutern Sie, wie das von Ihnen entwickelte Modell die Erweiterung um die Verwaltung der Sprachen für Episoden mit der Originalsprache (Anforderung i) umsetzt.

Hinweis: Entitäts- und Beziehungstypen sowie Attribute, die für die Erweiterung nicht relevant sind, müssen nicht dargestellt werden.

(11 Punkte)

e) Die Firma erweitert ein bestehendes Programm, um eine neue Funktion zu testen. Abbildung 3 zeigt einen Ausschnitt des Implementationsdiagramms.

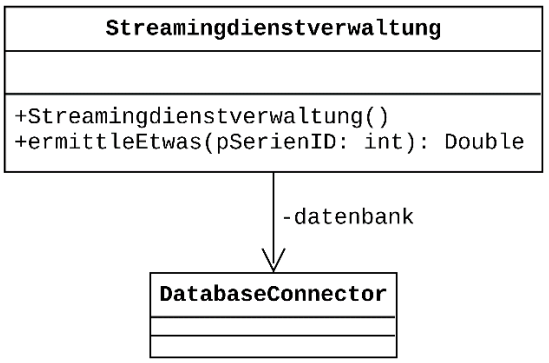


Abbildung 3: Teilmodellierung eines bestehenden Programms einer Streamingdienstverwaltung

Name: _____

Die Klasse `Streamingdienstverwaltung` verfügt über folgende Methode:

```
 1 public Double ermittleEtwas(int pSerienID) {
 2   String sql = "SELECT hatAngesehen.Bewertung "
 3              + "FROM hatAngesehen "
 4              + "  INNER JOIN Episode "
 5              + "    ON hatAngesehen.EpisodenID = Episode.ID "
 6              + "WHERE Episode.SerienID = " + pSerienID
 7              + "  AND hatAngesehen.Bewertung IS NOT NULL "
 8              + "ORDER BY hatAngesehen.Bewertung ASC";
 9
10   datenbank.executeStatement(sql);
11
12   QueryResult qr = datenbank.getCurrentQueryResult();
13   if (qr != null && qr.getRowCount() > 0) {
14     int index = qr.getRowCount() / 2;
15     if (qr.getRowCount() % 2 == 1) {
16       return Double.parseDouble(qr.getData()[index][0]);
17     } else {
18       return (Double.parseDouble(qr.getData()[index-1][0])
19             + Double.parseDouble(qr.getData()[index][0])) / 2;
20     }
21   } else {
22     return null;
23   }
24 }
```

Analysieren Sie die SQL-Anweisung (Zeilen 2 bis 8), indem Sie das Ergebnis der SQL-Anweisung auf Grundlage der Beispieldaten in der Anlage ermitteln, wenn Sie als Parameter `pSerienID` *einmal den Wert* `101` *und einmal den Wert* `222` *übergeben.*

Analysieren Sie die Methode `ermittleEtwas` *und ermitteln Sie die Rückgabe der Methode auf Grundlage der Beispieldaten in der Anlage, wenn Sie als Parameter* `pSerienID` *erneut einmal den Wert* `101` *und einmal den Wert* `222` *übergeben.*

Erläutern Sie die Funktionalität der Methode im Sachkontext.

(10 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anlage:

Ausschnitt aus den Beispieldaten zum Datenbankschema aus Abbildung 2

Kunde	
EMailAdresse	**Name**
ada@lovelace.de	Ada Lovelace
john@vonneumann.de	John von Neumann
alan@turing.de	Alan Turing
konrad@zuse.de	Konrad Zuse
tim@BernersLee.de	Tim Berners-Lee

Serie	
ID	**Titel**
101	Spiel der Throne
222	Auf die schiefe Bahn
369	Die Urknalltheorie
480	Kartenhaus

Episode				
ID	**Titel**	**Episodennr**	**Staffelnr**	**SerienID**
1184	Tief im Wald	1	7	101
1251	Die Mauer	2	7	101
8042	Angespannte Lage	1	8	101
13040	Wie alles begann	1	1	222
16060	Zwickmühle	2	1	222

Hinweis:
Episoden werden in Produktionsabschnitten, sogenannten Staffeln, produziert. Die `Staffelnr` gibt an, zu welcher Staffel eine Episode gehört.

hatAngesehen		
EMailAdresse	**EpisodenID**	**Bewertung**
ada@lovelace.de	1184	8
alan@turing.de	13040	1
john@vonneumann.de	1251	3
alan@turing.de	8042	9
konrad@zuse.de	13040	10
ada@lovelace.de	13040	9
john@vonneumann.de	13040	*NULL*
konrad@zuse.de	16060	8

Name: _____

Anhang:

Die Klasse DatabaseConnector

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

Dokumentation der Klasse `DatabaseConnector`

Konstruktor **`DatabaseConnector(String pIP, String pPort, String pDatabase, String pUsername, String pPassword)`**

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

Auftrag **`void executeStatement(String pSQLStatement)`**

Der Auftrag schickt den im Parameter `pSQLStatement` enthaltenen SQL-Befehl an die Datenbank ab.
Handelt es sich bei `pSQLStatement` um einen SQL-Befehl, der eine Ergebnismenge liefert, so kann dieses Ergebnis anschließend mit der Methode `getCurrentQueryResult` abgerufen werden.

Anfrage **`QueryResult getCurrentQueryResult()`**

Die Anfrage liefert das Ergebnis des letzten mit der Methode `executeStatement` an die Datenbank geschickten SQL-Befehls als Objekt vom Typ `QueryResult` zurück.
Wurde bisher kein SQL-Befehl abgeschickt oder ergab der letzte Aufruf von `executeStatement` keine Ergebnismenge (z.B. bei einem INSERT-Befehl oder einem Syntaxfehler), so wird `null` geliefert.

Anfrage **`String getErrorMessage()`**

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

Auftrag **`void close()`**

Die Datenbankverbindung wird geschlossen.

Name: _____

Die Klasse `QueryResult`

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

Dokumentation der Klasse `QueryResult`

Anfrage **`String[][] getData()`**

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

Anfrage **`String[] getColumnNames()`**

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

Anfrage **`String[] getColumnTypes()`**

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

Anfrage **`int getRowCount()`**

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

Anfrage **`int getColumnCount()`**

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

Unterlagen für die Lehrkraft

Abiturprüfung 2020

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation kontextbezogener Problemstellungen mit Schwerpunkt auf dem Inhaltsfeld Daten und ihre Strukturierung

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2020

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Datenbanken
     - Klassen `DatabaseConnector`, `QueryResult`

   Algorithmen
   - Analyse, Entwurf und Implementierung von Algorithmen

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

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- 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)

Die in Abbildung 1 gegebene Teilmodellierung besteht aus drei Entitätstypen und zwei Beziehungstypen.

Im Entitätstyp `Kunde` wird als Primärschlüssel `EMailadresse` verwendet.
Im Entitätstyp `Episode` wird als Primärschlüssel `ID` verwendet.
Im Entitätstyp `Serie` wird als Primärschlüssel `ID` verwendet.

Entitäten des Typs `Kunde` haben das weitere Attribut `Name`.
Entitäten des Typs `Episode` haben die weiteren Attribute `Titel`, `Episodennr` und `Staffelnr`.
Entitäten des Typs `Serie` haben das weitere Attribut `Titel`.

Der Beziehungstyp `hatAngesehen` modelliert, welche Kundin oder welcher Kunde welche Episode angesehen hat. Zusätzlich hat der Beziehungstyp das Attribut `Bewertung`.

Der Beziehungstyp `gehoertZu` modelliert, welche Episode Bestandteil welcher Serie ist.

`n:m`-Beziehungstyp `hatAngesehen`:
Eine Kundin bzw. ein Kunde kann mehrere Episoden angesehen haben und eine Episode kann von mehreren Kundinnen bzw. Kunden angesehen werden.

`1:n`-Beziehungstyp `gehoertZu`:
Eine Episode gehört zu genau einer Serie. Eine Serie besteht aus mehreren Episoden.

Der `n:m`-Beziehungstyp `hatAngesehen` wurde mit dem Relationenschema `hatAngesehen` umgesetzt. In dieser Relation werden die zwei Fremdschlüsselattribute `EMailAdresse` und `EpisodenID` verwaltet, welche jeweils die Primärschlüssel einmal in der Relation `Kunde` und einmal in der Relation `Episode` darstellen.

Der `1:n`-Beziehungstyp `gehoertZu` wurde so umgesetzt, dass in der Relation `Episode` zusätzlich das Fremdschlüsselattribut `SerienID` verwaltet wird, welches den Primärschlüssel aus der Relation `Serie` abbildet.

Teilaufgabe b)

Die folgenden SQL-Anweisungen realisieren die Anfragen:

i.
```
SELECT Titel
FROM Serie
WHERE Titel LIKE '%Spiel%'
ORDER BY Titel ASC
```

ii.
```
SELECT Serie.Titel, AVG(hatAngesehen.Bewertung)
FROM Serie
  INNER JOIN Episode
    ON Serie.ID = Episode.SerienID
  INNER JOIN hatAngesehen
    ON Episode.ID = hatAngesehen.EpisodenID
GROUP BY Serie.ID
ORDER BY AVG(hatAngesehen.Bewertung) DESC
```

iii.
```
SELECT Kunde.Name, COUNT(hatAngesehen.Bewertung)
FROM Kunde
  LEFT JOIN hatAngesehen
    ON Kunde.EMailAdresse = hatAngesehen.EMailAdresse
GROUP BY Kunde.EMailAdresse
ORDER BY COUNT(hatAngesehen.Bewertung) DESC, Kunde.Name ASC
```

Teilaufgabe c)

Die Unterabfrage (vgl. Zeilen 5 bis 7) ermittelt redundanzfrei alle `SerienID`s und die dazugehörigen Staffelnummern aus der Tabelle `Episode`.

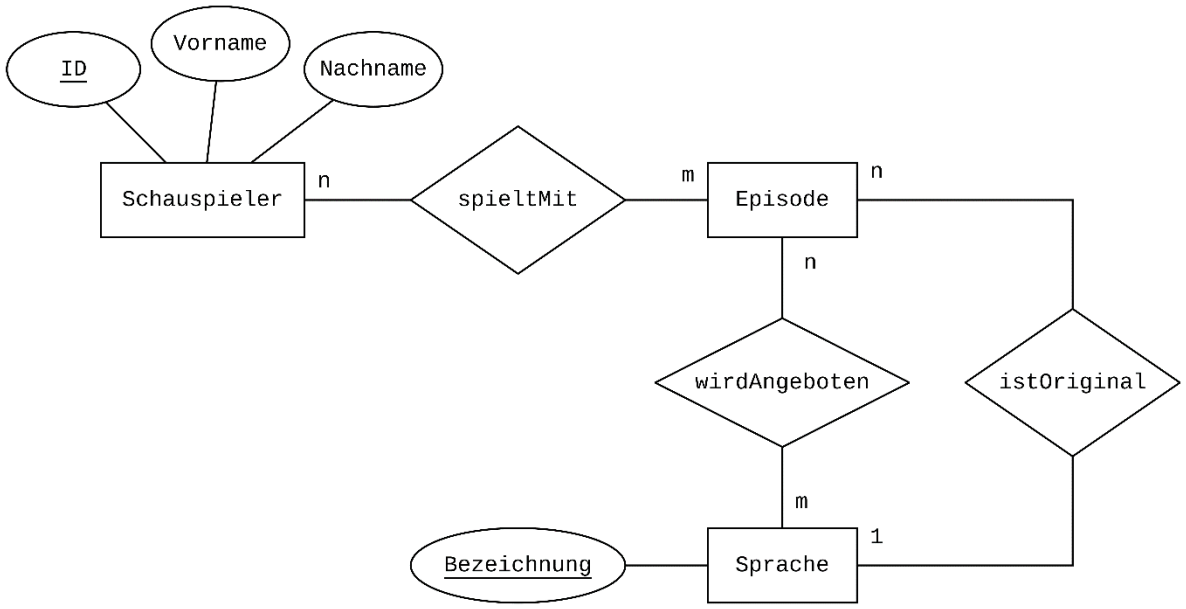
Die Abfrage in Zeilen 2 bis 10 verknüpft mit einem `INNER JOIN` die Tabelle `Serie` mit der Ergebnistabelle aus der genannten Unterabfrage über die `SerienID`. Außerdem gruppiert die SQL-Anweisung die Ergebnisse nach der `SerienID` und ermittelt mit einer Aggregatfunktion die Anzahl der Staffeln.

Die gesamte SQL-Anweisung filtert aus der genannten Abfrage alle Serien mit mehr als einer Staffel und sortiert diese nach dem Serientitel aufsteigend.

Im Sachzusammenhang liefert die gesamte SQL-Anweisung für alle Serien mit mehr als einer Staffel eine Übersicht über alle Serientitel aufsteigend sortiert mit der Anzahl der zur Verfügung stehenden Staffeln.

Teilaufgabe d)

Der Vorschlag des Praktikanten ist ungünstig, weil die Struktur der Relation `Episode` für jede neue Sprache verändert werden muss und die Originalsprache nicht erkennbar ist.



Die Informationen über die zur Verfügung stehenden Sprachen werden in Entitäten des Typs `Sprache` verwaltet. Entitäten dieses Typs werden durch den Primärschlüssel `Bezeichnung` eindeutig identifiziert.

Der `n:m`-Beziehungstyp `wirdAngeboten` zwischen den Entitätstypen `Episode` und `Sprache` modelliert, welche Episode in welchen Sprachen angeboten wird und zu welcher Sprache es welche Episoden gibt.

Der `1:n`-Beziehungstyp `istOriginal` zwischen den Entitätstypen `Episode` und `Sprache` modelliert, was die eine Originalsprache einer Episode ist. Eine Episode kann nur eine Originalsprache haben und eine Sprache kann die Originalsprache in mehreren Episoden sein.

Teilaufgabe e)

Die SQL-Abfrage ermittelt für die übergebene `pSerienID` die jeweils zugehörigen Bewertungen, die aufsteigend sortiert sind.

Ergebnistabelle mit dem Parameteraufruf `pSerienID = 101`
3
8
9

Ergebnistabelle mit dem Parameteraufruf `pSerienID = 222`
1
8
9
10

Die Methode `ermittleEtwas` ermittelt für die übergebene `pSerienID` aus den aufsteigend sortierten Bewertungen eine mittlere Bewertung einer Serie.

Rückgabe der Methode mit dem Parameteraufruf `pSerienID = 101`
8

Rückgabe der Methode mit dem Parameteraufruf `pSerienID = 222`
8.5

Die Methode ermittelt im Sachkontext zu einer bestimmten Serie den Median der Bewertungen. Es wird auf Grundlage der aufsteigend sortierten Bewertungen zu einer Serie die zentrale Bewertung in der Mitte zurückgeliefert. Bei einer ungeraden Anzahl von Datensätzen ist es das mittlere Element. Bei einer geraden Anzahl von Datensätzen ist es das arithmetische Mittel aus den beiden mittleren Elementen.

Falls die gesuchte Serie nicht vorhanden ist oder keine Bewertungen vorliegen, wird `null` zurückgeliefert.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	beschreibt die in Abbildung 1 als Entity-Relationship-Diagramm dargestellte Teilmodellierung.	3			
2	begründet die im Entity-Relationship-Modell gewählten Kardinalitäten für diesen Sachkontext.	3			
3	erläutert, wie der n:m-Beziehungstyp und der 1:n-Beziehungstyp aus dem Entity-Relationship-Modell in diesem Datenbankschema umgesetzt wurden.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe a)	**9**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft für die erste Anfrage i eine SQL-Anweisung.	3			
2	entwirft für die zweite Anfrage ii eine SQL-Anweisung.	4			
3	entwirft für die dritte Anfrage iii eine SQL-Anweisung.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe b)	**11**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert und erläutert die Zeilen 5 bis 7.	2			
2	analysiert und erläutert die Zeilen 2 bis 10.	3			
3	analysiert und erläutert die gesamte SQL-Anweisung.	2			
4	erläutert im Sachzusammenhang, welche Information die gesamte SQL-Anweisung ermittelt.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe c)	**9**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	begründet, warum der Vorschlag des Praktikanten ungünstig ist.	2			
2	modelliert die Erweiterung für das Entity-Relationship-Diagramm so, dass der neue Datenbankentwurf zusätzlich die beschriebenen Aspekte enthält.	5			
3	erläutert, wie das entwickelte Modell die Erweiterung, um die Verwaltung der Sprachen für Episoden mit der Originalsprache (Anforderung i) umsetzt.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe d)	**11**			

Teilaufgabe e)

	Anforderungen	Lösungsqualität			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die SQL-Anweisung, indem er das Ergebnis der SQL-Anweisung auf Grundlage der Beispieldaten in der Anlage ermittelt, wenn er als Parameter `pSerienID` einmal den Wert `101` und einmal den Wert `222` übergibt.	4			
2	analysiert die Methode `ermittleEtwas` und ermittelt die Rückgabe der Methode auf Grundlage der Beispieldaten in der Anlage, wenn er als Parameter `pSerienID` erneut einmal den Wert `101` und einmal den Wert `222` übergibt.	2			
3	erläutert die Funktionalität der Methode im Sachkontext.	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 2020

Informatik, Leistungskurs

Aufgabenstellung:

Widerstände sind wichtige Elemente fast jeder komplexeren elektrischen Schaltung. Mehrere solcher Widerstände lassen sich zu neuen Widerständen kombinieren, wenn das eigentlich erforderliche Bauteil nicht verfügbar ist. In der folgenden Aufgabe werden unterschiedlich komplexe Schaltungen betrachtet. Hierbei wird von Widerständen fester Größe ausgegangen, die in potentiell unbegrenzter Anzahl vorliegen. Aus ihnen sollen neue Widerstände kombiniert werden.

Folgende Abbildungen zeigen Widerstände, hier als Rechtecke dargestellt, in unterschiedlichen Schaltungen.

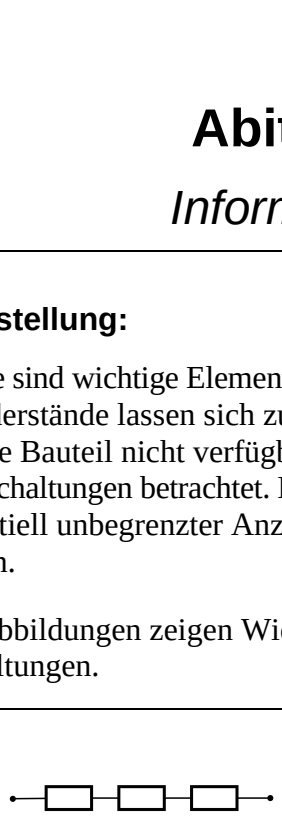


Abbildung 1: Reihenschaltung

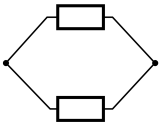


Abbildung 2: Parallelschaltung

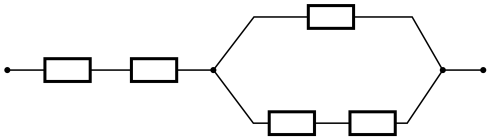


Abbildung 3: Kombination aus Reihen- und Parallelschaltungen

Schaltungen werden im Weiteren in Form von Zeichenfolgen gespeichert, die aus folgenden Zeichen bestehen:

- `w` steht für einen Widerstand
- `p` steht für den Beginn einer Parallelschaltung
- `t` steht als Trennzeichen zwischen parallelen Zweigen in einer Parallelschaltung
- `e` steht für das Ende einer Parallelschaltung

Werden mehrere Widerstände oder Parallelschaltungen hintereinander in einer Zeichenfolge aufgeführt, so steht das für eine Reihenschaltung. Die Schaltung aus Abbildung 1 wird somit durch die Zeichenfolge `www` repräsentiert.

Die Schaltung aus Abbildung 2 wird durch die Zeichenfolge `pwtwe` repräsentiert.

Name: _____

Um die Repräsentation einer Schaltung eindeutig zu halten, werden in Parallelschaltungen weiter oben dargestellte Zweige weiter vorne in der Zeichenfolge aufgeführt. In den Zweigen einer Parallelschaltung können auch Reihenschaltungen von Widerständen enthalten sein. Die Schaltung aus Abbildung 3 wird durch die Zeichenfolge `wwpwtwwe` repräsentiert. Hinter den beiden Widerständen wird mit `p` eine Parallelschaltung eröffnet. Im oberen Zweig der Parallelschaltung steht ein einzelner Widerstand (vor dem `t`), im unteren Zweig der Parallelschaltung (hinter dem `t`) ist eine Reihenschaltung von zwei Widerständen zu finden. Die Parallelschaltung wird mit dem Zeichen `e` abgeschlossen.

Zeichenfolgen, die der folgende nichtdeterministische endliche Automat A_1 mit dem Alphabet $M = \{w, p, t, e\}$ akzeptiert (vgl. Abbildung 4), werden im Weiteren Schaltungskodierungen genannt. Sie bilden die Sprache der Schaltungskodierungen.

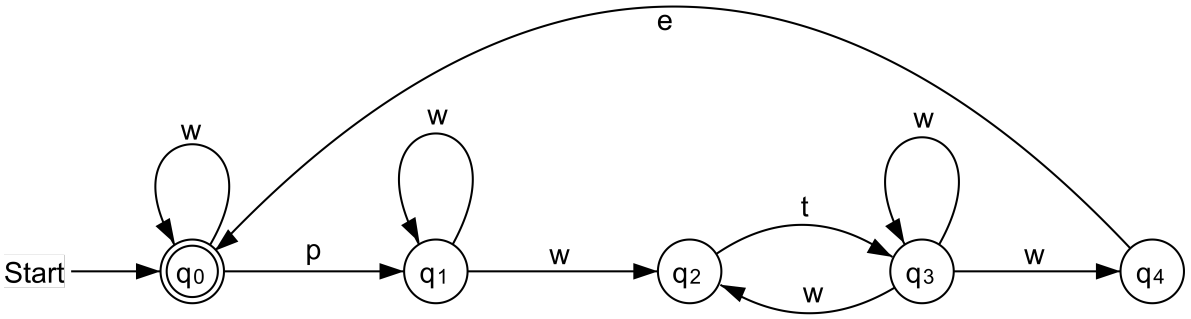


Abbildung 4: Zustandsübergangsdiagramm zum Automaten A_1

a) Abbildung 5 und Abbildung 6 zeigen zwei weitere Schaltungen nach dem oben genannten Prinzip:

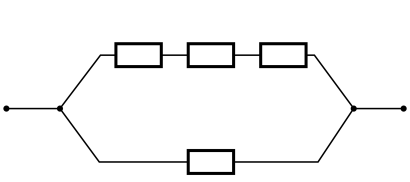


Abbildung 5

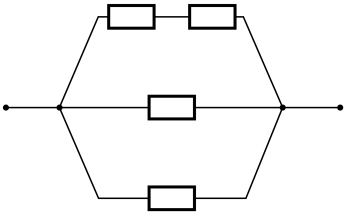


Abbildung 6

Geben Sie die Schaltungskodierungen an, die jeweils zu den Schaltungen in Abbildung 5 und in Abbildung 6 gehören.

Stellen Sie eine Zustandsfolge des Automaten A_1 (Abbildung 4) dar, die zum Akzeptieren der Schaltungskodierung `wpwtwwtwew` *führt.*

Zeigen Sie anhand des Automaten A_1 (Abbildung 4), warum es sich bei der Zeichenfolge `pwtww` *nicht um eine korrekte Schaltungskodierung handelt.*

(9 Punkte)

Name: _____

b) *Entwickeln Sie eine reguläre Grammatik* G_1*, die genau die vom Automaten* A_1 *(Abbildung 4) akzeptierte Sprache erzeugt.*

Erläutern Sie, warum es sich bei der von Ihnen entwickelten Grammatik um eine reguläre Grammatik handelt.

(7 Punkte)

c) Die Sprache der Schaltungskodierungen (vergleiche Abbildung 4) soll so erweitert werden, dass in den Zweigen einer Parallelschaltung neben Widerständen auch weitere, innere Parallelschaltungen enthalten sein können. Abbildung 7 zeigt eine solche Schaltung.

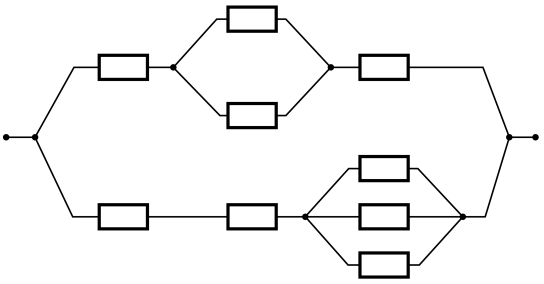


Abbildung 7

In dieser erweiterten Sprache sollen innere Parallelschaltungen selbst in jedem Zweig mindestens einen Widerstand enthalten, aber keine weiteren Parallelschaltungen enthalten können.

Erweitern Sie den Zustandsübergangsgraphen des Automaten A_1 *in der Anlage so, dass der neue Automat auch Schaltungskodierungen mit der oben genannten Erweiterung akzeptiert.*

Geben Sie die Schaltungskodierung zur Schaltung aus Abbildung 7 an und stellen Sie eine Zustandsfolge des von Ihnen erweiterten Automaten dar, die zum Akzeptieren dieser Schaltungskodierung führt.

(13 Punkte)

Name: _____

d) Die folgende Grammatik $G_2 = (N, T, S, P)$ produziert Schaltungskodierungen von Schaltungen, in denen auch Parallelschaltungen in Parallelschaltungen mit einer beliebigen Schachtelungstiefe enthalten sein können.

Nichtterminale: $N = \{S, B\}$
Terminale: $T = \{w, p, t, e\}$
Startsymbol: S
Produktionen: $P = \{$
$S \rightarrow \varepsilon \mid wS \mid pBtBeS,$
$B \rightarrow wS \mid pBtBeS$
$\}$

Entscheiden Sie, ob die Zeichenfolge ppwtwewtwew *ein Wort der von* G_2 *erzeugten Sprache ist.*

Beurteilen Sie, ob die von G_2 *erzeugte Sprache regulär ist.*

(10 Punkte)

e) In der Praxis sollen meist Schaltungskodierungen überprüft werden, in denen in jeder Parallelschaltung wieder Parallelschaltungen möglich sind, d. h. solche Parallelschaltungen beliebig tief geschachtelt sein können.

Um Schaltungskodierungen dieser Art überprüfen zu können, soll im Folgenden ein Kellerautomat entwickelt werden.

In einer ersten Version dieses Kellerautomaten soll zur Vereinfachung die folgende Einschränkung gemacht werden:

Bevor die Schaltungskodierung vom Kellerautomaten überprüft wird, werden alle Vorkommen des Zeichens `t` und alle Vorkommen des Zeichens `w` entfernt. Der Automat soll überprüfen, ob Parallelschaltungen korrekt geöffnet und wieder geschlossen werden.

Beispiel: `ppwtpwtweetpwtwee` wird zu `pppeepee`.

Entwerfen Sie einen Kellerautomaten, der die geforderte Überprüfung vornehmen kann.

Stellen Sie die Erkennung des Wortes pppeepee *dar.*

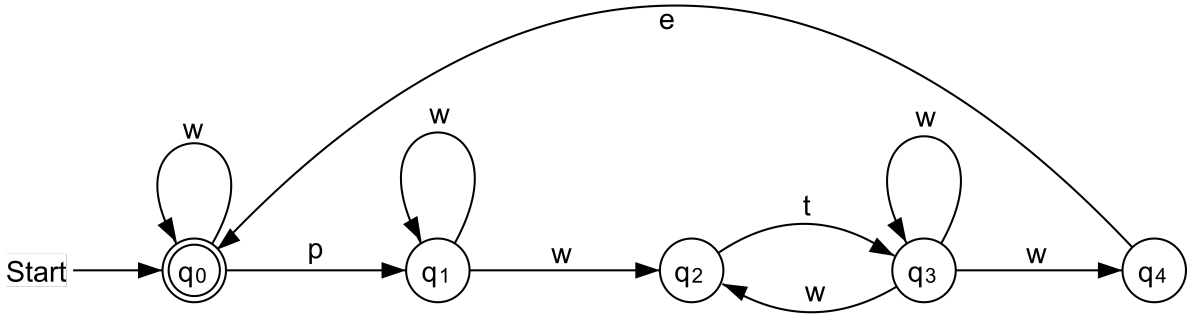
(11 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anlage zu Aufgabe c:



Unterlagen für die Lehrkraft

Abiturprüfung 2020

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 2020

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

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Formale Sprachen und Automaten
   - Endliche Automaten und Kellerautomaten
     - Nichtdeterministische endliche Automaten
     - Nichtdeterministische Kellerautomaten
   - Grammatiken regulärer und kontextfreier Sprachen
     - Produktionen mit ε
   - Möglichkeiten und Grenzen von Automaten und formalen Sprachen

2. *Medien/Materialien*
   - entfällt

5. Zugelassene Hilfsmittel

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- 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)

Schaltungskodierung zur Schaltung in Abbildung 5:
pwwwtwe

Schaltungskodierung zur Schaltung in Abbildung 6:
pwwtwtwe

Eine Zustandsfolge, die zum Akzeptieren der Schaltungskodierung wpwtwwtwew führt, ist folgende:

$$q_0 \xrightarrow{w} q_0 \xrightarrow{p} q_1 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_3 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_4 \xrightarrow{e} q_0 \xrightarrow{w} q_0$$

Die drei möglichen Zustandsfolgen für das Abarbeiten des gesamten Eingabeworts pwtww sind folgende:

$$q_0 \xrightarrow{p} q_1 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_3 \xrightarrow{w} q_2 \quad \text{und} \quad q_0 \xrightarrow{p} q_1 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_3 \xrightarrow{w} q_3$$

und $q_0 \xrightarrow{p} q_1 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_3 \xrightarrow{w} q_4$

Bei allen drei Zustandsfolgen befindet sich der Automat nach Abarbeitung der gesamten Zeichenfolge nicht in einem akzeptierenden Zustand. Daher handelt es sich nicht um eine Schaltungskodierung gemäß der durch den Automaten definierten Sprache.

Teilaufgabe b)

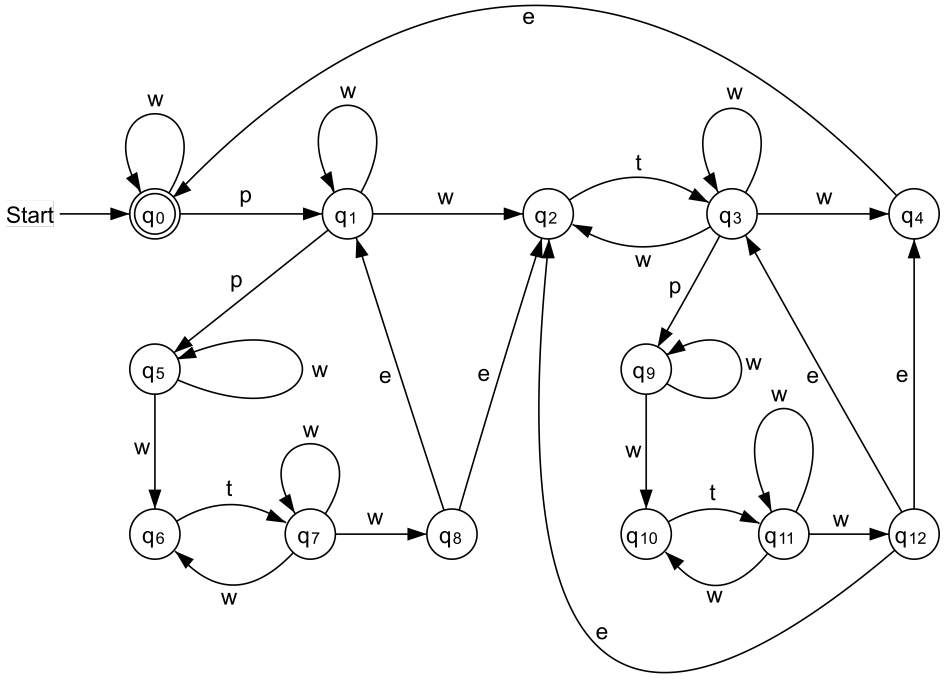
Die Grammatik $G_1 = (N, T, P, S)$ habe die Menge $N = \{S, A, B, C, D\}$ der Nicht-Terminale, die Menge $T = \{w, p, t, e\}$ der Terminale, das Startsymbol $S \in N$ und die folgenden Produktionen:

$S \rightarrow \varepsilon \mid pA \mid wS,$
$A \rightarrow wA \mid wB,$
$B \rightarrow tC,$
$C \rightarrow wB \mid wC \mid wD,$
$D \rightarrow eS$

Es handelt sich um eine reguläre Grammatik, da in sämtlichen Produktionen außer der Produktion $S \rightarrow \varepsilon$ ein einzelnes Nicht-Terminal auf ein einzelnes Terminal gefolgt von einem einzelnen Nicht-Terminal abgeleitet wird.

Teilaufgabe c)

Zustandsübergangsgraph:



Schaltungskodierung zur Schaltung aus Abbildung 7:
pwpwtwewtwwpwtwtwee

Zustandsfolge:

$q_0 \xrightarrow{p} q_1 \xrightarrow{w} q_1 \xrightarrow{p} q_5 \xrightarrow{w} q_6 \xrightarrow{t} q_7 \xrightarrow{w} q_8 \xrightarrow{e} q_1 \xrightarrow{w} q_2 \xrightarrow{t} q_3 \xrightarrow{w} q_3 \xrightarrow{w} q_3 \xrightarrow{p} q_9 \xrightarrow{w}$
$q_{10} \xrightarrow{t} q_{11} \xrightarrow{w} q_{10} \xrightarrow{t} q_{11} \xrightarrow{w} q_{12} \xrightarrow{e} q_4 \xrightarrow{e} q_0$

Teilaufgabe d)

Das Wort ppwtwewtwew ist ein Wort der von G_2 erzeugten Sprache, da es vom Startsymbol abgeleitet werden kann.

$S \Rightarrow pBtBeS \Rightarrow ppBtBeStBeS \Rightarrow ppwStBeStBeS \Rightarrow ppwtBeStBeS$
$\Rightarrow ppwtwSeStBeS \Rightarrow ppwtweStBeS \Rightarrow ppwtwewStBeS \Rightarrow ppwtwewtBeS$
$\Rightarrow ppwtwewtwSeS \Rightarrow ppwtwewtweS \Rightarrow ppwtwewtwewS \Rightarrow ppwtwewtwew$

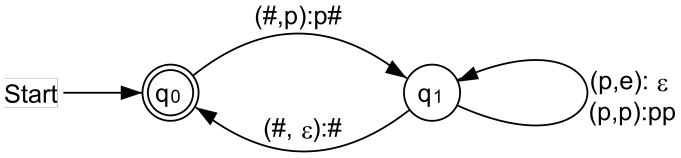
Die von G_2 erzeugte Sprache ist nicht regulär. Die Wörter der Sprache sind der Form, dass jedes p mit einem e abgeschlossen wird. Diese Paare können beliebig oft und tief ineinander geschachtelt werden. Dies ist an den Produktionen $S \rightarrow pBtBeS$ und $B \rightarrow pBtBeS$ zu erkennen. Wäre die Sprache regulär, so gäbe es einen endlichen Automaten, der genau ihre Wörter akzeptieren würde und sich daher die Anzahl der Vorkommen von p merken müsste, um sie mit der Anzahl der Vorkommen von e zu vergleichen. Da ein endlicher Automat nur über seine Zustände zählen kann, gibt es für jeden Automaten eine Anzahl der Vorkommen von p, die größer ist als die Anzahl seiner Zustände. Daher kann es keinen solchen endlichen Automaten geben.

Teilaufgabe e)

Der folgende Kellerautomat erfüllt die Anforderungen der Aufgabenstellung:

Zustandsmenge $Z = \{q_0, q_1\}$
Eingabealphabet $A = \{p, e\}$
Kelleralphabet $K = \{p, \#\}$
Anfangszustand $q_0 \in Z$
Kellerstartsymbol $\# \in K$
Menge der Endzustände $E = \{q_0\}$

Zustandsübergangsdiagramm:



Erkennung des Wortes pppeepee:

Zustand	*gelesenes Eingabesymbol*	*Kellerinhalt*	*Neuer Kellerinhalt*	*Neuer Zustand*
q_0	p	# 	p #	q_1
q_1	p	p #	p p #	q_1
q_1	p	p p #	p p p #	q_1
q_1	e	p p p #	p p #	q_1
q_1	e	p p #	p #	q_1
q_1	p	p #	p p #	q_1
q_1	e	p p #	p #	q_1
q_1	e	p #	# 	q_1
q_1	ε	#	#	q_0

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 Schaltungskodierungen beider Schaltungen an.	2			
2	stellt eine Zustandsfolge des Automaten dar, die zum Akzeptieren der Schaltungskodierung `wpwtwwtwew` führt.	3			
3	zeigt anhand des Automaten, warum es sich bei der Zeichenfolge `pwtww` nicht um eine korrekte Schaltungskodierung handelt.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (9) …… ……					
	Summe Teilaufgabe a)	**9**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine reguläre Grammatik, die genau die vom Automaten akzeptierte Sprache erzeugt.	5			
2	erläutert, warum es sich bei der entwickelten Grammatik um eine reguläre Grammatik handelt.	2			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe b)	**7**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erweitert den Zustandsübergangsgraphen des Automaten A_1 so, dass der neue Automat auch Schaltungskodierungen mit der genannten Erweiterung akzeptiert.	8			
2	gibt die Schaltungskodierung zur Schaltung aus Abbildung 7 an.	2			
3	stellt eine Zustandsfolge des erweiterten Automaten dar, die zum Akzeptieren dieser Schaltungskodierung führt.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (13) …… ……					
	Summe Teilaufgabe c)	**13**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entscheidet, ob die Zeichenfolge `ppwtwewtwew` ein Wort der von G_2 erzeugten Sprache ist.	4			
2	beurteilt, ob die von G_2 erzeugte Sprache regulär ist.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …… ……					
	Summe Teilaufgabe d)	**10**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwirft einen Kellerautomaten, der die geforderte Überprüfung vornehmen kann.	6			
2	stellt die Erkennung des Wortes pppeepee dar.	5			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe e)	**11**			

	Summe insgesamt	**50**			

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

	Lösungsqualität			
	maximal erreichbare Punktzahl	**EK**	**ZK**	**DK**
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	**50**			
Punktzahl der gesamten Prüfungsleistung	**150**			
aus der Punktsumme resultierende Note 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