Schriftliche Abiturprüfung
an den allgemein bildenden Gymnasien
im Fach Informatik ab 2017

(2. überarbeitete Fassung vom Oktober 2014)

Auszug aus dem Schwerpunktthemenerlass (Fach Informatik)
Auszug aus den Beurteilungs- und Korrekturrichtlinien (Fach Informatik)

Hinweise für die Abiturientinnen und Abiturienten

Abiturprüfung an den allgemein bildenden Gymnasien

Musteraufgabensatz

Pfüfungsfach: Informatik

Bearbeitungszeit: 240 Minuten

Hilfsmittel: Der an der Schule eingeführte grafikfähige Taschenrechner

Nachschlagewerke zur deutschen Rechtschreibung und Zeichensetzung

Hinweise: Sie erhalten **drei** Aufgabenteile:

Die Aufgabe A mit dem Schwerpunkt Objektorientierte Modellierung und Programmierung

Zwei Aufgaben B aus den Schwerpunktgebieten
– Datenbanken inklusive Verschlüsselung und Datenschutz
– Automaten und formale Sprachen
– Abstrakte Datentypen inklusive erweiterter Algorithmen / Rekursion

Sie sind verpflichtet, die Ihnen vorgelegten **drei** Aufgabenteile zu bearbeiten.

Verwenden Sie für die Reinschrift und den Entwurf je Aufgabenteil **einen neuen** Bogen.

Vermerken Sie auf **jedem Bogen** die Nummer der bearbeiteten Aufgabe.

Sie sind verpflichtet, die Ihnen vorgelegten Aufgaben auf ihre Vollständigkeit (Anzahl der Blätter, Anlagen usw.) zu überprüfen.

Lösungen auf den **Aufgabenblättern** werden **nicht** gewertet.

A Objektorientierte Modellierung und Programmierung

Ein Taxiunternehmen unterhält einen Fuhrpark mit verschiedenen Fahrzeugen zur Personenbeförderung, u.a. mit Autos und Fahrradrikschas (kurz: Rikschas). Das Unternehmen beauftragt Sie mit der Erstellung einer Software zur Verwaltung der Fahrzeugdaten und des täglichen Fahrgeschäfts.
Mit dem zu erstellenden Programm sollen eine ganze Reihe von Fahrzeugen verwaltet werden. Mit den Fahrzeugdaten werden -sofern vorhanden - auch deren Verbrauch, die gefahrenen Wegstrecken und die Einnahmen, die mit dem jeweiligen Fahrzeug erwirtschaftet werden, erfasst. Jedes Fahrzeug wird eindeutig über sein Kennzeichen (z.B: S-AB-123) identifiziert. Bei der Neuaufnahme der Fahrzeugdaten mit der Software werden neben dem Kennzeichen auch die Anzahl der Sitzplätze, der Grundpreis und der Kilometerpreis eingegeben.

Rikschas werden mit Muskelkraft angetrieben. Sie haben drei Sitzplätze, der Grundpreis beträgt 2,70 €, der Kilometerpreis 0,50 €.
Autos werden mit Benzin betrieben. Sie haben je nach Typ vier bis neun Sitzplätze. Um den Benzinverbrauch der Autos kontrollieren zu können, sollen bei der Neuerfassung der Autodaten zusätzlich das Volumen des Benzintanks und der durchschnittliche Benzinverbrauch auf 100 km erfasst werden. Der Grundpreis für ein Auto beträgt einheitlich 5,80 €, der Kilometerpreis 1,60 €. Bei allen Fahrzeugen sind die ersten drei gefahrenen Kilometer im Grundpreis enthalten.
Für den Stadtverkehr hat das Taxiunternehmen Hybrid-Fahrzeuge (Autos mit einem gewöhnlichen Verbrennungsmotor, der von einem Elektromotor unterstützt wird) angeschafft. Der Strom für den Elektromotor wird einer Batterie entnommen, die mit Hilfe des Verbrennungsmotors geladen wird. Bei Betriebsbeginn übernehmen die Fahrer die Fahrzeuge, Autos sind dann vollgetankt. Nach jeder Fahrt wird in Abhängigkeit von der zurückgelegten Wegstrecke für den Kunden der Fahrpreis berechnet. Für jedes Fahrzeug werden dabei der Kilometerstand, die Tagesfahrstrecke, die Tageseinnahmen und bei Autos der Tankinhalt aktualisiert.

A1
- Stellen Sie die Beziehungen zwischen den Klassen `FuhrparkVerwaltung`, `Fahrzeug`, `Rikscha`, `Auto` und `HybridFahrzeug` in einem Klassendiagramm dar. Die Klassen sollen nur den Klassennamen, keine Attribute oder Methoden enthalten.
- Entwerfen Sie ein UML-Klassendiagramm für die Klasse `Fahrzeug` unter Berücksichtigung des Geheimnisprinzips. Die Sichtbarkeit von allen Attributen, Konstruktoren und Methoden, sowie Parameterlisten und eventuelle Rückgabewerte sollen mit angegeben werden.
- Implementieren Sie für die Klasse `Fahrzeug` einen geeigneten Konstruktor.
- Implementieren Sie eine Methode `berechneFahrpreis(...)`, die in Abhängigkeit von der gefahrenen Strecke den Fahrpreis zurückgibt und zusätzlich den Kilometerstand, die Tagesfahrstrecke und die Tageseinnahmen aktualisiert.

(7 VP)

A2
- Implementieren Sie je einen Konstruktor für die Klassen `Rikscha` und `Auto`.
- Für die Autos muss die Methode `berechneFahrpreis(...)` aus der Klasse Fahrzeug überschrieben werden. Bei Eingabe der zurückgelegten Wegstrecke soll anhand des Durchschnittsverbrauchs der Tankinhalt aktualisiert werden.
  Implementieren Sie eine neue Methode `berechneFahrpreis(...)` für die Klasse `Auto`.
- Implementieren Sie für die Klasse `Auto` die Methode `volltanken()`, die in Abhängigkeit vom aktuellen Tankinhalt und vom Tankvolumen die getankte Benzinmenge in Litern zurückgibt.

(6 VP)

Bei Betriebsende werden per Software die Tagesdaten der Fahrzeuge ausgelesen, ausgewertet und auf Tages-Startwerte zurückgesetzt.

A3
- Geben Sie die Deklaration einer geeigneten Datenstruktur für die Liste der Fahrzeuge in der Klasse `FuhrparkVerwaltung` an.
- Implementieren Sie eine Methode, die es erlaubt, die gesamte gefahrene Strecke aller Fahrzeuge des Fuhrparks zu ermitteln.
- Beschreiben Sie, welche Programmerweiterungen Sie vornehmen müssen, damit Ihr Programm die Gesamteinnahmen sowie die gesamte getankte Benzinmenge am Betriebsende ermitteln kann.
- Die Liste der Fahrzeuge sei vorsortiert. Implementieren Sie eine Methode, die es ermöglicht, ein neues Fahrzeug anhand seines Kennzeichens an der alphabetisch korrekten Stelle einzufügen.

(7 VP)

Hinweise:

Die Implementierung von Programmen oder Programmteilen erfolgt jeweils in der in Ihrem Kurs eingeführten Programmiersprache.

Bei der Programmierung ist davon auszugehen, dass sämtliche erforderlichen Daten in einem vernünftigen Rahmen eingegeben werden. An eine Fehlerbehandlung von Falscheingaben ist nicht gedacht.

B1 Datenbanken

B1.1 Ein Autohaus speichert die Daten über zu verkaufende Fahrzeuge und deren Hersteller in einer Tabelle.

FID	Modell	KW	Preis	Hersteller	Ort	…
1	Golf	92	24 000 €	VW	38440 Wolfsburg	…
2	Focus	92	20 500 €	Ford	50742 Köln	…
3	Astra	85	17 800 €	Opel	65428 Rüsselsheim	…
4	Golf	63	19 300 €	VW	38440 Wolfsburg	…
5	Polo	55	15 200 €	VW	38440 Wulfsburg	
6	…	…	…	…	…	...

- Erläutern Sie anhand der Tabelle den Begriff Redundanz. Beschreiben Sie ein Problem, welches als Folge einer redundanten Datenspeicherung auftreten kann.

- Entwerfen Sie für die vorhandenen Daten ein relationales Datenbankschema, welches dieses Problem vermeidet und dessen Attribute atomar sind.

(5 VP)

B1.2 Die Mitgliederverwaltung eines Sportvereins wird mithilfe einer Datenbank durchgeführt. Dabei liegt das folgende relationale Datenbankschema zugrunde.

Mitglied (MID, Name, Vorname, Geburtsjahr, Geschlecht, Straße, PLZ↑)
Ort (PLZ, Ortsname)
Sportart (SpID, Bezeichnung, TID↑)
Trainer (TID, MID↑, Vergütung)
betreibt_Sportart (MID↑, SpID↑)

Geben Sie jeweils die passende SQL-Abfrage an:

- Wie viele Mitglieder hat der Sportverein?
- Welche Mitglieder wohnen in einem Ort, dessen Postleitzahl mit „67“ oder mit „68“ beginnt? Lassen Sie den Namen des Mitglieds und des Ortes ausgeben, aufsteigend sortiert nach den Namen der Mitglieder.
- In welchen Sportarten bekommt ein Trainer mindestens 80 Euro?
  Geben Sie den Namen des Trainers, die Sportart und die Vergütung an.

Bei SQL-Anweisungen sind auch Unterabfragen möglich. Dabei wird zuerst die Unterabfrage ausgewertet und das Ergebnis auf die eigentliche Abfrage angewendet.

```
SELECT Name, Vorname
FROM Mitglied
WHERE Geburtsjahr > (
      SELECT AVG (Geburtsjahr)
      FROM Mitglied
)
```

- Erläutern Sie, welche Daten mit dieser SQL-Anweisung ausgegeben werden.

(8 VP)

B1.3 Die Privatbahn HHR plant ein Zugauskunftssystem. Dabei sollen Daten über die Züge, die Ankunfts- und Abfahrtszeiten, die Bahnhöfe und ihre Standorte sinnvoll gespeichert werden.

Es ist Folgendes zu beachten:
Jeder Bahnhof hat eine gewisse Anzahl von Gleisen und liegt in einer Stadt. Dabei können in einer Stadt auch mehrere Bahnhöfe liegen. Bei Städten soll außer dem Namen auch die Region gespeichert sein. Jeder Zug hat eine Bezeichnung und eine Anzahl von Wagen. Für einen Zug sollen an jedem Bahnhof die Ankunfts- und Abfahrtszeit und das Gleis gespeichert werden.

- Modellieren Sie zu dem oben beschriebenen Sachverhalt ein ER-Diagramm.
  Tragen Sie die Kardinalitäten ein und begründen Sie diese. Geben Sie für jede Entität die Attribute und Primärschlüssel an.

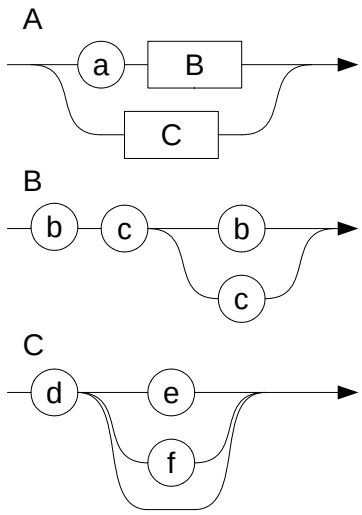
- Überführen Sie das ER-Diagramm in ein optimiertes relationales Datenbankschema.
  Kennzeichnen Sie dabei Primär- und Fremdschlüssel.

(7 VP)

B2 Automaten und formale Sprachen

B2.1 Das Syntaxdiagramm rechts beschreibt eine Sprache L. A ist dabei das Startsymbol.

- Geben Sie alle Wörter dieser Sprache an.



(3 VP)

B2.2 Wir betrachten im Folgenden nur die Kleinschreibweise. Das verwendete Alphabet Σ umfasst die Buchstaben von a bis z sowie die Umlaute ä, ö und ü.
Hinweis zur Notation: Es ist gestattet, Symbolmengen der Art „alle Zeichen außer dem x und dem y“ als Σ \ {x,y} zu schreiben.

- Entwerfen Sie einen endlichen Automaten, der nur Wörter akzeptiert, die die Zeichenfolge „tag“ enthalten (z. B. montag, wochentag, tagsüber, xtagy...)
- Erläutern Sie, wie man anhand der Zustandsfolge dieses Automaten erkennt, dass das Wort „mittags“ zur Sprache gehört.
- Geben Sie eine Grammatik zu dieser Sprache an.

(7 VP)

B2.3
- Erläutern Sie, wie man bei einem endlichen Automaten erkennen kann, ob die von ihm akzeptierte Sprache endlich viele oder unendlich viele Wörter enthält. Begründen Sie Ihre Antwort kurz.

(4 VP)

B2.4 Die `if`-Anweisung in der Programmiersprache Ruby ist wie folgt definiert:

```
if expr [then]
  {expr}
{elsif expr [then]
  {expr}}
[else
  {expr}]
end
```

`{ }` bedeutet, dass der Ausdruck in der Klammer beliebig oft vorkommen darf, auch keinmal

`[ ]` bedeutet, dass der Ausdruck in der Klammer genau einmal oder keinmal vorkommen darf

Terminale sind: `if` `then` `end` `elsif` `else`

Nichtterminale sind: `expr`

`expr` ist dabei ein syntaktisch korrekter Ruby-Ausdruck

- Analysieren Sie, ob die folgende Anweisung in Ruby syntaktisch korrekt ist und begründen Sie Ihre Aussage.
  Sie können davon ausgehen, dass die unterstrichenen Ausdrücke syntaktisch korrekt sind und nicht weiter überprüft werden müssen.

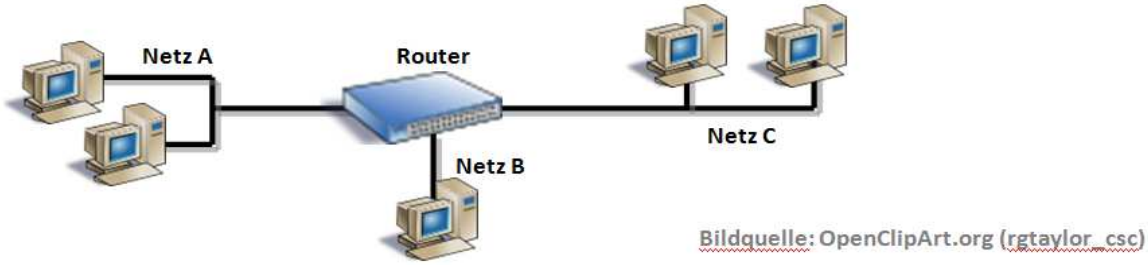
```
if age >= 18
  print "voller Preis"
elsif age >= 7 and age < 18 then
  print "halber Preis"
else
  print "freier Eintritt"
```

- Überführen Sie die obige allgemeine Definition der `if`-Anweisung in ein Syntaxdiagramm.

(6 VP)

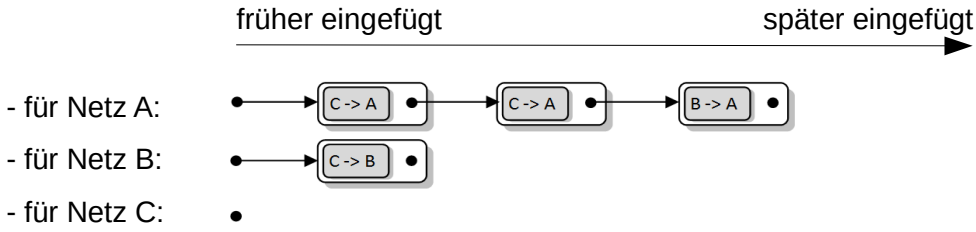
B3 Abstrakte Datentypen

Durch den zunehmenden Datenverkehr im Internet gibt es eine Diskussion wie mit Leitungsengpässen umzugehen ist. Zur Zeit gilt die Netzneutralität, d.h. alle Datenpakete - egal von welchem Absender an welchen Empfänger sie gehen und egal welchen Inhalt sie haben - werden gleich behandelt. Bei einer ausgelasteten Leitung werden ankommende Pakete vom Router zwischengespeichert und dann in der Reihenfolge ihres Eintreffens weitergeleitet. Die Betreiber von Telekommunikationsnetzwerken möchten diese Netzneutralität gerne abschaffen und die Daten in Prioritätsklassen einteilen. Pakete mit höherer Priorität würden dann bevorzugt weitergeleitet.



Zu Beginn sind schon folgende Pakete zwischengespeichert
(B -> A : stellt ein Paket von Netz B nach Netz A dar):

Zwischenspeicher am Anfang



Pro Zeittakt schickt der Router zuerst in jedes Netz jeweils ein Datenpaket. Danach empfängt er gegebenenfalls aus jedem Netz ein Paket (in der Reihenfolge Netz A, Netz B und zuletzt Netz C).

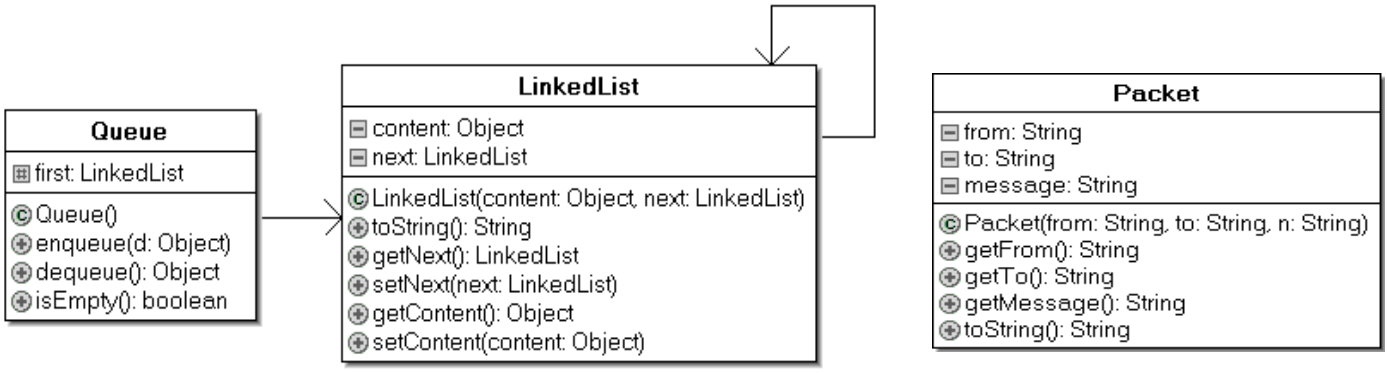
Folgende Pakete erreichen den Router:

Takt 1: Paket von B->A, Paket von C->A, Paket von A->C
Takt 2: Paket von C->A
Takt 3: Paket von A->B, Paket von C->B
Takt 4: Paket von A->B, Paket von C->B, Paket von B->C

B3.1 Das Steuerungsprogramm eines die Netzneutralität wahrenden Routers könnte den ADT Schlange (Queue) zur Verwaltung der Pakete verwenden. Für jedes angeschlossene Netz gibt es dann eine Schlange, die die noch abzuarbeitenden Datenpakete speichert.

- Beschreiben Sie die grundlegenden Eigenschaften einer Schlange und erläutern Sie, warum die Wahl des ADT Schlange für diese Anwendung sinnvoll ist.
- Geben Sie an, welche Operationen der Warteschlange für Netz A ausgeführt werden müssen, um
  - das nächste zu übermittelnde Paket von C->A aus der Warteschlange zu holen,
  - das in Takt 1 ankommende Paket aus Netz B für Netz A zu speichern.
- Stellen Sie die Schlangen nach dem 4. Takt dar, wenn der Router die Netzneutralität wahrt.
- Erläutern Sie die Bedeutung der Beziehungen zwischen den unten dargestellten Klassen. Gehen Sie insbesondere auf die Bedeutung der Beziehung einer Klasse mit sich selbst ein.
- Implementieren Sie die Methode enqueue(Object d) der Klasse Queue auf der Basis des unten dargestellten Klassendiagramms, so dass neue Pakete an das Ende der verketteten Liste angehängt werden. Sie können davon ausgehen, dass alle anderen Methoden schon

korrekt implementiert sind.



(12 VP)

B3.2 Um bestimmte Pakete zu bevorzugen, wird nun eine Priorität der Pakete eingeführt. Alle Pakete, die aus Netz C kommen, sollen mit höherer Priorität weitergeleitet werden. Um dies zu realisieren, wird der Datentyp PriorityQueue verwendet, der sich nur in der Methode enqueue von der normalen Schlange unterscheidet: Die Pakete mit der höheren Priorität werden vor den Paketen mit niedrigerer Priorität in der Schlange einsortiert.

- Stellen Sie die Schlangen nach dem 4. Takt dar, wenn der Router diese Priorität beachtet.
- Ergänzen Sie das Klassendiagramm um die neue Klasse PriorityQueue.
  Dabei sollen die nicht veränderten Methoden von der Klasse Queue übernommen werden.
  Erläutern Sie die dabei neu hinzugekommene Beziehung.

(4 VP)

B3.3 Es gibt Möglichkeiten eine PriorityQueue zu implementieren, so dass die Methoden enqueue und dequeue in einer durchschnittlichen Laufzeit proportional zu log(n) ausgeführt werden (z.B. mittels Heap). Dabei ist n die Anzahl der in der PriorityQueue gespeicherten Pakete.

- Analysieren Sie das Laufzeitverhalten von enqueue und dequeue bei einer PriorityQueue auf der Basis einer LinkedList. Beurteilen Sie, ob es bei einem Router im Hinblick auf die Laufzeit sinnvoller ist, einen Heap oder eine LinkedList als Datenstruktur für eine PriorityQueue einzusetzen.

(4 VP)

Lösungshinweise

Für die Fachlehrerin, den Fachlehrer

Die Lösungshinweise erheben nicht den Anspruch, die einzigen oder kürzesten Lösungswege aufzuzeigen. Sie sollen unter anderem eine Orientierungshilfe bei der Auswahl der Aufgaben durch die Fachlehrerin oder den Fachlehrer sein. Maßgebend für die Korrektur ist allein der Aufgabentext und jede nach diesem Text mögliche Lösung. Sofern in den Aufgaben nach Programmcode gefragt ist, kann über kleinere Syntaxfehler hinweg gesehen werden.

Aufgabe A:

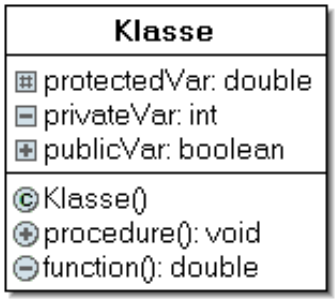
Hinweise für die Fachlehrerin, den Fachlehrer

Programmiersprache

In den Bildungsstandards wird keine Aussage darüber getroffen, in welcher Programmiersprache Algorithmen umgesetzt und Modelle implementiert werden sollen. Wollte man für die schriftliche Abiturprüfung eine oder zwei Programmiersprachen festlegen, müsste dies in einem Abiturerlass drei Jahre vor der Prüfung geschehen, da die Bildungsstandards für drei Kursjahre ausgelegt sind. Zudem stellt sich die (Streit)Frage, welche Sprache dann jeweils vorgegeben werden soll: Delphi, Java, Python, C++ oder eine ganz andere?
Daher wurde auf die Angabe einer Programmiersprache verzichtet und es wird im Aufgabentext an einer Stelle auf die im Kurs eingeführte Programmiersprache verwiesen. Es versteht sich, dass diese Sprache eine objektorientierte Programmierung unterstützen muss.

Klassendiagramme und Beziehungen

Obwohl in den Bildungsstandards nicht erwähnt, wird bei der objektorientierten Modellierung die Kenntnis der Beschreibung von Klassen und deren Beziehungen untereinander gemäß der Unified Modelling Language (UML) vorausgesetzt. Die Beschränkung liegt jedoch auf den Strukturdiagrammen „Klassendiagramm“ und „Objektdiagramm“, insbesondere sind Verhaltensdiagramme (z.B: Sequenzdiagramme) bisher in Baden-Württemberg nicht verpflichtend zu behandeln.



Sichtbarkeit:
```
# - protected
– - private
+ - public
```

Methoden:
```
C – Konstruktor
```

nicht verwendet, stattdessen werden die Rückgabetypen angeben:
```
! - Auftrag, procedure
? - Anfrage, function
```

Laut Duden Informatik werden Beziehungen zwischen Klassen **Assoziationen** genannt. Die **Kardinalität** einer Assoziation gibt an, mit wie viel Objekten der gegenüberliegenden Klasse ein Objekt assoziiert sein kann. Eine **Aggregation** ist ein Spezialfall der Assoziation: Die beteiligten Klassen bilden eine Ganze-Teile-Hierarchie. Objekte einer solchen Klasse bestehen aus einer Menge von Einzelteilen. Eine **Komposition** wiederum ist ein Spezialfall der Aggregation: Die Teile sind vom Ganzen existenzabhängig. In den bayerischen Lehrbüchern wird fast ausschließlich die Assoziation verwendet. In den nördlichen Bundesländern ist teilweise die Sprechweise ist-Beziehung (Vererbung), hat-Beziehung (Komposition) und kennt-Beziehung (Assoziation) eingeführt. In Baden-Württemberg werden lediglich die gerichtete Assoziation und die Vererbung vorausgesetzt.

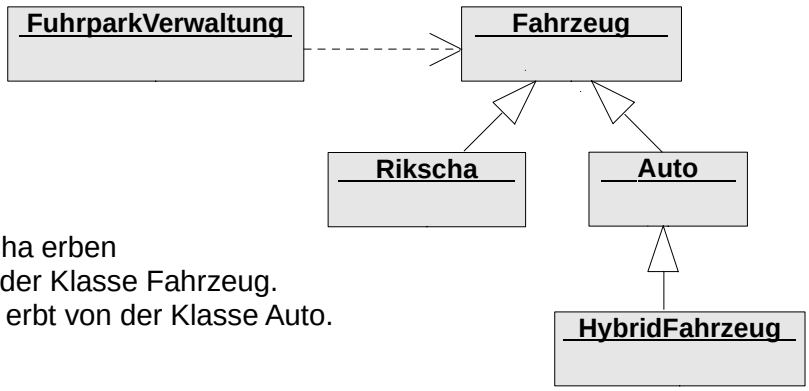
Lösungen

Hinweise:

Die Implementierung von Programmen oder Programmteilen erfolgt jeweils in der im Kurs eingeführten Programmiersprache. Bei der Programmierung ist davon auszugehen, dass sämtliche erforderlichen Daten in einem vernünftigen Rahmen eingegeben werden. An eine Fehlerbehandlung von Falscheingaben ist nicht gedacht.

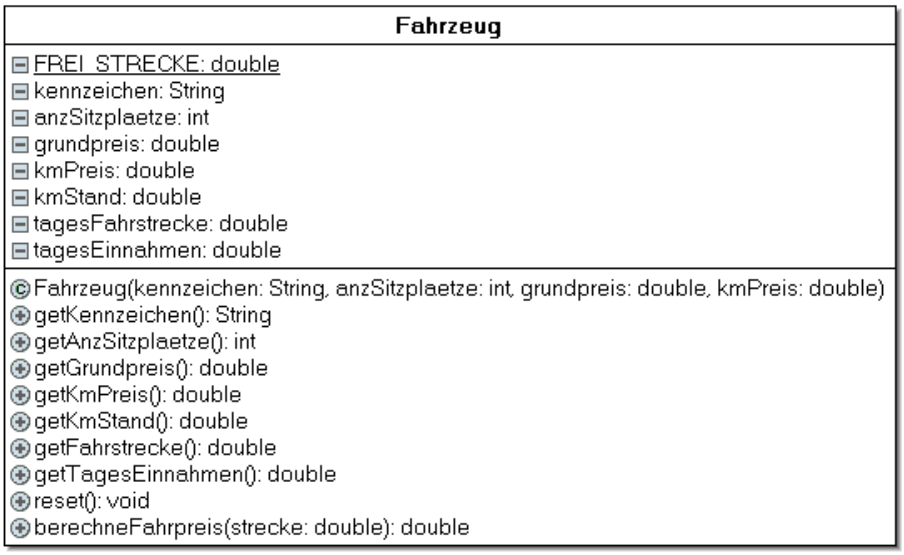
A1

- Klassenbeziehungen
  Assoziation: Die FuhrparkVerwaltung verwaltet eine Reihe von Fahrzeugen.



  Vererbung:
  Die Klassen Auto und Rikscha erben die Attribute und Methoden der Klasse Fahrzeug.
  Die Klasse HybridFahrzeug erbt von der Klasse Auto.

- UML-Klassendiagramm
  für die Klasse Fahrzeug (*enthält bereits Attribute und Methoden für A3*):



- Konstruktor für die Klasse Fahrzeug

```
public Fahrzeug(String kennzeichen, int anzSitzplaetze,
                double grundpreis, double kmPreis) {
  this.kennzeichen = kennzeichen;
  this.anzSitzplaetze = anzSitzplaetze;
  this.grundpreis = grundpreis;
  this.kmPreis = kmPreis;
  kmStand = 0.0;
  tagesFahrstrecke = 0.0;
  tagesEinnahmen = 0.0;
}
```

- Methode zur Berechnung des Fahrpreises

```
public double berechneFahrpreis(double strecke) {
  double preis;
  kmStand += strecke;
```

```
    tagesFahrstrecke += strecke;
    if(strecke <= FREI_STRECKE) {
      preis = grundpreis;
    } else {
      preis = grundpreis + (strecke – FREI_STRECKE)*kmPreis;
    }
    tagesEinnahmen += preis;
    return preis;
  }
```

A2

- Konstruktor für die Klasse Rikscha

```
public Rikscha(String kennzeichen) {
  super(kennzeichen, 3, 2.7, 0.5);
}
```

für die Klasse Auto wurden folgende Instanzvariablen vereinbart:

```
private final double tankVolumen;
private final double verbrauch100km;
private double tankInhalt;
//...weitere Instanzvariablen für A3
```

Konstruktor für die Klasse Auto

```
public Auto(String kennzeichen, int anzSitzplaetze,
            double tankVolumen, double verbrauch100km) {
  super(kennzeichen, anzSitzplaetze, 5.80, 1.60);
  this.tankVolumen = tankVolumen;
  this.verbrauch100km = verbrauch100km;
  this.tankInhalt = 0.0;
  //...weitere Initialisierungen für A3
}
```

- überschriebene Methode zur Berechnung des Fahrpreises für die Klasse Auto

```
public double berechneFahrpreis(double fahrStrecke) {
  double strecke = fahrStrecke;
  double verbrauchteMenge = fahrStrecke* verbrauch100km/100.0;
  if(verbrauchteMenge > tankInhalt) {
    // der Verbleib der Fahrgäste wird nicht behandelt
    strecke = tankInhalt / verbrauch100km * 100;
    tankInhalt = 0.0;
  }
  return super.berechneFahrpreis(strecke);
}
```

- Methode volltanken der Klasse Auto:

```
public double volltanken() {
  double menge = tankVolumen – tankInhalt;
  tankInhalt = tankVolumen;
  return menge;
}
```

A3

- Deklaration der Datenstruktur für die Liste der Fahrzeuge

```
ArrayList<Fahrzeug> fahrzeugListe;
```

Alternativ kann auch der Datentyp `Array` gewählt werden.
An abstrakte Datentypen ist hier nicht gedacht.

- Ermittlung der gesamten gefahrene Strecke aller Fahrzeuge
  Variante A:

```
public double getGesamtStrecke() {
  ListIterator<Fahrzeug> iter = fahrzeugListe.listIterator();
```

```
  double strecke = 0.0;
  while(iter.hasNext()) {
    Fahrzeug fahrzeug = iter.next();
    strecke += fahrzeug.getKmStand();
  }
  return strecke;
}
```

Variante B:

```
public double getGesamtStrecke() {
  double strecke = 0.0;
  for (Fahrzeug fahrzeug : fahrzeugListe) {
    strecke +=  fahrzeug.getKmStand();
  } //end of for
  return strecke;
}
```

- Beschreibung der Programmerweiterungen
  Die Klasse `Fahrzeug` benötigt ein Attribut `tagesEinnahmen`, welches bei der Ermittlung des Fahrpreises aktualisiert wird.
  Die Klasse `Auto` benötigt ein Attribut `tagesVerbrauch`, das ebenfalls bei der Ermittlung des Fahrpreises aktualisiert wird.
  Weiterhin erhält die Klasse `FuhrparkVerwaltung` je eine Methode `getTagesEinnahmen` und `getTagesBenzinVerbrauch`.

- Einsortierung eines neues Fahrzeugs anhand seines Kennzeichens

```
public void insertFahrzeug(Fahrzeug neuesFahrzeug) {
  String neuKennzeichen = neuesFahrzeug.getKennzeichen();
  ListIterator<Fahrzeug> iter = fahrzeugListe.listIterator();
  boolean gefunden = false;
  while(iter.hasNext() && !gefunden) {
    gefunden =
       neuKennzeichen.compareTo(iter.next().getKennzeichen()) < 0;
  }
  if(gefunden) {
    iter.previous();
  }
  iter.add(neuesFahrzeug);
}
```

Alternative Lösung zu A3

- Deklaration der Datenstruktur für die Liste der Fahrzeuge
  `Fahrzeug[] fahrzeugListe;`
  Alternativ kann auch der Datentyp `ArrayList` gewählt werden.
  An abstrakte Datentypen ist hier nicht gedacht.

- Ermittlung der gesamten gefahrene Strecke aller Fahrzeuge

```
public double getGesamtStrecke() {
  double strecke = 0.0;
  for(int i = 0; i < fahrzeugListe.length; i++) {
    strecke += fahrzeugDatenListe[i].getKmStand();
  }
  return strecke;
}
```

- Beschreibung der Programmerweiterungen
  Die Klasse `Fahrzeug` benötigt ein Attribut `tagesEinnahmen`, welches bei der Ermittlung des Fahrpreises aktualisiert wird.
  Die Klasse `Auto` benötigt ein Attribut `tagesVerbrauch`, das ebenfalls bei der Ermittlung des Fahrpreises aktualisiert wird.
  Weiterhin erhält die Klasse `FuhrparkVerwaltung` je eine Methode `getTagesEinnahmen` und `getTagesBenzinVerbrauch`.

- Einsortierung eines neues Fahrzeugs anhand seines Kennzeichens

```
public void insertFahrzeug(Fahrzeug neuesFahrzeug) {
  String neuKennzeichen = neuesFahrzeug.getKennzeichen();
  Fahrzeug[] tempFahrzeugListe =
                          new Fahrzeug[fahrzeugListe.length + 1];
  boolean gefunden = false;
  int pos = 0;
  while(pos < fahrzeugListe.length && !gefunden) {
    gefunden = neuKennzeichen.compareTo(
                        fahrzeugListe[pos].getKennzeichen()) < 0;
    pos++;
  }
  if(gefunden) {
    pos--;
  }
  for(int i = 0; i < tempFahrzeugListe.length; i++) {
    if(i < pos) {
      tempFahrzeugListe[i] = fahrzeugListe[i];
    } else if(i == pos) {
      tempFahrzeugListe[i] = neuesFahrzeug;
    } else {
      tempFahrzeugListe[i] = fahrzeugListe[i-1];
    }
  }
  fahrzeugListe = tempFahrzeugListe;
}
```

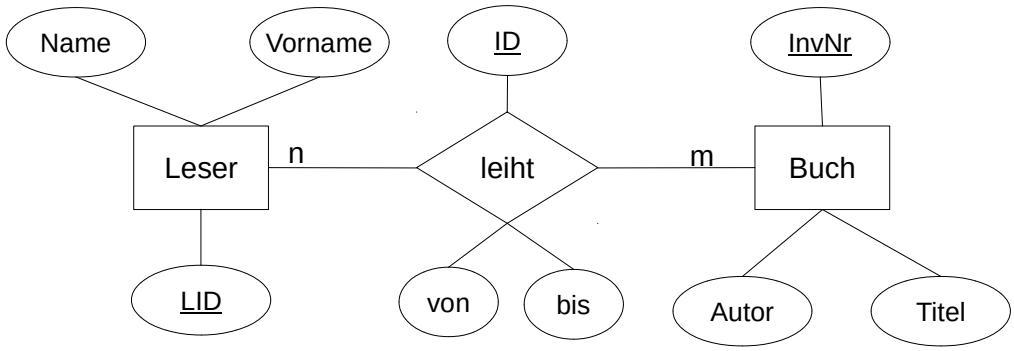
Aufgabe B1:

Hinweise für die Fachlehrerin, den Fachlehrer

Modellierung

Die Modellierung einer Datenbank wird mithilfe eines **Entity-Relationship-Modells** durchgeführt.
Als Beziehungstypen werden 1:1-, 1:n- und n:m-Beziehungen verwendet.
Primärschüssel werden im ER-Diagramm unterstrichen.

Beispiel:



Jeder Leser leiht m Bücher, jedes Buch wird von n Lesern ausgeliehen.

Datenbankschema und Schlüssel

In einem Datenbankschema werden zeilenweise die Namen der Tabellen einer Datenbank aufgelistet. Dem Tabellennamen folgen jeweils in Klammern die Namen der einzelnen Spalten der entsprechenden Tabelle. Im Datenbankschema werden Primärschlüssel unterstrichen und Fremdschlüssel mit einem Pfeil versehen.

Beispiel:
Schueler (SchuelerID, Vorname, Nachname, PLZ↑)
Ort (PLZ , Name)

Normalisierung

Gemeint ist die Herstellung der 3. Normalform. Auf eine darüber hinausgehende Normalisierung wird verzichtet.

SQL-Sprachelemente

Folgende Sprachelemente werden vorausgesetzt:

SELECT (DISTINCT) … FROM
WHERE
ORDER BY
GROUP BY
ASC, DESC
AS
NULL

Vergleichsoperatoren:	=, <>, >, <, >=, <=, LIKE, IS NULL
Arithmetische Operatoren:	+, -, *, /, (…)
Logische Verknüpfungen:	AND, OR, NOT
Funktionen:	COUNT, SUM, MAX, MIN, AVG

SQL-Abfragen können eine oder mehrere Tabellen benutzen.
Die Verknüpfung mehrerer Tabellen mittels JOIN ist nicht vorgesehen.

Lösungen

B1.1 • Unter Redundanz versteht man das mehrmalige Speichern von gleichen Daten. Dies ist in der vorgegebenen Tabelle zu erkennen, da z.B. ein Hersteller mehrere Modelle herstellt und somit Name und Ort mehrfach gespeichert sind.
Bei der Änderung mehrfach gespeicherter Daten kann es zu Fehlern kommen, z.B. bei einem falschem Eintrag des Herstellernamens oder des Ortes.
Die Daten sind dann inkonsistent.

- Ausgangspunkt ist die Tabelle
Fahrzeug(FID, Modell, KW, Preis, Hersteller, Ort, …)

  Modell, KW, Preis und Hersteller hängen von der FID ab.
Der Ort hängt vom Hersteller ab.
Der Ort steht in direkter Beziehung zur PLZ.
Somit ergibt sich :
Fahrzeug(FID, Modell, KW, Preis, HID↑)
Hersteller(HID, HName, PLZ↑)
Ort (PLZ, OName)

B 1.2a SQL-Abfragen:

- Wie viele Mitglieder hat der Sportverein?
```
SELECT count(*)
FROM Mitglied
```
- Welche Mitglieder wohnen in einem Ort, bei dem die Postleitzahl mit „67“ oder „68“ beginnt? Lassen Sie den Namen des Mitglieds und des Ortes ausgeben, aufsteigend sortiert nach den Namen der Mitglieder.
```
SELECT Name, Vorname, Ortsname
FROM Mitglied, Ort
WHERE Mitglied.PLZ = Ort.PLZ
  AND (Ort.PLZ LIKE '68%' OR Ort.PLZ LIKE '67%')
ORDER BY Name
```
- In welchen Sportarten bekommt ein Trainer mindestens 80 Euro?
Geben Sie den Namen des Trainers, die Sportart und die Vergütung an.
```
SELECT Name, Vorname, Bezeichnung, Vergütung
FROM Mitglied, Trainer, Sportart
WHERE Mitglied.MID = Trainer.MID
  AND Sportart.TID = Trainer.TID
  AND Vergütung >= 80
```

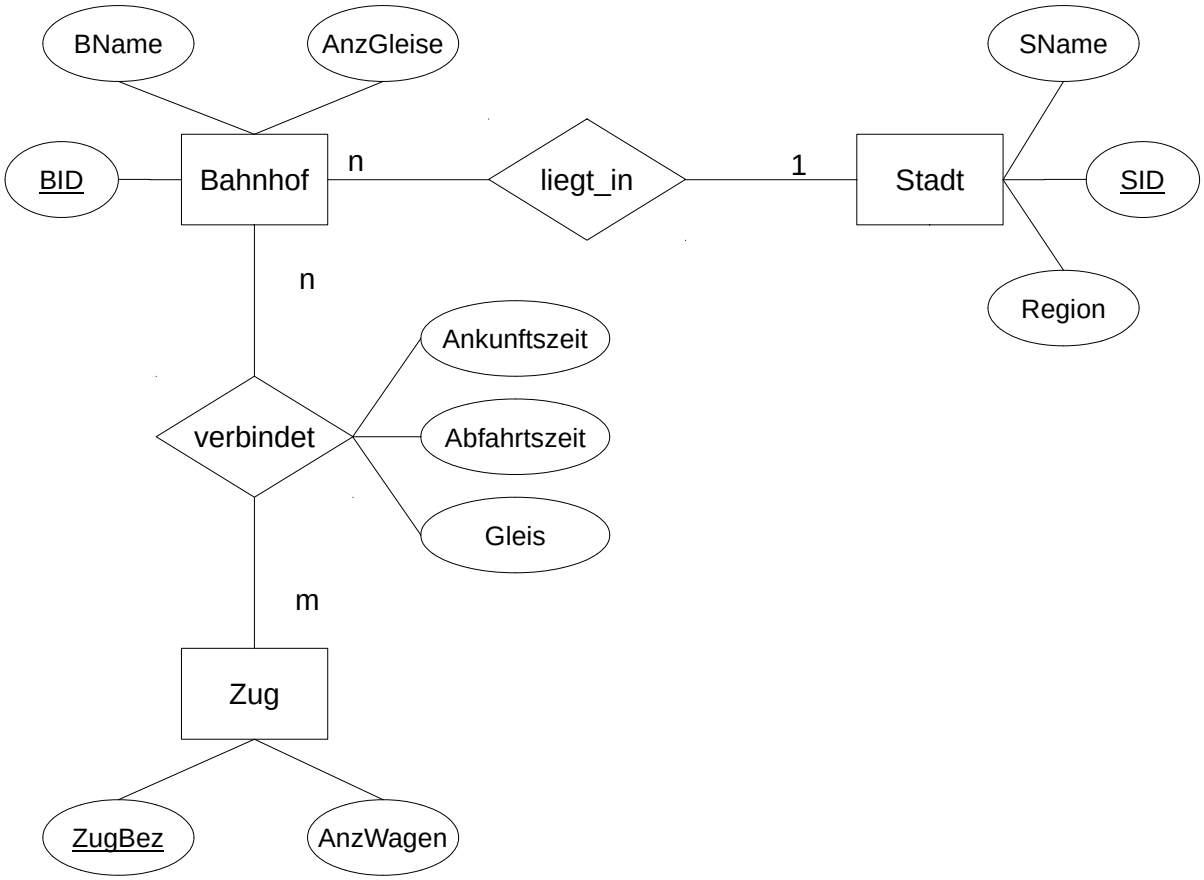
B1.2b
```
SELECT Name, Vorname
 FROM Mitglied
 WHERE Geburtsjahr > (
  SELECT AVG (Geburtsjahr)
  FROM Mitglied
 )
```

Unterabfrage: Hier wird der Mittelwert des Geburtsjahres aller Mitglieder berechnet.
Hauptabfrage: Nun werden die Namen der Mitglieder ausgegeben, die jünger als der Mittelwert des Geburtsjahres sind.

B1.3

- Entitäten und Kardinalitäten:
  - Ein Bahnhof liegt in einer Stadt; in einer Stadt können mehrere Bahnhöfe liegen.
    → 1:n-Beziehung
  - Entität Bahnhof: Name, Anzahl der Gleise
  - Entität Stadt: Name, Region
  - Entität Zug: ZugBez, Anzahl der Wagen
  - Speichern der Verbindungen:
    Es müssen der Zug, die Ankunfts- und Abfahrtszeit und das Gleis festgehalten werden.

  ER-Diagramm:

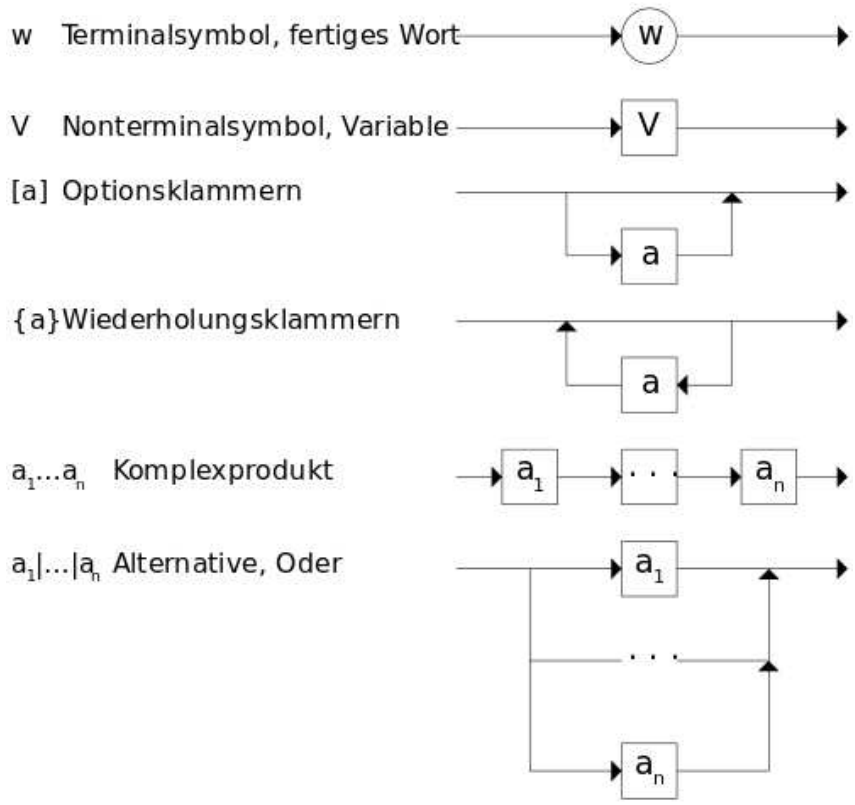


- Datenbankenschema
  Stadt (SID, SName, Region)
  Bahnhof (BID, BName, AnzGleise, SID↑)
  Zug (ZugBez, AnzWagen)
  verbindet (BID↑, ZugBez↑, Abfahrtszeit, Ankunftszeit, Gleis)

Aufgabe B2:

Hinweise für die Fachlehrerin, den Fachlehrer

Syntaxdiagramme:
Syntaxdiagramme (auch „Eisenbahndiagramme“) sind eine andere Art, kontextfreie (Typ 2-) Grammatiken darzustellen.
Terminalsymbole werden dabei in Kreisen, Nichtterminalsymbole in Rechtecken dargestellt. Ein Syntaxdiagramm zeigt, wie ein Nichtterminalsymbol abgeleitet werden kann. Jeder mögliche Pfad von links nach rechts durch das Diagramm entspricht einem Wort, das abgeleitet werden kann. Jedes Konstrukt der erweiterten Backus-Naur-Form kann durch ein entsprechendes Syntaxdiagramm dargestellt werden, siehe Grafik:



http://upload.wikimedia.org/wikipedia/de/thumb/d/df/Syntaxdiagramm.svg/500px-Syntaxdiagramm.svg.png

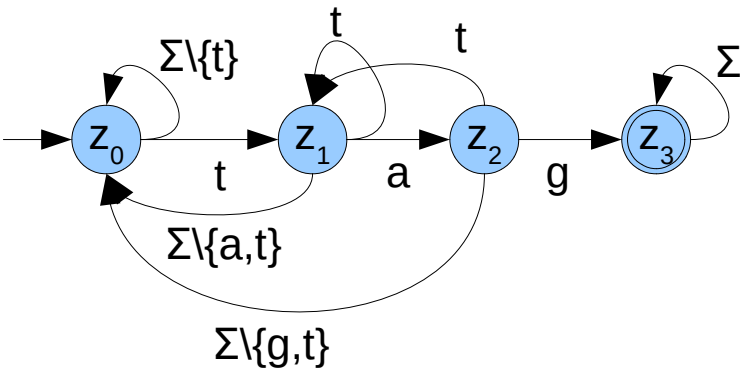
Automaten, formale Sprachen, Grammatik
Es sollte auf jeden Fall eingeführt worden sein, dass Σ die Menge der Terminalsymbole einer Sprache bzw. eines Automaten beschreibt. Die Mengenoperation „A \ B“ wird in der Aufgabenstellung eingeführt, muss daher nicht im Kurs bekanntgemacht werden.
Akzeptable Lösungen sind sowohl eine Aufstellung der Zustandsübergangsfunktion als auch eine graphische Darstellung des erkennenden Automaten, wobei letzteres wohl eher von Schülern gewählt würde.
Die notwendigen Bestandteile einer Grammatik müssen eingeführt worden sein:

- Σ: Menge der Terminalsymbole, Alphabet
- V: Menge der Nichtterminalsymbole, Variablen
- S: Startsymbol aus der Menge der Nichtterminalsymbole
- P: Menge der Produktionsregeln.

Lösungen

B2.1 • L = { abcb, abcc, de, df, d }

B2.2 •

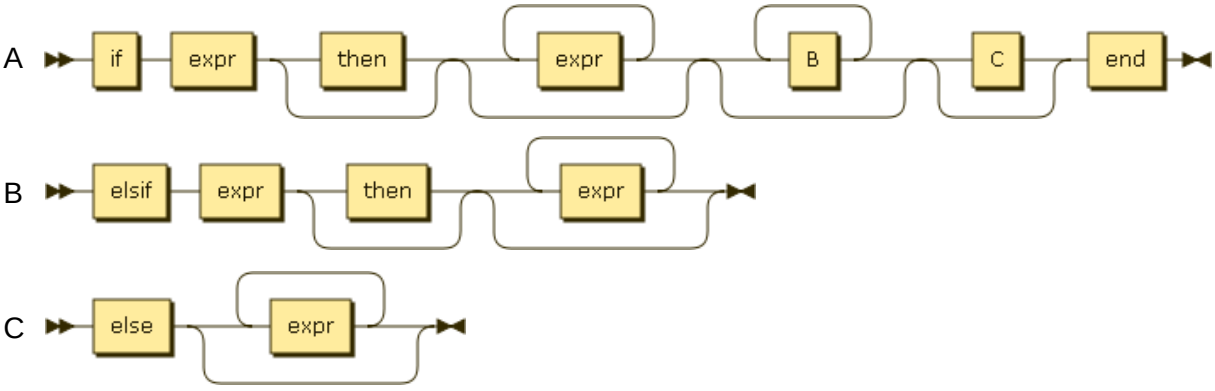


- Der endliche Automat durchläuft die folgenden Zustände
  $z_0 \xrightarrow{m} z_0 \xrightarrow{i} z_0 \xrightarrow{t} z_1 \xrightarrow{t} z_1 \xrightarrow{a} z_2 \xrightarrow{g} z_3 \xrightarrow{s} z_3$
  und endet im Endzustand z_3. Daher akzeptiert er das Wort „mittags“.

- Zum Beispiel:
  Σ = {a – z, ä, ö, ü}, V = { S, A, B }, S ist Startsymbol,
  P: { S → ABA, A → aA | bA | … | zA | äA | öA | üA | ε, B → tag }

B2.3 • Ein Automat erkennt genau dann unendlich viele Wörter, wenn der Graph des Automaten mindestens einen Zyklus enthält, der vom Startzustand aus erreichbar ist, und von dem aus man einen Endzustand erreichen kann.
Wenn ein Automat keine Zyklen enthält, gibt es nur eine endliche Anzahl von Wegen vom Start- zu einem Endzustand. Andernfalls kann ein Zyklus beliebig oft durchlaufen werden, was jedes Mal ein weiteres Wort zur Sprache hinzufügt.

B2.4 • Die Anweisung ist nicht korrekt, da das Terminalsymbol „end“ am Schluss stehen muss. Der Rest ist korrekt.

- 

Aufgabe B3:

Hinweise für die Fachlehrerin, den Fachlehrer

Standardisierte abstrakte Datentypen

Ein abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen[1]. Diese Kapselung legt nahe, ADTs objektorientiert zu behandeln[2] bzw. zu implementieren.
Eine einheitliche Definition der üblichen ADTs gibt es aber leider nicht. In der Literatur finden sich hunderte verschiedene Definitionen der ADTs Liste, Stapel, Schlange und Baum. Selbst die Unterscheidung zwischen einer einfachen Datenstruktur und der Definition eines ADTs ist oft nicht eindeutig, da die Übergänge fließend sind. Dies führt dazu, dass auch die verwendeten Definitionen für das Abitur sich von Bundesland zu Bundesland unterscheiden.

So beinhaltet die Definition der Liste aus NRW einen Zeiger auf ein aktuelles Element, der auf den Anfang oder das Ende der Liste gesetzt werden kann oder in der Liste nach hinten bewegt werden kann. In der Definition des Baumes gibt dieses aktuelle Element aber nicht. In NRW entfällt damit die Operation, auf ein bestimmtes Element der Liste über die Position des Elementes zuzugreifen.

Auch in Bayern gibt es dieses aktuelle Element nicht. Ein Zugriff auf bestimmte Elemente der Liste (oder des Baumes) ist nicht vorgesehen. Zur Verdeutlichung der Struktur gibt es nur eine Methode „AlleAusgeben“. Dafür wird in Bayern mehr Wert auf die Art der Implementierung mit dem Softwaremuster Kompositum (Knoten und Endknoten sind beide Listenelemente, wobei ein Knoten ein weiteres Listenelement als Nachfolger hat, ein Endknoten nicht). Diese Idee erlaubt es, viele Operationen als rekursive Methoden zu implementieren, wobei der Rekursionsabbruch durch unterschiedliche Implementation der Methoden beim Knoten bzw. Endknoten erfolgt.

Es stellt sich daher für das schriftliche Abitur in BW die Frage, was eine geeignete Standard-Definition der ADTs ist und welche Standard-Implementierung diese nach sich zieht.
Anzustrebende Eigenschaften eines gut programmierten ADT und meist auch einer gut spezifizierten Datenstruktur sind:

- Universalität (implementation independence): Der einmal entworfene und implementierte ADT kann in jedes beliebige Programm einbezogen und dort benutzt werden.
- Präzise Beschreibung (precise specification): Die Schnittstelle zwischen Implementierung und Anwendung muss eindeutig und vollständig sein.
- Einfachheit (simplicity): Bei der Anwendung interessiert die innere Realisierung des ADT nicht, der ADT übernimmt seine Repräsentation und Verwaltung im Speicher selbst.
- Kapselung (encapsulation): Die Schnittstelle soll als eine hermetische Grenze aufgefasst werden. Der Anwender soll sehr genau wissen, was ein ADT tut, aber keinesfalls, wie er es tut.
- Geschütztheit (integrity): Der Anwender kann in die interne Struktur der Daten nicht eingreifen. Die Gefahr, Daten ungewollt zu löschen bzw. zu verändern sowie Programmierfehler zu begehen, ist dadurch deutlich herabgesetzt.
- Modularität (modularity): Das modulare Prinzip erlaubt übersichtliches und damit sicheres Programmieren und leichten Austausch von Programmteilen. Bei der Fehlersuche können einzelne Module sehr isoliert betrachtet werden. Viele Verbesserungen können über ADTs nachträglich ohne die geringste Änderung in sämtlichen Umgebungs- bzw. Anwendungsprogrammen übernommen werden.[3]

---

1 Seite „Abstrakter Datentyp“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 20. Oktober 2012, 20:16 UTC. URL: http://de.wikipedia.org/w/index.php?title=Abstrakter_Datentyp&oldid=109551565 (Abgerufen: 2. November 2012, 12:21 UTC)

2 Historisch hat die Idee der ADTs, Daten und die darauf zulässigen Operationen zusammenzufassen, zur Objektorientierung geführt. (www2.cs.uni-paderborn.de/cs/info-cd/vorlesungen/.../kapII.2.ps)

3 Seite „Abstrakter Datentyp“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 20. Oktober 2012, 20:16 UTC. URL: http://de.wikipedia.org/w/index.php?title=Abstrakter_Datentyp&oldid=109551565 (Abgerufen: 2. November 2012, 12:21 UTC)

Bei den linearen Datenstrukturen werden diese Anforderungen von den meisten Implementierungen erfüllt. Liste, Schlange und Stapel lassen sich sowohl als verkettete Liste als auch als Array implementieren. Um die Integrität und die Kapselung nicht zu verletzen, darf dem Anwender nur der Zugriff auf die enthaltenen Daten gewährt werden, aber beispielsweise nicht auf die Knoten der Liste bei einer Implementation als verkettete Liste.
Diese Einschränkung wird bei Bäumen von den meisten Implementationen verletzt. Fast immer gibt es auch in einem sortierten Baum, die Operationen zur Abfrage des linken und des rechten Teilbaums. Der Zugriff auf die Teilbäume erlaubt aber gleichzeitig auch einen Zugriff auf die innere Struktur des Baumes und kann die sortierte Reihenfolge des Baumes zerstören. Allerdings ist es schwer, mit einem Baum zu arbeiten ohne auf die Teilbäume zugreifen zu können.

Ein Ausweg aus diesem Dilemma stellt folgender Ansatz dar: Die Klasse Binärbaum enthält neben den Attributen left, right und content nur die dazugehörigen get/set-Methoden. Diese sind public definiert. Damit erhält man volle Kontrolle über den Binärbaum. Ob man diese Klasse als ADT (es gibt ja ein Verhalten durch die set/get-Methoden) oder nur als Datenstruktur ansieht, ist dabei nicht so entscheidend. Verwendet wird dieser Baum, um ADTs wie einen Suchbaum, AVL-Baum, Heap oder ähnliches zu implementieren. Diese stellen komplexere Operationen (z.B. neues Datenelement einfügen) bereit. Sie liefern aber niemals den Zugriff auf einen Knoten des Binärbaums. Damit ist der zugrunde liegende Binärbaum geschützt. Ob der Baum Daten an den inneren Knoten enthält, ob er mit oder ohne Dummy-Abschlussknoten an den Blättern arbeitet, bleibt der Wahl des Lehrers überlassen. In der graphischen Veranschaulichung einer verketteten Liste oder eines Baumes werden aber keine Abschlussknoten dargestellt. Stellt der Standard-ADT nicht alle für eine Anwendung benötigten Operationen bereit, muss dieser erweitert werden bzw. ein spezieller ADT abgeleitet werden.

Bei der Liste lässt sich diese Vorgehensweise genauso anwenden. Es wird eine Klasse LinkedList mit den Attributen next und content inkl. der dazugehörigen set/get-Methoden erstellt. Diese Klasse kann dann verwendet werden, um einen Stapel, eine Schlange, eine Liste mit positionsbezogenem Zugriff, eine Liste mit Zeiger auf ein Element oder ähnliches zu realisieren. Genauso kann natürlich auch ein Array als Grundlage der Implementationen dieser ADTs eingesetzt werden.

Die Daten sollten in der Regel in einer Klasse gekapselt sein. Der Dateninhalt der Liste, bzw. des Baumes ist vom Typ Object, damit die Universalität des ADTs gewährleistet ist. Für sortierte Datenstrukturen oder falls eine Suchfunktion implementiert werden soll, greifen die Methoden mit einem Typecast auf die Attributwerte der Objekte zu. Damit geht zwar die Universalität verloren. Da in den Bildungsstandards keine sortierten Datenstrukturen gefordert sind, werden diese Datentypen aber ohnehin nur für einen konkreten Anwendungsfall implementiert und nicht als abstrakter Datentyp behandelt.

Im Abitur wird davon ausgegangen, dass die Schüler die rekursiven Datenstrukturen LinkedList und Tree kennen und damit umgehen können. Darauf aufbauend kann ein ADT entworfen, erweitert, implementiert oder analysiert werden. Die Operationen der ADTs Queue (enqueue, dequeue, isEmpty) und Stack (push, pop, isEmpty) müssen den Schülern bekannt sein.

Für die Realisierung der Operationen könnten sowohl iterative wie auch rekursive Implementationen zu Einsatz kommen. Für das Abitur wird vorausgesetzt, dass die Schüler in der Lage sind, beide Varianten umzusetzen.

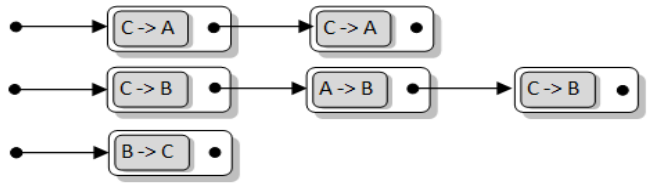
Lösungen:

B3.1

- Eine Schlange arbeitet nach dem FIFO-Prinzip (First-in-first-out), d.h. die zuerst gespeicherten Daten werden als erste aus der Schlange entnommen. Genau dies ist in der Aufgabe gefordert, wenn die Netzneutralität gewahrt bleiben soll.
  Die Schlange kennt nur die Operationen enqueue, dequeue und isEmpty. Diese reichen für eine Routerverwaltung vollkommen aus, da der Router nur neue Pakete zwischenspeichern und das jeweils erste Paket der Schlange abrufen können muss, um es weiter zu versenden.

- Es muss also eine dequeue-Operation ausgeführt werden, um das Paket aus der Warteschlange zu holen und dort zu entfernen.
  Das neue Paket wird mit enqueue(packet) der Warteschlange von Netz A hinzugefügt.

- Schlange A:
  Schlange B:
  Schlange C:



- Die Queue verwaltet die Datenpakete in einer linearen verketteten Liste. Daher muss die Queue das erste Element der verketteten Liste kennen (Assoziation). Eine verkettete Liste lässt sich aufbauen, indem jedes Element der Liste wieder eine Liste oder die leere Liste enthält. Dadurch entsteht eine rekursive Datenstruktur und wird die kennt-Beziehung (Assoziation) der Klasse LinkedList mit sich selbst erklärt.

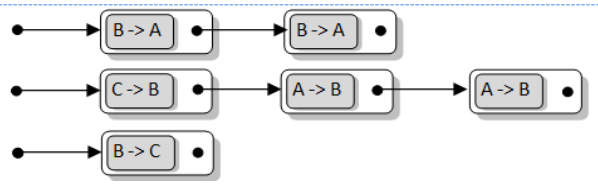
- ```
  public class Queue {
    private LinkedList first = null;
  ..
    public void enqueue(Object d) {
      LinkedList n = new LinkedList(d, null);
      if (first == null) {
        first = n;
      } else {
        LinkedList l = first;
        while (l.getNext() != null) {
          l = l.getNext();
        } // end of while
        l.setNext(n);
      } // end of if
    }
  ..
  }
  ```
  ```

B3.2 •

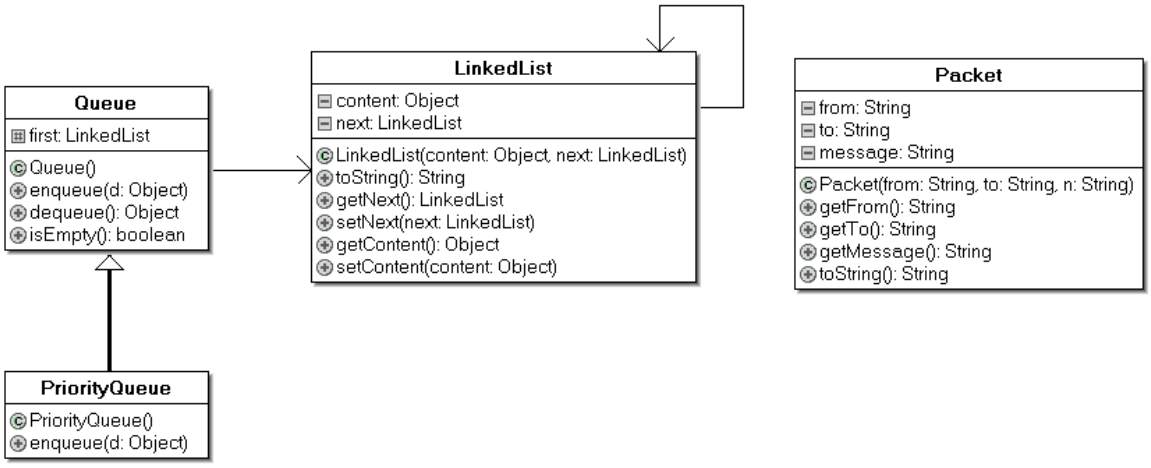
Schlange A:

Schlange B:

Schlange C:



•



Die PriorityQueue benötigt die gleichen Methoden wie die Queue. Nur die Implementation der Methode enqueue unterscheidet sich von der ursprünglichen Form in der Queue. Daher ist es sinnvoll die PriorityQueue von der Queue abzuleiten und die Methode enqueue zu überschreiben. Die PrioriyQueue steht also in einer „ist-Beziehung“ (Vererbung) zur Queue.

B3.3 • Bei einer PriorityQueue liegt die Laufzeit für das Einfügen in O(n), da im schlimmsten Fall die ganze Liste durchsucht werden muss. Auch im Durchschnitt wird die Hälfte der Liste durchsucht. Das Entfernen geht in O(1), da nur das 1. Element der Liste entfernt werden muss, auf das man direkten Zugriff hat.
Beim Heap liegen beide Operationen in O(log n).

Da gleich viele Einfüge- wie Entferne-Operationen auftreten, ist bei größeren Warteschlangen auf jeden Fall der Heap zu bevorzugen, da 2*log n viel kleiner als 1+n ist.
Sind die Warteschlangen sehr kurz, ist es egal, welche Implementierung man verwendet, da keine großen Unterschiede zu erwarten sind.

Operatorenliste für das Fach Informatik

Im zentral gestellten schriftlichen Abitur müssen die Prüfungsaufgaben für die Abiturientinnen und Abiturienten eindeutig hinsichtlich des Arbeitsauftrages und der erwarteten Leistung formuliert sein. Nur dann können die Bewertung und Beurteilung objektiv, gerecht und landesweit vergleichbar erfolgen. Daher werden bei der Formulierung der Arbeitsanweisungen in der Regel nur die hier festgelegten Operatoren benutzt.

Die angegebenen Definitionen und Beispiele sollen die Art der Verwendung und die erwartete Leistung der Schülerinnen und Schüler verdeutlichen. Zusätzlich ist eine Zuordnung zu den Anforderungsbereichen I, II und III der EPA Informatik angegeben, wobei die konkrete Zuordnung auch vom Kontext der Aufgabenstellung abhängt und eine scharfe Trennung der Anforderungsbereiche nicht immer möglich ist.

Die Liste der Operatoren soll einerseits die Lehrerinnen und Lehrer bei der Formulierung von Klausuraufgaben unterstützen, andererseits Schülerinnen und Schülern vermitteln, welche Tätigkeiten und welche Lösungsdarstellung von ihnen erwartet werden. Deshalb müssen die beim Formulieren der Aufgaben verwendeten Operatoren im Unterricht eingeführt und ihr Gebrauch an Beispielen geübt werden.

Operator	**Definition**	**Beispiele**	**Anforderungsbereich**
angeben	Ohne nähere Erläuterungen und Begründungen, ohne einen Lösungsweg zu beschreiben	Geben Sie vier Operationen für den ADT an. Geben Sie die Definition eines endlichen Automaten an.	I
beschreiben	Sachverhalte oder Verfahren in Textform unter Verwendung der Fachsprache in vollständigen Sätzen in eigenen Worten wiedergeben.	Beschreiben Sie den Aufbau eines binären Suchbaums. Beschreiben Sie die Grenzen endlicher Automaten.	I
darstellen	Zusammenhänge, Sachverhalte oder Arbeitsverfahren in strukturierter Form graphisch oder sprachlich wiedergeben.	Stellen Sie Ihr Ergebnis als UML-Klassendiagramm dar.	I – II
erläutern	Einen Sachverhalt nachvollziehbar und verständlich machen.	Erläutern Sie den Nutzen der Methode/Prozedur. Erläutern Sie die Datenstruktur.	I – II
dokumentieren	Einen gegebenen Sachverhalt mit erläuternden Hinweisen versehen.	Dokumentieren Sie die gegebene Klasse. Dokumentieren Sie Ihren Quelltext.	I – II
überführen	Eine Darstellung in eine andere Form bringen.	Überführen Sie das ER-Diagramm in ein Relationenmodell.	I – II
vereinfachen	Die Komplexität eines Sachverhalts nach bekannten Regeln verringern.	Vereinfachen Sie diesen Automatenentwurf.	I – II

bestimmen	Einen Zusammenhang oder einen Lösungsweg aufzeigen und das Ergebnis formulieren.	Bestimmen Sie die Anzahl der rekursiven Aufrufe bis zu dem Ergebnis. Bestimmen Sie die Kardinalitäten.	I – II – III
zeigen	Einen Sachverhalt auf Gesetzmäßigkeiten bzw. kausale Zusammenhänge zurückführen.	Zeigen Sie, dass das Wort aba nicht zur Sprache gehört.	II
begründen	Einen Sachverhalt oder eine Entwurfsentscheidung durch Angabe von Gründen erklären.	Begründen Sie die folgenden Aussagen. Begründen Sie Ihre Entwurfsentscheidung.	II – III
analysieren	Eine konkrete Materialgrundlage unter einer gegebenen Fragestellung auf wichtige Bestandteile, Eigenschaften oder Zusammenhänge untersuchen.	Analysieren Sie die Funktionsweise des Algorithmus. Analysieren Sie die Beziehungen im ER-Diagramm.	II – III
ergänzen / verändern	Eine vorgegebene Problemlösung abwandeln.	Verändern Sie das ER-Modell so, dass … Ergänzen Sie die Syntaxdiagramme. Ergänzen Sie den ADT in geeigneter Weise.	II – III
entwerfen	Nach vorgegebenen Bedingungen ein sinnvolles Konzept selbständig planen / erarbeiten.	Entwerfen Sie eine Datenstruktur. Entwerfen Sie ein ER-Modell. Entwerfen Sie einen Algorithmus.	II – III
modellieren	Zu einem Ausschnitt der Realität ein informatisches Modell anfertigen.	Modellieren Sie das vorgestellte Problem mit Hilfe eines ER-Diagramms.	II – III
implementieren	Datenstrukturen oder Algorithmen in einer Programmiersprache aufschreiben.	Implementieren Sie diesen Algorithmus.	II – III
vergleichen	Nach vorgegebenen oder selbst gewählten Gesichtspunkten Gemeinsamkeiten, Ähnlichkeiten und Unterschiede ermitteln und darstellen.	Vergleichen Sie symmetrische und asymmetrische Verschlüsselung. Vergleichen Sie die iterative mit der rekursiven Lösung.	II – III
beurteilen	Zu einem Sachverhalt ein eigenständiges Urteil unter Verwendung von Fachwissen und Fachmethoden formulieren und begründen.	Beurteilen Sie die gesetzlichen Einschränkungen bei der Verwendung kryptologischer Verfahren. Beurteilen Sie die These .	II – III
Stellung nehmen	Unter Heranziehung relevanter Sachverhalte die eigene Meinung zu einem Problem argumentativ entwickeln und darlegen.	Nehmen Sie bei der Gesundheitskarte bezüglich der Datenschutzproblematik Stellung.	II – III

29. Informatik

29.1 Schriftliche Abiturprüfung

29.1.1 Allgemeine Hinweise

Bearbeitungszeit: **240** Minuten

Hilfsmittel:
- siehe 29.2
- Nachschlagewerke zur deutschen Rechtschreibung

Der Fachlehrerin, dem Fachlehrer werden die Aufgabenstellungen

A: eine Aufgabe mit dem Schwerpunkt **Objektorientierte Modellierung und Programmierung**

sowie drei Aufgaben **B1**, **B2** und **B3** mit verschiedenen Schwerpunkten aus den folgenden Themengebieten vorgelegt:

B1: eine Aufgabe mit dem Schwerpunkt **Datenbanken** inklusive Verschlüsselung und Datenschutz
B2: eine Aufgabe mit dem Schwerpunkt **Automaten und formale Sprachen**
B3: eine Aufgabe mit dem Schwerpunkt **Abstrakte Datentypen** inklusive erweiterter Algorithmen/Rekursion

Die Fachlehrerin, der Fachlehrer wählt aus Gruppe B von den drei vorgelegten Aufgaben **zwei** Aufgaben aus.

Die Schülerin, der Schüler

- bearbeitet die **Aufgabe A** und die **beiden** ausgewählten **Aufgaben aus der Gruppe B;**
- vermerkt auf der Reinschrift, welche Aufgaben sie/er bearbeitet hat;
- ist verpflichtet, die Vollständigkeit der vorgelegten Aufgaben vor Bearbeitungsbeginn zu überprüfen (Anzahl der Blätter, Anlagen usw.).

29.1.2 Gegenstand der schriftlichen Prüfung

Die folgenden Themen des Bildungsplans sind **nicht** Gegenstand der schriftlichen Prüfung:

- Rechner (von-Neumann-Rechner)
- Rechnernetze

29.2 Hilfsmittel

Der an der Schule eingeführte **grafikfähige Taschenrechner (GTR)**.
Hierzu sind die Regelungen und Hinweise des Erlasses des Kultusministeriums vom 30.06.2002 zu beachten (45-6615.31/407).

Vor Prüfungsbeginn ist sicherzustellen, dass nachträglich zugefügte Daten, die nicht zum Lieferumfang oder einem Systemupdate des GTR gehören, entfernt werden. Dies muss durch einen **vollständigen Reset** und ggf. durch **Löschen von zusätzlich installierten Programmen bzw. Applikationen** erfolgen.

Die Arbeit an einem PC ist nicht gestattet.

29.3 Auf die gültigen Einheitlichen Prüfungsanforderungen (EPA) unter http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01_EPA_Informatik.pdf wird verwiesen.

XII. Beurteilungs- und Korrekturrichtlinien für das Fach Informatik

1. Allgemeine Hinweise für die Beurteilung der schriftlichen Prüfungsarbeit

1.1 Korrekturverfahren

Der Erstkorrektor korrigiert mit roter Farbe. Er muss alle Fehler anstreichen und seine Korrekturzeichen auf dem rechten Rand der Schülerarbeiten vermerken.

Der Zweitkorrektor korrigiert mit grüner Farbe. Er hält nochmals sämtliche Fehler auf dem linken Rand der Schülerarbeiten fest. Im Text selbst unterstreicht er diejenigen Stellen, bei denen er vom Erstkorrektor abweicht. Ist der Zweitkorrektor der Ansicht, dass ein vom Erstkorrektor angestrichener Fehler nicht als solcher bzw. mit einem anderen Gewicht zu werten sei, kennzeichnet er auch diese Stelle im Text durch Einklammern und hält dies am linken Rand fest.

Der Endbeurteiler korrigiert mit brauner Farbe (im Übrigen vgl. NGVO, § 21 Abs. 5).

Zur Charakterisierung der verschiedenen Arten von Fehlern sind die unter Ziffer 2 angeführten Abkürzungen zu verwenden.

1.2 Bewertung der Prüfungsarbeiten

Grundlage für die Bewertung der Prüfungsarbeiten ist die Reinschrift. Bietet diese etwas Falsches, der Entwurf aber das Richtige, so ist der Entwurf nur dann zu werten, wenn es sich offensichtlich um einen Übertragungsfehler handelt. Ist die Reinschrift nicht vollständig und enthält der Entwurf die fehlenden Teile in ausgearbeiteter, zusammenhängender Form, so kann der Entwurf anstelle der Reinschrift in die Bewertung einbezogen werden. Dieser Teil des Entwurfs ist zu kennzeichnen. Falls Teile des Entwurfs für die Bewertung herangezogen werden, ist dies in der Reinschrift mit "siehe Entwurf" zu vermerken.

In die Bewertung gehen Leistungen aus dem Kompetenzbereich Kommunikation ein. Erläuternde, kommentierende und begründende Texte, die die Schlüssigkeit der Argumentation belegen, sind unverzichtbare Bestandteile der Prüfungsleistung. Mangelhafte Gliederung, Fehler in der Fachsprache, Ungenauigkeiten in Diagrammen,

Skizzen, Zeichnungen sowie unzureichende oder falsche Bezüge zwischen Diagrammen, Skizzen, Zeichnungen und Text sind als fachliche Fehler zu werten.

Die Verrechnungspunkte für die Bewertung der Teilaufgaben bei vollständiger und richtiger Lösung sind den Prüfungsaufgaben zu entnehmen.

Es dürfen für die Teilaufgaben nur ganze oder halbe Verrechnungspunkte vergeben werden.

Ist die Gesamtsumme aller vergebenen Verrechnungspunkte nicht ganzzahlig, so ist sie auf den nächstgrößeren Verrechnungspunkt aufzurunden und das Ergebnis nach Abschnitt 3 in Notenpunkte umzusetzen.

Schwerwiegende und gehäufte Verstöße gegen die sprachliche Richtigkeit oder gegen die äußere Form führen zu einem Abzug von 1 bis 2 Notenpunkten.

Die erteilten Verrechnungs- bzw. Notenpunkte dürfen nicht in die Schülerarbeit eingetragen werden.

1.3 Lösungshinweise

Die Lösungshinweise erheben nicht den Anspruch, die einzigen oder kürzesten Lösungswege aufzuzeigen. Sie sollen unter anderem eine Orientierungshilfe bei der Auswahl der Aufgaben durch die Fachlehrkraft sein. Maßgebend für die Korrektur ist allein der Aufgabentext und jede nach diesem Text mögliche Lösung.

2. Verwendung von Korrekturzeichen

a) Jedes richtige Teilergebnis ist im Text mit "r", jedes falsche Teilergebnis mit "f" zu kennzeichnen.

b) Fehler werden einmal, grobe Fehler zweimal unterstrichen und am Rand folgendermaßen gekennzeichnet:

D	Denkfehler
Fs	Fachsprache, Fachbegriff fehlt oder wurde falsch verwendet
Log	Logik
R	Rechenfehler
S	Schreibfehler
Syn	Syntaxfehler, z. B. in der verwendeten Programmiersprache
uv	unvollständig
Vz	Vorzeichenfehler

Mit "Schreibfehler" sind solche Fehler gemeint, die offenbar durch falsche Übertragung aus dem Aufgabentext, dem Entwurf, dem vorausgehenden Teil der Lösung und Ähnliches entstanden sind. Weitere Fehler bzw. Unschärfen sind ohne Verwendung von Abkürzungen zu kennzeichnen: zum Beispiel "unscharf", "ab hier unbrauchbar" usw.

c) Hat sich ein Formulierungsfehler, schwerwiegender Verstoß gegen die Fachsprache ("Fs") oder ein Rechenfehler ("R") eingeschlichen, so werden die folgenden Lösungspassagen mit "(r)" gekennzeichnet, sofern die Aufgabe mit dem Fehler "richtig" weitergelöst wurde.

d) Nachlässigkeiten beim Zahlenrechnen (insbesondere beim Auf- und Abrunden) werden einmal unterstrichen und am Rand mit "ungenau" festgehalten.

e) Die Zeichnungen und Diagramme sind durch kurze Bemerkungen auf dem Rand zu beurteilen.

Beispiel: "r";
"r, aber ungenau"

f) Sprachlich formale Mängel sind wie folgt zu kennzeichnen:

Gr	Grammatik
Rs	Rechtschreibung
Z	Zeichensetzung

3. Tabelle der Verrechnungs-/Notenpunkte

Verrechnungspunkte	Notenpunkte	Note
60 - 57 56 - 54 53 - 51	15 14 13	sehr gut
50 - 48 47 - 45 44 - 42	12 11 10	gut
41 - 39 38 - 36 35 - 33	9 8 7	befriedigend
32 - 30 29 - 27 26 - 23	6 5 4	ausreichend
22 - 19 18 - 15 14 - 11	3 2 1	mangelhaft
10 - 0	0	ungenügend