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

Aufgabenfundus
mit Lösungshinweisen

(Fassung vom Februar 2015)

Vorbemerkungen:

Die vorliegende Aufgabensammlung ergänzt den Musteraufgabensatz um weitere acht Aufgaben im Abiturstil. Die insgesamt 12 Aufgaben können jedoch nicht das ganze Spektrum erwartbarer Aufgaben abdecken. Maßgeblich sind der aktuelle Bildungsplan für das vierstündige Fach Informatik und der jeweilige Abiturerlass für das entsprechende Jahr.

http://www.bildung-staerkt-menschen.de/unterstuetzung/schularten/Gym/faecher/Inf/listing_bildungsstandards

Im Jahr 2017 sollen
– Rechner (von-Neumann-Rechner)
– Rechnernetze
nicht Gegenstand der schriftlichen Prüfung sein.

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.).

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. Für eine vereinheitlichende Sprechweise wird auf die einführenden Hinweise in den Lösungen des Musteraufgabensatzes hingewiesen.

Die vorliegenden Aufgaben können und sollen zur Vorbereitung auf die Abiturprüfung im Unterricht und in Kursklausuren eingesetzt werden. Sie sind jedoch auch für Schülerinnen und Schüler im Internet zugänglich.

Allen an der Erstellung der Aufgaben beteiligten Lehrkräften der Versuchsschulen sei an dieser Stelle für die zeitintensive und aufwändige Arbeit an dieser Stelle herzlich gedankt.

Stuttgart, im Februar 2015

I A Objektorientierte Modellierung und Programmierung

Die Firma „Retro-Games“ bekommt den Auftrag ein rundenbasiertes (Zeitrechnung in Jahren, beginnend mit Jahr 1) Strategie-Spiel zu entwickeln.

Hier ein Auszug aus dem Regelheft:
Innerhalb der Spielwelt können Wohnhäuser, Bauernhäuser und Ställe erzeugt werden, wobei Bauernhäuser spezielle Wohnhäuser sind. Wohnhäuser, Bauernhäuser und Ställe werden unter dem Oberbegriff Gebäude zusammengefasst, wobei jedes Gebäude eindeutig eine genaue x- und y-Position hat. Außerdem hat jedes Bauwerk einen Gebäudezustand, fixe Baukosten und einen fixen Grundstückswert, die beide bei der Erstellung eines neuen Bauwerk-Objekts festgelegt und später nicht mehr verändert werden können.

A1
- Stellen Sie die Beziehungen zwischen den Klassen `Bauwerk`, `Wohnhaus`, `Bauernhaus` und `Stall` in einem Klassendiagramm dar. Die Klassen sollen nur den Klassennamen, keine Attribute oder Methoden enthalten. Abstrakte Klassen müssen als solche gekennzeichnet sein.
- Erläutern Sie an diesem Kontext das Grundprinzip der Vererbung und begründen Sie, warum es sinnvoll ist hier Vererbung einzusetzen.

  Ein Bauwerk kann sein Alter (in Jahren) und seinen derzeitigen Wert ausgeben, der sich aus Baukosten, Grundstückswert und Zustand des Gebäudes errechnet.
- Entwerfen Sie ein UML Klassendiagramm für die Klasse `Bauwerk` unter Berücksichtigung des Geheimnisprinzips
- Implementieren Sie für die Klasse `Bauwerk` einen geeigneten Konstruktor mit weiteren Parametern für Baukosten und Grundstückswert.

  Der Zustand eines Gebäudes, dessen Wert zwischen 0% (verfallen) und 100% (Top-Zustand) liegt, verschlechtert sich pro Jahr um 1,5 Prozentpunkte.
- Schreiben Sie für die Klasse `Bauwerk` eine Methode `zustandAktualisieren()`, die einmal pro Jahr im Spiel aufgerufen wird und das Alter und den Zustand des Gebäudes entsprechend anpasst.

  Die Klasse `Bauwerk` enthält das Array `inflation`. In diesem Array sind die prozentualen jährlichen Inflationsraten für die ersten 10 Jahre gespeichert, die die jährliche Preissteigerung abbilden. Danach bleibt die Inflationsrate konstant bei 2,0% pro Jahr.

im Jahr	1	2	3	4	5	6	7	8	9	10
Rate	0,8%	1,0%	1,1%	1,2%	1,5%	1,7%	1,8%	1,9%	1,9%	2,0%

- Implementieren Sie die Methode `wertAusgeben()`, die den aktuellen Kaufpreis eines Bauwerks zurückgibt. Darin geht der Wertverfall ein (also Grundstückswert + Baukosten * Gebäudezustand), zusätzlich muss aber auch auch die seit Spielbeginn aufgelaufene Inflation korrekt berücksichtigt werden.

(12 VP)

A2 In der Klasse `Bauwerk` wird zusätzlich die Methode `abstract istBewohnt() : boolean` definiert. Ein Wohnhaus speichert die Anzahl (>=0) seiner Bewohner. Ein Stall hat keine Bewohner.

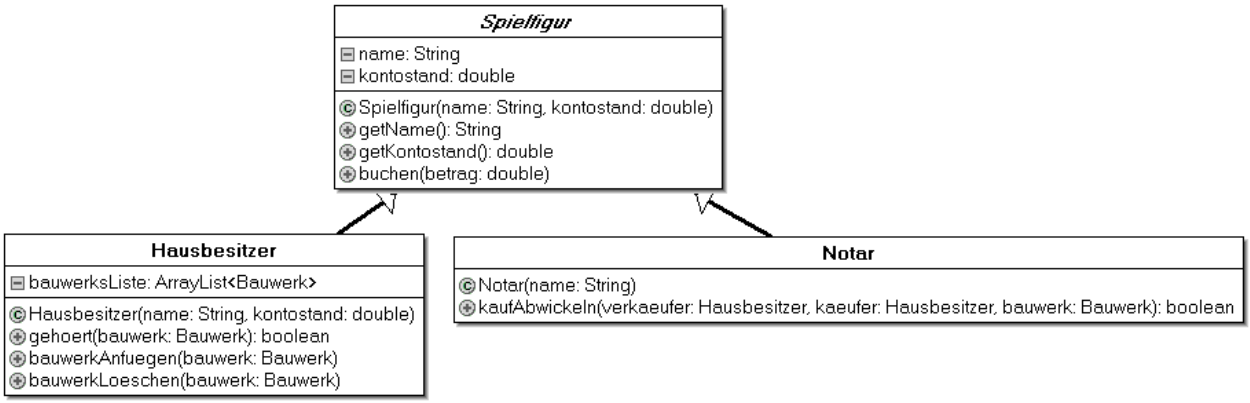
- Überschreiben Sie die Methode `istBewohnt()` für beide Unterklassen sinnvoll.
- Implementieren Sie für die Klasse `Wohnhaus` und `Stall` jeweils den Standardkonstruktor so, dass der Grundstückswert und die Baukosten zur Objektinitialisierung wie folgt festgelegt werden:
  Baukosten für einen Wohnhaus: 200 (Goldstücke),
  Grundstückswert für ein Wohnhaus: 100 (Goldstücke)
  Baukosten für einen Stall: 80 (Goldstücke)
  Grundstückswert für einen Stall: 50 (Goldstücke)

  Zu jedem Bauernhof gehört eine gewisse Anzahl an Ställen.
- Implementieren Sie eine geeignete Datenstruktur, zur Verwaltung der Ställe.

  Zum Wert eines Bauernhauses gehören auch die Werte seiner Ställe.
- Überschreiben Sie die Methode `wertAusgeben():double` in der Klasse `Bauernhaus` so, dass der Wert seiner Ställe mitberücksichtigt wird.

(5 VP)

A3 Im Folgenden sind alle Bauwerke außer den Ställen handelbar. Ställe werden zusammen mit dem jeweils zugehörigen Bauernhaus gehandelt. In der Klasse Bauwerk existiert dafür eine Funktion `abstract istHandelbar():boolean`, die von ihren Unterklassen entsprechend überschrieben wird. Hier ein Ausschnitt aus dem Klassendiagramm (die Liste der Bauwerke ist in Java als `ArrayList` bzw. in Delphi als `TobjectList` realisiert):



Wir betrachten nun zwei Hausbesitzer, die miteinander Handel treiben - einen Kaufinteressenten und einen potentiellen Verkäufer. Ein Notar überwacht den Verkauf. Er achtet darauf, dass das Gebäude tatsächlich handelbar ist, dem Verkäufer gehört und der Käufer über genügend Geldmittel verfügt, um den Preis des Gebäudes auch bezahlen zu können.

- Implementieren Sie die Methoden `kaufAbwickeln():boolean` der Klasse `Notar`, die alle notwendigen Voraussetzungen bei der Rolle des Hausbesitzers als Verkäufer bzw. als Käufer prüft und anschließende den Kauf abwickelt, indem die Besitzer wechseln und der Kaufpreis abgebucht bzw. gut geschrieben wird. Den Erfolg des Handels soll die Methode `kaufAbwickeln()` zurück geben.

(3 VP)

II A Objektorientierte Modellierung und Programmierung

Schulkiosk

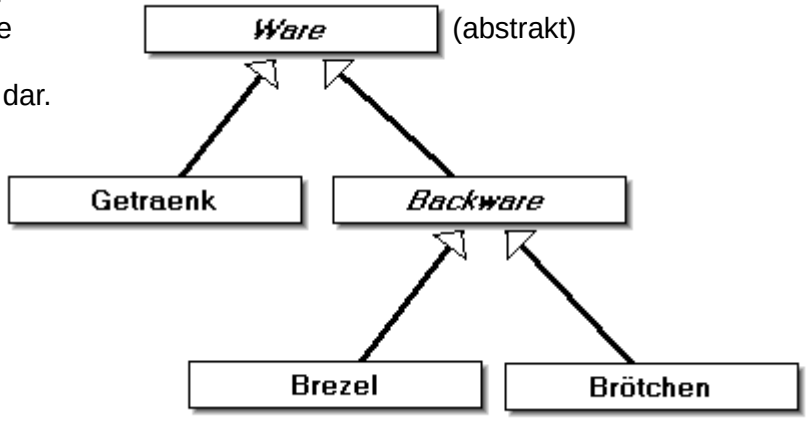
Der Hausmeister des Von-Neumann-Gymnasiums betreibt den Schulkiosk. Er möchte die Verwaltung nun über den Computer abwickeln und steuern. Er kommt mit folgendem Pflichtenheft zu Ihnen und möchte die Kioskverwaltung von Ihnen umsetzen lassen.

Mit dem Kioskverwaltungsprogramm sollen verschieden Waren verwaltet, der Gesamtbestand festgehalten und der Gesamtgewinn berechnet werden. In dem Kiosk gibt es als Waren Getränke und Backwaren. Als Backwaren werden ausschließlich Brezeln und belegte Brötchen verkauft.

Die Waren haben eine Warenbezeichnung, einen Einkaufspreis und einen Verkaufspreis. Getränke haben unterschiedliche Inhaltsgrößen. Für die belegten Brötchen muss die Art des Belags (Kräuter-Aufstrich, Wurst, Käse, …) angegeben werden, bei den Brezeln, ob sie mit Butter bestrichen sind oder nicht.

Die Waren werden über ihre Warenbezeichnung eindeutig festgelegt. Bei der Neuaufnahme einer Ware sollen die Warenbezeichnung, der Einkaufspreis und der Verkaufspreis eingegeben werden. Bei den Getränken wird zusätzlich die Inhaltsgröße und bei den belegten Brötchen die Art des Belags eingegeben.

A1 Das folgende vereinfachte Klassendiagramm stellt die Beziehung zwischen den einzelnen Waren-Klassen dar.



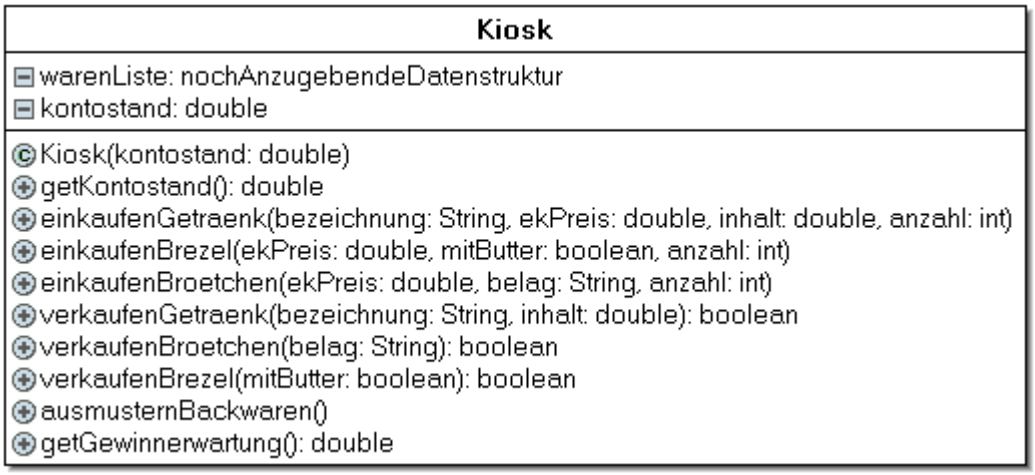
(abstrakt)

- Benennen Sie die Beziehungen zwischen den Klassen und erläutern Sie kurz die objektorientierten Prinzipien.
  Erläutern Sie, warum hier einige Klassen als abstrakt deklariert sind und geben Sie die Gründe für diese Deklaration an.
- Implementieren Sie für die Klassen `Ware`, `Getraenk`, `Backware`, `Brezel` und `Broetchen` jeweils einen Konstruktor.

(5 VP)

(bitte wenden ↪)

A2 Das Kiosk-Verwaltungsprogramm verwaltet den Warenbestand und den aktuellen Kontostand. Beim Einkauf von Waren wird der Kontostand belastet, beim Verkauf wird der Verkaufspreis dem Konto gut geschrieben.



- Deklarieren und initialisieren Sie in der Klasse Kiosk eine geeignete Datenstruktur zur Verwaltung der einzelnen Waren. Im Klassendiagramm finden Sie bereits den Namen des Attributs `warenListe` ohne Datenstruktur.

  Für jeden Warentyp enthält die Klasse `Kiosk` eine Methode, mit der beim Einkauf von einer Anzahl gleicher Artikel der Warenbestand verwaltet wird.

- Implementieren Sie beispielhaft für die Klasse `Broetchen` die Methode `einkaufenBroetchen(...)` mit der im Klassendiagramm angegebenen Parameterliste. Der Verkaufspreis soll dabei um 20% über dem Einkaufspreis liegen. Gleichzeitig soll der Warenbestand des Kiosks aktualisiert werden.

  Der Hausmeister will den zu erwarten Gewinn ermitteln. Dazu vergleicht er Einkaufs- und Verkaufspreis der vorhandenen Waren.

- Implementieren Sie in der Klasse `Kiosk` eine Methode `getGewinnerwartung():double`.

  Am Ende eines Verkaufstages werden die nicht verkauften Gebäckwaren an die lokale Hilfsorganisation „Tafel“ abgegeben.

- Implementieren Sie in der Klasse `Kiosk` eine Methode `ausmusternBackwaren()`, bei der aus der Warenbestandsliste alle Gebäckwaren entfernt werden.
  Sie können davon ausgehen, dass in jeder Klasse eine geeignete Methode `istKlassenTyp(Class cls):boolean` zur Feststellung der Klassenzugehörigkeit existiert.

(12 VP)

A3 Zur Inventur benötigt der Hausmeister eine sortierte Warenliste. Aufgrund der Art und Weise, wie Waren eingekauft und der Warenliste angefügt werden, ist diese Liste meistens schon vorsortiert.

- Welcher Sortieralgorithmus hat in diesem Fall das beste Laufzeitverhalten? Begründen Sie Ihre Antwort.

(3 VP)

I B1 Datenbanken

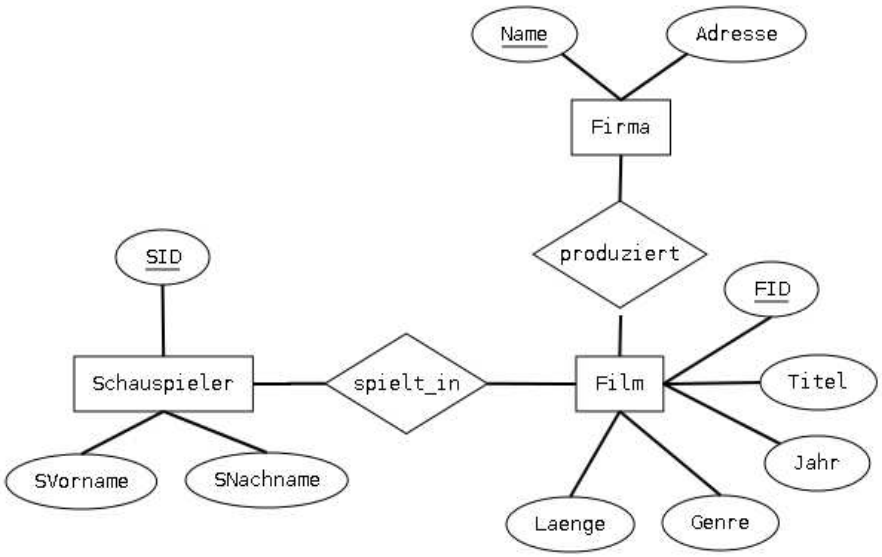
B1.1 Gegeben ist die folgende Tabelle einer Datenbank, mit der ein Elektronik-Händler seine verkauften Smartphones verwaltet:

Hersteller	Bezeichnung	Preis in €	Kaufdatum	Kunde
Tony	Expert3	539,50	10.10.2014	Harry Baumeister, Gmünd
Samson	Merkur4	682,85	22.10.2014	Robert Fruchtig, Semmelrode
Samson	Sun5	456,90	15.09.2014	Maria Lustig, Freiburg
Samson	Sun5	456,90	15.09.2014	Maria Lustig, Freiburg
Pear	xPhone3	889,00	11.10.2014	Harry Baumeister, Gmünd
Samson	Sun5	406,90	14.10.2014	Rita Richter, Freiburg
...	...	...	...	...

- Erläutern Sie, welche Probleme sich bei der Verwaltung der Datenbank ergeben können. Gehen Sie dabei insbesondere auf die Begriffe „Redundanz“ und „Dateninkonsistenz“ ein.
- Entwerfen Sie für die Datenbank ein optimiertes, relationales Datenbankschema, das diese Probleme vermeidet.

(6 VP)

B1.2 Gegeben ist folgendes Entity-Relationship-Diagramm:



- Geben Sie alle Beziehungskardinalitäten an und begründen Sie Ihre Wahl!

Geben Sie jeweils die passende SQL-Abfrage an:

- Welche Filme aus dem Jahr 2012 sind Action-Filme?
- Welche Filme haben eine Länge zwischen 90 und 120 Minuten?
- Wie viele Filme wurden im Jahr 2013 produziert?
- In welchen Filmen wirkt der Schauspieler „Bruce Wally“ mit?
  Lassen Sie nur die Filmtitel geordnet nach dem Produktionsjahr ausgeben.

(7 VP)

B1.3 Für einen Baumarkt soll eine Datenbank entworfen werden, die sowohl den Artikelbestand als auch die Artikelverkäufe erfasst.

Folgende Anforderungen werden an die Datenbank gestellt:

– Zu jedem Artikel werden der Name, der Preis pro Einheit und wie viele Einheiten im Regal sind gespeichert.

– Viele Kunden haben mindestens eine Kundenkarte des Baumarkts.
  Für jeden Kunden sind die jeweilige Kartennummer sowie der Name und die Adresse des Kunden gespeichert.

– Kommt ein Kunde nach dem Einkauf an die Kasse, werden alle Artikel eingescannt und über ihren Barcode erfasst.

– Der Kunde erhält einen Kassenbon, auf dem alle Artikel und die Anzahl der erworbenen Einheiten aufgelistet sind. Diese Daten werden mit der eindeutigen Kassenbon-Nummer des Einkaufs gespeichert.

– Zu jedem Einkauf werden auch Datum und Uhrzeit erfasst.

– Bezahlt der Kunde mit EC-Karte oder bar, werden keine weiteren Daten gespeichert, bezahlt er mit seiner Kundenkarte, wird die Nummer der entsprechenden Karte gespeichert.

- Entwickeln Sie ein Entity-Relationship-Diagramm für die Datenbank!

- Kennzeichnen Sie die Schlüsselattribute und geben Sie die Beziehungskardinalitäten an!

(7 VP)

II B1 Datenbanken

B1.1 Das Verkaufsportal Paybay verkauft DVDs und speichert Datensätze in den Tabellen *Kunden*, *Filme* und *bestellt*:

Kunden:

KID	Name	Vorname	Land
3767	Epli	Horst	CH
4521	Schmidt	Antonia	DE
5488	Müller	Hans	DE

Filme:

FID	Film	Land	Jahr	Preis	Bestand
1000	Avatar	USA	2009	7,99	900
1001	Titanic	USA	1997	4,99	100
1002	Das Boot	DE	1982	6,99	200
1003	Das Wunder von Bern	DE	2003	6,99	100
1004	Monsieur Claude und seine Töchter	FR	2014	9,99	800
1005	Fack ju Göhte	DE	2013	11,99	1500

bestellt:

FID	KID	Zeitpunkt	Anzahl
1001	4521	2014-10-21 10:39:19	1
1001	3767	2014-10-19 14:02:01	2
1002	3767	2014-10-19 14:02:33	1

- Geben Sie eine SQL-Abfrage an, um alle Filme samt Verkaufspreis aufzulisten, die nach dem Jahr 2000 erschienen sind.
- Geben Sie die Ergebnistabelle der SQL-Abfrage
```
SELECT Land, COUNT(*) AS Anzahl
FROM Filme
GROUP BY Land
```
an.
- Geben Sie eine SQL-Abfrage an, die auflistet, welche Filme der Kunde mit der Nummer 3767 im Jahr 2014 bestellt hat.
- Der Manager von Paybay benötigt eine Liste mit dem Gesamtumsatz seines Unternehmens in jedem Land. Geben Sie hierfür eine SQL-Abfrage an.
- Welche Attribute der Tabelle b*estellt* bilden den Primärschlüssel? Begründen Sie Ihre Antwort.

(9 VP)

B1.2 Ein Wahlforschungsinstitut möchte die Ergebnisse, die politische Parteien bei Landtagswahlen erzielt haben, in einer relationalen Datenbank speichern.

– Für jedes Land ist der Name, die Einwohnerzahl, die Anzahl der Sitze im Landesparlament und der Regierungssitz zu speichern. Weiterhin nehmen wir an, dass die Länder durch ihre Namen eindeutig identifiziert sind..

– Jede Partei hat einen voll ausgeschriebenen und einen abgekürzten Namen.

– Folgende bei Landtagswahlen erzielten Ergebnisse werden gespeichert:
- das Jahr, in dem die Wahl stattfand,
- der Stimmenanteil je Partei,
- die Anzahl der Sitze, welche eine Partei erringen konnte.

- Entwerfen Sie ein geeignetes ER-Modell mit passenden Relationen, das die Attribute, die Schlüssel und die Kardinalitäten enthält.
- Welche Zugriffsrechte benötigt ein Mitarbeiter des Instituts, um die Daten nach der nächsten Wahl erfassen zu können?

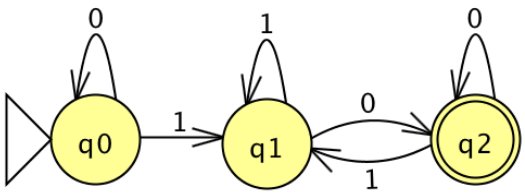
Aus Sicherheitsgründen muss man sich an der Datenbank anmelden, um den Datensatz zu bearbeiten. Das Kennwort darf nur die Kleinbuchstaben a..z enthalten und wird nach Vigenère verschlüsselt auf dem Server abgelegt: Der Schlüssel lautet „abi“.

- Verschlüsseln Sie das Kennwort „wahlurne“ des Mitarbeiters.
- Erklären Sie, wie man einen mit der Vigenère-Chiffre verschlüsselten Text bei bekannter Schlüssellänge vollständig entschlüsseln kann.

(11 VP)

I B2 Automaten und formale Sprachen

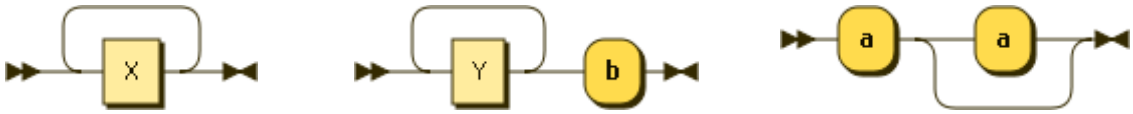
B2.1 Gegeben ist der endliche Automat M mit dem Eingabealphabet $\Sigma = \{0, 1\}$ durch das folgende Zustandsübergangsdiagramm:



- Überprüfen Sie, ob M die Wörter
  i. `1100`
  ii. `1011`
  akzeptiert und geben Sie jeweils die Folge der durchlaufenen Zustände an.
- Geben Sie ein Wort der Länge 5 an, das der Automat nicht akzeptiert.
- Beschreiben Sie die Menge der Wörter, die der Automat akzeptiert.
- Überführen Sie den Automaten in eine reguläre Grammatik, welche die von ihm erkannte Sprache erzeugt.

(5 VP)

B2.2 Gegeben ist das folgende Syntaxdiagramm der Sprache L:



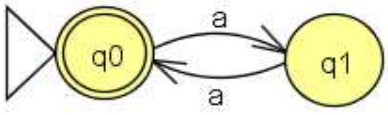
S X Y

- Geben Sie drei Wörter an, die in L liegen.
- Erläutern sie an Hand des Syntaxdiagramme die Begriffe Terminalsymbol und Nichtterminalsymbol.
- Geben Sie an, welche Sprache L dadurch erzeugt wird.
- Begründen Sie, dass Wort „`aabbaa`“ nicht in L liegt.
- Entwerfen Sie einen endlichen Automaten, der genau L akzeptiert.
- Geben Sie eine reguläre Grammatik an, die L erzeugt.

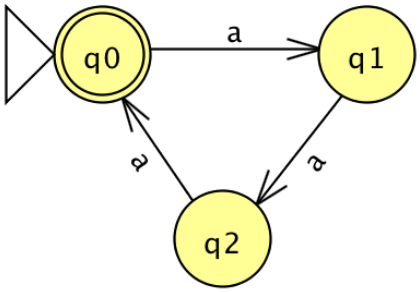
(7 VP)

B2.3 M_1 und M_2 sind zwei endliche Automaten mit dem Eingabealphabet $\Sigma = \{a\}$:

M_1:



M_2:



M_1 erkennt die Sprache L_1 und M_2 erkennt die Sprache L_2.

- Beschreiben Sie die Menge aller Wörter in L_1 und L_2.

  M_3 ist ein Automat, der genau die Wörter erkennt, die sowohl von M_1 als auch von M_2 erkannt werden. Es sei L_3 die von M_3 erkannte Sprache.

- Beschreiben Sie die Menge der Wörter in L_3.
- Entwerfen Sie den Automaten M_3.

(4 VP)

B2.4 Eine Ohrmarke dient der amtlichen Kennzeichnung und Registrierung von Haus- und Nutztieren. Diese besteht aus dem zweistelligen Ländercode `de` gefolgt von dem Bundesland, das durch eine Ziffernfolge `01,02,03,…,15,16` gekennzeichnet ist sowie einer weitere individuellen achtstelligen Zahl, die auch führende Nullen enthalten kann.

- Geben Sie eine reguläre Grammatik (Σ, V, S, P) über dem Terminalsymbolalphabet $\Sigma = \{$`0,1,2,3,4,5,6,7,8,9,d,e`$\}$ an, welche die oben beschriebene Sprache erzeugt.
- Begründen Sie an Hand Ihrer Grammatik, ob sich die folgenden Wörter aus Ihrer Grammatik ableiten lassen:
  i. `de0412345678`
  ii. `de1753198124`
  iii. `de107610358`
- Entwerfen Sie ein den Übergangsgraphen eines deterministischen endlichen Automaten, der die Sprache der korrekten Ohrmarkennummern erkennt.

(4 VP)

II B2 Automaten und formale Sprachen

B2.1 Ein einfacher Fahrscheinautomat kann wie folgt beschrieben werden:

- Eine Fahrkarte für Erwachsene (E) kostet 4.- €, eine Kinderfahrkarte (K) 2.- €.
- In den Automaten können 1.- € und 2.- € Münzen eingeworfen werden.
- Werden mehr Münzen eingeworfen, verbleiben diese zunächst im Automaten.
- Drückt man den Knopf für die Geldrückgabe (R), erhält man alle Münzen zurück.
- Wählt man eine der Fahrkartentasten (E, K), so wird die entsprechende Fahrkarte ausgedruckt und eventuelles Rückgeld ausgeworfen, sofern genügend eingezahlt wurde. Andernfalls passiert nichts.

Dieser Fahrscheinautomat wird durch einen endlichen Automaten modelliert.

- Geben Sie jeweils eine geeignete passende Eingabemenge E und Ausgabemenge A an.
- Welche Informationen über zurückliegende Handlungen, muss der Automat speichern, um korrekt zu reagieren? In welcher Form geschieht dies im Modell?

  Am Anfang (z. B. nach dem Einschalten) ist der Automat in einem definierten Anfangszustand z_0. Dieser ist zugleich auch ein Endzustand. Wenn man darauf verzichtet, die Art der eingeworfenen Münzen zu speichern, kann die Zustandsmenge $S = \{z_0, z_1, z_2, z_3, z_4, z_5\}$ mit den weiteren Zuständen z_1 bis z_5 wie folgt definiert werden:

  z_0: kein Geld eingeworfen
  z_1: 1.- € eingeworfen
  z_2: 2.- € eingeworfen
  z_3: 3.- € eingeworfen
  z_4: 4.- € eingeworfen
  z_5: um 1.- € überzahlt (weitere Überzahlungen sollen nicht möglich sein)

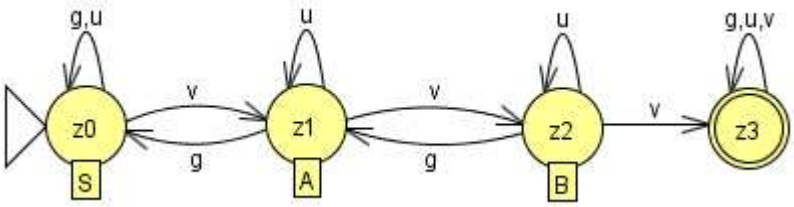
  Jedem Zustand $s \in S$ und jeder Eingabe $e \in E$ wird durch die Übergangsfunktion δ ein neuer Zustand $s' \in S$ zugeordnet.

- Geben Sie die Übergangsfunktion δ in Form einer Tabelle an.
- Zeichnen Sie das vollständige Zustandsübergangsdiagramm für diesen Fahrscheinautomaten, das auch die Ausgaben enthält.
- Begründen Sie, warum es sich bei diesem Modell eines Fahrkartenautomaten um einen deterministischen endlichen Automaten handelt.

(10 VP)

(bitte wenden ↳)

B2.1 Der Vorstand einer Fußballvereins möchte jeglichen Diskussionen über eine mögliche Trainerentlassung nach verlorenen Spielen entgegenwirken. Er hat daher ein Regelwerk aufgestellt, aus dem eindeutig hervorgeht, unter welchen Bedingungen ein Trainer entlassen wird. Der folgende Übergangsgraph eines endlichen Automaten modelliert dieses Regelwerk. Das Eingabealphabet ist $\Sigma = \{g, u, v\}$. (g: gewonnen, u: unentschieden, v: verloren)



- Geben Sie eine Folge von Zeichen aus dem Eingabealphabet Σ der Länge 10 an, die von dem Automaten akzeptiert wird.
- Erläutern Sie anhand des Übergangsgraphen, unter welche Bedingungen der Trainer entlassen wird.
- Entwickeln Sie eine Grammatik, die genau diejenigen Wörter erzeugt, die von dem obigen Automaten akzeptiert werden.
  Bestimmen sie für das Wort vvguvv die Ableitung aus dieser Grammatik.

Der Vereinsvorstand musste zurücktreten. Der neue Vorstand möchte ein neues Regelwerk für eine Trainersuspendierung einführen. Man ist sich einig, dass jeder der beiden folgenden Umstände zu einer Trainerentlassung führt.

- Das Team hat dreimal hintereinander verloren.
- Das Team hat viermal hintereinander nicht gewonnen.

- Geben Sie zu jedem der beiden Fälle eine Spielergebnisfolge aus {g, u, v] an, die zu einer Trainerentlassung nach 6 Spielen führt.
- Entwerfen Sie den Übergangsgraphen eines deterministischen endlichen Automaten, der das Eingabealphabet $\Sigma = \{g, u, v]$ besitzt und genau die Folgen von Spielausgängen akzeptiert, die zu einer Trainerentlassung nach dem neuen Regelwerk führen?
- Geben Sie für die Abarbeitung der folgenden Eingaben die zugehörigen Zustandsfolgen an und geben Sie begründet an, ob der Automat jeweils die Eingabe akzeptiert oder nicht.

  i: gugvvv

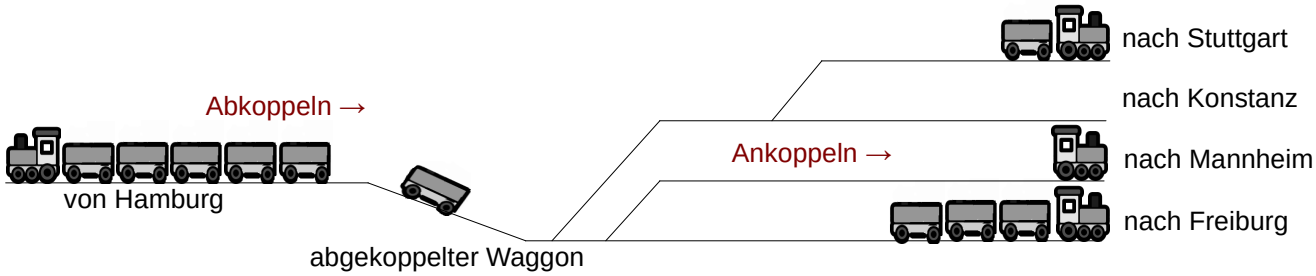
  ii: uvugvvu

(10 VP)

I B3 Abstrakte Datentypen

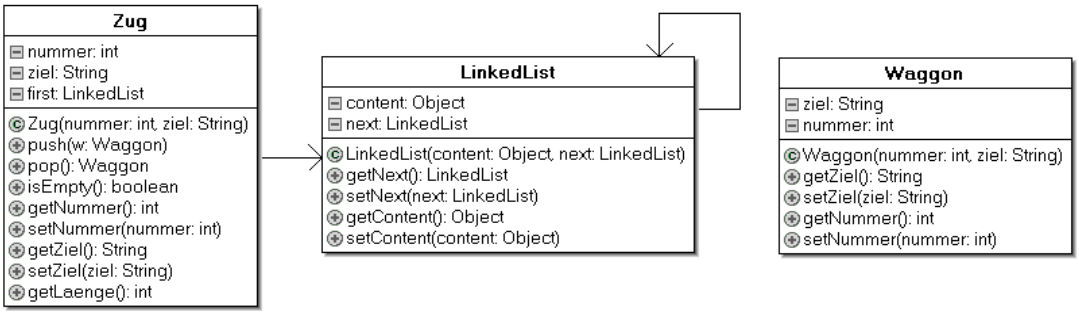
„Kornwestheim Rangierbahnhof 20070713“ von Rosenzweig. Lizenziert unter Creative Commons 3.0 über Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Kornwestheim_Rangierbahnhof_20070713.jpg

Im Güterverkehr der Bahn müssen die Waggons der Güterzüge immer wieder zu neuen Zügen zusammengestellt werden, so dass alle Waggons das gleiche Ziel haben. In Kornwestheim gibt es dafür einen großen Rangierbahnhof. Ankommenden Güterzügen wird dort von hinten betrachtet Waggon für Waggon abgehängt. Diese rollen dann über einen leicht geneigten Hang durch Weichen gesteuert von hinten an Züge heran, die zur Abfahrt auf verschiedenen Gleisen bereitstehen, und werden dort automatisch angekoppelt.



Für den Rangierbahnhof soll eine Software entwickelt werden, mit der diese Prozesse automatisiert werden können. Dazu müssen die Daten der Züge und der Gleise modelliert werden. Dabei hat jeder Zug eine Zugnummer und ein Ziel. Auch die Waggons haben Nummern und Ziele. Ist das Ziel eines Zuges ein Rangierbahnhof, werden die Waggons dort ihrem Ziel entsprechend auf neue Züge verteilt.
Bei den Gleisen sind die Weichen zu modellieren. Jede Weiche kann entweder auf „gerade“ oder auf „abbiegen“ gestellt sein.

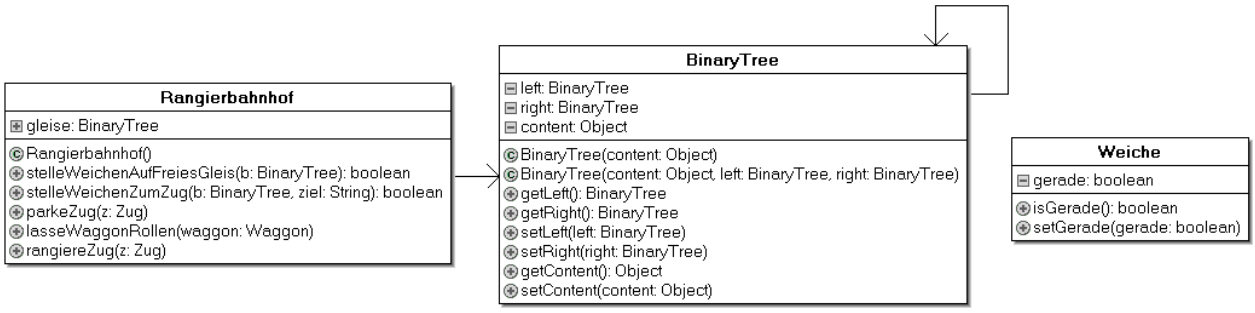
B3.1 Der Softwareentwickler für dieses Projekt schlägt vor, die Züge auf Grundlage des ADT Stapel (Stack) zu implementieren und für die Gleise die Datenstruktur Binärbaum zu verwenden. Das folgende UML-Diagramm stellt die dafür verwendeten Klassen dar:



- Beschreiben Sie die grundlegenden Eigenschaften eines Stapels und eines Binärbaums und erläutern Sie, warum der Vorschlag des Entwicklers sinnvoll ist.
  Erläutern Sie anschließend, wie Anhängen und Abkoppeln eines Waggons an einen Zug realisiert werden können.
- Die Methode `getLaenge():int` der Klasse Zug soll die Anzahl der Waggons zurückliefern. Implementieren Sie diese Methode auf der Basis des dargestellten Klassendiagramms. Sie können davon ausgehen, dass alle anderen Methoden schon korrekt implementiert sind.

(6 VP)

B3.2 Die Gleise des Rangierbahnhofs werden als Binärbaum gespeichert. Jeder innere Knoten enthält eine Weiche als „content“. An den Blättern können Zug-Objekte (oder nichts) gespeichert sein.



Die Implementation der Klasse Rangierbahnhof beruht auf folgendem Pseudocode.
Er beschreibt einen Ausschnitt des Konstruktors sowie den Algorithmus einer Methode, die die Weichen so stellt, dass sie zu einem freien Gleis führen:

```
t1 = neuer BinaryTree(leer)
t2 = neuer BinaryTree(leer)
t3 = neuer BinaryTree(leer)
t4 = neuer BinaryTree(leer)
t5 = neuer BinaryTree(leer)

t6 = neuer BinaryTree(neue Weiche(), t1, t2)
t7 = neuer BinaryTree(neue Weiche(), t3, t4)
t8 = neuer BinaryTree(neue Weiche(), t7, t5)
gleise = neuer BinaryTree(neue Weiche(), t8, t6)
...
```

Methode stelleWeichenAufFreiesGleis(b:BinaryTree):boolean

```
wenn (b.getLeft() == leer)
  wenn (b.getContent() == leer)
    gib wahr zurück
  sonst
    gib falsch zurück
  ende wenn
sonst
  w = (Weiche) b.getContent()
  wenn (stelleWeichenAufFreiesGleis(b.getRight()) == wahr)
    w.setGerade(wahr)
    gib wahr zurück
  sonst
    wenn (stelleWeichenAufFreiesGleis(b.getLeft()) == wahr)
      w.setGerade(falsch)
      gib wahr zurück
    sonst
      gib falsch zurück
    ende wenn
  ende wenn
ende wenn
```

- Zeichnen Sie die Gleisanlage auf, die durch diesen Konstruktor modelliert wird.
- Erläutern Sie, wie die Methode `stelleWeichenAufFreiesGleis(..)` dafür sorgt, dass die Weichen so gestellt werden, dass sie zu einem freien Gleis führen (um eine neue Lok dort zu platzieren). Erläutern Sie, was passiert, wenn es kein freies Gleis mehr gibt.
- Implementieren Sie die Methode `parkeZug(z:Zug)`, die dafür sorgt, dass der Zug auf dem Gleis (d.h. Blatt des Gleis-Baumes) als content gespeichert wird, das durch die aktuelle Weichenstellung vorgegeben ist.
- Untersuchen Sie, welches Laufzeitverhalten für die Methode `parkeZug(..)` im schlimmsten Fall bei n Gleisen zu erwarten ist. Untersuchen Sie, welchen Einfluss die Struktur der Gleisanlage auf Ihre Aussage hat.

(11 VP)

B3.3 Die Methode `stelleWeichenZumZug(b:BinaryTree; ziel:String): boolean` stellt die Weichen so, dass ein Zug mit dem angegeben Ziel erreicht wird. Gibt es keinen derartigen Zug, gibt die Methode false zurück.
Die Methode `lasseWaggonRollen()` hängt einen Waggon an den Zug an, der durch die aktuelle Weichenstellung erreicht wird

- Implementieren Sie die Methode `rangiereZug(z:Zug)`, die einen ganzen Zug abarbeitet und jeden Wagen des Zuges an den richtigen weiterführenden Zug anhängt. Sie können davon ausgehen, dass es für jedes Ziel eines Waggons einen entsprechenden Zug im Rangierbahnhof gibt (d.h. `stelleWeichenZumZug(..)` findet immer einen passende Weichenstellung).

(3 VP)

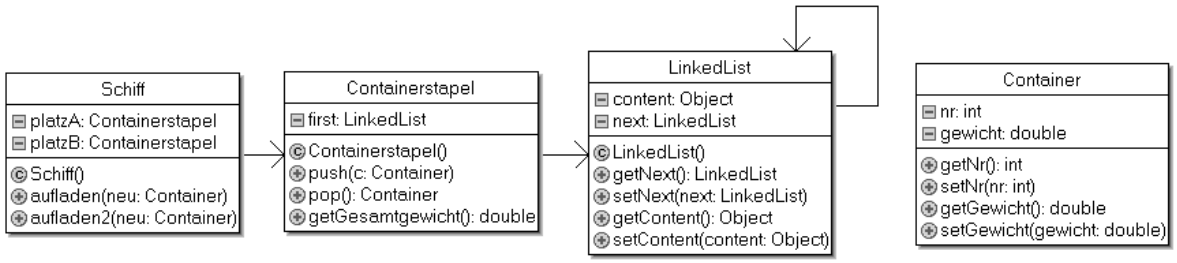
II B3 Abstrakte Datentypen

Ein Containerschiff ist ein speziell für den Transport von Containern gebautes Frachtschiff. Die Container sind genormte Behälter, die auf vorgesehenen Plätzen aufeinander gestapelt werden. Ein besonderer Schiffstyp, das sogenannte Feederschiff, hat nur eine geringe Anzahl von Plätzen zur Aufnahme von Containern und kann somit auch kleine Häfen anlaufen.

"Ship Aurora" by Tvabutzku1234 - Own work. Licensed under CC0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Ship_Aurora.jpg#mediaviewer/File:Ship_Aurora.jpg

Eine Reederei besitzt mehrere Feederschiffe mit jeweils genau zwei Stapelplätzen. Die Plätze werden auf jedem Schiff mit platzA und platzB bezeichnet. Jeder Platz kann genau einen Containerstapel aufnehmen. Ein Container auf einem solchen Stapel darf nicht mehr als 28 t wiegen.

B3.1 Die Reederei möchte die Frachten der Feederschiffe mit einem objektorientierten Computerprogramm verwalten. Das folgende Diagramm zeigt die beteiligten Klassen. Die Klasse Containerstapel stellt dabei eine Erweiterung des ADT Stapel dar:



- Erläutern Sie, welche Auswirkungen die Methoden push und pop auf einen Stapel haben. Begründen Sie, warum für diese Aufgabenstellung die Modellierung mit einem Stapel sinnvoll ist.
- Die Klasse Containerstapel besitzt die Methode `getGesamtgewicht():double`. Implementieren Sie diese Methode in der im Unterricht verwendeten Programmiersprache. Beachten Sie dabei, dass hierfür keine zusätzlichen Attribute in die Klasse Containerstapel eingefügt werden dürfen..

(6 VP)

B3.2 Das Schiff soll möglichst ausgeglichen beladen werden. Aus diesem Grund werden neue Container immer auf dem leichteren Stapel abgelegt, bei gleich schweren Stapeln auf platzA.

- Implementieren Sie die Methode `aufladen(neu:Container)` in der Klasse Schiff, die den übergebenen Container jeweils auf dem bisher leichteren der beiden Stapel ablegt, bzw. auf platzB, wenn beide Stapel gleich schwer sind.

(4 VP)

B3.3 Eine experimentelle Lademethode `aufladen2(neu:Container)` beruht auf dem folgenden Algorithmus, hier notiert in Pseudocode. Er soll nur dann angewendet werden, wenn auf beiden Plätzen mindestens ein Container liegt. Die Prüfung, ob beide Stapel Container enthalten, ist hier nicht vorgesehen.

Algorithmus aufladen2(neu:Container)

```
a = platzA.getGesamtgewicht();
b = platzB.getGesamtgewicht();
c = neu.getGewicht();
wenn (a < b)
  wenn (a + c > b + 20)
    platzA.push(platzB.pop());
    platzB.push(neu);
  sonst
    platzA.push(neu);
  ende wenn
sonst
  wenn (b + c > a + 20)
    platzB.push(platzA.pop());
    platzA.push(neu);
  sonst
    platzB.push(neu);
  ende wenn
ende wenn
```

Bei einem Ladevorgang befinden sich zu Beginn auf platzA und platzB je ein Container mit 10 t. Es sollen nun drei Container mit den Gewichten in der Reihenfolge 21 t, 27 t und 20 t geladen werden.

- Stellen Sie das Ergebnis des Ladevorgangs in einer Skizze dar.
- Analysieren Sie, welche Vorteile dieser Algorithmus gegenüber Ihrem Algorithmus aus Aufgabe B3.2 bietet.
- Beurteilen Sie, ob die experimentelle Lademethode in den Produktivbetrieb übernommen werden sollte.

(6 VP)

B3.4 Zur Vorbereitung des Ladevorgangs wird eine Ladeliste erstellt, in der die Container – bezogen auf die Zielhäfen – in umgekehrter Reihenfolge notiert werden, damit die Container, die zuerst ausgeladen werden müssen, ganz oben stehen.
Neu hinzu kommende Container werden an der richtigen Stelle in die Liste eingefügt. Dafür bietet eine verbesserte Containerstapel-Klasse eine Methode `einfuegenAnKorrekterPosition(c:Container)` an.

- Beschreiben Sie, wie innerhalb der Methode die Verkettung der Objekte verändert werden muss, um ein neues Objekt innerhalb einer verketteten Liste einzufügen. Veranschaulichen Sie die Beschreibung unter Verwendung einer aussagekräftigen Skizze.

(4 VP)

Lösungshinweise

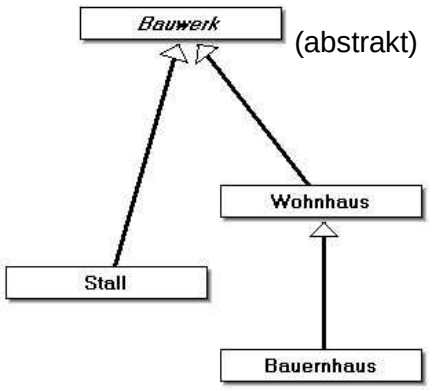
Für die Fachlehrerin, den Fachlehrer

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

Aufgabe I A:

A1

- Klassendiagramm (1 VP)

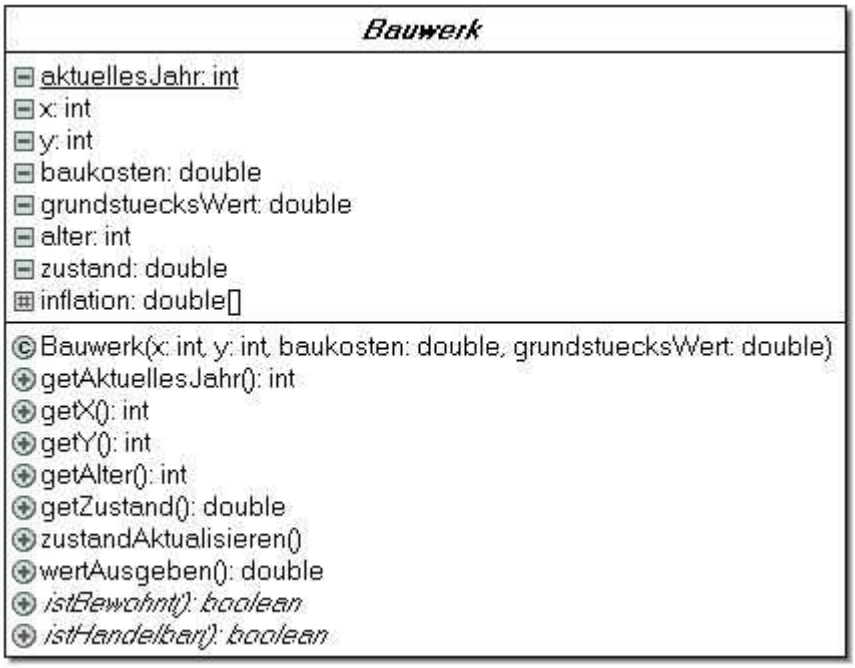


- Grundprinzip der Vererbung (1 VP)
  Prinzip der Vererbung ist die Generalisierung bzw. Spezialisierung. Allgemeine Attribute, die in mehreren Klassen zu finden sind, werden in einer Oberklasse zusammengefasst. Je weiter man im Vererbungsbaum nach unten geht, desto spezieller werden die Klassen in ihren Attributen und Methoden.

  So ist beispielsweise ein Bauernhaus ein Wohnhaus und erbt automatisch alle (nicht-privaten) Eigenschaften und Fähigkeiten des Wohnhauses und kann diese bei Bedarf auch überschreiben.

  Ein weiterer Vorteil der Vererbung ist die Wiederverwendbarkeit von Klassen und die Vermeidung von redundantem Code.

- UML-Klassendiagramm: (3 VP)
  für die Klasse `Bauwerk` (*enthält bereits Attribute und Methoden für A2 und A3*):



- Konstruktor für die Klasse Bauwerk (2 VP)

```
public Bauwerk(int x,int y,double baukosten,
                              double grundstuecksWert) {
  this.x = x;
  this.y = y;
  this.baukosten = baukosten;
  this.grundstuecksWert = grundstuecksWert;
  alter = 0;
  zustand = 100.0;
 }
```

- Methode zur jährlichen Aktualisierung (2 VP)

```
public void zustandAktualisieren() {
  alter++;
  zustand = zustand*0.985;
 }
```

- Methode zur Wertberechnung (3 VP)
  Die Klasse `Bauwerk` enthält das Array `inflation` mit vorgegebenen Prozentwerten:

```
protected double[] inflation = {0.0, 0.8, 1.0, 1.1, 1.2,
                                1.5, 1.7, 1,8, 1.9, 1.9, 2.0};
public double wertAusgeben() {
  double wert = grundstuecksWert + baukosten * zustand/100.0;
  int jahrDerFertigstellung = getAktuellesJahr()-getAlter();
  int inflationBerechnetBis = jahrDerFertigstellung;
  while(inflationBerechnetBis <= getAktuellesJahr()) {
    if(inflationBerechnetBis <= 10) {
      wert = wert*(1+inflation[inflationBerechnetBis]/100.0);
    } else {
      wert = wert * 1.02 ;
    }
    inflationBerechnetBis ++ ;
  }
  return wert;
}
```

A2

- überschriebene Methoden istBewohnt() (1 VP)

für das Wohnhaus:

```
public boolean istBewohnt(){
  return anzBewohner > 0;
}
```

für den Stall:

```
public boolean istBewohnt(){
 return false;
}
```

- Standardkonstruktoren (2 VP)

```
public Wohnhaus(int x,int y){
  super(x, y, 200.0, 100.0);
  anzBewohner = 0;
}
```

```
public Stall(int x,int y) {
  super(x, y, 80.0, 50.0);
}
```

- überschriebene Methode zur Wertberechnung eines Bauernhauses (2 VP)

  Die Ställe werden in einer ArrayList verwaltet:

```
private ArrayList<Stall> stallListe;
public double wertAusgeben() {
  double wert = super.wertAusgeben();
  for(Stall stall : stallListe) {
    wert = wert + stall.wertAusgeben();
  }
  return wert;
}
```

A3 - Kaufabwicklung (3 VP)

```
public boolean kaufAbwickeln(Hausbesitzer verkaeufer,
                    Hausbesitzer kaeufer, Bauwerk bauwerk) {
  boolean erfolgreich = false;
  double wert = bauwerk.wertAusgeben();
    if(kaeufer.getKontostand() >= wert
            && verkaeufer.gehoert(bauwerk)
              && !kaeufer.gehoert(bauwerk)
                && bauwerk.istHandelbar()) {
      verkaeufer.bauwerkLoeschen(bauwerk);
      kaeufer.bauwerkAnfuegen(bauwerk);
      verkaeufer.buchen(wert);
      kaeufer.buchen(-wert);
      erfolgreich = true;
    }
  return erfolgreich;
}
```

Aufgabe II A:

A1 - Erläuterung des Klassendiagramms (2,5 VP)

`Broetchen` und `Brezel` erben von der Klasse `Backware`. Diese und die Klasse `Getraenk` erben von der Klasse `Ware`. Die Klassen `Ware` und `Backware` sind abstrakte Klassen, da sie nur allgemeine Attribute beinhalten, die vererbt werden. Es können keine Objekte von abstrakten Klassen erzeugt werden.
Prinzip der Vererbung ist die Generalisierung bzw. Spezialisierung. Allgemeine Attribute, die in mehreren Klassen zu finden sind, werden in einer Oberklasse zusammengefasst. Je weiter man im Vererbungsbaum nach unten geht, desto spezieller werden die Klassen in ihren Attributen und Methoden.
Ein weiterer Vorteil der Vererbung ist die Wiederverwendbarkeit von Klassen und die Vermeidung von redundantem Code.

- Konstruktoren für die fünf Klassen (2,5 VP)

```
public Ware(String bezeichnung, double ekPreis,
                                 double vkPreis) {
  this.bezeichnung = bezeichnung;
  this.ekPreis = ekPreis;
  this.vkPreis = vkPreis;
}
```

```
public Getraenk(String bezeichnung, double ekPreis,
                      double vkPreis, double inhalt) {
  super(bezeichnung, ekPreis, vkPreis);
  this.inhalt = inhalt;
}

public Backware(String bezeichnung, double ekPreis,
                                    double vkPreis) {
  super(bezeichnung, ekPreis, vkPreis);
}

public Brezel(double ekPreis, double vkPreis,
                            boolean mitButter) {
  super("Brezel", ekPreis, vkPreis);
  this.mitButter = mitButter;
}

public Broetchen(double ekPreis, double vkPreis,
                                   String belag) {
  super("Broetchen", ekPreis, vkPreis);
  this.belag = belag;
}
```

A2

- Deklaration und Initialisierung der Datenstruktur (1 VP)

```
public class Kiosk {
  private ArrayList<Ware> warenListe;
  private double kontostand;

     public Kiosk(double kontostand) {
    warenListe = new ArrayList<Ware>();
    this.kontostand = kontostand;
  }
...
}
```

Hinweis:
Als geeignete Datenstruktur könnte statt `ArrayList` z.B. auch ein `Array` oder ein `Vector` verwendet werden.

- Methode einkaufenBroetchen(...) (3 VP)

```
public void einkaufenBroetchen(double ekPreis,
                      String belag, int anzahl) {
  for(int i=0; i<anzahl; i++) {
    Broetchen broetchen = new Broetchen(ekPreis,
                                    ekPreis*1.2, belag);
    konto -= ekPreis;
    warenListe.add(broetchen);
  }
}
```

- Methode getGewinnwerartung() (3 VP)

```
public double getGewinnerwartung() {
  double gewinn = 0.0;
  for(Ware ware: waren) {
    gewinn += (ware.getVkPreis() - ware.getEkPreis());
  }
  return gewinn;
}
```

- Methode ausmusternGebaeck() (5 VP)

```
public void ausmusternBackwaren() {
  ArrayList<Backware> wegwerfListe =
                                    new ArrayList<Backware>();
  for(Ware ware: warenListe) {
    if(ware.istKlassenTyp(Backware.class)) {
      Backware backware = (Backware) ware;
      wegwerfListe.add(backware);
    }
  }
  for(Backware backware: wegwerfListe) {
    kontostand -= backware.getEkPreis();
    warenListe.remove(backware);
  }
}
```

Alternativ:

```
public void ausmusternGebaeck() {
  Iterator<Ware> iter = warenListe.iterator();
  while(iter.hasNext()) {
    Ware ware = iter.next();
    if(ware.istKlassenTyp(Backware.class)) {
      kontostand -= ware.getEkPreis();
      iter.remove();
    }
  }
}
```

Fehlerhaft wäre z. B. folgender Algorithmus, da er die Listenstruktur zur Laufzeit zerstört:

```
public void ausmusternGebaeck() {
  for(Ware ware: warenListe) {
    if(ware.istKlassenTyp(Backware.class)) {
      kontostand -= ware.getEkPreis();
      warenListe.remove(ware);
    }
  }
}
```

A3

- Begründung für die Auswahl eines Sortieralgorithmus (3 VP)

Sortieren durch Einfügen (InsertionSort) hat in diesem Fall das beste Laufzeitverhalten. Bereits sortierte Elemente werden nicht mehrmals vertauscht wie beim Bubblesort, sondern nicht sortierte Elemente werden in die richtige Reihenfolge einsortiert.

Aufgabe I B1:

B1.1 • Probleme der Redundanz (2VP)

Bei den Herstellern, den Smartphone-Bezeichungen sowie bei den Kunden sind dieselben Daten mehrfach eingetragen, sie sind redundant.
Wechselt z.B. Rita Richter ihren Wohnsitz, müssen mehrere Datensätze aktualisiert werden. Wird nur einer aktualisiert, tritt Dateninkonsistenz auf.
Die Datensätze sind nicht eindeutig gekennzeichnet. Es fehlt der Primärschlüssel, wodurch ungewollte Duplikate entstehen können.

- optimiertes, relationales Datenbankschema (4 VP)

Der Herstellername steht in direkter Beziehung zur Hersteller-ID.
Der Name und der Preis eines Smartphones sind von der Smartphone-ID abhängig.
Name und Wohnort eines Kunden sind mit der Kunden-ID verknüpft.

Zwischen Hersteller und Smartphone besteht eine 1:n - Beziehung, die durch Aufnahme der Hersteller-ID in die Smartphone-Tabelle realisiert wird. Dasselbe gilt für die 1:n - Beziehung zwischen Kunde und Smartphone.

Somit ergibt sich :
Hersteller(HID, HName)
Smartphone(SID, SName, HID↑)
Kunde(KID, KName, KWohnort)
erwirbt(KID↑, SID↑, SPreis, Kaufdatum)

B 1.2 • Beziehungskardinalitäten (2 VP)

Ein Schauspieler spielt in mehreren Filmen, in einem Film wirken mehrere Schauspieler mit → n:m - Kardinalität
Eine Firma produziert mehrere Filme, ein Film wird von einer Firma produziert
→ 1:n - Kardinalität

SQL-Abfragen (5 VP)

- Welche Filme aus dem Jahr 2012 sind Action-Filme?

```
SELECT *
FROM Film
WHERE Jahr = 2012 AND Genre = „Action“
```

- Welche Filme haben eine Länge zwischen 90 und 120 Minuten?

```
SELECT *
FROM Film
WHERE Laenge BETWEEN 90 AND 120
```

- Wie viele Filme wurden im Jahr 2013 produziert?

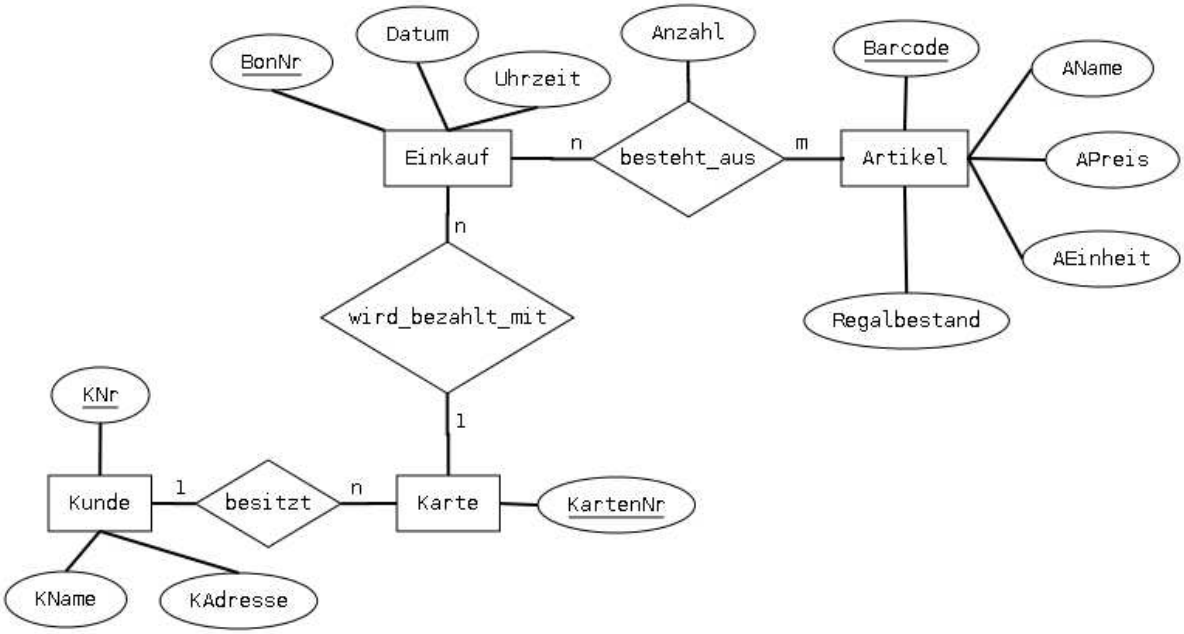
```
SELECT COUNT(*) AS Anzahl
FROM Film
WHERE JAHR = 2013
```

- In welchen Filmen wirkt der Schauspieler „Bruce Wally“ mit?
  Lassen Sie nur die Filmtitel geordnet nach dem Produktionsjahr ausgeben.

```
SELECT Titel
FROM Schauspieler, spielt_in, Film
WHERE Schauspieler.SID = spielt_in.SID
   AND spielt_in.FID = Film.FID
   AND SNachname = „Wally“ AND SVorname = „Bruce“
ORDER BY Film.Jahr DESC
```

B 1.3 • Entity-Relationship-Diagramm (4 VP)

Entitäten mit Schlüsselattributen und Beziehungen
Entität Artikel: Barcode, Name, Preis, Einheit, Regalbestand
Entität Kunde: Kundennummer, Name, Adresse
Entität Karte: Kartennummer
Entität Einkauf: Bonnummer, Datum Uhrzeit
Beziehung besteht_aus: Anzahl Artikelbestand
Beziehung wird_bezahlt_mit *(der Einkauf wird mit Karte bezahlt)*
Beziehung besitzt *(der Kunde besitzt eine Karte)*



- Beziehungskardinalitäten (3 VP)

  Ein Einkauf kann aus mehreren Artikel bestehen, ein Artikel kann bei mehreren Einkäufen erworben werden → n:m-Beziehung
  Ein Einkauf kann mit einer Kundenkarte bezahlt werden, eine Kundenkarte kann für mehrere Einkäufe verwendet werden → 1:n-Beziehung
  Ein Kunde kann mehrere Kundenkarten besitzen, eine Kundenkarte kann nur einem Kunden zugeordnet sein → 1:n-Beziehung

Aufgabe II B1:

B1.1 • Geben Sie eine SQL-Abfrage an, um alle Filme samt Verkaufspreis aufzulisten, die nach dem Jahr 2000 erschienen sind (1 VP)

```
SELECT Film, Preis
FROM Filme
WHERE Jahr>2000
```

- Ergebnistabelle (2 VP)

  Hinweis:
  Die Reihenfolge der Datensätze ist beliebig.

Land	***Anzahl***
USA	*2*
DE	*3*
Fr	*1*

- Geben Sie eine SQL-Abfrage an, die auflistet, welche Filme der Kunde mit der Nummer 3767 im Jahr 2014 bestellt hat. (2 VP)

```
SELECT Film
FROM bestellt
WHERE bestellt.FID = Filme.FID
  AND bestellt.KID == „3767“
  AND Zeitpunkt LIKE „2014%“
```

Hinweis: Alternative Lösungen z.B. mittels `DATEDIFF` *oder* `JOIN` *sind möglich.*

- Der Manager von Paybay benötigt eine Liste mit dem Gesamtumsatz seines Unternehmens in jedem Land. Geben Sie hierfür eine SQL-Abfrage an. (3 VP)

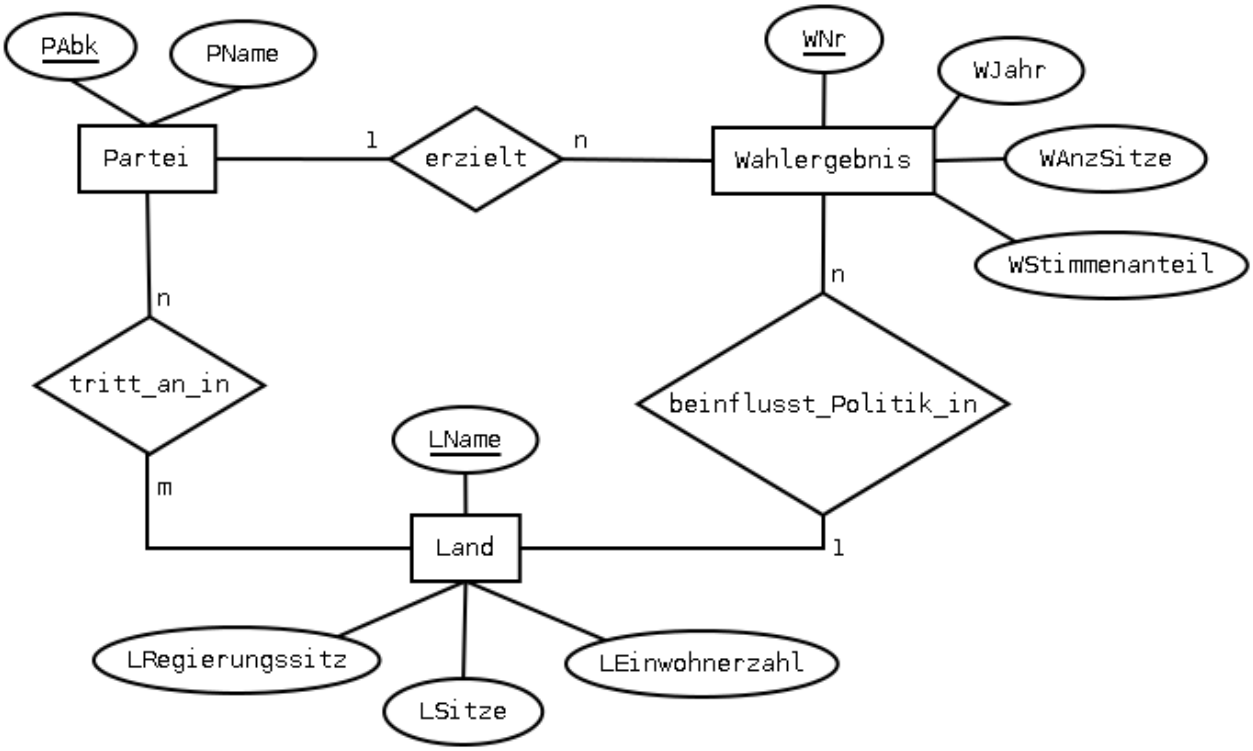
```
SELECT Kunden.Land, SUM(Aanzahl*Preis) AS Umsatz
FROM Kunden, Filme, bestellt
WHERE Kunden.KID = bestellt.KID
  AND FILME.FID = bestellt FID
  AND Zeitpunkt LIKE „2014%“
GROUP BY Kunden.Land
```

Hinweis: Alternative Lösungen mittels `JOIN` *sind möglich.*

- Welche Attribute der Tabelle b*estellt* bilden den Primärschlüssel? (1 VP)

  Die Spalten *FID*, *KID* und *Zeitpunkt* bilden den Primärschlüssel,
  da dies die minimale Spaltengruppe ist, die jeden Datensatz eindeutig identifiziert.
  FID und *KID* sind als Schlüsselattribute notwendig, um jedem Kunden den bestellten Film zuzuordnen. *Zeitpunkt* ist als Schlüsselattribut notwendig, um Mehrfachbestellungen des gleichen Films durch denselben Kunden identifizieren zu können.

B1.2 • Entity-Relationship-Diagramm (4 VP)



Hinweis: Auf die Beziehung beeinflusst_Poltik_in(WNr, LName) kann verzichtet werden.

- Zugriffsrechte (3 VP)

  Der Mitarbeiter benötigt Schreib- und Leserechte auf die Entität Wahlergebnis(WNr, WJahr, WAnzSitze, WStimmanteil) und auf die Relationen erzielt(WNr↑, PAbk↑), tritt_an_in(PAbk↑, LName↑) und beeinflusst_Politik_in(WNr↑, LName↑).

  Hinweis: Falls die 1:n Beziehungen aufgelöst wurden:
  Wahlergebnis(WNr, WJahr, WAnzSitze, WStimmanteil, PAbk↑, LName↑) und tritt_an_in(PAbk↑, LName↑).

  Für Partei(PAbk, PName) und Land(LName, LRegierungssitz, LSitze, LEinwohnerzahl) genügen Leserechte.

- Vigenère-Verschlüsselung (2 VP)

  Das Kennwort „`wahlurne`“ wird zu „`wbplvznf`“ verschlüsselt.

- Entschlüsselung (2 VP)

  Nachdem die Schlüssellänge n bekannt ist, müssen nur noch n Cäsar-Verschlüsselungen analysiert werden.
  Diese können mit einer Häufigkeitsanalyse oder auch mit der Brut-Force-Methode entschlüsselt werden, da es nur 26 Möglichkeiten der Verschlüsselung gibt.

Aufgabe I B2:

B2.1 • Überprüfung, ob M die angegebenen Wörter akzeptiert (1 VP)

i. Das Wort `1100` wird akzeptiert.

Die Folge der durchlaufenden Zustände lautet: $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_2$

ii. Das Wort `1011` wird verworfen.

Die Folge der durchlaufenden Zustände lautet: $q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_1 \xrightarrow{1} q_1$

- Wort der Länge 5, das der Automat nicht akzeptiert (1 VP)
  z.B. `11111`
- Menge der Wörter, die der Automat akzeptiert (1,5 VP)
  Der Automat akzeptiert alle geraden Binärzahlen (oder durch 2 teilbaren Binärzahlen)
  weniger präzise (1 VP): Der Automat akzeptiert alle Binär-Wörter,
  die mit mindestens einer Null enden und mindestens einer 1 beginnen.
- zugehörige Grammatik (1,5 VP)
  Es sei (Σ, V, S, P) die zum Automaten gehörige Grammatik, die die vom Automaten akzeptierte Sprache erzeugt: Dann ist:

  Σ = {ε, `0,1`}; V = {q_0, q_1, q_2}; S = q_0 und

  P = $\{ q_0 \to 0\,q_0,\ q_0 \to 1\,q_1,\ q_1 \to 0\,q_2,\ q_1 \to 1\,q_1,\ q_2 \to 0\,q_2,\ q_2 \to 1\,q_1,\ q_2 \to \epsilon \}$ oder

  Σ = {`0,1`}; V = {`S,A,B`}; Startzustand: S und

  P = $\{ S \to 0\,S,\ S \to 1\,A,\ A \to 0\,B,\ A \to 1\,A,\ A \to 0,\ B \to 0\,B,\ B \to 1\,A,\ B \to 0 \}$

B2.2 • Drei Wörter, die in L liegen (1 VP)
L = {`ab,aab,aaab,...abab,aabab,...,abaab,aabaab...`}

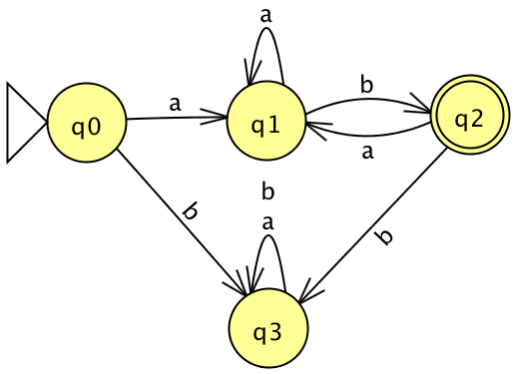
- Erläuterung der Begriffe Terminal- und Nichtterminalsymbol (1 VP)

  i: Terminalsymbole, sind Symbole, die nicht weiter ersetzt werden können und Bestandteil des Alphabets der Sprache sind, hier a und b

  ii: Nichtterminalsymbole können an Hand der Produktionsregeln durch eine Abfolge von Nichtterminalsymbolen oder Terminalsymbolen ersetzt werden, hier X und Y

- Beschreibung der zugehörigen Sprache (1 VP)
  Die durch die obigen Syntaxdiagramme erzeugte Sprache enthält genau die Wörter, die aus einer Abfolge von jeweils „mindestens einen a gefolgt von genau einem b“ bestehen
  Formal: a{a]b{a{a}b} – Dabei bedeuten die geschweiften Klammern eine Wiederholung
- Begründung, dass das Wort „aabbaa“ nicht in L liegt (1 VP)
  Nach einem b muss zunächst mindestens ein a erfolgen. Das ist hier aber nicht erfüllt.

- Endlicher Automat, der genau L akzeptiert (1,5 VP)
  Anmerkung: q_3 ist der Fehlerzustand



- reguläre Grammatik, die L erzeugt (1,5 VP)
  Es sei (Σ, V, S, P) die zum Automaten gehörige Grammatik, die die vom Automaten akzeptierte Sprache erzeugt: Dann ist:

  Σ = {a, b}; V = {q_0, q_1, q_2, q_3}; S = q_0 und

  P = $\{q_0 \rightarrow a\,q_1,\ q_0 \rightarrow b\,q_3,\ q_1 \rightarrow a\,q_1,\ q_1 \rightarrow b\,q_2,\ q_1 \rightarrow b,\ q_2 \rightarrow a\,q_1,\ q_2 \rightarrow b\,q_3,\ q_3 \rightarrow a\,q_3,\ q_3 \rightarrow b\,q_3\}$

B2.3

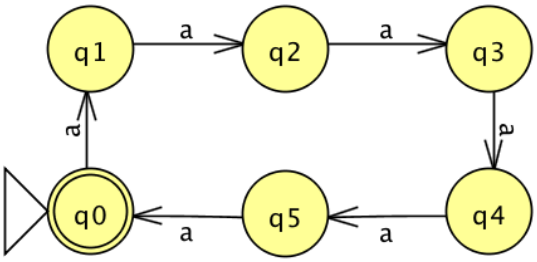
- Beschreibung der Menge aller Wörter in L_1 und L_2 (1 VP)

  i: Die Sprache L_1 umfasst alle Wörter mit einer durch 2 teilbaren Anzahl an a's.

  Ii: Die L_2 umfasst alle Wörter mit einer durch 3 teilbaren Anzahl an a's.

- Beschreibung der Wörter der Sprache L_3 (1 VP)
  L3 beinhaltet alle Wörter, die eine durch 2 und durch 3 teilbare Anzahl an a's beinhalten. Dies sind gerade die Wörter mit einer durch 6 teilbaren Anzahl an a's,

- Entwurf des Automaten M_3 (2 VP)
  Der Automat wird analog zu M_1 und M_2 entworfen.



B2.4

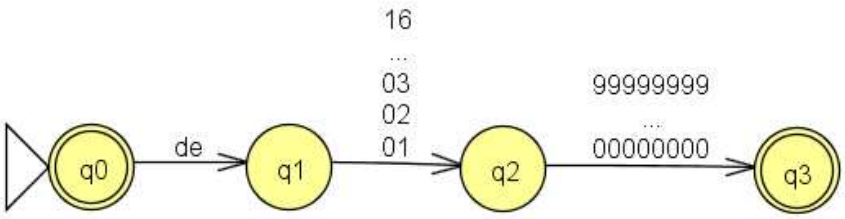
- reguläre Grammatik (1,5 VP)
  Es sei (Σ, V, S, P) die gesuchte Grammatik. Dann ist:

  Σ = {de,01,02,...09,10,11,...16,00000000,00000001,...,99999999};
  V = {L, B, Z} (**L**ändercode, **B**undesland und 8-stellige **Z**ahl); S = L und

  P = $\{L \rightarrow 'de'\,B, B \rightarrow '01'|...|'16'\,Z, Z \rightarrow '0000\,0000'|...|'9999\,9999'\}$

- Untersuchung auf Ableitbarkeit (1,5 VP)
  `de0412345678:` L → 'de' B → 'de' '04' Z → 'de' '04' '12345678': in L enthalten.
  `de1753198124:` L → 'de' B: B lässt sich nicht in 17 auflösen, nicht in L enthalten.
  `de107610358:` L → 'de' B → 'de' '10' Z: Z lässt sich nicht in eine 7-stellige Zahl auflösen, nicht in L enthalten.

- Akzeptor der obigen Sprache (1 VP)



Anmerkung: Nicht aufgeführte Eingaben führen immer zu einem Fehlerzustand.

Aufgabe II B2:

B2.1

- Eingabe- und Ausgabemenge des Fahrscheinautomaten (2 VP)
  E = {1, 2, K, E, R} (siehe Aufgabentext) und A = {–, 1, 2, F_K, F_E} mit
  –: keine Reaktion
  1,2: Rückgabe einer 1.- € oder 2.- € Münze
  F_K: Kinderfahrkarte
  F_E: Fahrkarte für einen Erwachsenen (oder ähnliche Bezeichner)

- Information über zurückliegende Handlungen (1 VP)
  Der Automat muss sich mindestens die Summe der bereits eingezahlten Münzbeträge merken (eventuell auch deren Art, wenn genau dieselbe Stückelung ausgezahlt werden soll, die eingeworfen wurde). Im Modell geschieht dies durch die erreichten Zustände.

- Tabelle der Übergangsfunktion δ (3 VP)

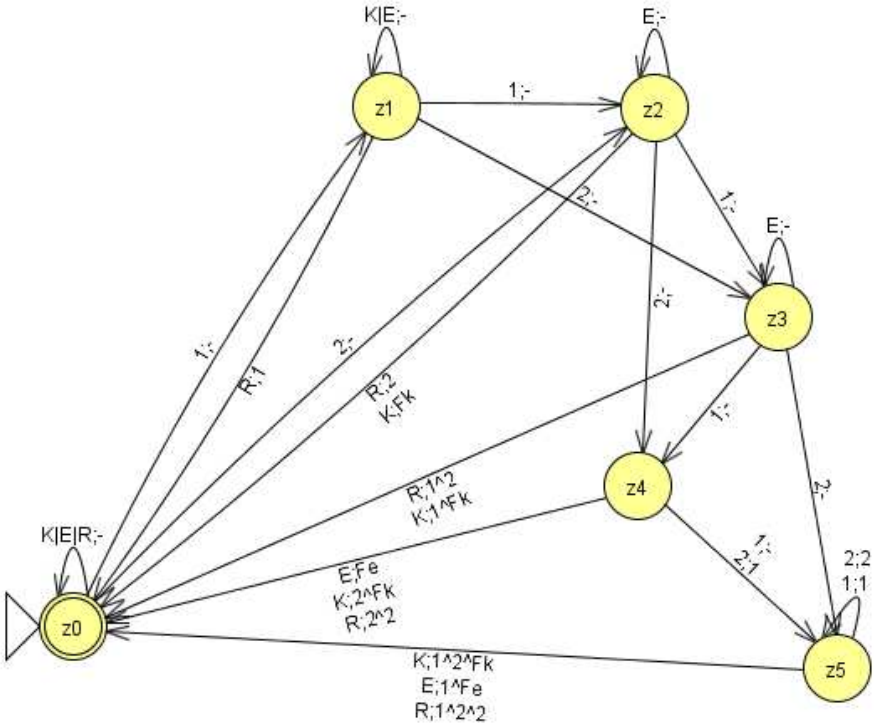
S \ E	1	2	K	E	R	* *Hinweis:*
z_0	z_1	z_2	z_0	z_0	z_0	Für die Reaktion des Automaten bei Überzahlung wird der Aufgabentext so interpretiert, dass nach Anforderung einer Fahrkarte diese mit dem Rückgeld ausgegeben wird (→ z0) und bei Überzahlung nichts passiert (→ z_5).
z_1	z_2	z_3	z_1	z_1	z_0	
z_2	z_3	z_4	z_0	z_2	z_0	Das Rückgeld könnte jedoch auch einbehalten werden, um eine weitere Fahrkarte auszulösen, wenn möglich, oder für weitere Fahrkarten nachzuzahlen (→ z_3 oder z_1). Zusätzliche Münzen können jedoch nicht angenommen werden, wenn nicht weitere Zustände eingeführt werden.
z_3	z_4	z_5	z_0	z_3	z_0	
z_4	z_5	z_5	z_0	z_0	z_0	
z_5*	z_5	z_5	z_0	z_0	z_0	

alternativ kann auch eine Tabelle mit Ausgabemenge angegeben werden:

E	S	S'	A
1	z_0	z_1	–
2	z_0	z_2	–
K | E | R	z_0	z_0	–
1	z_1	z_2	–
2	z_1	z_3	–
K | E	z_1	z_1	–
R	z_1	z_0	1
1	z_2	z_3	–
2	z_2	z_4	–
K	z_2	z_0	F_K
E	z_2	z_2	–
R	z_2	z_0	1∧1 | 2*
1	z_3	z_4	–
2	z_3	z_5	–
K	z_3	z_0	1∧ F_K
E	z_3	z_3	–
R	z_3	z_0	1∧1∧1 | 1∧2*
1	z_4	z_5	–
2	z_4	z_5	1**
K	z_4	z_0	1∧1∧F_K | 2∧F_K*
E	z_4	z_0	F_E
R	z_4	z_0	1∧1∧1∧1 | 1∧1∧2 | 2∧2*
1	z_5	z_5	1**
2	z_5	z_5	2**
K	z_5	z_0	1∧1∧1∧F_K | 1∧2∧F_K*
E	z_5	z_0	1∧F_E
R	z_5	z_0	1∧1∧1∧1∧1 | 1∧1∧1∧2 | 1∧2∧2*

* Der Automat kann nach Vorrat entscheiden, in welcher Stückelung die Münzen zurückgegeben werden.
** Da eine weitere Überzahlung nicht möglich sein soll, fallen die zusätzlichen Münzen einfach durch.

- <u>Zustandsübergangsdiagramm</u> (3 VP)



- Begründung, warum es sich um einen DEA handelt (1 VP)

  Da es von jedem Zustand zu einer Eingabe nur einen Übergang in einen eindeutig bestimmten Folgezustand gibt, handelt es sich um einen deterministischen endlichen Automaten.

B2.2

- Zeichenfolge, die vom Automaten akzeptiert wird (1 VP)
  z. B. gvgvuvgvuv oder gvvgggvuvv usw.

- Bedingungen, unter denen der Trainer entlassen wird (1,5 VP)

  Ein Trainer kann sich maximal zwei Niederlagen erlauben, ohne zwischenzeitlich gewinnen zu müssen. Eine weitere Niederlage führt schließlich zur Entlassung.
  Kann die letzte Niederlage durch einen Sieg ausgeglichen werden, so führen zwei Niederlagen – ohne zwischenzeitlichen Sieg – zum Verlust des Jobs. Kann er sogar die letzten beiden Niederlagen durch zwei aufeinanderfolgende Siege ausgleichen, so hat der Trainer wieder dasselbe Vertrauen wie zu Beginn. Unentschiedene Spiele beeinflussen diese Entscheidung nicht.

- reguläre Grammatik (1,5 VP)
  Es sei (Σ, V, S, P) die zum Automaten gehörige Grammatik, die die vom Automaten akzeptierte Sprache erzeugt: Dann ist:

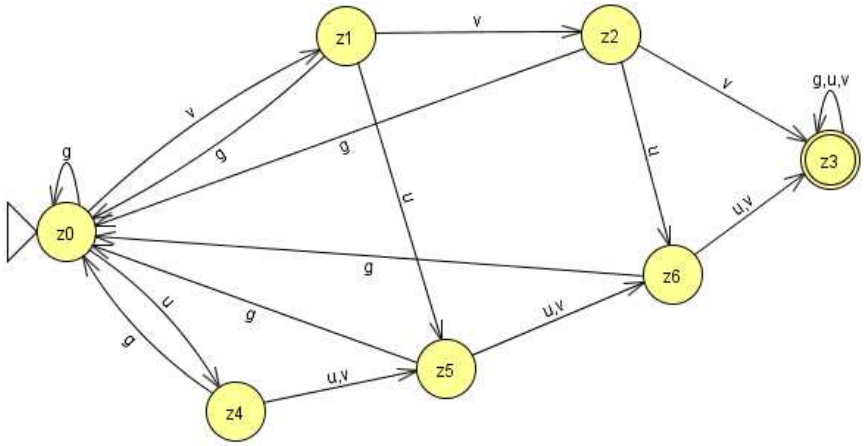
  - Menge der Terminalsymbole Σ = {g, u, v}
  - Menge der Nichtterminalsymbole V = {S, A, B}
  - Startsymbol S
  - Produktionsregeln P = $\{ S \rightarrow gS|uS|vA,\ A \rightarrow gS|uA|vB,\ B \rightarrow gA|uB|v \}$

- Ableitung (1 VP)

  S → vA → vvB → vvgA → vvguA → vvguvB → vvguvv.

- Ergebnisfolgen für das neue Regelwerk, die zu einer Entlassung führen (1 VP)
  z. B. gugvvv und gguuvu usw.

- Neuer Übergangsgraph (3 VP)



- Überprüfung der Zustandsfolgen (1 VP)

  i: gugvvv $z_0 \xrightarrow{g} z_0 \xrightarrow{u} z_4 \xrightarrow{g} z_0 \xrightarrow{v} z_1 \xrightarrow{v} z_2 \xrightarrow{v} z_3$

  Die Eingabe wird akzeptiert, weil die letzten drei Spiele verloren wurden und z_3 ein Endzustand ist.

ii: uvugvvu $z_0 \xrightarrow{u} z_4 \xrightarrow{v} z_5 \xrightarrow{u} z_6 \xrightarrow{g} z_0 \xrightarrow{v} z_1 \xrightarrow{v} z_2 \xrightarrow{u} z_6$

Die Eingabe wird nicht akzeptiert, weil nur drei Spiele in Folge nicht gewonnen wurden und z_6 kein Endzustand ist.

Aufgabe I B3:

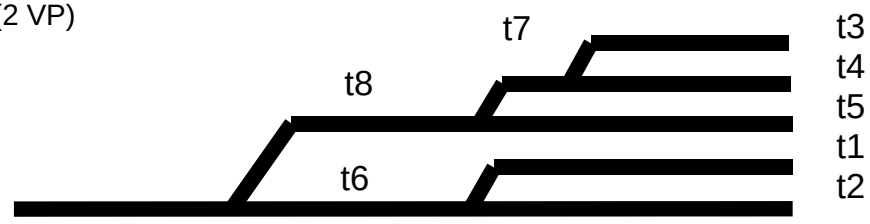
B3.1 • Beschreibung der ADT Stapel und Binärbaum (3 VP)
Ein Stapel stellt die Operationen push und pop zur Verfügung. Diese legen ein Element oben auf dem Stapel ab, bzw. entfernen es wieder von dort. Man nennt dies das LIFO-Prinzip (Last In - First Out). Beim Zug werden die Waggons nur hinten angehängt und hinten wieder entfernt. Durch eine Push-Operation wird also ein neuer Waggon angehängt, durch eine Pop-Operation ein Waggon abgekoppelt.
Ein Binärbaum besteht aus einer Menge von Knoten. Jeder innere Knoten hat zwei Nachfolger, die Kind-Knoten. Die Wurzel ist der Einstiegspunkt des Baumes und hat keinen Vorgänger. Die Datenstruktur Binärbaum ist geeignet, da sich die Gleisanlage ausgehend von einem einzigen Gleis (= Wurzel des Baums) immer weiter verzweigt. Eine Weiche kann dann durch einen Knoten mit zwei Nachfolgern dargestellt werden.

- Implementierung von getLaenge() (3 VP)

```
public int getLaenge() {
  LinkedList e = this.first;
  int i = 0;
  while (e != null) {
    i++;
    e = e.getNext();
  }
  return i;
}
```

Auch rekursive Lösungen sind denkbar!

B3.2 • Gleisanlage (2 VP)



- Erläuterung der Methode stelleWeichenAufFreiesGleis(b:BinaryTree):boolean (3 VP)
Zunächst wird geprüft, ob es einen linken Nachfolger gibt. Ist dies nicht der Fall, handelt es sich um das Ende eines Gleises, da es in diesem Modell dann auch keinen rechten Nachfolger geben darf. Ist kein Content gespeichert, ist das Gleis frei und es wird ***wahr*** zurückgegeben, andernfalls ***falsch***. Enthält der Content eine Weiche, müssen beide Wege verfolgt werden. Zunächst wird der rechte Weg überprüft. Wenn die Suche erfolgreich war, wird die Weiche auf gerade gestellt und die Suche erfolgreich abgebrochen (gib **wahr** zurück). Andernfalls wird die linke Fortführung auf die gleiche Weise überprüft. Wird kein freies Gleis gefunden, bleibt der Zustand der Weichen unverändert und es wird **falsch** zurückgegeben.

- Implementierung von parkeZug(z:Zug) (4 VP)

```
public void parkeZug(Zug z) {
  BinaryTree b = this.gleise;
  while (b.getLeft() != null) {
    Weiche w = (Weiche)b.getContent();
    if (w.isGerade() == true) {
      b = b.getRight();
    } else {
      b = b.getLeft();
    }
  }
  b.setContent(z);
}
```

- Laufzeitverhalten (2 VP)
  Bei geschickter Planung der Gleisanlagen (vollständiger balancierter Binärbaum) beträgt die Laufzeit im schlimmsten Fall (und auch im besten Fall) O(log n), da in einem Baum der Tiefe log(n) maximal n Blätter untergebracht werden können. Daher muss der Zug dann im schlimmsten Fall an O(log n) Weichen vorbeifahren. Ist die Gleisstruktur allerdings degeneriert, so dass alle Weichen hintereinander liegen, dann beträgt die Laufzeit im schlimmsten Fall O(n).

B3.3 • Implementierung von rangiereZug(z:Zug) (3 VP)

```
public void rangiereZug(Zug z) {
  while (!z.isEmpty()) {
    Waggon w = z.pop();
    stelleWeichenZumZug(this.gleise, w.getZiel());
    lasseWaggonRollen(w);
    }
}
```

Aufgabe II B3:

B3.1 • Beschreibung des ADT Stapel (3 VP)
Die Methode push nimmt als Parameter ein Objekt der Klasse Container entgegen und legt es als oberstes Element auf dem Stapel ab. Die Methode pop entfernt das oberste Objekt vom Stapel und liefert es als Ergebnis zurück.
Ein Stapel (Keller) ist eine lineare Datenstruktur. Elemente können ihr nur auf einer Seite hinzugefügt werden, und nur von dieser Seite können sie auch wieder entnommen werden. Dieses Verfahren entspricht dem LIFO-Prinzip (Last In – First Out).
Schiffscontainer können auf einem Stapel nur oben abgelegt werden (push) bzw. nur von oben aus dem Stapel entfernt werden (pop). Der ADT Stapel ist somit eine geeignetes Modell für einen Containerstapel.

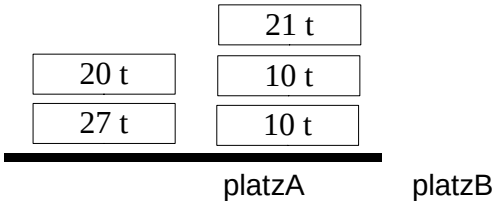
- Implementierung von getGesamtgewicht() (3 VP)

```
public double getGesamtgewicht() {
  LinkedList liste = first;
  double gewicht = 0.0;
  while (liste != null) {
    Container c = (Container) liste.getContent();
    gewicht = gewicht + c.getGewicht();
    liste = liste.getNext();
  }
  return gewicht;
}
```

B3.2 • Implementierung von aufladen(neu:Container) (4 VP)

```
public void aufladen(Container neu) {
  if (platzA.getGesamtgewicht() < platzB.getGesamtgewicht())
    platzA.push(neu);
  else
    platzB.push(neu);
}
```

B3.3 • Skizze der Ladesituation (2 VP)



- Vergleich der Algorithmen (2 VP)
  Der Algorithmus der einfachen Methode aufladen(..) führt zu diesem Ergebnis:

Zeitschritt	Gesamt-gewicht A	Gesamt-gewicht B	Gewicht neuer Container C	Entscheidung
0	10	10	21	Der neue Container kommt immer auf den Stapel mit dem geringeren Gesamtgewicht.
1	10	31	27	
2	37	31	20	
3	37	51		

Die folgenden Werte erhält man mit dem Algorithmus der experimentellen Methode aufladen2(..):

Zeitschritt	Gesamt-gewicht A	Gesamt-gewicht B	Gewicht neuer Container c	A < B	A + C > B + 20	B + C > A +20
0	10	10	21	nein	ja	–
1	21	20	27	nein	ja	–
2	27	41	20	ja	–	nein
3	47	41				

Im vorgegeben Beispiel führt Algorithmus 2 zu einer gleichmäßigeren Verteilung auf den beiden Ladeplätzen.

- Berurteilung der experimentellen Lademethode (2 VP)

  Zu manchen Zeitpunkten bringt Algorithmus 2 auch deutliche Verschlechterungen: Bei den für die Aufgabe gewählten Gewichten entsteht in Zeitschritt 2 in der einfachen Lademethode nur eine Gewichtsdifferenz von 6 t, bei der neuen Lademethode beträgt sie 14 t.

  Die Bewertung ergibt, dass eine Übernahme der experimentellen Lademethode in den Produktivbetrieb nicht zu empfehlen ist.

  Hinweis: zu dieser Empfehlung können auch selbst gewählte Zahlenbeispiele führen. So sind deutlichere Nachteile möglich, wie die Ladung eines 21 t - Containers auf zwei Stapel mit je einem 28 t - Container zeigt. Hier steigt bei Algorithmus 2 die Gewichtsdifferenz auf 35 t, während sie sonst bei den 21 t des letzten Containers liegt. Zur Begründung der abschließenden Empfehlung genügt die Angabe eines korrekten Zahlenbeispiels.

B3.4 • Neuverkettung der Objekte (4 VP)

Bei leerer Liste verweist der Listenkopf first nach dem Einfügen auf das neue Listenelement e. Der Nachfolger von e bleibt leer (null).

Ist die Liste nicht leer, wird nach dem ersten Listenelement f gesucht, das einen Container für einen später zu bedienenden Hafen enthält. Der Vorgänger v von f, der auch der Listenkopf first sein kann, erhält dann als neuen Nachfolger das einzufügende Element e. Der bisherige Nachfolger von v wird zum Nachfolger von e.

Visualisierung: