Beispielabitur

Hinweise zur Korrektur und Bewertung
der Abiturprüfungsaufgaben in

INFORMATIK

Nicht genannte, aber *gleichwertige* Lösungswege und Begründungsansätze sind *gleichberechtigt*.

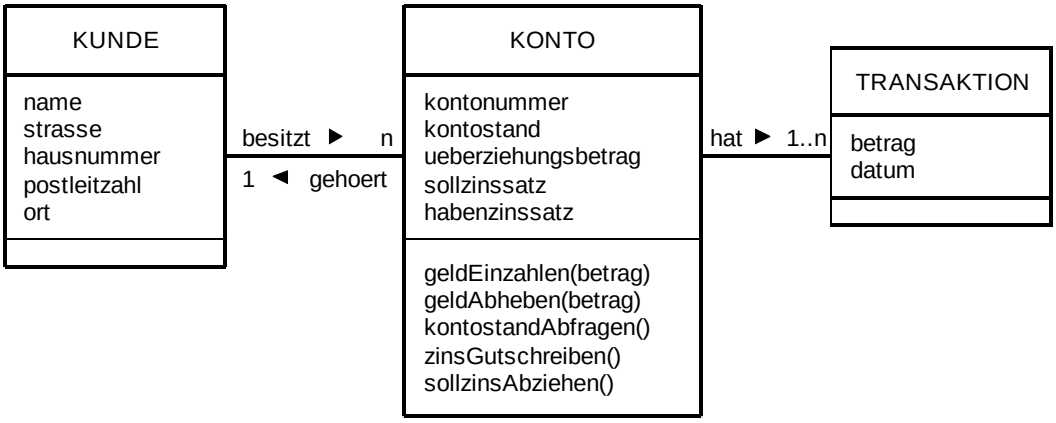
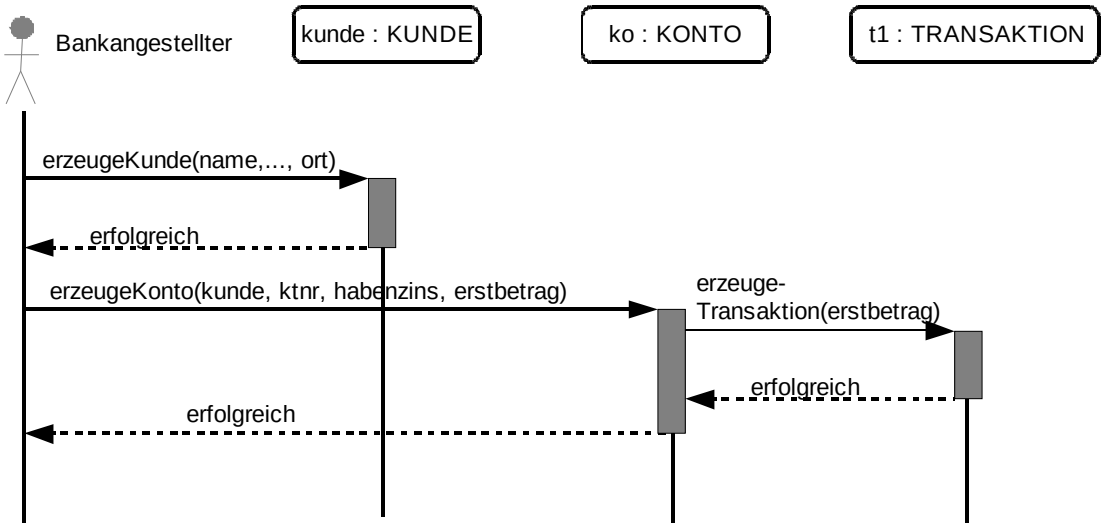
Hinweise

Die nachfolgenden Tabellen enthalten zunächst die Lösungen der Aufgaben des Beispielabiturs in ausführlicher Form. Zusätzlich finden sich, kursiv gesetzt, verschiedene Hinweise für die Lehrkraft, die erläuternden oder vertiefenden Charakter besitzen.

Die vorliegenden Lösungen sind als Vorschläge gedacht, bei denen einige Aspekte, je nach Schwerpunktsetzung im Unterricht, auch nicht in die Bewertung miteinbezogen werden können. Dies kann beispielsweise bei Aufgaben, die Faktenwissen oder Konzepte abfragen, der Fall sein.

Inf1. MODELLIERUNG UND PROGRAMMIERUNG

I.

Aufgabe	BE	Hinweise
1.	10	 *Bei Modellierungsaufgaben gibt es in der Regel verschiedene Lösungsmöglichkeiten. Im obigen Vorschlag könnten beispielsweise folgende Modellierungsvarianten auftreten:* • *Es wird auf das Attribut* kontostand *verzichtet, da eine Berechnung des aktuellen Kontostandes (wenn auch umständlich) grundsätzlich über Transaktionen erfolgen kann. Dies sollte dann aber in der Kommentierung vermerkt werden.* • *Das Einzahlen und Abheben wird durch eine Methode* kontostandAendern(geldbetrag) *modelliert. Im Vorzeichen von* geldbetrag *drückt sich dann die Unterscheidung nach Ein- und Auszahlung aus. Auch hier sollte durch eine entsprechende Kommentierung die Modellierungsentscheidung erläutert werden.*
2. a)	5	 *Die Botschaften im Sequenzdiagramm sollten mit aussagekräftigen Namen versehen werden, die sich beispielsweise an den Bezeichnungen im Klassendiagramm orientieren. Gleiches gilt für die mit den Botschaften übergebene Information, insbesondere im Falle bereits vorhandener Objekte. Kommentare an den Rückkehrpfeilen sind hier nicht erforderlich.*

2. b) | 6

KONTO
int kontonummer double kontostand double ueberziehungsbetrag double sollzinssatz double habenzinssatz KUNDE inhaber LISTE transaktionen
void geldEinzahlen(double betrag) void geldAbheben(double betrag) void kontostandAbfragen() void zinsGutschreiben() void sollzinsAbziehen()

Als Datentypen dürfen hier auch Klassen verwendet werden, die im Unterricht im Rahmen der rekursiven Datenstrukturen erarbeitet wurden, insbesondere

- LISTE
- STAPEL *(als Spezialfall der* LISTE*)*
- SCHLANGE *(als Spezialfall der* LISTE*)*
- BAUM *(insbesondere auch der* BINAERBAUM*)*

Entsprechendes gilt für die Standardmethoden wie z. B. einfuegen, loeschen, suchen.

Bei der Korrektur ist dabei auf die – auch im Vergleich zum Unterricht – stimmige Verwendung solcher Klassen zu achten.

Anstelle des Datentyps double *wäre beispielsweise auch* float *zulässig.*

2. c) | 11

- Erzeugung eines Objekts der Klasse KUNDE mit Festlegung der Kundendaten (siehe Attribute der Klasse KUNDE), das dem KONTO-Konstruktor dann übergeben werden kann.
- Eigentlicher Konstruktor:

```
KONTO(int ktnr, float einzahlung,
                        float zins, KUNDE kunde) {
   kontonummer = ktnr;
   habenzinssatz = zins;
   sollzinssatz = 0.11;
   ueberziehungsbetrag = 500;
   inhaber = kunde;
   kontostand = einzahlung;
   transaktionen = new LISTE();
   TRANSAKTION t1 = new TRANSAKTION(einzahlung);
   transaktionen.amEndeHinzufuegen(t1);
}
```

Beispiele für (mögliche) Anmerkungen durch die Schülerinnen und Schüler:

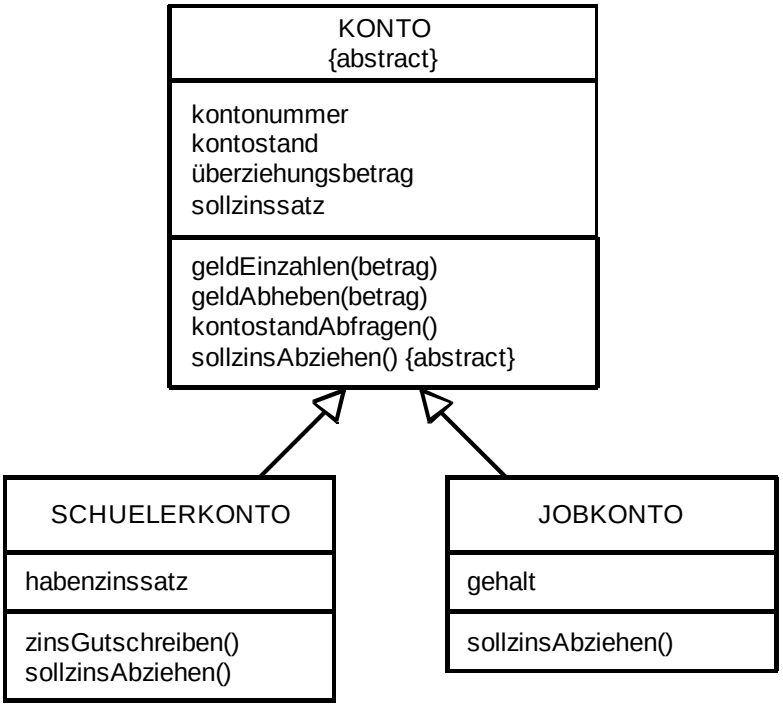
- Der Konstruktor der Klasse TRANSAKTION muss dabei das aktuelle Datum mithilfe einer geeigneten Klasse DATUM „erzeugen“ und dieses dem Attribut Datum zuweisen. Alternativ könnte das Datum über die Eingabeparameter des KONTO-Konstruktors übergeben werden.

- Sollzinssatz und Überziehungsbetrag werden fest vorbelegt. Man könnte diese Belegung natürlich auch über entsprechende Übergabeparameter des Konstruktors vornehmen.
- Im Konstruktor könnte überprüft werden, ob der Einzahlungsbetrag mindestens 10 € ist. Es kann aber auch davon ausgegangen werden, dass dies der Bankangestellte vorher abgeklärt hat.

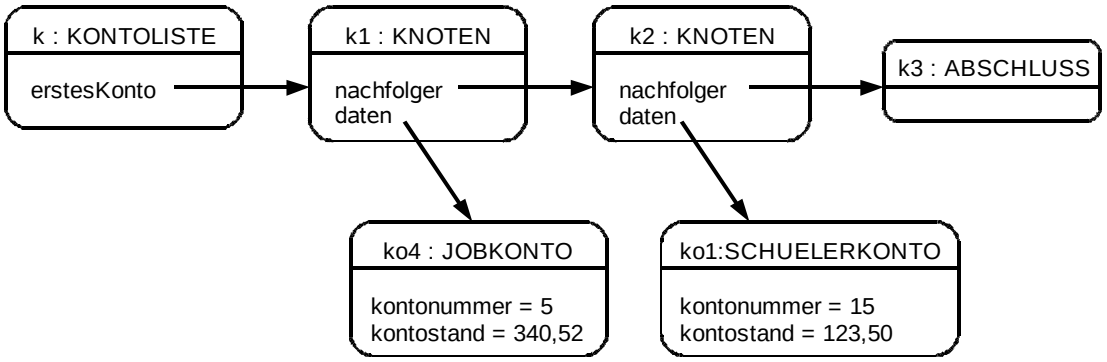
Im obigen Fall wurde als Datentyp für die Transaktionsliste die (im Unterricht erarbeitete) Klasse LISTE verwendet. Grundsätzlich sind hier mehrere Varianten denkbar.

Eine Implementierung ist im BlueJ-Projekt MusterabiturI2_Konto realisiert. Die Liste wird hierbei mithilfe der in Java bereits zur Verfügung stehenden Klasse ArrayList umgesetzt.

3. 5

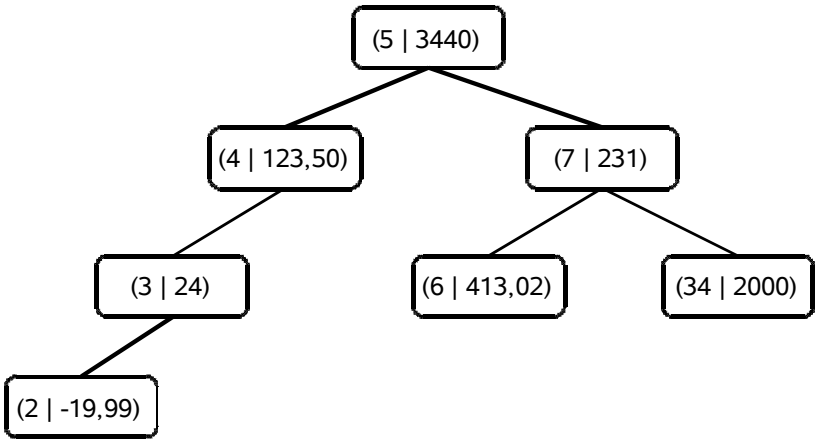


- *Die Umwandlung der Klasse KONTO in eine abstrakte Klasse ist sinnvoll, weil es gemäß Aufgabenstellung nur noch Schüler- und Jobkonten geben soll. Damit sollte also von der Klasse KONTO keine Instanz mehr erzeugt werden können. Prinzipiell ist aber auch eine „normale" Klasse vorstellbar.*
- *Die Methode* sollzinsAbziehen() *ist abstrakt, da sie in den Unterklassen verschieden implementiert werden muss. Auch hier wäre prinzipiell ebenso eine normale Methode denkbar, die überschrieben wird. In diesem Fall sollte dies aber in der Schülerlösung kommentiert sein.*
- *Habenzins und entsprechende Methoden wurden nun in die Klasse SCHUELERKONTO gezogen, da sie nur für Schülerkonten relevant sind. Auch hier wären prinzipiell andere Umsetzungen möglich, wenn diese nachvollziehbar begründet werden.*

4.	5	 *Im Objektdiagramm müssen die Beziehungsnamen des Klassendiagramms nicht unbedingt als entsprechende Attributbezeichner übernommen werden. Neu eingeführte Namen müssen aber aussagekräftig sein. Gegebenenfalls ist eine kurze Anmerkung erforderlich.* *Hier ist angedeutet, dass die Beziehungen „erstesKonto“, „nachfolger“ und „daten“ als entsprechende Attribute realisiert sind. Man könnte in dem Objektdiagramm auf diese Attribute verzichten, wenn die entsprechenden Beziehungsinstanzen (Pfeile) geeignet beschriftet werden.*
5.	6	• Erzeugen eines neuen Objekts *kontoNeu* von der Klasse SCHUELERKONTO, das die Daten des neuen Schülerkontos (mit *kontonummer* = 12 und *kontostand* = 10) enthält. • Erzeugen eines neuen Knotenobjekts *knotenNeu* mit Referenz auf das Objekt *kontoNeu.* • Suchen des Knotenobjekts, dessen Datenelement die größtmögliche Kontonummer ausweist, die kleiner als 12 ist, hier gemäß Lösung Aufgabe 5: *k1*. • Dem Knotenobjekt *knotenNeu* wird als Nachfolger der Nachfolger von *k1*, hier *k2*, zugewiesen. • Dem Knotenobjekt *k1* wird als neuer Nachfolger das Knotenobjekt *kontoNeu* zugewiesen.

6.	9	In der Klasse ABSCHLUSS:
		`KONTO kontoAbHierSuchen(int ktnr) {`
		`  return null;`
		`}`
		
		In der Klasse KNOTEN:
		`KONTO kontoAbHierSuchen(int ktnr) {`
		`  if (daten.kontonummerPruefen(ktnr)) {`
		`  return daten;`
		`  } else {`
		`  return nachfolger.kontoAbHierSuchen(ktnr);`
		`  }`
		`}`

		In der Klasse KONTOLISTE:
		`KONTO kontoSuchen(int ktnr) {`
		`  return erstesKonto.kontoAbHierSuchen(ktnr);`
		`}`
		
		Da die Methode nicht außerhalb des Pakets sichtbar sein soll, wird das Zugriffsrecht nicht auf public gesetzt. Das Setzen dieses Zugriffsrechts bei der Schülerlösung ist aber nicht bewertungsrelevant. Die entsprechende Implementierung findet man im Projekt MusterabiturI7_Konto.
		Die Methode kontoabHierSuchen() *der abstrakten Klasse KOMPONENTE wurde bewusst so genannt, um zu verdeutlichen, dass diese Methode nur das angesprochene Listenelement und alle darauf folgenden Listenelemente und nicht die Gesamtliste durchsucht. Im Gegensatz dazu würde man von einer Methode* suchen() *erwarten, dass alle Listenelemente durchsucht werden.*

7. a)	4	Mögliches Szenario: Die Software liest direkt hintereinander in Operation 1 und 2 den (gleichen) Kontostand ein. Operation 1 wird schneller durchgeführt und der neue Kontostand gespeichert. Anschließend wird die Berechnung von Operation 2 durchgeführt und der neue Kontostand ebenfalls abgespeichert. Damit wird aber der durch Operation 1 erzeugte Kontostand überschrieben. Operation 1 hat damit faktisch nicht stattgefunden.
7. b)	2	Jede Operation muss als Gesamtes durchgeführt werden, z. B. Kennzeichnung des Kontostandes als „kritischer“ Bereich, auf den jeweils nur eine Operation Zugang hat.
8. a)	2	In der Regel ist in einem geordneten Binärbaum eine schnellere Suche als in der Liste möglich.
8. b)	4	

8. c)	4	`KNOTEN linkerTeilbaum;`
		`KNOTEN rechterTeilbaum;`
		`KONTO daten;`

8. d)	7	Der inorder-Durchlauf ergibt automatisch die richtige Sortierung.
		In der Klasse ABSCHLUSS:
		`void kontolisteAbHierAusgeben() {`
		`}`

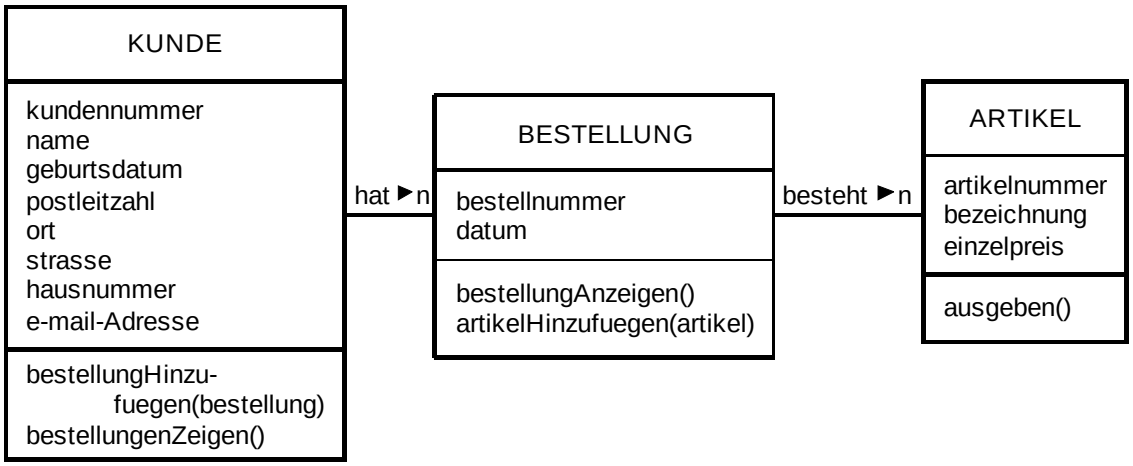
In der Klasse KNOTEN:

```
void kontolisteAbHierAusgeben () {
    linkerTeilbaum.kontolisteAbHierAusgeben();
    daten.datenAusgeben();
    rechterTeilbaum.kontolisteAbHierAusgeben();
}
```

Eine entsprechende Implementierung findet man im BlueJ-Projekt MusterabiturI9_Konto

Inf1. MODELLIERUNG UND PROGRAMMIERUNG

II.

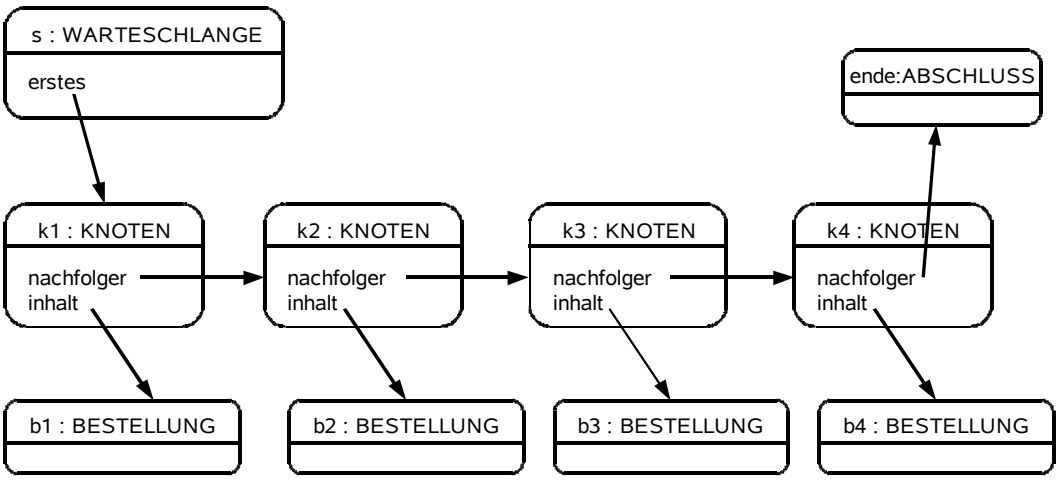
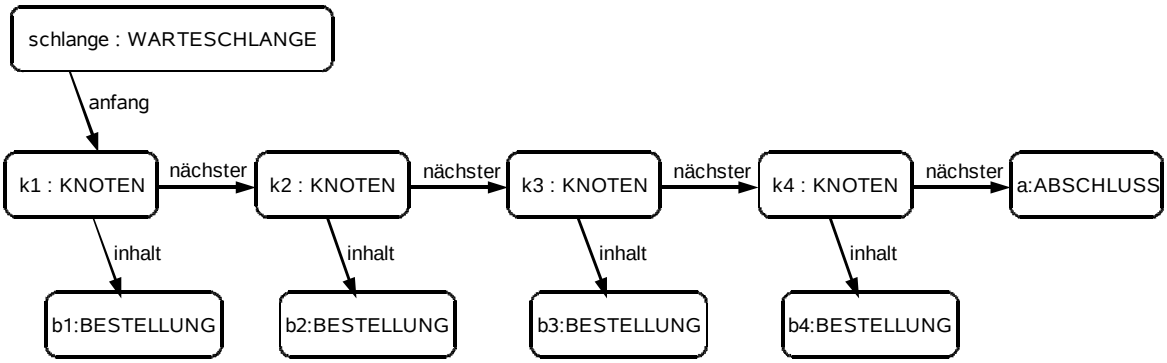
Aufgabe	BE	Hinweise
1. a)	10	

Die Aufgabenstellung enthält insbesondere folgende Anforderungen:

- Der Benutzer kann im Rahmen einer Bestellung Artikel kaufen. Dies erfolgt durch Anlegen eines neuen Objekts der Klasse BESTELLUNG und das evtl. wiederholte Aufrufen der Methode *artikelHinzufügen()*.
- Der Benutzer bekommt eine Übersicht über eine Bestellung, indem er die Methode *bestellungAnzeigen()* aufruft.
- Der Benutzer kann alle Bestellungen eines Kunden sehen. Dies erfolgt durch Aufruf der Methode *bestellungenZeigen()*.
- Der Benutzer selbst wird hier nicht modelliert.

Standardmethoden zum Lesen oder Setzen von Attributwerten müssen nicht unbedingt im Klassendiagramm aufgeführt werden, wenn sie keine anderen Anweisungen als das reine Lesen oder Setzen von Attributwerten enthalten und im Rahmen der Modellierung keine besondere Rolle spielen.

Die Methode preisGeben( ) *der Klasse ARTIKEL muss nicht unbedingt angegeben werden, da sie nur einen Attributwert ohne weitere Anweisungen zurückliefert. Die Methode* ausgeben( ) *der Klasse ARTIKEL stellt einen Artikel in geeigneter Weise in einem Text dar, der dann am Bildschirm ausgegeben wird. Insofern handelt es sich hier um eine Methode, die zu nennen ist, da sie nicht nur Attributwerte liest oder setzt.*

1. b)	10	i) Grundlegende Eigenschaften der Datenstruktur Schlange: • Die Warteschlange kann beliebig viele Objekte enthalten. • Neue Elemente werden an das Ende der Warteschlange angefügt. • Elemente können nur vom Anfang der Warteschlange entfernt werden. Die Bestellungen werden in der Reihenfolge ihres Eingangs abgearbeitet. Es gilt also das Prinzip „First in, first out“. Dieses Prinzip wird durch die Datenstruktur „Warteschlange“ unmittelbar umgesetzt. (Fakultativ: Eine allgemeine Liste ist hier nicht nötig, da eine Bestellung nicht an eine beliebige Stelle der Liste eingefügt werden muss und auch ein Sortieren nicht nötig ist.) ii) z. B.:  alternativ:  *Wesentliches Prinzip der Warteschlange ist FIFO (siehe oben). In dieser Lösung ist die Warteschlange mithilfe einer verketteten Liste realisiert, in der das Kompositum-Softwaremuster Anwendung findet.*

1. c)	14	

In der Klasse BESTELLUNG:

```
public void anzeigen( ){
//Das Attribut „erstes“ referenziert das erste
//Listenelement.
  erstes.abHierAusgeben( );
  System.out.println(„Gesamtpreis aller Artikel: “
         + erstes.abHierPreisGeben( ) + „ Euro.“ );
}
```

In der Klasse KNOTEN:

```
//Abstrakte Methode der Oberklasse muss in der Unterklasse
//implementiert werden.
public void abHierAusgeben( ){
  //Das Attribut „artikel“ referenziert das dem Knoten
  //zugeordnete Objekt der Klasse ARTIKEL
  artikel.ausgeben( );
  //rekursiver Aufruf:
  hatAlsNachfolger.abHierAusgeben( );
}
//Neue Methode zur Preisberechnung:
public double abHierPreisGeben( ){
  return artikel.preisGeben()
    + hatAlsNachfolger.abHierPreisGeben(); //rekurs. Aufruf
}
```

In der Klasse ABSCHLUSS:

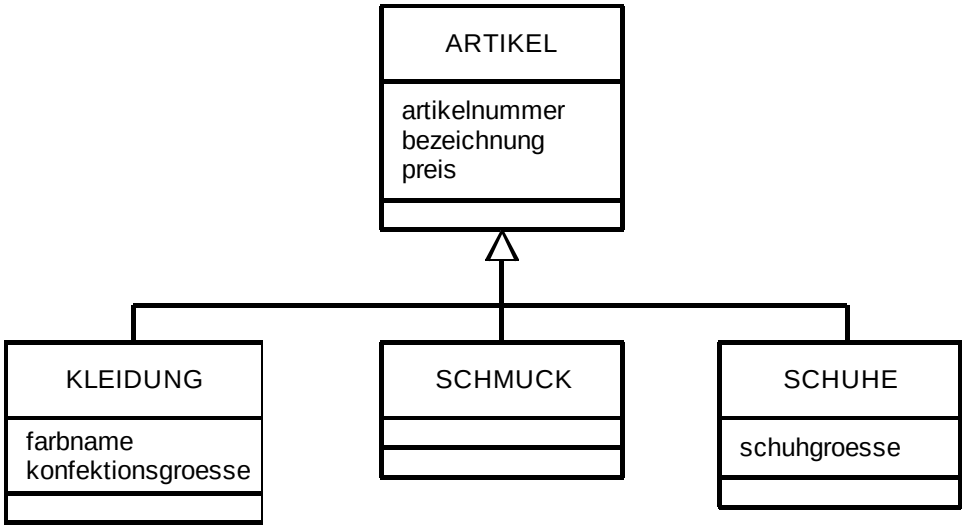
```
public void abHierAusgeben( ){
  //nichts zu tun
}
public double abHierPreisGeben( ){
  return 0;
}
```

In der Klasse ARTIKEL:

```
public void ausgeben( ){
  System.out.println(„Der Artikel „ + artikelname +
     „ mit der Nummer “ + artikelnummer + „ kostet “
     + einzelpreis + „ Euro.“ );
}
public double preisGeben( ){
     return einzelpreis;
}
```

Die Methode abHierAusgeben() *der Klasse LISTENELEMENT wurde bewusst so genannt, um zu verdeutlichen, dass diese Methode nur das angesprochene Listenelement und alle darauf folgenden Listenelemente und nicht die Gesamtliste ausgibt. Im Gegensatz dazu würde man von einer Methode* ausgeben() *erwarten, dass alle Listenelemente ausgegeben werden.*

1. d) | 12 | i)



Die Klassenhierarchie ist hier zwingend erforderlich (Grundwissen Jahrgangsstufe 10).

Die Klasse SCHMUCK hat weder eigene Attribute noch eigene Methoden. Daher gibt es mehrere Möglichkeiten der Modellierung:

- *Die Klasse SCHMUCK wird als eigene Klasse modelliert. Die Klasse ARTIKEL könnte dann abstrakt sein, und es gäbe keine anderen Artikel als KLEIDUNG, SCHMUCK oder SCHUHE.*
- *Die Klasse ARTIKEL ist nicht abstrakt. In diesem Fall kann auf die Klasse SCHMUCK verzichtet werden. Diese Entscheidung wäre von der Schülerin oder von dem Schüler zu kommentieren, um zu belegen, dass sie bewusst so gefällt wurde und SCHMUCK nicht einfach vergessen wurde.*

ii)

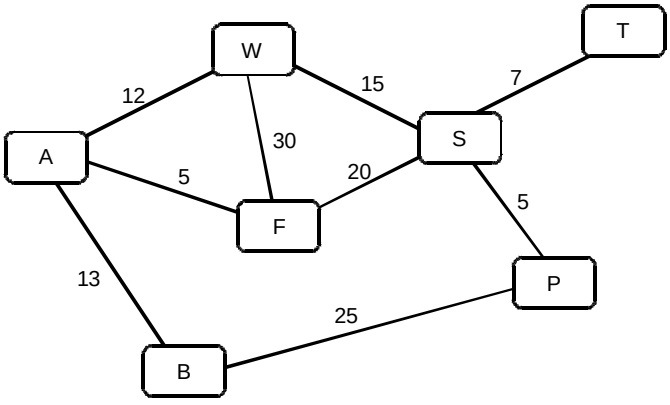
```
public void etikettieren( ){
  drucken(„Der Artikel“ + bezeichnung +
          „ und der Nummer “ + artikelnummer +
          „ kostet “ + preis + „Euro.“);
}
```

Hier wird der Einfachheit halber vorausgesetzt, dass für die String-Umwandlung des Attributwerts von preis *gesorgt wird.*

iii)

```
public KLEIDUNG(int artikelnummer, String bezeichung,
          double preis, String farbe, String groesse ){
  //Konstruktor der Oberklasse!
  super(artikelnummer, bezeichnung, preis);
  farbname = farbe;
  konfektionsgroesse = groesse;
}

public void etikettieren ( ){
  //Aufruf der Methode der Oberklasse!
  super.etikettieren( );
  drucken(„ Farbe “ + farbname +
          „ und Größe “ + konfektionsgroesse);
}
```

		*Das Prinzip der Datenkapselung bedeutet hier, dass in der Klasse KLEIDUNG der Konstruktor bzw. eine Methode der Oberklasse aufgerufen wird und nicht direkt auf die Attribute der Oberklasse (*artikelnummer, bezeichnung, preis*) zugegriffen wird. Eine Schülerlösung, die dies nicht berücksichtigt, enthält einen gravierenden Mangel.* *Der Datentyp des Attributes* konfektionsgroesse *ist* String*, damit Größen wie S, M, L, XL möglich sind. Es ist auch zulässig, den Datentyp* int *für die Größen 38, 40, 42 etc. zu wählen.*
1. e)	6	i) `SELECT * FROM artikel_tab WHERE Bezeichung = “Hose”;`
		ii) 1. Datenbank öffnen. 2. Statement erzeugen. 3. Abfrage abschicken. (Die SQL-Anweisung wird als String übergeben.) 4. Ergebnismenge wird empfangen. (Das Ergebnis einer SQL-Abfrage ist in der Regel eine Tabelle, die zeilenweise ausgelesen werden muss.) 5. Ergebnismenge mit einer Wiederholung auslesen und zeilenweise ausgeben. 6. Ergebnismenge schließen. 7. Statement schließen. 8. Datenbank schließen. *Im Rahmen des Lehrplan-Themas Inf 11.2.2. „Praktische Softwareentwicklung“ wird die Verbindung von Datenmodellierung und Objektmodellierung gefordert. Bei der Implementierung muss nun eine Verbindung zwischen einer Datenbank und einer Anwendung in einer objektorientierten Sprache erstellt werden. Dies wird im Unterricht auf viele verschiedene Arten geschehen, im Grundsatz sind jedoch die folgenden beschriebenen Schritte nötig. Diese Schritte müssen nicht genau in der o. g. Anzahl und Feinheit genannt werden. Die Bewertung der Lösung richtet sich nach den Vereinbarungen aus dem Unterricht.* *Beispielsweise wäre auch eine Lösung, die Aspekte zusammenfasst möglich, z. B. 1, 2+3, 4+5, 6-8.*
2. a)	5	Es handelt sich hier um einen bewerteten, ungerichteten Graph (mit Zyklen). 

2. b) 4

	W	S	T	A	F	P	B
W	-1	15	-1	12	30	-1	-1
S	15	-1	7	-1	20	5	-1
T	-1	7	-1	-1	-1	-1	-1
A	12	-1	-1	-1	5	-1	13
F	30	20	-1	5	-1	-1	-1
P	-1	5	-1	-1	-1	-1	25
B	-1	-1	-1	13	-1	25	-1

Der Wert –1 steht für keine Verbindung. Die entsprechende Zelle kann auch leer gelassen werden.

2. c) 4

Z. B.: Die Klasse GRAPH erhält ein Attribut *knoten,* dessen Datentyp ein eindimensionales Feld vom Typ String ist, und ein Attribut *kanten*, dessen Datentyp ein zweidimensionales Feld vom Typ int ist.

Mögliche Deklaration in der Klasse GRAPH in Java:

```
String[] knoten;
int[][] kanten;
```

Die Lösung kann auch allgemeiner gehalten werden. Die Klasse GRAPH benötigt zumindest folgende Attribute:

- *Ein (Referenz-)Attribut für die Referenz auf die „indizierte Gesamtheit" der Knoten des Graphen*
  - *mögliche Bezeichnung:* knotenliste
  - *möglicher Datentyp: Feld vom Typ String oder KNOTEN (falls die Knoten des Graphen durch eine eigene Klasse KNOTEN repräsentiert werden)*
- *Ein Attribut für die Repräsentation der Adjazenzmatrix*
  - *mögliche Bezeichnungen:* kanten *oder* adjazenzmatrix
  - *möglicher Datentyp: zweidimensionales Feld vom Typ int*

2. d) 3

Ein ungerichteter Graph ist ein Baum, wenn er keine Zyklen enthält. Obiger Graph ist kein Baum, da er unter anderem den Zyklus WAFW enthält.

2. e)	12	Lösung in Pseudocode:
		Voraussetzungen und Bezeichnungen:
		Die Klasse GRAPH enthält
		• ein Attribut *knotenliste* vom Typ String[]. Über den Index können die Knoten des Graphen identifiziert werden.
		• ein Attribut *matrix* vom Typ int[][]. Mit diesem wird die Adjazenzmatrix des Graphen repräsentiert. Eine Kante wird durch einen nicht-negativen Eintrag in der Matrix repräsentiert.
		• ein Attibut *besucht* vom Typ boolean[]. Ein Feldelement hat den Wert *wahr*, wenn der entsprechende Knoten schon besucht/bearbeitet wurde, sonst hat es den Wert *falsch*.
		Der Attributwert des Attributs *anzahlKnoten* gibt die Anzahl der Knoten des entsprechenden Graphen wieder. Der Parameter *startKnotenNr* bezieht sich auf den Index eines Knotens in *knotenliste*.
		
		• **Methode** tiefensuche (int startKnotenNr)
		wenn (startKnotenNr >= 0 und
		startKnotenNr <= Länge von knotenliste) **dann**
		zähle i von 0 bis anzahlKnoten-1
		besucht[i] = falsch
		endeZähle
		besuchen(startKnotenNr)
		endeWenn
		endeMethode
		
		• **Methode** besuchen (int knotenNr)
		//Nur etwas tun, wenn der Knoten noch nicht besucht worden ist
		wenn (**nicht** besucht[knotenNr]) **dann**
		//aktiven Knoten auf besucht setzen und evtl. Informationen
		//auf der Konsole ausgeben
		besucht[knotenNr] = wahr
		In der Konsole ausgeben: knotenliste[knotenNr]
		//Nachbar- bzw. Nachfolgeknoten besuchen:
		zähle i von 0 bis anzahlKnoten-1
		wenn (matrix[knotenNr][i] >= 0) **dann**
		besuchen(i)
		endeWenn
		endeZähle
		endeWenn
		endeMethode

Alternative Lösung in Java:

```
public class GRAPH_MIT_SUCHE{

//Darstellung des Graphen durch die Adjazenzmatrix
private String[] knotenliste =
           {"Wohnung","Schule","Thomas",
            "Anne","Fußballplatz","Pizzeria","Bad"};
private int[][] matrix =
           {   { -1, 15, -1, 12, 30, -1, -1},
               {15, -1, 7,  -1, 20,  5,  -1},
               {-1, 7 , -1, -1,  -1, -1, -1},
               {12, -1, -1, -1, 5 , -1 , 13},
               {30, 20, -1,  5, -1, -1, -1},
               {-1,  5, -1, -1, -1,  -1, 25},
               {-1, -1, -1, 13, -1, 25, -1} };

//Zustände d. Knoten: nicht besucht (false), besucht (true)
private boolean[] besucht =
                     new boolean[knotenliste.length];

//Hauptmethode zum Aufruf der Tiefensuche
public void tiefensuche(int startNr){
  //Initialisierung: Am Anfang sind alle Knoten unbesucht
  for(int i=0; i<knotenliste.length; i++){
     besucht[i] = false;
  }
  //Beginn der Tiefensuche mit dem Knoten, der die
  //angegebene Nummer hat.
  if (startNr >= 0 && startNr < knotenliste.length){
      besuchen(startNr);
  }
}
//Hilfsmethode: Rekursiver Aufruf zur Tiefensuche
private void besuchen(int knotenNr){
  //Nur etwas tun, wenn der Knoten noch nicht besucht wurde.
  if (!besucht[knotenNr]){
      //Knoten als besucht markieren und auf der Konsole
      //ausgeben.
      besucht[knotenNr] = true;
      System.out.println("Jetzt wurde der Ort "+
                     knotenliste[knotenNr] + " besucht." );
      //Benachbarte Knoten „besuchen“
      for (int i=0; i<knotenliste.length; i++){
         if (kanten[knotenNr][i] > 0){
            besuchen(i);
         }
      } //Ende for
  } //Ende if
}
} //Ende class GRAPH_MIT_SUCHE
```

		Der Klarheit und Lauffähigkeit wegen wird eine mögliche Implementierung für die ganze Klasse GRAPH_MIT_SUCHE *angegeben, obwohl nur die Methoden* tiefensuche() *und* besuchen() *zusammen mit Erläuterungen zu Randbedingungen wie den verwendeten Attributen gefragt sind.* *Der Einfachheit halber wird hier ein „spezieller Graph“ implementiert, der nur den angegebenen Graphen repräsentiert. Die Methoden hingegen sind, wie in der Aufgabenstellung gefordert, so allgemein gehalten, dass sie auch für beliebige Graphen Anwendung finden können.* *Laut Lehrplan ist ein Verfahren zum Graphendurchlauf verpflichtend. Die Art der Darstellung des Algorithmus und die Tiefe der Ausarbeitung hängt stark vom Unterricht ab. Im Unterricht sollte ein Algorithmus zum Graphendurchlauf implementiert worden sein. Der Algorithmus sollte zumindest so genau geschildert werden, wie es für eine Umsetzung in ein Programm erforderlich ist. Als eine mögliche Lösung ist hier die Tiefensuche vorgestellt.*
	80	

Inf2. THEORETISCHE UND TECHNISCHE INFORMATIK

III.

Aufgabe	BE	Hinweise
1. a)	3	Die Abarbeitung der gegebenen Codes liefert bei dem gegebenen deterministischen Automaten folgende Zustandsübergänge: $z_0 \xrightarrow{0} z_1 \xrightarrow{1} z_3 \xrightarrow{1} z_4 \xrightarrow{0} z_2$, d. h., nach der Abarbeitung von 0110 ist der Endzustand nicht eingenommen, der Code wird nicht akzeptiert. $z_0 \xrightarrow{0} z_1 \xrightarrow{0} z_1 \xrightarrow{0} z_1 \xrightarrow{1} z_3 \xrightarrow{1} z_4$, d. h., nach der Abarbeitung des Wortes 00011 ist der Endzustand erreicht, der Code wird akzeptiert.
1. b)	4	Alle Codes beginnen mit einer *0* und enden mit genau zwei Zeichen *1*. Dazwischen kann eine beliebige Anzahl von *0* stehen: 0{0}11 *Je nach Einführung im Unterricht sind verschiedene Schreibweisen der (erweiterten) Backus-Naur-Form möglich. {0} bedeutet hier, dass die 0 beliebig oft (auch null-mal) wiederholt werden kann (siehe Informatik-Duden, S. 52).*
1. c)	11	

```
class Automat {
//Attribut
private int zustand;
//Konstruktor
Automat () {
   zustand = 0;
}

//Methode zur Prüfung, ob Code akzeptiert wird
void codeTesten(String w) {
   zustand = 0;
   String original = w;
   //Abarbeitung des Codes
   for (int i=0; i<w.length(); i++) {
        schalten(w.charAt(i));
   }
   if (zustand == 4){
       System.out.println("Automat akzeptiert " +
                          original);
   }
   else{
       System.out.println("Automat akzeptiert " +
                          original + " nicht");
   } //Ende if
}
```

```
//Hilfsmethode zum Weiterschalten
private void schalten(char z){
   switch(zustand){
      case 0: if (z == '0'){
                  zustand = 1;
               } else {
                  zustand = 2;
               }
              break;
      case 1: if (z == '0'){
                  zustand = 1;
               } else {
                  zustand = 3;
               }
               break;
      case 2: zustand = 2;
               break;
      case 3: if (z == '0'){
                  zustand = 2;
               } else {
                  zustand = 4;
               }
              break;
      case 4:  zustand = 2;
               break;
   } //Ende switch
}
} //Ende Klasse Automat
```

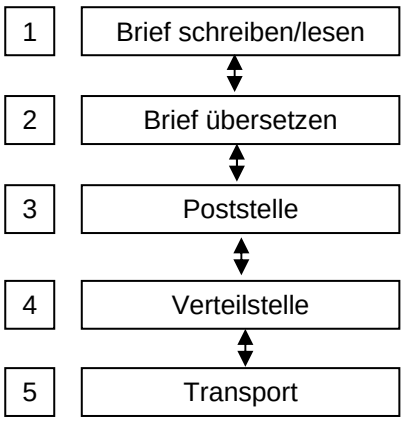
Eine Implementierung steht im BlueJ-Projekt MusterabiturII1_Schloss zur Verfügung.

2. a)	6	Von-Neumann-Modell: • Funktionseinheiten: Eingabe und Ausgabe, Speicher, Rechenwerk, Steuerwerk • Speichermodell: Der Speicher besteht aus gleich großen, fortlaufend nummerierten Zellen. Daten und Programme liegen im gleichen Speicher. • Bedeutung der Sprungbefehle: Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgenden Speicherzellen abgelegt. Sprungbefehle erlauben eine Abweichung von Bearbeitungs- und Speicherreihenfolge.
2. b)	2	• im Register 19: ursprüngliche ganze Zahl a • im Register 20: doppelter Wert von a • ansonsten keine Änderungen

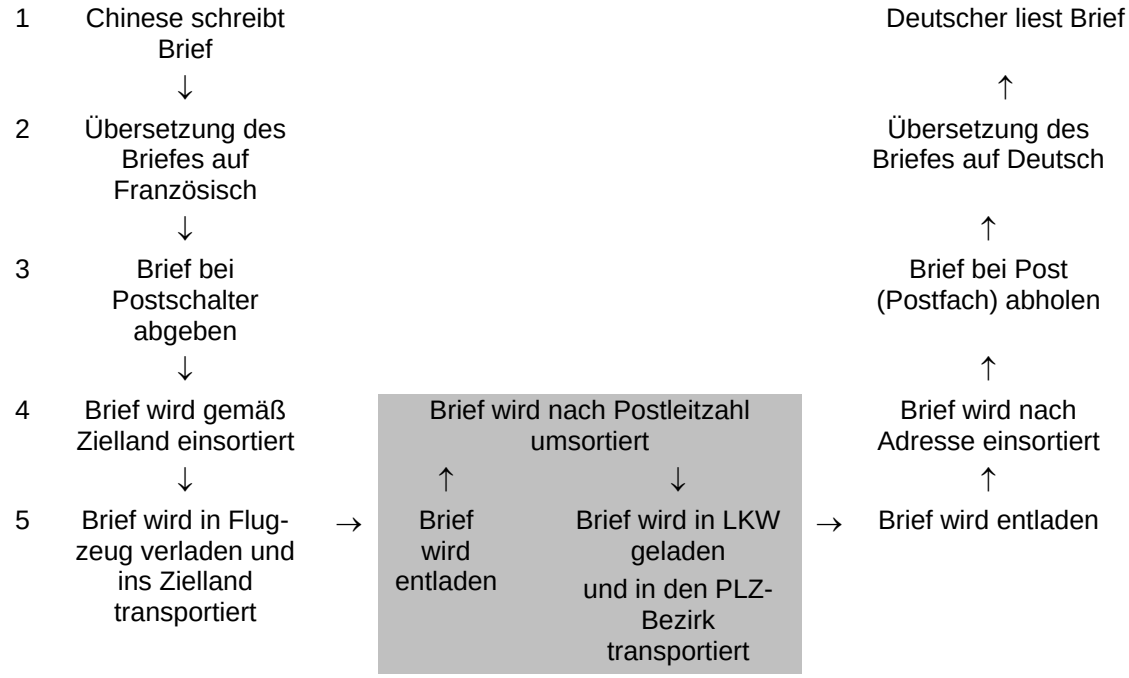
<table>
<tr><td>2. c)</td><td>3</td><td>Der Algorithmus nützt aus, dass jedes Produkt als Summe geschrieben werden kann.
$a \cdot b = (\underbrace{(0 + a) + \ldots + a)}_{b \text{ Summanden}}$, hier $4 \cdot 3 = \underbrace{(((0+4)+4)+4)}_{3 \text{ Summanden}}$
0 ist im obigen Fall der Ausgangswert der Variablen c, die Klammerung zeigt einen Term für das Ergebnis jedes Wiederholungsfalles, das in c zwischengespeichert wird, an.
Eine Hilfe zum Finden der Algorithmus-Idee ist das Erstellen einer Tabelle, in der die Werte der Variablen während des Algorithmusdurchlaufs aufgeführt sind, beispielsweise:
<table><tr><th></th><th>c</th><th>a</th><th>b</th></tr><tr><td>Vor Beginn des Durchlaufs</td><td>0</td><td>4</td><td>3</td></tr><tr><td>Nach dem 1. Wiederholungsdurchgang</td><td>4</td><td>4</td><td>2</td></tr><tr><td>Nach dem 2. Wiederholungsdurchgang</td><td>8</td><td>4</td><td>1</td></tr><tr><td>Nach dem 3. Wiederholungsdurchgang</td><td>12</td><td>4</td><td>0</td></tr><tr><td>Nach Ende des Durchlaufs</td><td>12</td><td>4</td><td>0</td></tr></table></td></tr>
<tr><td>2. d)</td><td>6</td><td><pre>1 load 20   2 add 18   3 store 20   4 load 19
5 dec   6 store 19 7 jap 1   8 end</pre></td></tr>
<tr><td>3. a)</td><td>2</td><td>Alle möglichen Buchstabenkombinationen werden im ungünstigsten Fall ausprobiert.</td></tr>
<tr><td>3. b)</td><td>3</td><td>26 Buchstaben; 6 x Ziehen mit Zurücklegen → 26^6 Möglichkeiten
Daher ergibt sich:
Zeitdauer: $t = 26^6/20\,000\,000$ s ≈ <u>15 s</u></td></tr>
<tr><td></td><td>40</td><td></td></tr>
</table>

Inf2. THEORETISCHE UND TECHNISCHE INFORMATIK

IV.

Auf-gabe	BE	Hinweise
1. a)	4	Schichtenmodelle dienen als Strukturierungsprinzip für den Aufbau komplexer Systeme. Grundlegende Prinzipien sind dabei: • Das Modell ist aus mehreren übereinander liegenden Schichten aufgebaut. • Jede Schicht stellt für die jeweils höhere Schichte bestimmte Dienste zur Verfügung, dabei werden aber Einzelheiten, wie die Dienste implementiert und angeboten werden, verborgen. Daraus ergeben sich folgende Vorteile: Reduzierung der Komplexität von Abhängigkeiten, beispielsweise in Hinblick auf übersichtliche und sauber definierte Schnittstellen, einfachere Wartbarkeit, leichtere Austauschbarkeit von Modulen usw. *Diese Lösung dieser Aufgabe ist ausführlich dargestellt.*
1. b)	8	Eine mögliche Umsetzung mit 5 Schichten: 

Mögliche Beschreibung des Kommunikationsablaufes:



- *Im obigen Lösungsvorschlag werden 5 Schichten verwendet. Die Aufgaben-formulierung ist hier bewusst bzgl. der Schichtenanzahl offen formuliert, da es bei vielen Lösungen eher schwierig ist, diese genau auf 4 Schichten zu reduzieren.*
- *Das obige Beispiel beinhaltet bereits den realistischen Fall des Umsortierens (graue Hinterlegung) und des Umladens des Briefes (mehrere Transportmittel). Eine Schülerlösung muss dies nicht enthalten. Sie muss lediglich schlüssig sein und alle vorher definierten Schichten verwenden.*

2. a)	3	Der kritische Bereich darf von den Robotern nur exklusiv genutzt werden, d. h., es ist wechselseitiger Ausschluss erforderlich. Realisiert werden kann dies durch das Semaphorprinzip, d. h. einen Sperrmechanismus, der Prozesse unter bestimmten Bedingungen blockiert.
2. b)	3	Eine Menge von Prozessen $\{P_1,\ldots,P_n\}$ sind in einer Verklemmung, wenn jeder Prozess auf ein Ereignis wartet, das nur von einem anderen Prozess ausgelöst werden kann.
2. c)	6	i. Das Lager ist (irgendwann) leer, der Montageroboter tritt in den kritischen Bereich ein und möchte ein Bauteil holen. Der Roboter kann im Lager kein Bauteil finden und deshalb den kritischen Bereich nicht verlassen. Er „wartet“ darauf, dass ein Bauteil geliefert wird. Der Zulieferroboter kann aber nicht in den kritischen Bereich eintreten, da dieser belegt ist. ii. Das Lager ist (irgendwann) voll, der Zulieferroboter tritt in den kritischen Bereich ein und möchte ein Bauteil ablegen. Dies ist nicht möglich, der Roboter kann aber den kritischen Bereich nicht verlassen. Er „wartet“ darauf, dass ein Bauteil abgeholt wird. Der Montageroboter kann aber nicht in den kritischen Bereich eintreten, da dieser belegt ist, und wartet ebenfalls.

2. d)	6	• Beim Zulieferroboter wird die in Teilaufgabe 2c aufgetretene Verklemmung vermieden. Begründung: Der Zulieferroboter fragt den Status ab: o Falls die Statusmeldung „Lager voll“ erfolgt, tritt der Roboter nicht in den kritischen Bereich ein. o Falls die Statusmeldung nicht erfolgt, kann er auf alle Fälle sein Bauteil im Lager ablegen. • Falls der Montageroboter zuerst abfragt, kann es zu einer Verklemmung kommen. Begründung: Das Lager ist leer; auf die Statusabfrage erhält der Roboter keine (aussagekräftige) Meldung, er schwenkt in das Bauteillager, und es tritt eine Situation analog der Antwort zu Teilaufgabe 2c auf.
3. a)	5	• Strategie 1: 19·t Millisekunden, weil 19 Elemente untersucht werden müssen. • Strategie 2: (ca.) 3·t Millisekunden Ein Durchspielen des Algorithmus ergibt folgende Situation, wobei die Nummern den Index des Feldelements angeben und das jeweils untersuchte Feldelement grau unterlegt ist:

1. Vergleich	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
2. Vergleich																16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
3. Vergleich																16	17	18	19	20	21	22								

Als Vergleichselement wird hier im Fall eines Feldes mit geradzahliger Elementanzahl das Element genommen, das sich links der rechnerischen Mitte befindet.

- *Ebenso kann natürlich auch das Feld rechts der rechnerischen Mitte genommen werden:*

1. Vergleich	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*
2. Vergleich																	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*
3. Vergleich																	*17*	*18*	*19*	*20*	*21*	*22*	*23*							
4. Vergleich																	*17*	*18*	*19*											
5. Vergleich																			*19*											

In diesem Fall wären 5 Vergleiche notwendig.

- *Es spielt in diesem Fall keine Rolle, ob der Index bei 0 oder 1 beginnt, da keine Implementierung erforderlich ist.*

3. b)	3	Die Suche bei Strategie 2 entspricht einer Suche in einem geordneten Binärbaum, bei dem jede Wurzel eines Teilbaums dem mittleren Element des im Teilbaum abgespeicherten Felds „entspricht“. Alle Pfade dieses Baums sind deshalb ungefähr gleich lang. Im ungünstigsten Fall befindet sich das gesuchte Objekt im letzten Knoten des Pfades.

3. c)	2	• Die Datenstruktur Feld ist für eine schnelle Suche nicht gut geeignet. • Bei einer Bibliothek weiß man nicht, wie viele Bücher letztendlich verwaltet werden müssen. Eventuell ist deshalb eine Vergrößerung der Elementanzahl bei „vollem“ Feld notwendig, was aber relativ umständlich ist. • Die Bücher sind nach der ISBN-Nummer geordnet. Das Einfügen eines neuen Buches innerhalb des Feldes ist ebenfalls relativ aufwendig. *Beide zuletzt genannten Probleme sind mit Hilfe eines Hilfsfeldes zwar lösbar, insgesamt ist das aber i. d. R. doch umständlich, da in ungünstigen Fällen sehr viele Kopieraktionen durchgeführt werden müssen.* *Für die Lösung der Aufgabe genügen zwei der drei angegebenen Argumente.*
	40	