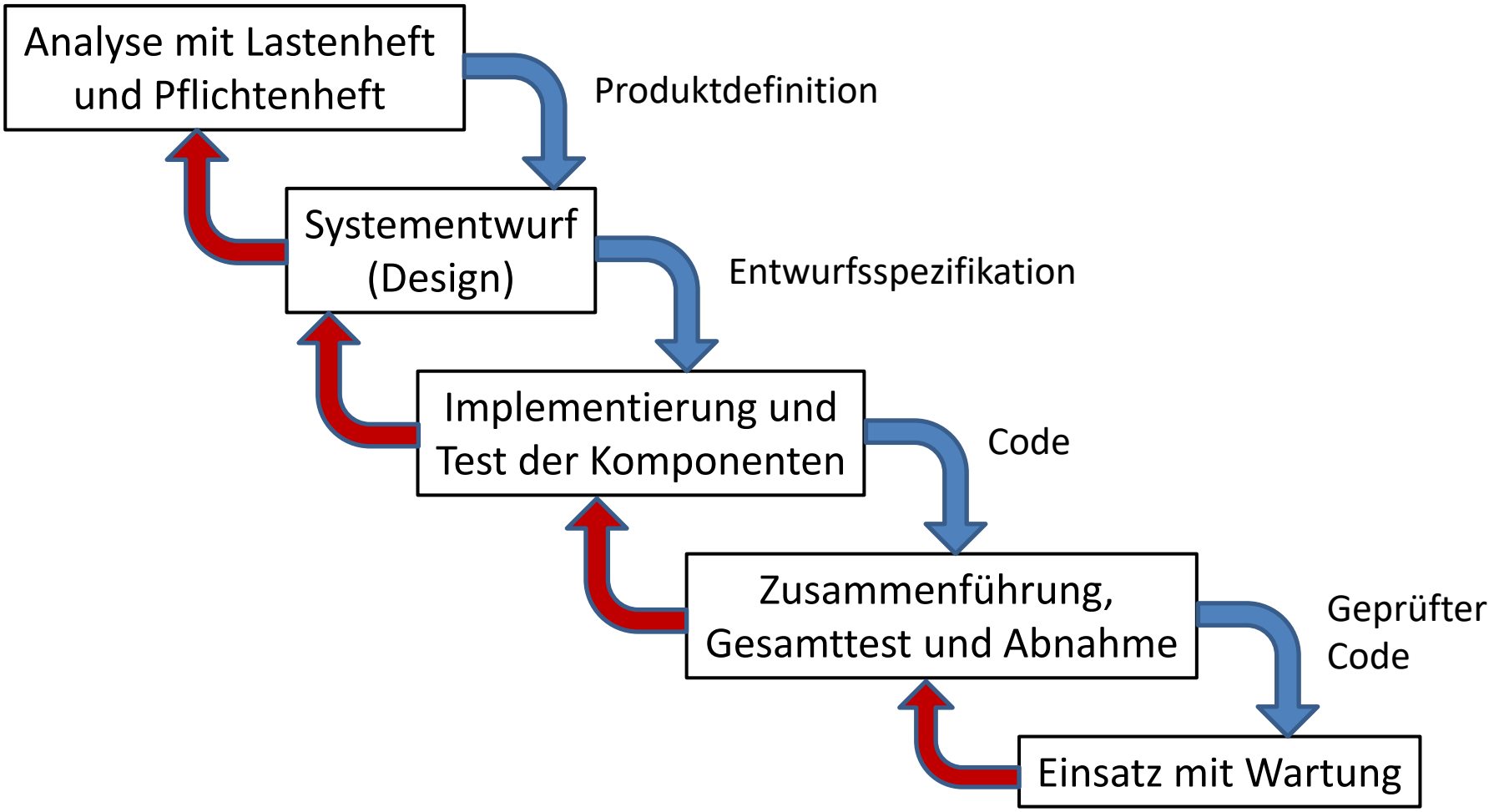
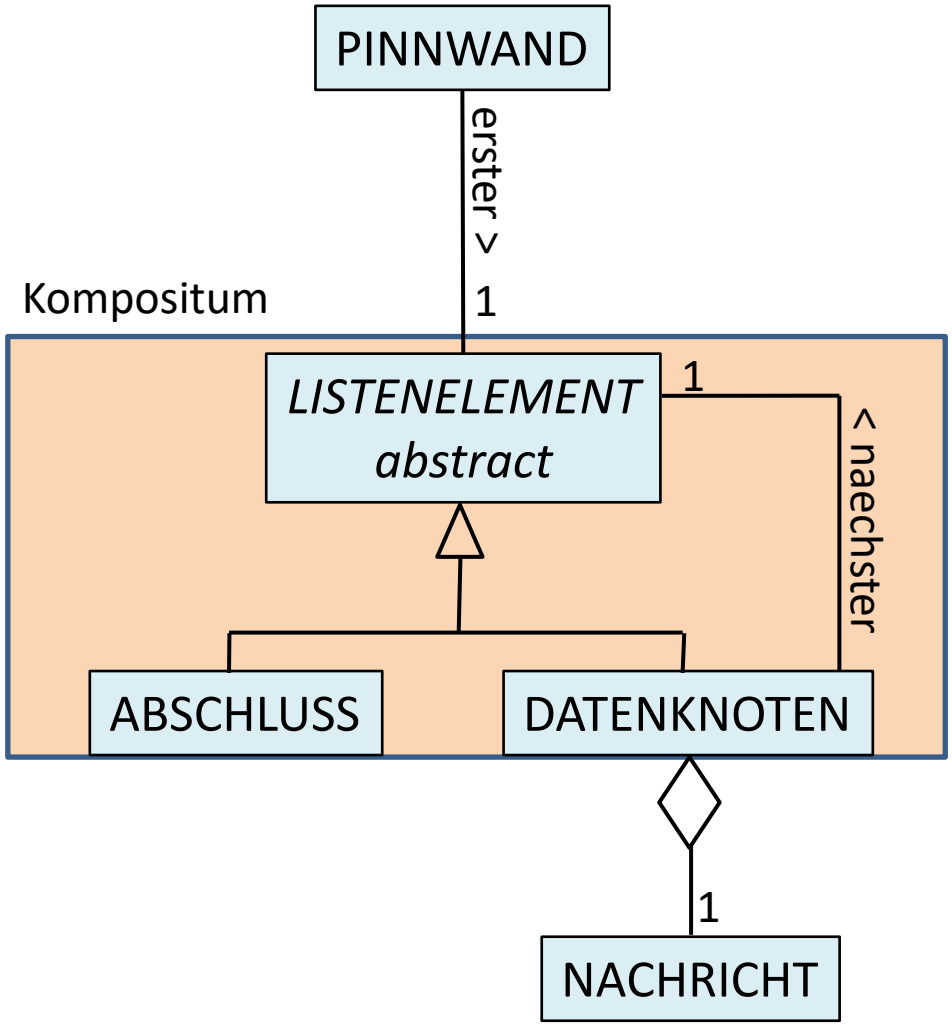
Abi Informatik 2018

I Modellierung und Programmierung (86%, ☺)

1a) **Erweitertes Wasserfallmodell**

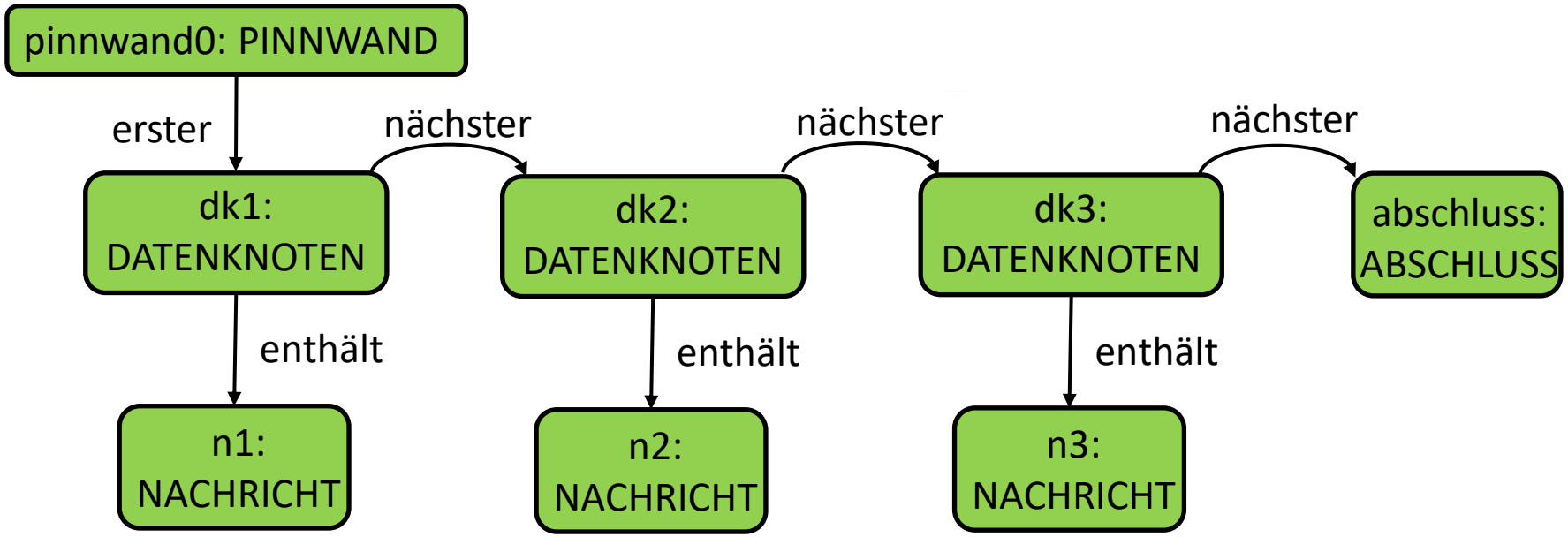


1b)


PINNWAND
erster >
1
Kompositum
LISTENELEMENT
abstract
1
< naechster
ABSCHLUSS
DATENKNOTEN
1
NACHRICHT

1c)
Mit einem Feld wäre die Anzahl der Nachrichten begrenzt, bei einer einfach verketteten Liste nicht.
Ein Feld mit großer Länge würde Speicherplatz verschwenden, eine verkettete Liste verbraucht nur den wirklich benötigten Speicherplatz.
Beim Entfernen von Nachrichten ist bei einem Feld aufwändiger als bei einer Liste.

1d)


pinnwand0: PINNWAND
erster
nächster
nächster
nächster
dk1: DATENKNOTEN
dk2: DATENKNOTEN
dk3: DATENKNOTEN
abschluss: ABSCHLUSS
enthält
enthält
enthält
n1: NACHRICHT
n2: NACHRICHT
n3: NACHRICHT

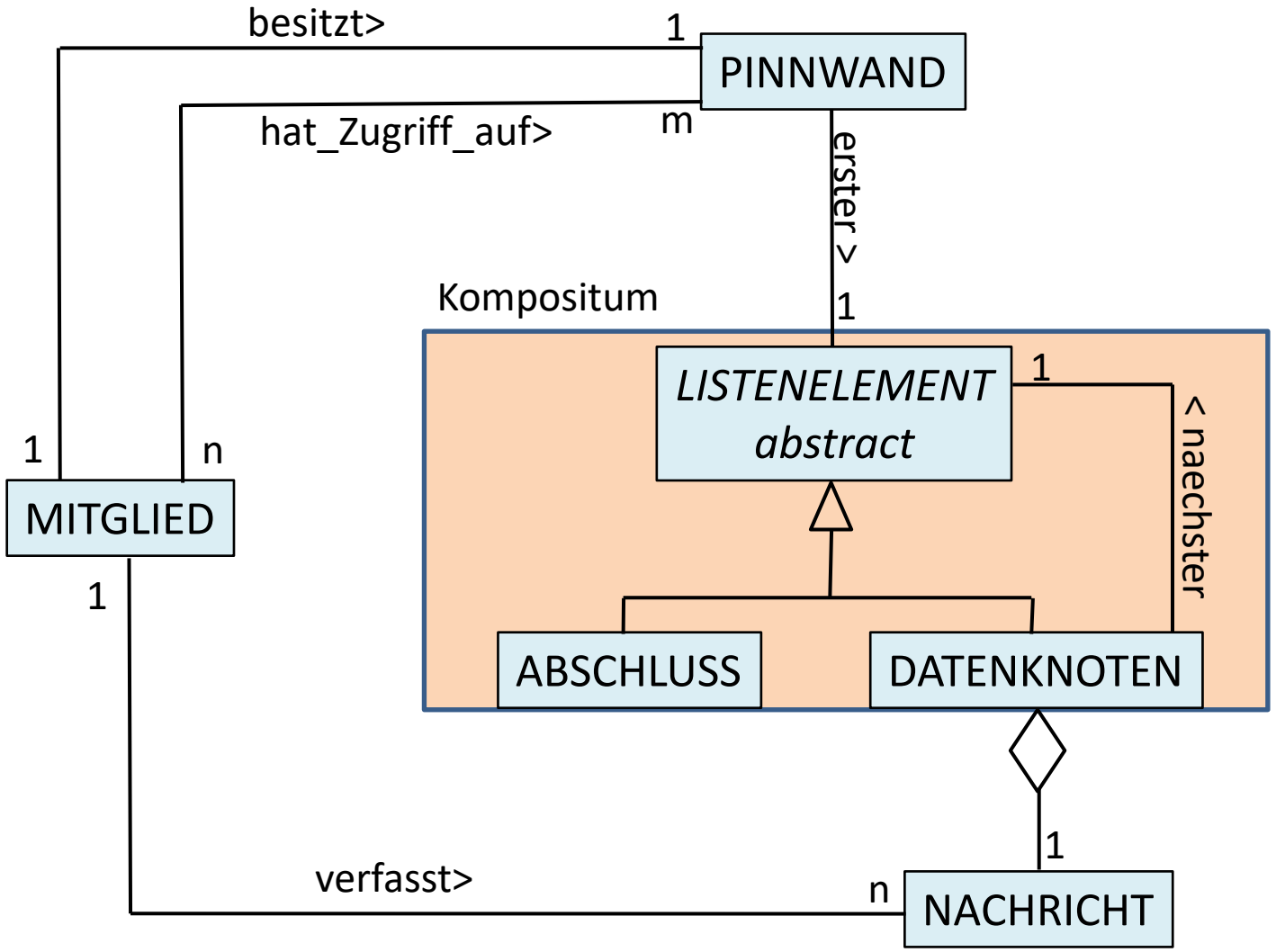
1e)
```
public class Pinnwand{
        private Listenelement erster;
        public int anzahlNachrichtenGeben(){
                return erster.anzahlDatenknotenGeben();
        }
}

public abstract class Listenelement{
        public abstract int anzahlDatenknotenGeben();
}

public class Datenknoten extends Listenelement{
        private Nachricht inhalt;
        private Listenelement naechster;
        public int anzahlDatenknotenGeben(){
                return naechster.anzahlDatenknotenGeben() + 1;
        }
}
```

```
public class Abschluss extends Listenelement {
        public int anzahlDatenknotenGeben(){
                return 0;
   }
```

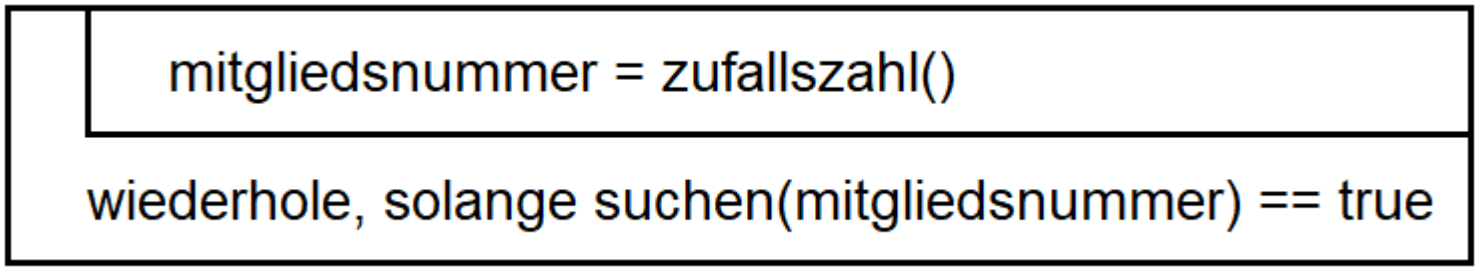
1f) Ergänze 1b)



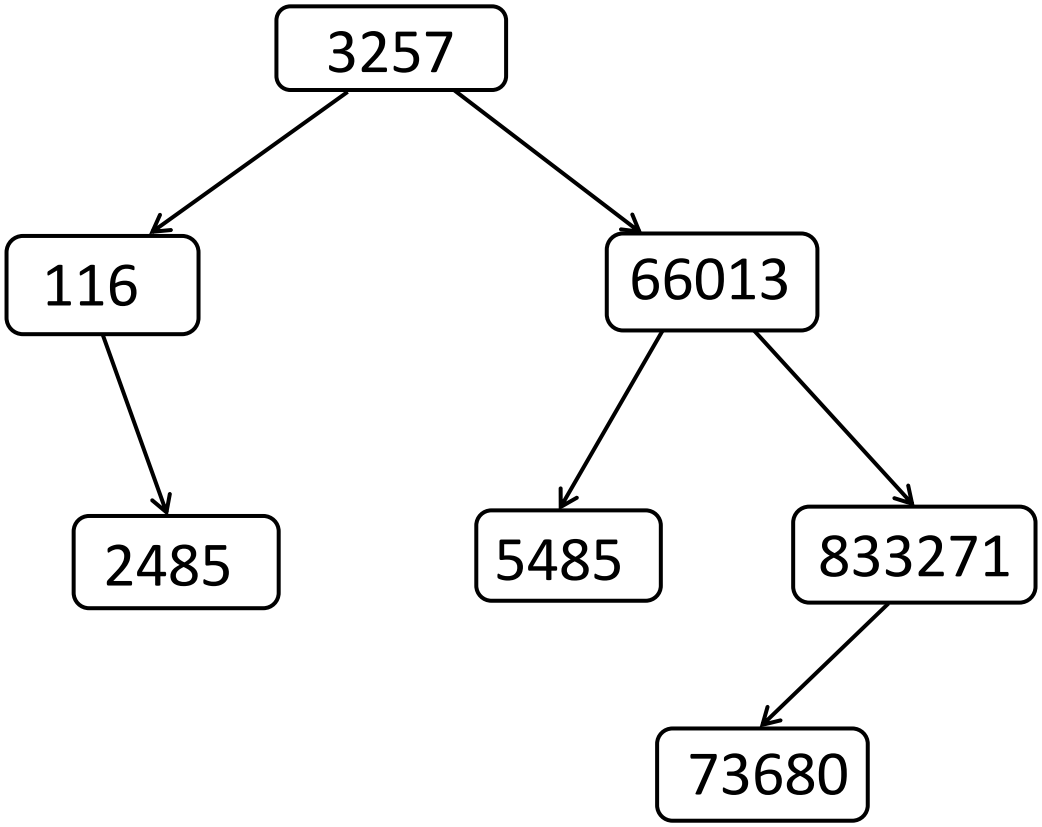
2a)

Wenn die Mitgliedsnummern in aufsteigender Reihenfolge in den Baum eingegeben werden, entsteht ein Baum, der zu einer Liste entartet ist. Die Vorteile des ausbalanzierten Binärbaums gegenüber der Liste durch schnellen Zugriff auf einzelne Elemente können dann nicht mehr genutzt werden.

2b)



2c)

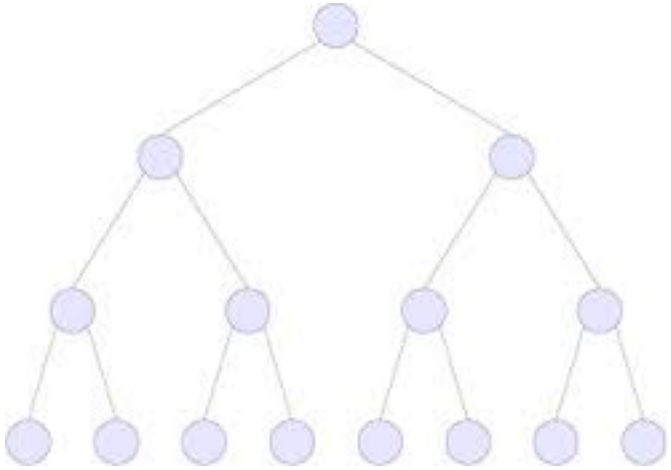


2d) 2485, 116, 5485, 73680, 833271, 66013, 3257

2e)

Wie viele Datenknoten bringt man unter in einem Binärbaum mit Ebenenzahl

1:	1	$=2^1 -1$
2:	1+2=3	$=2^2 -1$
3:	1+2+4=7	$=2^3 -1$
4:	1+2+4+8=15	$=2^4 -1$
n:	$1+2+4+\ldots+2^{n-1}$	$=2^n -1$



$2^n -1 = 250000 \rightarrow n = \log_2 250001 = 17{,}9\ldots$

Es sind mindestens 18 Ebenen nötig.

2f)

```
public class Baum {
private Baumelement wurzel;
public void mitgliederAusgeben(String suchnachname,
                                        String suchvorname){
        wurzel.mitgliederAusgeben(suchnachname,
                                        suchvorname);
}

public abstract class Baumelement{
        public abstract void mitgliederAusgeben(String snn,
                                                String svn);
}
```

```
public class Datenknoten extends Baumelement {
private Baumelement naechsterLinks;
private Baumelement naechsterRechts;
private Mitglied inhalt;

public void mitgliederAusgeben(String snn, String svn){
        naechsterLinks.mitgliederAusgeben(snn, svn);
        if (inhalt.istNachnameGleich(snn) &&
                inhalt.istVornameGleich(svn))
                inhalt.alleDatenAusgeben();
        naechsterRechts.mitgliederAusgeben(snn, svn);
}
```

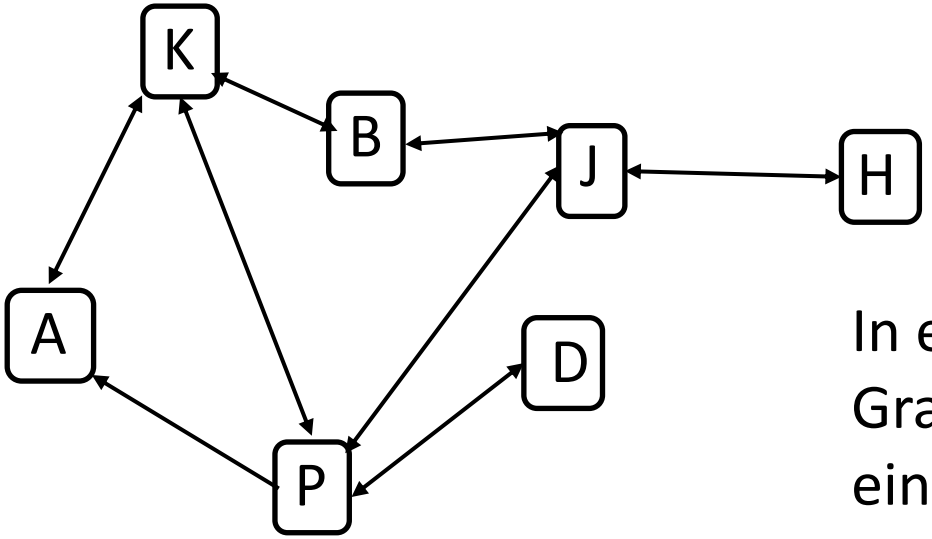
```
public class Abschluss extends Baumelement {

public void mitgliederAusgeben(String snn, String svn){
        }
```

2g)

```
SELECT *
FROM mitglieder
WHERE nachname = 'Meier' AND vorname = 'Laura';
```

3a)



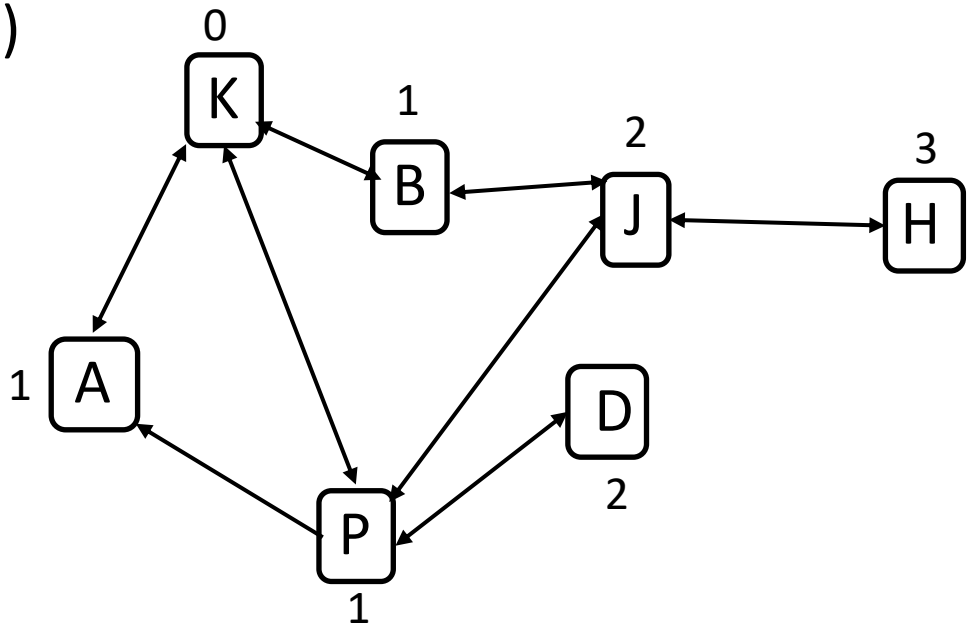
In einem ungerichteten Graphen könnte die Situation einer unbestätigten Freundschaftsanfrage nicht dargestellt werden.

3b) Adjazenzmatrix:

	A	B	D	H	J	K	P
A						x	
B					x	x	
D							x
H					x		
J		x		x			x
K	x	x					x
P	x		x		x	x	

3c)
Als Parameter wird ein beliebiger Knoten übergeben, z.B. derjenige mit dem Inhalt Jörg. Der Rückgabewert wird zunächst mit 0 belegt. Der Index des Knotens wird bestimmt. Dann wird geprüft, ob in der Adjazenzmatrix symmetrische Einträge mit diesem Index vorhanden sind, diese werden in der Variable rück gezählt.
Damit wird die Anzahl der Freunde der in dem Knoten gespeicherten Person bestimmt. Unbestätigte Anfragen werden nicht berücksichtigt.

3d)

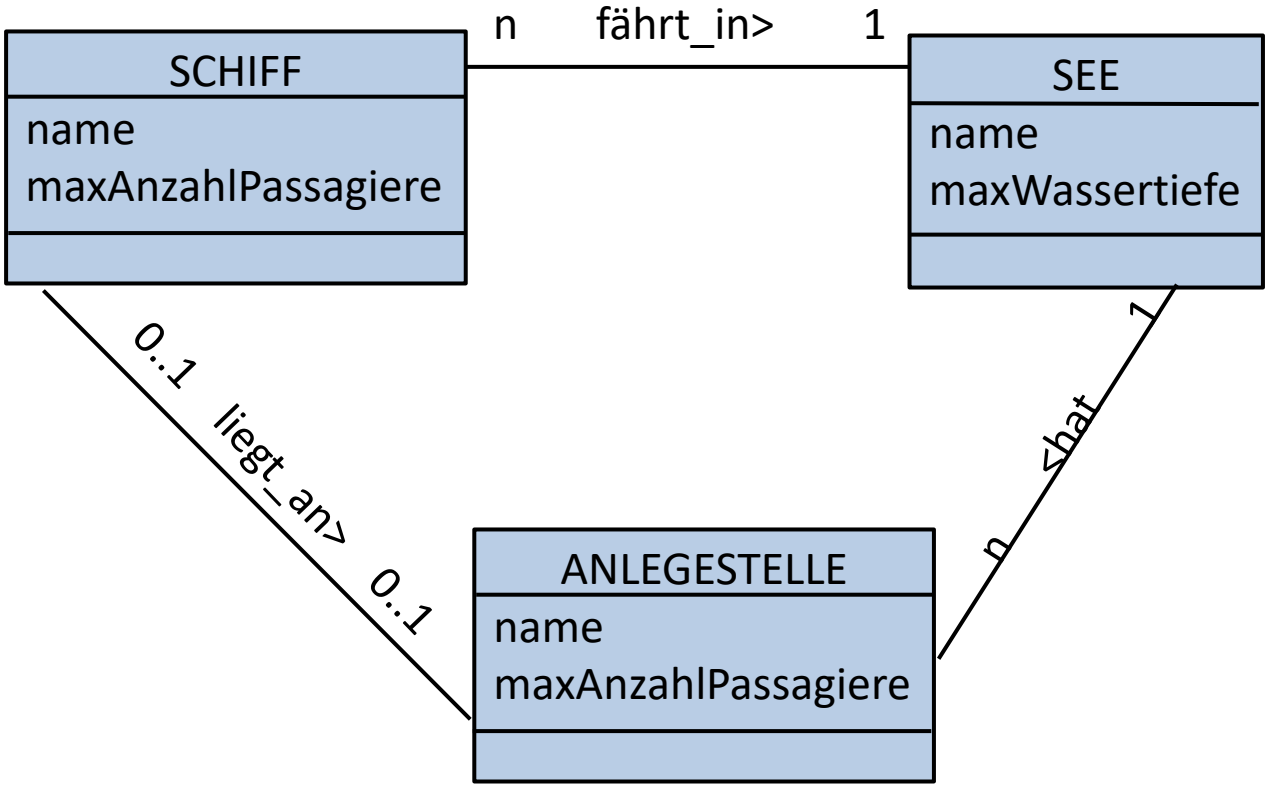


3e)

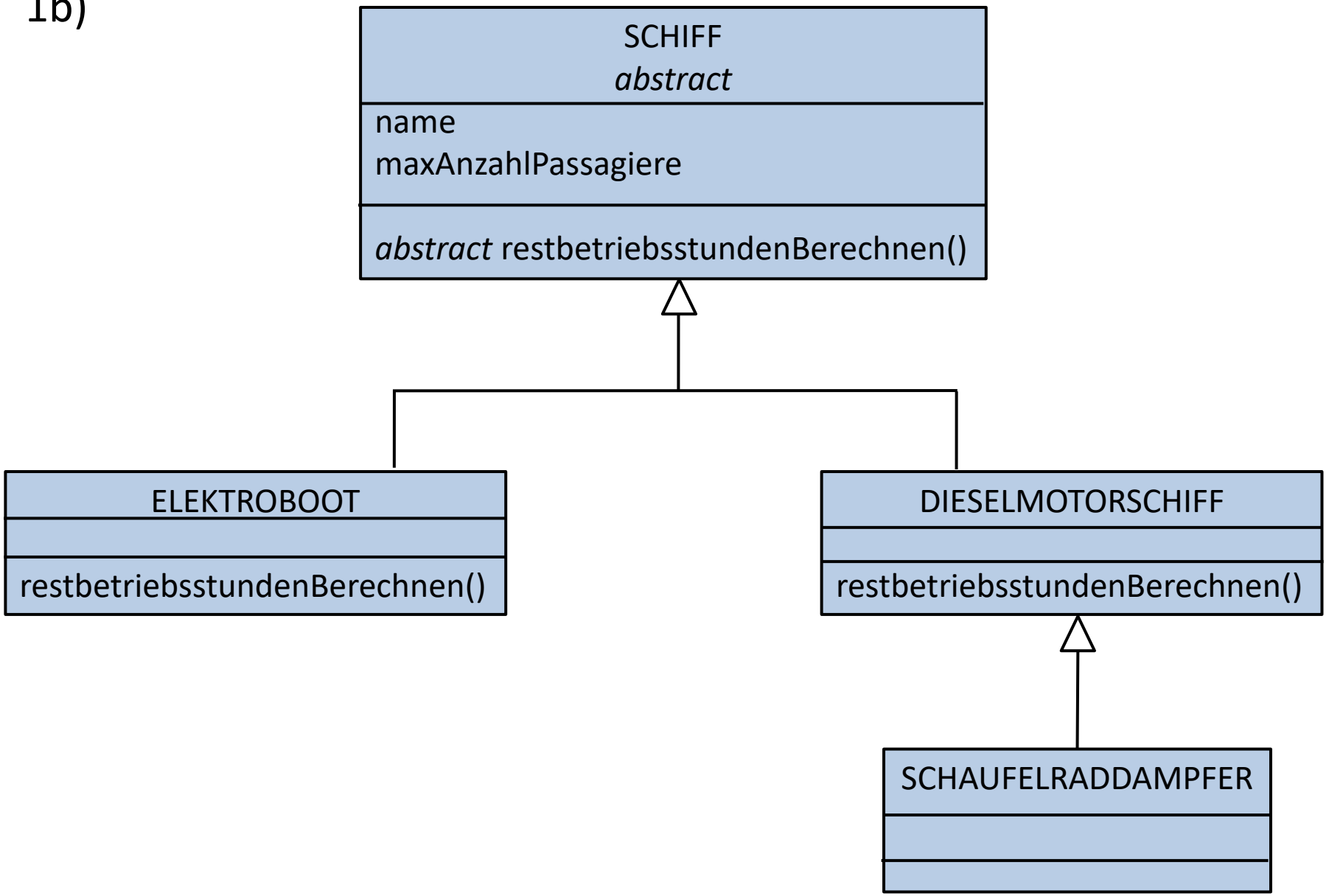
```
freundeVonFreunden(Knoten kn)
    i = index von kn im Feld knoten
        wiederhole für j von 0 bis n-1          //durchsuche Freunde
            wenn adjazenzmatrix[i][j]==true und adjazenzmatrix[j][i]==true
                wiederhole für k von 0 bis n-1  //durchsuche Freunde der Freunde
                    wenn (adjazenzmatrix[j][k]==true und
                                                adjazenzmatrix[k][j]==true)
                        und k<>i            //nicht die urspr. Person
                        und (adjazenzmatrix[i][k]==false oder
                                                    adjazenzmatrix[k][i]==false)
                                                //i und k sind nicht schon Freunde
                                                Gib k aus
                                            endeWenn
                                        endeWiederhole
                                endeWenn
                        endeWiederhole
endeMethode
```

II Modellierung und Programmierung (14%)

1) z.B.



1b)


SCHIFF
abstract
name
maxAnzahlPassagiere
abstract restbetriebsstundenBerechnen()
ELEKTROBOOT
restbetriebsstundenBerechnen()
DIESELMOTORSCHIFF
restbetriebsstundenBerechnen()
SCHAUFELRADDAMPFER

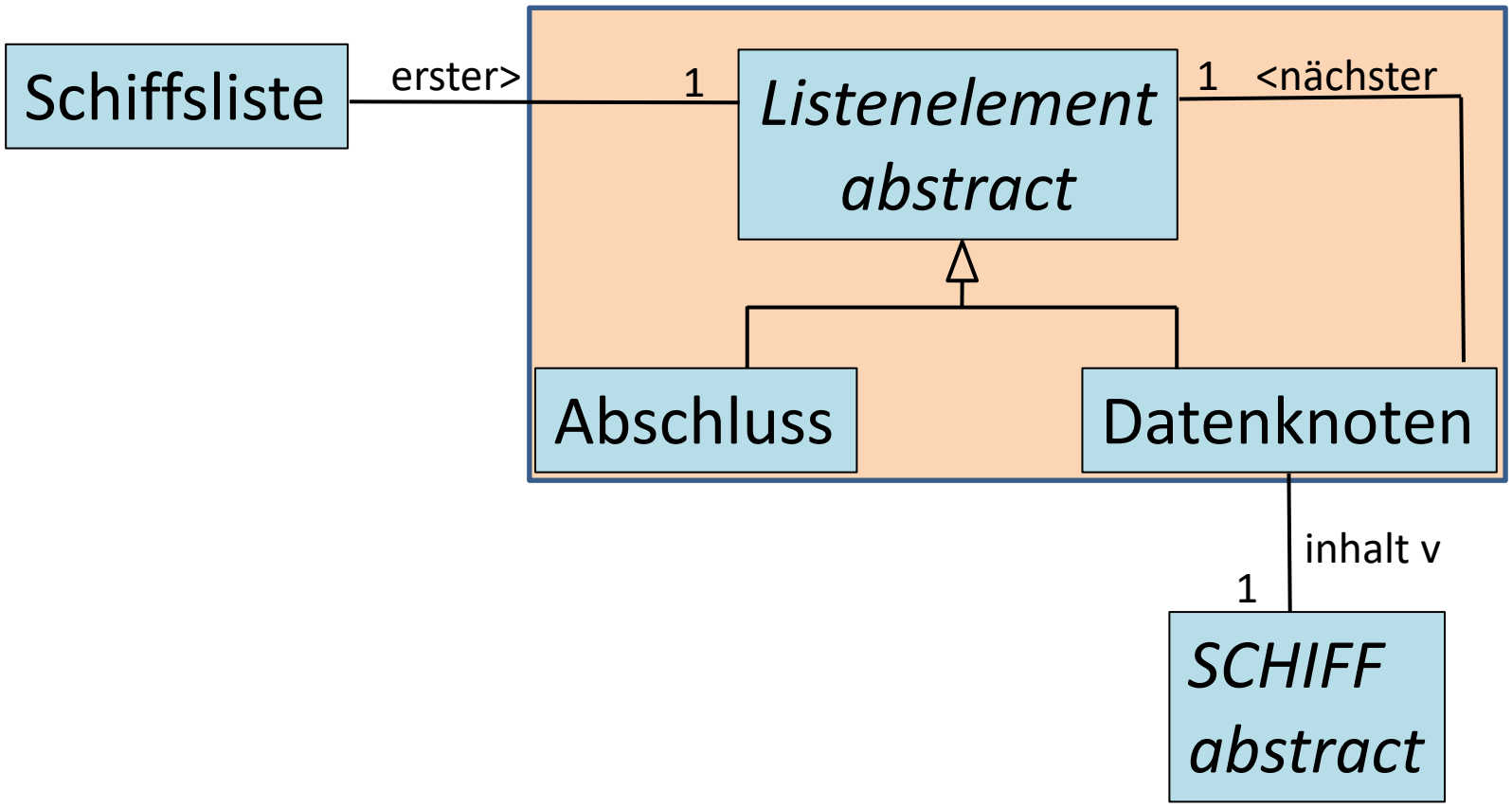
1c)

```
Datum wartungstermin;
public int anzahlTage(Datum saisonstart, Datum saisonende){
        if (wartungstermin.istVor(saisonstart) {
                return 0;
        }
        else {
                if (wartungstermin.istVor(saisonende){
                        return saisonstart.dauer(wartungstermin)-1;
                                //Wartungstag nicht mehr
                }
                else {
                        saisonstart.dauer(saisonende);
                }
        }
}
```

1d) Siehe Wikipedia:

Softwaremuster sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme. Sie stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem bestimmten Zusammenhang einsetzbar ist. Durch die Wiederverwendbarkeit wird die Problemlösung vereinfacht und beschleunigt.

1e) Sortierte Liste



1f)

```
public class Schiffsliste{
        private Listenelement erster;

        public int anzahlSchiffeBestimmen(){
                return erster.anzahlSchiffeBestimmen();
        }

        public void einfuegen(Schiff s){
                erster = erster.einfuegen(s);
}
```

```
public abstract class Listenelement{
        public abstract int anzahlSchiffeBestimmen();
        public abstract Listenelement einfuegen(Schiff s);
}
public class Datenknoten{
        private Listenelement nächster;
        private Schiff inhalt;

        public Datenknoten(Listenelement n, Schiff i){
                nächster = n;
                inhalt = i;
        }

        public int anzahlSchiffeBestimmen(){
                return nächster.anzahlSchiffeBestimmen()+1;
        }
```

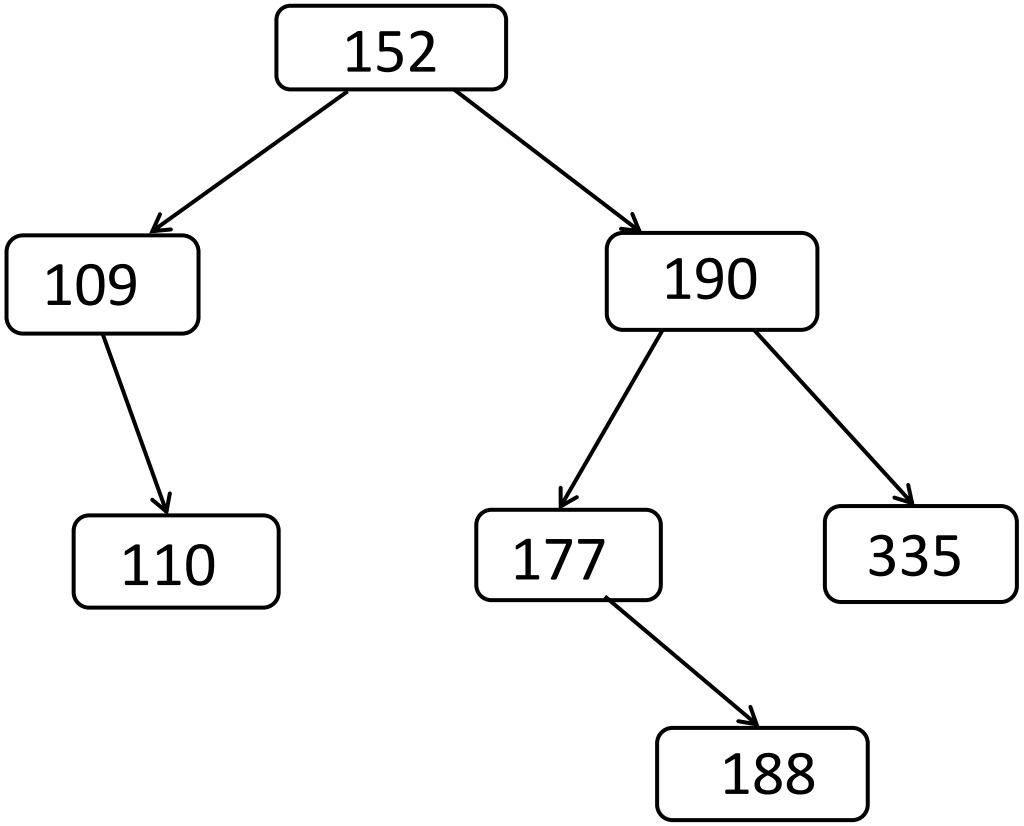
```
public Listenelement einfuegen(Schiff s){
    if(inhalt.wartungsterminGeben().istVor(s.wartungsterminGeben())){
            nächster = nächster.einfuegen(s);
            return this;
    }
    else {
            return new Datenknoten(this, s);
   }
}

public class Abschluss {
            public int anzahlSchiffeBestimmen() {
                        return 0;
            }
            public Listenelement einfuegen(Schiff s){
                        return new Datenknoten(this, s);
            }
}
```

2a)

```
SELECT persNr, name, vorname
FROM mitarbeiter
WHERE dienstort = 'Königssee';
```

2b)



Postorder:
110, 109, 188, 177, 335, 190, 152

2c)
Beim sortierten Einfügen entsteht ein zur einfach verketteten Liste entarteter Baum. Damit ist Einfügen und Suchennicht mehr von der Laufzeitordnung O(log(n)), sondern O(n) und damit viel langsamer. Verhindert werden kann dies durch eine andere Reihenfolge beim Einfügen, z.B. per Zufallszahl oder nach einem anderen Ordnungskriterium.

3a) Der Graph ist gerichtet und gewichtet.
Adjazenzmatrix:

	B/F	HI	P/S	HS	Ü/F	C	S	G	NS
B/F		23							
HI	23		15	10					
P/S		15					x		
HS		10			25		30	10	
Ü/F				25					
C				30					
S						30			
G				10					5
NS								5	

3b) Zyklus:

PS – HI – B/F – HI – HS – S – C – HS – G – HS – Ü/F – HS – HI – PS

Dauer in min:

15+23+23+10+30+30+30+10+10+25+25+10+15+12*10 = 376

Das sind 6 h 16 min. (12 Haltestellen mit 10 min Wartezeit).

3c)

```
public class Liniennetz{
        private Knoten[] knoten;
        private int[][] adjazenzmatrix;
        private int anzahl;

        public Liniennetz(int n){
                knoten = new Knoten[n];
                adjazenzmatrix = new int[n][n];
                anzahl = 0;
                for (int i= 0; i<anzahl; i++) {
                        for (int j= 0; j<anzahl; j++) {
                                adjazenzmatrix[i][j] = -1;
                        }
                }
        }
}
```

3d)

```
public int indexSuchen(Knoten k) {
        int index = -1;
        for(int i=0; i<anzahl;i++) {
                if(knoten[i].equals(k)) index = i;
        }
        return index;
}
```

3e) in Graph:

```
public void traversieren(Knoten start){
        //alle Knoten auf unbesucht setzen
    for (int i=0;i<anzahl;i++){
       knoten[i].markierungSetzen(false);
    }
    //System.out.println("Reihenfolge bei Tiefensuche:");
    tiefensucheKnoten(indexSuchen(start));
  }
public void tiefensucheKnoten(int vIndex){
    knoten[vIndex].markierungSetzen(true);
    System.out.println(knoten[vIndex].inhaltGeben().datenGeben());
    for (int i=0;i<anzahl;i++){
       if (adjazenzmatrix[vIndex][i]&& knoten[i].markierungGeben()){
          tiefensucheKnoten(i);
       }
    }
 }
```

```
public class Knoten{
    …
    private boolean markierung;

    …

    public void markierungSetzen(boolean m){
        markierung = m;
    }

    public boolean markierungGeben(){
        return markierung;
    }
}
```

III Theoretische und Technische Informatik (42%, ☺)

1a) Rechenausdruck = Strichoperand {('–'|'+')Strichoperand}
Strichoperand = Punktoperand {('*'|'/')Punktoperand}
Punktoperand = Zahl | Variable | '(' Rechenausdruck ')'
Zahl = Ziffer {Ziffer}
Variable = 'x' | 'y' |'z'
Ziffer = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

2a) 0110 wird akzeptiert.
1001 wird nicht akzeptiert.
0101 wird nicht akzeptiert.
1010 wird akzeptiert.

2b) Der Automat akzeptiert alle 4-Bit-Binärzahlen, die nicht mit 00 beginnen und mit einer 0 enden.

2c)

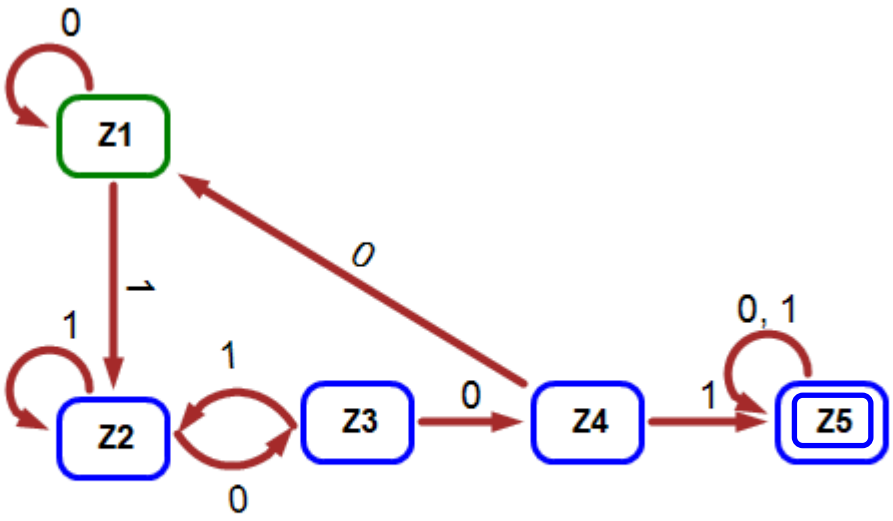
```
public class Automat{
private int zustand;

public boolean binaerzahlPruefen(String eingabe){
        zustand = 0;
        boolean akzeptiert = false;
        for (int i=0;i<eingabe.laenge(); i++) {
                zustandWechseln(eingabe.zeichenAn(i));
        }
        if (zustand == 5) {
                akzeptiert = true;
        }
        return akzeptiert;
}
```

2c)

```
public void zustandWechseln(char zeichen){
switch(zustand){
        case 2: {
                switch(zeichen) {
                        case '0': {zustand = 6;} break;
                        case '1': {zustand = 3;} break;
                }
        } break;
...
}
```

3)



4a) **Befehlszyklus**

- Fetch-Phase I
  Befehl holen (entsprechend der Adresse im Befehlszähler, hier Zahl die sub entspricht)
  BZ = BZ + 1
- Decode-Phase
  Opcode des Befehls bestimmen: sub
- Fetch-Phase II
  Operand laden: hier 100
  BZ um die Anzahl der gelesenen Speicherzellen weiter schalten (hier 1)
- Execute-Phase
  vom Wert im Akkumulator den Speicherinhalt von 100 subtrahieren, Ergebnis in Akkumulator schreiben;

4b) Programm und Daten sind in einem Speicher.

4c)

Befehl	BZ	A	SR	100	101	102
	0			4	5	
load 100	2	4		4	5	
cmp 101	4	4	N	4	5	
jmpnn 12	6	4	N	4	5	
load 101	8	5	N	4	5	
store 102	10	5	N	4	5	5
jmp 14	14	5	N	4	5	5
hold	16	5	N	4	5	5

Es wird die größere der beiden Zahlen aus 100 und 101 in 102 kopiert.

4d) n in Speicherzelle 100

```
schleife:   load 100
            cmpi 3
            jmpn ende
            modi 4
            (cmpi  0)
            jmpnz falsch1
            load 100
            divi 4
            store 100
            jmp schleife
falsch1:    load 100
            modi 2
            (cmpi 0)
            jmpnz falsch2
            load 100
            divi 2
            addi 1
            store 100
            jmp schleife
falsch2:    load 100
            addi 1
            divi 2
            store 100
            jmp schleife
ende:       load 100
            hold
```

4e)
Dann könnte es zu einer Endlosschleife kommen, denn wenn n=2 ist, dann wird 2/2+1 berechnet und in n gespeichert. Dies ist wieder 2, d.h. der Algorithmus terminiert nicht.

5a)
Fahrzeug 1 muss denn Gegenverkehr abwarten (FZ 3), da es links abbiegen will.
Fahrzeug 3 muss FZ 2 den Vortritt lassen, da dieses von rechts kommt.
FZ 2 muss FZ1 zuerst fahren lassen, da dieses von rechts kommt.
Jedes FZ wartet also auf ein anderes, zyklische Wartebedingung.
Lösung: Durch Winken dieses zyklische Warten unterbrechen, z.B. könnte FZ 3 FZ1 vor sich abbiegen lassen.

5b) Ein kritischer Abschnitt ist ein Bereich, der nur im wechselseitigen Ausschluss betreten werden darf.
Nebenläufige Prozesse können in der Informatik mit Hilfe eines Monitors synchronisiert werden. Dieser achtet darauf, dass auf kritische Abschnitte jeweils nur von einem Prozess zugegriffen wird.

IV Theoretische und Technische Informatik (58%)

1a)

Rechnungskennzahl = B B '-' Z ('P'|'G'('0' zn0 |'1' zk3)) '-' Z

B = 'A' | 'B' | … | 'Z'

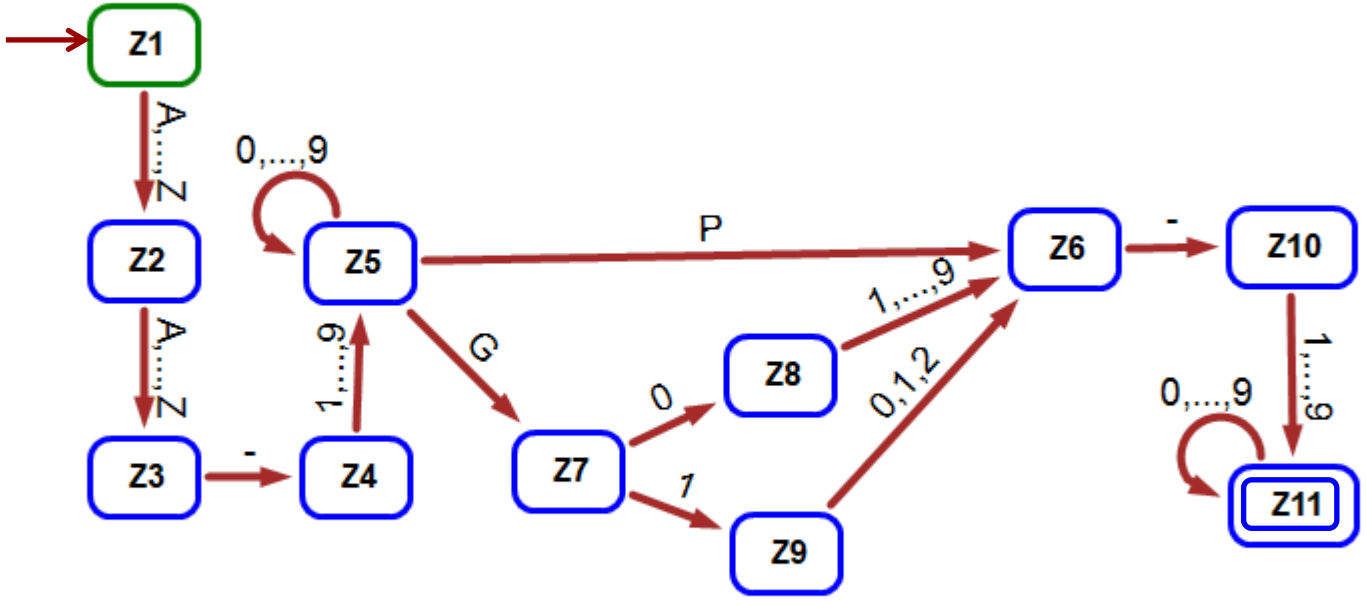
Z = zn0 {ziffer}

zn0 = '1' | '2' | '3' | … | '9'

ziffer = zn0 | '0'

zk3 = '0' | '1' | '2'

1b)



1c) Zustandswechsel (oder Ableitung): WA-27P-92
z1 führt mit 'W' in z2, mit 'A' in z3, mit '-' in z4, mit '2' in z5, mit '7' bleibt man in z5, mit 'P' in z6, mit '-' in z10, mit '9' in z11, mit 2 bleibt man in z11.
Damit ist das Wort abgearbeitet und man befindet sich in einem Endzustand.

1d)

```
public class Kennzahltester{
private int zustand;

public boolean kennzahlTesten(String kennzahl){
        zustand = 0;
        boolean akzeptiert = false;
        for (int i=0;i<kennzahl.laenge(); i++) {
                zustandWechseln(kennzahl.zeichenAn(i));
        }
        if (zustand == 11) {
                akzeptiert = true;
        }
        return akzeptiert;
}
```

```
public void zustandWechseln(char zeichen){
switch(zustand){
        case 7: {
                switch(zeichen) {
                        case '0': {zustand = 8;} break;
                        case '1': {zustand = 9;} break;
                        default: {zustand = -1;} break;
                }
        } break;
...
}
```

2a)
Es gibt eine zyklische Kette von Prozessen, von denen jeder mindestens eine Ressource belegt, die der nächste Prozess in der Kette benötigt. (zyklische Wartebedingung). Dabei sind die Ressourcen nur exklusiv nutzbar und können nicht entzogen werden. Ein Vorgang, der bereits Betriebsmittel belegt, kann weitere anfordern.
Wenn bei einer Rechts-Vor-Links-Kreuzung von allen Seiten gleichzeitig ein Fahrzeug kommt, muss jedes Fahrzeug seinem rechten Nachbarn die Fahrt gewähren lassen. Daher kommt es zu einer Verklemmung.

2b)
Mit einer Ampel wird nun nur für zwei aus gegenüberliegenden Richtungen kommende Fahrzeuge die Fahrt erlaubt. Beide können gleichzeitig geradeaus fahren und auch gleichzeitig abbiegen.
Wenn nur einer geradeaus fährt hat dieser automatisch Vorrang vor dem anderen.

3a) **Befehlszyklus für sub 102**

Da der Befehlszähler im Programm b) offensichtlich nur um 1 weiter geschaltet wird, gibt es hier nur eine Fetch-Phase

- Fetch-Phase
  Befehl und Operand holen (entsprechend der Adresse im Befehlszähler, hier Zahl die sub entspricht)
  BZ = BZ + 1
- Decode-Phase
  Opcode des Befehls bestimmen: sub
- Execute-Phase
  vom Wert im Akkumulator den Speicherinhalt von 102 subtrahieren, Ergebnis in Akkumulator schreiben;

3b)

Befehl	BZ	Akku	10115	102	103	104
	1		15	23	7	
load 101	2	15	15	23	7	
sub 102	3	-8	15	23	7	
jlt 10	10	-8	15	23	7	
load 102	11	23	15	23	7	
sub 103	12	16	15	23	7	
jlt 16	13	16	15	23	7	
load 102	14	23	15	23	7	
store 104	15	23	15	23	7	23
jmp 18	18	23	15	23	7	23
end						

Wert in Speicherzelle 104: 23

Allgemein:

Das Maximum der drei Zahlen in 101, 102 und 103 wird ermittelt und in 104 gespeichert.

3c)
Die Methode berechnet die Summe der Quadrate von 1 bis 10.
Rückgabewert in Speicherzelle 100

```
1: dload 0
2: store 100
3: dload 10
4: store 101
5: dload 1
6: store 102
7: load 101
8: jle 13
9: mul 101
10: add 100
11: store 100
12: load 101
13: sub 102
14: store 101
15: jmp 7
16: load 100
17: end
```