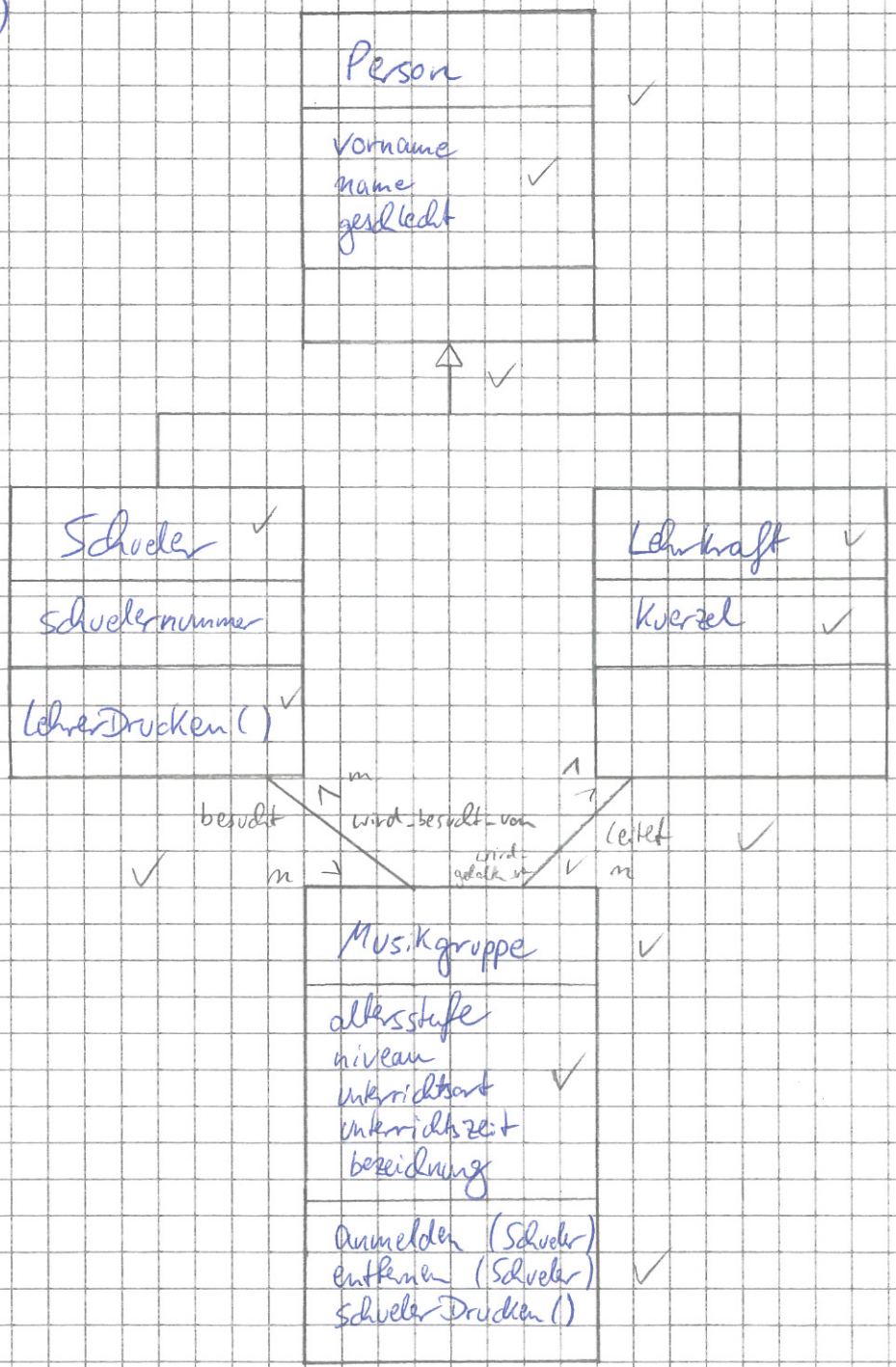
Abiturprüfung Informatik 2015

Erwartungshorizont

Inf 1 - II

1.)



12

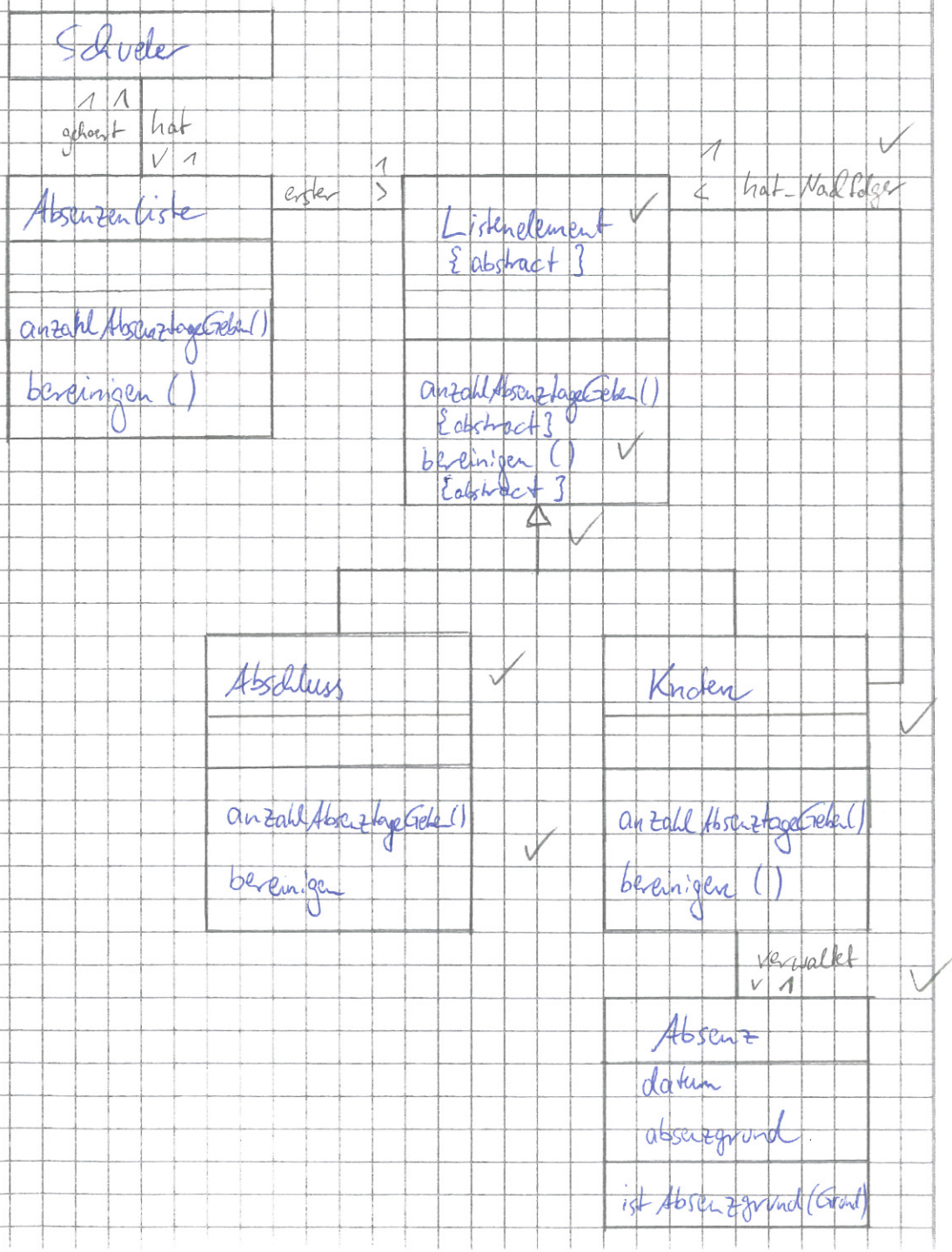
2a) Beim Feld wäre nur eine festgelegte Anzahl von Terminen eintragbar. Wird das Feld stets groß dimensioniert, wird unnötig (bei weniger Terminen) Speicher verbraucht. Diese Nachteile fallen bei der verketteten Liste weg.

4

Zudem entfallen in der Liste Verschiebeoperationen beim Löschen eines Termins.

b) In Warteschlangen werden Einträge vorne entnommen und am Ende eingefügt. Die Abarbeitung erfolgt also in der Reihenfolge des Eintragens (Fifo).
Da die Termine nicht zwingend in der zeitlichen Reihenfolge eingetragen werden, wie sie im Kalender erscheinen müssten, ist diese Art der Datenspeicherung für den Terminkalender ungeeignet.

3.)



Gründe: 1.) Struktur und Inhalt werden getrennt ✓
(⇒ bessere Wiederverwendbarkeit)
2.) Übersichtlicher durch Aufteilung der Aufgaben ✓

b) Absenzliste:

(*)
```
public int anzahlAbsenztageGeben () {
    return erster.anzahlAbsenztageGeben (); ✓
}

public void bereinigen () {
erster = erster.bereinigen (); ✓
}
}
```

17

(*)
```
public class Absenzliste ✓ {
    private Listenelement erster; ✓
```

Listenelement

```
public abstract ✓ class Listenelement {
    ~~private Listenelement~~
    public abstract int anzahlAbsenztageGeben (); ✓
    public abstract ~~void~~ Listenelement bereinigen (); ✓
}
```

Knoten

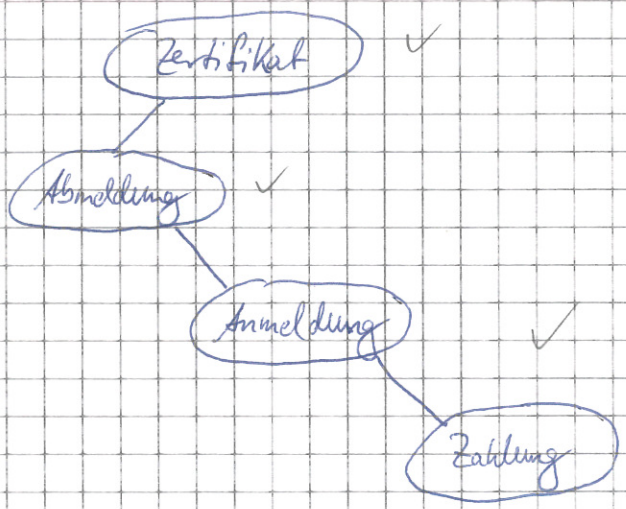
```
public class Knoten extends Listenelement ✓
    private Listenelement nachfolger; ✓
    private Absenz inhalt; ✓
    public int anzahlAbsenztageGeben () {
        return 1 + nachfolger.anzahlAbsenztageGeben (); ✓
    }
    public ~~void~~ Listenelement bereinigen ()
        nachfolger = nachfolger.bereinigen (); ✓
```

```
        if (inhalt.istAbszeugnund("öffentliche Auffuhrung"))
            return nachfolger;
        }
        else {
            return this;
        }
    }
```

Abschluss

```
public class Abschluss extends Listenelement {
    public int anzahlAbszeugnisseGeben() {
        return 0;
    }
    public Listenelement bereinigen() {
        return this;
    }
```

4a)



Der bisherige Baum stellt eine Liste dar; die Wahrscheinlichkeit für einen balancierten Binärbaum ist (zumindest bis zur Ebene „Anmeldung" sehr gering.
⇒ Die Gefahr, dass der Baum schlecht balanciert ist und damit der Vorteil bei der Suche gegenüber einer Liste abnimmt, ist relativ hoch.

7

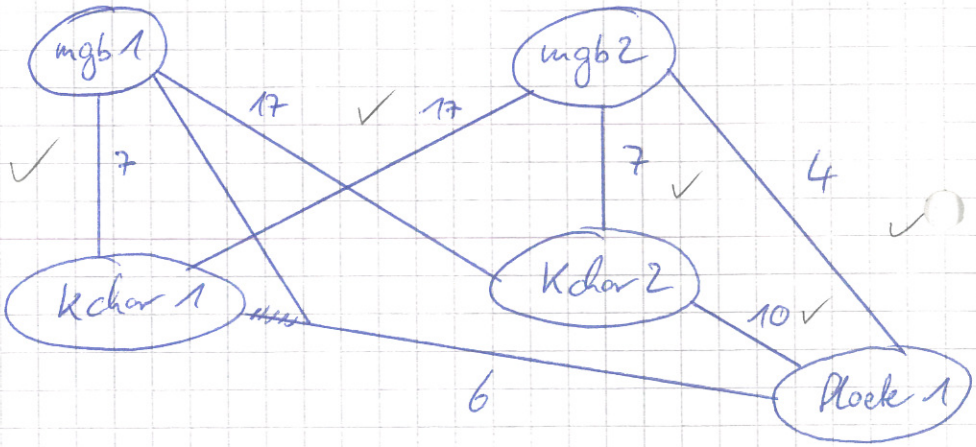
b) ~~public class~~ Knoten

4

```
public void alleZeigen()
    linkerNachfolger.alleZeigen();
    ~~System~~inhalt.ausgeben();
    rechterNachfolger.alleZeigen();
}
```

5a)

8



	mgb1	mgb2	Kchar1	Kchar2	Floete1
mgb1			7	17	6
mgb2			17	7	4
Kchar1	7	17			~~10~~
Kchar2	17	7			10
Floete1	6	4	~~0~~	10	

2 b) Der Graph ist ungerichtet, gewichtet (und zusammenhängend)

3 c) In einem unzusammenhängenden Graphen gäbe es im Zshg. disjunkte Musikgruppen.

d) Im Beispiel: $n = 5$; $m = 2$

8

$i = 0$: b = wahr
$j = 0$: i = indizes[0] $\Rightarrow$ b = falsch

j = 1 : matrix [0] [3] > 18 ⇒ b = false

Nichts ausgeben

i = 1 : b = wahr ✓

j = 0 : matrix [1] [0] = 0

j = 1 : matrix [1] [3] = 0

⇒ Ausgabe : piano 2

i = 2 : b = wahr ✓

j = 0 : matrix [2] [0] > 0 ⇒ b = false

⇒ Nichts ausgeben

i = 3 : b = wahr ✓

j = 0 : matrix [3] [0] = 0

j = 1 : i = indizes [1] ⇒ b = false

⇒ Nichts ausgeben

i = 4 : b = wahr ✓

j = 0 : matrix [4] [0] > 0 ⇒ b = false

j = 1 : matrix [4] [3] --

⇒ Nichts ausgeben

Allgemein: Es werden alle Musikgruppen ausgegeben, ~~die~~ in denen keine ✓ Mitglieder aus den übergebenen ✓ Musikgruppen spielen. ✓

Inf 2 - III

1a) WENN $u < 200000$, DANN $P = 0{,}002 * u$

SONST

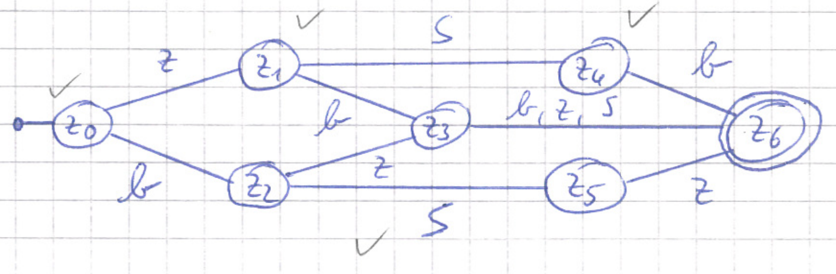
3 WENN $u < 400000$, DANN $P = 0{,}003 * u$

SONST

$P = 0{,}004 * u + 1000$

b)
```
1   LOAD 101
2   SUBI 200000
3   JLE 12
4   SUBI 200000
5   JLE 17
6   LOAD 101
7   MULI 4
8   DIVI 1000
9   ADDI 1000
10  STORE 102
11  JMP 21
12  LOAD 101
13  MULI 2
14  DIVI 1000
15  STORE 102
16  JMP 21
17  LOAD 101
18  MULI 3
19  DIVI 1000
20  STORE 102
21  HOLD
```
7

5 2a)



3 b) Passwort = b(s|b)z | bz(s|b|z) |
zb(s|b|z) | z(s|z)b

c)
8
```
public class PasswortTester {
    private int zustand;
    public boolean passwortTesten(String eingabe) {
        for (int i=0; i< eingabe.length; i++) {
            zustandWechseln (eingabe.charAt(i));
```

```
    }
    return (zustand == 6);
}
private void zustandWechseln (char z) {
    switch (zustand) {
        case 0: {
            if (z == 'b')  zustand = 1;
            else if (z == 'Z')  zustand = 2;
                else zustand = -1;
            break; }
        case 1: {
            if (z == 'Z')  zustand = 2;
            else zustand = 4;
            break; }
        ...
    }
}
```

d) Die Anzahl der möglichen Wörter wird durch diese Maßnahme erhöht. ⇒ Sicherer gegen Brute-Force-Attacken 2

e) Maßnahmen: • Längeres Passwort
• mehr Zeichen verwenden (z.B. Großbuchstaben) 5

⇒ $2 \cdot 26 + 10 + 32 = 94$ mögliche Zeichen pro Stelle

Annahme: 10 Zeichen

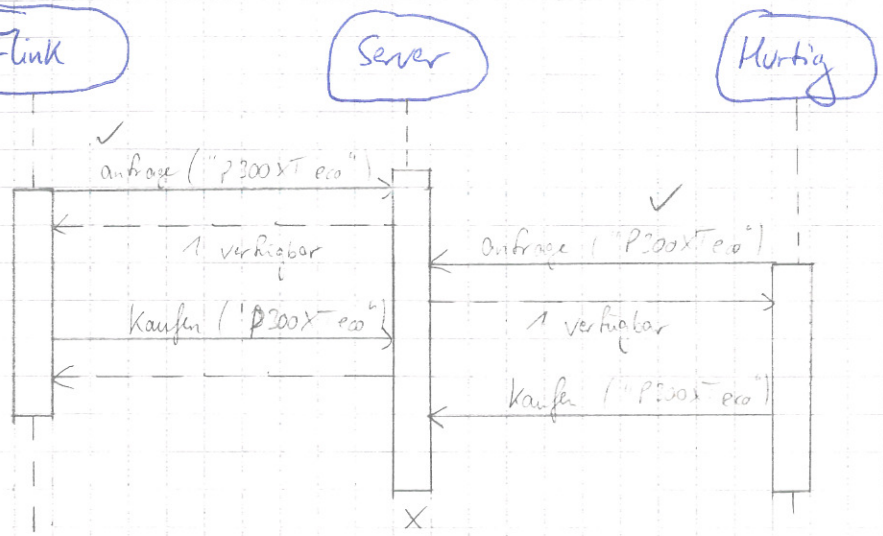
⇒ 94^{10} Möglichkeiten

Um alle Möglichkeiten zu testen: $\frac{94^{10}}{2 \cdot 10^9}\,s = 2{,}7 \cdot 10^{10}\,s$

$\approx \underline{854\ a}$

⇒ Sicher gegen Brute-Force-Angriffe.

3.) a)



5

„Nebenläufige Prozesse können zeitgleich ablaufen. Greifen sie dabei auf gemeinsame Resourcen (hier Datenbankserver) zu, kann es zu Inkonsistenzen kommen.
Ein kritischer Abschnitt ist ein Bereich im Programm, der zu Inkonsistenzen führen kann (hier: anfrage und kaufen).

b) Zur Absicherung kritischer Bereiche können Monitore verwendet werden. Sie erlauben jeweils nur einem Prozess den Zugriff auf den Bereich.

2