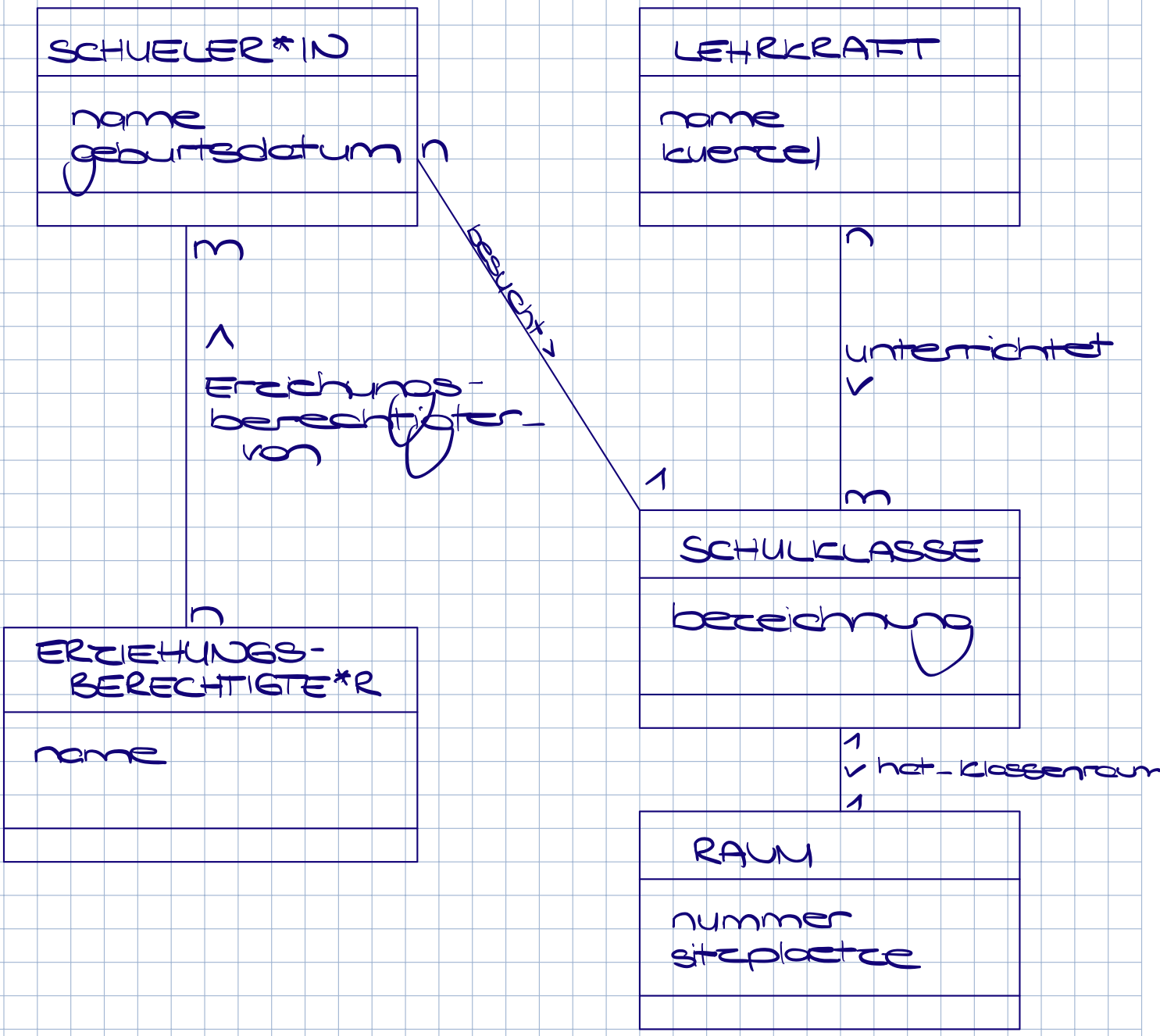
INF1. MODELLIERUNG UND PROGRAMMIERUNG

II.

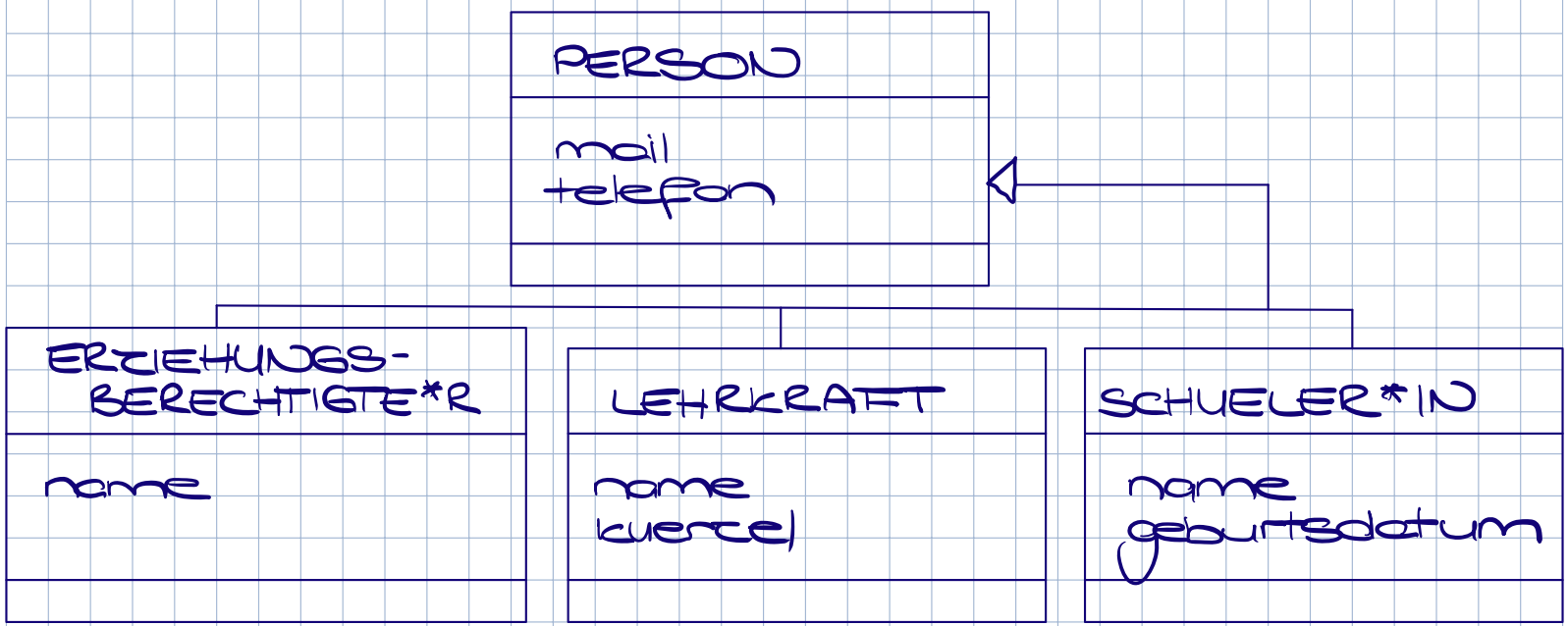
Goepmann

① a) gp



b) 4P

z.B. kann eine Oberklasse PERSON eingefügt werden, falls man neben den bisherigen Daten auch Kontaktdaten speichern möchte. Unterklassen von PERON wären dann ERZIEHUNGSBE-RECHTIGTE*R, LEHRKRAFT und SCHUELER*IN



(Alternativ: Raum mit Unterklassen Fachraum, Klassenzimmer, Büro)

(2)

a) 5P

- Die Anzahl der Wahlkurse ist nicht fixiert. Mit der Listenstruktur kann man effizienter mit neuen Eingaben / Löschungen umgehen.
- Das Abschlussobjekt hat den Vorteil, dass man in die Implementierungen von rekursiven Methoden keine weitere Fallunterscheidung einbauen muss.
- Die Trennung von Struktur und Daten wurde umgesetzt, da der Knoten nur eine Referenz auf die Daten (WAHLKURS) enthält.

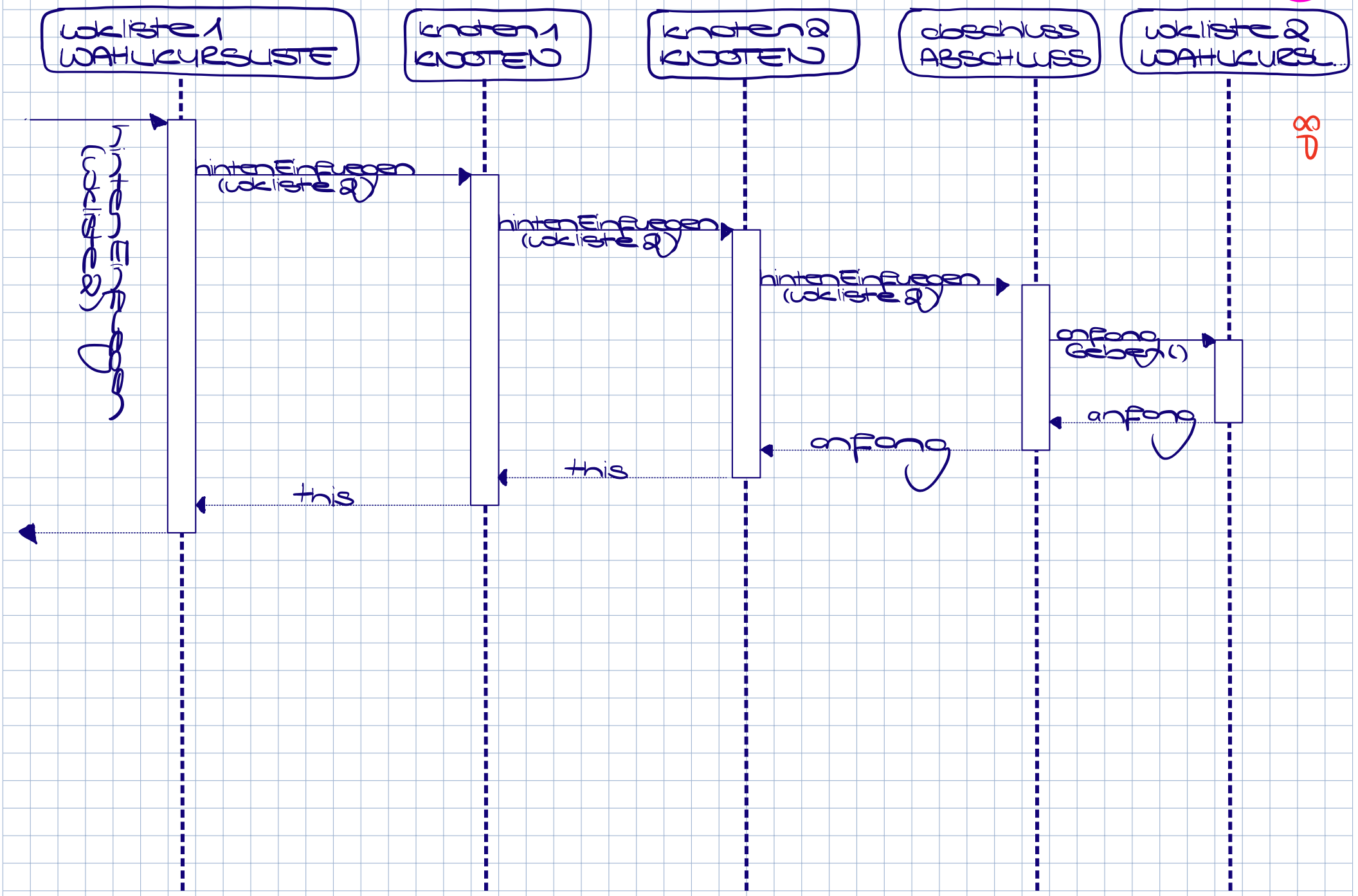
b) 16P

```
class WAHLKURSLISTE {

    private LISTENELEMENT anfang;

    int anzahlKurseGeben (String fachgebiet) {
        return anfang.kurseZaehlen(fachgebiet);
    }

    void zuKleineKurseLöschen (int minTeil-
                               nehmerzahl) {
        anfang = anfang.kurseloeschen(minTeil-
                                      nehmerzahl);
    }
}
```

```
abstract class LISTENELEMENT {
    abstract int kurseZaehlen(String fachgebiet);
    abstract LISTENELEMENT kurseLoeschen
                (int minTeilnehmerzahl);
}

class KNOTEN extends LISTENELEMENT{
    private LISTENELEMENT nachfolger;
    private WAHLKURS inhalt;
    int kurseZaehlen (String fachgebiet){
        int kurszahl = nachfolger.kurseZaehlen
                (fachgebiet);
        if (inhalt.gibFachgebiet.equals (fachgebiet){
            kurszahl ++ ;
        }
        return kurszahl;
    }
    LISTENELEMENT kurseLoeschen(int
                minTeilnehmerzahl){
        nachfolger= nachfolger.kurseLoeschen
                (minTeilnehmerzahl);
        if(inhalt.teilnehmerzahlGeben() <
                minTeilnehmerzahl){
            return nachfolger;
        }
        else {
            return this;
        }
    }
}
```

```
class ABSCHLUSS extends LISTENELEMENT{
    int kursePruefen(String fachgebiet){
        return 0;
    }
    LISTENELEMENT kurseLoeschen(int
                    minTeilnehmerzahl){
        return this;
    }
}
```

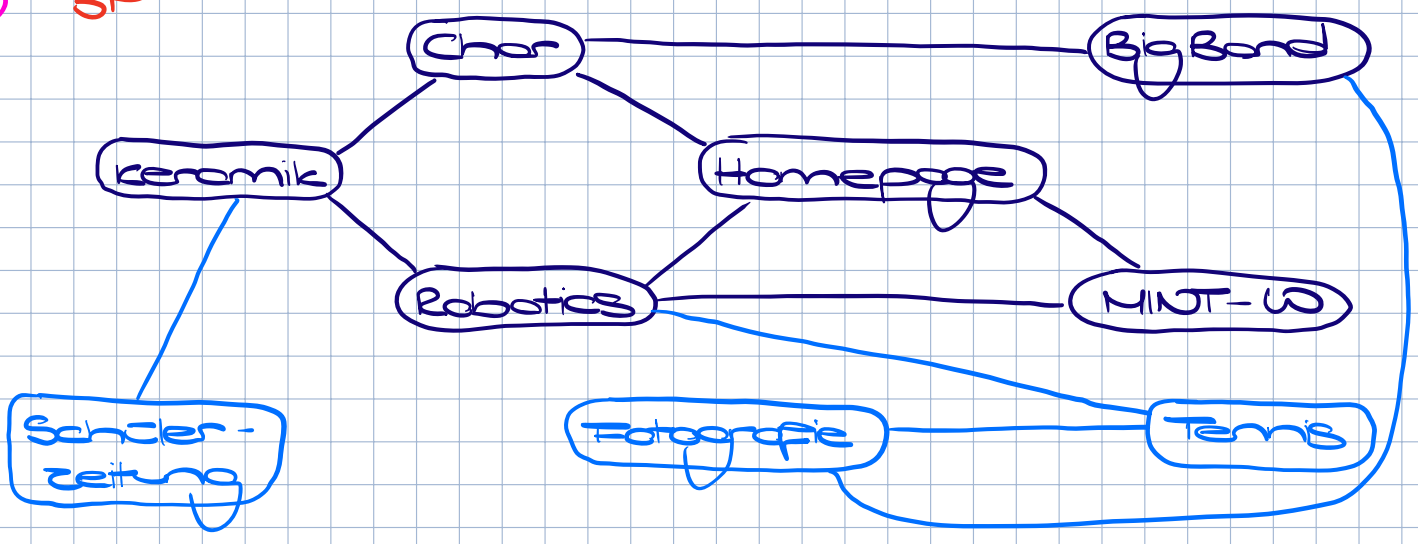
c)


wkliste1
WAHLKURSLISTE
knoten1
KNOTEN
knoten2
KNOTEN
abschluss
ABSCHLUSS
wkliste2
WAHLKURSL...
hintenEinfuegen (wkliste2)
hintenEinfuegen (wkliste2)
hintenEinfuegen (wkliste2)
anfang geben()
anfang
anfang
this
this

③ a) 3P

	Chor	Keramik	Robotics	Homepage	Big Band	MINT-W
Chor		x		x	x	
Keramik	x		x			
Robotics		x		x		x
Homepage	x		x			x
Big Band	x					
MINT-W			x	x		

b) 3P



c) 2P

Kategorie	Kurs
1	MINT-Wettbewerbe
2	Homepage, Robotics
3	Chor, Keramik
4	Big Band

d) GP

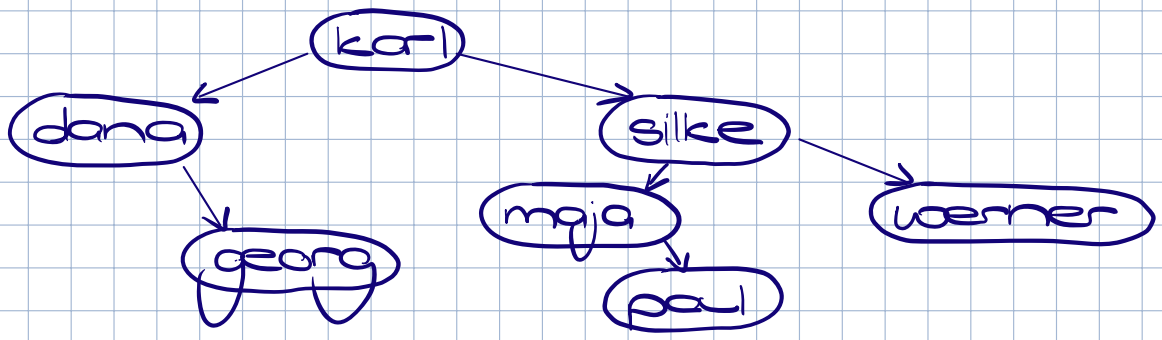
```
wahlkurseKategorisieren(kursname)
    kategorisieren( indexGeben(kursname),1)

kategorisieren( kIndex , kKategorie)
    kursliste[kIndex].kategorieSetzen(kKategorie)
    Für alle KURSE n in kursliste:
        wenn  matrix[indexGeben(n.gibName())]
                [kIndex] == 1 :
            wenn (n.gibKategorie > kKategorie+1
                  oder n.gibKategorie == 0 ):
                kategorisieren( indexGeben(
                    n.gibName()), kKategorie+1)
```

4)

6P a) Inorder: Ergibt lexikographisch sortierte Liste.

Hier: Karl nach Silke ↯

Preorder: Karl kann nicht als letzter in der Liste vorkommen.



3P b) Maja - Georg - Silke - Dana - Paul - Werner

4P c) Maximal: 2000
wenn Baum zu Liste entartet

Minimal: $\lceil \log_2 2000 \rceil \approx \lceil 10{,}97 \rceil = \underline{\underline{11}}$
↑ aufgerundet

wenn Baum vollständig befüllt.

d) 8P

In Klasse BAUM:

```
void emailsAusgeben(String von, String bis){
    wurzel.emailsAusgeben(von, bis);
}
```

In Klasse BAUMELEMENT:

```
abstract void emailsAusgeben(String von,
                             String bis);
```

In der Klasse KNOTEN

```
void emailsAusgeben(String von, String bis){
    if (inhalt.istGroesserAls(bis)){
        linkerNachfolger.emailsAusgeben(von, bis);
    } else if (inhalt.istKleinerAls(von)){
        rechterNachfolger.emailsAusgeben(von, bis);
    } else {
        linkerNachfolger.emailsAusgeben(von, bis);
        inhalt.ausgeben();
        rechterNachfolger.emailsAusgeben(von, bis);
    }
}
```

In der Klasse ABSCHLUSS:

void emailsAusgeben(String von, String bis){
}

Alternativ: Gesamten Baum ablaufen und für jede Mail prüfen ob diese ausgegeben werden soll.