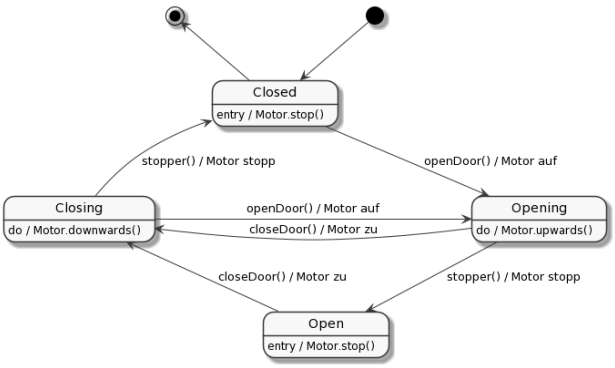
Softwaretechnologie, Übung 13

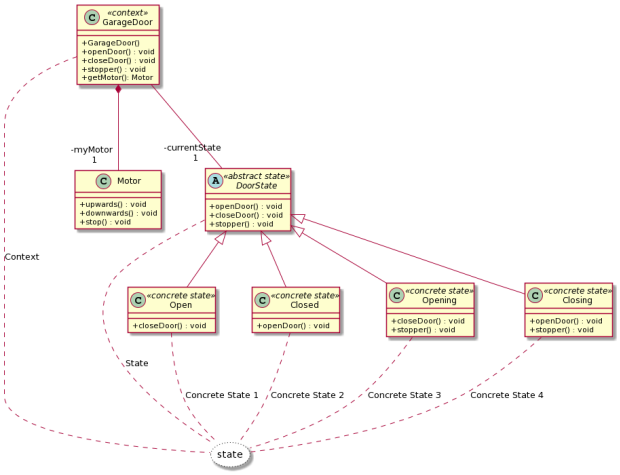
Henry Haustein

Aufgabe 1

(a) (Analyse-)Verhaltenszustandsmaschine



(b) Entwurfs-Klassendiagramm mit State-Entwirfsmuster



(c) Datei `GarageDoor.java`

```java
public class GarageDoor {
  private DoorState currentState;
  private Motor myMotor;

  public GarageDoor() {
```

```
6       this.currentState = new Closed();
7       this.myMotor = new Motor();
8     }
9
10    public void openDoor() {
11      this.currentState.openDoor();
12    }
13
14    public void stopper() {
15      this.currentState.stopper();
16    }
17
18    public void closeDoor() {
19      this.currentState.closeDoor();
20    }
21
22    public Motor getMotor() {
23      return this.myMotor;
24    }
25
26    public void setState(DoorState newState) {
27      if(newState == null){
28        throw new NullPointerException();
29      }
30      this.currentState = newState;
31    }
32
33
34    abstract class DoorState {
35      public void openDoor() {
36        throw new IllegalStateException();
37      }
38
39      public void closeDoor() {
40        throw new IllegalStateException();
41      }
42
43      public void stopper() {
44        throw new IllegalStateException();
45      }
46    }
47
48    class Closed extends DoorState {
49      @Override
50      public void openDoor() {
51        myMotor.upwards();
52        setState(new Opening());
53      }
54    }
55
56    class Opening extends DoorState {
```

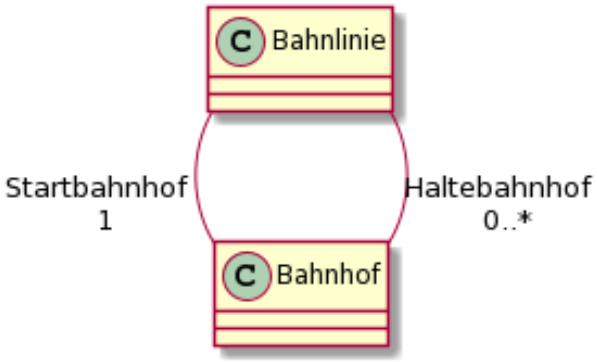
```
57      @Override
58      public void closeDoor() {
59        myMotor.downwards();
60        setState(new Closing());
61      }
62
63      @Override
64      public void stopper() {
65        myMotor.stop();
66        setState(new Open());
67      }
68    }
69
70    class Open extends DoorState {
71      @Override
72      public void closeDoor() {
73        myMotor.upwards();
74        setState(new Closing());
75      }
76    }
77
78    class Closing extends DoorState {
79      @Override
80      public void openDoor() {
81        myMotor.upwards();
82        setState(new Opening());
83      }
84
85    @Override
86      public void stopper() {
87        myMotor.stop();
88        setState(new Closed());
89      }
90    }
91  }
```

Datei `Motor.java`

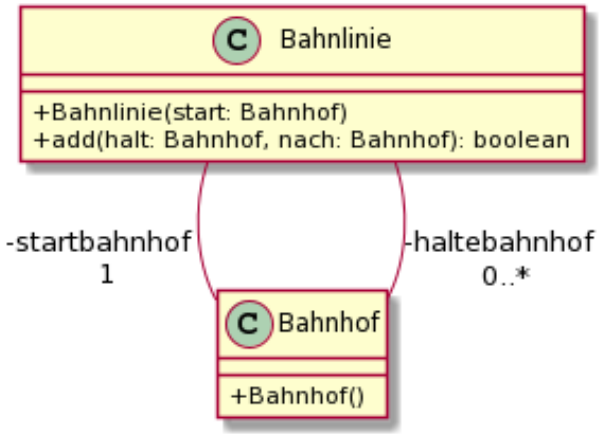
```
1  public class Motor {
2    public void upwards() {
3      System.out.println("Door goes up!");
4    }
5
6    public void downwards() {
7      System.out.println("Door goes down!");
8    }
9
10   public void stop() {
11     System.out.println("Moving completed!");
12   }
13 }
```

Aufgabe 2

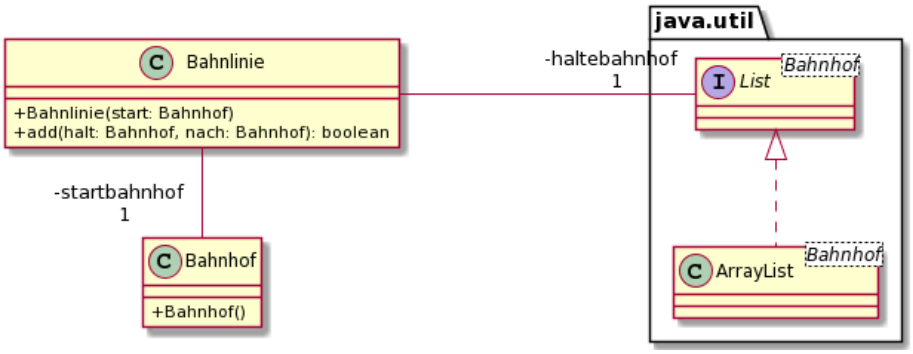
(a) aUML:



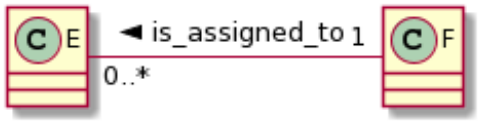
dUML (Sichtbarkeiten, Java-Datentypen, gerichtete Assoziationen, Konstruktoren, getter, setter, Parameter, Rückgabetypen von Methoden):



jUML (Unterscheidung zw. Klassen und Interfaces, konkrete Java-Implementierungen (hier java.util.List)):



(b) aUML:

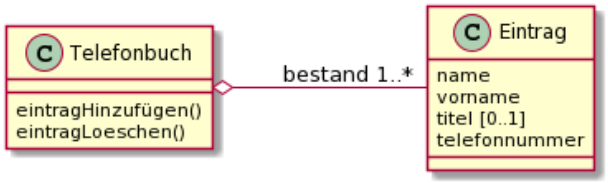


dUML:

Quelltext:

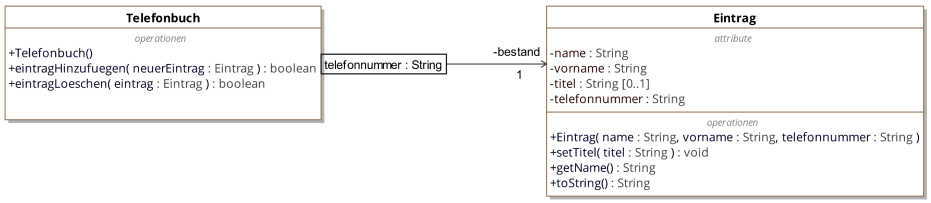
```
1  class E {
2    private F myF;
3
4    public E (F myF) {
5      if(F == null){
6        throw new NullPointerException("Gib F du Null!");
7      }
8      myF = myF;
9    }
10 }
```

(c) aUML:



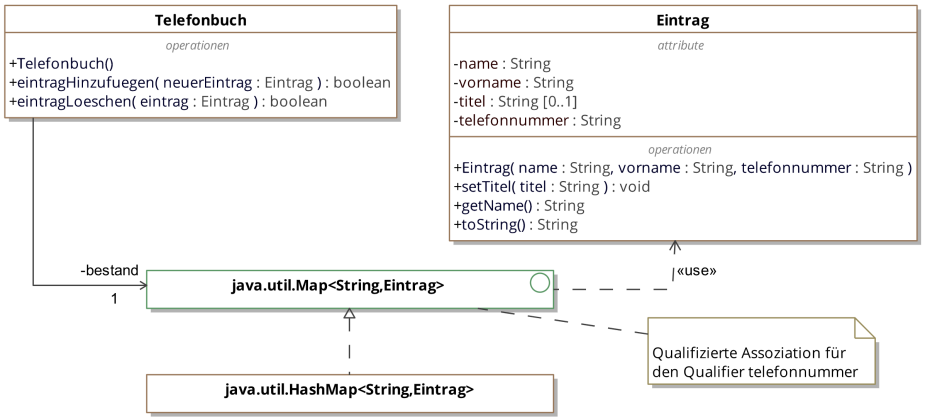
dUML:

- Qualifizierte gerichtete Assoziation mit dem Qualifier Telefonnummer → Multiplizität ändert sich auf 1 (Telefonnummer soll Schlüssel sein).
- Qualifizierte gerichtete Assoziation mit dem Qualifier Name → Multiplizität bleibt bei 1...*, weil es verschiedene Personen mit gleichem Namen aber unterschiedlicher Telefonnummer gibt.

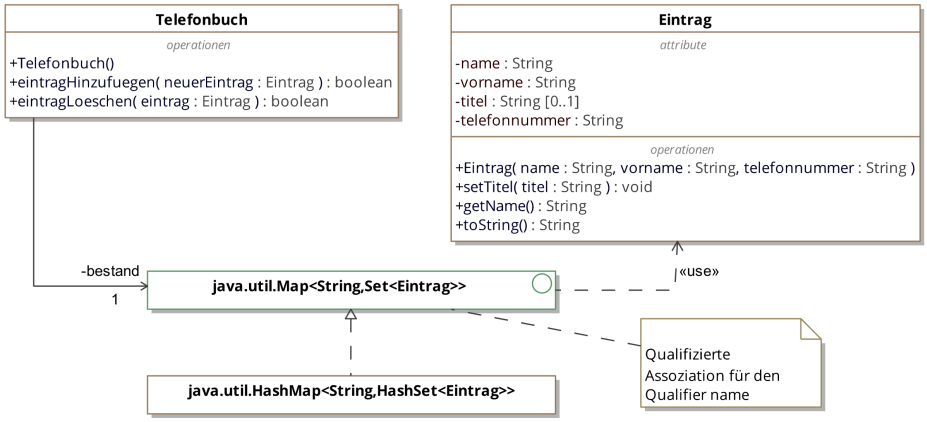


jUML (qualifizierte Assoziation als `Map<key, value>` implementieren, key ist der Qualifier und value sind die zugeordneten Objekte):

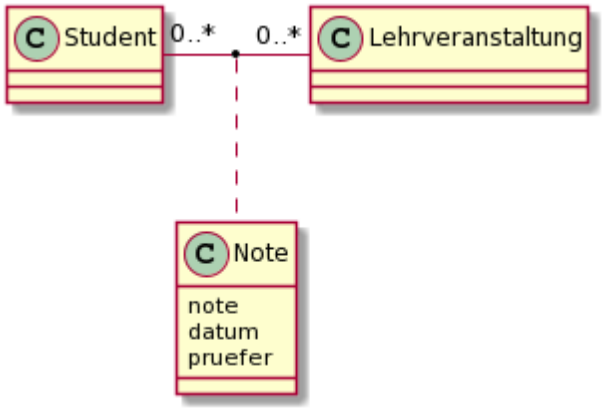
- Zu jeder Telefonnummer (Qualifier) genau ein Eintrag zugeordnet.



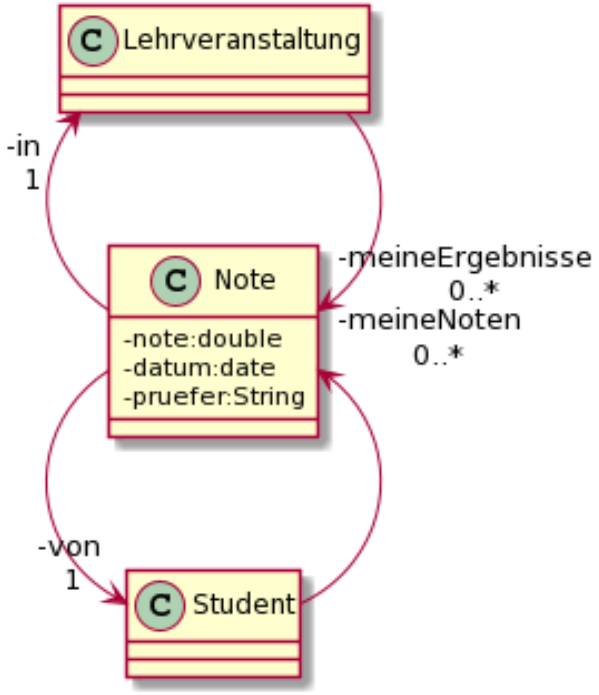
- Zu jedem Namen gibt es beliebig viele Einträge (z.B. Set mit Einträgen)



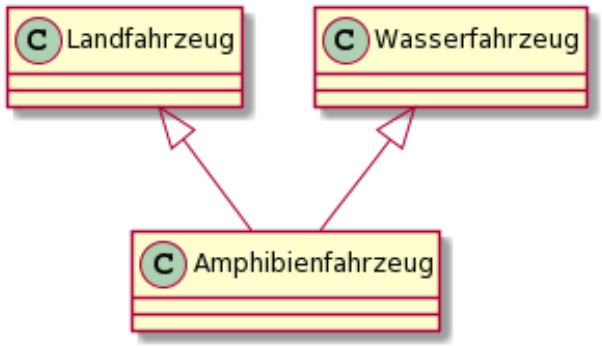
(d) aUML:



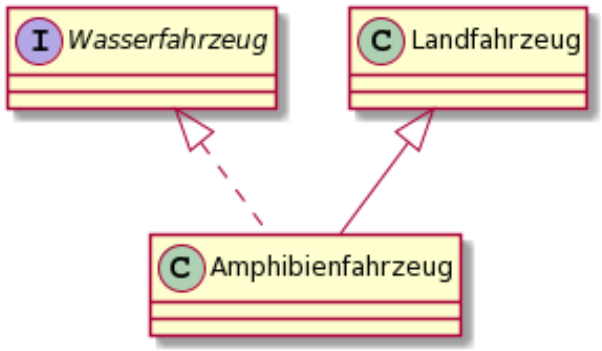
dUML:



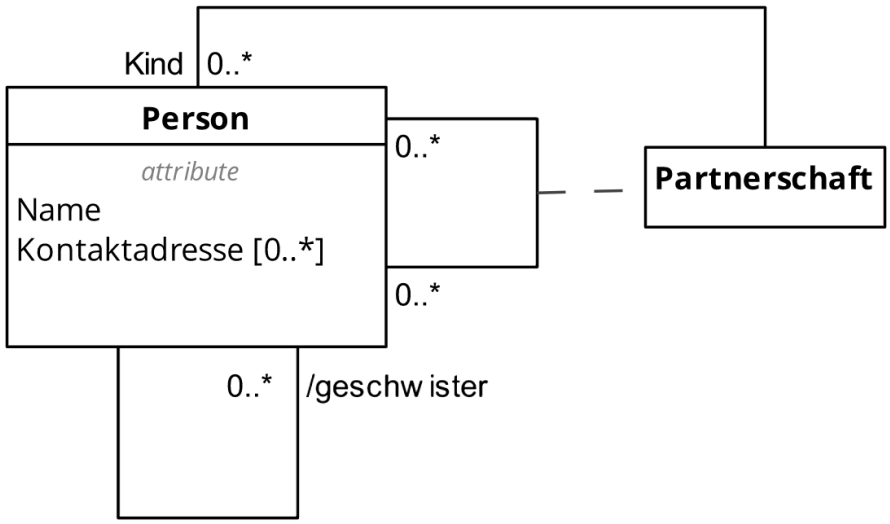
(e) aUML (Mehrfachvererbung):



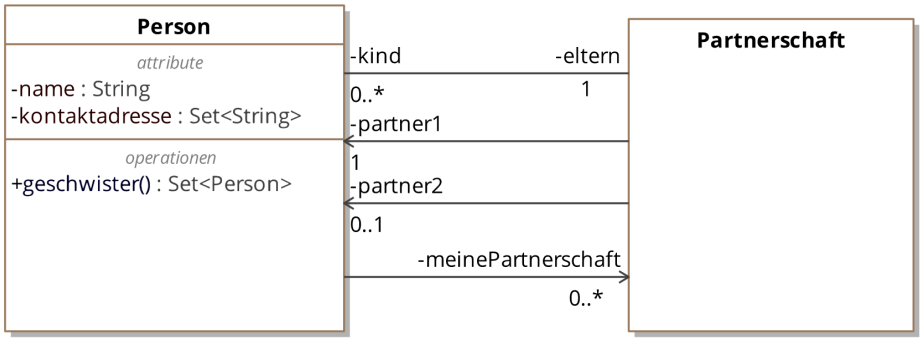
dUML (mit Interface):



(f) aUML:


Kind
0..*
Person
attribute
Name
Kontaktadresse [0..*]
0..*
Partnerschaft
0..*
0..*
/geschw ister

dUML:


Person
attribute
-name : String
-kontaktadresse : Set<String>
operationen
+geschwister() : Set<Person>
-kind
0..*
-partner1
1
-partner2
0..1
-meinePartnerschaft
0..*
-eltern
1
Partnerschaft