Prof. Dr. Thomas Breuel
Dipl.-Inf. Patrick Michel (AG ST)
Dipl.-Ing. Ludwig Schmidt-Hackenberg

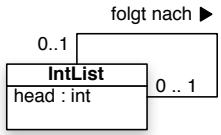
TU Kaiserslautern
Fachbereich Informatik
AG Bildverstehen und Mustererkennung

Lösungshinweise/-vorschläge zum Übungsblatt 12: Software-Entwicklung 1 (WS 2010/11)

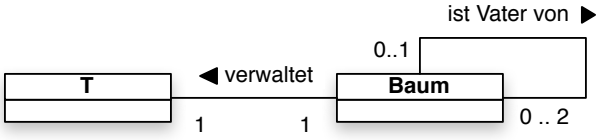
Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Objektorientierte Modellierung (Präsenzaufgabe)

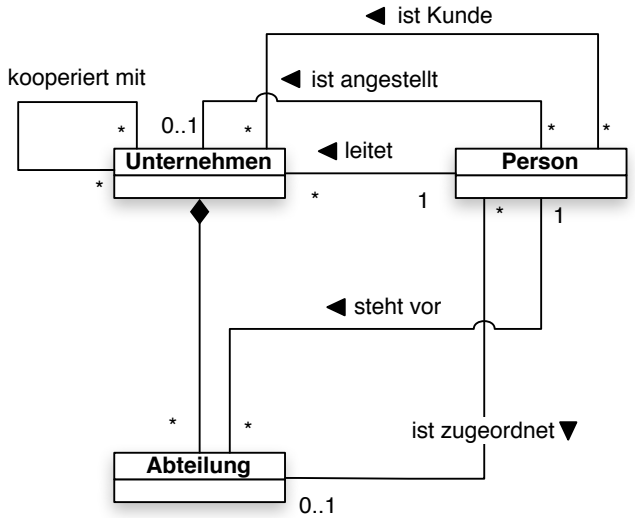
a) Entwerfen Sie ein Klassendiagramm für einfach verkettete Listen vergleichbar zu `IntList` aus der Vorlesung. Tragen Sie in dieses Diagramm auch Multiplizitäten und Namen der Beziehungen ein.



b) Entwerfen Sie ein Klassendiagramm für das Geflecht eines fast vollständigen binären Suchbaums mit Objekten des Typ `T` an den Elementen des Baums. Tragen Sie in dieses Diagramm auch Multiplizitäten und Namen der Beziehungen ein.



c) Entwerfen Sie ein Klassendiagramm mit Multiplizitäten und Beziehungen für folgendes Beispiel: `Personen` in einem `Unternehmen` können angestellt sein, es leiten, einer `Abteilung` vorstehen, einer Abteilung zugeordnet sein oder Kunde des Unternehmens sein. Ein Unternehmen kann auch mit einem Unternehmen kooperieren.



Aufgabe 2 Objektorientierte Modellierung (Einreichaufgabe)

Ihnen liegen die folgenden Informationen über Transportmittel vor:

a) Erstellen Sie mit Hilfe dieser Informationen ein Klassendiagramm, das die verschiedenen Transportmittel einschließlich ihrer Attribute und Methoden strukturiert darstellt.

b) Spezialisieren Sie die Klassen Wasser- und Lufttransportmittel, indem Sie mindestens fünf weitere Klassen, wie z.B. Motorboot und Flugzeug in das Klassendiagramm aus Aufgabenteil a) einfügen! Erweitern Sie die neuen Klassen um geeignete Attribute und Methoden.

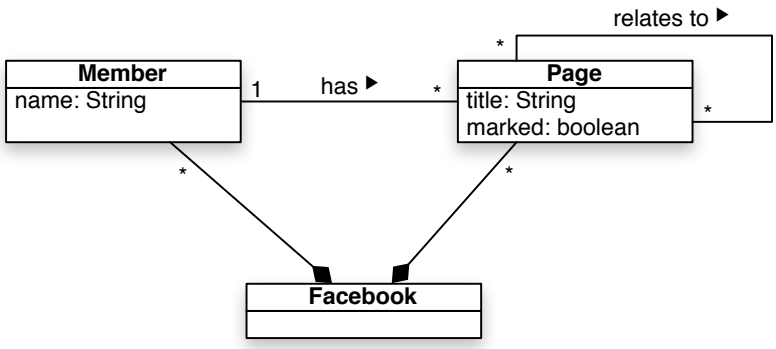
Hinweis: In der Vorlesung wurde die genaue Syntax von Methoden in UML-Diagrammen nicht im Detail behandelt, Sie werden diese in Folgeveranstaltungen genauer kennenlernen. Für diese Aufgabe ist die genaue Darstellung der Methoden also nicht als zentral anzusehen. Wichtig ist hier die Darstellung der Klassen und vor allem deren Beziehungen.

c) Entwerfen Sie ein Klassendiagramm mit Multiplizitäten und Beziehungen für das Beispiel Mini-Facebook aus der Vorlesung.

Der relevante Beschreibung und der Code für diese Aufgabe ist:

> We define a very simple "Facebook" application. There are members and there are pages. Each member has a name and a number of pages. Pages have a title and a list of related pages.

```
class Member {
  String name;
  Page[] pages;
}

class Page {
  String title;
  Page[] related;
  boolean marked;
}

class Facebook {
  Member [] members;
  Page [] pages;
}
```



Man hat hier natürlich ein paar Designfreiheiten, vor allem was Aggregation vs. Assoziation angeht. Man sollte allerdings schon die Arrays in solche umformen und nicht einfach nur als Attribute belassen. Die Multiplizitäten ergeben sich teils aus der Tatsache, dass Arrays verwendet worden sind, teilweise einfach aus der Anwendungsdomäne.

Aufgabe 3 Objekte, Kapselung, Fehlerbehandlung (Einreichaufgabe)

a) Entwickeln Sie eine Klasse `UncleKracker`.

Zum Knacken soll die Methode `crack` die öffentliche Schnittstelle von `Secret` verwenden, also im wesentlichen die Methode `unlock`. Fangen Sie alle möglichen Ausnahmen entsprechend ab und reagieren Sie sinnvoll darauf, so dass Sie nach einer endlichen Anzahl an Iterationen das Schloss geknackt haben.

Die Lösung ist schon etwas optimiert, d.h. sie kann insbesondere Schlösser jeder größe lösen ohne ewig zu laufen. Zur Strukturierung wurden auch noch mehr Methoden eingefügt und es gibt noch eine Ausgabefunktionalität für die aktuelle Kombination, was auch nett zum Debuggen ist.

Die Lösung benutzt die Anzahl der sonst richtigen Digits noch gar nicht, damit kann man noch optimieren. Die Komplexität liegt wohl bei $O(n * m)$, wobei n und m die Anzahl der Stellen und deren Maximalgröße sind.

```
class UncleKracker {
  private Secret s;
  private int locks = -1, keys = -1;
  private int[] combination;

  UncleKracker(Secret s) {
    this.s = s;
  }

  int locks() { return locks; }
  int keys() { return keys; }

  void crackInit() {
    locks = 16;
    combination = new int[locks];

    while (true) {
      try {
        s.unlock(combination);
        break;
      } catch (TooFewDigitsException e) {
        locks++;
        combination = new int[locks];
      } catch (TooManyDigitsException e) {
        locks--;
        combination = new int[locks];
      } catch (InvalidDigitException e) {
        keys = combination[0]--;
        break;
      } catch (WrongCombinationException e) {
        combination[0]++;
      }
    }
  }

  int[] crack() {
    if (keys < 0) crackInit();
    int blank = -1;

    for (int i = 0; i < keys; i++) {
      for (int j = 0; j < locks; j++) combination[j] = i;
      try {
        s.unlock(combination);
        break;
      } catch (WrongCombinationException e) {
        if (e.correctPlace() == 0) {
          blank = i;
          break;
        }
      }
    }
```

```
    if (blank == -1) return null;
    int[] comb = new int[locks];
    for (int i = 0; i < locks; i++) comb[i] = blank;

    for (int i = 0; i < keys; i++) {
      for (int j = 0; j < locks; j++) {
        comb[j] = i;
        try {
          s.unlock(comb);
          break;
        } catch (WrongCombinationException e) {
          if (e.correctPlace() == 1) combination[j] = i;
        }
        comb[j] = blank;
      }
    }

    return combination.clone();
  }

  void print() {
    boolean first = true;
    for (int i = 0; i < locks; i++) {
      if (first) {
        first = false;
        System.out.print("[");
      } else System.out.print(", ");
      System.out.print(combination[i]);
    }
    System.out.println("]");
  }
}
```

b) Testen Sie Ihre Implementierung mit verschiedenen Schlössern. Da die Schlösser sehr groß werden können, sollten Sie Ihre Tests mit kleinen, einfachen Schlössern durchführen. Dazu können Sie die folgenden Seeds benutzen, die alle ein Schloss mit vier Stellen und sechs Möglichkeiten pro Stelle lieferen:[1]

```
7422    111972  209338  265094  306944  499357  551273  779423  921639  978011
20025   146574  213305  266831  368652  509463  580264  783807  934502
82725   194839  222197  277350  381932  524567  665598  812081  938630
103080  204798  248655  294337  403542  538126  692080  825105  965144
```

c) *(freiwillige Zusatzaufgabe)* Die Klasse `Secret` protokolliert die Anzahl der Zugriffe auf die Methode `unlock`. Optimieren Sie Ihren Algorithmus zum Knacken von Schlössern, um im Mittel über viele zufällige Beispiele eine möglichst niedrige Zahl zu erreichen!

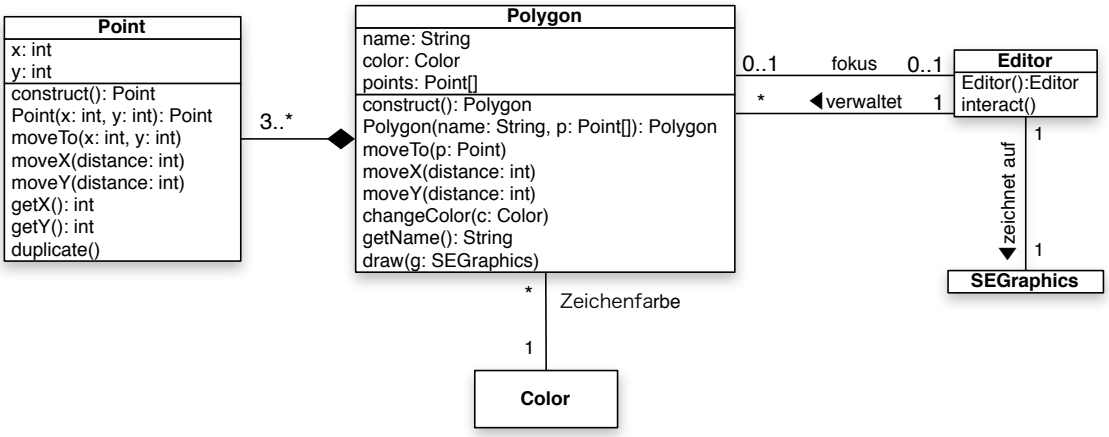
Keine weiteren Hinweise abgesehen von den Optimierungen und Vorschlägen in der Lösung zu a).

Hier könnte Ihre Lösung stehen!

[1] Sollten diese Seeds auf Ihrem System nicht diesen Effekt haben, müssen Sie selbst ermittlen hinter welchen Seeds sich kleine Schlösser verbergen.

Aufgabe 4 Grafikeditor (Präsenzaufgabe)

Lesen Sie die Teilaufgaben von Aufgabe 5 und entwerfen Sie ein Klassendiagramm mit allen Klassen, Methoden, Attributen und Beziehungen.



Aufgabe 5 Grafikeditor (Einreichaufgabe)

In dieser Aufgabe geht es darum, einen Editor für graphisch darstellbare Komponenten zu implementieren.

a) Entwerfen Sie eine Klasse `Point`. Der Konstruktor soll die Koordinaten des Punktes als Parameter nehmen. Schreiben Sie eine statische Methode `construct`, die vom Benutzer die Koordinaten des Punkts erfragt, ein Objekt vom Typ `Point` erzeugt und zurückgibt. Sie werden auch Methoden brauchen, die einen Punkt verschieben.

```
public class Point {
  private int x, y; //Koordinaten des Punktes

  // Get Methoden
  public int getX() { return x; };
  public int getY() { return y; };

  static public Point construct() {
    int x, y;
    IO.print("x-Koordinate: ");
    x = IO.readInt();
    IO.print("y-Koordinate: ");
    y = IO.readInt();
    return new Point(x, y);
  }

  protected Point(int x, int y) { this.x = x; this.y = y; }

  // move the point to new coordinates
  public void moveTo(int newX,int newY) { x = newX; y = newY; }

  // move relative by distance in x direction
  public void moveX(int distance) { x = x + distance; }

  // move relative by distance in y direction
  public void moveY(int distance) { y = y + distance; }
}
```

Es reicht für die Aufgabe relative oder absolute Verschiebungen zu unterstützen.

b) Entwerfen Sie eine Klasse `Polygon`, mit der Sie beliebige Polygone verwalten können. Um Polygone später identifizieren zu können, soll jedes Polygon einen Namen haben.

```
import java.awt.Color;

public class Polygon {
  private String name;
  protected Point[] points;
  protected Color color = Color.black;

  public void changeColor(Color c) { color = c;  }
  public String getName() { return name; }

  public static Polygon construct() {
    IO.println("Name des neuen Polygons:");
    String name = IO.readString();

    int no = 0;
    while (no < 3) {
      IO.print("Anzahl der Ecken (min 3): ");
      no = IO.readInt();
    }
    Point[] points = new Point[no]; //Array in der richtigen Groesse anlegen
    for (int i = 0; i<points.length; i++) {
      IO.println("Ecke " + (i+1) + ":");
      points[i] = Point.construct();
    }

    return new Polygon(name, points);
  }

  protected Polygon(String name, Point[] p) {
    this.name = name; this.points = p;
  }

  // bewegt den ersten Punkt an die angegebene Koordinate,
  // der Rest des Polygons wird entsprechend mitverschoben
  public void moveTo(Point p) {
    int distX = p.getX() - points[0].getX();
    int distY = p.getY() - points[0].getY();
    moveX(distX);
    moveY(distY);
  }

  // Relativ um distance verschieben in x-Richtung
  public void moveX(int distance) {
    for (int i = 0; i < points.length; i++) {
      points[i].moveX(distance);
    }
  }

  // Relativ um distance verschieben in y-Richtung
  public void moveY(int distance) {
    for (int i = 0; i < points.length; i++) {
      points[i].moveY(distance);
    }
  }

  public void draw(SEGraphics g) {
    int l = points.length;
    for (int i = 0; i < points.length; i++) {
      g.drawLine(points[i].getX(), points[i].getY(),
                 points[(i+1) % l].getX(), points[(i+1)%l].getY(), color);
    }
  }
}
```

c) Entwerfen Sie eine Klasse `Editor`, die eine Menge von Polygonen und eine Zeichenfläche verwaltet.

d) Schreiben Sie ein Hauptprogramm, das einen neuen Editor erzeugt und die Interaktionsmethode aufruft.

```
import java.util.LinkedList;
import java.util.Iterator;
import java.awt.Color;

public class Editor {
  private SEGraphics g;
  private LinkedList<Polygon> figures;
  private Polygon focus = null;

  public Editor() {
    g = new SEGraphics(800,700);
    figures = new LinkedList<Polygon>();
  }

  public void interact() {
    while (true) {
      int choice = menu();
      switch (choice) {
        case 1:   createFigure(); break;
        case 2:   changeFocus(); break;
        case 3:   moveTo(); break;
        case 4:   move(); break;
        case 5:   changeColor();break;
      }
      draw();
    }
  }

  private int menu() {
    IO.println("Was wollen Sie machen: ");
    IO.println("1) Neue Figur anlegen");
    IO.println("2) Fokus aendern");
    IO.println("3) Verschieben (absolut)");
    IO.println("4) Verschieben (relativ)");
    IO.println("5) Farbe aendern");
    int choice = 0;
    while (!((choice >= 1 && choice <=5))) {
      IO.print("\nIhre Wahl: ");
      choice = IO.readInt();
    }
    return choice;
  }

  private void changeFocus() {
    if (figures.isEmpty()) {
      IO.println("Mindestens ein Polygon vorher anlegen.");
      return;
    }
    IO.println("Name des Polygons, dass jetzt den Fokus haben soll:");
    String name = IO.readString();
    boolean done = false;
    Polygon next;
    Iterator<Polygon> iter = figures.iterator();
    while (iter.hasNext() && !done) {
      next = iter.next();
      if (next.getName().equals(name)) {
        focus = next;
        done = true;
      }
    }
    if (!done) IO.println("Polygon nicht gefunden. Fokus wurde nicht veraendert");
  }
```

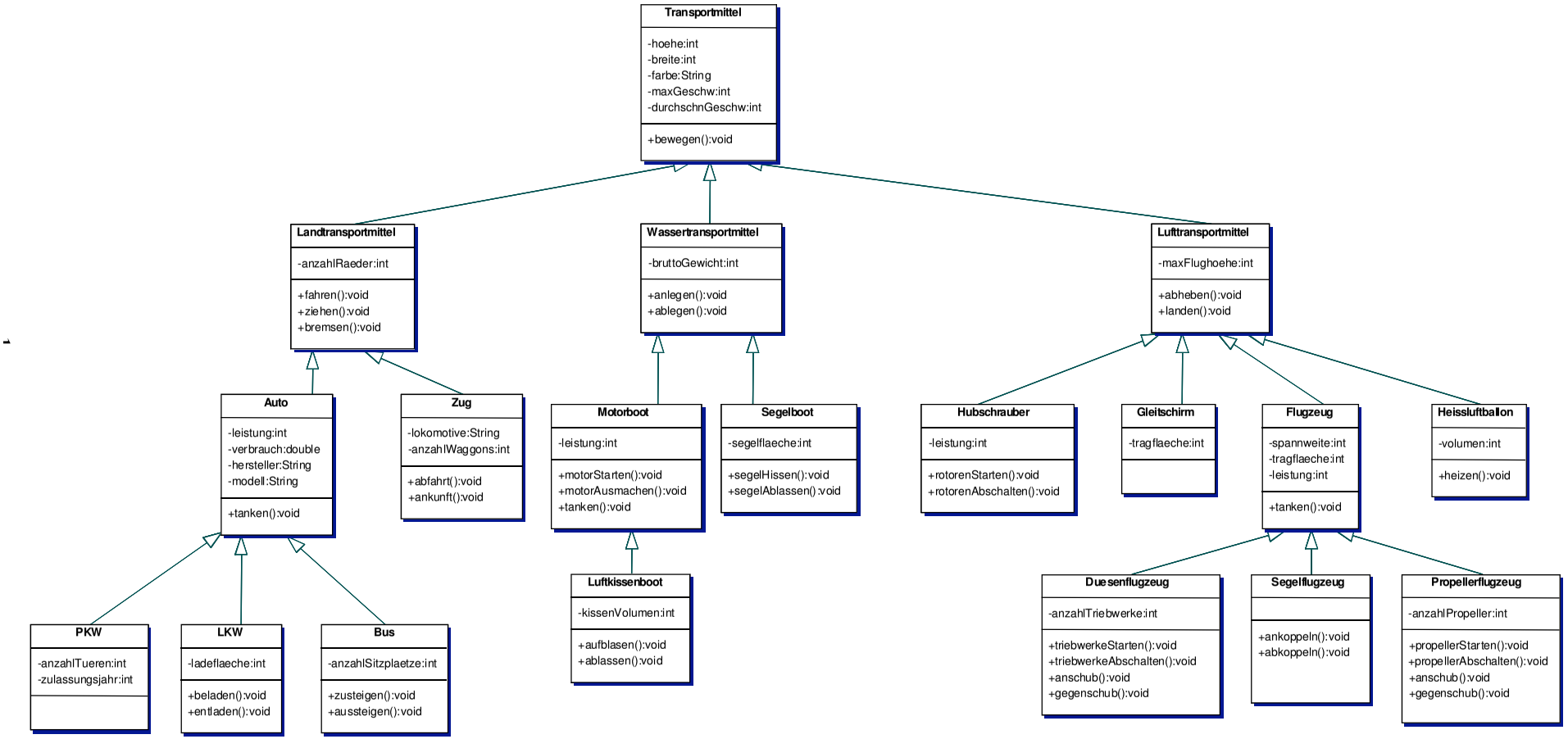
```
  private void createFigure() {
    Polygon f = Polygon.construct();
    figures.add(f);
    focus = f;
  }

  private void moveTo() {
    if (figures.isEmpty()) {
      IO.println("Mindestens ein Polygon vorher anlegen.");
      return;
    }
    IO.println("Neuen Startpunkt eingeben: ");
    Point start = Point.construct();
    focus.moveTo(start);
  }

  private void move() {
    if (figures.isEmpty()) {
      IO.println("Mindestens ein Polygon vorher anlegen.");
      return;
    }
    IO.print("Distanz in X-Richtung: ");
    int distX = IO.readInt();
    IO.print("\nDistanz in Y-Richtung: ");
    int distY = IO.readInt();
    focus.moveX(distX);
    focus.moveY(distY);
  }

  private void changeColor() {
    if (figures.isEmpty()) {
      IO.println("Mindestens ein Polygon vorher anlegen.");
      return;
    }
    int choice = 0;
    while (choice < 1 || choice > 3) {
      IO.println("Neue Farbe waehlen:");
      IO.println("1) Rot");
      IO.println("2) Blau");
      IO.println("3) Schwarz");
      IO.print("Neue Farbe:");
      choice = IO.readInt();
    }
    switch (choice) {
      case 1: focus.changeColor(Color.red); break;
      case 2: focus.changeColor(Color.blue); break;
      case 3: focus.changeColor(Color.black); break;
      default: focus.changeColor(Color.black);
    }
  }

  private void draw() {
    g.clear();
    for(Polygon p:figures) {
      p.draw(g);
    }
  }

  public static void main(String[] args) {
    Editor e = new Editor();
    e.interact();
  }
}
```

Um die Polygone zu verwalten dürfen natürlich auch eigene Listenimplementierungen verwendet werden.

Aufgabe 6 Basiswissen zur Vorlesung

Kreuzen Sie an, ob folgende Aussagen wahr oder falsch sind. Bereiten Sie diese Aufgabe bis zu Ihrer nächsten Übungsstunde vor, so dass Sie bei Unklarheiten nachfragen und die Antworten diskutieren können.

wahr	falsch	
☐	☒	Eine Ordnung ist genau dann noethersch, wenn jede aufsteigende Kette stationär wird. *Es muss jede* absteigende *sein.*
☒	☐	$(\mathbb{N}, <)$ ist keine noethersche Ordnung. *Das gegebene Paar ist keine Ordnung, da $<$ nicht reflexiv ist.*
☒	☐	$(\mathbb{Z}, \leq)$ ist keine noethersche Ordnung. *Es gibt in $\mathbb{Z}$ kein kleinstes Element.*
☐	☒	$(2\mathbb{N}, \leq)$ ist keine noethersche Ordnung. $(2\mathbb{N} = \{2n \mid n \in \mathbb{N}\})$ *Jede Teilmenge einer noethersch geordneten Menge führt wieder zu einer noetherschen Ordnung.*
☒	☐	Einer Programmvariablen können mehrere Speichervariablen zugeordnet sein. *Beispielsweise im Falle rekursiver Methoden.*
☐	☒	Ein vollständiger gerichteter Graph hat genau doppelt so viele Kanten wie ein vollständiger ungerichteter Graph. *Zu einer Schleife gibt es auch im gerichteten Graphen nur genau eine Kante. Ein vollständiger gerichteter Graph hat n^2 Kanten, der ungerichtete allerdings $\frac{n^2}{2} + \frac{n}{2}$ (also etwas mehr als die Hälfte).*
☒	☐	Eine Bedingung, die im Vorzustand einer Prozedur gelten muss, kann nicht unbedingt über den Rumpf der Prozedur als gültig angenommen werden. *Schon nach der ersten Anweisung können Teile der Bedingung wieder verletzt werden.*
☐	☒	Aus der Vorbedingung `true` lässt sich die Nachbedingung `false` für keine Prozedur beweisen. *Mit dem Rumpf `while(true){}` lässt sich mit der Invariante `true` auch `false` beweisen, was nicht schlimm ist, da dieser Zustand im Programm entsprechend nie erreicht wird.*
☐	☒	Ein Hoare-Tupel verbindet eine Vorbedingung P mit einer Nachbedingung Q. *Man spricht von Hoare-*Tripeln*, die eine Vorbedingung P über ein Programmfragment S mit einer Nachbedingung Q verbinden: $\{P\}\ S\ \{Q\}$.*
☒	☐	Kann die Spezifikation einer Prozedur mit Schleife bewiesen werden, gibt es automatisch unendlich viele Schleifeninvarianten, mit denen der Beweis geführt werden kann. *Man kann eine Invariante immer mit Aussagen verstärken, die von der Schleife nicht berührt werden. Damit lassen sich auch insbesondere Aussagen, die vor der Schleife gelten, über die Schleife hinweg transportieren.*


Transportmittel
-hoehe:int
-breite:int
-farbe:String
-maxGeschw:int
-durchschnGeschw:int
+bewegen():void
Landtransportmittel
-anzahlRaeder:int
+fahren():void
+ziehen():void
+bremsen():void
Wassertransportmittel
-bruttoGewicht:int
+anlegen():void
+ablegen():void
Lufttransportmittel
-maxFlughoehe:int
+abheben():void
+landen():void
Auto
-leistung:int
-verbrauch:double
-hersteller:String
-modell:String
+tanken():void
Zug
-lokomotive:String
-anzahlWaggons:int
+abfahrt():void
+ankunft():void
Motorboot
-leistung:int
+motorStarten():void
+motorAusmachen():void
+tanken():void
Segelboot
-segelflaeche:int
+segelHissen():void
+segelAblassen():void
Hubschrauber
-leistung:int
+rotorenStarten():void
+rotorenAbschalten():void
Gleitschirm
-tragflaeche:int
Flugzeug
-spannweite:int
-tragflaeche:int
-leistung:int
+tanken():void
Heissluftballon
-volumen:int
+heizen():void
PKW
-anzahlTueren:int
-zulassungsjahr:int
LKW
-ladeflaeche:int
+beladen():void
+entladen():void
Bus
-anzahlSitzplaetze:int
+zusteigen():void
+aussteigen():void
Luftkissenboot
-kissenVolumen:int
+aufblasen():void
+ablassen():void
Duesenflugzeug
-anzahlTriebwerke:int
+triebwerkeStarten():void
+triebwerkeAbschalten():void
+anschub():void
+gegenschub():void
Segelflugzeug
+ankoppeln():void
+abkoppeln():void
Propellerflugzeug
-anzahlPropeller:int
+propellerStarten():void
+propellerAbschalten():void
+anschub():void
+gegenschub():void