Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
Ausgabe: 01.12.2025
Bearbeiten bis: 08.12.2025

Übungsblatt 9

- Objektorientierte Programmierung 1 -

Aufgabe 1: Klasse Studierende

Ein Student / eine Studentin wird beschrieben durch seinen/ihren Vornamen, Nachnamen und Matrikelnummer. Erzeugen kann man einen Student / eine Studentin, indem alle diese Angaben angegeben werden. Geben Sie eine Klasse Student an, die dieser Beschreibung genügt. Über eine Methode public String toString() soll man eine textuelle Beschreibung zu einem Studenten / einer Studentin bekommen. Testen Sie ihre Klasse mit folgendem Testprogramm:

```
public static void main(String[] args){
  Student willi = new Student("Willi", "Wichtig", 900);
  Student helga = new Student("Helga" ,"Eifrig", 901);

  //hier wird jeweils implizit die toString Methode aufgerufen
  System.out.println(willi);
  System.out.println(helga);
}
```

Die Ausgabe dazu soll sein:

```
Willi Wichtig (900)
Helga Eifrig (901)
```

Ein einzelnes Prüfungsergebnis eines Studenten / einer Studentin wird beschrieben durch den entsprechenden Studenten / Studentin, eine Note (nehmen Sie dazu float, da auch z.B. 1,3 möglich ist) und eine Punktzahl (ganzzahlig). Für den Studenten / die Studentin sehen Sie eine Referenz auf ein Objekt vom Typ Student vor! Realisieren Sie dies in einer Klasse Ergebnis. Sehen Sie eine toString-Methode analog zu oben vor. Testen Sie ihre Klasse mit folgendem Testprogramm:

```
public static void main (String[] args){
  //zwei Studenten erzeugen
  Student willi = new Student("Willi" ,"Wichtig", 900);
  Student helga = new Student ("Helga" , "Eifrig" , 901);
  //zwei Pruefungsergebnisse erzeugen
  Ergebnis e1 = new Ergebnis(willi, 4.0f, 60);
  Ergebnis e2 = new Ergebnis(helga, 1.0f, 118);
  //hier wird jeweils implizit die toString Methode aufgerufen
  System.out.println(e1);
  System.out.println(e2);
}
```

Die Ausgabe dazu soll sein:

```
Willi Wichtig (900) Note=4.0 Punktzahl=60
Helga Eifrig (901) Note=1.0 Punktzahl=118
```

Eine Prüfung wird beschrieben durch einen Prüfungsnamen, das Prüfungsdatum (nehmen Sie einen String dazu) sowie die Creditanzahl (ganzzahlig). Weiterhin gibt es $n \in N$ (inkl. 0) Einzelergebnisse (s.o.), jeweils zu einem Studenten / einer Studentin. Geben Sie eine Klasse Pruefung an. Diese soll einen Konstruktor

haben, mit dem man eine Prüfung noch ohne Ergebnisse anlegen kann. Weiterhin soll es eine Methode void neuesErgebnis(Ergebnis e) geben, die jeweils ein Einzelergebnis zu dieser Prüfung hinzufügt. Sehen Sie eine toString-Methode analog zu oben vor. Testen Sie ihre Klasse mit folgendem Testprogramm:

```
public static void main (String[] args){
  //zwei Studenten erzeugen
  Student willi = new Student("Willi" ,"Wichtig", 900);
  Student helga = new Student ("Helga" , "Eifrig" , 901);

  //zwei Pruefungsergebnisse erzeugen
  Ergebnis e1 = new Ergebnis(willi, 4.0f, 60);
  Ergebnis e2 = new Ergebnis(helga, 1.0f, 118);

  //eine Pruefung erzeugen
  Pruefung prog1 = new Pruefung("Programmierung1", "28.01.2025", 9);
  prog1.neuesErgebnis(e1);
  prog1.neuesErgebnis(e2);
  System.out.println(prog1);
}
```

Die Ausgabe dazu soll sein:

```
Pruefung Programmierung1 vom 28.01.2025 mit 9 Credits
Willi Wichtig (900) Note=4.0 Punktzahl=60
Helga Eifrig (901) Note=1.0 Punktzahl =118
```

Mögliche Lösung:

```
/**
 * Student.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Student {
    String vorname;
    String nachname;
    int matrikelnummer;

    public Student(String vorname1, String nachname1, int matrikelnummer1) {
        // Wir werden spaeter noch eine Moeglichkeit kennen lernen, dass man z.B. nicht
vorname1 sagen muss
        vorname = vorname1;
        nachname = nachname1;
        matrikelnummer = matrikelnummer1;
    }

    public String toString() {
        return vorname + " " + nachname + " ( " + matrikelnummer + " ) ";
    }

    public static void main(String[] args) {
        Student willi = new Student("Willi", "Wichtig", 900);
        Student helga = new Student("Helga", "Eifrig", 901);

        // Hier wird jeweils implizit die toString-Methode aufgerufen
        System.out.println(willi.toString());
        System.out.println(helga.toString());
    }
}
```

```java
/**
 * Einzelnes Prüfungsergebnis.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Ergebnis {
    Student student;
    float note;
    int punktzahl;

    public Ergebnis(Student student1, float note1, int punktzahl1) {
        student = student1;
        note = note1;
        punktzahl = punktzahl1;
    }

    public String toString() {
        return student + " Note = " + note + " Punktzahl = " + punktzahl;
    }

    public static void main(String[] args) {
        // Zwei Studenten erzeugen
        Student willi = new Student("Willi", "Wichtig", 900);
        Student helga = new Student("Helga", "Eifrig", 901);

        // Zwei Pruefungsergebnisse erzeugen
        Ergebnis e1 = new Ergebnis(willi, 4.0f, 60);
        Ergebnis e2 = new Ergebnis(helga, 1.0f, 118);

        // Hier wird jeweils implizit die toString-Methode aufgerufen
        System.out.println(e1.toString());
        System.out.println(e2.toString());
    }
}
/**
 * Pruefung
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Pruefung {
  String name;
  String datum;
  int credits;
  Ergebnis[] ergebnisse;

  Pruefung(String name1, String datum1, int credits1) {
    name = name1;
    datum = datum1;
    credits = credits1;
    ergebnisse = new Ergebnis[0];
  }

  void neuesErgebnis(Ergebnis e) {
    // Feld anlegen, das ein laengere Liste ist
    Ergebnis[] neu = new Ergebnis[ergebnisse.length + 1];
    // Alte Ergebnisse kopieren
    for (int i = 0; i < ergebnisse.length; i++) {
```

```
    neu[i] = ergebnisse[i];
  }
  // Neues Ergebnis hinzufügen
  neu[ergebnisse.length] = e;
  // Alte Ergebnisliste ersetzen
  ergebnisse = neu;
}

public String toString() {
  String s = "Pruefung " + name + " vom " + datum + " mit " + credits + " Credits \n";
  for (int i = 0; i < ergebnisse.length; i++) {
    s += ergebnisse[i].toString() + "\n";
  }
  return s;
}

public static void main (String[] args){
  //zwei Studenten erzeugen
  Student willi = new Student("Willi" ,"Wichtig", 900);
  Student helga = new Student ("Helga" , "Eifrig" , 901);

  //zwei Pruefungsergebnisse erzeugen
  Ergebnis e1 = new Ergebnis(willi, 4.0f, 60);
  Ergebnis e2 = new Ergebnis(helga, 1.0f, 118);

  //eine Pruefung erzeugen
  Pruefung prog1 = new Pruefung("Programmierung1", "28.01.2025", 9);
  prog1.neuesErgebnis(e1);
  prog1.neuesErgebnis(e2);
  System.out.println(prog1);
}
```

Aufgabe 2: Getränkekarte

Achtung

Diese Aufgabe ist eine größere Aufgabe, bestehend aus mehreren Teilaufgaben mit ansteigendem Schwierigkeitsgrad, die aufeinander aufbauen. Hinweise vorab:

- Nutzen Sie ausschließlich Java-Konstrukte, die in der Vorlesung bis jetzt vorkamen.
- Nutzen Sie keine Umlaute.
- Sie können zum Beispiel für spätere Teilaufgaben auch weitere Methoden in die geforderten Klasse aufnehmen.
- Alle Methoden (außer `main`) sind Instanzmethoden, also ohne `static`.
- Sie dürfen diese Aufgabenstellung nach eigenem Belieben erweitern.

a) Geben Sie eine Java-Klasse `Person` an, die eine Person beschreibt. Eine Person hat einen Vornamen und ein Alter, beides muss man bei der Instanziierung angeben. Geben Sie eine Methode `public String toString()` an, die einen String mit Namen und Alter der Person liefert. Nutzen Sie folgendes Testprogramm in einem `main`, um ihre Person-Klasse zu testen:

```
public static void main(String[] args) {
    Person paul = new Person("Paul", 16);
    // Hier wird implizit die toString-Methode aufgerufen
    System.out.println(paul.toString());
}
```

Die Ausgabe dazu ist:

```
Paul (16)
```

b) Geben Sie in Java eine Klasse `Getraenk` an, die eine Getränkeportion in einem Restaurant / Bar beschreibt. Zu einer solchen Beschreibung gehören der Name, die Abgabemenge pro Glas in ml (ganzzahlig), der Preis in Cent dafür (ebenfalls ganzzahlig) sowie ein Freigabealter (ganzzahlig), ab dem ein Gast dieses Getränk bestellen kann. Sehen Sie einen Konstruktor vor, der aus solchen Angaben ein entsprechendes Java-Objekt erzeugt. Die Klasse soll eine `toString`-Methode haben, die einen Text zu den gespeicherten Angaben zu diesem Getränk liefert. Nutzen Sie folgendes Testprogramm in einem `main`, um ihre Getraenk-Klasse zu testen:

```
public static void main(String[] args) {
    // Getraenke erzeugen
    Getraenk cola = new Getraenk("Cola", 200, 250, 0);
    Getraenk bier = new Getraenk("Bier", 200, 200, 16);
    Getraenk vodka = new Getraenk("Vodka", 20, 300, 18);

    // Getraenk auf Bildschirm ausgeben
    // Hier wird implizit die toString-Methode aufgerufen
    System.out.println(cola.toString());
}
```

Die Ausgabe hierzu ist:

```
Name: Cola, Menge: 200, Preis: 250, Altersfreigabe: 0
```

c) Geben Sie eine Java-Klasse `Getraenkekarte` für Getränkekarten an. Auf einer Getränkekarte gibt es insgesamt $n \in \mathbb{N}$ Getränke (in Java also Getränkeobjekte), jeweils wie oben beschrieben. Man kann eine leere Getränkekarte erzeugen und anschließend ein Getränk(-objekt) nach dem anderen hinzufügen. Hinweis: Feld, das jeweils neu und größer angelegt wird mit kopiertem Inhalt des alten Feldes. Mit einer Methode `toString` kann man sich die gesamte Getränkekarte als `String` (über mehrere Zeilen) geben lassen, pro Zeile ein Getränk. Weiterhin kann man in einer Getränkekarte über einen Getränkenamen eine Getränk suchen. Das Ergebnis einer erfolgreichen Suche ist das entsprechende Java-Objekt für das Getränk. Wenn man kein solches Getränk findet, soll diese Methode `null` liefern. Nutzen Sie folgendes Testprogramm in einem `main`, um die Getränkekarte zu testen:

```
public static void main(String[] args) {
    // leere Karte erzeugen
    GetraenkeKarte gk = new GetraenkeKarte();

    // Getränke erzeugen
    gk.neuesGetraenk(new Getraenk("Cola", 200, 250, 0));
    gk.neuesGetraenk(new Getraenk("Bier", 200, 200, 16));
    gk.neuesGetraenk(new Getraenk("Vodka", 20, 300, 18));

    // Karte auf dem Bildschirm anzeigen lassen
    System.out.println(gk.toString());

    // Getraenk suchen
    Getraenk g = gk.suchen("Cola");
    if (g != null) {
        System.out.println(g.toString());
    }
}
```

Die Ausgabe dazu ist:

```
Name : Cola, Menge: 200, Preis: 250, Altersfreigabe: 0
Name : Bier, Menge: 200, Preis: 200, Altersfreigabe: 16
Name : Vodka, Menge: 20, Preis: 300, Altersfreigabe: 18

Name : Cola, Menge: 200, Preis: 250, Altersfreigabe: 0
```

d) Geben Sie eine Java-Klasse `Lounge` an, die eine Lounge (Bar) beschreibt. In einer Lounge gibt es bei Instanziierung eine Getränkekarte (mit Inhalt). Siehe Beispiel oben. Weiterhin kann eine `Person` in einer Lounge über den Getränkenamen eine Bestellung aufgeben. In diesem Fall muss überprüft werden, ob das Getränk auf der Karte steht und ob die Person alt genug für dieses Getränk ist. Im Fehlerfall soll eine entsprechende Nachricht auf dem Bildschirm ausgegeben werden. Nutzen Sie folgendes Testprogramm in einem main, um die Lounge zu testen:

```
public static void main(String[] args) {
    // Erzeuge Lounge mit StandardgetraenkeKarte
    Lounge lounge = new Lounge();

    // Erzeuge eine Person
    Person paul = new Person("Paul", 16);

    // Paul laesst sich die Karte zeigen
    // Zum Verständnis: hier wird implizit die toString-Methode des GetraenkeKarte-
Objekts aufgerufen
    System.out.println("Unsere Karte: \n" + lounge.getKarte().toString());

    // Paul bestellt
    lounge.bestellen(paul, "Vodka");
    lounge.bestellen(paul, "Bier");
}
```

Die Ausgabe dazu ist:

```
Unsere Karte:
Name: Cola, Menge: 200, Preis: 250, Altersfreigabe: 0
Name: Bier, Menge: 200, Preis: 200, Altersfreigabe: 16
Name: Vodka, Menge: 20, Preis: 300, Altersfreigabe: 18

nicht alt genug fuer das Getraenk
bitte schoen, ihr Getraenk
```

Mögliche Lösung:

```java
/**
 * Person.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Person {
  String vorname;
  int alter;

  public Person(String vorname, int alter) {
    this.vorname = vorname;
    this.alter = alter;
  }

  public int getAlter() {
    return alter;
  }

  public String toString() {
    return vorname + " (" + alter + ") ";
  }

  public static void main(String[] args) {
    Person paul = new Person("Paul", 16);
    // hier wird implizit die toString-Methode aufgerufen
    System.out.println(paul);
  }
/**
 * Getränk.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Getraenk {
  // Instanzvariablen
  String name;
  int abgabemenge;
  int preis;
  int altersfreigabe;

  // Konstruktor
  public Getraenk(String name, int abgabemenge, int preis, int altersfreigabe) {
    this.name = name;
    this.abgabemenge = abgabemenge;
    this.preis = preis;
    this.altersfreigabe = altersfreigabe;
  }

  public String getName() {
    return name;
  }

  public int getAltersfreigabe() {
    return altersfreigabe;
  }

  public String toString() {
    return "Name : " + name + ", Menge : " + abgabemenge + ", Preis : " + preis + ",
```

```
Altersfreigabe : " + altersfreigabe;
  }

  public static void main(String[] args) {
    // Getraenke erzeugen
    Getraenk cola = new Getraenk("Cola", 200, 250, 0);
    Getraenk bier = new Getraenk("Bier", 200, 200, 16);
    Getraenk vodka = new Getraenk("Vodka", 20, 300, 18);

    // Getraenk auf Bildschirm ausgeben
    // hier wird implizit die toString-Methode aufgerufen
    System.out.println(cola.toString());
  }
/**
 * Getraenkekarte.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Getraenkekarte {
  // Instanzvariable
  Getraenk[] getraenke = new Getraenk[0];

  // Konstruktor
  public Getraenkekarte() {
    // Nichts zu tun hier
    // Man könnte diesen Konstruktor auch weglassen
  }

  // Neues Getraenk kommt auf Karte
  public void neuesGetraenk(Getraenk g) {
    // Feld mit Getraenken eins groesser als vorher anlegen
    Getraenk[] neueKarte = new Getraenk[getraenke.length + 1];
    // Alte Getraenke kopieren
    for (int i = 0; i < getraenke.length; i++) {
      neueKarte[i] = getraenke[i];
    }
    // Haenge das neue Getraenk ans Ende dran
    neueKarte[getraenke.length] = g;
    // Ersetze Karte (Java-Feld) durch neue Version
    getraenke = neueKarte;
  }

  // Suche Getraenk über Namen
  public Getraenk suchen(String name) {
    // Suche alle Getraenke nach Namen durch
    for (int i = 0; i < getraenke.length; i++) {
      if (getraenke[i].getName().equals(name)) {
        // Gefunden
        return getraenke[i];
      }
    }
    // Nicht gefunden
    return null;
  }

  // Gesamte Karte als String über mehrere Zeilen geben
  public String toString() {
    String s = "";
```

```
    for (int i = 0; i < getraenke.length; i++) {
      s += getraenke[i] + "\n";
    }
    return s;
  }

  public static void main(String[] args) {
    // leere Karte erzeugen
    GetraenkeKarte gk = new GetraenkeKarte();

    // Getränke erzeugen
    gk.neuesGetraenk(new Getraenk("Cola", 200, 250, 0));
    gk.neuesGetraenk(new Getraenk("Bier", 200, 200, 16));
    gk.neuesGetraenk(new Getraenk("Vodka", 20, 300, 18));

    // Karte auf dem Bildschirm anzeigen lassen
    System.out.println(gk.toString());

    // Getraenk suchen
    Getraenk g = gk.suchen("Cola");
    if (g != null) {
      System.out.println(g.toString());
    }
  }
}
/**
 * Lounge.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Lounge {
  // Instanzvariablen
  Getraenkekarte karte;
  Person[] gaeste = new Person[0];

  // Konstruktor
  public Lounge() {
    // Getraenkekarte erzeugen
    karte = new Getraenkekarte();
    // Getraenke erzeugen
    karte.neuesGetraenk(new Getraenk("Cola", 200, 250, 0));
    karte.neuesGetraenk(new Getraenk("Bier", 200, 200, 16));
    karte.neuesGetraenk(new Getraenk("Vodka", 20, 300, 18));
  }

  public void bestellen(Person p, String getraenkname) {
    Getraenk g;
    if ((g = karte.suchen(gettraenkname)) == null) {
      System.out.println("Getränk " + gettraenkname + " nicht vorhanden");
    } else if (p.getAlter() < g.getAlterfreigabe()) {
      System.out.println("nicht alt genug für das Getränk");
    } else {
      System.out.println("bitteschön, Ihr Getränk");
    }
  }
  public static void main(String[] args) {
    // Erzeuge Lounge mit StandardgetraenkeKarte
    Lounge lounge = new Lounge();
```

```
    // Erzeuge eine Person
    Person paul = new Person("Paul", 16);

    // Paul laesst sich die Karte zeigen
    // Zum Verständnis: hier wird implizit die toString-Methode des GetraenkeKarte-Objekts
aufgerufen
    System.out.println("Unsere Karte: \n" + lounge.getKarte().toString());

    // Paul bestellt
    lounge.bestellen(paul, "Vodka");
    lounge.bestellen(paul, "Bier");
  }

}
```

Aufgabe 3: Rationale Zahlen 👾

Schreiben Sie eine Klasse `Rational`, die rationale Zahlen (also Brüche) darstellt. Ein Bruch besteht aus einem ganzzahligen Zähler und einem ganzzahligen Nenner, der nicht 0 ist. Beispiel: 1/2. Damit diese Werte eindeutig bestimmt und nicht unnötig groß sind, werden Brüche immer intern in gekürzter Form dargestellt. Dazu werden Zähler und Nenner durch ihren größten gemeinsamen Teiler geteilt. Nutzen Sie dazu eine Methode ggT in ihrer Klasse (siehe frühere Vorlesung), die Sie selbst schreiben müssen. Bitte beachten Sie, dass Sie auf Absolutwerten den ggT bilden müssen (falls Zähler und/oder Nenner negativ sind). Ist weiterhin der Nenner negativ, so werden die Vorzeichen von Zähler und Nenner gewechselt, so dass der Nenner auf jeden Fall positiv ist. Beispiele:

- Ein Bruch 2/4 wird intern als 1/2 dargestellt (ggT ist 2)
- Ein Bruch −2/-1 wird intern als 2/1 dargestellt
- Ein Bruch 1/-2 wird intern als −1/2 dargestellt

Statten Sie die Klasse mit zwei Konstruktoren aus:

- Zu einer ganzen Zahl x wird der Bruch erzeugt, der zu dieser Zahl gehört.
- Zu zwei ganzen Zahlen x, y initialisieren Sie das Objekt so, dass sein Wert dem Quotienten $\frac{x}{y}$ dieser Zahlen entspricht. Dabei sind Vorzeichen zu behandeln, zu kürzen und Fehler zu melden (falls der Nenner gleich 0 ist, so soll die Meldung ausgegeben werden: `Fehler: Nenner ist 0`).

Schreiben Sie eine Instanzmethode `toString`, die einen String zurückgibt, der den dargestellten Bruch als Zeichenkette `<zaehler>/<nenner>` repräsentiert (z.B. $\frac{3}{4}$ als `3/4`). Schreiben Sie ferner vier Instanzmethoden `add`, `sub`, `mul` und `div`, die jeweils eine Referenz auf ein weiteres Objekt vom Typ `Rational` annehmen und ein neues solches Objekt zurückgeben, wobei dieses neue Objekt entsprechend den arithmetischen Operationen gebildet werden soll. Z.B. soll `a.add(b)` den Wert von $a + b$ (als neues Objekt!) ergeben. Schreiben Sie schließlich eine Klasse `RationalTest` mit einer `main`-Methode, die die Klasse Rational testet. In main soll über die Tastatur zur Laufzeit Brüche und Operationssymbole eingegeben werden. Die erlaubte Eingabesyntax wird durch die folgende EBNF-Beschreibung angegegeben:

$$
\begin{aligned}
<Eingabe> &::= <Bruch>\{<Operator>\ <Bruch>\}\ . \\
<Bruch> &::= <Zahl>\ <Zahl> \\
<Zahl> &::= \text{ganze Zahl in Dezimaldarstellung} \\
<Operator> &::= + \mid - \mid * \mid /
\end{aligned}
$$

Die Operatoren `+`, `-`, `*`, `/` sind selbsterklärend. Die Eingabe soll (entsprechend diesen Syntaxregeln) streng von links nach rechts ausgewertet werden, also ohne Punkt-vor-String Regeln etc.

Beispiel

Beispiel 1 für eine Eingabe:

```
1 2
+
2 4
.
```

erzeugt als Ausgabe

```
1/1
```

Beispiel 2 für eine Eingabe:

```
1 2
+
2 4
-
4 -3
.
```

erzeugt als Ausgabe

```
7/3
```

Info

Ihre Lösung wird aus 2 Dateien bestehen: Rational.java und RationalTest.java, also für jede (öffentliche) Klasse eine Datei. Diese müssen Sie als einzelne Dateien im Praktomat hochladen, nicht als Archiv (.zip, .tgz, .jar, . . .)!

Aufgabe 4: Wegpunkte 👾

Schreiben Sie eine Klasse `Punkt` zu einem 2D-Punkt mit ganzzahligen x- und y-Koordinaten. Sehen Sie einen Konstruktor zum Erzeugen eines solchen Punktes vor, dem zwei ganzzahlige Werte übergeben werden. Die Klasse soll weiterhin eine Methode `public String toString()` enthalten, die zu einem Punkt einen String der Form `(x,y)` erzeugt, wobei x, y die entsprechenden Punktangaben zu diesem Punkt sind. Schreiben Sie weiterhin eine Klasse `Weg`, die Wege beschreiben kann. Ein Weg hat $n > 0$ Wegpunkte $p_1, ..., p_n$, die jeweils ein Punkt sind. Nutzen Sie dazu in der Klasse `Weg` sinnvoll die `Punkt`-Klasse im Zusammenhang mit einem Feld, das zu jedem Zeitpunkt die gleiche Länge hat wie die Länge des aktuellen Weges. Es gibt keine Längenbeschränkung für Wege! Hinweis: In einer Referenzvariablen lässt sich die Referenz auf ein Feld durch eine Referenz auf ein anderes (längeres) Feld ersetzen. Sehen Sie folgende Methoden in der Klasse Weg vor, die genau nach diesen Vorgaben existieren müssen:

- Es gibt genau einen Konstruktor, der zu zwei `int`-Werten `x, y` einen Weg mit genau einem Punkt erzeugt. Direkt über einen Konstruktor lassen sich also nur Wege der Länge 1 erzeugen.
- Es gibt eine Instanzmethode `public int getAnzahl()`, die die Anzahl der Wegpunkte des Weges liefert.
- Es gibt eine Instanzmethode `public void verlaengern(Weg w)`, die zu dem eigenen Weg (sprich das Bezugsobjekt) einen weiteren Weg anhängt, indem alle Punkte des zweiten Weges an den eigenen Weg angehängt werden. Dadurch wird also die eigene Instanz verändert. Beispiel: Zum Weg $w_1 = (0, 0)$ wird der Weg $w_2 = (1, 1)$ angehängt: `w1.verlaengern(w2)`. Danach gilt $w_1 = (0, 0) - (1, 1)$, er enthält also zwei Punkte. Hängt man an diesen Weg wiederum den Weg $w_3 = (3, 4)$ an (`w1.verlaengern(w3)`), so gilt anschließend $w_1 = (0, 0) - (1, 1) - (3, 4)$.
- Es gibt eine Methode public String `toString()`, die einen Weg als String liefert in der Form $(x_1, y_1) - (x_2, y_2) - ... - (x_n, y_n)$, wobei die x_i, y_i die ganzzahligen Koordinaten des entsprechenden Punktes sind. Nutzen Sie dabei sinnvoll die Methode toString der Punkt-Klasse.

Schreiben Sie in einer weiteren Testklasse `WegTest` ein `main`. Erzeugen Sie darin einen Weg $w_1 = (1, 1)$ und einen Weg $w_2 = (2, 2)$. Lassen Sie beide Wege nacheinander auf dem Bildschirm ausgeben. Dann verlängern

Sie w_1 um w_2 und lassen wieder den Weg w_1 auf dem Bildschirm ausgeben. Die Gesamtausgabe muss dann exakt so sein:

```
(1,1)
(2,2)
(1,1)-(2,2)
```

Info

Es gibt nicht eine Methode in irgendeiner dieser Klassen, die als Argument ein Feld übergeben bekommt. Denken Sie in Wegen und Punkten (und Instanzen)!

Aufgabe 5: Polynom 👾

Ein Polynom ist eine Funktion $P(x) : \mathbb{R} \to \mathbb{R}$ mit $P(x) = \sum_{i=0}^{n} a_i x^i, n \geq 0, a_0, ..., a_n \in \mathbb{R}$. Die Werte $a_0, ..., a_n$ heißen Koeffzienten. Oder anders ausgedrückt: durch Angabe von Koeffzientenwerten $a_0, ..., a_n \in \mathbb{R}$ ist ein Polynom im obigen Sinne definiert. Für ein Argument x lässt sich dieses Polynom dann auswerten mit $P(x) = \sum_{i=0}^{n} a_i x_i$. Beispiel: Die Koeffizienten $a_2 = 1, a_1 = 2, a_0 = 3$ defnieren das Polynom $P(x) = 1 \cdot x^2 + 2 \cdot x^1 + 3 \cdot x^0 = x^2 + 2x + 3$. Der Wert dieses Polynoms an der Stelle 2 wäre dann $P(2) = 4 + 4 + 3 = 11$. Entwickeln Sie eine Klasse Polynom in Java zur Repräsentation von Polynomen.

1. Über einen Konstruktor soll sich durch alleinige Angabe von Koeffzienten in Form eines Feldes von Fließkommawerten (`double[]`) ein neues Polynom anlegen lassen. Durch die Länge des übergebenen Feldes ist implizit auch der Grad des Polynoms bestimmt. Der Koeffizient mit der niedrigsten Wertigkeit steht im Feld an der Positon 0. Beispiel: Durch die Übergabe des Feldes `[3,2,1]` würde obiges Beispielpolynom definiert. Sorgen Sie außerdem dafür, dass nur normalisierte Polynome entstehen, in denen der höchstwertige Koeffizient ungleich 0 ist (sofern es sich nicht um das konstante Nullpolynom handelt). Beispiel: Werden bei der Erzeugung eines Polynoms die Koeffzienten 2, 1, 0 (für $0 \cdot x^2 + 1 \cdot x + 2$) angegeben, so soll daraus implizit das Polynom 2, 1 (für $1 \cdot x + 2$) entstehen. Hinweis: public `Polynom(double[] koeffizienten)`
2. Entwickeln Sie weiterhin eine Instanzmethode `toString()`, die das Polynom als lesbaren String zurück gibt. Beachten Sie, dass Polynome üblicherweise absteigend von der höchsten zur niedrigsten Potenz notiert werden. Hinweis: `public String toString()` Beispiel: `1*x^2 + 2*x^1 + 3*x^0`
3. Geben Sie schließlich eine Instanzmethode `public double auswerten(double x)` an, die für ein Argument x den Wert des Polynoms an der Stelle x zurückgibt (Beispiel siehe oben).
4. Geben Sie eine Klassenmethode `public static int getAnzahl()` an, die die Anzahl der bis dahin erzeugten Polynome zum Zeitpunkt des Methodenaufrufs liefert.

Schreiben Sie eine zweite Klasse `PolynomTest`. Diese Klasse soll eine `main`-Methode besitzen, in der Sie das Polynom $P(x) = 1 \cdot x^2 + 2 \cdot x + 3$ (als Objekt) erzeugen, dieses Polynomobjekt auf dem Bildschirm in lesbarer Form ausgeben und das Polynom auswerten an der Stelle 2 und diesen Wert auf dem Bildschirm ausgeben. Geben Sie anschließend auch noch die Anzahl der erzeugten Polynome auf dem Bildschirm aus. Diese drei Ausgaben sollen jeweils in einer eigenen Zeile stehen. Die Ausgabe zu obigem Beispiel wäre dann:

```
1.0*x^2 + 2.0*x^1 + 3.0*x^0
11.0
1
```

Überlegen Sie sich, wo mögliche Fehler in ihrer Polynomklasse auftreten können. Geben Sie an den Stellen in ihrem Programm eine Meldung auf dem Bildschirm aus, die jeweils beginnt mit `Fehler:`

Freiwillige Erweiterung: Recherchieren Sie, was das Horner-Schema im Zusammenhang mit der Polynomauswertung bedeutet. Implementieren Sie dieses Verfahren zur Auswertung des Polynoms.

Aufgabe 6: Rückgabeautomat

Programmieren Sie eine Klasse Rueckgabeautomat, die die minimale Anzahl an vorhandenen Geldstücken zurückgibt, die auf einen Einzahlbetrag einzahlung (Beispiel 10 Euro) und einen zu zahlenden Betrag zahlbetrag (Beispiel 8,57 Euro) sich aufgrund des aktuellen Münzbestands im Automaten ergeben. Ein Automat kann nur Geldstücke der Wertigkeit 1,2,5,10,20,50,100 und 200 Cent enthalten. Beim Erzeugen eines Geldautomaten gibt man als Argument an, wieviele Geldstücke davon zu Beginn im Automaten enthalten sein sollen (ein Feld mit 8 Zahlen). Geben Sie eine Instanzmethode an, die den aktuellen Münzbestand ermittelt. Das Ergebnis ist die Anzahl der Münzen in den entsprechenden Wertigkeiten (also wieder ein int-Feld der Größe 8). Sehen Sie weiterhin eine Instanzmethode vor, die für einen Einzahlbetrag und den tatsächlich zu zahlenden Betrag die Anzahl an Münzen mit der entsprechenden Wertigkeit des Rückgelds berechnet und als Ergebnis der Methode liefert. Falls der aktuelle Bestand kein korrektes Rückgeld erlaubt, geben Sie eine Fehlermeldung auf dem Bildschirm aus. Das Ergebnis der Methode ist wiederum ein Feld der Länge 8 mit der Anzahl zu der entsprechenden Wertigkeit.

Beispiel

Ein Rückgabeautomat enthält die folgende Anzahl an Geldstücken für die vorgegebenen Wertigkeiten `[200,100,50,20,10,5,2,1]: [2,2,2,2,2,2,2,2]` also jeweils zwei Geldmünzen der entsprechenden Wertigkeit. Auf den Einzahlbetrag 1 Euro und den tatsächlich zu zahlenden Betrag von 57 Cent gibt der Automat als Ergebnis `[0,0,0,2,0,0,1,1]` (43 Cent als 2 Zwanziger, 1 Zweier und 1 Einer) und hat als Bestand danach noch `[2,2,2,0,2,2,1,1]`. Instanziieren Sie zwei Rückgabeautomaten mit unterschiedlichem Bestand und lassen sich für den zu zahlenden Betrag von 57 Cent auf den Betrag von 100 Cent Rückgeld geben.

Mögliche Lösung:

```
/**
 * Geldrueckgabeautomat.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Rueckgabeautomat {
  // Die Werte sind für alle unsere Automaten gleich, daher Klassenvariable
  private static int[] wertigkeiten = {200, 100, 50, 20, 10, 5, 2, 1};

  // Anzahl getaetigter Auszahlungen dieses Automaten
  private int auszahlungen;

  // Aktueller Bestand dieses Automaten
  private int[] muenzbestand = {0, 0, 0, 0, 0, 0, 0, 0};

  // Erzeuge Rueckgabeautomat
  public Rueckgabeautomat(int[] anfangsbestand) {
    auszahlungen = 0;
    for (int i = 0; i < anfangsbestand.length; i++) {
      muenzbestand[i] = anfangsbestand[i];
    }
  }

  // Liefer aktuellen Münzbetstand
  public int[] getMuenzbestand() {
    //wir klonen das Feld, damit niemand in unserem feld rumpfuschen kan
    return muenzbestand.clone();
  }
  /**
```

```
 * Berechnet das Rückgeld.
 *
 * @param einzahlbetrag Der eingezahlte Betrag in Cent.
 * @param betrag Der zu zahlende Betrag in Cent.
 * @return Die Anzahl der Münzen mit der vorgebenen Wertigkeit.
 */
public int[] zahlen(int einzahlbetrag, int betrag) {
  // Eine Auszahlung mehr
  auszahlungen++;

  // Das ist unser Rückgeld, das wir auszahlen müssen
  int rueckgeld = einzahlbetrag - betrag;
  System.out.println("Rückgeld = " + rueckgeld);

  // Hier wird angegeben, wie viele Münzen
  // die entsprechende Wertigkeit des Rückgeldes ausmachen
  int[] muenzen = new int[wertigkeiten.length];

  // Ermittler Anzahl aller Münzen
  for (int i = 0; i < wertigkeiten.length; i++) {
    // Alles erledigt. Wir sind fertig.

    // Passiert die aktuelle Münze?
    if (wertigkeiten[i] <= rueckgeld) {
      // Diese Münze passt

      // Anzahl Münzen ermitteln
      int anzahl = rueckgeld / wertigkeiten[i];

      // Haben wir überhaupt noch so viel vorrätig?
      if (anzahl > muenzbestand[i]) {
        anzahl = muenzbestand[i];
      }

      // So viele Münzen geben wir wie aus
      muenzen[i] = anzahl;

      // Reduziere Bestand
      muenzbestand[i] -= anzahl;

      // Verbleibendes Rückgeld reduzieren
      rueckgeld -= anzahl * wertigkeiten[i];
      System.out.println(wertigkeiten[i] + " : " + anzahl);
    }
  }

  if (rueckgeld > 0) {
    System.out.println("Achtung: korrekte Rueckgabe war nicht möglich!");
  }

  return muenzen;
}

private static void zeigeGeld(String ueberschrift, int[] anzahl) {
  System.out.println(ueberschrift + " ");
  for (int i = 0; i < anzahl.length; i++) {
    if (anzahl[i] > 0) {
      System.out.print(wertigkeiten[i] + " Cent : " + anzahl[i] + ", ");
    }
```

```
    }
    System.out.println();
  }

  public static void main(String[] args) {
    int[] anzahl1 = {1,1,1,1,1,1,1,1};
    int[] anzahl2 = {2,2,2,2,2,2,2,2};

    Rueckgabeautomat a1 = new Rueckgabeautomat(anzahl1);
    Rueckgabeautomat a2 = new Rueckgabeautomat(anzahl2);

    zeigeGeld("Bestand Automat 1", a1.getMuenzbestand());
    zeigeGeld("Bestand Automat 2", a2.getMuenzbestand());

    int[] rueckgeld1 = a2.zahlen(100, 57);

    zeigeGeld("Rueckgeld", rueckgeld1);

    zeigeGeld("Bestand Automat 2", a2.getMuenzbestand());

  }
}
```

Aufgabe 7: Abbildung

Eine mathematische Funktion f bildet eine Zahl x (Parameter) auf eine neue Zahl $f(x)$ (Funktionswert) ab. Aus der Sicht von Java kann das durch eine Methode mit dem Kopf `double map(double x)` ausgedrückt werden, die den entsprechenden Funktionswert berechnet.

1. Entwickeln Sie eine Basisklasse `Function` mit dieser Methode. Diese realisierte Methode soll in der Basisklasse die Identitätsfunktion sein mit `map(x) = x`.
2. Eine Parabel ist eine Funktion der Form $f(x) = ax^2 + bx + c$. Definieren Sie eine Klasse Parabel, die von `Function` abgeleitet ist und deren Konstruktor Werte für a, b, c akzeptiert.
3. Defnieren Sie ebenso eine Klasse `Hyperbel` für Funktionen der Form $f(x) = \frac{a}{x}$. Wie muss hier der Konstruktor aussehen?
4. Ein Funktionsverlauf kann in einer Wertetabelle grob dargestellt werden, die für eine Reihe von x-Werten in regelmäßigen Abständen jeweils den Funktionswert $f(x)$ aurfilistet. Definieren Sie in der passenden Klasse eine Methode `print`, die drei Parameter akzeptiert (kleinster x-Wert, größter x-Wert, Abstand zwischen zwei x-Werten), und die eine derartige Wertetabelle auf dem Bildschirm ausgibt.
5. Schreiben Sie ein Java-Programm, das die Funktion $f(x) = x^2 - 2x + 2$ erzeugt und deren Wertetabelle im Bereich $-5 \leq x \leq 5$ in Schritten von $0,1$ ausgibt.
6. $(*)$ Zwei Funktionen f und g können verkettet werden. Die Verkettung ist eine neue Funktion $h(x) = g(f(x))$. Definieren Sie eine Klasse `Composed`, die von `Function` abgeleitet ist und im Konstruktor zwei andere Funktionen akzeptiert, deren Verkettung Sie repräsentiert.
7. $(*)$ Schreiben Sie ein Java-Programm, das die Funktion $f(x) = x^2 - 2x + 2$ erzeugt und deren Wertetabelle im Bereich $-5 \leq x \leq 5$ in Schritten von $0,1$ ausgibt.

Mögliche Lösung:

```
/**
 * Basisklasse für eine Funktion.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Function {
```

```
    public double map(double x) {
        return x;
    }

    public void print(double klein, double groess, double schritt) {
        for (double x = klein; x <= groess; x += schritt) {
            System.out.println("f(" + x + ") = " + map(x));
        }
    }

}
/**
 * Parabel-Klasse.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Parabel extends Function {

    private double a;
    private double b;
    private double c;

    public Parabel(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double map(double x) {
        return a*x*x + b*x + c;
    }

}
/**
 * Hyperbel-Klasse.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Hyperbel extends Function {

    private double a;

    public Hyperbel(double a) {
        this.a = a;
    }

    public double map(double x) {
        return a / x;
    }

}
/**
 * Komposition von Funktionen.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
```

```
 */
public class Composed extends Function {

    private Function f;
    private Function g;

    public Composed(Function f, Function g) {
        this.f = f;
        this.g = g;
    }

    public double map(double x) {
        return g.map(f.map(x));
    }

}
/**
 * Testprogramm fuer Funktionen (Version 1).
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Test1 {
    public static void main(String[] args) {
        // Erzeuge eine neue Parabel mit den Parametern a=1, b=-2 und c=2
        Function f = new Parabel(1.0, -2.0, 2);

        // Druckt die Funktion für bestimmte Werte aus
        f.print(-1.0, 1.0, 0.1);

  //zur Kontrolle
      System.out.println("zur Kontrolle: ");

      for (double x = -1.0; x <= 1.0; x += 0.1) {
          System.out.println("x=" + x + ", f(x)=" + (x*x-2*x+2));
      }
    }
}
/**
 * Testprogramm fuer Funktionen.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Test2 {
    public static void main(String[] args) {

        Function f = new Parabel(1.0, -2.0, 2);
        Function g = new Hyperbel(1.0);
        Function h = new Composed(f, g);

        h.print(-1.0, 1.0, 0.1);

  //zur Kontrolle
        System.out.println("zur Kontrolle: ");

        for (double x = -1.0; x <= 1.0; x += 0.1) {
          System.out.println("x=" + x + ", f(x)=" + (1.0 / (x*x-2*x+2)));
```

```
            }
        }
    }
```