Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Entwurf einer Klassenhierarchie): (11 Punkte)

In dieser Aufgabe sollen Sie einen Teil der Möbelwelt modellieren.

- Ein Möbelstück hat ein Gewicht und ein bestimmtes Material. In dieser Aufgabe betrachten wir Sitzmöbel, Tische und Behältnismöbel.
- Ein Tisch ist ein Möbelstück mit einer Anzahl von Tischbeinen und einer bestimmten Länge und einer bestimmten Breite.
- Ein Esstisch bietet einer gewissen Anzahl von Personen Platz. Er kann aber auch um eine bestimmte Länge und eine bestimmte Breite verlängert werden.
- Ein Sitzmöbelstück hat eine gewisse Anzahl von Sitzplätzen.
- Ein Stuhl ist ein Sitzmöbelstück. Stühle können an einen Tisch gestellt werden. Dies funktioniert nicht immer.
- Ein Schaukelstuhl ist ein besonderer Stuhl. Er kann schaukeln.
- Ein Behältnismöbelstück hat ein Volumen und kann gefüllt werden.
- Ein Schrank ist ein Behältnismöbelstück mit einer bestimmten Anzahl von Fächern. Einen Schrank kann man öffnen.
- Kleiderschränke können über einen Spiegel verfügen oder nicht. Ein Kleiderschrank kann aufgeräumt werden.
- Ein Bücherregal ist ein Behältnismöbelstück, welches natürlich Bücher enthält. Ein Buch kann entnommen werden, wenn es im Regal steht.
- Ein Buch hat einen Titel und eine Anzahl von Seiten. Bücher können gelesen werden.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Gegenständen. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden, falls dies sinnvoll ist. Ergänzen Sie außerdem geeignete Methoden, um das Verlängern eines Esstischs, das Stellen eines Stuhls an einen Tisch, das Schaukeln eines Schaukelstuhls, das Füllen eines Behältnismöbelstücks, das Öffnen eines Schranks, das Aufräumen eines Kleiderschranks, das Entnehmen eines Buchs aus einem Regal sowie das Lesen eines Buchs zu modellieren.
Verwenden Sie hierbei die Notation aus der entsprechenden Tutoriumsaufgabe.

Aufgabe 5 (Überschreiben, Überladen und Verdecken): (4 + 6 = 10 Punkte)

Betrachten Sie die folgenden Klassen:

Listing 1: A.java

```
1  public class A {
2      public static int x = 0;
3      public int y = 7;
4
5      public A () {                                  // Signatur: A()
6          this(x);
7          x++;
8      }
9
10     public A (int x) {                             // Signatur: A(I)
11         x = x + 2;
12         y = y - x;
13     }
14
15     public A (double x) {                          // Signatur: A(D)
16         y += x;
17     }
18
19     public void f(int i, A o) { }                  // Signatur: A.f(IA)
20     public void f(Long lo, A o) { }                 // Signatur: A.f(LA)
21     public void f(double d, A o) { }               // Signatur: A.f(DA)
22 }
```

Listing 2: B.java

```
1  public class B extends A {
2      public float x = 1.5f;
3      public int y = 1;
4
5      public B () {                                  // Signatur: B()
6          x++;
7      }
8
9      public B (float x) {                           // Signatur: B(F)
10         super(x);
11         super.y++;
12     }
13
14     public void f(int i, B o) { }                  // Signatur: B.f(IB)
15     public void f(int i, A o) { }                  // Signatur: B.f(IA)
16     public void f(long lo, A o) { }                // Signatur: B.f(LA)
17 }
```

Listing 3: C.java

```
1  public class C {
2      public static void main (String [] args) {
3          // a)
4          A a1 = new A(5);                           // (1)
5          System.out.println (A.x);
6          System.out.println (a1.y);
7          A a2 = new A();                            // (2)
8          System.out.println (A.x);
9          System.out.println (a2.y);
10         B b = new B();                             // (3)
11         System.out.println (A.x);
12         System.out.println (b.x);
13         System.out.println (((A) b).y);
14         System.out.println (b.y);
15         A ab = new B(3);                           // (4)
16         System.out.println (ab.y);
17         System.out.println (((B) ab).y);
18         // b)
19         int i = 1;
20         long l = 2;
21         double d = 3.0;
22         a1.f(i, a1);                               // (1)
23         a1.f(l, ab);                               // (2)
24         b.f(i, (B) ab);                            // (3)
25         b.f(d, ab);                                // (4)
26         ab.f(i, a1);                               // (5)
27         ab.f(i, b);                                // (6)
28     }
29 }
```

In dieser Aufgabe sollen Sie angeben, welche Methoden- und Konstruktoraufrufe stattfinden. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Verwenden Sie zur eindeutigen Bezeichnung die Funktionssignatur, die jeweils als Kommentar hinter jeder Funktionsdefinition steht. Begründen Sie Ihre Antwort kurz.

a) Geben Sie für die mit `(1)-(4)` markierten Konstruktoraufrufe in der Klasse `C` jeweils an, welche Konstruktoren in welcher Reihenfolge von Java aufgerufen werden. Notieren Sie auch die von Java implizit aufgerufenen Konstruktoren. Bedenken Sie, dass die Oberklasse von `A` die Klasse `Object` ist. Erklären Sie außerdem, welche Werte mit welchen Werten belegt werden und welche Werte durch die `println`-Anweisungen ausgegeben werden.

b) Geben Sie für die mit `(1)-(6)` markierten Aufrufe der Methode `f` in der Klasse `C` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Aufgabe 8 (Programmieren in Klassenhierarchien): (3 + 3 + 4 + 4 + 2 + 3 + 3 + 4 + 3 = 29 Punkte)

In dieser Aufgabe geht es um drei verschiedene Arten von geraden Linien im zweidimensionalen Raum, nämlich um Geraden, Strahlen und Strecken. Wir bezeichnen hier alle drei als Linien, ohne dass dieser Begriff später in der Klassenhierarchie auftaucht. Die drei Arten von Linien sind wie folgt definiert: Eine Gerade verläuft durch zwei Punkte und bis ins Unendliche in beide Richtungen weiter. Ein Strahl beginnt an einem Punkt und läuft dann durch einen anderen Punkt bis ins Unendliche weiter. Eine Strecke verläuft nur zwischen zwei Punkten und daher in keine der beiden Richtungen weiter. Hierbei gehen wir davon aus, dass der Endpunkt eines Strahls bzw. die Endpunkte einer Strecke jeweils zur Linie dazugehören.

Häufig werden Klassenhierarchien so modelliert, dass die Beziehungen der einzelnen Klassen die Realität nachbilden. So könnte es eine Oberklasse `Rechteck` geben, von der eine Unterklasse `Quadrat` erbt. Da jedes Quadrat ein Rechteck *ist*, erschließt sich intuitiv, warum diese Modellierung gewählt wurde. Man kann Klassenhierarchien aber auch anders aufbauen: Statt in der Hierarchie die Realität nachzubilden, bietet sich stattdessen ein Blick auf die zu implementierenden Eigenschaften an: Für ein Quadrat braucht es nur ein Attribut, da Breite und Länge gleich sind. Für ein Rechteck dagegen müssen Länge und Breite unabhängig voneinander gesetzt werden können. Da ein Quadrat also gewissermaßen ein Teilverhalten eines Rechtecks abbildet, könnte man auch `Quadrat` als Oberklasse und `Rechteck` als Unterklasse zu implementieren. Dies mag zwar zunächst kontraintuitiv erscheinen, kann aber in bestimmten Situationen die Vorteile, die die Vererbung uns bietet, besser ausnutzen[1]. Auch in dieser Aufgabe werden Sie eine Vererbungshierarchie implementieren, die der Logik der Implementierung und nicht der Realität folgt.

Um mit der nötigen Präzision zu rechnen, reichen die üblichen Typen zur Darstellung von Gleitkommazahlen - also `float` und `double` - nicht immer aus. Wir nutzen daher in dieser Aufgabe die Klasse `BigDecimal` aus dem Paket `java.math`, die eine Gleitkommazahl mit beliebig hoher Präzision repräsentieren kann. Um `BigDecimal`-Zahlen zu nutzen, schreiben Sie die `import`-Deklaration `import java.math.*;` als erste Zeile in der `.java`-Datei der entsprechenden Klasse. Dadurch können Sie dann alle Klassen aus dem Paket `java.math` verwenden. Auf Zahlen diesen Typs können alle üblichen arithmetischen Operationen ausgeführt werden: Die Addition zweier `BigDecimal`-Zahlen `bd1` und `bd2` wird bspw. durch den Ausdruck `bd1.add(bd2)` realisiert. Mit den anderen Operationen verhält es sich analog, schlagen Sie bei Bedarf in der Java-API[2] nach. Um Ihnen den Umgang mit der Klasse `BigDecimal` zu erleichtern, haben wir für den Vergleich zweier Werte die Methode `equalValues` sowie für das Ziehen der Quadratwurzel die Methode `sqrt` in der Klasse `BigDecimalUtils` im Moodle-Lernraum für Sie bereitgestellt.

a) Erstellen Sie die Klasse `Punkt`, die einen Punkt im zweidimensionalen Raum repräsentieren soll. Dieser ist durch eine x-Koordinate und eine y-Koordinate gegeben, die Attribute vom Typ `BigDecimal` sind. Ein Punkt soll nicht mehr verändert werden können, wenn er einmal erstellt worden ist. Schreiben Sie unter Beachtung der Prinzipien der Datenkapselung die entsprechenden Selektoren. Schreiben Sie außerdem zwei verschiedene Konstruktoren, um einen Punkt zu erstellen. Der eine soll für jedes Attribut einen entsprechenden Parameter haben und die Attribute auf die übergebenen Werte setzen. Der andere soll es dem Nutzer ermöglichen, einen neuen Punkt aus zwei Werten vom Typ `double` zu erzeugen. Implementieren Sie auch die Methode `toString()`, wobei bspw. für den Ursprung des Koordinatensystems der String `"(0,0)"` zurückgegeben werden soll.

b) Ergänzen Sie die Klasse `Punkt` um eine Methode `BigDecimal abstand(Punkt other)`. Diese soll den euklidischen Abstand zwischen dem Punkt, auf dem die Methode aufgerufen wird und dem übergebenen Punkt `other` berechnen. Beachten Sie, dass dieser Abstand nie negativ ist. Ergänzen Sie die Klasse außerdem um die Methode `boolean equals(Object obj)`, die genau dann `true` zurückgibt, wenn das übergebene Objekt `obj` vom Typ `Punkt` ist und die gleichen Koordinaten besitzt wie der Punkt, auf dem die Methode aufgerufen wird. Nutzen Sie zum Vergleich von Werten vom Typ `BigDecimal` die Methode `equalValues(BigDecimal bd1, BigDecimal bd2)` aus der Klasse `BigDecimalUtils`.

c) Erstellen Sie nun die Klasse `Gerade`, die in den Attributen `p1` und `p2` die beiden Punkte enthalten soll, durch die die Gerade verläuft. Auch eine Gerade soll nicht mehr verändert werden können, wenn sie einmal

[1] https://de.wikipedia.org/wiki/Kreis-Ellipse-Problem
[2] https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/math/BigDecimal.html

erstellt wurde. Schreiben Sie entsprechende Selektoren. Schreiben Sie außerdem einen Konstruktor, der für jedes Attribut einen entsprechenden Parameter hat und die Attribute auf die übergebenen Werte setzt. Der Konstruktor soll erkennen, wenn die beiden Punkte der Geraden die gleichen Koordinaten haben. In diesem Fall soll der Nutzer per Kommandozeilen-Ausgabe über seine ungültige Eingabe informiert werden. Die Attribute werden dann auf `null` gesetzt.
Beachten Sie außerdem, dass es beim Erstellen einer Gerade zunächst keinen Unterschied macht, in welcher Reihenfolge die Punkte den Attributen zugewiesen werden. Daher wollen wir an dieser Stelle eine gewisse Normierung einführen und dafür sorgen, dass der erste Punkt stets links vom zweiten Punkt liegt. Liegen die Punkte genau untereinander, soll der erste Punkt stets unter dem zweiten liegen.

Implementieren Sie auch die Methode `toString()`, die einen String der Form `"Gerade durch (0,0) und (1,1)"` zurückgeben soll, wobei statt `(0,0)` und `(1,1)` die `toString()`-Repräsentation der beiden `Punkt`-Attribute stehen soll.

d) Ergänzen Sie die Klasse `Gerade` um die folgenden drei Hilfsmethoden. Auf diese darf von der Klasse selbst und von Unterklassen, nicht aber von außerhalb zugegriffen werden, was über den entsprechenden Zugriffsmodifikator erreicht werden kann. Die erste Methode, `boolean zwischenp1p2(Punkt p0)`, soll genau dann `true` zurückgeben, wenn `p0` auf der Geraden zwischen den Punkten `p1` und `p2` oder auf einem dieser beiden Punkte liegt. Die Methode `boolean vorp1(Punkt p0)` soll genau dann `true` zurückgeben, wenn `p0` so auf der Geraden liegt, dass der Abstand zu `p1` kleiner als zu `p2` ist und `p0` außerdem nicht zwischen `p1` und `p2` liegt. Die dritte Methode, `boolean hinterp2(Punkt p0)` soll genau dann `true` zurückgeben, wenn `p0` zwar auf der Geraden liegt, aber keine der ersten beiden Methoden beim Aufruf mit `p0` als Parameter `true` zurückgibt. Schreiben Sie anschließend die öffentliche Methode `boolean enthaelt(Punkt p0)`, die genau dann `true` zurückgeben soll, wenn `p0` auf der Geraden liegt.

Hinweise:

- Sie können sich in dieser Teilaufgabe den Sonderfall der Dreiecksungleichung[3] zunutze machen, dass eine Seite des Dreickes genau dann so lang ist wie die Summe der beiden anderen Seiten, wenn die beiden kürzeren Seiten auf der langen Seite liegen. In diesem Fall entspricht das Dreieck also einer Strecke.

e) Nutzen Sie die soeben implementierte Methode `enthaelt`, um die Klasse `Gerade` um die Methode `boolean equals(Object obj)` zu ergänzen. Beachten Sie, dass die in dieser Aufgabenstellung gewählte Repräsentation einer Geraden über zwei Punkte nicht eindeutig ist. So kann bspw. die x-Achse durch die Punkte $(0,0)$ und $(1,0)$, aber ebenso durch die Punkte $(1,0)$ und $(2,0)$ repräsentiert werden. Eine Überprüfung auf die Gleichheit der beiden Attribute reicht hier also offensichtlich nicht aus. Gestalten Sie Ihre Implementierung mit Blick auf die folgenden Aufgabenteile bereits so, dass alle durchgeführten Abfragen, bei denen dies Sinn ergibt, symmetrisch sowohl für `this` als auch für `obj` durchgeführt werden.

Hinweise:

- Nutzen Sie die Methode `obj.getClass()`, um die Klasse eines Objekts `obj` zu bestimmen. Der Rückgabewert der Methode ist vom Typ `Class`. Mit `obj1.getClass().equals(obj2.getClass())` können Sie prüfen, ob die Objekte `obj1` und `obj2` von exakt derselben Klasse sind. Wäre bspw. die Klasse von `obj1` Unterklasse der Klasse von `obj2`, würde dieser Ausdruck zu `false` auswerten.

f) Erstellen Sie die Klasse `Strahl` als Unterklasse von `Gerade`. Zusätzlich zu den beiden Punkten, die die Linie festlegen, muss `Strahl` ein Attribut haben, das codiert, ob der Strahl in `p1` oder in `p2` beginnt. Wegen der Normierung der Positionen ist es nämlich nicht möglich, bspw. immer `p1` als Anfangspunkt des Strahls festzulegen. Schreiben Sie einen Konstruktor, so dass `new Strahl(a,b)` einen Strahl durch die Punkte `a` und `b` mit Anfangspunkt `a` erzeugt. Dieser Konstruktor soll auf den Konstruktor der Klasse `Gerade` zurückgreifen. Sorgen Sie dafür, dass die Codierung des Startpunkts des Strahls innerhalb dieses Konstruktors korrekt initialisiert wird und danach nicht mehr änderbar ist. Schreiben Sie rückgreifend auf die Codierung des Startpunkts zwei Methoden `boolean startsFromp1()` und `boolean startsFromp2()`, die jeweils genau dann `true` zurückgeben, wenn der Strahl am entsprechenden Punkt beginnt.

Implementieren Sie auch die Methode `toString()` mit einer sinnvollen Ausgabe nach dem Vorbild der zu überschreibenden Methode der Oberklasse. Im zurückgegebenen String sollte deutlich werden, wo der Strahl beginnt und welchen Punkt der Strahl lediglich passiert.

[3] https://de.wikipedia.org/wiki/Dreiecksungleichung

g) Schreiben Sie in der Klasse `Strahl` eine Methode `verlaengern`, die diejenige Gerade zurückgibt, die entsteht, wenn man den Strahl über den Punkt, an dem der Strahl beginnt, ins Unendliche hinaus verlängert. Erläutern Sie im PDF-Teil der Abgabe, ob und ggf. warum es in Ihrer Implementierung möglich ist, die zurückgegebene Gerade nachträglich über die `Punkt`-Attribute des Strahls zu verändern.

Schreiben Sie auch in der Klasse `Strahl` die Methode `boolean enthaelt(Punkt p0)`, welche die `enthaelt`-Methode der Oberklasse überschreibt. Die Methode soll genau dann `true` zurückgeben, wenn `p0` auf dem Strahl liegt. Implementieren Sie dann unter Nutzung der zu überschreibenden Methode aus der Oberklasse außerdem die Methode `boolean equals(Objekt obj)`. Wieder soll genau dann `true` zurückgegeben werden, wenn beide `Strahl`-Objekte den gleichen Strahl repräsentieren.

h) Erstellen Sie die Klasse `Strecke` als Unterklasse von `Strahl`. Überlegen Sie für jede Methode der Oberklasse, ob eine Überschreibung notwendig ist. Wenn ja, schreiben Sie in der Klasse `Strecke` eine entsprechende Methode. Wenn nein, begründen Sie kurz im PDF-Teil der Abgabe, warum die Methode aus der Oberklasse weiterhin ausreicht. Schreiben Sie außerdem einen Konstruktor, der auf den Konstruktor der Klasse `Strahl` zurückgreift.

Schreiben Sie in der Klasse `Strecke` eine Methode `verlaengern(boolean swap)`, die denjenigen Strahl zurückgibt, der entsteht, wenn man die Strecke über einen der Endpunkte ins Unendliche hinaus verlängert. Die Verlängerung soll über `p2` hinaus vorgenommen werden, wenn der Parameter `swap` den Wert `true` hat und ansonsten über `p1` hinaus.

i) Welche Methoden werden in welcher Reihenfolge aufgerufen, wenn ein Aufruf `s1.equals(s2)` erfolgt, wobei `s1` und `s2` Strecken sind? Begründen Sie zudem, warum gerade diese Methoden aufgerufen werden und keine anderen. Stellen Sie die aufgerufenen Methoden zu dem Zeitpunkt dar, zu dem zum ersten Mal eine Methode der Klasse `BigDecimal` aufgerufen wird.

Beantworten Sie diese Fragen im PDF-Teil der Abgabe und beziehen Sie sich dabei auf die Zeilennummern der `.java`-Dateien Ihrer Abgabe. Es ist wichtig, dass deutlich wird, *welche* Methode aus *welcher* Klasse aufgerufen wird. Im Folgenden soll `Class.method` nicht bedeuten, dass `method` statisch ist, sondern dass `method` eine Methode der Klasse `Class` ist. Wenn also bspw. in Zeile 17 der Methode `Class.method` ein Aufruf von `BigDecimal.add` geschieht, und die Methode `Class.method` in Zeile 42 ganz zu Beginn von `Class.main` aufgerufen wird, könnte Ihre Antwort etwa so aussehen:

`Class.method` aus `Class.main:42`, einfacher Aufruf einer Methode derselben Klasse
`BigDecimal.add` aus `Class.method:17`, Aufruf einer öffentlichen Methode einer anderen Klasse

Aufgabe 9 (Deck 6): (Codescape)

Lösen Sie die Missionen von Deck 6 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Samstag, den 16.01.2021, um 23:59 Uhr abschicken.