Prof. Dr. J. Giesl
J.-C. Kassing, M. Leven Rosarius, N. Lommen, E. Meyer

Allgemeine Hinweise:

- Die **Deadline** zur **Abgabe** der Hausaufgaben ist am **Donnerstag, den 11.12.2025, um 14 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine*n Abgabepartner*in zu finden. Es darf **nur ein*e** Abgabepartner*in die Abgabe hochladen. Diese*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als `.zip`-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass unsere Hiwis ihn mit Feedback versehen können. Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von `.java`-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen dürfen nicht mit abgegeben werden.

  Stellen Sie sicher, dass Ihr Programm von `javac` **in der Version 25 akzeptiert** wird. Generell sollten alle Programme für alle Eingaben terminieren, solange in der Spezifikation (bzw. der Aufgabenstellung) nicht explizit etwas anderes verlangt wird!

Übungsaufgabe 1 (Überblickswissen):

a) Wie kann die Nutzung von Interfaces dabei helfen, die Entwicklung eines größeren Programms auf mehrere Entwickler*innen zu verteilen?

b) Welches Problem kann auftreten, wenn man zu viele `default`-Implementierungen in Interfaces nutzt? Und warum sind `default`-Implementierungen in Interfaces manchmal dennoch sinnvoll?

c) Sammeln Sie verschiedene "populäre" Java-Exceptions, z.B. in der Vorlesung eingeführte Exceptions oder sonstige bereits bekannte Exceptions, und diskutieren Sie, ggf. an Beispielen, wann und warum die jeweiligen Exceptions auftreten.

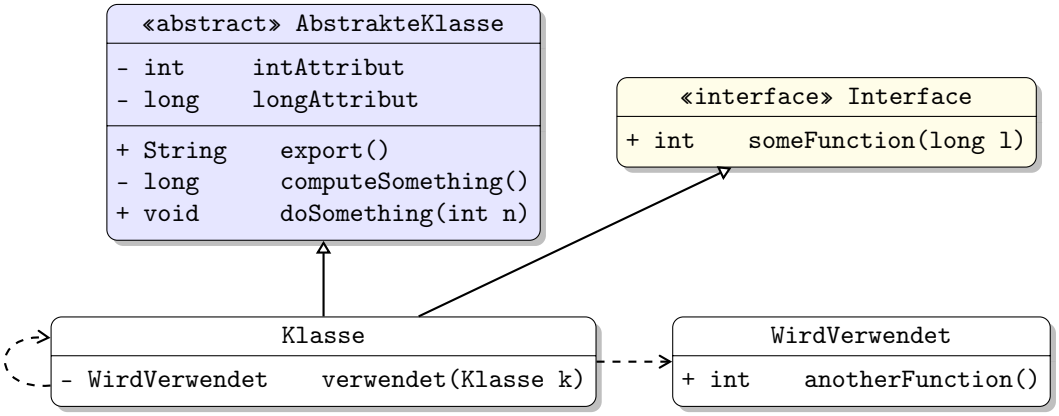
d) Wofür benötigt ein*e Java-Entwickler*in Module und Pakete?

Übungsaufgabe 2 (Entwurf einer Klassenhierarchie):

In dieser Aufgabe soll der Zusammenhang verschiedener Getränke zueinander in einer Klassenhierarchie modelliert werden. Dabei sollen folgende Fakten beachtet werden:

- Jedes Getränk hat ein bestimmtes Volumen.
- Wir wollen Apfelsaft und Kiwisaft betrachten. Apfelsaft kann klar oder trüb sein.
- Alle Saftarten können auch Fruchtfleisch enthalten.
- Wodka und Tequila sind zwei Spirituosen. Spirituosen haben einen bestimmten Alkoholgehalt.
- Wodka wird häufig aromatisiert hergestellt. Der Name dieses Aromas soll gespeichert werden können.
- Tequila gibt es als silbernen und als goldenen Tequila.
- Ein Mischgetränk ist ein Getränk, das aus verschiedenen anderen Getränken besteht.
- Mischgetränke und Säfte kann man schütteln, damit die Einzelteile (bzw. das Fruchtfleisch) sich gleichmäßig verteilen. Sie sollen daher eine Methode `schuetteln()` ohne Rückgabe zur Verfügung stellen.
- In unserer Modellierung gibt es keine weiteren Getränke.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die Getränke. Notieren Sie keine Konstruktoren, Getter und Setter. Sie müssen nicht markieren, ob Klassen `sealed` oder ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden und alle Klassen, bei denen dies sinnvoll ist, als `abstract` markiert werden. Benennen Sie Klassen und Methoden passend zu ihrer Funktion.
Verwenden Sie hierbei die folgende Notation:



Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Konkrete Methoden, die eine abstrakte Methode überschreiben, müssen nicht angegeben werden. Weiterhin bedeutet der Pfeil $B \longrightarrow\!\!\triangleright\ A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist) und A `-->` B, dass A den Typ B verwendet (z.B. als Typ eines Attributs oder in der Signatur einer Methode). Benutzen sie `+` und `-` um `public` und `private` abzukürzen.
Tragen Sie keine vordefinierten Klassen (`String`, etc.) oder Pfeile dorthin in Ihr Diagramm ein.

Hausaufgabe 3 (Entwurf einer Klassenhierarchie): (14 Punkte)

In dieser Aufgabe soll ein kleiner Ausschnitt aus der Welt der Vögel modelliert werden.

- Ein Vogel ist entweder ein Kiwi, ein Albatross oder ein Maronensperling. Es gibt keine weiteren Vögel in unserer Modellierung. Jeder Vogel hat einen bestimmten Namen und ein Nest. Die Vögel in unserem Modell sollen die Eier in ihren Nestern ausbrüten können. Dazu bieten sie die Methode `ausbrueten`, die ein Array von Vögeln zurückgibt.
- Kiwis sind das Nationaltier der Neuseeländer und Namensgeber der gleichnamigen Frucht. Jeder Kiwi bietet die Methode `laufen` ohne Rückgabe.
- Albatrosse haben die größte Flügelspannweite im Vogelreich. Um sehr lange Distanzen zurücklegen zu können, segelt der Albatross meistens. Dabei hilft ihm eine spezielle Sehne, welche die Flügel im Segelmodus fixiert. Es ist von Interesse, ob sich die Flügel zur Zeit im Segelmodus befinden.
- Maronensperlinge sind interessant, da sie manchmal die Nester anderer Vögel nutzen, um ihre Eier abzulegen. Dazu bieten sie die Methode `eierAuswechseln`, welche das Wirtsnest als Parameter erhält und keine Rückgabe hat.
- Sowohl Albatrosse als auch Maronensperlinge können fliegen und sollen daher eine Methode `fliegen` ohne Parameter und Rückgabe bieten. Leider sind nicht alle Vögel flugfähig. Ein Beispiel dafür ist der Kiwi.
- Ein Nest ist entweder eine Bruthöhle oder ein Bodengelege. Es gibt keine weiteren Nesttypen in unserer Modellierung. Jedes Nest hat ein Array von Eiern, zu welchem ein Ei mit der Methode `add` hinzugefügt und mithilfe von `remove` entfernt werden kann. Dazu erhalten beide Methoden jeweils ein Ei als Parameter und haben keine Rückgabe.
- Eine Bruthöhle hat eine Tiefe in cm.
- Für jedes Ei ist die Vogelmutter, welche das Ei gelegt hat, und die Farbe des Eis von Interesse. Ein Ei bietet die Methode `schluepfen`, welche den geschlüpften Vogel zurückgibt.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für den modellierten Ausschnitt aus der Vogelwelt. Notieren Sie keine Konstruktoren, und auch keine Getter oder Setter. Sie müssen nicht markieren, ob Klassen `sealed` oder ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden und alle Klassen, bei denen dies sinnvoll ist, als `abstract` markiert werden. Benennen Sie Klassen und Methoden passend zu ihrer Funktion. Verwenden Sie hierbei die Notation aus Aufgabe 2.

Übungsaufgabe 4 (Programmieren in Klassenhierarchien mit Interfaces, Sealed Classes und Exceptions):

In dieser Aufgabe soll es um Softwaretests gehen. Nehmen Sie folgende Situation an: bei einem Programmierwettbewerb soll ein Programm geschrieben werden, welches zwei nicht-negative Zahlen x und y miteinander multipliziert. Bei der Eingabe von negativen Zahlen soll eine `NegativeNumbersException` geworfen werden. Sie haben eine Vielzahl an Einreichungen, und möchten diese automatisch auf Korrektheit testen. Um das Entwickeln einer dafür geeigneten Software und die Rahmenbedingungen des Wettbewerbs soll es in dieser Aufgabe gehen. Die Wettbewerbsbeiträge haben die Form einer Klasse, die denen von Ihnen gegebenen Einschränkungen entspricht.
Im Folgenden werden wir teilweise von Abstraktion sprechen, und damit eine (eventuell abstrakte) Klasse, ein Interface oder ein Record meinen. Wählen Sie die jeweils geeignetste Variante. Ebenso sprechen wir nur von Vererbung auch dann, wenn die Implementierung eines Interfaces möglich wäre. Machen Sie sich auch über sinnvolle Modifikatoren (wie z.b. `public`, `private`, `protected`, `final`, `sealed`, `non-sealed`, etc.) Gedanken.

a) Erstellen Sie zunächst eine Abstraktion `Identifiable`, bei der jede Klasse, die von ihr erbt, die Methode `String getName()` bereitstellt. Diese wird von den eingereichten Programmen und von den Tests genutzt werden, um einen menschlich lesbaren Namen anzugeben.

b) Erstellen Sie eine neue `Exception`-Klasse `NegativeNumbersException`. Diese hat ein Attribut `isX` des Typs `boolean`, welches aussagt, ob der x oder der y-wert negativ war, und ein Attribut `value` des Typs `int`, welches den negativen Wert speichert. Beide Attribute sollen nach der Erstellung nicht mehr verändert werden können. Der Konstruktor dieser Klasse nimmt einen Parameter des Typs `boolean` und einen Parameter des Typs `int` entgegen und setzt die Werte der Attribute auf die Werte der übergebenen Parameter.

Implementieren Sie auch die Methode `toString()`, die einen String der Form `"Der x-Wert -5 ist negativ!"` zurückgeben soll, wobei statt `-5` die `toString()`-Repräsentation des `value`-Attributs stehen soll und statt x-Wert soll y-Wert geschrieben werden, wenn das Attribut `isX` den Wert `false` besitzt.

c) Jedes eingereichte Programm muss die Methode `int calculate(int x,int y)` implementieren, die das Ergebnis einer Multiplikation zweier Zahlen zurückliefern soll. Diese Multiplikation soll die in Aufgabe b) implementierte `NegativeNumbersException` werfen, wenn die Methode negative Parameter übergeben bekommt. Außerdem soll, wie bereits erwähnt, jedes Programm auch die Methoden von `Identifiable` bereitstellen.

Erstellen Sie eine Abstraktion `Program`, von dem jede Einreichung erben könnte, um diese Bedingungen zu erfüllen.

d) Wir wollen verschiedene Arten von Tests auf den Programmen laufen lassen. Schreiben Sie eine Abstraktion `Test`, von der alle zukünftigen Tests erben werden. Die Abstraktion `Test` stellt die Methode `TestResult runTest(Program p)` bereit. `TestResult` ist dabei eine weitere Abstraktion, die Sie erstellen sollen. `TestResult` beinhaltet dabei nur ein Boolesches Attribut `error` und ein Attribut `message` vom Typ `String`. Sobald ein `TestResult` erstellt worden ist, wird keine Änderung mehr an den Attributen vorgenommen, lediglich ein Zugriff auf die gespeicherten Werte ist nötig.

Weiterhin stellt `Test` die Methode von `Identifiable` bereit. Jeder `Test` hat auch ein `String`-Attribut `identifier`, das bei der Erstellung eines `Test` gesetzt werden muss und das als Standardrückgabewert von `getName()` dienen soll.

e) Wir möchten uns beim Behandeln von Tests auf zwei Arten von Tests beschränken: `PerformanceTest` und `FunctionalTest`. Andere Arten von Tests möchten wir ausschließen, modifizieren Sie `Test` entsprechend.

`PerformanceTest` ist eine konkrete Klasse, die bereits implementiert ist. In der `runTest`-Methode eines `PerformanceTest` wird lediglich die Zeit gemessen, die das übergebene Programm benötigt. Diese wird als `message`-Teil eines `TestResult`s zurückgegeben, der `error`-Wert ist bei einem `PerformanceTest` immer `false`. Die Eingabe, für die das Programm getestet wird, wird bei Erstellung des Tests festgelegt und danach nicht mehr geändert. Außerdem soll `PerformanceTest` auf eine eindeutige Weise durchgeführt werden, es soll keine Unterklassen geben. Es fehlen noch sinnvolle Modifikatoren, ergänzen Sie diese.

`FunctionalTest` ist lediglich als eine Überkategorie für die Tests gedacht, die die Programme auf Korrektheit überprüfen werden. Diese ist ebenfalls, bis auf Modifikatoren, bereits implementiert.

f) Nun geht es darum, konkrete Tests zu schreiben. Versuchen Sie, sinnvolle Kategorien von Eingaben zu finden, und für jede dieser Kategorien eine Klasse zu erstellen, die von `FunctionalTest` erbt. Jede dieser Testklassen soll dann in der `runTest`-Methode für mindestens eine Eingabe das Programm auswerten und das Ergebnis auf Richtigkeit überprüfen. Hierbei muss auch berücksichtigt werden, ob die korrekte Exception geworfen wurde bei einer falschen Eingabe. Beispielhafte Einreichungen für den Wettbewerb finden Sie in den Klassen `MultiplyX` mit $X \in \{1, \ldots, 5\}$.

Würde es bei dem Wettbewerb um das Halbieren einer Zahl gehen, wären bspw. gerade und ungerade Zahlen mögliche Eingabekategorien und die Test-Klasse, die für gerade Zahlen zuständig wäre, könnte das Programm mit der Eingabe 4 laufen lassen und überprüfen, ob 2 zurückgeliefert wird

Bei einem Fehler soll ein `TestResult` mit einem wahren `error`-Wert und einer aussagekräftigen Nachricht zurückgegeben werden, andernfalls mit einem unwahren `error`-Wert und beliebiger Nachricht.

g) Die Klasse `TestManager` ist vorgegeben, in deren `main`-Methode die Programm-Objekte erzeugt werden und in einem Array gespeichert werden. Danach wird ein Array von `Test`-Objekten erstellt. Momentan befinden sich in dem Array nur die Performance Tests, ergänzen Sie die von ihnen implementierten Tests.

Schließlich müssen Sie noch die markierten Zeilen in der Methode `runTests` der Klasse `TestManager` ergänzen und können dann die Programme testen lassen. Drei der fünf Programme sind fehlerhaft, finden Ihre Tests die problematischen Programme?

h) Aus der Vorlesung kennen Sie formale Methoden, wie den Hoare-Kalkül, um ein Programm zu verifizieren. Wie unterscheidet sich eine solche Herangehensweise von Tests?

Hausaufgabe 5 (Programmieren in Klassenhierarchien mit Interfaces, Sealed Classes und Exceptions): (3+3+3+6+4+2+4+3+6+2=36 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 Anfangspunkt eines Strahls bzw. die Endpunkte einer Strecke jeweils zur Linie dazugehören.

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[1] 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.

Hinweise:

- Wir wollen zwar durch die Nutzung von `BigDecimal`-Zahlen die Präzision erhöhen, sind jedoch bei falschen Ausgaben, die auf Rundungsfehler basieren, tolerant in dieser Aufgabe.

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 eine neue `Exception`-Klasse `SinglePointException`. Diese hat ein Attribut des Typs `Punkt`, das nach der Erstellung nicht mehr verändert werden kann. Der Konstruktor dieser Klasse nimmt einen Parameter des Typs `Punkt` entgegen und setzt den Wert seines einzigen Attributs auf den Wert des übergebenen Parameters.

Implementieren Sie auch die Methode `toString()`, die einen String der Form `"Doppelte Benutzung des Punktes (0,0)"` zurückgeben soll, wobei statt `(0,0)` die `toString()`-Repräsentation des `Punkt`-Attributs stehen soll.

d) Erstellen Sie nun die Klasse `Gerade`, die in den Attributen `p1` und `p2` die beiden Punkte enthalten soll, durch die die Gerade verläuft. Sorgen Sie dafür, dass nur die Klasse `Strahl`, welche Sie später

[1] https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/math/BigDecimal.html

implementieren, von der Klasse `Gerade` erben kann und keine weitere. Auch eine Gerade soll nicht mehr verändert werden können, wenn sie einmal 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 eine `SinglePointException` geworfen werden mit diesem Punkt.
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.

e) 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[a] zunutze machen, dass eine Seite des Dreiecks 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.

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

f) 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.

g) Erstellen Sie die Klasse `Strahl` als Unterklasse von `Gerade`. Sorgen Sie dafür, dass nur die Klasse `Strecke`, welche Sie später implementieren, von der Klasse `Strahl` erben kann und keine weitere. 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. Achten Sie darauf, korrekt mit möglicherweise auftretenden Exceptions umzugehen. 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.

h) 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. Da das Verlängern eines Strahls immer möglich ist, soll eine ggf. beim Erzeugen des `Gerade`-Objekts auftretende `SinglePointException` abgefangen werden. Um anzuzeigen, dass dieses Verhalten unerwartet ist, soll in diesem Fall ein `AssertionError` mit einer passenden Nachricht geworfen werden. Dieser muss nicht mittels `throw` in der Signatur gekennzeichnet werden.

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.

i) Erstellen Sie die Klasse `Strecke` als Unterklasse von `Strahl`. Sorgen Sie dafür, dass keine Klasse von der Klasse `Strecke` erben kann. 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. Achten Sie wieder darauf, korrekt mit möglicherweise auftretenden Exceptions umzugehen.

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.

j) Angenommen es gäbe ein Interface `bisInsUnendliche`, welches unendlich lange Linien darstellen soll. In unseren bisherigen Klassen gilt dies für die Klassen `Gerade` und `Strahl`, aber nicht für die Klasse `Strecke`. Lässt sich bei der momentanen Klassenstruktur das Interface nur für die vorgesehenen Klassen implementieren? Falls Ihre Antwort "ja" ist, dann ergänzen Sie Ihr Programm geeignet um ein neues Interface `bisInsUnendliche`, welches keine weiteren Attribute oder Methoden besitzt. Falls Ihre Antwort "nein" ist, dann geben Sie eine schriftliche Begründung dazu in Ihrem PDF-Teil der Abgabe ab.