Aufgabenblatt 08

Bearbeitungsende: 04.12.2022

Hinweis: Dieses Blatt ist ziemlich umfangreich, um Ihnen Gelegenheit zum Üben zu geben. Das heißt aber nicht, dass Sie alle Aufgaben bearbeiten sollten (oder eine Krise bekommen sollten, falls Sie das nicht schaffen).

Aufgabe 1 [Programmierung]

Schreiben Sie eine Klasse `Datum`, die ein Datum modelliert, also die Angabe eines bestimmten Tages durch das Jahr (als positive ganze Zahl), den Monat (als ganze Zahl im Bereich $[1; 12]$) und den Tag (als ganze Zahl im Bereich $[1; 31]$).

Diese 3 Werte sollten dazu als (nicht unbedingt einzige) Attribute verwaltet werden.

Die Klasse soll folgende Konstruktoren bereitstellen:

- Wird kein Argument übergeben, stellt das erzeugte Objekt den 1. Januar des Jahres 1 dar.
- Wird ein Wert für das Jahr übergeben, stellt das erzeugte Objekt den 1. Januar dieses Jahres dar.
- Werden zwei Werte für Jahr und Monat (in dieser Reihenfolge) übergeben, stellt das erzeugte Objekt den 1. Tag in diesem Monat in diesem Jahr dar.
- Werden Werte für Jahr, Monat und Tag (in dieser Reihenfolge) übergeben, stellt das erzeugte Objekt das entsprechende Datum dar.
- Wird eine `Datum`-Referenz übergeben, wird das Objekt als Kopie davon erzeugt.

Es kann jeweils angenommen werden, dass übergebene Werte gültig sind.

Die Klasse soll zudem folgende Methoden bereitstellen:

- `jahr`, `monat` und `tag` nehmen kein Argument an und geben den entsprechenden Wert zurück.
- `equals` nimmt eine `Datum`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob das durch die Instanz beschriebene Datum gleich ist zu dem durch das Argument beschriebenen Datum.
- `istFrueher` nimmt eine `Datum`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob das durch die Instanz beschriebene Datum ein früheres ist als das durch das Argument beschriebene Datum.
- `toString` gibt das Datum als Zeichenkette zurück, zunächst in der Form `"jahr-monat-tag"`.

  Dabei soll der Monats- oder Tageszahl, wenn sie einstellig ist, eine `0` vorangestellt werden, so dass die Zeichenkette immer die Länge 6 plus Stellen der Jahreszahl hat.

  Beispiel: `"2019-03-08"`

  Wenn aber eine der beiden unten genannten Methoden aufgerufen wird, erzeugt `toString` im Anschluss Darstellungen

  - mit einer anderen Reihenfolge von *jahr*, *monat* und *tag* oder/und
  - mit einem anderen Trennzeichen als `'-'`.

- `setFormatRF` nimmt eine Zeichenkette als Argument an und gibt kein Ergebnis zurück. Es kann angenommen werden, dass die Zeichenkette einen der drei Werte `"jmt"`, `"tmj"` oder `"mtj"` hat. Fortan wird die entsprechende Reihenfolge für die Anordnung von Jahres-, Monats- und Tagesangabe in allen Darstellungen von `Datum`-Objekten als Zeichenketten verwendet.

  Z.B. gibt nach dem Aufruf mit dem Argument `"tmj"` der Aufruf von `toString` für beliebige Objekte Zeichenketten der Form `"`*tag-monat-jahr*`"` zurück (ggf. mit einem anderen Trennzeichen).

- `setFormatTZ` nimmt ein Zeichen als Argument an und gibt kein Ergebnis zurück. Fortan wird dieses Zeichen als Trennzeichen zwischen Jahres-, Monats- und Tagesangabe in allen Darstellungen von `Datum`-Objekten als Zeichenkette verwendet.

  Z.B. gibt nach dem Aufruf mit dem Argument `'/'` der Aufruf von `toString` für beliebige Objekte Zeichenketten in der Form `"`*jahr/monat/tag*`"` zurück (ggf. in einer anderen Reihenfolge).

Hinweis: Speichern Sie die zu verwendende Reihenfolge und das zu verwendende Trennzeichen in jeweils einem Attribut.

Sofern möglich, sollten Methoden Klassenmethoden sein.

Schreiben Sie ein Testprogramm, in dem Sie mehrere `Datum`-Objekte erzeugen, vergleichen und unterschiedlich formatiert darstellen. Reichen Sie dieses Testprogramm jedoch nicht ein.

Aufgabe 2 [Programmierung]

Schreiben Sie eine Klasse `Rational`, die rationale Zahlen darstellt, also Brüche der Form $\frac{\text{Zähler}}{\text{Nenner}}$. Brüche werden, um sie eindeutig zu beschreiben und keine unnötig großen Werte zu verwenden, in *normalisierter* Form dargestellt:

- Zähler sind ganzzahlig, Nenner positiv ganzzahlig (nicht 0).
- Brüche sind *gekürzt*, d.h. der größte gemeinsame Teiler von Zähler und Nenner ist 1.

Verwenden Sie zur Darstellung von Brüchen Attribute für Zähler und Nenner.

Statten Sie die Klasse mit folgenden Konstruktoren aus:

- Wird kein Argument gegeben, soll der Bruch den Wert 0 darstellen.
- Für eine ganze Zahl als Argument soll der Bruch den Wert dieser Zahl darstellen.

  Beispiel: Für den Wert `4` soll der Bruch $\frac{4}{1}$ erzeugt werden.

- Für zwei ganze Zahlen – Dividend und Divisor – als Argumente soll der Bruch den Wert des Quotienten dieser Zahlen darstellen.

  Um den Bruch intern in normalisierter Form darzustellen, ist das Vorzeichen von Zähler und Nenner zu wechseln, falls letzterer negativ ist. Zudem sind Dividend und Divisor durch ihren größten gemeinsamen Teiler zu teilen („kürzen“). Verwenden Sie zur Berechnung des ggT die Methode `Mathe.ggT` aus B03A1.

  Gehen Sie davon aus, dass gültige Werte gegeben werden, dass also der Divisor nicht 0 ist.

  Beispiel: Für die Werte `28` als Dividend und `35` als Divisor soll der Bruch $\frac{4}{5}$ erzeugt werden, für die Werte `60` als Dividend und `-135` als Divisor der Bruch $\frac{-4}{9}$.

- Für eine `Rational`-Referenz als Argument werden deren Zähler und Nenner übernommen.

Stellen Sie in der Klasse die folgenden Methoden bereit:

- `zaehler` und `nenner` geben die entsprechenden Werte zurück..
- `equals` hat die üblichen Eigenschaften. Da die normalisierte Darstellung eindeutig ist, sind zwei Brüche nur dann gleich, wenn ihre normalisierten Darstellungen gleich sind.
- `istKleiner` nimmt eine `Rational`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob der von der Instanz dargestellte Wert kleiner ist als der von dem Argument dargestellte.

  Um zwei Brüche zu vergleichen, muss man sie zuerst auf einen gemeinsamen Nenner bringen.

  Beispiel: Für $\frac{5}{11}$ und $\frac{3}{4}$ liefert die Methode `true`, weil

  $$\frac{5}{11} = \frac{5 \cdot 4}{11 \cdot 4} = \frac{20}{44} < \frac{33}{44} = \frac{3 \cdot 11}{4 \cdot 11} = \frac{3}{4} \qquad \text{beziehungsweise} \qquad 5 \cdot 4 = 20 < 33 = 3 \cdot 11$$

- `abs` nimmt kein Argument an und gibt den Absolutbetrag der Instanz als neues `Rational`-Objekt zurück.
- `rez` nimmt kein Argument an und gibt den Kehrwert (Reziprokwert) der Instanz als neues `Rational`-Objekt zurück.

  Gehen Sie davon aus, dass gültige Werte gegeben werden, dass also der Divisor nicht 0 ist.

  Beispiel: Für den Bruch $\frac{-3}{7}$ ist der Kehrwert $\frac{-7}{3}$.
- `add`, `sub`, `mul` und `div` nehmen jeweils eine Referenz auf ein weiteres Objekt vom Typ `Rational` an. Sie geben ein neues `Rational`-Objekt zurück, das das Resultat der entsprechenden arithmetischen Operation zwischen der Instanz und dem anderen Objekt bildet. Gehen Sie für `div` davon aus, dass der Divisor nicht 0 ist.

  Z.B. soll `a.add(b)` den (normalisierten) Wert von `a` + `b` ergeben.

  Erinnerung: Grundrechenregeln für Brüche sind

  $$\begin{aligned} \frac{a}{b} + \frac{c}{d} &= \frac{a \cdot d + c \cdot b}{b \cdot d} \\ \frac{a}{b} - \frac{c}{d} &= \frac{a \cdot d - c \cdot b}{b \cdot d} \\ \frac{a}{b} \cdot \frac{c}{d} &= \frac{a \cdot c}{b \cdot d} \\ \frac{a}{b} / \frac{c}{d} &= \frac{a \cdot d}{b \cdot c} \end{aligned}$$

- `toString` gibt den modellierten Bruch als Zeichenkette zurück, in der Form `"`*`zaehler/nenner`*`"`

  Beispiel: Für den Bruch $\frac{3}{4}$ wird die Zeichenkette `"3/4"` geliefert.

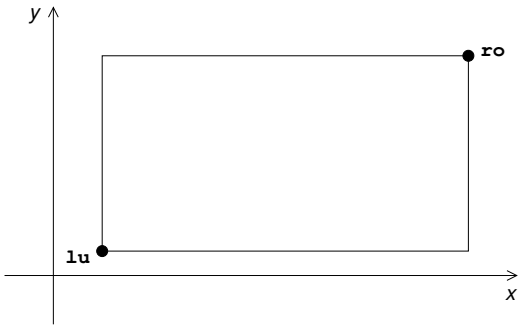
Schreiben Sie ein Testprogramm, in dem Sie mehrere `Rational`-Objekte erzeugen, mit diesen rechnen, und sie und die Ergebnisse ausgeben. Reichen Sie dieses Testprogramm jedoch nicht ein.

Einzureichen ist nur die Klasse `Rational`; die Klasse `Mathe` mit der Methode `ggT` wird zum Testen im Praktomat bereitgestellt.

Aufgabe 3 [Programmierung – nicht bewertet]

Schreiben Sie eine Klasse `RechteckXYPerEcke`, die achsparallele Rechtecke in der 2-dimensionalen Ebene modelliert (deren Seiten also parallel zu der x- und y-Achse des Koordinatensystems liegen).

Ein Rechteck soll intern durch seine linke untere Ecke und seine rechte obere Ecke beschrieben werden, die durch Objekte vom (in der Vorlesung vorgestellten) Typ `Punkt2D` dargestellt werden.



Statten Sie die Klasse mit folgenden Konstruktoren aus:

- Es werden die x-Koordinate des linken Randes, die des rechten Randes, die y-Koordinate des unteren Randes und die des oberen Randes in dieser Reihenfolge als Argumente übergeben.
- Es werden als Argumente zwei `Punkt2D`-Objekte übergeben, welche diagonal gegenüberliegende Eckpunkte des Rechtecks beschreiben.

  Es werden also entweder die linke untere und die rechte obere Ecke übergeben oder die linke obere und die rechte untere (oder jeweils umgekehrt).

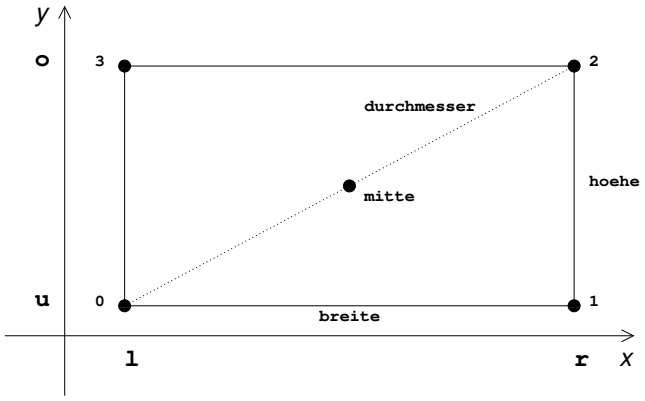
  Das Rechteck soll die Ecken als eigene Kopien verwalten, die von Änderungen der übergebenen Ecken unabhängig sind.
- Als Argument wird eine `Rechteck`-Referenz übergeben. Das Objekt wird als Kopie(!) der referenzierten `Rechteck`-Instanz angelegt.

Stellen Sie zudem folgende Methoden bereit:

- `equals` mit den üblichen Eigenschaften – tolerieren Sie dabei Rundungsfehler
- `breite` und `hoehe` nehmen keine Argumente an und geben die Ausdehnung des Rechtecks in x- bzw. y-Richtung zurück.
- `ecke` nimmt zwei Wahrheitswerte `istRechts` und `istOben` an. Je nach Wert der Argumente gibt sie die linke untere, linke obere, rechte untere oder rechte obere Ecke des Rechtecks als `Punkt2D`-Objekt zurück.
- `ecke` (überladen) betrachtet die Ecken als durchnummeriert: Die linke untere Ecke hat die Nummer 0, dann geht es gegen den Uhrzeigersinn weiter. Die rechte untere Ecke hat also die Nummer 1.

  Die Methode nimmt eine nichtnegative Zahl im Bereich $[0; 3]$ als Nummer einer Ecke an und gibt die entsprechende Ecke als `Punkt2D`-Objekt zurück.
- `mitte` gibt die Mitte des Rechtecks als `Punkt2D`-Objekt zurück.

- `durchmesser`, `umfang` und `flaeche` nehmen keine Argumente an und geben den (größten, also diagonalen) Durchmesser, den Umfang bzw. die Fläche des Rechtecks zurück.



- `verschiebe` nimmt einen x-Wert und einen y-Wert als Gleitkommawerte an und verschiebt das Rechteck um diese Werte. Sie gibt eine Referenz auf die `RechteckXYPerEcke`-Instanz zurück.
- `toString` nimmt keine Argumente an und gibt eine Darstellung des Rechtecks als Zeichenkette der Form `"[`*`lup`*`,`*`rop`*`]"` zurück, worin *`lup`* für die Darstellung des linken unteren Eckpunkts als Zeichenkette steht und *`rop`* für die des rechten oberen.

Schreiben Sie ein Testprogramm, in dem Sie mehrere `RechteckXYPerEcke`-Objekte erzeugen, ihre Abmessungen abfragen, und sie und die Ergebnisse ausgeben. Reichen Sie dieses Testprogramm jedoch nicht ein.

Einzureichen ist nur die Klasse `RechteckXYPerEcke`.

Aufgabe 4 [Programmierung – nicht bewertet]

Schreiben Sie eine Klasse `Uhrzeit`, die eine Uhrzeit modelliert, also die Angabe eines bestimmten Zeitpunktes an einem Tag durch die Stunde (als ganze Zahl im Bereich $[0; 23]$) und die Minute (als ganze Zahl im Bereich $[0; 59]$).

Diese 2 Werte sollten dazu als Attribute verwaltet werden.

Die Klasse soll folgende Konstruktoren bereitstellen:

- Wird kein Argument übergeben, stellt das erzeugte Objekt 0:00 Uhr dar.
- Wird nur ein Wert `s` für die Stunde übergeben, stellt das erzeugte Objekt `s`:00 Uhr dar.
- Werden Werte für Stunde und Minute (in dieser Reihenfolge) übergeben, stellt das erzeugte Objekt die entsprechende Uhrzeit dar.
- Wird eine `Uhrzeit`-Referenz übergeben, wird das Objekt als Kopie davon erzeugt.

Es kann angenommen werden, dass übergebene Werte gültig sind.

Die Klasse soll zudem folgende Methoden bereitstellen:

- `stunde` und `minute` nehmen kein Argument an und geben den entsprechenden Wert zurück.

- `equals` nimmt eine `Uhrzeit`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob die durch die Instanz beschriebene Uhrzeit gleich ist zu der durch das Argument beschriebenen Uhrzeit.
- `istFrueher` nimmt eine `Uhrzeit`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob die durch die Instanz beschriebene Uhrzeit eine frühere ist als die durch das Argument beschriebene Uhrzeit.
- `toString` gibt die Uhrzeit als Zeichenkette zurück, zunächst in der Form "*`stunde`*`:`*`minute`*".

  Dabei soll der Stunden- oder Minutenzahl, wenn sie einstellig ist, eine `0` vorangestellt werden, so dass die Zeichenkette immer die Länge 5 hat.

  Beispiel: `"09:05"`
- `set12h24h` nimmt keine Argumente an und gibt kein Ergebnis zurück. Nach Aufruf dieser Methode erzeugen Aufrufe von `toString` Darstellungen der Uhrzeit im jeweils anderen Modus:
  - (Ausgangsmodus:) 24-Stunden-Modus: Form der Zeichenkette ist "*`stunde`*`:`*`minute`*", wobei *`stunde`* von 0 bis 23 läuft.
  - 12-Stunden-Modus: Form der Zeichenkette ist "*`stunde`*`:`*`minute apm`*", wobei *`stunde`* von 1 bis 12 läuft und *`apm`* entweder für `"a.m."` oder für `"p.m."` steht.

    `"a.m."` wird für Vormittagszeiten verwendet, von 00:00 bis 11:59 Uhr in 24-Stunden-Rechnung.

    `"p.m."` wird für Nachmittagszeiten verwendet, von 12:00 bis 23:59 Uhr in 24-Stunden-Rechnung.

    Achtung: Die 12. Stunde liegt somit bereits in der anderen Tageshälfte:

    Eine 12-Stunden-Uhrzeit `12:XY a.m.` entspricht der 24-Stunden-Uhrzeit 00:XY, liegt also in der Stunde nach Mitternacht.

    Eine 12-Stunden-Uhrzeit `12:XY p.m.` entspricht der 24-Stunden-Uhrzeit 12:XY, liegt also in der Stunde nach Mittag.

  Hinweise: Verwenden Sie ein Attribut `is12h`, das angibt, ob der 12h- oder der 24-Modus zu verwenden ist. Verwenden Sie zur Umrechnung der Stunden auf 12h den Modulo-Operator, desweiteren Fallunterscheidungen.

Sofern möglich, sollten Methoden Klassenmethoden sein.

Lösungen zu mit [**Programmierung**] markierten Aufgaben sind im **Praktomat** einzureichen.

Lösungen zu mit [**Programmierung – nicht bewertet**] markierten Aufgaben können ebenfalls im **Praktomat** eingereicht werden, werden jedoch nicht bewertet.

Allgemeine **Fragen** zu den Aufgaben können Sie im **LEA-Forum „Übungsaufgaben“** stellen.

Hilfe bei der Lösung der Aufgaben erhalten Sie in den **Übungen** und in der **Studierwerkstatt** .