

Aufgabe T5.1: Code verstehen (1+1=2 Punkte)

Sie finden online einen fertigen Programmcode, der wohl etwas hastig geschrieben wurde. Die sehr ineffizient gestaltete Funktion wurde wie folgt implementiert:

```
checkerFunc :: Float -> Float -> Bool
checkerFunc n m = if (n < m) then
    if (n * m) > 42
        then True
    else if (n * m) == 42
        then True
    else
        False
else if (n == m) then
    if (n * m) > 42
        then True
    else if (n * m) == 42
        then True
    else
        False
else
    if (n * m) < 42
        then False
    else if (n * m) == 42
        then False
    else
        False
```

- (a) Fassen Sie zusammen, welche Eigenschaften von n und m erfüllt sein müssen, damit die Funktion True zurückgibt.

$$(n \leq m) \wedge (n \cdot m \geq 42)$$

- (b) Die Funktion lässt sich in einem einzigen booleschen Ausdruck darstellen. Geben Sie die entsprechende Funktion als Haskellcode (schriftlich) an.

```
1 checkerFunc :: Float -> Float -> Bool
2 checkerFunc n m = (n <= m) && (n*m >= 42)
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

Aufgabe T5.2: Abgeleitete Klassen (2 Punkte)

In der Vorlesung haben sie Standard-Typklassen wie Ord, Enum oder Num kennengelernt. Im folgenden Programmcode wird ein neuer Datentyp Season definiert:

```
data Season = Spring | Summer | Autumn | Winter

-- Testing if a season is cold
isCold :: Season -> Bool
isCold s
    | s == Winter = True

    | otherwise = False

-- Is there sill time until it is autumn?
beforeAutumn :: Season -> Bool
beforeAutumn s
    | s < Autumn = True
    | otherwise = False

-- Print first season
printFirstSeason :: Season
printFirstSeason = toEnum 0
```

Ausgehend von den gegebenen Funktionen, überprüfen Sie, ob für die neue Klasse alle notwendigen Standard-Typklassen in Form einer abgeleiteten Instanzdeklaration angegeben worden sind. Falls nicht, fügen Sie diese noch hinzu und begründen Sie Ihre Antwort.

Hinweis: Verwenden Sie nur die in aus der Vorlesung bekannten Standard-Typklassen.

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1 {-
2   Zum Vergleich des Argumentes s in isCold ist eine Ableitung der Klasse Eq notwendig.
3   Durch das Ableiten aus der Klasse Ord werden die Jahreszeiten in eine totale
4   Ordnung gebracht und die Operationen <, >, <=, >= für die Funktion beforeAutmn ermöglicht.
5   Durch das Ableiten aus der Klasse Enum werden die Jahreszeiten den Integern von 0-3
6   ↪ zugeordnet.
7   Das Ableiten der Klasse Show ermöglicht die Ausgabe von Spring bzw Enum 0 in printFirstSeason
8   in der Konsole.
9 -}
10
11 data Season = Spring | Summer | Autumn | Winter deriving (Eq, Ord, Enum, Show)
12
13 -- Testing if a season is cold
14 isCold :: Season -> Bool
15 isCold s
16     | s == Winter = True
17     | otherwise = False
18 -- Is there sill time until it is autumn?
19 beforeAutumn :: Season -> Bool
20 beforeAutumn s
21     | s < Autumn = True
22     | otherwise = False
23 -- Print first season
24 printFirstSeason :: Season
25 printFirstSeason = toEnum 0
```

Aufgabe P5.3: Klassen (1 + 2 + 5 = 8 Punkte)

In der Vorlesung wurden die Datentypen Point und Vector umgesetzt. Nun soll eine Klasse Polygon implementiert werden. Dabei ist ein Polygon eine Menge von Punkten. Die Klasse Polygon definiert die Funktionen:

```
area      :: Polygon p => p -> Float
translate_poly :: Polygon p => p -> Vector -> p
scale_poly  :: Polygon p => p -> Float -> p
```

Dabei nimmt die Funktion area ein Polygon entgegen und gibt die Fläche des Polygons zurück. Die Funktion translate verschiebt jeden Punkt eines Polygons. Die Verschiebung wird durch einen Vektor angegeben. Die Funktion gibt danach das verschobene Polygon zurück. Die Funktion scale multipliziert jeden Punkt eines Polygons mit einer Gleitkommazahl s und gibt das skalierte Polygon zurück. Die Ergebnisse aller Funktionen sollen in der Konsole angezeigt werden können. Bearbeiten Sie nun folgende Aufgaben:

- (a) Implementieren Sie die Datentypen Point und Vector, die Sie in der Vorlesung kennengelernt haben. Dabei sollen die beiden Datentypen keine alternative Schreibweise für Tupel sein, sondern als eigene Datentypen definiert werden. Implementieren Sie die Funktionen translate und scale mit folgender Signatur:

translate : : Point \rightarrow Vector \rightarrow Point

scale : : Point \rightarrow Float \rightarrow Point

Dabei verschiebt die Funktion translate einen Punkt um einen Vektor und die Funktion scale multipliziert einen Punkt komponentenweise mit einer Gleitkommazahl.

- (b) Definieren Sie die Klasse Polygon und die Datentypen Triangle (allgemeines Dreieck) und Quad (allgemeines Viereck).
- (c) Implementieren Sie die Datentypen Triangle und Quad als Instanzen von der Klasse Polygon und implementieren Sie dabei die von der Klasse Polygon definierten Funktionen.

Hinweis: Falls notwendig, recherchieren Sie die Formeln für die benötigten Flächenberechnungen.

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
2 {-# HLINT ignore "Use guards" #-}
3 {-# LANGUAGE ConstrainedClassMethods #-}
4 {-
5 Aufgabe 5_3 a
6 -}
7 -- Es werden die Datentypen Point und Vektor erstellt.
8 data Point = MkPoint {
9     px :: Float,
10    py :: Float
11 } deriving (Eq, Ord, Show)
12
13 data Vector = MkVector {
14     vx :: Float,
15     vy :: Float
16 } deriving (Eq, Ord, Show)
17 -- Die Funktion addiert einen Vektor mit einem Punkt und gibt einen neuen Punkt aus
18 --Beispiel translate x1 x2 ergibt (3,3)
19 translate :: Point -> Vector -> Point
20 translate (MkPoint p1 p2) (MkVector v1 v2) = MkPoint (p1+v1) (p2+v2)
21 --Die Funktion multipliziert einen Punkt mit einem Float und skaliert den Punkt
22   ↳ dementsprechend.
23 --Beispiel scale 2 x1 ergibt (4,4) da 2+2 gleich 4 ist
24 scale :: Float -> Point -> Point
25 scale factor (MkPoint p1 p2) = MkPoint (factor * p1) (factor * p2)
26
27 {-
28 Aufgabe 5_3 b
29 -}
30 class Polygon p where
31     area :: Polygon p => p -> Float
32     translate_poly :: Polygon p => p -> Vector -> p
33     scale_poly :: Polygon p => p -> Float -> p
34
35 data Triangle = MkTriangle {
36     p1 :: Point,
37     p2 :: Point,
38     p3 :: Point
39 } deriving Show
40
41 data Quad = MkQuad {
42     q1 :: Point,
43     q2 :: Point,
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
44     q3 :: Point,
45     q4 :: Point
46 } deriving Show
47
48 {-
49 Aufgabe 5_3 c
50 -}
51
52 {-
53 Funktionen zum Berechnen von Dreiecken
54 -}
55
56 {-
57 Die Funktion berechnet die Distanz zwischen zwei Punkten und gibt die Länge zurück.
58 Beispiel lengthBetweenTwoPoints x1 x2 ergibt 6.0
59 -}
60 lengthBetweenTwoPoints :: Point -> Point -> Float
61 lengthBetweenTwoPoints (MkPoint x1 y1) (MkPoint x2 y2) =
62     sqrt((x1-x2)^2 + (y1-y2)^2)
63
64 {-
65 Die Funktion gibt die längste Seite eines allgemeinen Dreiecks zurück.
66 Beispiel findGround x1 x2 x3 ergibt 6.0.
67 -}
68 findGround :: Point -> Point -> Point -> Float
69 findGround p1 p2 p3
70     | (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p1 p3) || (lengthBetweenTwoPoints
71     ↪ p1 p2 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p2
72     | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints
73     ↪ p1 p3 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p3
74     | otherwise = lengthBetweenTwoPoints p2 p3
75
76 {-
77 Die Funktion berechnet die Summe der Kantenlängen und halbiert diese.
78 Beispiel calculateHalfOfSumEdge x1 x2 x3 ergibt 7.4966145.
79 -}
80
81 calculateHalfOfSumEdge :: Point -> Point -> Point -> Float
82 calculateHalfOfSumEdge p1 p2 p3 = 0.5 * (lengthBetweenTwoPoints p1 p2 + lengthBetweenTwoPoints
83     ↪ p1 p3 + lengthBetweenTwoPoints p2 p3)
84
85 {-
86 Die Funktion berechnet die Höhe eines allgemeinen Dreiecks ausgehend von der längsten Seite.
87 Beispiel calculateHeight x1 x2 x3 ergibt 2.9999995
88 -}
89
90 calculateHeight :: Point -> Point -> Point -> Float
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
86 calculateHeight p1 p2 p3 = (2/findGround p1 p2 p3)*sqrt(calculateHalfOfSumEdge p1 p2
  ↳ p3*(calculateHalfOfSumEdge p1 p2 p3-lengthBetweenTwoPoints p1 p2)*(calculateHalfOfSumEdge
  ↳ p1 p2 p3-lengthBetweenTwoPoints p1 p3)*(calculateHalfOfSumEdge p1 p2
  ↳ p3-lengthBetweenTwoPoints p2 p3))
87
88 {-
89 Funktionen zum Berechnen von Quadraten
90 -}
91
92 {-
93 Hilfskonstruktion, weil wir offensichtlich die Aufgabe falsch verstehen.
94 -}
95 data Line = MkLine {
96     l1 :: Point,
97     l2 :: Point
98 } deriving (Eq, Show)
99
100 {-
101 Die Funktion gibt die beiden Punkte eines Dreiecks zurück,
102 die am weitestens auseinander liegen.
103 (analog zu findGround nur mit Punkten)
104 Beispiel findLongestDistance x1 x2 x3 gibt aus MkLine {l1 = MkPoint {px = 2.0, py = 2.0}, l2 =
  ↳ MkPoint {px = 8.0, py = 2.0}}. Da die Linie zwischen den beiden entferntesten Punkten
  ↳ ausgegeben werden soll
105 -}
106 findLongestDistance :: Point -> Point -> Point -> Line
107 findLongestDistance p1 p2 p3
108     | (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p1 p3) || (lengthBetweenTwoPoints
  ↳ p1 p2 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p2
109     | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints
  ↳ p1 p3 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p3
110     | otherwise = MkLine p2 p3
111
112 {-
113 Die Funktion gibt eine beliebige der beiden Diagonalen in einem
114 allgemeinen Viereck zurück.
115 Beispiel findDiagonal q ergibt MkLine {l1 = MkPoint {px = 2.0, py = 2.0}, l2 = MkPoint {px =
  ↳ 8.0, py = 2.0}}, da sie eine Diagonale des Quadrates als Linie zurückgeben soll.
116 -}
117 findDiagonal :: Quad -> Line
118 findDiagonal quad = findLongestDistance (q1 quad) (q2 quad) (q3 quad)
119
120 {-
121 Die Funktion gibt die andere Diagonale zurück.
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
122 Beispiel findOtherPoints q gibt MkLine {l1 = MkPoint {px = 7.0, py = 5.0}, l2 = MkPoint {px =
    ↪ 3.0, py = 1.0}} aus. Da diese Linie die zweite Diagonale des Quadrates q ist. und die
    ↪ erste schon in der Funktion findDiagonal ausgegeben wird.
123 -}
124 findOtherPoints :: Quad -> Line
125 findOtherPoints quad
126     | findDiagonal quad == MkLine (q1 quad) (q2 quad) = MkLine(q3 quad) (q4 quad)
127     | findDiagonal quad == MkLine (q1 quad) (q3 quad) = MkLine(q2 quad) (q4 quad)
128     | findDiagonal quad == MkLine (q2 quad) (q3 quad) = MkLine(q1 quad) (q4 quad)
129
130 {-
131 Die Funktion gibt den 1. Punkt einer Linie zurück.
132 Beispiel breakDownLineToFirst line1 gibt MkPoint {px = 2.0, py = 2.0} also x1 aus.
133 -}
134 breakDownLineToFirst :: Line -> Point
135 breakDownLineToFirst line = l1 line
136
137 {-
138 Die Funktion gibt den 2. Punkt einer Linie zurück.
139 Beispiel breakDownLineToSecond line1 gibt MkPoint {px = 8.0, py = 2.0} also x2 aus.
140 -}
141 breakDownLineToSecond :: Line -> Point
142 breakDownLineToSecond line = l2 line
143
144 {-
145 Die Funktionen area, translate_poly und scale_poly wurden implementiert.
146 Implementierungen der Aufgabenstellungen.
147 Beispiele area t1 gibt ca. 9 aus
148     translate_poly t1 v1 gibt MkTriangle {p1 = MkPoint {px = 3.0, py = 3.0}, p2 =
    ↪ MkPoint {px = 9.0, py = 3.0}, p3 = MkPoint {px = 8.0, py = 6.0}} als neues Dreieck aus.
149     scale_poly t1 2 gibt MkTriangle {p1 = MkPoint {px = 4.0, py = 4.0}, p2 = MkPoint {px =
    ↪ 16.0, py = 4.0}, p3 = MkPoint {px = 14.0, py = 10.0}} aus. Die Größe des Dreieck wurde
    ↪ verdoppelt
150     area q ist ca. 12
151     translate_poly q v1 gibt MkQuad {q1 = MkPoint {px = 3.0, py = 3.0}, q2 = MkPoint {px =
    ↪ 9.0, py = 3.0}, q3 = MkPoint {px = 8.0, py = 6.0}, q4 = MkPoint {px = 8.0, py = 6.0}}
    ↪ als neues Quadrat aus.
152     scale_poly q 2 gibt MkQuad {q1 = MkPoint {px = 4.0, py = 4.0}, q2 = MkPoint {px =
    ↪ 16.0, py = 4.0}, q3 = MkPoint {px = 14.0, py = 10.0}, q4 = MkPoint {px = 6.0, py = 2.0}}
    ↪ aus. Die größe des Quadrats wurde verdoppelt.
153 -}
154 instance Polygon Triangle where
155     area (MkTriangle p1 p2 p3) = 0.5* findGround p1 p2 p3 * calculateHeight p1 p2 p3
156     translate_poly (MkTriangle p1 p2 p3) v = MkTriangle (translate p1 v) (translate p2 v)
    ↪ (translate p3 v)
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
157     scale_poly (MkTriangle p1 p2 p3) f = MkTriangle (scale f p1) (scale f p2) (scale f p3)
158 instance Polygon Quad where
159     area quad = area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
        ↪ (breakDownLineToSecond (findDiagonal quad))(breakDownLineToFirst (findOtherPoints
        ↪ quad))) + area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
        ↪ (breakDownLineToSecond (findDiagonal quad))(breakDownLineToSecond (findOtherPoints
        ↪ quad)))
160     translate_poly quad v = MkQuad (translate (q1 quad) v) (translate (q2 quad) v) (translate
        ↪ (q3 quad) v) (translate (q3 quad) v)
161     scale_poly quad f = MkQuad (scale f (q1 quad)) (scale f (q2 quad)) (scale f (q3 quad))
        ↪ (scale f (q4 quad))
162
163 {-
164   Konstanten zum Testen der Funktionen
165 -}
166 x1 :: Point
167 x1 = MkPoint 2 2
168
169 v1 :: Vector
170 v1 = MkVector 1 1
171
172 x2 :: Point
173 x2 = MkPoint 8 2
174
175 x3 :: Point
176 x3 = MkPoint 7 5
177
178 x4 :: Point
179 x4 = MkPoint 3 1
180
181 t1 :: Triangle
182 t1 = MkTriangle x1 x2 x3
183
184 t2 :: Triangle
185 t2 = MkTriangle x1 x2 x4
186
187 q :: Quad
188 q = MkQuad x1 x2 x3 x4
189
190 line1 :: Line
191 line1 = MkLine x1 x2
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

Aufgabe P5.4: Aufzählung (1 + 2 + 3 = 6 Punkte)

Es soll eine Klasse Card für Spielkarten implementiert werden. Eine Spielkarte besteht aus einem Rank (Wert) {Seven, Eight, Nine, Ten, Jack, Queen, King, Ace} in aufsteigender Reihenfolge und einem Suit (Farbe) {Diamond, Heart, Spade, Club }.

- (a) Implementieren Sie die Datentypen Rank, Suit und Card.
- (b) Implementieren Sie Card als eine Instanz der Klasse Ord. Dabei entscheidet der Wert (Rank) welche Karte größer ist, bei zwei Karten gleichen Wertes entscheidet die Farbe (Suit), welche Karte größer ist. Bei gleichem Wert und gleicher Farbe sind beide Karten gleich groß.
- (c) Implementieren Sie den Datentypen Hand, der aus drei Karten besteht. Implementieren Sie anschließend eine Funktion `value :: Hand → Integer`, welchen den Wert der Hand zurückgibt. Dabei seien folgende Handkombinationen definiert:

Hand	Beispiel	Value
nichts	$7\clubsuit B\heartsuit 10\spadesuit$	0
Paar (2 Karten selben Wertes)	$8\heartsuit D\spadesuit 8\clubsuit$	1
Drilling (3 Karten selben Wertes)	$B\clubsuit B\heartsuit B\spadesuit$	2
Flush (3 Karten selber Farbe)	$K\heartsuit 9\heartsuit 7\heartsuit$	3

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1  --Erstellt wird der Datentyp Rank der aus den Zahlen von 7 bis Ass eines Kartenspiels
   besteht.
2  data Rank = Seven | Eight | Nine | Ten | Jack | Queen | King | Ace deriving (Show, Eq, Ord)
3  --Erstellt wird der Datentyp Suit der aus den Farben/Typen eines Kartenspiels besteht. Also
   ↳ Herz,Karo,Piek,Kreuz.
4  data Suit = Diamond | Heart | Spade | Club deriving (Show, Eq, Ord)
5  --Erstellt wird der Datentyp Card der jeweils aus einer Zahl und einer Farbe/Typ besteht.
6  data Card = MkCard {
7      rank :: Rank,
8      suit :: Suit
9  }
10  deriving (Show, Eq)
11  -- Card wird als Instanz der Klasse Ord erstellt. Nun können 2 Karten verglichen werden welche
   ↳ einen höheren Wert hat. Der Wert Rank entscheidet welche Karte größer ist, bei zwei Karten
   ↳ gleichen Wertes entscheidet die Farbe/Typ welche Karte größer ist. Bei gleichem Wert und
   ↳ gleicher Farbe sind sie gleich groß-
12  --Beispiel card1 <= card2 ergibt False da King größer als die 9 ist
13  --Beispiel card3 <= card2 ergibt True da die 7 kleiner als die 9 ist.
14  instance Ord Card where
15      (MkCard rank1 suit1) <= (MkCard rank2 suit2) = if rank1 == rank2 then suit1 <= suit2 else
   ↳ rank1 <= rank2
16
17  -- Es wird der Datentyp Hand implementiert, eine Hand besteht aus 3 Karten.
18  data Hand = MkHand Card Card Card
19  --Die Funktion Value nimm eine Hand und gibt ihr einen Wert von 0 bis 3. Falls Die Farben/Typ
   ↳ der 3 Karten gleich ist erhält sie den Value Wert 3. Falls der Rank der drei Karten gleich
   ↳ ist, sie also ein Drilling sind erhält die Hand den Value Wert 2. Falls zwei von den drei
   ↳ Karten denselben Rank haben also ein Paar vorhanden ist, erhält sie den Value Wert 1.
   ↳ Falls nichts von dem oben genannten der Fall ist erhält sie den Value Wert 0.
20  --Beispiel value myHand ergibt 0, da Die Farben der drei Karten nicht gleich sind und weder
   ↳ ein Drilling noch ein Paar vorhanden ist.
21  --Beispiel value myHand1 ergibt 3, da alle 3 Farben der drei Karten gleich sind
22  value :: Hand -> Integer
23  value (MkHand (MkCard rank1 suit1) (MkCard rank2 suit2) (MkCard rank3 suit3))
24      | suit1 == suit2 && suit1 == suit3 && suit2 == suit3 = 3
25      | rank1 == rank2 && rank1 == rank3 && rank2 == rank3 = 2
26      | rank1 == rank2 || rank1 == rank3 || rank2 == rank3 = 1
27      | otherwise = 0
28  -- Erstellt wird eine Hand die aus drei Karten besteht
29  myHand :: Hand
30  myHand = MkHand card1 card2 card3
31
32  card1 :: Card
33  card1 = MkCard King Club
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
34
35 card2 :: Card
36 card2 = MkCard Nine Heart
37
38 card3 :: Card
39 card3 = MkCard Seven Heart
40 -- Erstellt wird eine Hand die aus drei Karten besteht
41 myHand1 :: Hand
42 myHand1 = MkHand aceclub kingclub queenclub
43
44 aceclub :: Card
45 aceclub = MkCard Ace Club
46
47 kingclub :: Card
48 kingclub = MkCard King Club
49
50 queenclub :: Card
51 queenclub = MkCard Queen Club
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

Aufgabe P5.5: Eigene Datentypen (1 + 2 + 2 + 2 = 7 Punkte)

In dieser Aufgabe sollen Sie einen Datentyp Currency erstellen. Dabei sei eine Currency entweder Euro, Dollar oder Yen. Der Wechselkurs zwischen den verschiedenen Currencies sei dabei:

1 Dollar = 0,90 Euro

1 Yen = 0,0083 Euro

1 Dollar = 108,59 Yen

Bearbeiten Sie hierfür folgende Schritte:

- (a) Implementieren Sie den Datentyp Currency, der entweder Euro, Dollar oder Yen repräsentiert. Dabei bestehen Euro und Dollar aus zwei Integern für Euro und Cents bzw. Dollar und Cents und Yen besteht nur aus einem Integer.
- (b) Leiten Sie den Datentyp Currency von der Klasse Show ab und geben Sie Euro und Dollar in der Form 12,03€ bzw. 14,60\$ an und Yen in der Form 120¥. Verwenden Sie hierbei nicht deriving (Show).
- (c) Schreiben Sie eine Funktion toEuro, die eine Currency entgegennimmt, den entsprechenden Betrag in Euro umrechnet und diesen dann als Currency zurück gibt.
- (d) Leiten Sie den Datentyp Currency von den Klassen Eq und Ord ab. Beachten Sie dabei, dass verschiedene Currencies verglichen werden können. Verwenden Sie hierbei nicht deriving (Eq, Ord). Geben Sie ein Beispiel als Kommentar an.

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
2 {-# HLINT ignore "Use newtype instead of data" #-}
3 {-# HLINT ignore "Eta reduce" #-}
4 -- Erstellt wird der Datentyp Currency der aus coins (Integer), cents(Integer) und currency
   ↳ (String) besteht.
5 data Currency = MkCurrency {
6     coins :: Integer,
7     cents :: Integer,
8     currency :: String
9 }
10 -- Funktion nimmt 3 Elemente entgegen und gibt coins aus.
11 -- Beispiel getCoins myEuro gibt 12 aus da er nur die ganzen Euros ausgibt
12 getCoins :: Currency -> Integer
13 getCoins (MkCurrency coins cents currency) = coins
14 -- Funktion nimmt 3 Elemente entgegen und gibt cents aus.
15 -- Beispiel getCoins myEuro gibt 03 aus da er nur die Cents ausgibt
16 getCents :: Currency -> Integer
17 getCents (MkCurrency coins cents currency) = cents
18 -- Funktion nimmt 3 Elemente entgegen und gibt currency aus.
19 -- Beispiel getCoins myEuro gibt Euro aus da er die aktuelle Währung aïs gibt.
20 getCurrency :: Currency -> String
21 getCurrency (MkCurrency coins cents currency) = currency
22 -- Festlegung für die Anzeige der Geldbeträge mit einem Komma in Euro und Dollar (bzw. Yen
   ↳ wenn keiner dieser )
23 instance Show Currency where
24     show(MkCurrency coins cents currency) =
25         if currency == "Euro" || currency == "Dollar"
26         then show coins ++ "," ++ show cents ++ " " ++ currency
27         else show coins ++ " " ++ currency
28
29 -- Die Funktionen erstellen Beispiele für jeweils Euro, Dollar und Yen aus dem Datentyp
   ↳ Currency
30 myEuro :: Currency
31 myEuro = MkCurrency 12 03 "Euro"
32
33 myDollar :: Currency
34 myDollar = MkCurrency 14 60 "Dollar"
35
36 myYen :: Currency
37 myYen = MkCurrency 120 0 "Yen"
38
39 -- Die Funktion nimmt eine Currency die entweder aus einem oder zwei Integer besteht und
   ↳ wandelt diese in ein Float um
40 -- Beispiel currencyToFloat myEuro sollte 12.03 ausgeben
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
41 -- Beispiel currencyToFloat myDollar sollte 14.60 ausgeben
42 -- Beispiel currencyToFloat myYen sollte 120.0 ausgeben
43 currencyToFloat :: Currency -> Float
44 currencyToFloat currency = fromIntegral (getCoins currency) + fromIntegral (getCents
    ↪ currency)/(10^2)
45
46 -- Die Funktion nimmt eine Currency und einen beliebigen Kurs und wandelt die Currency
    ↪ dementsprechend um.
47 --Beispiel exchangeRate myEuro 2 sollte 6.015 ausgeben da dieser durch 2 geteilt wird.
48 --Beispiel exchangeRate myDollar 2 sollte 7.3 ausgeben da dieser durch 2 geteilt wird.
49 exchangeRate :: Currency -> Float -> Float
50 exchangeRate currency rate = if rate >= 1.0 then currencyToFloat currency/rate else
    ↪ currencyToFloat currency*rate
51
52 -- Die Funktion wandelt eine Währung in eine andere um. Es muss die aktuelle Währung
    ↪ eingegeben werden, die exchange rate und in welche Währung es umgewandelt werden soll. Als
    ↪ Ergebnis wird die neue Währung als Currency ausgegeben.
53 --Beispiel exchangeCurrency myDollar 2 "Euro" gibt 7.30 Euro aus. Da in diesem Beispiel die
    ↪ exchange rate gleich 2 ist.
54 exchangeCurrency :: Currency -> Float -> String -> Currency
55 exchangeCurrency currency rate newcurrency = MkCurrency (truncate (exchangeRate currency
    ↪ rate)) (floatToDecimalPlace(exchangeRate currency rate)) newcurrency
56
57 -- Die Funktion gibt die letzten beiden Nachkommastellen eines Float als Integers zurück
58 -- Beispiel: floatToDecimalPlace 12.45 sollte 45 ergeben
59 floatToDecimalPlace :: Float -> Integer
60 floatToDecimalPlace f = round ((f-fromIntegral (truncate f))*10^2)
61
62 -- Die Funktion toEuro nimmt eine Currency rechnet sie in Euro um und gibt den Euro als
    ↪ Currency Wert ab.
63 --Beispiele toEuro myDollar sollte 13,14 Euro ausgeben, da 14.60 / 0.9 = 13.14 ergibt
64 --Beispiele toEuro myYen sollte 0.99 Euro ausgeben, da 120 * 0.0083 = 13.14 ergibt
65 toEuro :: Currency -> Currency
66 toEuro currency
67   | getCurrency currency == "Dollar" = exchangeCurrency currency 0.9 "Euro"
68   | getCurrency currency == "Yen" = exchangeCurrency currency 0.0083 "Euro"
69   | otherwise = currency
70
71 eqEuroDollar :: Currency
72 eqEuroDollar = MkCurrency 13 14 "Euro"
73
74 -- Die Funktion leitet Gleichheit von der Klasse Eq ab und
75 -- Überprüft ob die coins und cents von zwei
76 --currencies in Euro umgerechnet übereinstimmen.
77 -- Beispiel: myDollar == eqEuroDollar sollte True ergeben
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
78 -- Beispiel: myYen == myDollar sollte false ergeben.
79
80 instance Eq Currency where
81     (==) currency1 currency2 = (getCoins (toEuro currency1) == getCoins(toEuro currency2)) &&
      ↪ (getCents (toEuro currency1) == getCents(toEuro currency2))
82
83 -- Die Funktion leitet größer/kleiner gleich von der Klasse Ord ab und
84 -- Überprüft ob die coins von zwei
85 -- currencies in Euro umgerechnet kleiner gleich sind.
86 -- Beispiel: myDollar == eqEuroDollar sollte True ergeben
87 -- Beispiel: myYen == myDollar sollte True ergeben.
88 -- Beispiel: myDollar == myYen sollte False ergeben.
89 instance Ord Currency where
90     (<=) currency1 currency2 = getCoins (toEuro currency1) <= getCoins(toEuro currency2)
```
