Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

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 - Übungblatt 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 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
{-
Zum Vergleich des Argumentes s in isCold ist eine Ableitung der Klasse Eq notwendig.
Durch das Ableiten aus der Klasse Ord werden die Jahreszeiten in eine totale
Ordnung gebracht und die Operationen <, >, <=, >= für die Funktion beforeAutmn ermöglicht.
Durch das Ableiten aus der Klasse Enum werden die Jahreszeiten den Integern von 0-3
↪  zugeordnet.
Das Ableiten der Klasse Show ermögicht die Ausgabe von Spring bzw Enum 0 in printFirstSeason
in der Konsole.
-}


data Season = Spring | Summer | Autumn | Winter deriving (Eq, Ord, Enum, Show)

-- 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
```

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

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 → Vector → Point

scale : : Point → Float → 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 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use guards" #-}
{-# LANGUAGE ConstrainedClassMethods #-}
{-
Aufgabe 5_3 a
-}
-- Es werden die Datentypen Point und Vektor erstellt.
data Point = MkPoint {
    px :: Float,
    py :: Float
} deriving (Eq, Ord, Show)

data Vector = MkVector {
    vx :: Float,
    vy :: Float
}deriving (Eq, Ord, Show)
-- Die Funktion addiert einen Vektor mit einem Punkit und gibt einen neuen Punkt aus
--Beispiel translate x1 x2 ergibt (3,3)
translate :: Point -> Vector -> Point
translate (MkPoint p1 p2) (MkVector v1 v2) = MkPoint (p1+v1) (p2+v2)
--Die Funktion multipliziert einen Punkt mit einem Float und skaliert den Punkt
↪  dementsprechend.
--Beispiel scale 2 x1 ergibt (4,4) da 2+2 gleich 4 ist
scale :: Float -> Point -> Point
scale factor (MkPoint p1 p2) = MkPoint (factor * p1) (factor * p2)

{-
Aufgabe 5_3 b
-}

class Polygon p where
    area           :: Polygon p => p -> Float
    translate_poly :: Polygon p => p -> Vector -> p
    scale_poly     :: Polygon p => p -> Float -> p

data Triangle = MkTriangle {
    p1 :: Point,
    p2 :: Point,
    p3 :: Point
} deriving Show

data Quad = MkQuad {
    q1 :: Point,
    q2 :: Point,
```

```
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
        ↪  p1 p2 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p2
71      | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints
        ↪  p1 p3 > lengthBetweenTwoPoints p2 p3) = lengthBetweenTwoPoints p1 p3
72      | otherwise = lengthBetweenTwoPoints p2 p3
73
74  {-
75  Die Funktion berechnet die Summe der Kantenlängen und halbiert diese.
76  Beispiel calculateHalfOfSumEdge x1 x2 x3 ergibt 7.4966145.
77  -}
78  calculateHalfOfSumEdge :: Point -> Point -> Point -> Float
79  calculateHalfOfSumEdge p1 p2 p3 = 0.5 * (lengthBetweenTwoPoints p1 p2 + lengthBetweenTwoPoints
    ↪  p1 p3 +lengthBetweenTwoPoints p2 p3)
80
81  {-
82  Die Funktion berechnet die Höhe eine allgemeinen Dreiecks aus gehend von der Längsten Seite.
83  Beispiel calculateHeight x1 x2 x3 ergibt 2.9999995
84  -}
85  calculateHeight :: Point -> Point -> Point -> Float
```

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
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))

{-
Funktionen zum Berechnen von Quadraten
-}

{-
Hilfskonstruktion, weil wir offensichtlich die Aufgabe falsch verstehen.
-}
data Line = MkLine {
    l1 :: Point,
    l2 :: Point
} deriving (Eq, Show)

{-
Die Funktion gibt die beiden Punkte eines Dreiecks zurück,
die am weitestens auseinander liegen.
(analog zu findGround nur mit Punkten)
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
-}
findLongestDistance :: Point -> Point -> Point -> Line
findLongestDistance p1 p2 p3
    | (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p1 p3) || (lengthBetweenTwoPoints p1 p2 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p2
    | (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p1 p2) || (lengthBetweenTwoPoints p1 p3 > lengthBetweenTwoPoints p2 p3) = MkLine p1 p3
    | otherwise = MkLine p2 p3

{-
Die Funktion gibt eine beliebige der beiden Diagonalen in einem
allgemeinen Viereck zurück.
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.
-}
findDiagonal :: Quad -> Line
findDiagonal quad = findLongestDistance (q1 quad) (q2 quad) (q3 quad)

{-
Die Funktion gibt die andere Diagonale zurück.
```

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)
```

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
    scale_poly (MkTriangle p1 p2 p3) f = MkTriangle (scale f p1) (scale f p2) (scale f p3)
instance Polygon Quad where
    area quad = area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
    ↪  (breakDownLineToSecond (findDiagonal quad))(breakDownLineToFirst (findOtherPoints
    ↪  quad))) + area (MkTriangle (breakDownLineToFirst (findDiagonal quad))
    ↪  (breakDownLineToSecond (findDiagonal quad))(breakDownLineToSecond (findOtherPoints
    ↪  quad)))
    translate_poly quad v = MkQuad (translate (q1 quad) v) (translate (q2 quad) v) (translate
    ↪  (q3 quad) v) (translate (q3 quad) v)
    scale_poly quad f = MkQuad (scale f (q1 quad)) (scale f (q2 quad)) (scale f (q3 quad))
    ↪  (scale f (q4 quad))

{-
Konstanten zum Testen der Funktionen
-}
x1 :: Point
x1 = MkPoint 2 2

v1:: Vector
v1 = MkVector 1 1

x2 :: Point
x2 = MkPoint 8 2

x3 :: Point
x3 = MkPoint 7 5

x4 :: Point
x4 = MkPoint 3 1

t1 :: Triangle
t1 = MkTriangle x1 x2 x3

t2 :: Triangle
t2 = MkTriangle x1 x2 x4

q :: Quad
q = MkQuad x1 x2 x3 x4

line1 :: Line
line1 = MkLine x1 x2
```

Informatik I - Gruppe 1 - Übungblatt 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♣ *B*♦ 10♠	0
Paar (2 Karten selben Wertes)	8♦ *D*♠ 8♣	1
Drilling (3 Karten selben Wertes)	*B*♣ *B*♥ *B*♠	2
Flush (3 Karten selber Farbe)	*K*♥ 9♥ 7♥	3

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
--Erstellt wird der Datentyp Rank der aus den Zahlen von 7 bis Ass eines Kartenspiels besteht.
data Rank = Seven | Eight | Nine | Ten | Jack | Queen | King | Ace deriving (Show, Eq, Ord)
--Erstellt wird der Datentyp Suit der aus den Farben/Typen eines Kartenspiels besteht. Also Herz,Karo,Piek,Kreuz.
data Suit = Diamond | Heart | Spade | Club deriving (Show, Eq, Ord)
--Erstellt wird der Datentyp Card der jeweils aus einer Zahl und einer Farbe/Typ besteht.
data Card = MkCard {
    rank :: Rank,
    suit :: Suit
    }
    deriving (Show, Eq)
-- 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 endscheidet die Farbe/Typ welche Karte größer ist. Bei gleichem Wert und gleicher Farbe sind sie gleich groß-
--Beispiel card1 <= card2 ergibt False da King größer als die 9 ist
--Beispielt card3 <= card2 ergibt True da die 7 kleiner als die 9 ist.
instance Ord Card where
    (MkCard rank1 suit1) <= (MkCard rank2 suit2) = if rank1 == rank2 then suit1 <= suit2 else rank1 <= rank2

-- Es wird der Datentyp Hand implementiert, eine Hand besteht aus 3 Karten.
data Hand = MkHand Card Card Card
--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.
--Beispiel value myHand ergibt 0, da Die Farben der drei Karten nicht gleich sind und weder ein Drilling noch ein Paar vorhanden ist.
--Beispiel value myHand1 ergibt 3, da alle 3 Farben der drei Karten gleich sind
value :: Hand -> Integer
value (MkHand (MkCard rank1 suit1) (MkCard rank2 suit2) (MkCard rank3 suit3))
    | suit1 == suit2 && suit1 == suit3 && suit2 == suit3 = 3
    | rank1 == rank2 && rank1 == rank3 && rank2 == rank3 = 2
    | rank1 == rank2 || rank1 == rank3 || rank2 == rank3 = 1
    | otherwise = 0
-- Erstellt wird eine Hand die aus drei Karten besteht
myHand :: Hand
myHand = MkHand card1 card2 card3

card1 :: Card
card1 = MkCard King Club
```

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 - Übungblatt 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:

$$1 \text{ Dollar } = 0,90 \text{ Euro}$$
$$1 \text{ Yen } = 0,0083 \text{ Euro}$$
$$1 \text{ Dollar } = 108,59 \text{ Yen}$$

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 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use newtype instead of data" #-}
{-# HLINT ignore "Eta reduce" #-}
-- Erstellt wird der Datentyp Currency der aus  coins (Integer), cents(Integer) und currency (String) besteht.
data Currency = MkCurrency {
    coins :: Integer,
    cents :: Integer,
    currency :: String
}
-- Funktion nimmt 3 Elemente entgegen und gibt coins aus.
-- Beispiel getCoins myEuro gibt 12 aus da er nur die ganzen Euros ausgibt
getCoins :: Currency -> Integer
getCoins (MkCurrency coins cents currency) = coins
-- Funktion nimmt 3 Elemente entgegen und gibt cents aus.
-- Beispiel getCoins myEuro gibt 03 aus da er nur die Cents ausgibt
getCents :: Currency -> Integer
getCents (MkCurrency coins cents currency) = cents
-- Funktion nimmt 3 Elemente entgegen und gibt currency aus.
-- Beispiel getCoins myEuro gibt Euro aus da er die aktuelle Währung aisgibt.
getCurrency :: Currency -> String
getCurrency (MkCurrency coins cents currency) = currency
-- Festlegung für die Anzeige der Geldbeträge mit einem Komma in Euro und Dollar (bzw. Yen wenn keiner dieser )
instance Show Currency where
    show(MkCurrency coins cents currency) =
        if currency == "Euro" || currency == "Dollar"
        then show coins ++ "," ++ show cents ++ " " ++ currency
        else show coins ++ " " ++ currency

-- Die Funktionen erstellen Beispiele für jeweils Euro, Dollar und Yen aus dem Datentyp Currency
myEuro :: Currency
myEuro = MkCurrency 12 03 "Euro"

myDollar :: Currency
myDollar = MkCurrency 14 60 "Dollar"

myYen :: Currency
myYen = MkCurrency 120 0 "Yen"

-- Die Funktion nimmt eine Currency die entweder aus einem oder zwei Integer besteht und wandelt diese in ein Float um
-- Beispiel currencyToFloat myEuro sollte 12.03 ausgeben
```

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
-- Beispiel currencyToFloat myDollar sollte 14.60 ausgeben
-- Beispiel currencyToFloat myYen sollte 120.0 ausgeben
currencyToFloat :: Currency -> Float
currencyToFloat currency = fromIntegral (getCoins currency) + fromIntegral (getCents currency)/(10^2)

-- Die Funktion nimmt eine Currency und einen beliebigen Kurs und wandelt die Currency dementsprechend um.
--Beispiel exchangeRate myEuro 2 sollte 6.015 ausgeben da dieser durch 2 geteilt wird.
--Beispiel exchangeRate myDollar 2 sollte 7.3 ausgeben da dieser durch 2 geteilt wird.
exchangeRate :: Currency -> Float -> Float
exchangeRate currency rate = if rate >= 1.0 then currencyToFloat currency/rate else currencyToFloat currency*rate

-- 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.
--Beispiel exchangeCurrency myDollar 2 "Euro" gibt 7.30 Euro aus. Da in diesem Beispiel die exchange rate gleich 2 ist.
exchangeCurrency :: Currency -> Float -> String -> Currency
exchangeCurrency currency rate newcurrency = MkCurrency (truncate  (exchangeRate currency rate)) (floatToDecimalPlace(exchangeRate currency rate)) newcurrency

-- Die Funktion gibt die letzten beiden Nachkommastellen eines Float als Integers zurück
-- Beispiel: floatToDecimalPlace 12.45 sollte 45 ergeben
floatToDecimalPlace :: Float -> Integer
floatToDecimalPlace f = round ((f-fromIntegral (truncate f))*10^2)

-- Die Funktion toEuro nimmt eine Currency rechnet sie in Euro um und gibt den Euro als Currency Wert ab.
--Beispiele toEuro myDollar sollte 13,14 Euro ausgeben, da 14.60 / 0.9 = 13.14 ergibt
--Beispiele toEuro myYen sollte 0.99 Euro ausgeben, da 120 * 0.0083 = 13.14 ergibt
toEuro :: Currency -> Currency
toEuro currency
    | getCurrency currency == "Dollar" = exchangeCurrency currency 0.9 "Euro"
    | getCurrency currency == "Yen" = exchangeCurrency currency 0.0083 "Euro"
    | otherwise = currency

eqEuroDollar :: Currency
eqEuroDollar = MkCurrency 13 14 "Euro"

-- Die Funktion leitet Gleichheit von der Klasse Eq ab und
-- Überprüft ob die coins und cents von zwei
--currencies in Euro umgerechnet übereinstimmen.
-- Beispiel: myDollar == eqEuroDollar sollte True ergeben
```

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
-- Beispiel: myYen == myDollar sollte false ergeben.

instance Eq Currency where
    (==) currency1 currency2 = (getCoins (toEuro currency1) == getCoins(toEuro currency2)) &&
    ↪  (getCents (toEuro currency1) == getCents(toEuro currency2))

-- Die Funktion leitet größer/kleinergleich von der Klasse Ord ab und
-- Überprüft ob die coins von zwei
-- currencies in Euro umgerechnet kleiner gleich sind.
-- Beispiel: myDollar == eqEuroDollar sollte True ergeben
-- Beispiel: myYen == myDollar sollte True ergeben.
-- Beispiel: myDollar == myYen sollte False ergeben.
instance Ord Currency where
     (<=) currency1 currency2 = getCoins (toEuro currency1) <= getCoins(toEuro currency2)
```