Aufgabenblatt 12

Bearbeitungsende: 08.01.2023

Hinweis: Alle Punkte, die bis zum 08.01. auf diesem Blatt erlangt werden, werden als Bonuspunkte verrechnet.

Das Blatt kann dann weiter bis zum 15.01. bearbeitet werden (Einreichungen unter „X“-Tasks im Praktomat).

Aufgabe 1 [Programmierung – nicht einzureichen]

Schreiben Sie eine Klasse `Feld` mit folgenden Klassenmethoden:

- `anzahlIdentisch` nimmt ein Array von `Object`-Referenzen an sowie eine weitere `Object`-Referenz. Sie gibt die (ganze) Zahl von Objekten in dem Array zurück, die identisch mit dem gegebenen Objekt sind.
- `anzahlGleich` nimmt ein Array von `Object`-Referenzen an sowie eine weitere `Object`-Referenz. Sie gibt die (ganze) Zahl von Objekten in dem Array zurück, deren Wert gleich dem Wert des gegebenen Objekts ist.

Aufgabe 2 [Programmierung]

Erstellen Sie zuerst (vor dem Code) das UML-Klassendiagramm der zu implementierenden Klassen.

Schreiben Sie eine Klasse `Stromvertrag`, die einen Vertrag mit einem Energieversorger modelliert. Der Vertrag hat Fixkosten von 1000 Cent (€10) pro Monat und einen variablen Strompreis von 30 Cent pro kWh.

Zu einem Vertrag existieren folgende Daten:

- Ganzzahlwert `anzahl` ist die Anzahl aller abgeschlossenen Verträge.
- Ganzzahlwert `nummer` ist die Nummer des Vertrages – die Verträge sind fortlaufend numeriert, der erste Vertrag hat die Nummer 1.
- Zeichenkette `name` ist die Person, mit der der Vertrag abgeschlossen wurde.
- Ganzzahlwert `monat` enthält den Monat der letzten Abrechnung, gerechnet in Monaten seit Dezember 2014. (Januar 2015 war Monat 1.)
- Ganzzahlwert `zaehler` enthält den (nichtnegativen) Zählerstand (in kWh) zum Zeitpunkt der letzten Abrechnung.

Es gibt einen Konstruktor, der den Namen, den Monat (> 0) des Vertragsabschlusses und den (nichtnegativen) Zählerstand zu diesem Zeitpunkt als Argumente annimmt.

Die Klasse bietet folgende Methoden:

- `anzahl`, `nummer`, `monat` und `zaehler` nehmen keine Argumente an und liefern die entsprechenden Werte.
- `kostenFix` nimmt als Argument eine (nichtnegative ganze) Anzahl von Monaten an und gibt die entsprechenden Fixkosten zurück.
- `kostenVariabel` nimmt als Argumente eine Anzahl von Monaten und eine Anzahl von kWh an (beide Werte nichtnegativ und ganzzahlig) und gibt die entsprechenden variablen Kosten zurück.
- `rechnung` führt eine Abrechnung durch und nimmt dazu als Argumente den aktuellen Monat und den Zählerstand an. Daraus bestimmt sie, wieviele Monate seit der letzten Abrechnung vergangen sind und wieviele kWh verbraucht wurden. Sie aktualisiert Abrechungsmonat und -zählerstand des Vertrags mit den neuen Werten und gibt die seit der letzten Abrechnung entstandenen Kosten zurück.

Alle Geldbeträge sind ganzzahlig in Cent anzugeben.

Schreiben Sie eine Klasse `StromvertragOeko`, die `Stromvertrag` spezialisiert. Dieser Vertrag hat Fixkosten von 1200 Cent (€12) pro Monat und einen variablen Strompreis von 35 Cent pro kWh.

Diese Klasse hat ihren eigenen Zähler `anzahl`, der die Anzahl von Objekten der Klasse zählt. Dessen Wert wird ebenfalls von einer Methode `anzahl()` zurückgegeben.

Überschreiben Sie in dieser Klasse Methoden von `Stromvertrag` nur dann, wenn es notwendig ist, wenn also die geerbte Methode nicht das richtige Verhalten bietet.

Für ungültige Argumentwerte soll jeweils eine `IllegalArgumentException` geworfen werden.

Aufgabe 3 [Programmierung]

Erstellen Sie zuerst (vor dem Code) das UML-Klassendiagramm der zu implementierenden Klassen.

Schreiben Sie Klassen `Mensch`, `Frau` und `Mann` so, die folgende Eigenschaften modellieren:

- Frauen und Männer sind Menschen.
- Man kann Frau- und Mann-Instanzen erzeugen, nicht jedoch Mensch-Instanzen.
- Für jede der drei Klassen liefert eine Methode `anzahl`, wieviele Instanzen von dem jeweiligen Typ existieren.
- Frauen und Männer werden (ausschließlich) unter Angabe von Name, Alter, (tatsächlicher) Körpergröße und (tatsächlichem) Körpergewicht (in dieser Reihenfolge) instanziiert.
- Der Name eines Menschen wird bei Instanziierung unveränderlich festgelegt und von Methode `name()` geliefert.
- Eine Frau hat die Chromosomen `"XX"`, ein Mann `"XY"`. Methode `chromosomen()`, liefert das Chromosomenpaar eines Menschen als Zeichenkette.
- Methode `alter()` liefert das (ganzzahlige) Alter eines Menschen.
- Methode `volljaehrigkeitsAlter()` liefert das Alter der Volljährigkeit von Menschen. Dieses Alter liegt für alle Menschen bei 18 Jahren. Die Methode soll selbst dann aufgerufen werden können, wenn keine Instanz eines Menschen vorliegt.

- Methode `istVolljaehrig()` liefert als Wahrheitswert, ob der jeweilige Mensch volljährig ist.
- Wenn man einen Mann nach seinem Körpergewicht fragt, behauptet er, 5 kg schwerer zu sein als er tatsächlich ist. Frauen dagegen nennen stets ihr wirkliches Gewicht. Methode `gewicht()` liefert den von dem jeweiligen Menschen genannten Wert (ganzzahlig, in Kilogramm).
- Wenn man eine Frau nach ihrer Körpergröße fragt, behauptet sie, 5 cm kleiner zu sein als sie tatsächlich ist. Männer dagegen nennen stets ihr wirkliches Maß. Methode `groesse()` liefert den von dem jeweiligen Menschen genannten Wert (ganzzahlig, in Zentimetern).
- `setAlter`, `setGroesse` und `setGewicht` nehmen jeweils ein ganzzahliges Argument an und setzen für einen Menschen die entsprechende Angabe auf den gegebenen Wert. Dabei handelt es sich jeweils um den tatsächlichen Wert, nicht unbedingt um den nachher von dem Menschen behaupteten. Die Methoden geben kein Ergebnis zurück.
- Der Preis eines Haarschnitts in Euro wird für eine Frau bekanntlich nach der Formel „$20 + \frac{2}{3} \cdot$ Alter“ bestimmt, für einen Mann als „$10 + \frac{1}{4} \cdot$ Alter“, wobei die errechneten Werte auf ganze Euro gerundet werden (`Math.round`). Die Methode `preisHaarschnitt()` soll für jeden Menschen den so berechneten Wert zurückgeben.
- Methode `toString()` gibt Name, Chromosomen, Alter, Erreichen der Volljährigkeit, (angebliche) Größe, (angebliches) Gewicht und den Preis eines Haarschnitts, durch Leerzeichen getrennt, als einzeilige Zeichenkette zurück – z.B.:

  `"Cristiano␣Ronaldo␣XY␣32␣true␣191␣83␣18"`

Verwenden Sie in angemessener Weise Vererbung, abstrakte und finale Konstrukte. Alle Variablen sollen `private` sein. Speichern Sie auch Konstanten in Variablen ab. Bemühen Sie sich, die Lösung so einfach wie möglich zu halten.

Zusatzfrage (muss nicht in der einzureichenden Lösung berücksichtigt werden)**:** Wie müssten Sie Ihre Lösung anpassen, wenn sich Frauen als 3 Jahre älter, Männer als 3 Jahre jünger ausgäben als sie tatsächlich sind, wenn sich aber alle Verwendungen des Alters in den obigen Beschreibungen auf das tatsächliche Alter beziehen sollten? Wie gingen Sie vor, um Schwindeleien dieser Art zu verhindern?

Aufgabe 4 [Programmierung]

Erstellen Sie zuerst (vor dem Code) das UML-Klassendiagramm der zu implementierenden Klassen.

Verwenden Sie im Folgenden die aus der Vorlesung bekannte Klasse `Punkt2D`, die einen 2D-Punkt modelliert.

Modellieren Sie zweidimensionale geometrische Figuren, namentlich (achsparallele) Rechtecke. Deklarieren Sie dazu die folgenden Typen und überlegen Sie dabei insbesondere, welche Klassen und Methoden abstrakt oder konkret deklariert werden sollten. Stellen Sie Methodenimplementierungen nur bereit, wenn sie sinnvolle Ergebnisse liefern.

`Figur` modelliert zweidimensionale Figuren und stellt folgende Eigenschaften bereit:

- einen Konstruktor ohne Argumente
- einen Kopie-Konstruktor, der eine `Figur`-Referenz als Argument annimmt
- `mitte` liefert den Mittelpunkt der Figur (als von der Figur unabhängiges Objekt)

- `durchmesser`, `umfang` und `flaeche` liefern den (maximalen) Durchmesser, den Umfang und den Flächeninhalt der Figur.
- `istEnthalten` nimmt eine `Punkt2D`-Referenz als Argument an und gibt als Wahrheitswert zurück, ob der Punkt innerhalb der Figur (incl. ihres Randes) liegt.
- `verschiebe` nimmt zwei Gleitkommawerte `dx` und `dy` als Argumente an, in dieser Reihenfolge, und verschiebt die Figur um diese Werte in x- und y-Richtung. Die Methode gibt eine Referenz auf die Figur zurück.
- `toString` liefert eine Darstellung der Figur als Zeichenkette.

`FigurPerMitte` hat die Eigenschaften von `Figur` und beschreibt intern die Lage einer Figur im Raum durch ihren Mittelpunkt (als private Instanz, auf die keine externe Partei Zugriff haben soll). Sie spezialisiert die Eigenschaften von `Figur` oder ergänzt sie in folgender Weise:

- Ein Konstruktor nimmt eine Referenz auf ein `Punkt2D`-Objekt als Argument an und verwendet dieses als Vorgabe für den Mittelpunkt.
- Ein weiterer Konstruktor nimmt eine Referenz auf ein `FigurPerMitte`-Objekt als Argument an und legt das Objekt als Kopie von diesem an.

`RechteckXYPerMitteLaengen` ist eine `FigurPerMitte` und beschreibt Rechtecke intern zudem durch ihre (nichtnegativen) Ausdehnungen in x- und y-Richtung. Sie spezialisiert die Eigenschaften von `FigurPerMitte` oder ergänzt sie in folgender Weise:

- Ein Konstruktor nimmt eine Referenz auf ein `Punkt2D`-Objekt an, das den Mittelpunkt des Rechtecks vorgibt, sowie zwei nichtnegative Werte `breite` und `hoehe` die die Ausdehnungen des Rechtecks in x- und y-Richtung vorgeben.
- Ein weiterer Konstruktor nimmt eine Referenz auf ein `RechteckXYPerMitteLaengen`-Objekt als Argument an und legt das Objekt als Kopie von diesem an.
- `breite` und `hoehe` liefern die Ausdehnungen des Rechtecks in x- und y-Richtung.
- `ecke` nimmt zwei Wahrheitswerte `istRechts` und `istOben` an. Je nach Wert der Argumente gibt die Methode die linke untere, linke obere, rechte untere oder rechte obere Ecke des Rechtecks als `Punkt2D`-Objekt zurück.
- `equals` vergleicht mit einem anderen Objekt auf Gleichheit. Dabei sollen Rundungsfehler ignoriert werden. Verwenden Sie dazu die aus der Vorlesung bekannte Methode `Mathe.equals`.
- `toString` 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.

Verwenden Sie `final` überall, wo möglich und angemessen.

Hinweis: Verwenden Sie so wenige Methodenimplementierungen wie möglich; überlegen Sie, welche Methoden gemeinsam für mehrere Klassen implementiert werden können.

Aufgabe 5 [Programmierung]

Erweitern Sie zuerst (vor dem Code) das UML-Klassendiagramm aus der vorherigen Aufgabe um die in dieser Aufgabe zu erstellende Klasse.

Ergänzen Sie die vorherige Aufgabe, so dass auch Kreise modelliert werden. Deklarieren Sie dazu die folgende Klasse und überlegen Sie dabei, welche Klassen und Methoden konkret oder abstrakt deklariert werden sollten. Stellen Sie nur Methoden bereit, die sinnvolle Ergebnisse liefern.

`KreisPerMitteRadius` ist eine `FigurPerMitte` und beschreibt Kreise intern zudem durch ihren (nichtnegativen) Radius. Sie spezialisiert die Eigenschaften von `FigurPerMitte` oder ergänzt sie in folgender Weise:

- Ein Konstruktor nimmt eine Referenz auf ein `Punkt`-Objekt an, das den Mittelpunkt des Kreises vorgibt, sowie einen nichtnegativen Wert `radius`, der den Radius vorgibt.
- Ein weiterer Konstruktor nimmt eine Referenz auf ein `KreisPerMitteRadius`-Objekt als Argument an und legt das Objekt als Kopie von diesem an.
- `radius` liefert den Radius des Kreises.
- `toString` gibt die Darstellung des Kreises im Format `[`*`mitte`*`,`*`radius`*`]` zurück, worin *`mitte`* für die Darstellung der Mitte im Format von `Punkt` steht und *`radius`* für den Zahlwert des Radius.

Verwenden Sie `final` überall wo möglich und angemessen.

Hinweis: Verwenden Sie `Math.PI` als Wert für π.

Frage: Welche Schwierigkeiten ergeben sich, wenn man die Klasse `RechteckXYPerEcke` aus B08A3 in die vorliegende Hierarchie einfügen möchte?

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** .