9. Übung Programmierung I: Java

Ausgabe: 17.12.2018
Abgabe: 08.01.2019 $12^{\underline{00}}$

Aufgabe	Punkte	Basislösung	Abgabetests
1	105 + 29 (Bonus)	-	Übersetzbarkeit, Einhaltung der OO-Formatierungsrichtlinien, (Teil-) Korrektheit des Algorithmus‘
Gesamtpunkte	**105 + 29**		

1. Aufgabe:

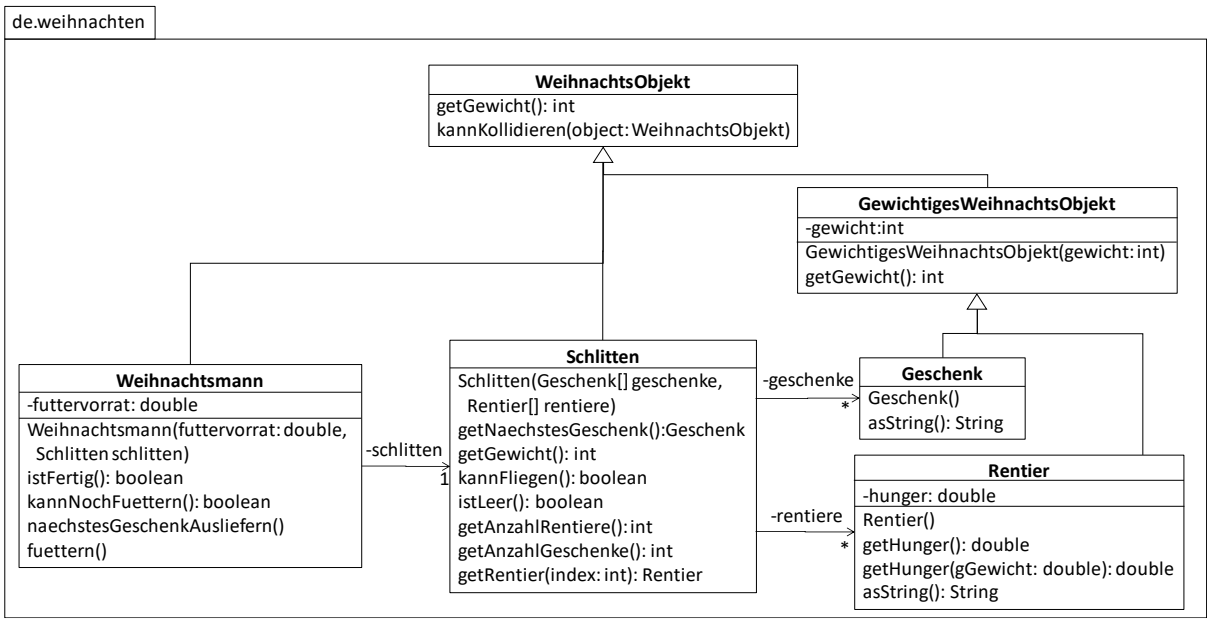
Der Weihnachtsmann muss eine Menge Geschenke rechtzeitig ausliefern. Dafür hat er einen Schlitten, der von Rentieren gezogen wird. Sofern der Schlitten durch das Gewicht der Geschenke nicht zu schwer ist, kann die Reise losgehen. Da diese Reise besonders für die Rentiere sehr anstrengend ist, müssen sie regelmäßig gefüttert werden, sodass der Weihnachtsmann hoffentlich zu Beginn seiner Reise genug Futter eingesteckt hat.

Die Beschreibung der Aufgabe ist sehr detailliert, insbesondere um die Beziehungen zwischen Klassen und die Verteilung von (kleinen) Funktionalitäten und Verantwortlichkeiten zu beschreiben. Lesen Sie Sich die Aufgabe zunächst komplett durch. Insbesondere für Teil a) ist es sinnvoll, die Klassen und Methoden in der gegebenen Reihenfolge zu planen / zu realisieren. Das Punkteschema orientiert sich grob an der Vorgehensweise für Testate.

Bitte benennen Sie in ihrem Hauptprogramm, das in Aufgabenteil b) beschrieben wird, die Teilaufgaben die Sie bearbeitet haben und markieren Sie Programmabschnitte für Teil c) oder d) durch entsprechende Kommentare.

Aufgabenteil a) (74 Punkte)

Gegeben sei das folgende Klassendiagramm als verpflichtender Teil des Plans für diese Aufgabe. Eine illustrative Zielbeschreibung finden Sie am Ende der Aufgabe.



Das Programm soll einige Eigenschaften erfüllen. Wir beschreiben zunächst die einfachen und dann die komplexeren Klassen. Die Methoden der komplexeren Klassen sollen durch die Methoden der einfacheren Klassen realisiert werden.

- Ein **`WeihnachtsObjekt`** ist alles was mit dem Weihnachtsmann zu tun hat, potentiell ein Gewicht hat und mit einem anderen `WeihnachtsObjekt` kollidieren kann.
- Ein **`GewichtigesWeihnachtsObjekt`** ist ein `WeihnachtsObjekt` mit eigenem Gewicht in kg. Der Weihnachtsmann – Eigenname, daher gegen alle Konventionen kein großes „M“ –ist zwar ein `WeihnachtsObjekt` aber kein `GewichtigesWeihnachtsObjekt`, da er glaubt nichts zu wiegen (hat also ein Gewicht von 0), denn sonst macht er sich nicht auf die Reise und es gibt keine Geschenke. Aus magischen Gründen können weder der `Weihnachtsmann`, die Rentiere noch die Geschenke mit anderen Weihnachtsobjekten kollidieren.
- Einige Eigenschaften werden magisch, d.h., per Zufall, andere per Berechnung bestimmt. Für beides ist die Klasse `java.lang.Math` sehr hilfreich. An einigen Stellen benötigen wir eine zufällige Fließkommazahl aus einem vorgegebenen Bereich. Hierfür macht es Sinn, die Klasse `XMasUtils` (nicht im Diagramm dargestellt) zu erstellen. Eine statische Methode dieser Klasse gibt für gegebene Minimum/Maximum-Grenzen eine zufällige Fließkommazahl innerhalb dieses Bereichs zurück. Analog benötigen wir an einigen Stellen zufällige Ganzzahlen innerhalb gegebener Minimum/Maximum-Grenzen – sinnigerweise unter Verwendung der Methode für Fließkommazahlen.
- Ein **`Geschenk`** hat ein zufälliges Gewicht in Kilogramm. Dieses wird beim Erstellen des Geschenks im Konstruktor bestimmt. Das Gewicht muss zwischen 1kg und 20kg (beides inklusive) liegen. Die Methode `asString()` gibt einen Text zurück, mit dem das Geschenk mit Gewicht beschrieben wird. Wir nehmen an, dass der Weihnachtsmann auf magische Weise die Ziel-Adresse des Geschenks kennt und wir diese nicht verwalten müssen.
- Jedes **`Rentier`** hat ein zufälliges Gewicht (Minimalgewicht 200kg, Maximalgewicht 300kg) und verdrückt pro Mahlzeit eine unterschiedliche Anzahl an Essensrationen. Das eigene Gewicht und der individuelle `hunger` wird bei der Erstellung des Rentiers im Konstruktor zufällig ermittelt. Der `hunger` beträgt zwischen einer und zwei Essensrationen (beides inklusive) und wird von der Methode `getHunger()` zurückgegeben. Die Methode `getHunger(double gGewicht)` Klasse `Rentier` gibt die Menge an Essensrationen zurück, die dieses Rentier pro Mahlzeit abhängig vom Gesamtgewicht der Geschenke `gGewicht` verdrückt. Die Menge an Essensrationen (`getHunger`) wird mit `hunger` + 0.01 * `gGewicht` berechnet. Bei 100kg Geschenken und einem individuellen `hunger` von 1.5 benötigt das Rentier also 1.5 + 0.01 * 100 = 2.5 Essensrationen. Die Methode `asString()` gibt eine Beschreibung des Rentiers mit seinem individuellen Hungerwert zurück.
- Die Methode `getNaechstesGeschenk()` in der **Klasse `Schlitten`** nimmt das oberste Geschenk vom Schlitten (Index 0) und gibt es zurück. Beachten Sie, dass durch das Entnehmen des Geschenks dieses aus dem Schlitten entfernt werden muss und ggf. die restlichen Geschenke im Schlitten aufrutschen. `getGewicht()` gibt das Gesamtgewicht aller Geschenke in kg zurück - der Schlitten selbst wiegt magischerweise nichts. `kannFliegen()` gibt an, ob die Rentiere ausreichen, um den Schlitten mit seinem aktuellen Geschenkgewicht in die Luft zu bringen. *n* Rentiere sind in der Lage ein Geschenkgewicht von $100 * \sqrt{n}$ kg zu ziehen. `istLeer()` gibt an, ob alle Geschenke bereits vom Schlitten entnommen wurden. `getAnzahlRentiere()` bzw. `getAnzahlGeschenke()` liefern die aktuelle

Anzahl an Rentieren bzw. Geschenken. `getRentier(int index)` gibt das Rentier an der spezifizierten Stelle zurück, wobei `index` zwischen 0 und `getAnzahlRentiere()` liegen muss (beides inklusive).

- `istFertig()` in der **Klasse `Weihnachtsmann`** gibt an, ob alle Geschenke vom Schlitten ausgeliefert wurden. Die Methode `kannNochFuettern()` gibt an, ob der aktuelle Futtervorrat ausreicht, um den Hunger seiner Rentiere zu stillen. Die Methode `naechstesGeschenkAusliefern()` gibt das nächste auszuliefernde Geschenk auf der Konsole aus, sofern der Schlitten noch nicht leer ist. Die Methode `fuettern()` verringert den Futtervorrat des Weihnachtsmanns um die Menge, die seine Rentiere einmalig fressen.
- Bei `schlitten`, `geschenke` und `rentiere` handelt es sich um Assoziationen. Die Klassen von denen der Pfeil im Diagramm ausgeht beinhalten also ein entsprechendes Attribut mit dem jeweiligen Namen. Bei Assoziationen die mehrere Elemente repräsentieren können (*) ist ein Array zur Realisierung angebracht.
- Auch wenn im Diagramm nicht dargestellt, sollen alle Klassen öffentlich (`public`) sichtbar sein. Alle Klassen liegen im Paket `de.weihnachten`. Für diese Aufgabe genügt es, wenn die Methoden/Konstruktoren der Klassen, wie bislang in der Vorlesung, keine Sichtbarkeit haben – `public` ist aber auch möglich. Attribute müssen `private` sein. Jede Klasse hat eine `main`-Methode, die die jeweilige Klasse demonstriert/testet. Jede Methode ist geeignet dokumentiert, insbesondere durch eine Zeile über dem Methodenkopf, in der die Methode kurz beschrieben wird. Methoden die Attribute verändern werden nicht benötigt – die Konstruktoren genügen. Weitere Hilfsmethoden zum Berechnen von Werten, z.B., dem aktuellen Hunger aller Rentiere am Schlitten können angelegt werden.
- Alle Methoden im Hauptprogramm müssen wie bisher üblich `static` sein. In den Klassen sind nur die test-Methoden `static`. Alle andere Methoden sind nicht static.

Aufgabenteil b) (31 Punkte)

Die Hauptklasse `Weihnachten` des Programms liegt ebenfalls im Paket `de.weihnachten`. Erstellen Sie einen geeigneten Plan für die folgende Beschreibung:

Zunächst soll der Benutzer eine gewünschte (positive) Anzahl an Rentieren und Geschenken, sowie die (positive) Größe des Futtervorrats eingeben. Fehlerhafte Zahleneingaben sind zu erkennen und solange abzuweisen, bis der Benutzer eine korrekte Eingabe tätigt. Fehleingaben wie Buchstaben für Zahlen können ignoriert werden.

Legen Sie nun einen neuen Weihnachtsmann an und übergeben Sie ihm den Futtervorrat als Konstruktor-Argument. Erstellen Sie einen neuen Schlitten, indem Sie beim Konstruktoraufruf die die Rentiere und Geschenke angeben. Anschließend wird der Schlitten dem Weihnachtsmann zur Verfügung gestellt.

Überprüfen Sie, ob die Rentiere in der Lage sind den Schlitten mit den Geschenken zum Fliegen zu bringen. Beenden Sie das Programm mit einer Fehlermeldung, falls dies nicht der Fall ist. Ist der Schlitten flugfähig, wird die Anzahl der Geschenke, die Anzahl der Rentiere und das Gewicht des Schlittens (über Methoden des Schlittens) ausgegeben. Anschließend werden alle Rentiere einzeln durch Ausgabe ihres Texts (`asString`) vorgestellt.

Nun kann die Reise beginnen. Wenn der Weihnachtsmann noch nicht fertig ist, wird das nächste Geschenk ausgeliefert. Danach werden, sofern der Futtervorrat reicht, die Rentiere gefüttert. Reicht der Vorrat nicht mehr, bevor alle Geschenke ausgeliefert werden konnten, wird eine entsprechende Meldung ausgegeben und das Programm wird beendet. Konnten alle Geschenke ausgeliefert werden, wird eine entsprechende Meldung ausgegeben und das Programm wird ebenfalls beendet.

Hinweis: Eclipse markiert ggf. die Scanner-Instanz als nicht geschlossen und daher als potentielles Ressourcen-Problem. Schließen Sie dann den Scanner wenn sie ihn nicht mehr benötigen durch die Scanner-Methode `close()`.

Aufgabenteil c) (12 Punkte, Bonus)

Erweitern Sie das Programm, d.h., zunächst Plan und Klassendiagramm, indem Sie die Klasse `Rudolph` von `Rentier` erben lassen. Rudolph überschreibt die `asString()`-Methode und stellt sich anders vor als seine Rentier-Kollegen. Damit seine Nase auch rot leuchtet, hat Rudolf grundsätzlich doppelt so viel Hunger wie der hungrigste seiner Rentier-Kollegen, d.h., sein Hungerfaktor ändert sich von 0.01 zu 0.02. Realisieren Sie dies durch geeignete Anpassung von `Rudolph` und `Rentier` aber ohne zusätzliches Attribut und ohne die Berechnungsformel in `Rudolph` zu wiederholen. Sorgen Sie dafür, dass bei der Erstellung der Rentiere zuerst ein `Rudolph` und danach die restlichen Rentiere angelegt werden, wobei die Gesamtanzahl weiterhin der Konsoleneingabe entsprechen soll, also ein normales Rentier weniger angelegt werden muss. Natürlich muss auch Rudolph getestet werden.

Aufgabenteil d) (17 Punkte, Bonus)

Letztes Jahr ist der Weihnachtsmann mit seinem Schlitten bei Nebel im Wald an einem Tannenbaum hängengeblieben. Für dieses Jahr wünscht er sich eine einfache Kollisionsprüfung, die vor der Ausführung der Tour ausgeführt wird. Wird eine Kollision erkannt, wird die Tour wie bei falschem Gewicht abgebrochen. Eine echte Smartphone-Navigation wäre natürlich besser, aber der Weihnachtsmann ist ja bescheiden. Erweitern Sie zunächst Plan und Klassendiagramm anhand der Folgenden Beschreibung.

Ein `Tannenbaum`, ist ein `WeihnachtsObjekt` ohne Gewicht ist. Allerdings kann ein `Tannenbaum` mit einem Schlitten kollidieren (Methode `kannKollidieren`) - wir wollen ja nicht, dass dem Weihnachtsmann oder den Rentieren etwas passiert. Ob ein Objekt `objekt` vom Typ `T` ist, kann man mit `objekt instanceof T` herausfinden.

Tragen Sie in der Klasse `Weihnachten` dann eine zufällige Anzahl von 10-50 Tannenbäumen an zufälligen Positionen in eine Karte (ein zwei-dimensionales Array der Größe 30x30 mit Elementen vom Typ `WeihnachtsObjekt)` ein, und fragen Sie dann mit der Methode `kannKollidieren` ab, ob eine diagonale Route durch die Karte ohne Kollision mit dem Schlitten möglich ist.

Geben Sie in einem Kommentar an, wie Sie Ihr Programm anpassen, müssen, wenn zusätzlich noch `Riesenrad` und `Glühweinbude` als Hindernisse berücksichtigt werden sollen.

Beispiel Teilaufgaben für Teile a-c

```
Bitte geben Sie die Anzahl der Rentiere ein:
4
Bitte geben Sie die Anzahl der Geschenke ein:
10
Bitte geben Sie den Futtervorrat in kg ein:
100
Wir starten Weihnachten mit 10 Geschenke 4 Rentiere und einem Schlitten mit 85 kg.
Die Rentiere sind:
 - Ich bin Rudolph und meine Nase leuchtet. Mein indiv. Hunger ist 1.0 und mein
Gewicht ist 272 kg
 - Ich bin ein Rentier mit indiv. Hunger 1.0 und Gewicht 245 kg
 - Ich bin ein Rentier mit indiv. Hunger 1.0 und Gewicht 236 kg
 - Ich bin ein Rentier mit indiv. Hunger 1.0 und Gewicht 233 kg
Die Tour beginnt:
 - Der Weihnachtsmann liefert aus: Geschenk: 1 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 11 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 15 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 12 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 4 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 2 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 1 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 19 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 14 kg
 - Der Weihnachtsmann liefert aus: Geschenk: 6 kg
Der Weihnachtsmann ist fertig mit der Auslieferung. Schöne Weihnachten.
```

Optionale Navigationskarte für Teil d

```
>---------------- KARTE -----------------
0:    T    T                   T
1:       T             T      T
2:       T
3:      T    T          T     T
4:                        T     T
5:          T               T
6:
7:                 T
8:
9:                              T
10:
11:        TT          T   T
12:  T
13:             T             T
14:                  T
15:      T      T
16:
17:                              T
18:
19:                    T
20:                                T
21:
22: T                              T
23:   T            T      T
24:    T              T
25:  T      T
26:                     T
27: T   T   T
28:      T             T
29:
<----------------- KARTE -----------------
```

Lego-Komposition aus Weihnachtsmann-Instanz, Schlitten-Instanz, 3 Rentier-Instanzen, einer Rudolph-Instanz (vorne links), diversen Geschenk-Instanzen und einigen Tannenbaum-Instanzen. Haus und Wichtel sind nicht (auch noch) Bestandteile der Aufgabe. Die Diagonal-Route ist (offensichtlich) frei, der Weihnachtsmann hat genügend Futter, die Geschenke sind nicht zu schwer und so kann die Reise beginnen.

Das Team von Programmierung I – Java
wünscht Ihnen (trotz Weihnachtsaufgabe)
schöne Weihnachten
und einen Exception-freien Rutsch
ins Jahr 2019.