

Einführung in die Programmierung  
WS 2018/19

Übungsblatt 11: Vererbung

Besprechung: 21.01 - 25.01.2019

**Aufgabe 11-1**      *Wichtel Industries*

Wie Ihnen sicherlich bekannt ist, ist der Weihnachtsmann in der heutigen globalisierten Welt nicht mehr in der Lage, die harten SLA's einzuhalten, die ihn dazu zwingen, pünktlicher als jedes Versandhaus seine Lieferungen abzuschließen. Ihm steht daher ein großer Mitarbeiterstab zur Seite, der für ihn die Versandgüter fertigt. Die Mitarbeiter heißen Wichtel und sind genetisch augmentierte Superarbeiter, die in der Weihnachtszeit jeden Tag ein großes Pensum an Aufgaben zu bearbeiten haben.

In dieser Aufgabe sollen Sie eine Wichtel-Werkstatt simulieren und die Fertigung der Geschenke protokollieren. Die Werkstatt-Klasse liegt Ihnen schon vor. Das Hauptprogramm erstellt eine einzelne Werkstatt, befüllt sie mit Wichteln und Geschenkaufträgen und lässt die Wichtel solange arbeiten, bis alle Geschenke gefertigt wurden. Am Ende der Schicht sollen die Wichtel nach Effizienz sortiert werden. Die Klassen `Wichtel`, `RoterWichtel`, `GelberWichtel`, `BlauerWichtel`, `Geschenk`, `Spielzeug`, `Kleidung`, `Essbares` sowie `WerkstattTools` haben folgende Funktionalität:

- Jeder Wichtel hat einen Namen von Geburt an, der von keiner anderen Klasse verändert werden darf. Außerdem soll die gearbeitete Zeit (ganzzahlig) und die Anzahl gefertigter Geschenke festgehalten werden. Keines dieser Attribute darf für eine andere Klasse sichtbar sein. Zusätzlich soll eine Variable `dauer` die Zeit festhalten, die ein beschäftigter Wichtel für die derzeitige Aufgabe noch benötigt.
- Der Name eines Wichtels kann zufällig mit `Zufall.koboldname()` generiert werden. Fügen Sie eine `toString`-Methode hinzu, die den Namen des Wichtels als String zurückgibt.
- Jeder Wichtel ist entweder ein roter, blauer oder gelber Wichtel. Modellieren Sie dies entsprechend.
- Jeder Wichtel hat eine Methode `arbeite(Geschenk g)`, die dem Wichtel einen neuen Auftrag zuweist. Dabei unterscheidet sich die Ausführung von Wichtelart zu Wichtelart.
- Jeder Wichtel soll eine Methode `arbeitetNoch()` bereitstellen, die seine Beschäftigung anzeigt.
- Außerdem soll eine Methode `arbeiteWeiter()` ihn dazu bringen, am derzeitigen Stück weiterzuarbeiten. Bei jedem Aufruf soll sich die `dauer` um 1 verringern. Der Wichtel hat eine weitere Zeiteinheit gearbeitet und wenn die gesamte Dauer gearbeitet wurde, hat er ein weiteres Geschenk fertig gestellt.

- Ein Geschenk kann ein Spielzeug, etwas Essbares oder ein Kleidungsstück sein. Es kann aber auch etwas ganz anderes sein, wird dann aber allgemein als Geschenk betitelt. Ein Geschenk hat einen Namen und eine Schwierigkeit. Der Typ von Schwierigkeit ist `double`. Benutzen Sie für diese Attribute ebenfalls die vorgegebenen Generierungsfunktionen in der Klasse `Zufall`. Gerne dürfen Sie auch weitere kreative Geschenke ergänzen. Ein Geschenk soll mit einer `toString`-Methode den Namen und die Schwierigkeit zurückgeben. Beide Attribute müssen in den Subklassen sichtbar sein, aber nur dort. Implementieren Sie die getter-Methoden `name()` und `schwierigkeit()`. Ein normales Geschenk hat eine Schwierigkeit zwischen 0 und 25.
- **Essbares** hat zusätzlich noch ein Attribut `gesund`. Etwas Essbares ist genau dann gesund, wenn der ganzzahlige Anteil der Schwierigkeit gerade ist (vgl. Sie dazu die Beispielausgabe). Überschreiben Sie die `toString` Methode und ergänzen Sie eine Ausgabe über das zusätzliche Attribut. Die Schwierigkeit liegt zwischen 0 und 15.
- **Kleidung** hat eine Eleganz, die der Namenslänge des Geschenks entspricht. Die Schwierigkeit liegt zwischen 0 und 5. Die `toString`-Methode soll auch hier zusätzlich das weitere Attribut ausgeben.
- **Spielzeug** ist zu einem bestimmten Grad spannend. Das Attribut `Spannung` ist vom Typ `double` und entspricht der Schwierigkeit\*Namenslänge/10. Die Schwierigkeit liegt zwischen 0 und 10. Ebenfalls soll die `toString`-Methode entsprechend überschrieben werden.
- Jeder Sub-Wichtel soll die Methode `toString()` überschreiben, sodass zusätzlich noch die Farbe des Wichtels ausgegeben wird. Weiter soll jeder Subwichtel die Methode `arbeite()` überschreiben. Normalerweise arbeiten alle Wichtel solange an einem Geschenk, wie es schwierig ist (gerundet). Allerdings hat jeder Wichtel seine Ausnahmen.  
Rote Wichtel arbeiten generell sehr schnell und brauchen für alles zwei Zeiteinheiten weniger, außer bei Spielzeug - da arbeiten sie dann so schnell wie alle anderen.  
Gelbe Wichtel sind sehr künstlerisch und modeorientiert. Sie brauchen 2 Zeiteinheiten länger für alle Kleidung und eine Zeiteinheit länger für alles andere.  
Blaue Wichtel mögen gerne alles Essbare. Da sie einen Teil selber verspeisen, brauchen sie für diese nur die Hälfte der Zeit. Bei allem anderen sind sie behäbiger und brauchen 3 Zeiteinheiten länger.  
Hinweis: `object instanceof Class` wertet genau dann zu `true` aus, wenn `object` eine Instanz der gegebenen Klasse ist.
- Die Klasse `Wichtel` benötigt noch die Methoden aus Teilaufgabe d).
- Die Klasse `WerkstattTools` benötigt die Methoden aus Teilaufgabe c).

Beachten Sie, dass in der Klasse `Werkstatt` die Ausgabe (z.B. `System.out.println`) so umgeschrieben wurde, dass die ganze Ausgabe in eine Datei Namens `Werkstattlog.txt` umgeleitet wird. Wenn Sie das zum Programmieren nicht wollen, dann kommentieren Sie die angesprochenen 4 Zeilen aus.

Achten Sie auch darauf, dass Ihre Ausgaben denen des vorliegenden `Werkstattlog`-Files gleichen.

- Erstellen Sie ein UML-Diagramm, wobei die Sichtbarkeiten angegeben sein sollen, von den vorgegebenen und den zu implementierenden Klassen. Dabei soll stets die restriktivste Sichtbarkeit für Methoden und Attribute gewählt werden, wenn nicht anders vorgegeben. Beachten Sie die Konzepte der Datenkapselung.
- Implementieren Sie alle Klassen entsprechend der Spezifikation. Dabei soll stets die restriktivste Sichtbarkeit für Methoden und Attribute gewählt werden, wenn nicht anders vorgegeben. Beachten Sie die Konzepte der Datenkapselung.

- (c) Implementieren Sie eine Klasse `WerkstattTools`, die die Methoden `Geschenk[] generiereGeschenke(int anzahl)` sowie `Wichtel[] generiereWichtel(int anzahl)` enthält. Es sollen dabei jeweils zufällige Geschenke bzw. Wichtel in einem Array erzeugt werden.
- (d) Nach der Arbeit soll das Wichtelarray sortiert werden. Ergänzen Sie also für einen Wichtel die Methode `effizienz()`, die den Quotienten aus der Anzahl gefertigter Geschenke und der gearbeiteten Zeit zurückgibt. Wichtel sollen dann das vordefinierte Interface `Comparable<T>` implementieren. Der Typ `T` muss natürlich Wichtel selbst sein. Die Klasse Wichtel muss dann die Funktion `int compareTo(Wichtel andererWichtel)` implementieren. Schauen Sie in der Java API die Funktionalität dieser Methode nach und implementieren Sie sie geeignet. Anschließend kann das Hauptprogramm Werkstatt auch das Wichtelarray sortieren, weil es nun weiß, wie man zwei Wichtel miteinander vergleicht.
- (e) Nun können in der `main`-Methode der Hauptklasse Werkstatt die Wichtel der Effizienz nach sortiert werden. Der Weihnachtsmann ist letztlich auch ein Businessman und da es keine Geschlechter bei Wichtel gibt (Es sind letztlich nur surreale Geschöpfe), werden die besten jeder Kategorie geklont. Fügen Sie der Wichtelklasse einen Konstruktor hinzu, der eine tiefe Kopie eines Wichtels erstellt. Klonen Sie dann die drei Wichtel, von denen jeder jeweils der effizienteste seiner Farbklasse ist.