AP 11: Übung zu OOP

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` und Interfaces `IWichtel` und `IGeschenk` haben folgende Funktionalität:

1. Jeder Wichtel hat einen **Namen** von Geburt an, der nicht mehr 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 ganzzahlige Variable **Dauer** die Zeit festhalten, die ein beschäftigter Wichtel für die derzeitige Aufgabe noch benötigt. Dieses Attribut soll in abgeleiteten Klassen sichtbar sein.

2. Der Name eines Wichtels soll zufällig mit der Methode `koboldname()` generiert werden. Fügen Sie eine `toString`-Methode hinzu, die den Namen des Wichtels als String zurückgibt.

3. Jeder Wichtel ist entweder ein roter, blauer oder gelber Wichtel. Modellieren Sie dies entsprechend.

4. Jeder Sub-Wichtel soll die Methode `toString()` überschreiben, so dass zusätzlich noch die Farbe des Wichtels ausgegeben wird.

5. Jeder Wichtel soll eine Methode `boolean arbeitetNoch()` bereitstellen, die zurückgibt, ob ein Wichtel gerade arbeitet.

6. Außerdem soll eine Methode `void 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.

7. In einem Interface `IWichtel` sollen alle Methoden eines Wichtels erfasst werden, die Klasse `Wichtel` soll dieses Interface implementieren. Beachten Sie außerdem die Methode `arbeite`, die in Punkt 13 genauer beschrieben wird.

8. 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** vom Typ int. Benutzen Sie für diese Attribute ebenfalls die vorgegebenen Generierungsfunktionen in der Klasse `Zufall`. Dabei müssen Sie noch die Methode einer zufälligen Schwierigkeit implementieren (siehe Aufgabe 2.). 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 `getName()` und `getSchwierigkeit()`. Ein normales Geschenk hat eine Schwierigkeit zwischen 1 und 25.

9. In einem Interface `IGeschenk` sollen alle Methoden eines Geschenks erfasst werden, die Klasse `Geschenk` soll dieses Interface implementieren.

10. `Essbares` hat zusätzlich noch ein Attribut `gesund`. Etwas Essbares ist genau dann gesund, wenn die Schwierigkeit gerade ist. Überschreiben Sie die `toString`-Methode und ergänzen Sie eine Ausgabe über das zusätzliche Attribut. Die Schwierigkeit liegt zwischen 1 und 15.

11. `Kleidung` hat eine `Eleganz`, die der Namenslänge des Geschenks entspricht. Die Schwierigkeit liegt zwischen 1 und 5. Die `toString`-Methode soll auch hier zusätzlich das weitere Attribut ausgeben.

12. `Spielzeug` ist zu einem bestimmten Grad spannend. Das Attribut `Spannung` ist vom Typ double und entspricht der Schwierigkeit * Namenslänge / 10. Die Schwierigkeit liegt zwischen 1 und 10. Ebenfalls soll die `toString`-Methode entsprechend überschreiben werden.

13. Jeder Wichtel hat eine Methode `void arbeite(IGeschenk g)`, die dem Wichtel einen neuen Auftrag zuweist. Dies soll über das Setzen des Attributs `dauer` geschehen. Dabei unterscheidet sich die Ausführung von Wichtelart zu Wichtelart. Normalerweise arbeiten alle Wichtel solange an einem Geschenk, wie es schwierig ist. Allerdings hat jeder Wichtel auch seine Ausnahmen.
    Rote Wichtel arbeiten generell sehr schnell und brauchen für alles zwei Zeiteinheiten weniger (aber immer mindestens eine Zeiteinheit), 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 (gerundet). Bei allem anderen sind sie behäbiger und brauchen 3 Zeiteinheiten länger.
    (Hinweis: Verwenden Sie das Schlüsselwort `instanceof`)

Aufgaben:

1. Erstellen Sie für die Klassen `Werkstatt`, `Wichtel`, `RoterWichtel`, `BlauerWichtel`, `GelberWichtel`, `Geschenk`, `Spielzeug`, `Kleidung`, `Essbares` und den Interfaces `IWichtel` und `IGeschenk` ein UML-Klassendiagramm entsprechend der obigen Beschreibung. Die Methoden in der Klasse `Werkstatt` dürfen weggelassen werden. Wählen Sie für Attribute und Methoden stets die restriktivste Sichtbarkeit, wenn nichts anderes vorgegeben ist. Berücksichtigen Sie ebenfalls Konstruktoren im Klassendiagramm.

2. Nun sollen alle oben beschriebenen Klassen implementiert werden. Gehen Sie dabei wie folgt vor:
   (a) In der Projektvorlage finden Sie schon alle notwendigen Klassen, sowie eine Klasse `Werkstatt` (ignorieren Sie die Compilerfehler, diese werden bei der schrittweisen Bearbeitung der weiteren Aufgaben behoben), die eine Bearbeitung simulieren soll, sowie eine Klasse `Util`, die Sie für die jeweiligen `toString` Methoden verwenden können.
   (b) Ergänzen Sie in der Klasse `Util` eine Methode `public static int getZufallszahl(int max)`, die eine zufällige ganze Zahl zwischen 1(inklusive) und der übergebenen maximalen Zahl (ebenfalls inklusive) erzeugen soll. Achten Sie auf ungültige Werte für den Parameter `max`. Diese Methode können Sie unter anderem für die Klasse `Geschenk` nutzen.
   (c) Implementieren Sie nun alle Klassen gemäß ihrer oben beschriebenen Spezifikation.

3. Nun sollen Generierungsmethoden für Wichtel und Geschenke implementiert werden. Gehen Sie dabei wie folgt vor:
   (a) Um später bei Erweiterungen (neue Art von Geschenken oder Entdeckung einer neuen Wichtelart) so wenig Änderungen wie möglich am bisherigen Quellcode durchzuführen, erstellen Sie jeweils eine Enum `WichtelType` und `GeschenkType`. Informieren Sie sich über den Aufbau von Enums.
   (b) Ergänzen Sie die Enums um die jeweiligen Wichtel- bzw. Geschenkarten.
   (c) Ergänzen Sie die Enum der Wichtel um ein private-Attribut `type` vom Datentyp String und einen private-Konstruktor der dieses Attribut setzen kann. Die einzelnen Enum-Einträge sollen diesen Konstruktor mit einem passen String aufrufen.
   (d) Überschreiben Sie die `toString`-Methode in `WichtelType`, so dass sie den Wert von `type` zurückgibt. Passen Sie dann die `toString`-Methode in den abgeleiteten Klassen von `Wichtel` an.
   (e) Erstellen Sie ein Methode `public static List<Wichtel> generiereWichtel(int anzahlWichtel)` in der Klasse `Wichtel` und eine Methode `public static List<IGeschenk> generiereGeschenke (int anzahlGeschenke)` in der Klasse `Geschenk`. Diese beiden sollen Listen mit der übergebenen Anzahl an zufälligen Wichteln und Geschenken zurückgeben. Verwenden Sie dazu die erstellten Enums. (Hinweis: switch/case bei Enums, die Methode `values` bei Enums, die Methode `getZufallszahl`)

4. Nach der Arbeit soll die Liste von Wichteln sortiert werden. Ergänzen Sie dafür zuerst im Interface `IWichtel` die Methode `double effizienz()`, die den Quotienten aus der Anzahl fertiger Geschenke und der gearbeiteten Zeit zurückgeben soll und implementieren Sie diese in der Klasse `Wichtel`.
   Das Interface `IWichtel` soll danach um das generische Interface `Comparable<T>` entsprechend erweitert werden, damit in der Klasse `Werkstatt` die Liste von Wichteln mit `Collections.sort` sortiert werden kann. Die Klasse `Wichtel` muss dafür die Methode `int compareTo(Object o)` geeignet implementieren. Informieren Sie sich dazu in der Java API über die Funktionalität dieser Methode und implementieren Sie diese Methode geeignet. Ergänzen Sie diese ebenfalls Erweiterung im Klassendiagramm.

5. 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.