Übung 03: Vererbung

Abgabetermin: 26.03.2020, 8:15

Aufgabe	Punkte	abzugeben über Moodle	Punkte
Übung 3	24	Java Programmcode in Zip-Datei Java Programmcode als pdf-Datei	

Übung 03: Zaubertränke (24 Punkte)

In dieser Übung soll ein Klassensystem entwickelt werden, das beispielsweise in Computerspielen genutzt werden kann, um Zaubertränke herzustellen. Dabei soll das Composite-Pattern angewendet werden, da sich ein Zaubertrank aus verschiedenen Teilen zusammensetzen kann, unter anderem aus anderen Zaubertränken.

a) Interfaces und Default-Methoden

Interface Priced und Enum DiscountRate:

Entwickeln Sie zuerst ein Interface `Priced`, welches von Klassen implementiert werden soll, die einen Preis haben, um beispielsweise bei einem Händler im Computerspiel gekauft oder verkauft werden können. Dabei können Rabatte zum Einsatz kommen, deren Höhe durch eine Enumeration `DiscountRate` mit den Werten `LOW`, `MEDIUM` und `HIGH` bestimmt wird.

Das Interface soll folgende abstrakte Methoden umfassen:

- `float getPrice()`

  Liefert den Preis (in Einheit „Geld“) des jeweiligen Produktes.

- `float getDiscountPercent(DiscountRate rate)`

  Liefert den Rabattprozentsatz des jeweiligen Produktes, abhängig von der übergebenen Rabattrate.

Darüber hinaus soll das Interface folgende default-Methoden zur Verfügung stellen:

- `float getDiscountPercent()`

  Liefert den Standard-Rabattprozentsatz. Diese Methode soll standardmäßig den Rabattprozentsatz liefern, der mit der Rabattrate `DiscountRate.LOW` erzielt wird.

- `float getDiscount(DiscountRate rate)`

  Liefert den Rabatt (in Einheit „Geld“), also jene Menge an Geld, die gespart wird, wenn der Rabattprozentsatz der angegebenen Rabattrate angewendet wird.

- `float getDiscount()`

  Liefert den Standardrabatt (in Einheit „Geld“), also jene Menge an Geld, die gespart wird, wenn der Standard-Rabattprozentsatz angewendet wird.

- `float getReducedPrice(DiscountRate rate)`

  Liefert den verbilligten Preis, also jene Menge an Geld, die bezahlt werden muss, wenn vom Originalpreis der Rabatt entsprechend der angegebenen Rabattrate abgezogen wird.

- `float getReducedPrice()`

  Liefert den verbilligten Preis, also jene Menge an Geld, die bezahlt werden muss, wenn vom Originalpreis der Standardrabatt abgezogen wird.

b) Composite-Pattern

Um Zaubertränke (`Potion`) darzustellen, die sich aus verschiedenen Bestandteilen (`Item`) zusammensetzen können, unter anderem aus anderen Zaubertränken, soll folgende Klassenstruktur dazu umgesetzt werden:

```
Item
  |--- Potion
  |--- Resource
        |--- Herb
        |--- AnimalProduct
```

Klasse Item

- Stellt die abstrakte Basisklasse des Kompositums dar.
- Implementiert das Interface `Priced`.
    - Überschreiben Sie `getDiscountPercent(DiscountRate rate)` so, dass bei einer Rabattrate von `LOW` 1% Rabatt, bei `MEDIUM` 5% Rabatt und bei `HIGH` 15% Rabatt zurückgegeben wird.
    - Überschreiben sie `getPrice()` nicht, dies wird von den Kinderklassen implementiert.
- Jedes `Item` hat neben dem Preis
    - einen *Namen*,
    - eine Heilkraft *Power* (dh, beim Verzehr des Items wird die angegebene Menge an Lebensenergie des Spielers wiederhergestellt),
    - sowie eine Abklingzeit *Cooldown* (dh, die Zeit, die verstreichen muss, bis dasselbe Item erneut verzehrt werden kann – angegeben in Sekunden).
- Definieren Sie für die Eigenschaften passende Getter-Methoden, gegebenenfalls als abstrakt, um von den Kinderklassen implementiert zu werden.
- Definieren Sie einen Konstruktor, der einen Namen entgegennimmt und diesen in einem finalen privaten Feld speichert.
- Überschreiben Sie die `toString()`-Methode, indem Sie den Namen zurückgeben.

Klasse Resource

- Ist eine abstrake Klasse und stellt die atomaren Teile des Kompositums dar, also jene Items, die sich aus keinen weiteren Items zusammensetzen.
- Entwickeln Sie einen Konstruktor, der den Namen der Ressource entgegennimmt.
- Der Preis einer `Resource` ergibt sich aus der Länge des Namens, dh, die Ressource kostet ein Geld pro Buchstabe im Namen. Beispiele: Ein „Ei“ kostet 2 Geld, „Schnittlauch“ kostet 12 Geld. Überschreiben Sie den Preis-Getter entsprechend.

Klasse Herb

- Ist eine `Resource`, die Kräuter darstellt.
- Die Heilkraft und die Abklingzeit sollen zusätzlich zum Namen als Konstruktor-Parameter übergeben, in finalen privaten Felder gespeichert und über die jeweiligen Getter nach außen zur Verfügung gestellt werden.
- Überschreiben Sie zudem die `toString()`-Methode, indem sie `super.toString()` aufrufen und den String `„Herb: “` voranstellen.

Klasse AnimalProduct

- Ist eine `Resource`, die ein Tierprodukt darstellt.
- Entwickeln Sie einen Konstruktor, der den Namen des Tierproduktes entgegennimmt.
- Jedes Tierprodukt hat eine Abklingzeit von 10 Sekunden.
- Die Heilkraft entspricht dem doppelten Preis.
- Überschreiben Sie die `toString()`-Methode, indem sie `super.toString()` aufrufen und den String `"AnimalProduct: "` voranstellen.

Klasse Potion

Stellt Zaubertränke dar, die sich aus beliebig vielen verschiedenen `Item`s zusammensetzen können (also aus `Herb`, `AnimalProduct` und wiederum aus `Potion`). `Potion` soll folgend implementiert werden:

- Übergeben Sie zusätzlich zum Namen die Items, aus denen sich der Zaubertrank zusammensetzt, als vararg-Paramter im Konstruktur und legen Sie diese in einem finalen privaten Feld ab.
- Der Preis eines Zaubertranks ergibt sich aus der Summe der Preise der beinhalteten `Item`s plus einem Aufschlag von 20%.
- Die Heilkraft entspricht dem Doppelten der Summe der Heilkräfte der beinhalteten `Item`s.
- Die Abklingzeit entspricht dem Doppelten der maximalen Abklingzeit der beinhalteten `Item`s.
- Diese Werte sollen *nicht* in Feldern gespeichert werden, sondern entsprechend dem Composite-Pattern in den einzelnen Gettern berechnet werden.
- Überschreiben Sie des Weiteren die `toString()`-Methode, in der Sie `"Potion: "`, gefolgt von `super.toString()`, gefolgt von einer öffnenden eckigen Klammer, gefolgt von (durch Beistriche getrennt) den textuellen Darstellungen der beinhaltenden Items (abgefragt über `toString()`) und abschließend gefolgt von einer schließenden eckigen Klammer zurückgeben. Verwenden Sie zum Aufbauen dieses Strings einen `StringBuilder`.

Beispiel:

```
Herb „Schnittlauch“:
        Preis:          12 Geld
        Heilkraft:      10 (übergeben als Parameter)
        Abklingzeit:    5 (übergeben als Parameter)
        toString():     "Herb: Schnittlauch"
AnimalProduct „Spinnenbein“:
        Preis:          11 Geld
        Heilkraft:      22
        Ablinkzeit:     10
        toString():     "AnimalProduct: Spinnenbein"
Potion „Dunkeltrank“:
        Bestandteile:   Schnittlauch, Spinnenbein (übergeben als Parameter)
        Preis:          27,6 Geld
        Heilkraft:      64
        Abklingzeit:    20
        toString():     "Potion: Dunkeltrank [ Herb: Schnittlauch, AnimalProduct: Spinnenbein ]"
Potion „Nachttrank“:
        Bestandteile:   Dunkeltrank, Spinnenbein (übergeben als Parameter)
        Preise:         46,32 Geld
        Heilkraft:      172
        Abklingzeit:    40
        toString():     "Potion: Nachttrank [ Potion: Dunkeltrank [ Herb: Schittlauch, AnimalProduct: Spinnenbein ],
                        AnimalProduct: Spinnenbein ]"
```

c) Test

Erstellen Sie eine Klasse `App` mit einer `main`-Methode, in der Sie mindestens zwei `Herb`-Objekte, zwei `AnimalProduct`-Objekte sowie zwei `Potion`-Objekte anlegen. Das letzte `Potion`-Objekt soll aus den anderen Items bestehen.
Schreiben Sie eine Methode, die für ein gegebenes `Item` die Rückgabewert der folgenden Methoden ausgibt:

- `toString()`
- `getPrice()`
- `getReducedPrice(DiscountRate.HIGH)`
- `getReducedPrice()`
- `getPower()`
- `getCooldown()`

Rufen Sie diese Methode für alle von Ihnen angelegten `Items` auf und prüfen Sie die Werte auf Richtigkeit.

Download:

Zusätzlich zu der von Ihnen zu erstellenden App-Klasse stellen wir eine Klasse `Example` zur Verfügung. In dieser befindet sich eine `main`-Methode, welche die oben genannte `Potion` „Nachttrank“ umsetzt und ihre Werte ausgibt. Diese können Sie zu Testzwecken verwenden.