Testat-Übung 1: Vererbung

Semesterwoche 4

Abgabe

Spätestens bis und mit Sonntag, 18. Oktober 2020 23:00 Uhr

Gruppenarbeit bitte nur max. 3 Leute. Abgabe via MS Teams. Bitte Teilnehmende einer Gruppenarbeit in einem zusätzlichen File `group.txt` vermerken. Bei allfälligen Problemen bitte umgehend ein Email an den Übungsbetreuenden senden.

Geben Sie nur `.java`-Files ab! Kein Eclipse-Projekt und kein Zip-Archiv!

Lernziele

- Vererbung verstehen und anwenden.
- Überschreiben und dynamisches Binden von Methoden üben.
- Objektorientiertes Design vertiefen.

Aufgabe 1: Bestellungssystem

Für ein Bestellungssystem sollen Bestellungen als Objekte verwaltet werden können. Ihre Aufgabe ist es, entsprechende Klassen dafür anzubieten. Folgende Eigenschaften sollen für die Bestellungen gelten:

- Jede Bestellung besteht aus mehreren Positionen, die entweder Produkt-Positionen oder Dienstleistungs-Positionen sind.
- Eine Produkt-Position hat eine Bezeichnung, eine Anzahl und einen Stückpreis.
- Eine Dienstleistungs-Position hat eine Bezeichnung und einen Pauschalpreis.
- Stückpreis, Pauschalpreis und Anzahl sollen veränderlich sein, die Bezeichnung nicht.
- Jede Position ist in der Lage, ihren Totalpreis zu berechnen: Bei einer Produkt-Position ist dies die Anzahl mal Stückpreis, bei einer Dienstleistungs-Position der Pauschalpreis.
- Jede Position kann ihre Daten (Bezeichnung, Anzahl, Stückpreis bzw. Pauschalpreis) ausgeben.
- Die Bestellung kann sowohl den Totalpreis berechnen als auch die Liste aller enthaltenen Positionen ausgeben.

Entwickeln Sie die Klassen für dieses Bestellungswesen und verwenden Sie Vererbung für die Positionen. Eine mögliche Organisation als Klassen ist im untenstehenden Diagramm skizziert. Die Bestellung speichert ihre Positionen in einem Array (`Item[]`).

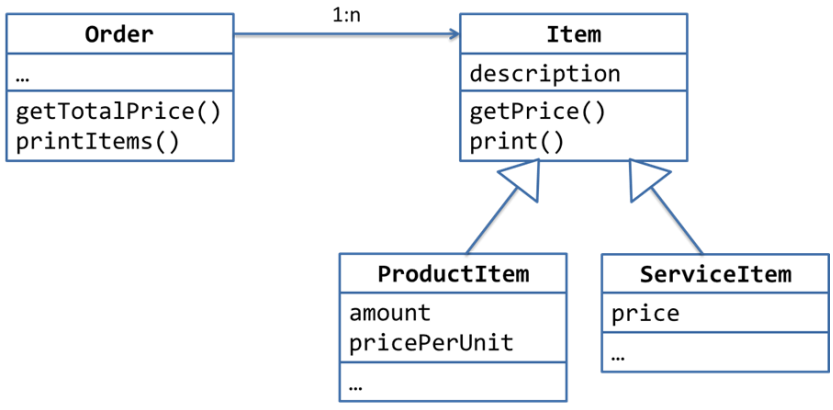

Abbildung 1: Klassen-Diagramm des Bestellwesens

Treffen Sie Ihre eigenen Entscheidungen für die Implementierung der Klassen. Testen Sie Ihr Programm. In der Vorlage finden Sie die Klasse `OrderSystemTest` mit einer leeren `main()`-Methode. Sie können die Tests auf einzelne statische Methoden aufteilen, dass die `main()`-Methode nicht zu lang wird.

Aufgabe 2: Bundle-Position

Das Bestellsystem aus Aufgabe 1 soll nun erweitert werden: Neu gibt es eine zusätzliche Position, nämlich die Bundle-Position. Es gilt hierfür:

- Die Bundle-Position besteht wiederum aus beliebig vielen Positionen.
- Die Bundle-Position hat auch einen Bezeichner und einen Rabatt (in Prozent).
- Der Rabatt ist veränderlich. Unterpositionen sollen hinzufügbar und entfernbar sein.
- Der Preis der Bundle-Position ist die Summe seiner beinhalteten Position abzüglich des Rabatts auf der Summe.
- Die Ausgabe der Bundle-Position liefert seinen Bezeichner und Rabatt und gibt auch alle beinhalteten Positionen aus.

Erweitern Sie Ihre Lösung aus Aufgabe 1 um die zusätzliche Klasse für die neue Bundle-Position. Ein mögliches Klassendiagramm ist unten angeben. Das Bundle kann hier die Unterpositionen mit Hilfe eines Array verwalten (allenfalls mit einer Limite an Unterpositionen).

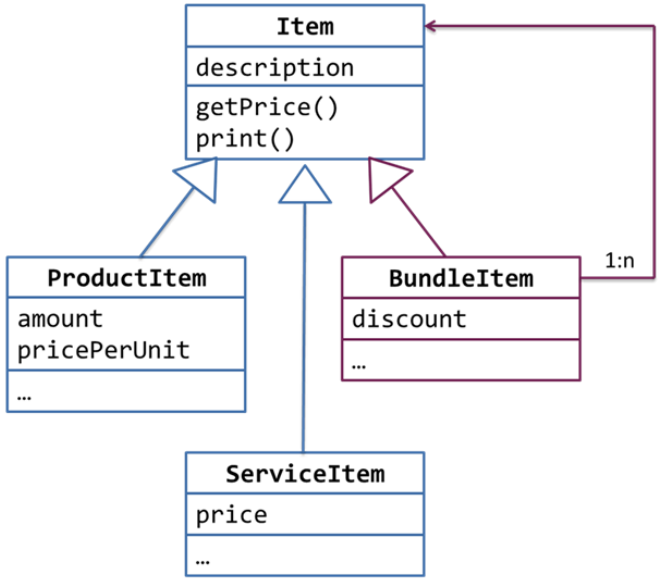

Abbildung 2: Klassen-Diagramm mit Bundle-Position

Testen Sie auch hier Ihr Programm. Sie sollten insbesondere auch möglich sein, dass ein Bundle wiederum aus einem oder mehreren Bundles besteht, weil ein Bundle ja auch eine Position ist.

Hinweis: Diese Struktur nennt man *Composite Design Pattern*.

Aufgabe 3: Zyklen-Vermeidung (fakultativ)

Da Bundles in Bundles bestehen können, sind Zyklen grundsätzlich möglich. So könnte es vorkommen, dass ein Bundle sich selbst direkt oder indirekt wieder beinhaltet. Ohne Vorkehrungen führt dies zu Fehlern, z.B. eine Endlosrekursion bei der Ausgabe bzw. Preisberechnung. Erweitern bzw. ändern Sie Ihr Design, so dass solche Zyklen in jedem Fall verhindert werden. Testen Sie Ihre Lösung.