HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN
WÜRZBURG-SCHWEINFURT

FH·W-S

Programmieren II
Sommersemester 2016

Übungsblatt 6 (2 Punkte)

Polymorphie

Polymorphie ist ein mächtiges Werkzeug, um Programmlogik auf höherer Ebene allgemein zu formulieren und erst in den Unterklassen konkret zu implementieren. Abstrakte Methoden in einer abstrakten Klasse oder auch Interfaces verzichten sogar vollständig auf die Bereitstellung einer Implementierung und delegieren die gesamte Verantwortung an ihre Unterklassen – mit der Konsequenz, dass abstrakte Klassen oder auch Interfaces nie instanziiert werden können.

In der vorliegenden Übung sind Sie gefordert, konkrete Implementierungen für abstrakte Methoden zu entwickeln bzw. vorhandene Methoden zu überschreiben, so dass am Ende ein allgemein formulierter Algorithmus funktioniert – egal, welche der möglichen Implementierungen er verarbeiten muss.

Aufgabe „exercise06“ (Abgabe vom 19.5. - 25.05.2016, bZv-relevant)

Sie arbeiten in der IT-Abteilung eines Unternehmens, das ausschließlich nur einen Typ von 100 Watt Glühbirnen vertreibt – ausdrücklich nicht als Beleuchtungsmittel (das ist nach EU-Recht verboten), sondern als Raumzusatzheizung (die zufälligerweise auch noch leuchtet).

Ihre Verantwortung ist es, die Lagerlogistik Ihres Unternehmens mit Hilfe von Java-Klassen abzubilden. Einige Vorgaben haben Sie nach Gesprächen mit Ihren Auftraggebern bereits in Code gegossen:

Zentrales Objekt der Lösung ist der Materialbuchungssatz, der mit Hilfe der Klasse `MovementRecord` abgebildet wird. Ein solcher Buchungssatz umfasst die materialabgebende Stelle `source`, die materialannehmende Stelle `sink`, die Anzahl der Glühbirnen `count` und eine Markierung `done`, mit der festgehalten wird, ob die Buchung bereits durchgeführt wurde.

Die entscheidende Methode zur Verbuchung einer Materialbewegung besitzt folgende Logik:

1. Falls der Buchungssatz schon verbucht wurde, wird nichts getan und `false` zurückgegeben (d.h. ein Buchungssatz kann nicht zweimal verbucht werden).
2. Andernfalls wird die gewünschte Anzahl von der materialabgebenden Stelle abgebucht, der materialaufnehmenden Stelle zugebucht und der Satz als gebucht markiert. Der Rückgabewert ist in diesem Fall `true`.
3. Sollte die abgebende Stelle – aus welchen Gründen auch immer – nicht in der Lage sein, die gewünschte Anzahl zu liefern, unterbleibt natürlich die Verbuchung und es wird `false` zurückgegeben.

Sie finden diese Methode bereits fertig implementiert in der bereitgestellten Klasse `MovementRecord` im Verzeichnis `Übungsaufgabe_6`. Die Typdefinitionen für `source` und `sink` werden in Form der Interfaces `GoodsSource` und `GoodsSink` ebenfalls im Verzeichnis `Übungsaufgabe_6` bereitgestellt. Die in der Klasse MovementRecord enthaltene `main`-Methode sollte nach Lösung der Aufgabe ablauffähig sein.

Ihre Aufgabe:

Um das System zu vervollständigen, müssen Sie je eine Klasse für Kunden (Klassenname `Customer`), Lieferanten (Klassenname `Vendor`) und Lager (Klassenname `Stock`) entwickeln. Für diese Klassen gelten folgende Vorgaben:

- Sowohl Kunden als auch Lieferanten sind Partner des Unternehmens. Daher sollen diese beiden neuen Klassen Subklassen der bereitgestellten abstrakten Klasse `Partner` sein.
- Lieferanten sind materialabgebende Stellen. Daher sollte die Klasse `Vendor` das Interface `GoodsSource` implementieren. Sie können dabei davon ausgehen, dass unsere Lieferanten immer lieferfähig sind. Überschreiben Sie bitte außerdem die `toString()`-Methode der Oberklasse, indem Sie vor das Ergebnis der Oberklasse den String „Lieferant:“ stellen.
- Kunden sind eine materialannehmende Stelle (Interface `GoodsSink`). Die an einen Kunden gelieferte Menge soll im `Customer`-Objekt aufsummiert werden. Überschreiben Sie bitte außerdem die `toString()`-Methode der Oberklasse, indem Sie vor das Ergebnis der Oberklasse den String „Kunde:“ stellen.
- Am Monatsende soll an alle Kunden eine Rechnung über die bezogenen Glühbirnen gesandt werden. Implementieren Sie hierzu in der Klasse `Customer` eine zusätzliche Methode

  ```
  public void sendInvoice()
  ```

  die folgende Funktionalität bietet:
  - Auf der Konsole soll das Ergebnis von `toString()` sowie die Anzahl der gelieferten Glühbirnen ausgegeben werden (genaues Format ist nicht vorgegeben).
  - Natürlich muss die Zählung nach der Rechnungslegung wieder bei null beginnen.
- Ein Lager ist sowohl eine materialannehmende als auch eine materialabgebende Stelle. Bitte beachten Sie, dass ein Lager nur dann liefern kann, wenn es zuvor mit der entsprechenden Menge bestückt wurde. Überschreiben Sie in der Klasse `Stock` auch die Methode `toString()`, so dass ein aussagekräftiger String zurückgeliefert wird.