Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2011/`

Java-Übung Blatt 5 (Vererbung)

2011-05-30

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Exportieren Sie das Projekt als ZIP-Datei mit Namen *accountname-projectname*`.zip`. Beispiel: Peter Lustig (`lustigp`) gibt `ex02_1` ab als Datei `lustigp-ex02_1.zip`
- ZIP-Export in Eclipse: Rechtsklick auf die Wurzel des Projects, "Export...", "General/Archive File..."
- Mailen Sie die ZIP-Files an Ihren Tutor.

Abgabe: Montag, 6. Juni 2011, um 9.00 Uhr.

Aufgabe 1 (Inventur, 4 Punkte)

Projekt: `ex05_1`. Package: `inventory`.

In dieser Aufgabe sollen Sie abstrakte Klassen einsetzen, um das vorgegebene Design zu verbessern und Redundanzen zu vermeiden. Laden Sie sich zunächst das Projekt-Skelett `ex05_1` von der Vorlesungswebsite herunter.

Im Projekt befinden sich innerhalb des Pakets `inventory` die Klassen `Antique` und `Vegetable`. Diese Klassen beschreiben Produkte eines Gemischtwarenhändlers. Instanzen dieser Klassen repräsentieren ein Produkt dieser Produktkategorie mit einer eindeutigen ID und einem Brutto-Verkaufspreis. Beispielsweise könnte eine Instanz von `Vegetable` einen Kohlkopf repräsentieren.

Für die jährliche Inventur muss der aktuelle Warenwert der Produkte im Lager bestimmt werden. Daher implementiert jedes Produkt das Interface `CurrentAsset`. Dessen Methode `netCashEquivalents` gibt den aktuellen Nettowert des Produktes zurück. Der Nettowert ist der Bruttowert des Produkts nach Abzug der Mehrwertsteuer. Der Bruttowert eines Produktes ist der Wert der unter Einbeziehung der warenspezifischen Risiken und Faktoren sofort am Markt zu erzielen wäre.

Es gilt ein allgemeiner Mehrwertsteuersatz von 19%, dieser ist für alle Waren identisch. Die Klassen `Antique` und `Vegetable` berechnen diesen Nettowert auf unterschiedliche Weise, aber beide berechnenen zunächst den eigenen Bruttowert, also den Wert der unter Einbeziehung der warenspezifischen Risiken und Faktoren sofort am Markt zu erzielen wäre. Die Klasse `Antique` berücksichtigt den Markt für Antiquitäten (den wir nicht genauer betrachten), die Klasse `Vegetable` das Alter des Gemüses. Der Bruttowert von frischem Gemüse entspricht seinem Bruttopreis, verliert aber pro Tag Alter ein siebtel des Wertes, ist also nach 7 Tagen wertlos.

(a) Vor dem Refactoring sollen Sie Tests schreiben, um sicherzustellen, dass die neue Implementierung die gleichen Ergebnisse liefert wie die alte. Schreiben Sie einen Test der bestätigt, dass die Methode `netCashEquivalents` den Wert `60` zurückliefert, wenn diese auf einem `new Vegetable("Cabbage", 100, 2)` aufgerufen wird. Zeigen Sie in einem weiteren Test, dass Gemüse nach 7 Tagen wertlos ist. Die Implementierung von `netCashEquivalents` in der Klasse `Antique` betrachten wir im Rahmen dieser Aufgabe als nicht ohne weiteres mit Unittests testbar.

Ändern Sie nun das Design so, dass sich die Gemeinsamkeiten von `Antique` und `Vegetable` in einer abstrakten Basisklasse `AProduct` wiederfinden, die das Interface `CurrentAsset` implementiert. Ausschließlich die Basisklasse soll dieses Interface implementieren. Die abgeleiteten Klassen sollen dabei den produktspezifischen Teil der Wertberechnung übernehmen. Lagern Sie möglichst viel Funktionalität in die Basisklasse aus, insbesondere auch gleiche Felder. Felder sollen dabei bestenfalls `private`-, keinesfalls jedoch `public`-Sichtbarkeit haben. Die abgeleiteten Klassen sollen

den produktspezifischen Teil der Wertberechnung übernehmen. Dabei soll möglichst spät auf ganze Centbeträge gerundet werden. Die Funtionalität der Klassen soll sich durch das Redesign der Klassen nicht verändern, d.h. Tests, die vor Ihren Änderungen erfolgreich waren, sollen weiterhin erfolgreich sein.

(b) Implementieren Sie zwei Klassen `Clothing` (Kleidungsstück) und `Book` (Buch), um zu demonstrieren, dass Ihre abstrakte Basisklasse auch für neue Produktarten als Basis geeignet ist. Bücher sollen eine Preisbindung haben, d.h. die Klasse `Book` darf keine nach öffentlich sichtbare (`public`) Setter-Methode für den Verkaufspreis (Bruttopreis) haben. Für Kleidung ist der Bruttowert immer die Hälfte des Bruttopreises, für Bücher 80% des Bruttopreises.
Überprüfen Sie Ihre Implementierung mit Tests. Stellen Sie sicher, dass die Methode `netCashEquivalents` für ein Buch mit einem Verkaufspreis von 1000 Cents 672 Cents zurückgibt. Stellen Sie weiterhin sicher, dass `netCashEquivalents` für ein Kleidungsstück mit einem Verkaufspreis von 2000 Cents 840 Cents zurückliefert.

Aufgabe 2 (Elektronik, 4 Punkte)
Projekt: `ex05_2`. Package: `circuit`.
Wir modellieren digitale Schaltkreise. Es gibt Komponenten, die durch Drähte verbunden sind. Die Information, ob auf einem Draht gerade eine 1 oder eine 0 ist, wird im Drahtobjekt gespeichert. Komponenten lesen ihre Eingänge (Drähte) ein und setzen ihre Ausgänge (weitere Drähte) neu.

(a) Fangen wir mit Drähten an. Ein `Wire` hat ein `boolean value`, das den logischen Wert (1 oder 0) beschreibt. Den initialen Wert übergibt man dem Konstruktor. Es gibt Getter und Setter für den Wert. Schreiben Sie eine entsprechende `Wire`-Klasse. Sie brauchen keine Tests für `Wire` zu schreiben.

(b) Alle Komponenten implementieren das Interface `IComponent`. Eine einfache Komponente ist der Inverter. Er hat einen Eingang und einen Ausgang. An seinen Ausgang legt er die logische Negation des Wertes seines Eingangs an. Implementieren Sie ihn als Klasse `Inverter`, die das Interface `IComponent` implementiert:

```
1  /** A digital circuit component. */
2  public interface IComponent {
3    /** Compute new values for all outputs from the current state
4        of the inputs and write the values to the outputs. */
5    void updateOutputs();
6  }
```

Testen Sie mit mindestens zwei Fällen, dass der Inverter bei Aufruf von `updateOutputs()` den Zustand seines Ausgangsdrahtes korrekt neu setzt. (Ob Sie richtig testen: Kommentieren Sie vorübergehend die Zeile aus, in der der Inverter invertiert. *Beide* Tests sollten scheitern!)

(c) Es gibt einige Gatter mit zwei Eingängen und einem Ausgang: AND, OR, NAND, NOR, XOR,... Sie unterscheiden sich nur in der logischen Operation; das Lesen und Schreiben der Drähte erledigen sie alle gleich. Ein Fall für eine abstrakte Basisklasse!

Implementieren Sie zwei Gatter `AndGate` und `OrGate` mittels einer abstrakten Basisklasse `ABinaryGate`, die `IComponent` implementiert. Die Basisklasse soll zwei Eingangs-`Wire`s und einen Ausgangs-`Wire` besitzen (aber `private`!) und die Berechnung der konkreten logischen Funktion in einer abstrakten Methode `boolean computeFunction(boolean input1, boolean input2)` von der Subklasse erledigen lassen.

Testen Sie `AndGate` und `OrGate` so, dass in den Tests jedes Gatter jeden Ausgabewert mindestens einmal produziert (also mindestens vier Fälle).

Tipp: auf Booleans `x`,`y` schreibt man "x und y" als `x&y`, "x oder y" als `x|y`, "x exklusiv-oder y" als `x^y`, und "nicht x" als `!x`.

Aufgabe 3 (Streams, 4 Punkte)
Projekt: `ex05_3`. Package: `stream`.
Aus Blatt 3 kennen Sie noch `WordIterator`. In dieser Aufgabe geht es darum, wie man in der Implementation von Filtern ähnlicher Bauart mit abstrakten Basisklassen Code einsparen kann.

Holen Sie sich zunächst das Skelett von der Übungsseite – dort ist schon das Interface `IWordIterator` und die `Words`-Klasse definiert.

(a) Die Filter, die jedes Wort für sich transformieren, lassen sich als Subklassen einer abstrakten Basisklasse `ATransformFilter` mit abstrakter Methode `String transformWord(String word)` herstellen. Bauen Sie so `UpperCaseFilter` (wandelt jedes Wort in Großschreibung um) und `ReplaceWordFilter` (ersetzt ein bestimmtes bei der Initialisierung festgelegtes Wort durch ein anderes).

Testen Sie, dass Ihre konkreten Filter auf leeren und nichtleeren `WordIterator`s korrekt funktionieren.

(b) Die Filter, bei denen bestimmte Wörter unterdrückt werden, lassen sich als Subklassen einer abstrakten Basisklasse `ASkipFilter` mit abstrakter Methode `boolean shouldSkip(String word)` herstellen. Bauen Sie so `RemoveWordFilter` (entfernt jedes Vorkommen eines bestimmten Wortes) und `RemoveLongWordsFilter` (entfernt alle Wörter, die mindestens n Zeichen lang sind, für ein bestimmtes n).

Testen Sie das Verhalten von `RemoveWordFilter` auf leeren sowie auf drei interessanten nichtleeren Wortfolgen (Sie haben den Code geschrieben – wo sind die komplexen Stellen?) Testen Sie `RemoveLongWordsFilter` auf einer interessanten nichtleeren Wortfolge.

Nachholaufgaben für Thema 3: Rekursive Klassen

Mit dieser Aufgabe können Sie den Stoff von Thema 3 (dem Thema von Blatt 3) wiederholen und nochmal Punkte für Thema 3 erwerben.

Aufgabe 4 (Adventure, 4 Punkte)
Projekt: `ex03_4` Package: `adventure`
Wir bauen eine interaktive Geschichte nach Art der Bücher, in denen unten auf der Seite immer etwas steht wie "Wenn Du den schlafwandelnden Grafen ansprechen willst, blättere um zu Seite 22. Wenn Du ihn mit dem Kerzenleuchter niederschlagen willst, weiter auf Seite 106". Nur haben wir einen Computer und brauchen deshalb keine Seitenzahlen. Stattdessen geben wir auf der Konsole `1` für Ansprechen und `2` für Niederschlagen ein.

Die Geschichte ist ein gerichteter Graph von `IStoryState`-Knoten. Jeder `IStoryState` enthält ein Stück Text. Es gibt Endzustände `FinalState`, die Enden der Geschichte entsprechen. Es gibt andererseits Entscheidungszustände `ChoiceState`, in denen der Held die Wahl zwischen genau zwei Aktionen hat (die im Text des Zustands angekündigt werden sollten).

Holen Sie sich das Skelett von der Übungenseite. Dort ist schon für Ein- und Ausgabe gesorgt: die Klasse `Shell` spricht mit der Konsole und füttert einen `ILineHandler` mit den eingegebenen Zeilen. Außerdem ist in der Klasse `Main` eine auskommentierte kleine Geschichte mit vier Zuständen enthalten.

Implementieren Sie `FinalState`, `ChoiceState` sowie eine `ILineHandler`-Implementation namens `AdventureLineHandler`, so dass der auskommentierte Code der `Main`-Klasse spielbar wird. Sie dürfen dabei *nicht* `Shell`, `ILineHandler` oder die Konstruktoraufrufe in `Main` verändern.

Das Eclipse-Konsolenfenster finden Sie in "Window/Show View/Console", wenn es noch nicht irgendwo offen ist.

```
Du sitzt in der Java-Übung. Dein Programm tut nicht, was es soll.
(1) Du erzählst dem Tutor genau, warum es eigentlich funktionieren sollte.
(2) Du prüfst, ob im Internet gerade jemand Unrecht hat.
```

> 1
Du erzählst... ’Und aus der Schleife springt er dann raus, wenn keine mehr da sind oder... ach so! ich einen gefunden habe!’ Das ist es!

Du beseitigst den Fehler. Der Rest ist einfach. Draußen scheint die Sonne.

The End.