**Programmieren in Java**

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

# Java-Übung Blatt 3 (Einfache Klassen)

2011-05-16

## 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 `ex1b` ab als Datei `lustigp-ex1b.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, 23. Mai 2011, um 9.00 Uhr.

Testen heißt in diesem und in folgenden Aufgabenblättern Testen mittels JUnit-Tests.

## Aufgabe 1 (Papier, 4 Punkte)

Projekt: `ex03_1`. Package: `paper`.

Wir betrachten rechteckige Papierformate mit einer Länge und einer Breite, jeweils als `double` in Millimetern. Zusätzlich haben Papierformate einen Namen (z.B. für den Seite-Einrichten-Dialog). Man kann ein Papierformat nach Länge und Breite in Millimetern fragen. Man kann ein Papierformat auch fragen, was herauskommt, wenn man die längere Seite halbiert – die Antwort ist wieder ein Papierformat, und zwar mit Laenge $\geq$ Breite, also hochkant.

(a) Definieren Sie nach dieser Beschreibung das Interface `PaperSize` für Papierformate.

(b) Implementieren Sie das Interface für allgemeine Papiere (`GeneralPaperSize`). Als Name soll $\ell$`x`$w$ zurückgegeben werden, wobei $\ell$ und $w$ die Länge bzw Breite in Millimetern sind, etwa "`305x210`" für ein Format mit Länge 305mm und Breite 210mm. Testen Sie Namensberechnung und Halbieren. Beim Halbieren gibt es wegen der Hochkant-Regel viele interessante Fälle. Passen Sie beim `assert` auf wegen Rundungsfehlern!

(c) Die DIN-A-Reihe der Papiere sei so definiert:

- Die Länge ist das $\sqrt{2}$-Fache der Breite.
- $A0$ hat die Fläche $1m^2$.
- $A(i+1)$ ist ein halbiertes $Ai$.

Implementieren Sie `PaperSize` für A-Papiere als Klasse `ASeriesPaperSize`. Die Klasse darf als einziges Feld die Nummer innerhalb der A-Reihe enthalten. Als Name soll "`A`$n$" herausgegeben werden, wobei $n$ die Nummer innerhalb der A-Reihe ist.
Testen Sie Größen- und Namensberechnung und Halbieren.

## Aufgabe 2 (Stream Editor, 4 Punkte)

Projekt: `ex03_2`. Package: `stream`.

In dieser Aufgabe sollen Sie sich mit verketteten Objekten und dem Iterator-Pattern vertraut machen.
Sie modellieren dazu das Grundgerüst eines einfachen Stream-Editors, der eine Folge von Wörtern filtern und transformieren kann. Mit Hilfe verschiedener Filter soll dabei die Folge der Wörter modifiziert werden können. Mehrere Filter-Instanzen sollen hintereinander geschaltet werden können.
Zur Vorwärtsnavigation durch die Wortfolgen dient ein gemeinsames Iterator-Interface `WordIterator`, das nur die Operationen `hasNext` und `next` spezifiziert.
Das Interface soll wie folgt aussehen:

```java
public interface WordIterator {
  /**
   * Returns true if the iteration has more strings.
   * @return true if next would return a valid string.
   */
  public boolean hasNext();

  /**
   * Returns the next string in the iteration.
   * @return the next string if there is one
   */
  public String next();
}
```

(a) Schreiben Sie zunächst eine Klasse `Words`, die das Interface `WordIterator` implementiert. Die Klasse dient als Eingabestrom von Wörtern. Die Klasse soll einen Konstruktor enthalten, der einen `String` übergeben bekommt und diesen in die einzelnen durch Leerzeichen getrennten Wörter zerlegt. Über die einzelnen Wörter soll dann mit `hasNext` und `next` navigiert werden können.
Testen Sie alle Methoden aus `WordIterator`.

(b) Es sollen nun verschiedene Arten von Filtern realisiert werden, die mit Hilfe des `WordIterator`-Interfaces auf Eingabeströmen (wie z.B. `Words`) arbeiten und dieses Interface auch selbst implementieren (um wiederum anderen Filtern als Eingabestrom zu dienen). Die folgenden Filter sollen mindestens implementiert werden:

- Ein Filter, der die Schreibweise zu Großschreibung ändert (`UpperCaseFilter`); z.B. wird `("aaa", "bbb", "ccc")` zu `("AAA", "BBB", "CCC")`.
- Ein Filter, der ein Wort durch ein anderes ersetzt (`ReplaceWordFilter`); z.B. wird `("aaa", "bbb", "ccc")` zu `("ddd", "bbb", "ccc")`, wenn ein `ReplaceWordFilter` benutzt wird, der `"aaa"` gegen `"ddd"` ersetzt.
- Ein Filter, der ein bestimmtes Wort herausfiltern kann (`RemoveWordFilter`); z.B. wird `("aaa", "bbb", "ccc")` zu `("bbb", "ccc")`, wenn ein `RemoveWordFilter` benutzt wird, der das Word `"aaa"` herausfiltert.

Insbesondere sollen beliebige Filter miteinander kombinierbar sein.

Testen Sie alle Methoden aus `WordIterator` und verschiedene Kombinationen von Filtern auf interessanten Eingabedaten.

**Aufgabe 3** (Dungeon, 4 Punkte)
Projekt: `ex03_3`. Package: `dungeon`.
Wir betrachten eines der Computerspiele, in denen man auf Monster eindrischt, ihnen die Ausrüstung wegnimmt und diese dann trägt, um besser auf die nächsten Monster eindreschen zu können. Dabei lernen Sie immutable Datenobjekte und das Decorator-Pattern kennen.

(a) An Ausrüstungsgegenständen (`EquipmentItem`) gibt es mindestens Schwerter (`Sword`) und Rüstungen (`Armor`). Man kann jeden Ausrüstungsgegenstand nach seinem Namen fragen (`getName()`). Jedes Schwert und jede Rüstung hat einen Namen. – Setzen Sie diese Beschreibung in Interface(s) und Klasse(n) um (nach dem Muster für Vereinigungstypen in der Vorlesung).

(b) Ein `PlayerStatSheet` beschreibt die Eigenschaften eines Spielers in Form von ganzzahligen Punkten für

- Magische Fähigkeit (`magic`)
- Schaden, den der Spieler austeilt (`damage`)

- Schutzwirkung der Rüstung (`armorClass`)

Schreiben Sie zunächst eine Klasse `PlayerStatSheet` mit diesen Attributen und `get`-Methoden zum Auslesen der Attribute[1].
Die Klasse soll aber auch für jedes Attribut *attr* eine Methode `update`*Attr* haben, die einen neuen Wert nimmt und ein `PlayerStatSheet` zurückliefert, das sich vom vorigen `PlayerStatSheet` nur darin unterscheidet, dass dieses Attribut den neuen Wert hat. Lassen Sie nicht zu, dass existierende Instanzen von `PlayerStatSheet` verändert werden!
Testen Sie alle `update...`-Methoden.

(c) Ausrüstung wirkt sich auf die Eigenschaften des Spielers aus. Erweitern Sie `EquipmentItem` um eine Methode `PlayerStatSheet influence(PlayerStatSheet pss)`. Schwerter erhöhen `damage`, Rüstungen erhöhen `armorClass`. Um wieviele Punkte, soll jeweils als Konstruktorparameter angegeben werden können.
Testen Sie, dass Schwerter und Rüstungen das neue `PlayerStatSheet` korrekt berechnen.

(d) Special Items sind Ausrüstungsgegenstände, die neben ihrer normalen Wirkung noch den Wert für `magic` um ein paar Punkte erhöhen und außerdem einen Namenszusatz tragen. Der komplette Name hat immer die Form "$X$ of the $Y$", wobei $X$ der Name des Gegenstands ist und $Y$ irgendetwas Schönes (z.B. "Chainmail of the Zodiac" statt nur "Chainmail"). Wir wollen, dass jeder gegenwärtige *und zukünftige* Gegenstand special sein kann. Gleichzeitig wollen wir die genannte Special-Item-Funktionalität nur einmal implementieren müssen.
Schreiben Sie eine Klasse `SpecialItemDecorator`, die das `EquipmentItem`-Interface implementiert. Sie soll ein inneres `EquipmentItem` besitzen und dessen Verhalten um das genannte Special-Item-Verhalten ergänzen.
Testen Sie Ihren `SpecialItemDecorator`.

(e) Nun bauen wir noch den Spieler (`Player`). Der Einfachheit halber kann man ihn nur nach seinem effektiven `PlayerStatSheet` fragen.
Ein nackter Spieler ist ein Spieler, und wenn man einem Spieler einen Gegenstand umhängt, ist das Ergebnis auch ein Spieler, aber mit entsprechend verbesserten Eigenschaften. – Setzen Sie diese Konstruktion nach dem Vorbild der rekursiven Klassen in der Vorlesung um und testen Sie, dass nackte und ausgerüstete Spieler ihre `PlayerStatSheet`s korrekt berechnen.

[1] Probieren Sie unbedingt einmal die Eclipse-Funktionen "Source/Generate Getters and Setters" und "Source/Generate Constructors using Fields" aus.