
Programmieren in Java

<http://proglang.informatik.uni-freiburg.de/teaching/java/2013/>

Java-Übung Blatt 4 (Abstraktion)

2013-05-07

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Identifier und Kommentare bitte auf *englisch*!
- Schreiben Sie *sinnvolle* Kommentare
- Laden Sie Ihre Lösungen mit subversion (svn) ins Übungssystem hoch. Den entsprechenden Pfad finden Sie online.
- Das Übungssystem kann überprüfen, ob Sie Ihr Quelltext den Anforderungen genügt und ob Sie alle Klassen erstellt haben, etc. Nutzen Sie dies!
- Sollte das Übungssystem Ihre Lösungen ablehnen, dann werden sie *nicht* korrigiert! Akzeptanzkriterien:
 - Compiliert erfolgreich
 - Checkstyle bringt keine Fehler
 - Alle Packages, Klassen, Interfaces, Methoden, Typen, Argumente sind exakt wie auf dem Übungsblatt gefordert.
- Ihr korrigierender Tutor wird die Korrektur Ihrer Abgabe in Ihr svn-Repository unter dem Namen `Feedback-<login>-ex<XX>.txt` comitten. (<login> ist dabei Ihr myAccount name und <XX> die Kennziffern des Übungsblatts)
- Zusätzlich können Sie Ihre Gesamtpunktzahl im Übungsportal einsehen.

Abgabe: Freitag, 17. Mai 2013, um 23.59 Uhr.

Aufgabe 1 (Inventur, 14 Punkte)

Projekt: `ex04_1`. Package: `controller`.

In dieser Aufgabe sollen Sie abstrakte Klassen einsetzen, um das vorgegebene Design zu verbessern und Redundanzen zu vermeiden. Verwenden Sie als Ausgangspunkt das Projekt aus `ex04.zip`.

Das Thema ist die Unterstützung verschiedener Controller für ein Video-Spiel. Im Projekt befinden sich innerhalb des Pakets `controller` die Klassen `Keyboard` und `Mouse`, die das Interface `IController` implementieren. Instanzen von `IController` repräsentieren angeschlossene Eingabe-Geräte. Die Spiel-Engine erwartet drei Kommandos vom Spieler: `LEFT`, `RIGHT` und `FIRE`. Diese werden in regelmäßigen Abständen für jeden Spieler abgefragt. Das Kommando `NONE` besagt, dass keine neuen Eingaben vorliegen.

Die beiden Eingabegeräte unterscheiden sich vor allem in der Art und Weise welche Daten ihre „Sensoren“ liefern.

- (a) Vor dem Refactoring sollen Sie Tests schreiben, um sicherzustellen, dass die neue Implementierung die gleichen Ergebnisse liefert wie die alte.
- (b) Benutzen Sie eine abstrakte Basisklasse um gemeinsam nutzbare Funktionalität der beiden Controller zusammenzufassen.
- (c) Implementieren Sie noch zwei zusätzliche Controller und vermeiden Sie duplizierten Code:

- `Joystick` mit einer Sensormethode `updatePosition(int excursion, int angle)`, die die Joystickposition als Polarkoordinaten kodiert (Millimeter Auslenkung von der Mitte und Richtung in Grad). Der Joystick verfügt auch über Knöpfe. Die Schwelle, nach der ein Bewegungs-Kommando an die Spiel-Engine weitergereicht wird soll 50mm sein.
- `TwoPlayerKeyboard`, eine Tastatur, die zwei Spieler unterstützt, indem jedem Spieler andere Tasten zugeordnet werden.

Weiter Details können Sie den Interfaces `IJoystick` und `ITwoPlayerKeyboard` aus `ex04.zip` entnehmen.

Aufgabe 2 (Adapter, 5 Punkte)

Projekt: `ex04_2`. Package `menuadapter`

Letzte Woche haben Sie in Aufgabe `ex03_2` eine Speisekarte modelliert. Ihr (imaginärer) Chef möchte die Implementierung gegen eine externe Testsuite validieren, von der Sie noch nichts gewusst haben. Sie finden diese in `ex04.zip` im Package `menuadapter`. Implementieren Sie nun einen *Adapter* für Ihre Implementierung von letzter Woche, so dass die Testsuite ausgeführt werden kann und alle Tests durchlaufen: Lassen Sie ihre Implementierung im Package `menu` unverändert und benutzen Sie sie um die vom den Tests verlangten Interfaces zufriedenzustellen. Bessern Sie danach eventuelle Fehler aus, die von der Testsuite aufgezeigt werden. Anstatt Ihrer eigenen Abgabe können Sie auch die Vorlage aus dem Package `menuexample` Verwenden (benennen Sie das Package in `menu` um).

Tip: Hier ist weniger Programmierkönnen, als geschickter Umgang mit Eclipse gefragt.

Aufgabe 3 (Undo, 14 Punkte)

Projekt: `ex04_3`. Package: `undo`.

Implementieren Sie eine kleine Tabellenkalkulation mit Undo-Funktion.

- (a) Bauen Sie zunächst eine Klasse `Sheet`, die eine (eindimensionale) Tabelle von `int`-Werten repräsentiert. Man soll `Sheet` mit der Anzahl Zellen instanziiieren können. `Sheet` braucht Methoden um Modifikationen an gegebenen Indizes vornehmen zu können. Außerdem soll es möglich sein aus einer Zelle den Wert auszulesen (`int get(int index)`). Bei ungültigen Indizes darf die Klasse sich beliebig verhalten.

Des Weiteren soll `Sheet` die Inhalte aller Zellen, mit einem Trennzeichen getrennt, in einen String umwandeln können (`String display()`) (Tipp: `StringBuilder` kann hilfreich sein). Das Trennzeichen soll verstellbar sein.

Testen Sie `Sheet`.

- (b) Das Tabellenkalkulationsprogramm ist eine Klasse `SpreadSheetApp`.

```
1 package undo;
2 class SpreadSheetApp {
3     public SpreadSheetApp(Sheet s) { /* ... */ }
4     public void put(int cell, int value) { /* ... */ }
5     /**
6      * Set the separation character for display.
7      * @param sep the separation character.
8      */
9     public void setSeparator(char sep) { /* ... */ }
10    /**
11     * @return a String representation of the spreadsheet.
12     */
13    public void undo() { /* ... */ }
14    public String display() { /* ... */ }
15    /**
16     * Set the size of the undo history.
17     * @param n number of actions that can be undone (0 for arbitrarily many).
18     */
19    public void setHistorySize(int sz) { /* ... */ }
20 }
```

Sie lässt den Zugriff auf ihr inneres `Sheet` nur über Methoden `put` und `setSeparator` zu, die vorher auf geeignete Weise Undo-Information aufbewahren. Implementieren Sie die Klasse. Finden Sie einen Weg, eine Undo-Funktionalität anzubieten. Es soll wählbar sein, wie viel Undo-Information gespeichert wird (`setHistorySize`). Für Teilpunkte reicht ein Undo für eine einzige Aktion.