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

Java-Übung Blatt 5 (Swing & ein bisschen Vererbung)

2013-05-14

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, 31. Mai 2013, um 23.59 Uhr.

Aufgabe 1 (Bäume, 14 Punkte)Projekt: `ex05_1`. Package: `trees`.

In dieser Aufgabe sollen Sie Binärbäume als Java-Objekte repräsentieren und darstellen.

Ein Baum (`ITree`) kann sein:

- ein Blatt `Leaf` (enthält keine Daten)
- ein innerer Knoten `InnerNode`, in dem ein `int`-Wert gespeichert ist und an dem ein rechter und ein linker Teilbaum (jeweils `ITree`) hängen.

Bäume können ihre Höhe berechnen¹ und sich auf ein `ICanvas` zeichnen (siehe unten).

```

1 package trees;
2 interface ITree {
3     /**
4      * @return the height of this tree.
5      */
6     public int height() { /* ... */ }
7     /**
8      * Draws this tree within the specified rectangular area.
9      * @param x the topLeft corner's x coordinate
10     * @param y the topLeft corner's y coordinate
11     * @param width the
12     * @param height the height of the area to draw in
13     * @param canvas the canvas to draw this node's subtree in
14     */
15     public void draw(int x, int y, int width, int height, ICanvas canvas) { /* ... */ }
16 }

```

- (a) Schreiben Sie Klassen für Blätter und für Innere Knoten. Testen Sie für beide die Höhenberechnung.

¹nach der üblichen Definition, siehe z.B. [http://en.wikipedia.org/wiki/Tree_\(data_structure\)#Terminology](http://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology)

- (b) Implementieren Sie die **draw**-Methoden. Beim Aufruf wird ein Rechteck übergeben (durch linke obere Ecke, Breite und Höhe) und ein **ICanvas** object. In diesem Rechteck sollen die Knoten und Kanten des Baumes so gezeichnet werden, dass er die Grenzen des Rechtecks nicht überschreitet und sich keine Knoten oder Kanten überlappen. Außerdem soll der Baum „vernünftig“ zu erkennen sein. Sollte dies alles aufgrund der Größe des Baumes nicht möglich sein, soll anstatt des Baumes eine „Fehlermeldung“ gezeichnet werden; das **ICanvas** Interface stellt eine entsprechende Methode zur Verfügung. Sorgen Sie aber zumindest dafür, dass es möglich ist einen balancierten Baum mit kleiner oder gleich sieben Knoten auf einem Rechteck der Größe 600×600 Pixel zu zeichnen.

Testen Sie die Darstellung ausnahmsweise durch Ausprobieren mit einer Main-Klasse. Die **draw** Methode und die Methoden aus dem Package **draw** müssen *nicht* mit JUnit getestet werden. Beim Aufruf der Main Methode sollen zwei balancierte Bäume mit jeweils mindestens fünf Knoten angezeigt werden.

Einige Anmerkungen zu **ICanvas**:

```
1 package draw;
2 interface ICanvas {
3     public void drawCircle(ICircle circle) { /* ... */ }
4     public void drawLine(ILine line) { /* ... */ }
5     /**
6      * Abort drawing with an error message.
7      */
8     public void abort(String message) { /* ... */ }
9 }
```

Im Skelett zu dieser Aufgabe finden sie das **ICanvas** Interface im Package **draw**, zusammen mit einer Implementierung **Canvas**. Dort finden Sie ebenfalls die Interfaces **ICircle** und **ILine**. Im Package **tree** befindet sich auch eine main-Methode, die Sie vervollständigen sollen. Sie enthält aber bereits alles Nötige, um Grafiken darzustellen. Versuchen Sie trotzdem zu verstehen, was passiert (nützlich für Aufgabe 3). (Falls Sie Lust zum experimentieren haben, können Sie die Details der grafischen Darstellung nach Ihren Vorlieben anpassen (Klasse **Canvas**), solange die Bäume noch korrekt und erkennbar dargestellt werden).

Aufgabe 2 (Elektronik, 8 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 package circuit;
2 /** A digital circuit component. */
3 interface IComponent {
4     /**
5      * Compute new values for all outputs from the current
6      * state of the inputs and write the values to the outputs.
```

```

7      */
8      public void updateOutputs() { /* ... */ }
9  }

```

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-Wires 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).

- (d) Implementieren Sie eine Klasse `HalfAdder` welche **intern die schon vorhandenen Klassen nutzt**² (`AndGate`, `OrGate`, ...). Ein `HalfAdder` hat zwei Eingänge `x` und `y` und zwei Ausgänge `sum` und `carry`. Dabei soll folgende Funktionalität implementiert werden:

x	y	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Testen Sie ob Ihre Implementierung funktioniert.

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 (Adventure, 11 Punkte)

Projekt: `ex05_3` 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 erschrecken willst, weiter auf Seite 106”. Nur haben wir einen Computer und brauchen deshalb keine Seitenzahlen. Stattdessen möchten wir die Auswahl mit Buttons treffen.

In dieser Aufgabe verwenden wir das GUI-Framework Swing welches in Java integriert ist³. Es baut auf dem Model-View-Controller (MVC) Entwurfsmuster auf. Eine Anwendung wird dabei in drei Teile unterteilt: Dabei bezeichnet Model die Daten(-modelle), View bezeichnet die Anzeige von Daten. Der Controller verknüpft Model mit View, z.B. “wenn dieser Button auf der Anzeige gedrückt wird, dann zeige neue Daten an”.

- (a) Package: `adventure.model`

Unser Datenmodell besteht aus der Geschichte, welche wir abbilden wollen. 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

²Sie können auch ein XOR-Gatter implementieren und dieses verwenden. Dann müssen Sie dies aber auch testen.

³Sollten Sie noch ein Tutorial benötigen, schauen Sie hier: <http://docs.oracle.com/javase/tutorial/uiswing/start/index.html>

der Spieler die Wahl zwischen mehreren Aktionen hat, welche im Text des Zustands angekündigt werden sollten. Speichern Sie die weiteren Aktionen in einer Liste (`java.util.List`)

In der Projektvorlage für das Übungsblatt ist ein Klasse `ExampleStory` enthalten, die eine auskommentierte kleine Geschichte enthält, welche Sie verwenden können.

Implementieren Sie `IStoryState`, `FinalState`, `ChoiceState`, sodass der auskommentierte Code in `ExampleStory` erfolgreich kompiliert.

(b) Package: `adventure.viewcontroller`

Die Anzeige und der Controller wird in Swing meist zusammen implementiert indem anonyme Klassen benutzt werden.

Implementieren Sie eine Klasse `StoryWindow` welche von `JFrame` ableitet und ein Fenster erzeugt, welches eine `JEditorPane` enthält um den Text der Geschichten aufgrund eines `IStoryState` darzustellen. Weiterhin sollte das Fenster auch eine entsprechende Anzahl Buttons haben, je nachdem wie viele Forsetzungsmöglichkeiten die Geschichte am aktuellen Punkt gerade hat.

In der Vorlage findet sich eine Klasse `Main` die eine leere Methode `run` enthält. Schreiben Sie in dieser Methode den Quelltext zur Erzeugung und zum Anzeigen Ihres Fensters. Die Startklasse des Projektes ist `Adventure`.

(c) Schreiben Sie nun `ActionListener` für die Buttons, die dafür sorgen, dass das Fenster bei einem Click auf einen Button zum jeweils nächsten `IStoryState` wechselt. Ob Sie dabei ein neues Fenster erzeugen, oder das Alte wiederverwenden ist egal.

Hinweise:

- Aus den Containerklassen wie `JPanel`, etc. können auch Komponenten wieder entfernt werden (`remove`, bzw. `removeAll`).
- Testen: ausnahmsweise genügt diesmal Ausprobieren.

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! Wenn, ich ein Element gefunden habe!' Das ist es!

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

The End.