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

Java-Übung Blatt 6 (Testen mit Zustand, Swing, Events)

2013-05-28

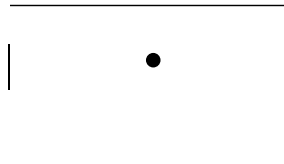
Hinweise

- 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, 07. Juni 2013, um 23.59 Uhr.

Aufgabe 1 (Pong-Zustand, 10 Punkte)Projekt: `ex06_1`. Package: `pongstate`.

In dieser Aufgabe implementieren Sie ein 1-Spieler Pong¹. Das Spiel besteht aus einem zweidimensionalen Spielfeld, welches einen Ball und einen Schläger enthält.



Der Schläger ist vertikal am linken Spielfeldrand verankert, hat eine gewisse Höhe und kann nach oben und unten innerhalb des Spielfeldes bewegt werden.

Der Ball hat einen gewissen Durchmesser, und wird von den Rändern des Spielfeldes oben, unten und rechts abgestossen. Auf der linken Seite wird er nur am Schläger abgestossen. Trifft er den Schläger nicht, ist das Spiel vorbei.

Das Spiel selbst hat zwei verschiedene Spielzustände: (1) das Spiel läuft gerade, (2) das Spiel ist beendet (GameOver). Weiterhin müssen im Spiel die Position des Balls (x, y), die aktuelle Bewegungsrichtung des Balles (dx, dy) und die aktuelle Position des Schlägers (nur y) gespeichert werden. Außerdem müssen die Spielfeldmaße und die Schlägermaße gespeichert werden.

Mögliche Zustandsänderungen, welche nur im 1. Zustand durchgeführt werden können, sind:

- Der Ball bewegt sich um einen Schritt bzgl. der aktuellen Bewegungsrichtung. Die Spielfeldmaße müssen hier berücksichtigt werden, d.h. stößt der Ball an eine Wand, ändert er seine Bewegungsrichtung entsprechend. Beispiel: Wenn der Ball an den oberen Spielfeldrand anstößt, dann wird dy zu $-dy$. Der Ball darf das Spielfeld nicht verlassen!
- Der Schläger wird um eine gewisse Distanz nach oben/unten bewegt werden. Der Schläger darf das Spielfeld aber natürlich nicht verlassen.

¹Pong ist eines der ersten Computerspiele gewesen. <http://de.wikipedia.org/wiki/Pong>

- Der Zustand wechselt zu GameOver wenn der Ball links aus dem Spielfeld austritt ohne den Schläger zu berühren.

Befindet sich das Spiel im GameOver-Zustand, dürfen keine der gerade genannten Zustandsänderungen möglich sein.

Implementieren Sie die verschiedenen Zustände und Ihre Änderungen in Schnittstellen und Klassenhierarchien in einem Package `pongstate`. Erweitern Sie hierzu das gegebene Interface `pongstate.IState` aber verändern Sie keine der vorhandenen Methoden.

Testen² Sie die Zustandsklasse(n) und überprüfen Sie so, dass sich der Zustand so verhält wie sie es erwarten. Dies betrifft vor allem den Kontakt des Balles mit den Spielfeldrändern und dem Schläger. Achten Sie darauf, alle Fälle abzudecken, u.a.:

- Kontakt mit dem Rand: oben, unten, rechts
- Kontakt mit dem Schläger: am oberen Ende, in der Mitte, am unteren Ende.
- Schläger wird verfehlt.

Ein Beispieltestfall ist schon im Paket `pongstateexampletest` vorhanden.

Aufgabe 2 (Pong-Interaktion, 13 Punkte)

Projekt: `ex06_2`. Package: `pong`.

In dieser Aufgabe wird das Pong Spiel, das in Aufgabe `ex06_1` begonnen wurde fertiggestellt. Wenn Sie Aufgabe `ex06_1` nicht bearbeitet haben, können Sie die Mock-Implementierungen aus dem Package `pongmock`, das Sie im Template finden, benutzen. Diese verhält sich allerdings nicht ganz korrekt!

1. Grafische Darstellung (package `pongview`)

Nun fügen wir dem ganzen eine grafische Oberfläche hinzu. Ein Beispiel dafür finden Sie im Template unter `pongexample`.

- Legen Sie eine Klasse `PongPanel` an, welche von `JPanel` erbt. Die Klasse soll für einen gegebenen Spielzustand ein Spielfeld zeichnen. Zunächst zeichnen wir nur den Hintergrund. Dazu muss die Methode `protected void paintComponent(Graphics g)` von `JPanel` überschrieben werden. In dieser Methode kann dann gezeichnet werden:

- Kreise `g.drawOval(int x, int y, int width, int height);`
- Linien `g.drawLine(int x1, int y1, int x2, int y2);`
- Rechtecke `g.drawRect(int x, int y, int width, int height);`

Jede dieser Methoden (außer `drawLine`) gibt es auch in einer fill-Variante (e.g. `fillRect`) bei der die entsprechende geometrische Figur mit der aktuellen Farbe gefüllt wird.

Die aktuelle Farbe setzt man mit `g.setColor(Color.black);`

Weitere Hinweise zu den Zeichnen-Methoden finden sich in der API-Beschreibung³.

- Schreiben Sie die Klasse `PongPanel` so, dass sie einen aktuellen Spielzustand nimmt (Konstruktor und Setter) und diesen zeichnen kann, d.h. den Ball und den Schläger an den aktuellen Positionen. Ob sie noch weitere Dinge wie Hintergrund, Wände etc. zeichnen ist Ihnen überlassen.
- Fügen Sie nun in die Klasse `PongWindow` das neue `PongPanel` ein. Auch hier finden Sie ein Beispiel unter `pongexample`. Erstellen Sie einen initialen Spielzustand und zeichnen Sie diesen. Wenn Sie nun die Klasse ausführen, sollten Sie den initialen gezeichneten Spielzustand sehen.

²Junit

³<http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/Graphics.html>

2. Timer

Um dem Spiel Dynamik zu verleihen, muss der Ball sich regelmäßig fortbewegen. Wir lösen dieses Problem mit einem Timer. Ein Timer sorgt dafür, dass eine Methode alle n Millisekunden aufgerufen wird. Das n ist hierbei bestimmbar.

Ein Timer wird wie folgt benutzt:

```
1 import java.awt.event.ActionListener;
2 import javax.swing.Timer;
3
4 class TimerExample implements ActionListener {
5     private Timer timer;
6     MyClass() {
7         // Create new timer which is fired every 100ms
8         Timer timer = new Timer(100, this);
9         timer.start();
10    }
11
12    @Override
13    public void actionPerformed(ActionEvent arg0) {
14        System.out.println("Timer is fired");
15    }
16 }
```

Als ersten Parameter nimmt der Timer-Konstruktor ein Zeitintervall in Millisekunden. Dies gibt an, wie lange zwischen den einzelnen Aufrufen gewartet wird. Als zweiten Parameter wird eine Klasse, die `ActionListener` implementiert, erwartet. Deren Methode `public void actionPerformed(ActionEvent arg0)` wird alle n Millisekunden aufgerufen. Im obigen Beispiel wird alle 100ms **Timer is fired** auf der Konsole ausgegeben.

Nutzen Sie diese Funktionalität um den Ball einen Schritt zu bewegen. Sie müssen hier vermutlich mit dem Zeitintervall und der Schrittweite ein wenig experimentieren. Die Methode `repaint()` Ihres Panels kann zum neu zeichnen des Inhaltes genutzt werden.

Zum Timer findet sich auch ein Beispiel unter `pongexample`.

3. Steuerung mit Tasten

Zum Schluss möchten wir natürlich, dass der Benutzer den Schläger auch steuern kann. Dies wird in Swing mit einem `KeyListener` implementiert. Das entsprechende Grundgerüst findet sich schon im Template. Ändern Sie den Zustand bei einem entsprechenden Tastendruck. Die Methode in der das geht heißt `keyPressed` und findet sich in der Klasse `PongWindow`.

Diese Aufgabe ist mit Absicht etwas ungenauer spezifiziert als bisher. Nutzen Sie frühzeitig die Übungsgruppen, die Java API und das Forum wenn Sie fragen haben. Ansonsten sind Sie hier relativ frei in Ihrer Implementierung.

Aufgabe 3 (Chat Terminal, 10 Punkte)

Projekt: `ex06_3`. Package: `terminal`.

Anstatt nur auf Events zu reagieren sollen in dieser Aufgabe auch Events generiert werden. In Chat-Programmen beispielsweise wird einem oft angezeigt, ob der Chat-Partner gerade am Schreiben einer Nachricht ist. Dieses Feature erfordert, dass der Client des Partners Events generiert, die den Start und das Ende des Tippvorgangs markieren.⁴

Die gewünschten Events sind durch das Interface `ITypingListener` definiert:

```
1 package terminal;
2
3 /**
```

⁴Diese Events werden schließlich über das Netz vermittelt; wie das geht interessiert uns für diese Aufgabe nicht

```

4  * Listener for terminals that accept character entry
5  */
6  public interface ITypingListener {
7      /**
8       * invoked when the user starts typing on the terminal
9       */
10     public void onStartTyping();
11
12     /**
13      * invoked when the user stops typing on the terminal
14      *
15      * @param duration
16      * The time since typing started in ms (tolerance 100 ms)
17      * @param newText
18      * The string of characters that was typed since starting typing.
19      * @param allText
20      * The complete text that is buffered in the terminal since the
21      * last submit.
22      */
23     public void onStopTyping(long duration, String newText, String allText);
24
25     /**
26      * invoked when the user has submitted a message
27      *
28      * @param s
29      * the complete message that was submitted
30      */
31     public void onSubmit(String s);
32
33 }

```

Für unsere Zwecke beginnt das Tippen wenn gerade nicht getippt wurde und ein Zeichen eingegeben wird. Das Tippen endet, wenn etwa 1500 ms lang (100 ms Toleranz sind akzeptabel) kein Zeichen mehr eingegeben wurde. Beachten Sie ansonsten die Kommentare aus dem Template. Im Template sind zwei `ITypingListener` implementiert die auf Tipp-Start und Tipp-Ende, sowie auf das Absenden einer Nachricht reagieren: `CountSmileys` zählt die in der aktuellen Nachricht auftauchenden Smileys und `TypingStats` berechnet simple Statistiken über das Tippverhalten.

Ziel der Aufgabe ist es die Events für die `ITypingListener` korrekt zu generieren. Wir gehen davon aus, das dazu vom „Betriebssystem“ Events für Timeout und die Eingabe von Zeichen zur Verfügung gestellt werden:

- Timeouts funktionieren ähnlich der Swing-Timer aus `ex06_2`. Siehe Interface `terminal.ITimer` im Template.
- Um auf Timeouts zu reagieren, muss das `terminal.ITimer.Handler` Interface implementiert werden.
- Auf Zeicheneingabe und Senden der Nachricht kann über die Implementierung des `terminal.ICharBuffer` Interfaces reagiert werden.

Näheres können Sie dem Template zu dieser Aufgabe entnehmen. Die abstrakte Basisklasse für Chat Terminals heißt `terminal.ATerminal`.

1. Implementieren Sie eine von `terminal.ATerminal` abgeleitete Klasse, die es erlaubt `ITypingListener` zu registrieren und die die entsprechenden Events korrekt generiert. Testen Sie ihre Implementierung mit JUnit. Insbesondere für das zeitliche Verhalten sollen Sie mittels Mock-Objekten künstliche Test-Eingaben verwenden. Es gibt Punktabzug, wenn sich Ihre Implementierung von `ATerminal` nicht vollständig und unabhängig von Systembibliotheken und externen Bibliotheken (wie z.B. Swing) testen lässt.

Die Methode `getAllText` in `onStopTyping` können Sie abstrakt belassen und bei den Tests Mock-Implementierungen verwenden. Alle andere Funktionalität von `terminal.ATerminal` soll von Ihnen implementiert werden.

2. Probieren Sie die beiden `ITypingListener` Implementierungen aus, indem Sie Ihre `ATerminal` Implementierung an eine GUI anbinden. Weiter Tests sind nicht nötig. Sie können die Swing Komponente aus dem Package `terminal.view` benutzen und nach Ihren Wünschen anpassen, oder eine eigene Komponente implementieren. Die Funktionalität muss aber der gegebenen Komponenten entsprechen. **Hinweis:** die `getKeyChar` Methode der `java.awt.event.KeyEvent` Klasse ist bei der Umrechnung von Tastatur-Tasten zu Zeichen nützlich.