252-0027-00: Einführung in die Programmierung

Übungsblatt 9

Abgabe: 26. November 2019, 10:00

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus.

Aufgabe 1: Klassen Rätsel (Bonus!)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

In dieser Aufgabe sollen Sie zeigen, dass Sie mit Klassen und Vererbung umgehen können. Im Anhang A finden Sie ein Programm, welches Instanzen von Klassen erstellt und Methoden aufruft. Das Programm macht nichts sinnvolles und dient nur dem Testen ihrer Fähigkeiten. Die Ausgabe des Programms soll am Ende aussehen, wie im Anhang B gezeigt. In der Datei "Klassen.java" finden Sie das Programm aus Anhang A. Zusätzlich enthält "KlassenTest.java" einen Unit-Test, welcher prüft, ob der Output des Programms dem Output aus Anhang B entspricht.

Erstellen und implementieren Sie alle Klassen und Methoden, welche dafür notwendig sind, dass das Programm kompiliert und den korrekten Output ausgibt. Diese Klassen müssen sich in einer oder mehreren zusätzlichen Datei(en) (im "src"-Ordner) befinden, *nicht* in der "Klassen.java"-Datei. Die "Klassen.java"-Datei dürfen Sie *nicht* verändern. In dieser Aufgabe können Sie auch Teilpunkte für eine nicht compilierende "Klassen.java"-Datei erhalten. Beachten Sie jedoch, dass die Datei(en), in welchen sich Ihre Klassen befinden, keinerlei Kompilierfehler enthalten dürfen, da Sie sonst keine Teilpunkte erhalten.

Tipp: Konzentrieren Sie sich zuerst darauf, dass das Programm compiliert *ohne* den Output zu beachten. Passen Sie danach die Implementierung an, damit der Output passt.

Aufgabe 2: Umkehrung

Auf einem vorherigen Übungsblatt haben Sie eine Linked List für Integer implementiert. In dieser Aufgabe fügen Sie dieser `LinkedIntList` eine weitere Methode hinzu, welche die Liste umkehrt. Eine Liste gilt als umgekehrt, wenn für jedes Paar von Nodes `a` und `b`, für welche zuvor `a == b.next` gegolten hat, in der neuen (umgekehrten) Liste `b == a.next` gilt. Zusätzlich entspricht

nach der Umkehrung die erste Node der neuen Liste, der letzten Node der ursprünglichen Liste (und umgekehrt).

Vervollständigen Sie die Methode `reverse()` in der Klasse `LinkedIntList`. Die Methode soll, wie oben definiert, die Liste umkehren. Achten Sie darauf, dass Sie wirklich die Reihenfolge der Nodes selbst umkehren. Es reicht nicht aus, die Reihenfolge der enthaltenen `int`-Werte umzukehren. Es müssen auch in der umgekehrten Liste dieselben Instanzen von `IntNodes` wie in der ursprünglichen Liste verwendet werden. Erstellen Sie also *keine* neuen `IntNodes` mit `new IntNode()`. In der Datei "UmkehrungTest.java" finden Sie einen einfachen Test.

Aufgabe 3: Miles and More

In dieser Aufgabe sollen Sie zeigen, dass Sie Ein- und Ausgabe mit Dateien und vor allem die `Scanner`-Klasse beherrschen. Sie sollen ein Programm schreiben, welches eine Liste mit getätigten Flügen von verschiedenen Personen aus einer Datei einliest, die Bonus-Meilen für jede Person berechnet und diese in eine neue Datei schreibt. Eine Datei mit Flügen sieht z.B. so aus:

```
Michaela Meier
LX326 05.12.2016 ECONOMY
LX317 10.01.2017 ECONOMY
A3851 12.05.2017 BUSINESS
LX8 12.10.2017 FIRST 4433
.
Stefan Oliver Schmid
LX4150 19.10.2017 BUSINESS 6404
.
```

Nach einer Zeile mit dem Namen einer Person folgen Zeilen mit Flügen der Person und schliesslich eine Zeile mit einem einzigen Punkt. Jede Flugzeile enthält die Flugnummer, das Datum, die Flugklasse und, falls es sich um einen Interkontinentalflug handelt, die Distanz des Fluges (in Meilen). *Das genaue Format einer solchen Datei finden Sie als EBNF-Beschreibung in Anhang C.*

Die Ausgabedatei soll für jede Person eine Zeile mit dem vollen Namen der Person und den zusammengerechneten Meilen enthalten. Für Interkontinentalflüge gilt die Distanz des Fluges, ansonsten eine Pauschale von 125 Meilen. Die Meilen jedes Flugs werden noch mit einem Faktor multipliziert, der von der Flugklasse abhängig ist: Economy: $\times 1$, Business: $\times 2$ und First: $\times 3$. Für die Beispieldatei oben sollte die Ausgabe deshalb so aussehen:

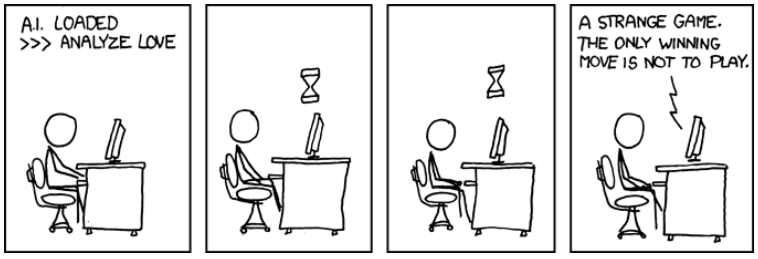
```
Michaela Meier: 13799
Stefan Oliver Schmid: 12808
```

In Ihrer Übungsvorlage finden Sie eine Klasse `Flights` mit einer `calculateMiles()`-Methode. In dieser Methode sollen Sie das Einlesen, Berechnen und Ausgeben der Meilen implementieren. Verwenden Sie die Argumente der Methode als Ein- und Ausgabedateien und ändern Sie unter keinen Umständen die Signatur von `calculateMiles()`.

`Flights` enthält auch eine `main`-Methode, welche `calculateMiles()` mit einer "flights.txt"-Datei aufruft. Wenn `calculateMiles()` richtig implementiert ist, sollten Sie nach dem Ausführen des Programms eine "miles.txt"-Datei mit der oben gegebenen Ausgabe in Ihrem Projektordner

finden. (Rechtsklicken Sie wenn nötig auf das Projekt und wählen Sie *Refresh*.) Die `main`-Methode sowie die beiden Text-Dateien sind irrelevant für die Abgabe; es zählt einzig `calculateMiles()`.

Tipps: Überlegen Sie sich, welche der `Scanner.next...()`-Methoden sich für diese Aufgabe eignen. Beachten Sie ausserdem, dass sich `nextLine()` im Zusammenhang mit anderen Methoden nicht immer intuitiv verhält: Wenn z.B. mit `next()` schon ein Wort einer Zeile eingelesen wurde, gibt `nextLine()` den Rest dieser Zeile *inklusive aller Leerzeichen* zurück. Falls `next()` zuvor das *letzte* Wort der Zeile gelesen hat, kann `nextLine()` also auch einen leeren String zurückgeben!



xkcd: Game Theory by Randall Munroe (CC BY-NC 2.5)

Aufgabe 4: Künstliche Intelligenz für das Ratespiel

In Übung 5 implementierten Sie ein Spiel, in welchem der Computer ein Wort auswählt und der Spieler dieses erraten muss. Dort war der Spieler der Benutzer des Programms. In dieser Aufgabe sollen Sie verschiedene "künstliche" Spieler entwickeln. Das heisst, anstelle des Menschen, der über die Konsole Tipps eingibt, werden die Tipps von (mehr oder weniger "intelligenten") Programmen abgegeben. Ihr Ziel ist es, einen künstlichen Spieler zu entwickeln, der über mehrere Spiele hinweg die Wörter in so wenigen Versuchen wie möglich errät.

Die Übungsvorlage enthält bereits den Code für das Rate-Spiel. Gegenüber Übung 5 ist dieser nun in verschiedene Klassen aufgeteilt. Die drei Hauptklassen sind `RateSpiel`, `Computer` und `Spieler`. Die Klasse `RateSpielApp` enthält eine `main()`-Methode, welche das Spiel aufsetzt und durchführt. Durch die Aufteilung ist es möglich, mittels Vererbung Spieler mit unterschiedlichem Verhalten zu schreiben. Die Klasse `Spieler` enthält nämlich nur die Deklarationen der benötigten Methoden aber keine (sinnvolle) Funktionalität. Subklassen von `Spieler` überschreiben diese Methoden und definieren damit das Verhalten eines Spielers.

Ein konkreter Spieler ist ebenfalls schon in der Vorlage vorhanden: der `KonsolenSpieler`. Dieser besitzt allerdings keine eigene "Intelligenz", sondern holt sich die Tipps über die Konsole vom Benutzer. Ein `RateSpiel` mit einem `KonsolenSpieler` verhält sich also so wie das Spiel in Übung 5. Starten Sie die `RateSpielApp` und überzeugen Sie sich selbst[1].

a) Schreiben Sie als erstes eine Klasse `ZufallsWortSpieler`, welche einen Spieler implementiert, der in jeder Runde zufällig ein Wort aus der Liste der verwendeten Wörter tippt. Die Klasse soll von `Spieler` erben und die benötigten Methoden überschreiben.

   In der `neuesSpiel()`-Methode, welche immer zu Beginn eines Spiels vom `RateSpiel` aufgerufen wird, soll sich der Spieler das Array der im Spiel verwendeten Wörter merken. Speichern

[1] Beachten Sie, dass die Wörter-Datei jetzt 500 Wörter enthält!

Sie eine Referenz dazu in ein `woerter`-Feld. Mit der `gibTipp()`-Methode gibt der Spieler seinen nächsten Tipp ab. Überschreiben Sie diese Methode, so dass sie einen zufälligen Index für das `woerter`-Array (siehe `Random.nextInt(int)`) erzeugt und das entsprechende Wort zurück gibt. Das Generieren des Index sollen Sie in einer separaten Methode `zufallsWortIndex()` implementieren. (Sie sehen später weshalb.) Überschreiben Sie ausserdem `name()`.

Ändern Sie jetzt das `RateSpielApp`-Programm so ab, dass statt einem `KonsolenSpieler` ein `ZufallsWortSpieler` am Spiel teil nimmt. Erhöhen Sie ausserdem die Anzahl Spiele, die durchgeführt werden, indem Sie das Argument zur `nSpiele()`-Methode von `1` z.B. auf `1000` ändern. Sie sollten ungefähr folgende Ausgabe erhalten:

```
...
Spiel 999
Spiel 1000
Zufalls-Wort-Spieler hat durchschnittlich 495.138 Versuche benötigt.
```

b) Erstellen Sie einen zweiten Spieler, `ZufallsWortSpielerMitGedaechtnis`, der sich in jedem Spiel merkt, welche Wörter er schon ausprobiert hat. Da dieser einige Ähnlichkeit zum `ZufallsWortSpieler` hat, sollen Sie ihn als Subklasse von `ZufallsWortSpieler` entwerfen.

Das Gedächtnis des Spielers können Sie als `boolean[]`-Feld ausdrücken, welches für jedes mögliche Wort angibt, ob dieses Wort schon ausprobiert wurde. Dieses Array sollte zu Beginn jedes Spiels neu initialisiert werden. Überschreiben Sie dazu die `neuesSpiel()`-Methode vom `ZufallsWortSpieler`. **Vorsicht:** der Code in der `neuesSpiel()`-Methode von `ZufallsWortSpieler` sollte trotzdem ausgeführt werden, denn da merkt er sich ja die Liste der Wörter! Fügen Sie deshalb einen entsprechenden `super`-Methoden-Aufruf hinzu[2].

Da Sie in a) das Erzeugen des Wort-Index in einer separaten `zufallsWortIndex()`-Methode implementierten, können Sie nun diese Methode überschreiben, um das Verhalten von `gibTipp()` dieses Spielers zu ändern. (Dafür darf die Sichtbarkeit von `zufallsWortIndex()` in `ZufallsWortSpieler` nicht `private` sein). Überschreiben Sie sie so, dass sie das Gedächtnis des Spielers einbezieht. Sie müssen dafür einen zufälligen Index generieren, welcher nicht schon verwendet wurde.

Überschreiben Sie auch `name()` und ändern Sie dann `RateSpielApp` erneut, so dass beide Zufalls-Spieler spielen. Die Ausgabe sollte jetzt etwa so aussehen:

```
...
Zufalls-Wort-Spieler hat durchschnittlich 492.513 Versuche benötigt.
Zufalls-Wort-Spieler mit Gedächtnis hat durchschnittlich 242.493 Versuche benötigt.
```

c) Die beiden Zufalls-Spieler sind noch nicht wirklich "intelligent". Der Grund ist, dass sie gar keinen Nutzen aus den Hinweisen des Computers ("enthält", "enthält nicht", usw.) ziehen. Schreiben Sie deshalb einen (oder mehrere) Spieler, welche bessere Tipps abgeben und die Hinweise nützen um die Menge der noch möglichen Wörter einzuschränken. Dazu müssen Sie die `bekommeHinweis()`-Methode von `Spieler` überschreiben.

Wenn Sie verschiedene Ideen ausprobieren wollen, schreiben Sie verschiedene `Spieler`-Subklassen und vergleichen Sie sie mithilfe der `RateSpielApp`. Schauen Sie sich auch den `RateSpiel`-Konstruktor an, welcher zwei hilfreiche Parameter zur Verfügung stellt.

Wie schlägt sich ihr bester Spieler im Vergleich zu den Spielern Ihrer Mitstudierenden?

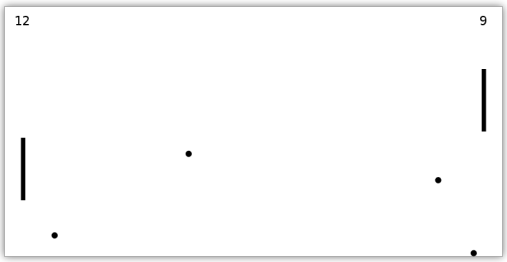
Tipps:

[2] Siehe Vorlesung "Vererbung", Folie 35

- Beginnen Sie einfach. Zum Beispiel mit einem Spieler, der zuerst alle Buchstaben des Alphabets tippt und dann die noch möglichen Wörter durchprobiert.
- Heuristiken (z.B. “e” kommt öfter vor als ”x”) sind hilfreich.
- Überlegen Sie sich, ob es so etwas wie die “optimale” Strategie gibt.

Aufgabe 5: Pong

In dieser Aufgabe geht es darum, den Klassiker Pong von 1972 nach zu bauen. Es sollen zwei Spieler gegeneinander spielen können. Dafür kommt die bereits bekannte `Window`-Klasse zum Zug. Sie bietet einige Methoden, um Input von Maus und Tastatur aufzunehmen. Eine davon ist `isKeyPressed(String keyName)`, welche `true` zurück gibt, wenn die spezifizierte Taste zum Zeitpunkt des Aufrufs gedrückt ist:




```
if(window.isKeyPressed("up")) {
    // move something around
}
```

Das Spiel besteht aus einem Ball und aus zwei Spielern, welche je einen vertikalen Balken kontrollieren und versuchen, den Ball im Spiel zu halten. Wenn der Ball das Spiel seitlich verlässt, erhält der gegenüberliegende Spieler einen Punkt. Wenn der Ball hingegen die Wände oben und unten oder einen Spielerbalken berührt, prallt er ab.

a) In Ihrem Projekt finden Sie bereits Vorlagen für folgende Klassen:

`PongGame`: Verantwortlich für das Speichern und Berechnen der Spielsituation.

`PongGUI`: Verantwortlich für die grafische Umsetzung des Spielgeschehens. Beinhaltet die `main`-Methode und ist für die graphische Oberfläche des Spiels verantwortlich.

Erstellen Sie zwei weitere Klassen `Player` und `Ball`, welche alle Informationen zu Spieler und Ball beinhalten. Ein Spieler hat eine (p_x, p_y)-Position, eine Balkenlänge und einen Punktestand. Ein Ball hat ebenfalls eine Position und zusätzlich eine (v_x, v_y)-Geschwindigkeit (in Pixel/Spielschritt).

Ändern Sie ausserdem den Konstruktor von `PongGame`, dass er zwei `Player`- und eine `Ball`-Instanz erstellt und in Feldern speichert. Der Ball soll zu Beginn des Spiels in der Mitte starten und eine zufällige Geschwindigkeit haben.

b) Ergänzen Sie `PongGui` so, dass Ball und Spielerbalken angezeigt werden und man die Balken mittels Tastatur (z.B. mit `up`, `down` und `w`, `s`) bewegen kann. In jedem Durchlauf der `main`-Schleife sollen Sie die `step()`-Methode von `PongGame` aufrufen. Ändern Sie `step()` so ab, dass sich der Ball entsprechend seiner Geschwindigkeit bewegt (noch ohne Kollisionen).

Starten Sie das Programm und überprüfen Sie, ob sich Spieler und Ball wie erwartet bewegen.

c) Fangen Sie jetzt in `step()` die Kollisionen des Balls mit den Wänden ab. Es gilt "Einfallswinkel ist gleich Ausfallswinkel".

   Tipp: Da alle Wände und Balken senkrecht oder waagrecht sind, ändert sich immer nur eine Komponente der Geschwindigkeit.

d) Stellen Sie das Spiel fertig, indem Sie in `step()` auch Kollisionen zwischen Ball und Spielerbalken abfangen. Sie können z.B. eine Methode in der Klasse `PongGame` schreiben, welche überprüft, ob ein Ball mit einem gegebenen Spieler kollidiert.

   Zählen Sie jetzt auch die Punkte der Spieler. Wenn der Ball eine Seitenwand berührt, gibt es einen Punkt und der Ball startet von Neuem in der Mitte des Feldes. Um den Punktestand zu zeichnen, können Sie die `Window.drawString()`-Methode verwenden.

Zusatz-Aufgabe: Erweitern Sie das Spiel so, dass mehrere Bälle gleichzeitig im Spiel sind. Immer nach 50 Spielschritten soll ein neuer Ball hinzukommen und wieder entfernt werden, wenn er das Spielfeld verlässt. Dafür sollen Sie die Klassen der Doubly-linked List kopieren und für `Ball`-Objekte anpassen.

Anhang A: Testprogramm Bonusaufgabe

```
public class Klassen {

        ...

        public static void klassen(PrintStream output) {

                Lambda l = new Lambda();
                Sigma s = new Sigma();
                Alpha a = new Alpha();
                Kappa k = new Kappa();
                Iota i = new Iota();
                Zeta z = new Zeta();
                Beta b = new Beta();
                Omega o = new Omega();

                Omega[] os = {z, k, l, (Omega) i};
                for (int j = 0; j < os.length; j += 1) {
                        output.println(os[j].name());
                        output.println("---");
                }

                b = doSomething(l);
                int n = choose(l, z);

                b = l.choose(a, s, i);
                output.println(b == a);
                output.println("---");

                boolean c = s.calc() > i.calc().length() && z.calc();

                Lambda ll = o.create();
                output.println(ll.name());
                output.println("---");
        }

        public static int choose(Kappa k, Alpha a) {
                return 0;
        }

        public static Kappa choose(Alpha a, Kappa k) {
                return a;
        }
```

```
        public static Sigma doSomething(Iota i) {
                return null;
        }

        public static int doSomething(Kappa k) {
                return 0;
        }
}
```

Anhang B: Ausgabe Bonusaufgabe

```
Zeta
---
Kappa
---
Lambda
---
Iota
---
true
---
Kappa
---
```

Anhang C: Format der Flüge-Dateien

Die folgende EBNF-Beschreibung für `<format>` definiert das Format für die Flüge-Dateien. *Ihr Programm sollte alle Dateien, die dieser Beschreibung entsprechen, unterstützen*. Das Symbol ↩ steht für einen Zeilenumbruch und ␣ für ein Leerzeichen.

```
<buchst> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
            a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<alpha>  <= <buchst> { <buchst> }
<zif>    <= 0|1|2|3|4|5|6|7|8|9
<num>    <= <zif> { <zif> }
<datum>  <= <zif> <zif> . <zif> <zif> . <zif> <zif> <zif> <zif>
<klasse> <= ECONOMY | BUSINESS | FIRST
<name>   <= <alpha> ␣ [ <alpha> ␣ ] <alpha>
<flug>   <= <alpha> <num> ␣ <datum> ␣ <klasse> [ ␣ <num> ]
<format> <= { <name> ↩ { <flug> ↩ } . ↩ }
```