252-0027-00: Einführung in die Programmierung

Übungsblatt 5

Abgabe: 29. Oktober 2019, 10:00

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Beachten Sie, dass Sie mehrere unabhängige Programme im selben Eclipse-Projekt haben werden. Bevor Sie ein Programm starten, achten Sie deshalb darauf, dass Sie die richtige Datei im Package Explorer ausgewählt oder im Editor geöffnet haben. *Vergessen Sie nicht, Ihren Programmcode zu kommentieren!*

Aufgabe 1: Testen mit JUnit

Bisher haben Sie Ihre Programme "von Hand" getestet, das heisst, Sie haben das Programm mehrmals ausgeführt und verschiedene Eingaben ausprobiert. In dieser Aufgabe lernen Sie das Konzept der *Testautomatisierung* kennen. Das Testen eines Programms mit verschiedenen Eingaben wird dabei wiederum von einem Programm übernommen. Dabei kann ein Programm als ganzes oder es können einzelne Teile davon separat getestet werden. Automatische Tests haben den Vorteil, dass sie nur einmal geschrieben werden müssen und danach bei jeder Änderung des Programms ohne Aufwand ausgeführt werden können.

In der Übungs-Vorlage finden Sie das Programm "PerpetualCalendar.java", welches für jedes (gültige) Datum den Wochentag berechnet. Leider enthält das Programm noch Fehler, die aber vom Compiler nicht erkannt werden. Das Programm ist also ein gültiges Java-Programm, aber es verhält sich nicht so, wie der Author es beabsichtigt hat. Glücklicherweise hat der Author Tests geschrieben, welche die Korrektheit der Teile des Programms überprüfen. Sie finden diese in der Datei "PerpetualCalendarTest.java", welche sich im Ordner "test" befindet. Ihre Aufgabe ist nun, mithilfe dieser Tests die Fehler im Programm zu finden und zu beheben.

xkcd: PermaCal by Randall Munroe

a) Öffnen Sie zuerst das Programm "PerpetualCalendar.java" und führen Sie es aus. Geben Sie ein paar Daten ein und überprüfen Sie die Ausgabe. Sie werden feststellen, dass das Programm noch nicht korrekt funktioniert.

b) Öffnen Sie nun die Datei "PerpetualCalendarTest.java" und drücken Sie um alle Tests in dieser Datei auszuführen. Es öffnet sich die *JUnit*-Ansicht und die Tests werden ausgeführt. Im oberen Bereich der Ansicht sehen Sie "Runs: 5/5, Errors: 0, Failures: 4". Es sind also 5 Tests ausgeführt worden, von denen 4 fehlgeschlagen sind.

c) Im mittleren Bereich der Ansicht sehen Sie eine Auflistung der Tests. Für jede Hilfsmethode im Programm gibt es einen Test. Im Moment scheint nur die `isLeapYear()`-Methode korrekt zu sein. Klicken Sie auf einen fehlgeschlagenen Test, um im unteren Bereich der Ansicht den Fehler anzuzeigen. Beim Test *testCountDaysInYear* steht "expected:<365> but was:<366>".

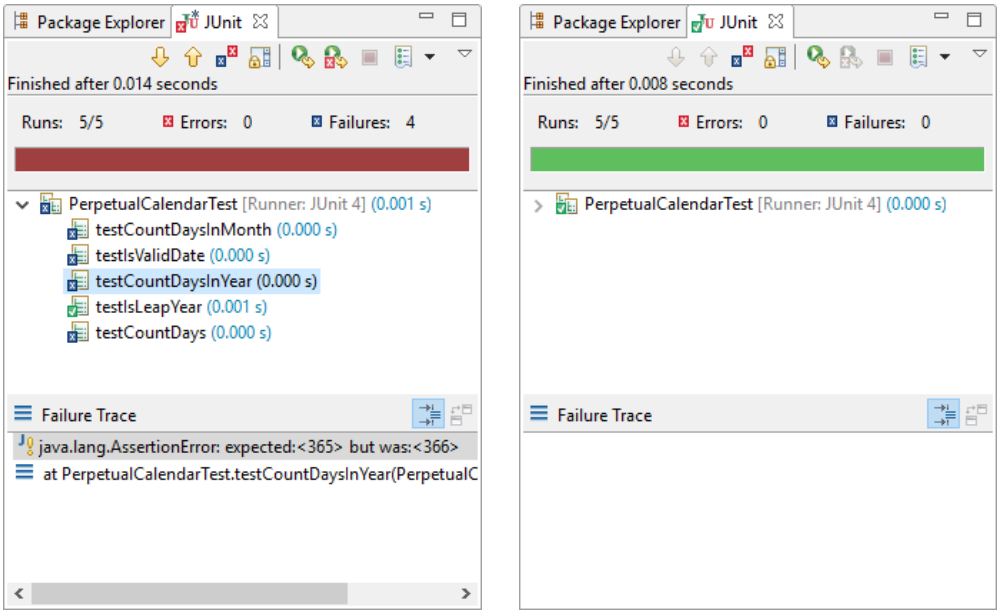
d) Doppelklicken Sie auf *testCountDaysInYear*, um zum Code für diesen Test zu springen. Sie sehen, dass die Methode `countDaysInYear()` der Klasse `PerpetualCalendar` mit dem Argument `1900` aufgerufen wird. Ausserdem wird die spezielle Methode `assertEquals()` aufgerufen, welche das Resultat von `countDaysInYear()` mit dem erwarteten Wert `365` vergleicht. Dass der Test auf dieser Zeile fehlschlägt, bedeutet also, dass `countDaysInYear(1900)` nicht den erwarteten Wert `365` zurückgibt (sondern 366).

   In anderen Tests werden statt `assertEquals()` die Methoden `assertTrue()` und `assertFalse()` verwendet. Diese Methoden überprüfen, ob der boolesche Ausdruck, der als zweites Argument übergeben wird, `true` bzw. `false` ist.

e) Öffnen Sie wieder das Programm "PerpetualCalendar.java" und gehen Sie zur Methode `countDaysInYear()`. Finden und beheben Sie den Fehler in dieser Methode und führen Sie die Tests erneut aus. Wenn Sie den Fehler korrekt behoben haben, sollte der Test *testCount-DaysInYear* ohne Fehler durchlaufen und oben sollte "Failures: 3" stehen.

f) Finden Sie nun die restlichen Fehler, indem Sie den fehlschlagenden Tests nachgehen. Wenn Sie alle Fehler behoben haben, zeigt die *JUnit*-Ansicht einen grünen Balken an.

Keep the bar green to keep the code clean!



Aufgabe 2: Wörter Raten

Das Programm "WoerterRaten.java" enthält Fragmente eines Rate-Spiels, welches Sie vervollständigen sollen. In dem Spiel wählt der Computer zufällig ein Wort w aus einer Liste aus und der Mensch muss versuchen, das Wort zu erraten. In jeder Runde kann der Mensch eine Zeichenfolge z (welche einen oder mehrere Buchstaben enthält) eingeben und der Computer gibt einen Hinweis dazu. Folgende Hinweise sind möglich:

1. w beginnt mit z
2. w endet mit z
3. w enthält z
4. w enthält nicht z

Die Hinweise 1 und 2 können kombiniert werden. Beachten Sie ausserdem, dass die Hinweise 1 und 2 den Hinweis 3 schon enthalten. Das Spiel endet, wenn der Mensch das Wort vollständig eingibt. Dann gibt der Computer den Pseudo-Hinweis "w ist z" und die Anzahl Versuche aus.

a) Öffnen Sie die Text-Datei "woerter.txt", welche sich direkt im Projekt-Ordner befindet. Aus dieser Datei liest das Programm (in der Methode `liesWoerter()`) die Wörter ein. Auf der ersten Zeile steht die Anzahl der Wörter, danach folgt auf jeder Zeile ein neues Wort. Fügen Sie der Liste einige eigene Wörter hinzu und ändern Sie die Zahl oben entsprechend.

b) Vervollständigen Sie das Programm "WoerterRaten.java", indem Sie die noch leeren Methoden ausfüllen. Beachten Sie, dass die Datei "WoerterRatenTest.java" Tests für die Methoden `hinweis()` und `zufallsWort()` enthält. Wenn Sie sich an die Vorgaben gehalten haben, sollten alle Tests erfolgreich durchlaufen. Folgende Methoden könnten sich als hilfreich erweisen: `String.equals()`, `String.startsWith()`, `String.endsWith()`, `String.contains()` und `Random.nextInt()` (um `nextInt()` aufzurufen, müssen Sie zuerst per `new Random()` ein Objekt vom Typ `Random` erstellen).

   Für die Haupt-Methode `rateSpiel()` gibt es keine Tests. Sie sollten diese Methode so schreiben, dass sich das Programm ungefähr wie in folgendem Beispiel-Ablauf verhält (die Benutzer-Eingaben sind grün dargestellt):

```
Tipp? e
Das Wort enthält nicht "e"!
Tipp? a
Das Wort endet mit "a"!
Tipp? j
Das Wort beginnt mit "j"!
Tipp? v
Das Wort enthält "v"!
Tipp? java
Das Wort ist "java"!
Glückwunsch, du hast nur 5 Versuche benötigt!
```

Tipp: Das Spiel wird richtig schwierig, wenn Sie die Liste der möglichen Wörter nicht kennen (oder die Liste sehr lang ist). Tauschen Sie doch mal Ihre "woerter.txt"-Datei mit der eines Mitstudierenden aus, ohne sie anzusehen.

Aufgabe 3: Datenanalyse

In dieser Aufgabe werden Sie die Körpergrössen einiger Personen analysieren, welche für eine Studie[1] in Kalifornien erhoben wurden. Im Programm "DatenAnalyse.java" sind schon ein paar (leere) Methoden zu Ihrer Hilfe vorgegeben.

a) Das Programm soll als erstes die Körpergrössen aus der Datei "groessen.txt" im Projekt-Verzeichnis in ein Array einlesen. Die Datei hat ein ähnliches Format wie die "woerter.txt"-Datei der letzten Aufgabe. Die Körpergrössen liegen als ganze Zahlen, in cm vor. Implementieren Sie dazu die Methode `liesGroessen()`, indem Sie die nötigen Daten aus dem gegebenen `Scanner` auslesen. Falls Sie Schwierigkeiten dabei haben, können Sie sich an der `WoerterRaten.liesWoerter()`-Methode orientieren.

   Verwenden Sie den Test *testLiesGroessen* in der Datei "DatenAnalyseTest.java", um Ihren Code zu testen.

b) Führen Sie als nächstes eine einfache Analyse der Daten durch, indem Sie das Minimum, das Maximum und den Durchschnitt und ausserdem die Anzahl der Körpergrössen ausgeben. Füllen Sie dazu die Methode `einfacheAnalyse()` aus. Beachten Sie folgende Methoden: `Math.min()` und `Math.max()`.

c) Die drei Werte, die Sie in b) berechnet haben, sagen nicht viel über die Daten aus. Um die Daten besser zu verstehen, soll Ihr Programm ein Histogramm berechnen und ausgeben. Sie können das Histogramm in Text-Form auf der Konsole ausgeben, oder auch mit der `Window`-Klasse auf ein Fenster zeichnen. Die Textausgabe könnte ungefähr so aussehen:

```
[140,143)
[143,146)
[146,149) |
[149,152) ||||
[152,155) ||||||||||||
[155,158) ||||||||||||||||||||
[158,161) |||||||||||||||||||||||||||||||||||||||||||
[161,164) |||||||||||||||||||||||||||||||||||||||||||||||||||
[164,167) |||||||||||||||||||||||||||||||||||||||||
[167,170) |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
[170,173) |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
[173,176) |||||||||||||||||||||||||||||||||||||||||||||||||||||||||
[176,179) |||||||||||||||||||||||||||||||||||||||||||||||||||||||
[179,182) |||||||||||||||||||||||||||||||||||||
[182,185) |||||||||||||||||||||||||||
[185,188) ||||||||||||||
[188,191) ||||||||||||||||||||
[191,194) ||||||
[194,197)
[197,200) ||
```

[1] Grete Heinz, Louis J. Peterson, Roger W. Johnson, and Carter J. Kerk, *Exploring Relationships in Body Dimensions*, Journal of Statistics Education 11, no. 2 (2003).

Implementieren Sie die Methode `histogrammAnalyse()`. Diese Methode soll zuerst den Benutzer nach der Anzahl Histogramm-Klassen fragen, dann das Histogramm berechnen und schliesslich ausgeben. Zwei (leere) Methoden sind schon vorgegeben: `erstelleHistogramm()` und `klassenBreite()`. Für diese beiden Methoden existieren Tests in "DatenAnalyseTest.java" und Kommentare, welche Ihnen beim Schreiben des Programms helfen. Überlegen Sie sich, wie Sie das Problem weiter aufteilen möchten, und erstellen Sie entsprechende Methoden.

Aufgabe 4: Hotellerie (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 Ihres Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

In dieser Aufgabe analysieren Sie die Zimmerbuchungsdaten eines Hotels innerhalb eines Kalenderjahrs. Ein Beispiel für die Zimmerbuchungsdaten finden Sie in der Datei "hotelDaten1.txt". Die Datei enthält eine Liste der Buchungen, welche das Hotel getätigt hat. In der ersten Zeile steht nur die Anzahl Buchungen. Für jede Zimmerbuchung gibt es genau eine weitere Zeile, in der Sie (durch Leerzeichen getrennt) folgende Informationen finden (in dieser Reihenfolge):

1. Die Nummer des Zimmers (eine positive ganze Zahl zwischen 1 und 256 inklusive).
2. Den Tag im Jahr, an dem die Buchung des Zimmers begonnen hat (eine positive ganze Zahl zwischen 1 und 366 inklusive).
3. Den Tag im Jahr, an dem die Buchung des Zimmers geendet hat (eine positive ganze Zahl zwischen 1 und 366 inklusive).
4. Der Preis des Zimmers pro angefangenem Tag (Eine nicht negative reelle Zahl).
5. Der Discount des Zimmers (eine ganze Zahl zwischen 0 und 100, welche den Discount als Prozent angibt: Zum Beispiel, 0 ist kein Discount, 50 ist halber Preis und 100 heisst, dass das Zimmer Gratis ist).

Alle Buchungen enden im gleichen Jahr, da das Hotel an Sylvester geschlossen hat. Es gibt für jede Zimmernummer ein Zimmer, auch wenn dieses in einem Jahr eventuell nicht gebucht wurde und daher nicht in der Datei erscheint. Ein Zimmer kann an einem Tag höchstens einmal gebucht werden; das gilt auch für den Tag, an dem die Buchung endet. Es gibt keine angebrochenen Tage, Zimmer werden immer für ganze Tage vermietet.

Ihr Programm soll die Datei einmal einlesen und anschliessend einige Fragen beantworten. Dazu vervollständigen Sie die Methode `analyse()` in der Klasse `Hotellerie`. Die Methode hat zwei Argumente: ein `input`-Argument zum Lesen der Datei und ein `output`-Argument zum Ausgeben der Antworten. Sie sollen also vom `input` lesen und Antworten zu den folgenden Fragen auf dem `output` ausgeben:

1. Welches Zimmer wurde am häufigsten ausgeliehen (d.h. die grösste Anzahl an Buchungen)? Geben Sie die Zimmernummer aus, mit diesem `println()`-Statement (ändern Sie nicht den Text):

```
output.println("Am haeufigsten gebucht: " + zimmerNummer);
```

2. Welches Zimmer wurde am meisten besetzt (d.h. für die grösste Gesamtanzahl an Tagen)? Geben Sie die Zimmernummer aus, mit diesem `println()`-Statement (ändern Sie nicht den Text):

   ```
   output.println("Am meisten besetzt: " + zimmerNummer);
   ```

3. Welches Zimmer hat dem Hotel den grössten Betrag eingebracht? Geben Sie die Zimmernummer aus, mit diesem `println()`-Statement (ändern Sie nicht den Text):

   ```
   output.println("Groessten Betrag eingebracht: " + zimmerNummer);
   ```

4. Was sind die Gesamteinnahmen des Hotels in diesem Jahr? Geben Sie den Betrag aus, mit diesem `println()`-Statement (ändern Sie nicht den Text):

   ```
   output.println("Gesamteinnahmen des Hotels: " + summe);
   ```

Jede der Informationen 1.–4. soll (in dieser Reihenfolge) auf einer separaten Zeile ausgegeben werden. Beträge werden als ungerundete reelle Zahlen ausgegeben. Zimmernummern werden als natürliche Zahlen ausgegeben. Falls bei einer Information mehrere gleichwertige Ausgaben möglich sind, kann Ihr Programm *eine* beliebige der gleichwertigen Lösungen ausgeben.

Ihr Programm muss nur wohlgeformte, nicht-leere Eingabe-Dateien unterstützen. Ein Beispiel einer solchen Datei finden Sie im Projekt unter dem Namen "hotelDaten1.txt". Exceptions im Zusammenhang mit Ein- und Ausgabe müssen nicht behandelt werden. Für die Beispiel-Datei sollte die Ausgabe wie folgt aussehen:

```
Am haeufigsten gebucht: 1
Am meisten besetzt: 42
Groessten Betrag eingebracht: 256
Gesamteinnahmen des Hotels: 22000.75
```

Testen Sie Ihr Programm ausgiebig—am besten mit JUnit—und pushen Sie die Lösung vor dem Abgabetermin. Wir haben Ihnen einen JUnit Test bereits erstellt.

Aufgabe 5: Chaos Game (GUI)

Im letzten Übungsblatt haben Sie ein erstes Mal die `Window`-Klasse benutzt um die Schweizerfahne anzuzeigen. In dieser Aufgabe werden Sie lernen, wie Sie mithilfe einer Hauptschleife ("Main-Loop") ein sich änderndes Bild anzeigen können.

1. Erstellen Sie ein neues Programm "ChaosGame.java". In der `main`-Methode erstellen Sie eine Instanz der `Window`-Klasse und zeigen das Fenster gleich an:

   ```
   Window window = new Window("Chaos", 800, 800);
   window.open();
   ```

2. Damit wir den Inhalt des Fensters fortlaufend ändern können, rufen wir nun nicht mehr die `waitUntilClosed`-Methode auf, da deren Ausführung erst beendet ist, wenn das Fenster

geschlossen wird. Stattdessen schreiben wir eine Hauptschleife, die selbst überprüft, ob das Fenster noch offen ist. So lange das Fenster nicht geschlossen wurde, können wir unser Bild in jeder Wiederholung (Iteration) ändern:

```
while (window.isOpen()) {
    // ändere den Fensterinhalt
    window.refresh();
}
```

Am Ende der Schleife rufen wir die `refresh`-Methode auf. Diese Methode teilt dem Fenster mit, dass wir unser aktuelles Bild anzeigen wollen. Erst dadurch werden unsere Zeichnungsbefehle sichtbar.

Zeichnen Sie in jeder Iteration der Schleife an einer zufälligen Position ein Quadrat mit Seitenlänge 10. Um eine zufällige Position zu bestimmen, können Sie die Methode `Math.random()` benutzen, die bei jedem Aufruf einen zufälligen `double`-Wert aus dem rechtsoffenen Intervall $[0.0, 1.0[$ zurückgibt. Indem Sie einen solchen Wert mit einer Zahl n multiplizieren, erhalten Sie eine Zufallszahl zwischen 0 und n.

3. Nachdem Sie sich mit der Hauptschleife vertraut gemacht haben, können wir jetzt das Chaos-Game spielen. Der folgende Algorithmus zeichnet eine Figur, erraten Sie welche?

   (a) Wählen Sie 3 Punkte im Fenster-Koordinatensystem, welche ein Dreieck definieren (zufällig oder nach Geschmack).

   (b) Wählen Sie eine zufällige Startposition innerhalb des Fensters.

   (c) Wählen Sie zufällig einen der 3 Eckpunkte von (a) aus.

   (d) Von der aktuellen Position aus, gehen Sie die Hälfte der Distanz in Richtung des ausgewählten Eckpunktes und zeichnen diese Position ein. Dies ist die neue Position.

   (e) Gehen Sie zurück zu (c).

   Ändern Sie Ihr Programm nun so ab, dass es mit jeder Iteration der Hauptschleife eine weitere Position gemäss diesem Algorithmus einzeichnet. Wenn Sie alles richtig gemacht haben, entsteht mit der Zeit eine interessante Figur. (Das Zeichnen der zufälligen Quadrate in Schritt 2 können Sie wieder entfernen.)

   Tipps: Um einen zufälligen Eckpunkt auszuwählen, ist der Ausdruck `(int)(Math.random()*3)` nützlich. Das Einzeichnen einer Position können Sie mit `fillRect(x, y, 1, 1)` machen. Und wenn es zu lange dauert, bis die Figur entsteht, können Sie mit einer Schleife innerhalb der Hauptschleife auch gleich mehrere Positionen berechnen und einzeichnen.