252-0027-00: Einführung in die Programmierung

Übungsblatt 6

Abgabe: 28. Oktober 2025, 23:59

Checken Sie mit IntelliJ wie bisher die neue Übungsvorlage aus. Importieren Sie beide IntelliJ-Projekte (das Projekt für den Bonus und das Projekt für die restlichen Aufgaben). Beachten Sie, dass Sie mehrere unabhängige Programme im bonusunabhängigen IntelliJ-Projekt haben werden. Bevor Sie ein Programm starten, achten Sie deshalb darauf, dass Sie die richtige Datei ausgewählt oder im Editor geöffnet haben. *Vergessen Sie nicht, Ihren Programmcode zu kommentieren!*

Ab Woche 5, werden die TAs Ihnen nur dann Feedback geben, wenn Sie eine entsprechende Datei namens "requestfeedback.txt" in dem Projekt-Ordner hochladen. In der Text-Datei geben Sie an, für welche Aufgaben Sie Feedback erhalten möchten. Um eine neue Text-Datei in IntelliJ zu erstellen, rechts-klicken Sie den uXX Projekt-Ordner und wählen *New* → *File*. Im erscheinenden Fenster, benennen Sie die Datei mit "requestfeedback.txt" und drücken die Enter-Taste. Danach wählen Sie *Add*, falls IntelliJ Sie fragt, ob Sie die Datei zu git hinzufügen möchten.

Aufgabe 1: Präfixkonstruktion

Gegeben seien zwei Strings `s` und `t` und ein Integer `n` mit $\texttt{n} \geq 0$. Schreiben Sie ein Programm, das zurückgibt, ob `s` eine Konkatenation von maximal `n` vielen Präfixen von `t` ist.
Beispiele:

- `s` = `"abcababc"`, `t` = `"abc"`, `n` = 4: Das Programm sollte `true` zurückgeben, da `"abc"` und `"ab"` Präfixe von `t` sind und `s` eine Konkatenation von `"abc"`, `"ab"`, `"abc"` ist.
- `s` = `"abcbcabc"`, `t` = `"abc"`, `n` = 4: Das Programm sollte `false` zurückgeben, da `"bc"` kein Präfix von `t` ist.
- `s` = `"abab"`, `t` = `"abac"`, `n` = 2: Das Programm sollte `true` zurückgeben, da `"ab"` ein Präfix von `t` ist und `s` eine Konkatenation von `"ab"`, `"ab"` ist.

Implementieren Sie die Methode `isPrefixConstruction(String s, String t, int n)` in der Klasse `PrefixConstruction`. Die Methode hat drei Argumente: die beiden Strings `s` und `t` und den Integer `n`. Sie dürfen davon ausgehen, dass n grösser oder gleich 0 ist. In der Datei "PrefixConstructionTest.java" finden Sie Tests.

Tipp: Lösen Sie die Aufgabe rekursiv.

Aufgabe 2: Weakest Precondition

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Speichern Sie diese in der Text-Datei "WeakestPrecondtion.txt", welche sich in Ihrem "u06"-Ordner befindet. Bitte verwenden Sie Java-Syntax. Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1.
```
P: { ?? }
S:   m = n * 4;  k = m - 2;
Q: { n > 0 && k > 5 }
```

2.
```
P: { ?? }
S:   m = n * n;  k = m * 2;
Q: { k > 0 }
```

3.
```
P: { ?? }
S:   y = x + 3; z = y + 1;
Q: { z > 4 }
```

4.
```
P: { ?? }
S:   y = x + 1; z = y - 3;
Q: { z == 10 }
```

5.
```
P: { ?? }
S:   x = y + z; x = y * y;
Q: { x > 0 }
```

6.
```
P: { ?? }
S:   x = a + 1; sum = x + a;
Q: { sum > 5 }
```

7.
```
P: { ?? }
S:   x = y; x = x + 1;
Q: { x > 0 }
```

8.

```
P: { ?? }
S:   x = 2 * y; z = x + y;
Q: { z != 0 }
```

Aufgabe 3: Wörter Raten

Das Programm "WoerterRaten.java" enthält Fragmente eines Ratespiels, 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 der 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 Hauptmethode `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 4: Datenanalyse

In dieser Aufgabe werden Sie die Kelchblattlänge von Iris Blumen, welche auch Schwertlilien genannt werden, analysieren. Dazu verwenden Sie ein öffentlich zugängliches Dataset [1] welches die Längen vom Kelchblatt (Sepal Length) von jeweils 150 verschiedenen Iris Blumen enthält. Es gibt sehr viele Arten von Iris Blumen; insgesamt sind 285 Arten bekannt. Diese zu unterscheiden, ist ein komplexer Prozess. Wir werden jedoch versuchen, mittels der Länge des Kelchblattes drei Arten von Iris Blumen zu unterscheiden, indem wir die gegebenen Daten analysieren. Wir werden uns auf die folgenden drei Iris Blumen konzentrieren:

- Iris setosa, auch Borsten-Schwertlilie genannt.
- Iris versicolor, auch Verschiedenfarbige Schwertlilie gennannt.
- Iris virginica, auch blaue Sumpfschwertlilie genannt.

[1] Iris by UC Irvine Machine Learning Repository

http://www.lac.inpe.br/~rafael.santos/Docs/CAP394/WholeStory-Iris.html

1. Das Programm soll als erstes die Kelchblattgrössen für die drei Arten von Iris Blumen aus den Dateien "sepal_length_setosa.txt", "sepal_length_versicolor.txt", und "sepal_length_virginica.txt" im Projekt-Verzeichnis in ein Array einlesen. Die Dateien haben ein ähnliches Format wie die "woerter.txt"-Datei der letzten Aufgabe. Die Kelchblattgrössen liegen als ganze Zahlen in Millimetern [mm] vor. Die erste Zahl gibt die Anzahl Daten im File an. Implementieren Sie dazu die Methode `liesLaengen()`, indem Sie die nötigen Daten aus dem gegebenen `Scanner` auslesen. Falls Sie Schwierigkeiten dabei haben, können Sie sich an der `WoerterRaten.liesWoerter()`-Methode orientieren. Sie können Ihre Methode anhand eines der drei Files testen um zu schauen, ob diese funktioniert.

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

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

3. Die drei Werte, die Sie in 2. berechnet haben, sagen nicht viel über die Daten aus. Um die Daten besser zu verstehen, soll Ihr Programm ein Histogramm berechnen und auf der Konsole ausgeben. Die Textausgabe könnte ungefähr so aussehen:

```
[40,43)
[43,46)
[46,49)
[49,52) |
[52,55)
[55,58) ||
[58,61) ||||||
[61,64) ||||||||||
[64,67) |||||||||
[67,70) ||||||||||
[70,73) ||||
[73,76) ||
[76,79) |||||
[79,82) |
[82,85)
```

   Implementieren Sie die Methode `histogrammAnalyse()`. Diese Methode soll zuerst den Benutzer nach der Anzahl der 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.

4. (Optional) Betrachten Sie nun die Histogramme für die drei verschiedenen Arten von Blumen. Fällt Ihnen was auf? Wenn Ihnen nun die Kelchblattlängen von den drei Iris Blumen gegeben

werden mit jeweils Wert 49 und 66, wie würden Sie diese einordnen? Beachten Sie, dass dies keine definitive Antwort ist und nur eine Approximation.

Aufgabe 5: Matrix (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. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

Implementieren Sie die Methode `countAssimilated` in der Klasse `Matrix`, welche eine $m \times n$-Matrix A (mit $m > 2, n > 2$) von `int`-Werten als Input akzeptiert und einen `int`-Wert zurückgibt. Für alle Elemente $a_{i,j}$ von A gilt $a_{i,j} \neq 0$.

Diese Methode soll die Anzahl der *an ihre Umgebung angepassten* Elemente in der Input-Matrix A berechnen und zurückgeben. Ein Element $a_{i,j}$ von A gilt als angepasst, wenn die Summe seiner direkten Nachbarn (vertikal, horizontal und diagonal, soweit in der Matrix A definiert) ohne Rest durch $a_{i,j}$ teilbar ist. Das heisst, dass ein Element $a_{i,j}$ genau dann angepasst ist, wenn

$$(a_{i-1,j-1} + a_{i-1,j} + a_{i-1,j+1} + a_{i,j-1} + a_{i,j+1} + a_{i+1,j-1} + a_{i+1,j} + a_{i+1,j+1}) \% a_{i,j} == 0$$

gilt. Wenn das Element $a_{x,y}$ nicht existiert (da $x < 0$ oder $x \geq m, y < 0, y \geq n$), dann wird der Wert in der Berechnung durch 0 ersetzt.

Sie können davon ausgehen, dass der Parameter `matrix` nie den Wert `null` hat und nie auf eine $m \times n$ Matrix mit $m \leq 2$ oder $n \leq 2$ verweist. Sie können auch davon ausgehen, dass die Summe der Nachbarn sich ohne Overflow oder Underflow berechnen lässt.

Hier sind ein paar Beispielaufrufe der Methode sowie das erwartete Ergebnis (im "test"-Ordner finden Sie die entsprechenden JUnit-Tests):

1. `countAssimilated([[10, 10, 10], [10, 10, 10], [10, 10, 10]])`: 9
2. `countAssimilated([[5, 10, 3], [6, 9, 6], [3, 3, 15]])`: 4
3. `countAssimilated([[4, 7, 13], [-2, -12, 32], [20, 15, -8], [17, 3, 1111]])`: 3
4. `countAssimilated([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12], [13, 14, 15, 16, 17, 18], [19, 20, 21, 22, 23, 24], [25, 26, 27, 28, 29, 30]])`: 15

In der Klasse `Matrix` finden Sie eine `main`-Methode, welche Sie verwenden können, um Ihre Implementierung zu testen. Tests finden Sie in der Datei "MatrixTest.java" im "test"-Ordner. Diese Aufgabe ist eine alte Prüfungsaufgabe. Die Datei "GradingMatrixTest.java" enthält die Tests, welche wir bei der Prüfung für die Korrektur verwendet haben. Wir empfehlen, diese Tests erst zu verwenden, wenn Sie denken, dass Ihre Lösung korrekt ist, damit Sie sehen können, wie Sie bei einer Prüfung abgeschnitten hätten.