Heinrich Heine Universität Düsseldorf

Programmierung – WS 2025/26
Dr. Markus Brenneis
Abgabefrist für Feedback:
26.01.26, 10 Uhr

13. Übungsblatt (Lösungsvorschlag[1])

Denken Sie daran, sich über das Studierendenportal für die Klausur anzumelden. (Anmeldefrist: 2 Wochen)

Aufgabe 1: Ein kleiner Taschenrechner nach VL 23

Da wir uns jetzt mit Fehlerbehandlung (hoffentlich) gut auskennen, wollen wir einen kleinen Taschenrechner bauen, der gegen alle Arten von Fehler gewappnet ist. Unser Taschenrechner soll Ausdrücke der Form `1 + 9`, `-4.5 - 30.2`, `3.14 * 2`, `10 / 5.2` berechnen können, erwartet also zwei Fließkommazahlen und einen Operator (für Addition, Subtraktion, Multiplikation oder Division). Die drei Werte sollen als Kommandozeilenargumente übergeben werden (siehe Beispiele unten).

Ihr Programm `Calculator` soll nur das Ergebnis der Berechnung auf der Standardausgabe ausgeben. Außerdem sollen folgende Fehlerfälle behandelt werden:

- Wenn zu wenige Argumente angegeben werden, soll eine Fehlermeldung ausgegeben werden, die `too few arguments` enthält.
- Wenn zu viele Argumente angegeben werden, soll eine Fehlermeldung ausgegeben werden, die `too many arguments` enthält.
- Wenn ein unbekannter Operator verwendet wird, soll eine Fehlermeldung ausgegeben werden, die `unknown operator` enthält.
- Wenn eine Zahl nicht als Double verarbeitet werden kann, soll eine Fehlermeldung ausgegeben werden, die `first operand is no number` bzw. `second operand is no number` enthält.
- Wenn durch 0 geteilt werden soll, soll eine Fehlermeldung ausgegeben werden, die `division by zero` enthält.

In keinem Fehlerfall soll es eine unbehandelte Exception geben. Geben Sie beim Fangen von Exceptions den Exception-Typ so genau wie möglich an. Fangen Sie nur dann Exceptions, wenn dies wirklich sinnvoll ist; Fehlerfälle, die Sie bisher mit Verzweigungen abgefangen haben, sollten Sie weiterhin mit Verzweigungen behandeln.

[1] Bei den meisten Programmieraufgaben gibt es mehr als einen funktierenden Lösungsweg. Diskutieren Sie gerne untereinander Ihre Lösungsansätze und lerne Sie damit verschiedene Lösungsstrategien und verschiedene Anwendungsmöglichkeiten der Java-Funktionalitäten kennen. Ihre Abgabe müssen Sie aber final selbst formulieren/eintippen.

Beispielaufrufe[2]:

```
% java Calculator 1.2 + 2.2
3.4
% java Calculator 10 / -4
-2.5
% java Calculator 8 /
too few arguments, expected: operand operator operand
% java Calculator 8 / 4 4
too many arguments, expected: operand operator operand
% java Calculator 8 % 4
unknown operator %, supported operators: + - * /
% java Calculator two + 2
first operand is no number: two
% java Calculator 2 / zero
second operand is no number: zero
% java Calculator 2 / 0
division by zero
```

Aufgabe 2: Space Invaders nach VL 22

In der Vorgabe finden Sie ein kleines Spiel. Leider kann die Hauptklasse `de.hhu.progra.spaceinvaders.Game` nicht korrekt ausgeführt werden, weil bei der Aufteilung des Codes in Packages ein paar Fehler gemacht wurden.

Fügen Sie fehlende `import`- und `package`-Statements hinzu und passen Sie, wo nötig, Sichtbarkeiten an, sodass der Code ausgeführt[3] werden kann. Den Code der Klassen selbst müssen Sie nicht verändern und auch nicht nachvollziehen.

Testen Sie selbst auf Ihrem Rechner, ob Ihr angepasster Code funktioniert.

Bonus: Wenn man das Spiel gewinnt, gibt es eine Exception-Meldung. Beheben Sie dieses Problem.

Lösungsvorschlag

- Im Code sind die korrigierten Stellen mit `BUGFIX` gekennzeichnet. Sie können `grep -R BUGFIX .` im Ordner mit dem Lösungsvorschlag ausführen, um diese Stellen zu finden.

[2] Achtung: Bei den meisten Shells wird ein `*` als Wildcard interpretiert und vorm Aufruf des Java-Programms durch die Namen aller Dateien und Ordner im aktuellen Verzeichnis ersetzt. Um das zu verhindern, können Sie das Sternchen in Anführungszeichen setzen (diese Anführungszeichen werden von der Shell interpretiert und nicht an das Java-Programm durchgereicht): `java Calculator 3 "*" 5`

[3] Entweder durch Compilieren (`javac de/hhu/progra/spaceinvaders/Game.java`) und Ausführen (`java de.hhu.progra.spaceinvaders.Game`) oder durch direkte Ausführung von `java de/hhu/progra/spaceinvaders/Game.java`

Aufgabe 3: Vokabeln nach VL 23

M. speichert seine Schwedischvokabeln in einer CSV-Datei. Tabellenzeilen sind dabei durch Zeilenumbrüche getrennt, einzelne Einträge durch Kommata.[4] M. speichert in der ersten Spalte die schwedischen Begriffe, in der zweiten Spalte die zugehörigen Übersetzungen:

```
% cat vokabeln.csv
försvinna,verschwinden
halm,Stroh
hej,hallo
livstid,Lebenszeit
i,in
redigera,bearbeiten
strå,Halm
visa,zeigen
```

M. möchte nun zum Lernen der Vokabeln ein Programm `Vocabulary` haben, das eine Vokabel-CSV-Datei einliest und die Vokabel wie folgt ausgibt: In jeder Zeile steht das schwedische Wort gefolgt von einem Leerzeichen und der deutschen Übersetzung, wobei die Reihenfolge der Buchstaben in der deutschen Übersetzung **zufällig** ist und **nicht** dem Ursprungswort entspricht. Die Reihenfolge der Vokabeln soll beibehalten werden.

Der Dateiname wird als Argument übergeben. Das Programm soll beispielsweise für die mitgelieferte Beispieldatei folgende Ausgabe machen:

```
% java Vocabulary vokabeln.csv
försvinna svrnhidcnwee
halm htroS
hej ahllo
livstid tLneeseibz
i ni
redigera atnebbieer
strå Hmal
visa inzgee
```

Wenn weniger als ein Argument übergeben wird oder die Datei nicht gelesen werden kann, soll eine Fehlermeldung ausgegeben werden, die mit `ERROR` beginnt (aber keine unbehandelte Exception auftreten).

Sie dürfen davon ausgehen, dass deutsche Übersetzungen aus mindestens zwei verschiedenen Zeichen bestehen und das oben beschriebene CSV-Format eingehalten wird.

[4] Das „richtige“ CSV-Format ist eigentlich noch etwas komplizierter (z. B. um Kommata innerhalb einer Tabellenzelle zu behandeln).