Programmieren II (Java)

3. Praktikum: Arrays und Strings

Sommersemester 2025
Christopher Auer

Lernziele

- Arrays: Erstellung, Zugriff und Literale
- `Strings`: arbeiten mit `Strings` und `StringBuilder`

- Schauen Sie sich die Tutorial-Videos auf der *Moodle-Seite* mit *wichtigen Hinweisen* zum Praktikum und VS-Code an!
- Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- Sie müssen *4 der 5* Praktika bestehen
- *Kommentieren* Sie Ihren Code
  - Jede *Methode* (wenn nicht vorgegeben)
  - *Wichtige* Anweisungen/Code-Blöcke
  - *Nicht kommentierter* Code führt zu *Nichtbestehen*
- Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*.
- *Wichtig*: Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*!

Aufgabe 1: SOS

In dieser Übung implementieren wir das Spiel SOS. Das Spiel wird auf meinem *quadratischen Gitter* mit $n \times n$ Feldern (6×6, 9×9, etc.) von *zwei Spielern* gespielt. Der Spieler, der am Zug ist, wählt zunächst ein *freies Feld* und entscheidet dann ob darin `S` oder ein `O` platziert werden soll. Hier z.B. in das Feld $1, 2$ (Zeile, Spalte) ein `S`.

```
   0 1 2 3 4 5 6
  +-+-+-+-+-+-+-+
0 | | | | | | | |
  +-+-+-+-+-+-+-+
1 | | |S| | | | |
  +-+-+-+-+-+-+-+
2 | | | | | | | |
  +-+-+-+-+-+-+-+
3 | | | | | | | |
  +-+-+-+-+-+-+-+
4 | | | | | | | |
  +-+-+-+-+-+-+-+
5 | | | | | | | |
  +-+-+-+-+-+-+-+
```

Dann ist der *zweite Spieler* an der Reihe und entscheidet sich z.B. für $2, 2$ und `O`.

```
   0 1 2 3 4 5 6
  +-+-+-+-+-+-+-+
0 | | | | | | | |
  +-+-+-+-+-+-+-+
1 | | |S| | | | |
  +-+-+-+-+-+-+-+
2 | | |O| | | | |
  +-+-+-+-+-+-+-+
3 | | | | | | | |
  +-+-+-+-+-+-+-+
4 | | | | | | | |
  +-+-+-+-+-+-+-+
5 | | | | | | | |
  +-+-+-+-+-+-+-+
```

Der *erste Spieler* kann nun ein `SOS` *vervollständigen* mit dem Zug $3, 2$ und einem `S`:

```
   0 1 2 3 4 5 6
  +-+-+-+-+-+-+-+
0 | | | | | | | |
  +-+-+-+-+-+-+-+
1 | | |$| | | | |
  +-+-+-+-+-+-+-+
2 | | |0| | | | |
  +-+-+-+-+-+-+-+
3 | | |$| | | | |
  +-+-+-+-+-+-+-+
4 | | | | | | | |
  +-+-+-+-+-+-+-+
5 | | | | | | | |
  +-+-+-+-+-+-+-+
```

Der Spieler bekommt *einen Punkt* und ist *sofort wieder an der Reihe*, solange bis kein `SOS` mehr vervollständigt werden kann. Zur klareren Darstellung, symbolisieren `$` und `0` dass die Einträge *gewertet wurden*. `$` und `0` können im weiteren Verlauf in weiteren `SOS` nochmals verwendet werden.

Ein `SOS` ist *horizontal*, *vertikal* oder *diagonal* gültig. Und mit einem cleveren Zug können sogar mehrere `SOS` auf einmal vervollständigt werden. In folgender Situation *sogar drei* — finden Sie heraus wie!

```
   0 1 2 3 4 5 6
  +-+-+-+-+-+-+-+
0 | | | | | | | |
  +-+-+-+-+-+-+-+
1 | | |$| | | | |
  +-+-+-+-+-+-+-+
2 | |O|0| | | | |
  +-+-+-+-+-+-+-+
3 | |O|$| | | | |
  +-+-+-+-+-+-+-+
4 | |O| | | | | |
  +-+-+-+-+-+-+-+
5 | | |S| | | | |
  +-+-+-+-+-+-+-+
```

Die Antwort ist an der Stelle $3, 0$ mit einem `S`:

```
   0 1 2 3 4 5 6
  +-+-+-+-+-+-+-+
0 | | | | | | | |
  +-+-+-+-+-+-+-+
1 | | |$| | | | |
  +-+-+-+-+-+-+-+
2 | |0|0| | | | |
  +-+-+-+-+-+-+-+
3 |$|0|$| | | | |
  +-+-+-+-+-+-+-+
4 | |0| | | | | |
  +-+-+-+-+-+-+-+
5 | | |$| | | | |
  +-+-+-+-+-+-+-+
```

Importieren Sie das Gradle-Projekt `sos`. Wie immer finden Sie *JUnit-Tests* deren Inhalte noch *auskommentiert sind*.

Die `enum Entry`

Bevor wir mit der Implementierung des Spiels starten, brauchen wir die **`enum`** `Entry`, die die Belegung eines Felds modelliert und folgende folgende Attribute besitzt:

Wert	`char display`	`boolean scored`
`S_UNSCORED`	`S`	**`false`**
`O_UNSCORED`	`O`	**`false`**
`S_SCORED`	`$`	**`true`**
`O_SCORED`	`0`	**`true`**

Das Attribut **`char`** `display` mit dem *Getter* `getDisplay` definiert das Zeichen für die *Bildschirmausgabe*. Das Attribut **`boolean`** `scored` mit *Getter* `isScored` gibt zurück, ob es sich um einen bereits *bewerteten Eintrag* handelt.

- Implementieren Sie `Entry` und vergessen Sie die *Getter* nicht!
  JUnit-Test: `EntryTest.testValues`
- Implementieren Sie die *statische öffentliche Methode* `Entry fromDisplay(`**`char`** `display)`, die zu `display` den entsprechenden Wert der **`enum`** `Entry` zurückgibt, bspw. `Entry.O_SCORED` für `0`!
  JUnit-Test: `EntryTest.testFromDisplay`
- Implementieren Sie die *öffentlichen Methode* `Entry toScored()` und `Entry toUnscored()`, die für den jeweiligen **`enum`**-Wert die *bewertete* bzw. *unbewertete* Variante zurückgibt:

Wert	toScored	toUnscored
S_UNSCORED	S_SCORED	S_UNSCORED
O_UNSCORED	O_SCORED	O_UNSCORED
S_SCORED	S_SCORED	S_UNSCORED
O_SCORED	O_SCORED	O_UNSCORED

JUnit-Tests: `EntryTest.testToScored` und `EntryTest.testToUnscored`

Im folgenden implementieren wir SOS in einer Klasse. Dabei speichern wir die Einträge der Spieler in einem *zweidimensionalen Array vom Typ* `Entry[][]`. Der erste Index gibt die *Zeile*, der zweite Index die *Spalte* an. In obigen Beispiel wäre z.B. `board[1][2] == Entry.S_UNSCORED`; leere Felder haben den Wert **`null`**.

Die Klasse `SOS`

Deklarieren Sie die Klasse `SOS` und darin die *öffentliche statische Methode* `Entry[][] getExample()`, die eine vorbereite Belegung mit folgendem Inhalt zurückgibt:

```
   0 1 2 3 4 5
  +-+-+-+-+-+-+
0 | | | | | |S|
  +-+-+-+-+-+-+
1 | | |S| |$| |
  +-+-+-+-+-+-+
2 | |O| |S|0| |
  +-+-+-+-+-+-+
3 | |S| | |$| |
  +-+-+-+-+-+-+
4 | | | | | | |
  +-+-+-+-+-+-+
5 | | | |S| |O|
  +-+-+-+-+-+-+
```

Gehen Sie bei der Implementierung von `getExample` wie folgt vor:

- Deklarieren Sie einen *zwei-dimensionalen* **`char`**-Array, dessen Belegung Sie mit Hilfe eines *Array-Literals* definieren. Wenn Sie nicht wissen, was ein *Array-Literal* ist, dann recherchieren Sie zuerst nach!
- Wandeln Sie dann den zwei-dimensionalen **`char`**-Array *mit Hilfe der Methode* `Entity.fromDisplay` in einen zwei-dimensionalen `Entity`-Array um.
- Geben Sie das Ergebnis zurück.

JUnit-Test: `SOSTest.testGetExample`

Die Methode `checkBoard`

Um die Prüfung des Parameters `Entry[][] board` im folgenden zu vereinfachen, implementieren Sie die *statische öffentliche Methode* **`void`** `checkBoard(Entry[][] board)`. Diese prüft, ob es sich bei `board` um einen gültigen *quadratischen zwei-dimensionalen Array mit mindestens der Größe* 3×3 handelt. Wenn nicht, dann generiert `checkBoard` eine `IllegalArgumentException`. Implementieren Sie `checkBoard` und testen Sie Ihre Implementierung mit `SOSTest.testCheckBoard`!

Ausgabe einer Belegung

Implementieren Sie die *statische öffentliche* Methode `printBoard(Board[][] board)`, die die Belegung wie bisher gezeigt (siehe oben) ausgibt. *Testen* Sie Ihre Implementierung, indem Sie `SOS`

mit einer `main`-Methode ausstatten, in der Sie `printBoard` mit dem Rückgabe von `getExample` aufrufen. Vergessen Sie nicht, in `printBoard` die Gültigkeit des Parameters `board` mit `checkBoard` zu prüfen!

Prüfen auf leere Felder

Implementieren Sie die *statische öffentliche Methode* **`boolean`** `boardFull(Entity[][] board)`, die **`true`** zurückgibt, wenn sich *keine leeren Felder* mehr auf `board` befinden; ansonsten **`false`**.

JUnit-Test: `SOSTest.testBoardFull`

Benutzereingaben

Erweitern Sie Ihre `main`-Methode wie folgt: Erstellen Sie eine *leere Belegung* der Größe 3×3 Solange es *noch leere Felder* gibt (`boardFull` liefert **`false`**), wiederhole:

- Geben Sie die *aktuelle Belegung* aus.
- Lesen Sie eine *Zeile*, eine *Spalte* und eine *Belegung* von der Eingabe ein:

```
Enter row: 3
Enter col: 0
Enter entry (S or O): s
```

- Für den Moment, setzen Sie den Eintrag — *ohne Prüfung*, ob das Feld bereits belegt ist, und ohne Prüfung auf ein `SOS`.

Testen Sie Ihre Implementierung, indem Sie:

- nach und nach *alle Felder belegen*, um zu testen ob Ihr Programm *korrekt beendet*.
- mit fehlerhaften Eingaben prüfen, ob *Ihr Programm richtig* reagiert (d.h. *nicht abbricht* und den Nutzer zu einer *erneuten Eingabe* auffordert).

In den folgenden Aufgaben, implementieren wir die *statische öffentliche* Methode **`int`** `move(Entry[][] board, Entry entry, int row, int col)`: `move` setzt den Eintrag `entry` auf `board` an die Position in Zeile `row` und in Spalte `col`. Als Ergebnis gibt `move` die Anzahl der *vervollständigten* `SOS` zurück.

`move` Teil 1 — Prüfen der Argumente

Deklarieren Sie `move` wie angegeben und implementieren Sie zunächst nur die *Prüfung der Argumente*!

- `board` prüfen Sie mit `checkBoard`.
- `entry` darf nur `S_UNSCORED` oder `O_UNSCORED` sein
- `row` und `col` müssen eine gültige *Zeile* und *Spalte* sein.
- Das zu belegende Feld muss *leer sein*.

Geben Sie zunächst `0` als Ergebnis zurück.

JUnit-Test: `SOSTest.testMoveInvalidArgument`

`move` Teil 2 — Setzen von `O`

Ergänzen Sie die Implementierung von `move` für den Fall, dass `entry` gleich `O_SCORED` ist! Ermitteln Sie dabei wieviele `SOS` dabei vervollständigt werden und geben Sie die Anzahl als *Ergebnis zurück*. Beispiel, für `row==1` und `col==1`:

Der Rückgabewert von `move` ist hier 2.

JUnit-Test: `SOSTest.testMoveScoreO`

Wenn Sie nicht weiter wissen, hier ein paar Hinweise:

- Mit diesem Zug können maximal *vier* `SOS` vervollständigt werden.
- Ein *vertikales* `SOS` entsteht, wenn an der Position `row-1` und `col` und an der Position `row+1` und `col` jeweils ein `S` stehen.
- Wandeln Sie die `S` und `O` erst in `$` und `0` um, wenn ein `SOS` vervollständigt ist.
- Definieren Sie eine *lokale Variable* `int score`, die Sie immer hochzählen, wenn ein `SOS` vervollständigt ist.
- *Achtung*: Es kann sein, dass das zu setzende `O` am Rand des Felds liegt — achten Sie darauf, dass Sie nicht auf *ungültige Felder* zugreifen.

`move` Teil 2 — Setzen von `S`

Ergänzen Sie schließlich die Implementierung von `move` für den Fall, dass `entry` gleich `S_SCORED` ist! Beispiel, für `row==2` und `col==2`:

Der Rückgabewert von `move` ist 4.

JUnit-Test: `SOSTest.testMoveScoreS`

Wenn Sie nicht weiter wissen, hier ein paar Hinweise:

- Mit diesem Zug können maximal *acht* `SOS` vervollständigt werden.
- Eine Möglichkeit für ein `SOS` ist, wenn an der Position `row-1` und `col` ein `O` und an der Position `row-2` und `col` ein `S` steht. Sie müssen also *zwei Felder* in jede der *acht Richtungen* „sehen“.
- *Achtung*: Achten Sie wieder darauf, dass Sie nicht Felder „außerhalb“ zugreifen.

Vervollständigen von `main`

Vervollständigen `main` so, dass nach jedem Zug die Anzahl der vervollständigten SOS angezeigt wird, z.B.:

```
...
Enter row: 2
Enter col: 5
Enter entry (S or O): S
Completed SOS: 1
...
```

Optional: Computer-Mitspieler

Implementieren Sie einen Computer-Mitspieler, der immer auf das Feld setzt, das die meisten SOS vervollständigt.

Vorschlag zur Vorgehensweise:

- Erweitern Sie `move` um das Argument **`boolean`** `pretend`. Ist `pretend` **`true`**, so gibt `move` nur die Anzahl der *vervollständigten* SOS zurück, *falls dort der entsprechende Eintrag gemacht werden würde* — ohne aber den Zug wirklich auszuführen!
- Der Computer-Mitspieler ermittelt mit Hilfe von `move` und `pretend==`**`true`** die vielversprechenste Stelle.
- Sollte nirgends ein SOS vervollständigt werden können, macht der Computer eine zufällige Wahl.

Aufgabe 2: Temperaturentwicklung in Landshut

Seit 2017 stehen die Daten des Deutschen Wetterdienstes für jeden unter der *Creative Commons BY 4.0 - License* zum Download bereit. So gibt es z.B. die Temperaturdaten der Station *Landshut-Reithof* (Stations-ID: 13710) im Stundentakt von 01.04.2008 bis 31.12.2023 hier zum Download. Diese Daten liegen dieser Aufgabenstellung in Form einer CSV-Datei („*comma-separated values*'") unter

`SupportMaterial/csv2plot/app/src/main/resources/produkt_tu_stunde_20080401_20231231_13710.txt`

Betrachten Sie sich diese Datei:

```
STATIONS_ID;MESS_DATUM;QN_9;TT_TU;RF_TU;eor
13710;2008040100;  3;   7.2;  81.0;eor
13710;2008040101;  3;   7.0;  82.0;eor
13710;2008040102;  3;   6.6;  83.0;eor
...
```

Bei einer CSV-Datei handelt es sich um eine *Tabelle*. Die Felder in jeder Zeile sind durch einen *Semikolon* (;) getrennt. Die erste Zeile zeigt die *Spaltennamen*, die restlichen Zeilen beinhalten die *Messungen*. In der ersten Spalte steht die Stations-ID 13710 (Landshut-Reithof). Interessant für uns sind noch folgende Spalten:

- `MESS_DATUM` — Zeitpunkt der Messung im Format `JahrMonatTagStunde`.
  Wichtig: Die Zeilen nach dem Aufnahmezeitpunkt aufsteigend sortiert.

- TT_TU — Temperaturwert in *Grad Celsius*

Ihre Aufgabe ist es, die Daten aus der CSV-Datei für eine *grafische Darstellung* (Plot) zusammenzufassen und umzuwandeln.

Vorbereitungen

Importieren Sie das Gradle-Projekt im Verzeichnis `SupportMaterial/csv2plot`:

- Die Java-Datei `CSV2Plot.java` beinhaltet bereits eine `main`-Methode, die die CSV-Datei mit den Temperaturdaten einliest und an die Methode `String csvToPlot(String csv)`. Ändern Sie *nicht die Implementierung* der `main`-Methode!
- Aktuell gibt die Methode einen *fixen String* in folgendem Format zurück:

```
2008/04 8.181806
2008/05 14.888575
2008/06 17.763092
2008/07 17.966353
```

  Jede Zeile beinhaltet einen Monat der Form `Jahr/Monat` und die *Durchschnittstemperatur* dieses Monats (z.B. 8.18 Grad Celsius im April 2008).
- Die `main`-Methode schreibt den *Inhalt des Strings*, der von `csv2plot` zurückgegeben wird, in die Datei `app/temperatures.txt`.
- Führen Sie das Programm aus und prüfen Sie, ob die aktuelle Version des Programs die *vier Werte* von oben in die Datei `temperatures.txt` schreibt.

Implementierung von `csvToPlot`

Implementieren Sie `csvToPlot`:

- Verarbeiten Sie Zeile für Zeile und *extrahieren* (*parsen*) Sie die das Jahr, den Monat und den Temperaturwert. Temperaturwerte von -999 stehen für *ungültige Messungen*. Diese müssen Sie ignorieren.
- Berechnen Sie für *jeden Monat* die *Durchschnittstemperatur*.
- Schreiben Sie die den Durchschnittswert für jeden Monat wie oben gezeigt in einen `StringBuilder` in je eine Zeile.
- Geben Sie den *resultierenden String* zurück.

Wenn Sie nicht wissen, wie Sie vorgehen sollen, hier ein paar Tipps:

- Zum Vergleich, hier die berechneten Werte der ersten vier Monate:

```
2008/04 8.181806
2008/05 14.888575
2008/06 17.763092
2008/07 17.966353
```

- Sie können einen `String` mit der Methode `String[] split(String pattern)` an den von `pattern` definierten Stellen (z.B. `"\r\n"` für Zeilenumbrüche) *auftrennen*.
- `String.substring(int startIndex, int endIndex)` liefert den *Teilstring* von Index `startIndex` bis `endString` (ausgeschlossen).
- `Double.parseDouble` und `Integer.parseInt` wandeln einen `String` in einen **`double`** bzw. **`int`** um.
- Die *erste Zeile* beinhaltet *keine Werte*!

- Summieren Sie die Temperaturwerte für einen Monat auf und berechnen Sie den Durchschnitt bei einem *Monatswechsel*. Hängen Sie das Ergebnis *direkt an den* `StringBuilder` an.
- Vergessen Sie *nicht den letzten Monat*!

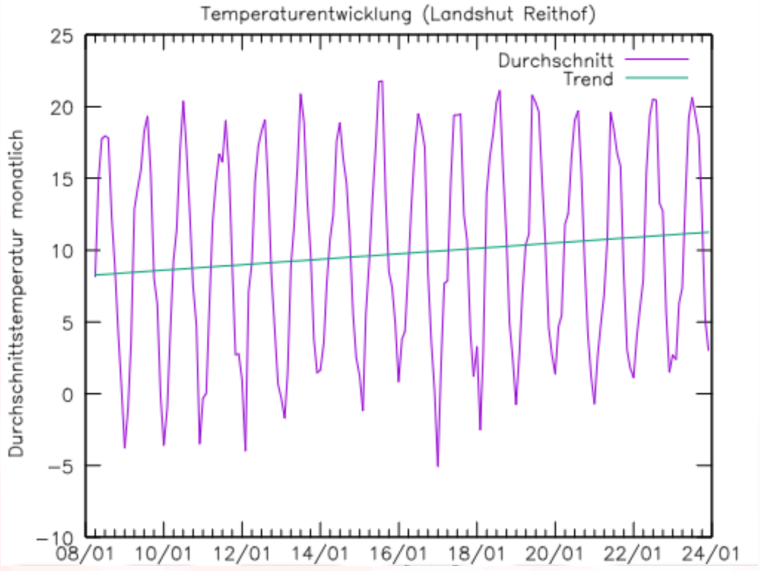
Optional: Prüfen und Plotten der Daten

gnuplot ist ein Opensource-Projekt zum Erstellen von *Graphen/Plots*. Die Daten in `app/temperatures.txt` können mit Hilfe von gnuplot grafisch dargestellt werden:

- Die *Fort Hays State University* bietet eine Online-Version von gnuplot an. Öffnen Sie die Seite in Ihrem Browser.
- Kopieren Sie folgendes gnuplot-Skript in das Feld *Script*

```
set title 'Temperaturentwicklung (Landshut Reithof)'
set xlabel 'Monat [J/m]'
set xdata time
set timefmt "%Y/%m"
set format x '%y/%m'
set ylabel 'Durchschnittstemperatur monatlich'
set autoscale
f(x)=b*x+c
fit f(x) 'data.txt' using 1:2 via b,c
plot 'data.txt' using 1:2 w l title "Durchschnitt", f(x) w l title "Trend"
```

- Kopieren Sie den Inhalt von `app/temperatures.txt` in das Feld *Data*.
- Sollte alles korrekt sein, taucht rechts oben ein Graph mit der Temperaturentwicklung in Landshut auf:



Die *violette Linie* zeigt die *monatliche Durchschnittswerte*. Sie sehen die Temperaturschwankungen durch die *Jahreszeiten*. Die *türkise Linie* zeigt den *linearen Trend* (*lineare Regression*), der Temperaturen — sie zeigen einen *Anstieg*.