Programmieren II (Java)

3. Praktikum: Arrays und Strings

Sommersemester 2023
Christopher Auer, Tobias Lehner

Abgabetermine

Lernziele

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

Hinweise

- 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: ChatCheapyT-Chatbot ★☆ bis ★★

Wir springen auf den aktuellen Hype um den Chatbot ChatGPT auf und schreiben unseren eigenen Chatbot. Allerdings sind unsere Resourcen doch arg begrenzt und so verwenden wir ein paar `String`-Methoden statt eines komplexen Sprachmodells. Wir nennen unseren Chatbot passenderweise *ChatCheapyT*. Als Startpunkt verwenden wir die Klasse ChatCheapyT.java, die bereits eine `main`-Methode enthält. Importieren Sie dazu das *Gradle-Projekt* in `SupportMaterial/chatcheapyt` in Ihre Entwicklungsumgebung.

Erster Test ★☆

Führen Sie `ChatCheapyT` aus! Es wird Ihnen ein Eingabeprompt angezeigt:

```
Prompt: _
```

Allerdings kann unser Chatbot noch nicht viel:

```
Prompt: Hallo ChatCheapyT!
ChatCheapyT: Ich verstehe Sie leider nicht!
```

Mit der Eingabe `bye` können Sie sich abmelden. Im Folgenden erweitern wir unseren Chatbot um *Ausgaberegeln*.

Ausgaberegel „Schweigen“ ★☆

Betrachten Sie sich die Methode `String handleSilence(String input)`! Die Methode prüft zunächst, ob die Eingabe `input` *leer* ist (`String.isBlank()`). Ist dies der Fall, gibt Sie die Antwort `"Dann sage ich aber auch nichts!"` zurück. Ansonsten **`null`**, was bedeutet, dass die *Ausgaberegel* nicht angewendet wurde.

ChatCheapyT übergibt die Eingabe an eine Reihe von Methoden, die nach diesem Schema funktionieren. Sobald eine dieser Methoden eine Antwort `!=` **`null`** zurückgibt, ist das die Antwort unseres Chatbots. Führen Sie ChatCheapyT aus, so dass die Ausgaberegel greift (z.B. indem Sie einfach Enter drücken)!

Weitere Ausgaberegeln (★☆ bis ★★)

Implementieren Sie folgende Regeln (die Methodendeklarationen sind bereits vorhanden). Testen Sie jede Ihrer Methoden durch verschiedene Eingaben. Ihre Implementierung sollte *robust* sein, d.h. durch eine bestimmte Eingabe nicht durch eine *Ausnahme* abstürzen.

- *Methode*: `handleTooLong`
  Bedingung: Anzahl der Zeichen größer als 50
  Ausgabe: `"Das ist mir zuviel zu lesen! Bitte kürzen Sie Ihre Anfrage!"`
- *Methode*: `handleExam`
  Bedingung: Eingabe ist *genau* `"Was kommt in der Klausur dran?"`
  Ausgabe: `"Die Klausur orientiert sich an den Praktika!"`

- *Methode*: `handleQuestion`
  Bedingung: Eingabe *endet* mit einem Fragezeichen
  Ausgabe: `"Tut mir leid, aber die ChatCheapyT-Server sind gerade ausgelastet! Schließen Sie bitte ein ChatCheapyT-Pro-Abo ab!"`
- *Methode*: `handleExclamation`
  Bedingung: Eingabe *beinhaltet* mindestens ein Ausrufezeichen
  Ausgabe: `Wie ist das Zauberwort?` falls die Eingabe *nicht das Wort* `bitte` enthält (Groß-/Kleinschreibung von `bitte` soll ignoriert werden). Enthält die Eingabe ein `bitte`, so ist die Ausgabe `"Als Antwort habe ich ein YouTube-Video generiert: https://youtu.be/dQw4w9WgXcQ"`.
- *Methode*: `handleChatGPT`
  Bedingung: Eingabe *beinhaltet* mindestens einmal genau das Wort `"ChatGPT"`
  Ausgabe: Alle Vorkommen von `"ChatGPT"` werden durch `"ChatCheapyT"` *ersetzt* und das Ergebnis zurückgegeben.
- *Methode*: `handleScream`
  Bedingung: Die Anzahl der *Großbuchstaben*, die groß geschrieben sind, macht *mindestens die Hälfte* der Eingabelänge aus.
  Ausgabe: `"Bitte schreien Sie mich nicht an!"`
- *Methode*: `handleReverse`
  Bedingung: Die Eingabe beginnt mit `"Umdrehen:"` (Groß-/Kleinschreibung werden *ignoriert*)
  Ausgabe: Die Ausgabe sind die Zeichen nach dem Doppelpunkt der Eingabe in *umgekehrter Reihenfolge*. Beispiel:

```
Prompt: Umdrehen: Hallo ChatCheapyT
ChatCheapyT: TypaehCtahC ollaH
```

  Verwenden Sie zur Erstellung der *umgekehrten Zeichenfolge* die Klasse `StringBuffer`.

Addieren ★★

ChatGPT hat manchmal Probleme mit dem korrekten *Rechnen*. Wir wittern hier unsere Chance ChatGPT zu übertrumpfen und implementieren eine Regel zur Addition. Beginnt die Eingabe mit `Addiere` (Groß-/Kleinschreibung *ignoriert*), dann werden die beiden darauf folgenden **`double`**-Zahlen addiert:

```
Prompt: Addiere 3.1415 -91.119
ChatCheapyT: 3.141500 plus -91.119000 ist gleich -87.977500! Take that, ChatGPT!
```

Hinweise:

- Um einen `String` an (Leer-)zeichen *aufzutrennen* gibt es die Methode `String.split`.
- Um aus einem `String` eine **`double`**-Zahl zu *parsen*, verwenden Sie `Double.parseDouble`.
- Sollte die Eingabe *fehlerhaft* sein, geben Sie eine Fehlermeldung zurück.

Weitere Ausgaberegel

Implementieren Sie *mindestens eine weitere* Ausgaberegel nach Ihrer Wahl und Kreativität!

Aufgabe 2: Schiffe bergen ★☆ bis ★★

In dieser Aufgabe implementieren wir ein einfaches Einspieler-Spiel, das an das bekannte Spiel „Schiffe versenken“ angelehnt ist. Aber anstatt die Schiffe zu versenken, *bergen* wir sie.

„Schiffe bergen“ spielt man auf einem 10×10-Feld:

```
   ABCDEFGHIJ
  +----------+
 1|          |
 2|          |
 3|          |
 4|          |
 5|          |
 6|          |
 7|          |
 8|          |
 9|          |
10|          |
  +----------+
```

Die Spalten sind mit `A` bis `J` bezeichnet, die Zeilen mit `1` bis `10`. Ein Feld wird im Format `SpalteZeile` identifiziert, z.B., `E6`. Unter jedem Feld kann sich potentiell ein versunkenes Schiff verbergen und wir schicken Taucher in der Hoffnung was zu finden, bspw. auf das Feld `E6`:

```
   ABCDEFGHIJ
  +----------+
 1|          |
 2|          |
 3|          |
 4|          |
 5|          |
 6|    X     |
 7|          |
 8|          |
 9|          |
10|          |
  +----------+
```

Das `X` bedeutet, dass wir dort leider nichts gefunden haben. Versuchen wir es daneben auf dem Feld `F6`:

```
   ABCDEFGHIJ
  +----------+
 1|          |
 2|          |
 3|          |
 4|          |
 5|          |
 6|    X#    |
 7|          |
 8|          |
 9|          |
10|          |
  +----------+
```

Wir haben ein versunkenes Schiff gefunden, dessen Ausmaße noch unbekannt sind. So kann es sich z.B. nach rechts, oben oder unten erstrecken. Jedes Schiff belegt mindestens zwei Felder, entweder in horizontaler oder vertikaler Richtung. Drei weitere Tauchgänge ergeben z.B.:

```
   ABCDEFGHIJ
  +----------+
 1|          |
 2|          |
 3|          |
 4|          |
 5|      X   |
 6|     X#   |
 7|      #   |
 8|      X   |
 9|          |
10|          |
  +----------+
```

Wir haben das ganze Schiff geborgen und die Suche nach mehr versunkenen Schiffen kann weitergehen. Im Folgenden implementieren „Schiffe bergen“ Methode für Methode. Um die Korrektheit Ihrer Implementierung zu testen, müssen Sie die gegebenen *JUnit-Tests* verwenden. Importieren Sie dazu das bereitgestellte *Gradle-Projekt* in `SupportMaterial/shipsalvage` in Ihre Entwicklungsumgebung. Sie sollten zusätzlich in der `main`-Methode ihre Methoden ausprobieren!

Das `enum FieldState` (★☆ bis ★★)

Zuerst müssen wir das **`enum`** `FieldState` definieren, das die Zustände eines Feldes angibt:

Konstante	`char output`	Beschreibung
`EMPTY`	`' '`	leeres Feld
`MISS`	`'X'`	leeres Feld, das untersucht wurde
`OCCUPIED_HIDDEN`	`'O'`	gesunkenes Schiff, nicht entdeckt
`OCCUPIED_SALVAGED`	`'#'`	gesunkenes Schiff, entdeckt

Das Attribut `output` gibt das Zeichen an, das für die Ausgabe verwendet wird.

Deklarieren Sie das **`enum`** `FieldState` wie oben definiert! Fügen Sie auch eine *öffentliche statische* Methode `FieldState fromOutput(`**`char`** `output)` ein, die für ein Zeichen den entsprechenden Wert zurückgibt, z.B., `MISS` für `'X'`. Erzeugen Sie eine `IllegalArgumentException` wenn es keinen Wert für das Zeichen gibt.

Die Klasse `ShipSalvage` und `getExample` (★☆)

Erstellen Sie nun eine *öffentliche* Klasse `ShipSalvage` mit einer `main`-Methode, die Sie zum Testen Ihrer bisherigen Lösung verwenden können. Deklarieren Sie eine *öffentliche statische* Methode `getExample()`, die eine Beispiel-Karte zurückgibt. Gehen Sie dazu wie folgt vor:

- Deklarieren Sie zunächst ein *privates, unveränderliches und statisches* Attribut `exampleMap` vom Typ zweidimensionaler **`char`**-Array.
- *Initialisieren* Sie das Attribut `exampleMap` über ein *Array-Literal* der Dimension 10×10. Wenn Sie sich nicht sicher sind, was ein *Array-Literal* ist, dann schauen Sie in den Vorlesungsunterlagen nach!

```
  0123456789
 +----------+
0|OO O  OOOO|
1|   O      |
2|O  O OOO O|
3|O        O|
4|O         |
5|O    O   O|
6|     O   O|
7|OO       O|
8|          |
9|    OOOOOO|
 +----------+
```

Die Zeichen `'O'` und `' '` entsprechen dabei den Ausgabesymbolen, wie sie in `FieldState` definiert sind (`EMPTY` und `OCCUPIED_HIDDEN`) und die Indizes am Rand den Indizes des zweidimensionalen Arrays.

- Implementieren Sie nun eine *öffentliche statische* Methode `getExample()`, die `exampleMap` in einen *zweidimensionalen* 10×10-Array vom Typ `FieldState` umwandelt und zurückgibt. Verwenden Sie dazu die Methode `FieldState.fromOutput` für die Umwandlung von **`char`** in `FieldState`.
- *Testen* Sie Ihre Implementierung mit dem Test `testGetExample`!

Die Methode `checkValidMap` (★★)

In den folgenden Methoden wird fast immer eine Karte (zweidimensionaler `FieldState`-Array) als Parameter übergeben, dessen Gültigkeit wir prüfen müssen. Diese Prüfung lagern wir in eine eigene *öffentliche statische* Methode **`void`** `checkValidMap(map)` aus, die als Parameter eine *Karte* übergeben bekommt, und auf Gültigkeit prüft. Sollte die Karte *ungültig* sein, so wird eine `IllegalArgumentException` mit *aussagekräftiger Fehlermeldung* erzeugt. Gültig ist die Karte `map`, wenn:

- weder `map` noch ein Eintrag `map[i]` **`null`** ist,
- `map` ein *quadratischer* zweidimensionaler Array von *genau der Dimension* 10×10 ist,
- kein Eintrag von `map` **`null`** ist.

Implementieren Sie `checkValidMap` und testen Sie Ihre Implementierung mit dem Test `testCheckValidMap`! Rufen Sie im Folgenden bei allen Methoden, die eine Karte als Parameter akzeptieren, die Methoden `checkValidMap`!

Ausgeben einer Karte (★★)

Implementieren Sie eine *statische öffentliche* Methode **void** `printMap(map, showHidden)`, die die übergebene Karte auf dem Terminal nach folgendem Format ausgibt:

```
   ABCDEFGHIJ
  +----------+
 1|          |
 2|     XXX  |
 3|  X       |
 4|  X  XXX  |
 5|  X     X |
 6|   XX# XX#|
 7| XX  #X X |
 8|     X    |
 9| XXX      |
10|          |
  +----------+
```

Die Zeichen entsprechen dabei dem Attribut `FieldState.output`. Der **boolean**-Parameter `showHidden` gibt an, ob die Felder mit dem Wert `OCCUPIED_HIDDEN` angezeigt werden sollen oder nicht. Im Normalfall ist der Wert von `showHidden` **false**, da die Schiffswracks nicht sichtbar sein sollen. Nur zur Fehlersuche („debuggen“) macht es Sinn, für `showHidden` **true** zu übergeben und dabei z.B. folgende Ausgabe (für `getExample` nach ein paar Versuchen) zu bekommen:

```
   ABCDEFGHIJ
  +----------+
 1|OO O  OOOO|
 2|   #      |
 3|O  # OOO O|
 4|O        O|
 5|O XXX     |
 6|O    #   O|
 7|     #   O|
 8|OO       O|
 9|          |
10|    OOOOOO|
  +----------+
```

Das Zeichen `'O'` entspricht dabei `OCCUPIED_HIDDEN`. Testen Sie Ihre Implementierung indem Sie die Ausgabe für verschiedene Karten erzeugen lassen!

Ende des Spiels ermitteln (★★)

Implementieren Sie eine *öffentliche statische Methode* **boolean** `allSalvaged(map)`, die für die übergebene Karte prüft, ob bereits alle Schiffe geborgen wurden! D.h. `allSalvaged` gibt **true** zurück, wenn kein Feld mit dem Wert `OCCUPIED_HIDDEN` mehr existiert; sonst **false**. Testen Sie `allSalvaged` mit dem Test `testAllSalvaged`!

Interpretieren der Benutzereingabe (★★)

Implementieren Sie eine Methode `probeField(map, field)`! Bei `field` handelt es sich um einen `String` in dem das Feld steht, das untersucht werden soll, z.B. `"G8"` für Spalte `G`, Zeile `8`. Das erste Zeichen in `field` ist die Spalte (`A` bis `J`), danach kommt die Zeile als Zahl. Die Methode `probeField` ermittelt das Feld, ändert es entsprechend dem aktuellen Wert und macht eine Ausgabe:

Alter Zustand	Neuer Zustand	Ausgabe
EMPTY	MISS	"Nichts zu finden!"
OCCUPIED_HIDDEN	OCCUPIED_SALVAGED	"Wrack gefunden!"
OCCUPIED_SALVAGED	OCCUPIED_SALVAGED	"Bereits untersucht!"
MISS	MISS	"Bereits untersucht!"

Hinweise:

- Sie können davon ausgehen, dass `field` keine Leerzeichen enthält (wenn doch, dann handelt es sich um eine *fehlerhafte* Eingabe).
- Die Spalte darf groß oder klein geschrieben werden.
- In dem zwei-dimensionalen Karten-Array entspricht der *erste Index* der *Zeile* und der *zweite Index* der *Spalte*. Bei der Notation, die vom Nutzer eingelesen wird, ist dies genau *umgekehrt*.
- Bei einer fehlerhaften Eingabe, soll eine *Fehlermeldung* ausgegeben werden und die *Karte unverändert* bleiben.
- Ihre Implementierung *muss* mit korrekten Eingaben funktionieren, was Sie mit dem Test `testProbeValid` prüfen können. Ihre Implementierung *sollte* auch mit *fehlerhaften Eingaben* umgehen können. Dies können Sie mit dem Test `testProbeFieldInvalid` überprüfen.

Hauptprogramm (★☆☆ bis ★★☆)

Implementieren Sie die `main`-Methode wie folgt:

- Starte mit der Karte von `getExample`
- *Solange* `allSalvaged` auf Karte **`false`** liefert, wiederhole:
  - Gib Karte aus (`printMap`)
  - Lies eine Zeile vom Nutzer ein (`Scanner.next()`)
  - Übergib Karte und Eingabe an `probeField`
- Sind alle Schiffe geborgen, gib eine Meldung (`"Alle Schiffe geborgen!"`) und ein letztes Mal die Karte aus.

Optional: Zufälliges Erzeugen von Karten (★★☆)

Anstatt immer die gleiche Beispielkarte zu verwenden, implementieren Sie in einer Methode `generateRandomMap` ein Verfahren, das zufällige Karten erzeugt. Dabei muss gelten:

- Es gibt eine bestimmt Anzahl an Schiffen je Größe:

Größe	Anzahl
2	4
3	3
4	2
6	1

- Jedes Schiff muss *vollständig* innerhalb der Karte liegen.
- Die horizontale und vertikale Ausrichtung wird ebenfalls *zufällig* gewählt.
- Keine zwei Schiffe dürfen sich *überlagern/schneiden*.

Hinweis: Verwenden Sie die Klasse `Random` (Zufallsgenerator), die eine Vielzahl von Möglichkeiten bietet zufällige Werte unterschiedlicher Typen zu generieren.