Übungsblatt 4: IO, Reguläre Ausdrücke und XML

Abgabetermin: 26.5.2017, 23:59 Uhr

Aufgabe 1: Einlesen mit `Scanner` und `BufferedReader` (2 + 2 + 1)

Lade die Datei `bigInput.txt` aus Moodle herunter. Die Datei enthält zehn Millionen Zahlen, jede auf einer eigenen Zeile.

1. Schreibe ein Programm, das `new Scanner(new FileInputStream("bigInput.txt"))` verwendet um alle Zeilen als `int` einzulesen und ihre Summe auszugeben. Messe dabei die Zeit, die dein Programm braucht um alle Zahlen einzulesen. Du kannst dafür zum Beispiel`System.currentTimeMillis()` [3] verwenden.
2. Schreibe ein zweites Programm, das die `BufferedReader`-Klasse verwendet, um alle Zeilen als `String` einzulesen. Konvertiere diese Zeilen mit `Integer.parseInt()` zu `Integer` und gib die Summe aller Zahlen aus. Messe dabei wieder die Zeit, die dein Programm braucht.
3. Wie lange brauchen die beiden Programme in Millisekunden? Erkläre, warum eines der Programme deutlich langsamer ist.

Hinweis: Die Summe aller Zahlen passt nicht in einen `int` oder `Integer`. Du kannst für die Summe stattdessen `long` verwenden.

Aufgabe 2: Sokoban – Benutzereingabe (4)

Im Verlauf der Vorlesung werden wir das Sokoban Spiel erweitern, immer passend zum aktuellen Vorlesungsstoff. (Das heißt, am Ende werden wir auch eine GUI schreiben.) Weil diese Entwicklung von den Übungsblättern unabhängig ist, tue Folgendes:

- Erzeuge ein neues maven-Projekt basierend auf dem *maven-archetype-quickstart* archetype.
- Gib dem Projekt die *Group-Id* **de.uulm.sp.pvs** und die *Artifact-ID* **sokoban**.
- Achte darauf dass sowohl der Java Compiler als auch die JRE System Library des Projektes eine möglichst aktuellen Java Version enthalten (Java Compiler und Java Build Path Einstellungen in den Projekt-Eigenschaften). Hierfür ist es auch notwendig die *maven.compiler.source* und *maven.compiler.target* Einstellungen unter `<properties>` in der *pom.xml* auf die korrekte Java version zu setzen. D.h 1.8 für Java 1.8 oder 11 für Java 1.11/11. (Siehe Abbildung 1)
- um ein möglichst einfaches erzeugen einer jar des Spiels zu ermöglichen nutze das Assembly-Plugin [7]. Füge dafür den Code aus Abbildung 2 in deine *pom.xml* ein.
- Erzeuge ein *package* mit dem damen `de.uulm.sp.pvs.util` und kopiere deine bisherige Lösung (die Sokoban und Pair Klassen) in dieses.

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
```

Abbildung 1: Maven-Compiler Code

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>fully.qualified.MainClass</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Abbildung 2: Maven Assembly-Plugin Code

Erweitere nun dein Sokoban Programm vom letzten Übungsblatt so, dass es auf Benutzereingaben reagiert (der Code dafür kommt in die erzeugte `App.java` Klasse). Gib dazu in einer Schleife den aktuellen Zustand des Sokoban-Arrays aus und frage den Benutzer nach einer Richtungsangabe in der Konsole. Wie du die Benutzereingabe genau abfragst bleibt dir überlassen. Auch kannst du dir überlegen ob du dem Nutzer Feedback über seine Aktion geben möchtest (wenn z.B die Bewegung nicht ausgeführt werden konnte).

Hier ein Beispiel:

```
#######
#@$.$.#
#######
Where do you want to go? (N/E/S/W or X to exit)
> E
#######
#.@$$.#
#######
Where do you want to go? (N/E/S/W or X to exit)
> X
Bye.
```

Hinweise:

- Um dein Projekt zu bauen führe `mvn clean compile assembly:single` in deinem Projekt aus.
- Um dein Programm auszuführen nutze: `java -cp target/sokoban-0.0.1-SNAPSHOT-jar-with-dependencies.jar de.uulm.sp.pvs.sokoban.App`.

Aufgabe 3: Log-Parsing mit Regulären Ausdrücken (0 + 3 + 2 + 1)

Die Datei `build.log` im Moodle enthält das Log eines Kompiliervogangs, bei dem ein Compiler sehr viele Warnungen und andere Ausgaben erzeugt hat. Deine Aufgabe ist es, den Inhalt des Logs mithilfe von Regulären Ausdrücken aufzubereiten. Dazu sollst du alle Warnungen mit ihrer Nummer, der Datei die sie betreffen und ihrer Zeilennummer aus dem Log extrahieren und ausgeben.

1. Die Klassen `Matcher` [1] und `Pattern` [2] implementieren Reguläre Ausdrücke in Java. Recherchiere, wie sie verwendet werden.
2. Im folgenden siehst du einen Ausschnitt aus `build.log` ab Zeile 9712 (wichtige Teile sind farblich hervorgehoben):

```
9712  Warning␣552␣of␣731:
9713
9714  ␣␣in␣module␣DOM.Node.Types
9715  ␣␣at␣/home/.../Types.purs␣line␣8,␣column␣1␣-␣line␣10,␣column␣1
9716
9717  ␣␣␣␣Import␣uses␣the␣deprecated␣'qualified'␣syntax:
9718  ␣␣␣␣ // more lines
```

   Schreibe einen Regulären Ausdruck mithilfe der `Pattern`-Klasse, der in einem `String` die gezeigte Warnung findet. Verwende Capture Groups um die Nummer der Warnung, die Datei und die Zeilennummer zu extrahieren, sieh dir dazu die `Matcher`-Klasse an.

3. Lese die gesamte `buid.log` Datei in einem `String` ein und finde alle Warnungen darin. Gib für jede Warnung die Nummer der Warnung, die Zeilennummer und die Datei **in dieser Reihenfolge** aus. Hier ist eine Beispielausgabe: `Warning 552 at line 8 in file /home/.../Types.purs`
4. Gebe zusätzlich am Ende die Anzahl der gefundenen Warnungen aus.

Hinweise:

- Die Log-Datei enthält 747 Warnungen.
- Der Pfad der Dateien enthält keine Leerzeichen.
- Achte beim Einlesen der gesamten Datei darauf, dass die Zeilenumbrüche nicht verloren gehen.
- Du kannst den Buchstaben `'a'`, gefolgt von einen Zeilenumbruch, gefolgt vom Buchstaben `'b'` zum Beispiel so finden: `new Pattern("a\nb")`

Aufgabe 4: Die Neuerfindung des Rades (10)

Das Spielen von Sokoban in einer Konsole wie in Aufgabe 2 erinnert sehr an CLI [5]-Anwendungen aus UNIX. Diese brüsten sich mit Features wie Tab-Completion und Command-Histories.

In dieser Aufgabe erweitern wir unsere Sokoban-Anwendung um genau diese Features. Weil diese Features schon so bekannt sind gibt es bereits fertige Implementierungen auf die wir hier zurückgreifen können so dass wir nicht alles selbst neu implementieren müssen. Jline3 [6] ist eine dieser Implementierungen.

1. Füge jline3 als dependency zu dem pom.xml file deines Sokoban Projektes hinzu. Den nötigen Code dazu findest du unter *Maven Usage* auf der github Seite.
2. Erweitere deine Lösung aus Aufgabe 2 so, dass statt dem Standard Terminal ein jline Terminal verwendet wird, welches `System.in` als input-Stream und `System.out` als output-Stream verwendet und sowohl Tab-Completion als auch eine Command-History unterstützt. Siehe Abbildung 3 für ein Beispiel.

Hinweise:

- Die drei wichtigsten Klassen für diese Aufgabe sind `Terminal`, `Completer` und `LineReader`
- Verwende das Factory-Pattern für das `Terminal` und `LineReader`.
- Die Angepriesenen Funktionalitäten funktionieren **NICHT** wenn das Programm von der IDE aus gestartet wird. Es ist notwendig das maven-Projekt zu bauen und die resultierende jar auszuführen!

```
Press Tab to see a list of available commands.
#######
#.....#
#..$..#
#.$@$.#
#..$..#
#.....#
#######

sokoban>
EXIT          MOVE_EAST     MOVE_WEST     REDO_GAME     MOVE_NORTH    MOVE_SOUTH

#######
#.....#
#..$..#
#$@.$.#
#..$..#
#.....#
#######

sokoban>MOVE_
MOVE_EAST     MOVE_WEST     MOVE_NORTH    MOVE_SOUTH
```

Abbildung 3: Beispiel der Tab-Completion

Abgabe

Beachte bei der Abgabe folgendes:

- Die Bearbeitung kann in Gruppen erfolgen, muss jedoch nicht. Wichtig ist, dass, sollte die Abgabe in einer Gruppe erfolgen die Namen der Mitglieder erkenntlich sein sollten.
- Die Abgabe erfolgt über Moodle.
- Der abgegebene Code muss kompilieren.
- Alle von Ihnen erzeugten Klassen und Methoden müssen ausreichend mit JavaDoc und normalen Kommentaren versehen sein.
- Alle Fragen müssen hierbei sinnvoll beantwortet sein.

Fragen

Wenn du Fragen hast, nutze bitte das Forum im Moodle Kurs oder frage deinen Tutor.

Literatur

[1] https://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html

[2] https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

[3] https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis%28%29

[4] https://docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html

[5] https://en.wikipedia.org/wiki/Command-line_interface

[6] https://github.com/jline/jline3

[7] https://maven.apache.org/plugins/maven-assembly-plugin/