Prof. Dr. Christoph Bockisch
MSc. Steffen Dick
MSc. Teresa Dreyer
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Programmierpraktikum: Sommersemester 2023

Aufgaben Tag 1

Aufgabe 1: Serpensortia! 23 Punkte

Wir wollen heute und morgen zusammen mit Ihnen einen Interpreter für eine einfache Shape-Sprache in Java schreiben. Zweck der Sprache ist es, Shapes zu erzeugen, die als json-Datei ausgegeben werden. Die Sprache beinhaltet Variablenzuweisungen, Kontrollfluss, arithmetische und Boolesche Ausdrücke, Labels, jump-Anweisungen und draw-statements. Mit einem draw-statement wird ein Shape beschrieben, mit den anderen Sprachelementen wird das "Drumherum" erzeugt, so dass man unter anderem automatisch eine Reihe von Shapes erstellen kann. Ziel ihres Interpreters ist die Übersetzung zwischen eingelesenem Programm und JSON-Repräsentation. Dazu erhält Ihr Interpreter wird eine Datei mit dem Programmcode erhalten und soll am Ende die JSON-Datei als Ergebnis liefern. Sie können Ihre erzeugte Datei unter `https://se.mathematik.uni-marburg.de/swt/ppt` testen und das generierte Bild anzeigen lassen.

Ein Interpreter ist ein Programm, das Programmcode einliest, übersetzt und direkt ausführt. Im Gegensatz dazu steht ein Compiler, der Programmcode in Maschinensprache übersetzt. In diesem Fall wollen wir, dass der eingelesene Code in JAVA übersetzt und direkt ausgeführt wird. Sie erhalten für jeden Schritt ein kleines Programm, mit dem Sie testen können, ob Ihre Implementierung funktioniert.

Im Ilias finden Sie ein IntelliJ-Projekt, das Sie als Vorlage verwenden können. In diesem Projekt ist bereits eine Klasse `Tokenizer` vorhanden. Ein `Tokenizer`, auch Lexer genannt, nimmt für Sie bereits die *lexikalische Analyse* des Codes vor. Die *lexikalische Analyse* übersetzt den eingelesenen Code zunächst in eine Reihe von sinnvollen Einheiten, auch *Tokens* genannt, welche ebenfalls eine Kategorisierung erhalten. Die Kategorisierung ist für die weitere Analyse des Codes wichtig. Innerhalb unseres Programmes werden Anweisungen mit einem Zeilenumbruch abgeschlossen. Der Zeilenumbruch ist also vergleichbar mit einem `;` in JAVA.

Bitte beachten Sie: Um für die Aufgaben Punkte zu erhalten, müssen Sie eine Form von Fehlerbehandlung haben. Insbesondere sollte Ihr Programm mit unerwarteten Eingaben umgehen können.

a) Beginnen wir zunächst mit der Klasse `Shape`. Die Klasse `Shape` braucht die folgenden Eigenschaften: **2**
   - `type`, für den Typen der `Shape`. Hier dürfen lediglich `triangle`, `circle` oder `quad` eingetragen werden.
   - `color` soll eine Farbe als Hexadezimalzahl im Format *RGBA* sein und beschreibt die Farbe der Linie.
   - `fillColor` soll eine Farbe als Hexadezimalzahl im Format *RGBA* sein und beschreibt die Farbe, mit der die Shape ausgemalt werden soll.
   - `lineWidth` ist eine Zahl und beschreibt die Dicke der Linie.
   - `positionX` ist eine Zahl und beschreibt die X-Koordinate des Mittelpunkts der Shape.
   - `positionY` ist eine Zahl und beschreibt die Y-Koordinate des Mittelpunkts der Shape.
   - `scaleX` ist eine Zahl und beschreibt die Verzerrung in X-Richtung der Shape.
   - `scaleY` ist eine Zahl und beschreibt die Verzerrung in Y-Richtung der Shape.
   - `rotation` beschreibt die Drehung der Shape um ihren Mittelpunkt in Grad.
   - `content` soll eine Shape sein. Dieses Feld soll zunächst `null` als Wert haben.

b) Implementieren Sie die folgenden Methoden in `Shape`: **4**

- Einen Konstruktor, der alle Felder, bis auf `content` als Argumente erhält und diese setzt.
- Eine Set- und Get-Methode für `content`
- Eine Methode `String toJson()`, die die Klasse in ein JSON-Objekt konvertiert. Informieren Sie sich hierzu im Internet, welche Möglichkeiten Ihnen hier zur Verfügung stehen. Sie dürfen für diesen Aufgabenteil Bibliotheken verwenden.

c) Implementieren Sie eine Klasse `Interpreter`. Legen Sie in dieser Klasse eine Methode `String interpret(List<Token> tokens)` an. Diese Methode soll im Verlauf der folgenden Aufgaben immer weiter befüllt werden. **1**

d) Als erstes Konzept wollen wir ein `draw`-Statement erlauben. Ein `draw`-Statement hat die folgende Form im Code: **4**

```
draw circle #000000ff #00ffffaa 2 20 20 10 20 0
```

e) Implementieren Sie nun eine Main-Methode, mit der Sie eine Code-Datei einlesen können. Verwenden Sie danach den `Tokenizer` und Ihren `Interpreter`, um das eingelesene Programm auszuführen. Verwenden Sie das folgende kurze Programm, um Ihre Implementierung zu testen: **2**

```
draw circle #000000ff #00ffffaa 2 20 20 10 20 0
draw triangle, #000000ff #f0daaaaa 2 20 20 10 20 90
draw quad #000000ff #00ffffaa 2 20 20 10 20 90
```

f) Als nächstes Konzept wollen wir Variablen einführen. Eine Variable soll mit `var <name> = <value>` erschaffen werden können, wobei `<name>` durch den Namen und `<value>` durch den Wert der Variable ersetzt werden sollen. Eine erneute Zuweisung an eine Variable soll ebenfalls mit `var <name> = <value>` erfolgen können. Variablen dürfen an jeder beliebigen Stelle auftauchen, an der auch ein anderer Wert stehen könnte. **4**

Verwenden Sie das folgende Programm, um Ihre Implementierung zu testen:

```
var lineColor = #000000ff
var degree = 0
draw circle lineColor #00ffffaa 2 20 20 10 20 degree
var degree = 90
draw triangle lineColor, #f0daaaaa 2 20 20 10 20 degree
draw quad lineColor #00ffffaa 2 20 20 10 20 degree
```

g) Variablen sind ja ganz schön und nett, allerdings wollen wir auch ein wenig mit den Variablen anstellen können. Daher sollen Sie nun arithmetische Ausdrücke einführen. Für arithmetische Ausdrücke sollen die vier Grundrechenarten in der Infix-Schreibweise (1 + 2) implementiert werden. Für Ausdrücke gelten die folgenden Regeln: **6**

- Ausdrücke dürfen nur beim der Zuweisung von Variablen verwendet werden.

  Erlaubt ist `var x = 10 + 10`, aber nicht `draw 10 + 10 [...]`.
- Ausdrücke werden von Links nach Rechts ausgewertet. Es wird dabei keine Operator-Präzedenz berücksichtigt.
- In Ausdrücken können keine Klammern verwendet werden.

Verwenden Sie das folgende Programm, um Ihre Implementierung zu testen:

```
var lineColor = #000000ff
var degree = 0
draw circle lineColor #00ffffaa 2 20 20 10 20 degree
var degree = degree + 90
draw triangle lineColor #f0daaaaa 2 20 20 10 20 degree
draw quad lineColor #00ffffaa 2 20 20 10 20 degree
```