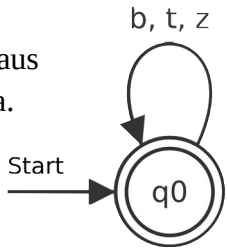


Implementierung des Lexers, Parsers und Interpreters der Sprache MyKa

Aufgaben:

1. Bei der Auswertung des Quelltextes wird auf eine Tokenliste zurückgegriffen. Überlege dir zunächst, welche Token benötigt werden. Gib dazu die verschiedenen Typen und ihre zugehörigen Werte an. Hierbei hilft es sich die Befehlsübersicht der Sprache anzuschauen. Mach dich **dann** mit der Klasse `Token.java` vertraut und vollziehe die Funktionsweise nach. Überprüfe ob und wie deine Überlegungen in der Klasse `Token.java` abgebildet werden können.
 2. Der Lexer soll als endlicher Automat implementiert werden, der zudem eine Tokenliste erzeugt. Da es sich nicht um einen reinen Deterministischen Endlichen Automaten (DEA) handelt, da auch Ausgaben erzeugt werden, sind Abwandlungen nötig. Entwickle einen Automaten der einen Text akzeptiert, der aus Wörtern (aus Buchstaben b), Zahlen (aus Ziffern z) und Trennzeichen in Form von u.a. Zeilenwechselln und Leerzeichen besteht.
Das Eingabealphabet besteht also aus Ziffern z, Buchstaben b und Trennzeichen t.
Diese Aufgabe würde auch von dem einfachen Automaten (siehe Abbildung rechts) erfüllt. Allerdings soll der Automat so gestaltet sein, dass immer dann, wenn ein Token erzeugt werden soll, ein Zustandswechsel stattfindet. Erstelle einen solchen Automaten und markiere die Übergänge bei denen ein Token erzeugt werden soll.
Zusatzaufgaben: Wie können die Kommentare implementiert werden?
- 
- ```
graph LR; Start((Start)) --> q0(((q0))); q0 -- "b, t, z" --> q0;
```
- a. Von dieser Klasse wird kein Objekt erzeugt. Alle Variablen und Methoden sind statisch. Wo erkennt man das am Quelltext? Worauf musst du bei der Erstellung von Attributen somit achten - Hinweis `private`?
  - b. Der aktuelle Zustand soll in einem Attribut `zustand` gespeichert werden und eine Fehlersenke soll implementiert werden
  - c. Es muss eine Liste von Token erzeugt werden (`List.java` aus den Abiturklassen)
  - d. Die für die Generierung des Tokens benötigte Zeichenfolge soll in einem String-Attribut (zwischen) gespeichert werden. Nach Erzeugung eines Tokens muss dieses Attribut wieder zurückgesetzt werden.
  - e. Es sollen Fehlermeldungen generiert werden (`System.err.println`) sobald ein unzulässiger Übergang auftritt oder aus der Zeichenfolge kein Token generiert werden kann (z.B. Fehler-Ziffer-in-Wort, Token wiederhole nicht bekannt, ...)
  - f. **Zusatzaufgaben:** Kannst du in einem Token auch die Position im Quelltext speichern zu der das Token erzeugt wurde (ggf. sogar die Zeile?)  
Wandle die Liste in ein `Atray` um. Gib die Tokenliste auf der Konsole aus

4. Plane nun den Parser. Erstelle hierzu eine kontextfreie Grammatik, insbesondere die Produktionsregeln, für die MyKa-Sprache.  
Überführe diese Grammatik **danach** in einen Kellerautomaten (überlege die passende Symbole für den Kellerspeicher)  
Überlege dir wie die Zustandsübergänge in deinem Automaten aussehen, wenn Schleifen und wenn-Abfragen untereinander oder ineinander verschachtelt sind.
5. Implementiere den Parser als Kellerautomat mit einem Kellerspeicher (Stack.java aus den Abiturklassen). Nutze hierzu auch die Dateivorlage - implementiere die Zustandsübergänge mit einer switch-case-Anweisung (ggf. geschachtelt oder kombiniert mit if-else-Anweisungen). Der Parser soll auch sinnvolle Fehlermeldungen generieren sobald er in die Fehlersenke wechselt.  
Ob du den Parser so nutzt, dass er als Input die Tokenliste oder ein Tokenarray verwendet ist dir überlassen.
6. Implementiere den Interpreter! Hier ist es sinnvoll mit dem Tokenarray zu arbeiten, da z.B. bei einer Wiederholung zu einer Position zurückgesprungen werden muss.  
Verwende die Dateivorlage und vollziehe die Aufgabe der in der Datei verwendeten Methoden nach, in dem du den jeweils zugehörigen Kommentar liest.

## Erläuterungen

- In allen Dateien (Lexer.java, Parser.java, Interpreter.java) gibt es eine Methode `public static void debug(String text)`, die abhängig vom Klassenattribut `debug` (boolean) eine Ausgabe erzeugt, so dass debug-Ausgaben durch das Attribut ein- bzw. ausgeschaltet werden können.
- Mit `private static final int T_Move = 0;` werden Bezeichner für Zustände oder Token-Typen festgelegt, die im Quelltext zu besserem Verständnis und Nachvollziehbarkeit führen. Es ist leichter nachvollziehbar und besser lesbar, wenn da steht `zustand = Z_wiederhole` als `zustand = 3`. Man erkennt unmittelbar die Bedeutung des nächsten Zustands oder die Funktion eines Tokens.

# Befehlsübersicht der MyKa-Sprache

Befehle/Anweisungen:

- Schritt
- LinksDrehen
- RechtsDrehen
- Hinlegen
- Aufheben
- MarkeSetzen
- MarkeLöschen

Bedingungen

- IstWand / NichtIstWand
- IstMarke / NichtIstMarke
- IstZiegel / NichtIstZiegel

Steuerung

- wiederhole {n} mal {Anweisungen} endewiederhole
- wiederhole solange {Bedingung} {Anweisungen} endewiederhole
- wenn {Bed} dann {Anweisungen} [sonst {Anweisungen}] endewenn

**Kommentare** können in geschweiften Klammern eingefügt werden {Kommentar} und werden ignoriert. Sie dürfen jedoch nicht unmittelbar an eine Anweisung/Bedingung/Steuerung angehängt werden