Opgave 3 [30%] (Recursive-descent parsing)

In deze opgave wordt een herkennende parser gevraagd voor de volgende grammatica voor eenvoudige programma's:

```
<program>    ::= <statements> .
<statements> ::= { <statement> ';' } .
<statement>  ::= <increment> | <decrement> | <assignment> | <repetition> .
<increment>  ::= 'INC' <var> [ <value> ] .
<decrement>  ::= 'DEC' <var> [ <value> ] .
<assignment> ::= 'LET' <var> '=' <value> .
<repetition> ::= 'WHILE' <var> 'DO' <statements> 'END' .
<value>      ::= <var> | <num> .
<var>        ::= <char> { <char> } .
<num>        ::= <digit> { <digit> } .
<char>       ::= 'a' | 'b' | ... | 'z' .
<digit>      ::= '0' | '1' | ... | '9' .
```

De karakters in een variabelenaam (hulpsymbool <var>) en de cijfers in een getal (hulpsymbool <num>) staan aaneengesloten; tussen alle andere elementen mag een willekeurige hoeveelheid witruimte staan. Onder witruimte verstaan we spaties, tab- en regelovergangs-symbolen. Een programma dient gevolgd te worden door een einde-bestand (ook wel end-of-file) symbool.

(a) Beschrijf wat een tokenizer doet.

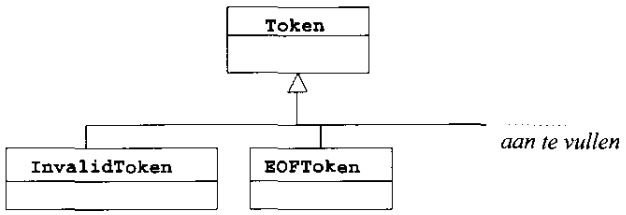
(b) Gesteld dat je een Tokenizer voor bovenstaande grammatica zou moeten gaan maken, geef aan welke tokens deze kunnen opleveren. Je mag uit gaan van een abstracte klasse Token met de volgende publieke constructoren en methoden:

```
/** creeert een Token met stringrepresentatie s */
public Token(String s)

/** creeert een Token met characterrepresentatie ch */
public Token(char ch)

/** retourneert de stringrepresentatie van het Token */
public String toString()
```

Om je op weg te helpen hebben we hieronder een deel van de Token-hiërarchie gegeven; vul deze verder in. (Je mag ook volstaan met een opsomming van alle tokens).



>>> lees verder >>>

Ga er vanuit dat Tokenizer een klasse is die de opgesomde tokens oplevert. Hieronder geven we nog kort even de beschikbare constructoren en methoden van de klasse Tokenizer zoals deze op college en het practicum aan de orde zijn geweest:

```
public Tokenizer(InputStream is) // constructor

public Tokenizer(Reader r)       // constructor

/** Levert het huidige token als resultaat */
public Token getCurrent()

/** Schuift de tokenizer door naar het volgende token */
public void moveNext()
```

Gevraagd wordt een recursive-descent parser te schrijven voor de gegeven grammatica. We geven alvast een klasse Parser

```
absract class Parser {
  protected static void reportError(String message) {
    System.out.println("FOUT: " + message);
    System.exit(0);
  }
}
```

(c) Beschrijf voor welke hulpsymbolen er subklassen van `Parser` gemaakt moeten worden.

(d) Geef voor elk van deze klassen de bijbehorende implementatie van de volgende methode

```
public static boolean tryParse(Tokenizer tok)
```

Merk op dat er niet gevraagd wordt een parse-tree op te bouwen, of excepties op te gooien. Er wordt slechts een *herkennende* parser gevraagd. Bij het constateren van een fout is het voldoende om de reportError methode aan te roepen.

>>> volgende opgave >>>

Opgave 1 [20%] (OO-theorie)

Belangrijke begrippen in object-georiënteerd programmeren zijn: *klasse*, *object*, *abstracte* klasse en *interface*

(a) Beschrijf kort en duidelijk het verschil tussen de begrippen *klasse* en *object*.
(b) Beschrijf kort en duidelijk wat een *abstracte* klasse is.
(c) Leg uit wat de overeenkomsten en verschillen tussen een *abstracte* klasse en een *interface* zijn.

Geef zo mogelijk voorbeelden om je antwoorden te verduidelijken!

Opgave 2 [30%] (Datastructuren)

In deze opgave wordt gevraagd een zogenaamde *bag* datastructuur te maken. Een bag is niet veel anders dan een multiset, d.w.z. een verzameling waarin waarden meermalen mogen voorkomen. Bijvoorbeeld $\{1, 3, 1, 1, 1, 3, 2\}$ is een multiset met 7 waarden, waarin waarde 1 viermaal voorkomt, waarde 2 eenmaal voorkomt en waarde 3 tweemaal voorkomt.

Een bag is een lijst van elementen die bestaan uit een waarde en een *multipliciteit* (het aantal voorkomens van de waarde). De lijst is geordend; kleine waarden komen voor grote waarden. De multipliciteit is altijd positief; waarden met nul voorkomens slaan we niet op.

De voorgenoemde multiset zou met een bag als de volgende lijst van elementen worden gerepresenteerd: $< (1, 4), (2, 1), (3, 2) >$.

We representeren een element in de bag met de volgende klasse:

```
class BagNode {
    int val;        // de opgeslagen waarde
    int mult;       // multipliciteit: aantal voorkomens van waarde val

    BagNode next; // het volgende element in de bag

    BagNode() {    // default constructor
        val = 0;
        mult = 1;
        next = null;
    }

}
```

(a) Voorzie de klasse BagNode van een constructor met een **int** als parameter, de waarde van een nieuw element.

>>> lees verder >>>

De bag zelf representeren we met de klasse Bag. We geven alvast een aanzet:

```
class Bag {
    BagNode head;  // kop van de lijst

    public Bag() { // default constructor
        head = null;
    }

}
```

In de volgende onderdelen van deze opgave wordt gevraagd deze gegeven code aan te vullen. Het staat je hierbij vrij om, waar je dat nodig acht, hulpmethoden te introduceren.

(b) Voeg aan de klasse Bag een methode

```
public int size()
```

toe, die het totaal aantal in de bag opgeslagen waarden oplevert.

(c) Voeg aan de klasse Bag een methode

```
public boolean contains(int v)
```

toe, die oplevert of een waarde v in de bag is opgeslagen.

(d) Voorzie de klasse Bag van een methode

```
public void add(int v)
```

die een waarde v toevoegt aan de bag. Zorg er voor dat de lijst die de bag representeert na afloop nog steeds voldoet aan de volgende eigenschappen:

- de lijst van elementen is geordend van kleine naar grote waarden,
- de lijst bevat geen meerdere elementen met gelijke waarde.

(e) Implementeer in de klasse Bag een methode

```
public void remove(int v)
```

die één voorkomen van de waarde v uit de bag verwijdert, indien deze waarde in de bag is opgeslagen.

Zorg ervoor dat na afloop de lijst geen elementen met multipliciteit 0 bevat, en dat deze nog steeds voldoet aan de eigenschappen genoemd bij het vorige onderdeel.

>>> volgende opgave >>>

Opgave 4 [25%] (I/O en excepties)

De volgende expressie voldoet aan de grammatica van de vorige opgave:

```
LET x = 1; LET y = 4; WHILE y DO LET z = y;
WHILE z DO INC x z; DEC z; END; DEC y; END;
```

Een (goed geschreven) parser heeft er helemaal geen moeite mee om de structuur van deze expressie te ontrafelen. Een menselijke lezer heeft echter meer profijt van een betere layout. Het bovenstaande programma is veel beter te begrijpen als we wat meer indentatie en regelovergangen introduceren. Bijvoorbeeld:

```
LET x = 1;
LET y = 4;
WHILE y DO
  LET z = y;
  WHILE z DO
    INC x z;
    DEC z;
  END;
  DEC y;
END;
```

Door een nieuw blok van statements twee spaties in te laten springen (indentatie) t.o.v. de omgeving en elk simpel statement (increment, decrement en assignment) op een nieuwe regel te plaatsen, onstaat een opmaak die de structuur van het programma beter weergeeft.

Schrijf een programma dat met één parameter wordt aangeroepen. Deze parameter is de naam van een bestand waarvan je programma ten eerste moet nagaan of de inhoud van dit bestand voldoet aan de grammatica van **opgave 3**. Is dat niet zo, dan dient een foutmelding afgedrukt te worden. Voldoet de inhoud wel aan de grammatica, dan moet een geformatteerde versie zoals hier boven beschreven van deze inhoud naar het scherm afgedrukt worden.

Zorg voor een nette afhandeling van situaties die niet aan de gestelde eisen voldoen. Maak daar waar nodig gebruik van excepties.

Aanwijzingen:

- Als de invoer aan de grammatica voldoet, is het redelijk eenvoudig om de uitvoer in het gevraagde formaat op te leveren. Het is *echt niet nodig* om de expressie eerst in een boomstructuur op te slaan. Kijk goed naar wanneer er een regelovergang moet komen, en wanneer het inspringniveau groter danwel kleiner moet worden.
- Voor het inlezen kent Java een FileReader en BufferedReader. In **opgave 3** staan de constructoren van de Tokenizer.
- Pas allereerst de parser uit **opgave 3** aan om de geformatteerde uitvoer op te leveren. De tryParse methoden zullen wellicht geen Boole'se resultaat moeten opleveren. De reportError methode uit Parser moet wellicht herschreven worden om een exceptie op te gooien. Schrijf na deze aanpassingen vervolgens het gevraagde hoofdprogramma.

einde tentamen <<<