Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 1 - Grundlagen

(Stand 20.10.2014)

Aufgabe 1:

Gegeben sei folgende EBNF:

```
<Start>       ::=   <Digits> '.' [ <sVocal> ] <bVocal>
                  | '.' <Digits> { <bVocal> }
                  | <sVocal> { <bVocal> } <Digit>
<Digits>      ::= <Digit> { <Digit> }
<Digit>       ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<sVocal>      ::= 'a' | 'e' | 'i' | 'o' | 'u'
<bVocal>      ::= 'A' | 'E' | 'I' | 'O' | 'U'
```

Welche der folgenden Zeichenketten sind bezüglich der obigen EBNF syntaktisch korrekt, welche nicht (Begründen Sie Ihre Entscheidung!):

```
(1)  4711.E
(2)  .AE
(3)  i56
(4)  iAE
(5)  88.eeAA
(6)  eAEE7
(7)  815+aA
```

Aufgabe 2:

Gegeben sei folgende EBNF für Gleitkomma-Literale in Java:

```
<FloatPointLit>   ::=   <Digits> '.' [<Digits>] [<ExpPart>] [<FloatTypeSuffix>]
                      | '.' <Digits> [<ExpPart>] [<FloatTypeSuffix>]
                      | <Digits> <ExpPart> [<FloatTypeSuffix>]
                      | <Digits> [<ExpPart>] <FloatTypeSuffix>
<Digits>          ::= <Digit> {<Digits>}
<Digit>           ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<ExpPart>         ::= <ExpIndicator> <SignedInteger>
```

```
<ExpIndicator>    ::= ‘e’ | ‘E’
<SignedInteger>   ::= [<Sign>] <Digits>
<Sign>            ::= ‘+’ | ‘-’
<FloatTypeSuffix> ::= ‘f’ | ‘F’ | ‘d’ | ‘D’
```

Welche der folgenden Zeichenketten sind bezüglich der obigen EBNF syntaktisch korrekt, welche nicht (Begründen Sie Ihre Entscheidung!):

```
(1)  4711.08155e23D
(2)  0815.f
(3)  .ef
(4)  +4.5
(5)  3456.7<ExpPart>F
(6)  6.4E-23f
(7)  78.2f
(8)  78g.2f
(9)  99999999999.99999999e+99999999999999
(10) 6767.7676.7e56
```

Aufgabe 3:

Gegeben sei folgende EBNF:

```
<InterfaceDecl>   ::=    [ <InterfaceMod> ] 'interface' <Identifier>
                         [ 'extends' <Identifier> { ',' <Identifier> } ]
                         ( <InterfaceBody> | ';' )
<InterfaceMod>    ::=    ( 'public' | 'protected' | 'private' )
<Identifier>      ::=    <Letter> { ( <Letter> | <Digit> ) }
<InterfaceBody>   ::=    '{' { <InterfaceMemD> } '}'
<InterfaceMemD>   ::=    ( 'method' | 'constant' | 'class' )
<Letter>          ::=    ( 'a' | 'b' | '$' )
<Digit>           ::=    ( '0' | '1' )
```

Welche der folgenden Zeichenketten sind bezüglich der obigen EBNF syntaktisch korrekt, welche nicht (Begründen Sie Ihre Entscheidung!):

```
(1)  interface ab1a1a1;
(2)  public interface aaa extends bbb, ab$ { method method }
(3)  protected interface 0$ab extends aaa method constant
(4)  interface aaa, bbb extends aaa, bbb { aaa }
(5)  InterfaceMod interface extends Identifier ;
(6)  private public interface extends aaa  , { class extends bbb }
```

Aufgabe 4:

Gegeben sei die folgende Grammatik in EBNF (die Syntax entspricht übrigens in etwa der Syntax der funktionalen Programmiersprache *Scheme*):

```
<Ausdruck>          ::= <Atom> | <Liste>
<Atom>              ::= <Zahl> | <Zeichenkette>
<Liste>             ::= "(" <Ausdruck-Folge> ")"
<Ausdruck-Folge>    ::= <Ausdruck> <Ausdruck-Folge> | ∈
<Zahl>              ::= <Ziffer> { <Ziffer> }
<Ziffer>            ::= "0" | "1" | "2" | "3" | "4"
<Zeichenkette>      ::= <Zeichen> <Zeichenkette> | ∈
<Zeichen>           ::= "i" | "f" | "/" | "+" | "=" | "a"
```

Welche der folgenden Sätze/Programme sind korrekt, welche nicht! Begründen Sie Ihre Entscheidung!

```
(1)     (/ 22 (+ 41 3))
(2)     (if (= a 0)
            0
            (/ 1 a))
(3)     (+ 41 3) (+ 3 41)
(4)     (/ 1 2 3 (4 3 (* 2 1)
(5)     (* 2.3 4.6)
(6)     (+ Ziffer Zeichenkette)
(7)     (fifi (iffi iffi) ifif)
```

Aufgabe 5:

Die Sprache Mini-Pascal (aus dem Buch "Vom Problem zum Programm" von Herbert Klaeren, erschienen beim Teubner Verlag) ist in etwa durch die folgende EBNF definiert:

```
<program>   := "program" <ident> ";" <block> ".".
<block>     := [<vardecl>] "begin" <statement> {";" <statement>} "end".
<vardecl>   := "var" <ident> {"," <ident>} ":" "Integer" ";".
<statement> := <ident> ":=" <expr>
                | "begin" <statement> {";" <statement>} "end"
                | "if" <condition> "then" <statement>
                | "while" <condition> "do" <statement>
                | "read" <ident>
                | "write" <ident>.
<condition> := <expr> ("=" | "<>" | "<" | ">" | "<=" | ">=") <expr>.
<expr>      := <term> {("+" | "-") <term>}.
```

```
<term>      := <factor> {("*" | "/") <factor>}.
<factor>    := <ident> | <number> | "(" expr ")".
<ident>     := <letter> {<letter>}
<number>    := <digit> {<digit>}
<letter>    := "a" | "b" | "c"
<digit>     := "0" | "1"
```

Konstruieren Sie syntaktisch korrekte Mini-Pascal-Programme.