Obligatorisk uppgift: Numerisk kalkylator

Programmet skall läsa in aritmetiska uttryck samt beräkna och skriva ut deras värden.

Programmet ska läsa från standard input (`System.in`) och skriva till standard output (`System.out`).

Exempel: (programmet använder `>` som prompter):

```
> 1 - (5 - 2*2)/(1+1) - (-2 + 1)
1.5
> sin(3.14159265)
3.5897930298416118E-9
> exp(4*0.5 - 1)
2.7182818284590455
> exp(log(10))
10.000000000000002
> sin(2)*sin(2) + cos(2)*cos(2)
1.0
> 1 + 2 + 3 = x
6.0
> x
6.0
> x/2 + x
9.0
> (1=x) + sin(2=y)
1.9092974268256817
> x
1.0
> y
2.0
> 2*PI
6.283185307179586
```

Kommentarer:

1. Programmet skall hantera uttryck med konstanter, variabler, de aritmetiska operatorerna `+`, `-`, `*` och `/` samt de elementära funktionerna `sin`, `cos`, `exp` och `log`.
2. De vanliga prioritetsreglerna skall gälla och parenteser skall kunna användas för att ändra beräkningsordning på vanligt sätt.
3. Alla beräkningar görs i *flyttalsaritmetik* (`double`).
4. Variabeltilldelning görs *från vänster till höger*.

   Exempel: Uttrycket

   ```
   1+2*3 = y
   ```

   skall tilldela `y` värdet `7`.
5. Variabeltilldelningar skall kunna göras i deluttryck.

   Exempel:

   ```
   (2=x) + (3=y=z) = a
   ```

   skall ge värden till `x`, `y`, `z` och `a`.

6. Den fördefinierade variabeln `ans` skall innehålla värdet av det senast beräknade fullständiga uttrycket. Denna variabel kan användas i nästa uttryck. Exempel:

```
> 1+1
2.0
> ans
2.0
> exp(2)
7.38905609893065
> ans
7.38905609893065
> ans + 3
10.38905609893065
> 3 + ans
13.38905609893065
```

7. Programmet skall upptäcka och diagnostisera fel. Exempel:

```
> 1++2
*** Syntax error. Expected number, word or '('
*** The error occured at token '+' just after token '+'
> 1+-2
-1.0
> 1--2
3.0
> 1**2
*** Syntax error. Expected number, word or '('
*** The error occured at token '*' just after token '*'
> 1/0
*** Evaluation error. Division by zero
> 1+2*y-4
1.0
> 1+2*k-4
*** Evaluation error. Undefined variable: k
> 1+2=3+4**x - 1/0
*** Syntax error. Expected variable after '='
*** The error occured at token '3.0' just after token '='
> 1+2*(3-1 a
*** Syntax error: Expected ')'
*** The error occured at token 'a' just after token '1.0'
> 1+2+3+
*** Syntax error: Expected number, word or '('
*** The error occured at token '*EOL*' just after token '+'
>
```

   Syntaxfel ska ge upphov till följande felutskrift:

```
"*** Syntax error. <Beskrivning av felet>
<Beskrivning av var felet uppkom>"
```

   (Observera radbrytningen.) Ett evalueringsfel ska ge upphov till följande felutskrift:

```
*** Evaluation error. <Beskrivning av felet>
```

   Observera att uttrycket skall "kastas bort" när ett fel har upptäckts så att inte man får fler diagnoser på samma uttryck. Variabeln `ans` skall *inte* ändras om ett uttryck är felaktigt.

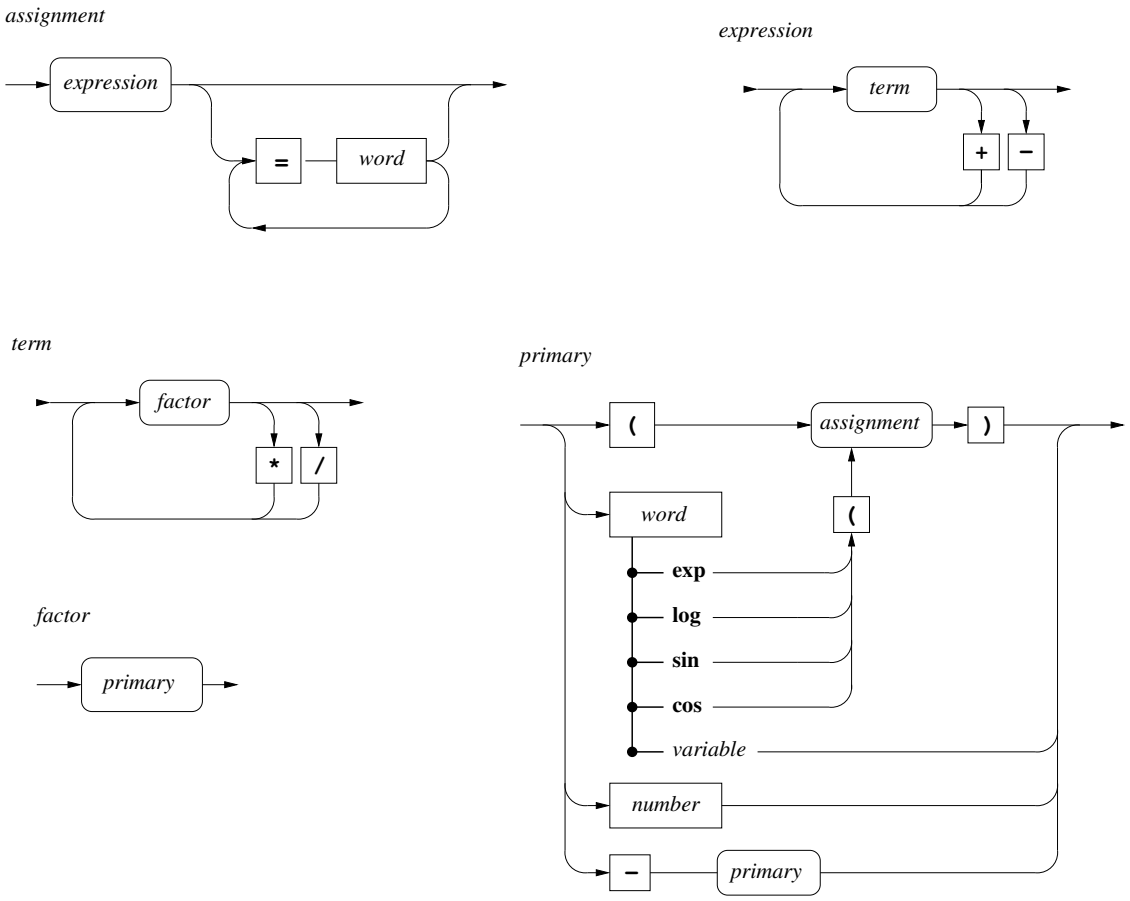
8. Kommandot `vars` visar alla lagrade variabler med värden och kommandot `quit` avslutar körningen.

   Exempel:

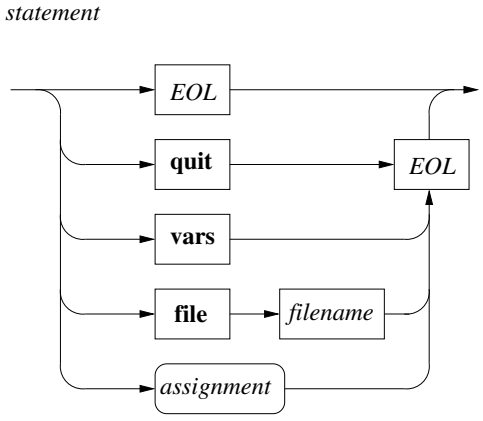
```
> vars
 ans     : 13.38905609893065
 E       : 2.718281828459045
 PI      : 3.141592653589793
 x       : 1.0
 y       : 2.0
> quit
Bye!
```

Syntaxdiagram

Syntaxen för uttrycken definieras av följande diagram:



Det syntaktiska elementet *factor* finns med som en ”platshållare” för uppgift 5.



(*EOL* står för ”End Of Line” dvs radslut).

Programdesign

Klasser

Programmet skall ha följande klasser (med angivna namn):

- `Parser`: Sköter tolkandet och beräkningarna. Klassen skall ha en metod för varje syntaktiskt element (dvs *assignment*, *expression*, *term*, *factor* och *primary*) Dessa metoder skall fungera som syntaxdiagrammen beskriver. (Metoden *statement* med dess hjälpmetod *line* finns i klassen `Calculator`).
- `Stokenizer`: Sköter uppdelningen av indatasträngen i symboler (ord, tal mm). Denna klass som tillhandahålls av oss *skall* användas.
- `Calculator`: Denna klass är given av oss. Den innehåller bland annat programmets `main`-metod.
- `SyntaxException`: Beskriver syntaxfel
- `EvaluationException`: Beskriver beräkningsfel

Variabelvärden

För att hålla reda på variablers värden skall en `TreeMap` användas. Eftersom värden variablerna identifieras av sitt namn och värden är symboliska blir deklarationen:

```
Map<String,Double> variables = new TreeMap<String,Double>();
```

Även den fördefinierade variabeln `ans` ligger i listan!

Felhantering

Om användaren skriver ett uttryck som inte stämmer med syntaxen enligt syntaxdiagrammen (t ex `a ++b` eller `sin + 4` eller `2*3)+ 8`) är det ett *syntaxfel*. En annan typ av fel är att att användaren ger ett uttryck som inte kan beräknas. Exempelvis så är uttrycken `1/x` och `sin(a)` skrivna med korrekt syntax men går inte att beräkna om `x` har värdet 0 eller `a` är odefinierad.

Dessa två fel skall hanteras med undantagen `SyntaxException` och `EvaluationException`.

Klassen `Calculator`

Klassen `Calculator` som definierar den översta nivån i kalkylatorn är given. Den innehåller `main`- och `statement`-metoderna samt några hjälpmetoder. Den *enda* ändring du skall göra i den är att lägga till hanteringen av undantagen `SyntaxException` och `EvaluationException` på indikerade platser i `statemet`:

```
public void statement() {
    try {
       line();
    } catch (SyntaxException syntaxError) {
      ...
    }
    } catch (EvaluationException evaluationException) {
      ...
    }
}
```

Krav för att bli godkänd på uppgiften

1. Programmet *skall* fungera i enlighet med specifikationen och enligt det körexempel som finns i början på detta dokument. Även felhateringen skall fungera enligt exemplen.
2. Koden *skall* följa kodkonventionerna och även på andra sätta vara ”läsvänlig”.
3. Den givna tokenizern (`Stokenizer`) och `Calculator`-klassen skall användas.
4. ”Recursive descent” skall användas som parse-metod. Det skall finnas en metod för *varje* syntaktisk enhet (varje enskilt syntaxdiagram)
5. Du skall kunna förklara hur programmet inklusive `Calculator`-klassen (förutom metoden `fileInput`) fungerar och beskriva hur man kan utöka det till exempel med nya funktioner och operatorer.