CSci 340, Spring 2003, Project 3

This project is due *Friday, March 21* at 11:20am. There is no electronic handin; instructions for composing your report are at the end of this assignment.

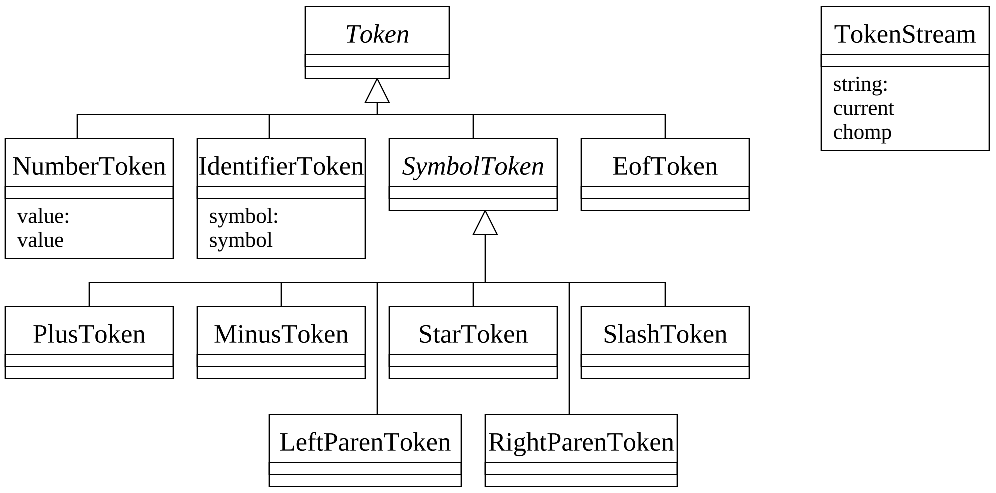
In this assignment you are to extend the results of Project 2 to incorporate parsing a string into an expression. In particular, you should develop a class method called “parse:” for the Expr class (in the “ExprEval–Parser” category). This method should take a String instance and return an Expr instance representing the expression represented by the string. Using this class, you might for example evaluate the following Squeak code, which would print 42 into the Transcript window.

```
c ← (Context new set: #i value: 8) set: #j value: 31.
t ← Expr parse: ’i * (5 + 2 * i) / 4’
Transcript show: (t eval: c) printString.
```

The grammar supported by your program should be the following.

$$\begin{array}{rcl}
\textit{\textbf{expr}} & \rightarrow & \textit{\textbf{term}} \ \{ \ (\textit{plus} \mid \textit{minus}) \ \textit{\textbf{term}} \ \} \\
\textit{\textbf{term}} & \rightarrow & \textit{\textbf{factor}} \ \{ \ (\textit{star} \mid \textit{slash}) \ \textit{\textbf{factor}} \ \} \\
\textit{\textbf{factor}} & \rightarrow & \textit{number} \mid \textit{identifier} \mid \textit{left\_parenthesis} \ \textit{\textbf{expr}} \ \textit{right\_parenthesis}
\end{array}$$

When you run *inisqueak*, it will place a Squeak image incorporating a variety of new classes in the “ExprEval–Scanner” category.[1] These classes represent the variety of tokens that can occur in an expression, organized according to the following hierarchy.



Token objects represent various tokens that may occur in the token stream (except for EofToken, which represents the end of the stream). The TokenStream object is for iterating through the tokens of a stream; you can set it up to go through a particular string using the “string:” method. The “current” method returns the current token, and the “chomp” method iterates forward to the next token of the stream.

To create your report, print the Expr class. The easiest way to accomplish this is to middle-click on its name in Squeak’s System Browser and choose “fileOut.” This will place a file named `Expr.st` into the Unix directory containing the Squeak image. The following command will edit this files into a text file named `report` containing your code.

```
unix% squeak_cat Expr.st
```

You should review it before printing it out to turn in.

[1] As before, I advise you first to remove the `squeak.image` and `squeak.changes` files that you’ve used in the past, to reduce the chance of exceeding your quota.

Ada code

Basically, your code will be a translation of the following Ada code into Smalltalk.

```
procedure Parse_Expr(Stream : Token_Stream) return Expr is
    Ret : Expr;
begin
    Ret := Parse_Term(Stream);
    while true loop
        if Current(Stream) = Plus_Token then
            Chomp(Stream);           -- Chomps Plus_Token from Stream
            Ret := New_Add_Expr(Ret, Parse_Term(Stream));
        elsif Current(Stream) = Minus_Token then
            Chomp(Stream);           -- Chomps Minus_Token from Stream
            Ret := New_Sub_Expr(Ret, Parse_Term(Stream));
        else
            return Ret;
        end if;
    end loop;
end Parse_Expr;

procedure Parse_Term(Stream : Token_Stream) return Expr is
    Ret : Expr;
begin
    Ret := Parse_Factor(Stream);
    while true loop
        if Current(Stream) = Star_Token then
            Chomp(Stream);           -- Chomps Star_Token from Stream
            Ret := New_Mult_Expr(Ret, Parse_Factor(Stream));
        elsif Current(Stream) = Slash_Token then
            Chomp(Stream);           -- Chomps Slash_Token from Stream
            Ret := New_Div_Expr(Ret, Parse_Factor(Stream));
        else
            return Ret;
        end if;
    end loop;
end Parse_Term;

procedure Parse_Factor(Stream : Token_Stream) return Expr is
    Ret : Expr;
begin
    if Current(Stream) = Left_Paren_Token then
        Chomp(Stream);           -- Chomps Left_Paren_Token from Stream
        Ret := Parse_Expr(Stream);
        Chomp(Stream);           -- Chomps Right_Paren_Token from Stream
        return Ret;
    elsif Current(Stream) = Number_Token then
        Ret := New_Const_Expr(Current_Data(Stream));
        Chomp(Stream);           -- Chomps Number_Token from Stream
        return Ret;
    elsif Current(Stream) = Identifier_Token then
        Ret := New_Const_Ident(Current_Data(Stream));
        Chomp(Stream);           -- Chomps Identifier_Token from Stream
        return Ret;
    end if;
end Parse_Factor;
```