The purpose of this assignment is to get you back into the Java programming groove, and to give you a practical introduction to some of the theoretical topics you will see over the next several weeks. This is not a trivial assignment! Start early, and come by my office to ask questions. For this project you will create a SimpleLogic parser. That is, you will write a program that takes as input a SimpleLogic program, and determines if that program contains any syntax errors.

1 SimpleLogic Programming Language

1.1 Tokens

The legal tokens in SimpleLogic are:

<table>
<thead>
<tr>
<th>Token name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Brace</td>
<td>{</td>
</tr>
<tr>
<td>Right Brace</td>
<td>}</td>
</tr>
<tr>
<td>Left Parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>Right Parenthesis</td>
<td>)</td>
</tr>
<tr>
<td>Equals</td>
<td>=</td>
</tr>
<tr>
<td>Not</td>
<td>!</td>
</tr>
<tr>
<td>Semicolon</td>
<td>;</td>
</tr>
<tr>
<td>Keyword if</td>
<td>if</td>
</tr>
<tr>
<td>Keyword else</td>
<td>else</td>
</tr>
<tr>
<td>Keyword true</td>
<td>true</td>
</tr>
<tr>
<td>Keyword false</td>
<td>false</td>
</tr>
<tr>
<td>Identifier</td>
<td>Sequence of letters that is not a keyword Examples: x, foo, bigshot</td>
</tr>
</tbody>
</table>

Note that all letters are lower-case in a valid SimpleLogic program.

1.2 SimpleLogic Statements

A SimpleLogic statement is either an assignment statement, an if statement, or a block statement.

1.2.1 Assignment statement

An assignment statement is of the form:
<Identifier> = <expression>;
where <Identifier> is just an identifier token as described above, and an <expression> is either an identifier token, the keyword true, the keyword false, !<expression>, or ( <expression> ). Thus all of the following are valid assignment statements:

x = y;
foo = (true);
w = z;
elsex = w;
z = ((z));
w = !((!!z));

And none of the following are valid assignment statements:

x = if;
else = y;
true = false;
true = true;
x = ;
y = ((false));
z = x!;

1.2.2 if statement

An if statement is of the form

if (<expression>) <statement> else <statement>

where <expression> is an expression as defined above (identifier, boolean literal, negation, parentheses allowed), and a <statement> is any SimpleLogic statement. Thus the following are all legal SimpleLogic statements:

if (z) x = false; else x = true;
if (true) w = false; else w = true;
if (!x) y = !y else y = (!(!z));

Note that whitespace (tabs, spaces, end of line characters) are allowed between SimpleLogic tokens, so the following is a valid SimpleLogic statement:

if (x)
  if (!y)
    x = y;
  else
    w = !z;
else
  q = r;

Note that in simpleLogic, the else part of an if statement is required!. This actually makes your job much easier, as you will see when we discuss the dangling else problem.
1.2.3 Block Statement

A block Statement consists of a list of zero or more statements within braces. So, the following is a legal block statement:

```plaintext
{x = true;
 y = z;
 z = w;
}
```

The following is also a legal block statement:

```plaintext
{x = true;
 if (z)
 { w = y;
   y = true;
 }
 else { }
}
```

Finally, the following is also a legal block statement:

```plaintext
{}
```

A program in the SimpleLogic programming language consists of a single statement (which could be a block statement, and could have arbitrarily many statements nested inside of it).

1.3 Examples

1.3.1 Valid SimpleLogic Programs

```plaintext
{x = true;
 { y = z;
   w = carpark;
 }
 if (foolsgold)
   train = wreck;
 else
 { cat = dog;
 }
}
```

and
1.3.2 Invalid SimpleLogic Programs

Missing a }:

```java
{ x = true;
  { y = z;
    w = carpark;
  }
  Missing two ;'s:

{ x = true;
y = z;
  if (y) x = 3 else x = 4 }

   Numbers are not valid in indentifiers:

{ x = ident1fi3r; }

   If test requires parentheses:

if x y = 3; else y = 4;
```

2 Assignment

Write a java program MiniParser.java that takes as a command line parameter the name of the input file. Your program should either write “valid program” or “invalid program” to standard out, depending upon whether the input file contains a syntactically correct SimpleLogic program. Your program does not need to identify what the errors are, or how many errors there are – it just needs to return if the entire program is valid or not. Your program should never throw an exception! Your program should also only print out “valid program” or “invalid program” and nothing else! (An end-of-line (“\n”) after valid / invalid program is OK!)

2.1 Assignment Hints

- You should probably first write a function / set of functions that get the next token before moving on to the actual parsing.

- I used a 1-token buffer to do the parsing. Come see me if you are not sure what I am talking about here!
• Recursion is your friend!

• Make sure your code works on all the provided examples!

This is not a trivial assignment. Start Early! Today would be good. Also, come by my office with questions. I have not spelled out exactly how to do this assignment, so I am expecting you to come by my office for help!