```
 1 /**
 2  * Symbol represents what will be placed on the stack.
 3  */
 4 class Symbol
 5 {
 6     char  token;
 7     int   theLine;
 8
 9     Symbol( char tok, int line )
10     {
11         token = tok;
12         theLine = line;
13     }
14 }
```

Figure 11.3 Object that is placed on the stack

11.1.2 Implementation

Figure 11.2 (page 305) shows the `JavaAnalyzer` class that does all the work. In addition to the constructor, the only other publicly visible routine is `checkBalance`, shown at line 26. Everything else is a supporting routine or a class data field. We begin by describing the data fields.

`in` is a reference to a `PushbackReader` object and is initialized at construction. A `PushbackReader` is like a `BufferedReader`, except that it also provides an `unread` method. The current character being scanned is stored in `ch`, and the current line number is stored in `currentLine`. The balanced symbol algorithm requires that we place opening symbols on a stack. In order to print diagnostics, we store a line number with each symbol, as shown in the `Symbol` class in Figure 11.3. The stack itself is `pendingTokens`, declared at line 32. Finally, an integer that counts the number of errors is declared at line 33.

The constructor, shown at lines 17 to 24, initializes the error count to 0 and the current line number to 1 and sets the `PushbackReader` reference. The remaining data fields, namely `ch` and `pendingTokens`, are initialized; in the case of `pendingTokens` a zero-parameter constructor is used to create an empty stack.

Lexical analysis is used to ignore comments and recognize symbols.

We can now examine some of the supporting routines. Many of them are concerned with keeping track of the current line and attempting to differentiate symbols that represent opening and closing tokens from those that are inside comments, character constants, and string constants. This general process is called *lexical analysis*. Figure 11.4 shows a pair of routines, `nextChar` and `putBackChar`. `nextChar` reads the next character, assigns it to `ch`, and updates `currentLine` if a newline is seen. It returns `false` if the end of the file is reached. The complementary procedure `putBackChar` puts the current