In `skipComment`, at any point, it has matched either 0, 1, or 2 characters of the `*/` terminator, corresponding to states 0, 1, and 2. If it matches two characters, it can return. Thus, inside the loop, it can be in only state 0 or 1, since if it is in state 1 and sees a `/`, it returns immediately. Thus the state can be represented by a Boolean variable that is true if the state machine is in state 1. If it does not return, then it goes back either to state 1 if it sees a `*` or to state 0 if it does not. This is stated succinctly at line 23.

At any point, it is in some state, and each input character takes it to a new state. Eventually, the state machine reaches a state in which a symbol has been recognized.

If we never find the comment-ending token, then eventually `nextChar` returns `false` and the `while` loop terminates, resulting in an error message. `skipQuote`, shown in Figure 11.6, is similar. Here, the parameter is the opening quote character, which is either `"` or `'`. In either case, we need to see that character as the closing quote. However, we must be prepared to handle the \ character; otherwise, our program will report errors when it is run on its own source. Thus we repeatedly digest characters. If the current character is a closing quote, we are done. If it is a newline, we have an unterminated character or string constant. And if it is a backslash, we digest an extra character without examining it.

```
 1      /**
 2       * Precondition: We are about to process a quote; have
 3       *                   already seen beginning quote.
 4       * Postcondition: Stream will be set immediately after
 5       *                   matching quote
 6       */
 7      private void skipQuote( char quoteType )
 8      {
 9          while( nextChar( ) )
10          {
11              if( ch == quoteType )
12                  return;
13              if( ch == '\n' )
14              {
15                  errors++;
16                  System.out.println( "Missing quote at line " +
17                                      currentLine );
18                  return;
19              }
20              else if( ch == '\\' )
21                  nextChar( );
22          }
23      }
```

Figure 11.6 `skipQuote` routine to move past an already-started character or string constant