A Mistake to Avoid When Doing TinyJ Assignment 1

A common mistake in writing recursive descent parsing code is to write

`getCurrentToken() ==` *X*

or `accept(`*X*`)` [which performs a `getCurrentToken() ==` *X* test]

using a `Symbols` constant *X* that represents a ***non***terminal. This is wrong, as `getCurrentToken()` returns a `Symbols` constant that represents a ***token***. Here are two examples of this kind of mistake.

1. When writing the method `argumentList()`, which should be based on the EBNF rule

   ```
   <argumentList>   ::=   '('[<expr3>{,<expr3>}]')'
   ```

   it would be wrong to write:

   ```
   accept(LPAREN);
   if (getCurrentToken() == NTexpr3) /* INCORRECT! */ {
      expr3();
       ...      // a while loop that deals with  {,<expr3>}
   }
   accept(RPAREN);
   ```

   Here it would be correct to write code of the following form:

   ```
   accept(LPAREN);
   if (getCurrentToken() != RPAREN) /* CORRECT */ {
      expr3();
       ...      // a while loop that deals with  {,<expr3>}
   }
   accept(RPAREN);
   ```

2. When writing the method `expr1()`, one case you need to deal with relates to the following part of the EBNF rule that defines `<expr1>`:

   ```
   IDENTIFIER ( . nextInt '(' ')' | [<argumentList>]{'[' <expr3> ']'} )
   ```

   Here it would be wrong to write something like:

   ```
   case IDENT:
     nextToken();
     if (getCurrentToken() != DOT) {
       if (getCurrentToken() == NTargumentList /* INCORRECT! */ ) argumentList();
       ... // a while loop that deals with {'[' <expr3> ']'}
     }
     else {
       ... // code to deal with   . nextInt '(' ')'
     }
     break;
   ```

   Instead, you can write something like:

   ```
   case IDENT:
     nextToken();
     if (getCurrentToken() != DOT) {
       if (getCurrentToken() == LPAREN /* CORRECT */ ) argumentList();
       ... // a while loop that deals with {'[' <expr3> ']'}
     }
     else {
       ... // code to deal with   . nextInt '(' ')'
     }
     break;
   ```

   The use of `LPAREN` in the above code is correct because the first token of any instance of `<argumentList>` must be a left parenthesis, as we see from the EBNF rule

   ```
   <argumentList>   ::=   '('[<expr3>{,<expr3>}]')'
   ```