ParseException.java

```java
/* Generated By:JavaCC: Do not edit this line. ParseException.java Version 3.0 */
package jminusminus;

/**
 * This exception is thrown when parse errors are encountered.
 * You can explicitly create objects of this exception type by
 * calling the method generateParseException in the generated
 * parser.
 *
 * You can modify this class to customize your error reporting
 * mechanisms so long as you retain the public fields.
 */
public class ParseException extends Exception {

  /**
   * This constructor is used by the method "generateParseException"
   * in the generated parser.  Calling this constructor generates
   * a new object of this type with the fields "currentToken",
   * "expectedTokenSequences", and "tokenImage" set.  The boolean
   * flag "specialConstructor" is also set to true to indicate that
   * this constructor was used to create this object.
   * This constructor calls its super class with the empty string
   * to force the "toString" method of parent class "Throwable" to
   * print the error message in the form:
   *     ParseException: <result of getMessage>
   */
  public ParseException(Token currentTokenVal,
                        int[][] expectedTokenSequencesVal,
                        String[] tokenImageVal
                       )
  {
    super("");
    specialConstructor = true;
    currentToken = currentTokenVal;
    expectedTokenSequences = expectedTokenSequencesVal;
    tokenImage = tokenImageVal;
  }

  /**
   * The following constructors are for use by you for whatever
   * purpose you can think of.  Constructing the exception in this
   * manner makes the exception behave in the normal way - i.e., as
   * documented in the class "Throwable".  The fields "errorToken",
   * "expectedTokenSequences", and "tokenImage" do not contain
   * relevant information.  The JavaCC generated code does not use
   * these constructors.
   */

  public ParseException() {
    super();
    specialConstructor = false;
  }

  public ParseException(String message) {
    super(message);
    specialConstructor = false;
  }

  /**
   * This variable determines which constructor was used to create
   * this object and thereby affects the semantics of the
   * "getMessage" method (see below).
   */
  protected boolean specialConstructor;

  /**
```

```java
   * This is the last token that has been consumed successfully.  If
   * this object has been created due to a parse error, the token
   * followng this token will (therefore) be the first error token.
   */
  public Token currentToken;

  /**
   * Each entry in this array is an array of integers.  Each array
   * of integers represents a sequence of tokens (by their ordinal
   * values) that is expected at this point of the parse.
   */
  public int[][] expectedTokenSequences;

  /**
   * This is a reference to the "tokenImage" array of the generated
   * parser within which the parse error occurred.  This array is
   * defined in the generated ...Constants interface.
   */
  public String[] tokenImage;

  /**
   * This method has the standard behavior when this object has been
   * created using the standard constructors.  Otherwise, it uses
   * "currentToken" and "expectedTokenSequences" to generate a parse
   * error message and returns it.  If this object has been created
   * due to a parse error, and you do not catch it (it gets thrown
   * from the parser), then this method is called during the printing
   * of the final stack trace, and hence the correct error message
   * gets displayed.
   */
  public String getMessage() {
    if (!specialConstructor) {
      return super.getMessage();
    }
    StringBuffer expected = new StringBuffer();
    int maxSize = 0;
    for (int i = 0; i < expectedTokenSequences.length; i++) {
      if (maxSize < expectedTokenSequences[i].length) {
        maxSize = expectedTokenSequences[i].length;
      }
      for (int j = 0; j < expectedTokenSequences[i].length; j++) {
        expected.append(tokenImage[expectedTokenSequences[i][j]]).append(" ");
      }
      if (expectedTokenSequences[i][expectedTokenSequences[i].length - 1] != 0) {
        expected.append("...");
      }
      expected.append(eol).append("    ");
    }
    String retval = "Encountered \"";
    Token tok = currentToken.next;
    for (int i = 0; i < maxSize; i++) {
      if (i != 0) retval += " ";
      if (tok.kind == 0) {
        retval += tokenImage[0];
        break;
      }
      retval += add_escapes(tok.image);
      tok = tok.next;
    }
    retval += "\" at line " + currentToken.next.beginLine + ", column " +
currentToken.next.beginColumn;
    retval += "." + eol;
    if (expectedTokenSequences.length == 1) {
      retval += "Was expecting:" + eol + "    ";
    } else {
      retval += "Was expecting one of:" + eol + "    ";
    }
    retval += expected.toString();
    return retval;
  }

  /**
   * The end of line string for this machine.
   */
  protected String eol = System.getProperty("line.separator", "\n");

  /**
   * Used to convert raw characters to their escaped version
   * when these raw version cannot be used as part of an ASCII
   * string literal.
   */
  protected String add_escapes(String str) {
      StringBuffer retval = new StringBuffer();
      char ch;
      for (int i = 0; i < str.length(); i++) {
        switch (str.charAt(i))
        {
           case 0 :
              continue;
           case '\b':
              retval.append("\\b");
              continue;
           case '\t':
              retval.append("\\t");
              continue;
           case '\n':
              retval.append("\\n");
              continue;
           case '\f':
              retval.append("\\f");
              continue;
           case '\r':
              retval.append("\\r");
              continue;
           case '\"':
              retval.append("\\\"");
              continue;
           case '\'':
              retval.append("\\\'");
              continue;
           case '\\':
              retval.append("\\\\");
              continue;
           default:
              if ((ch = str.charAt(i)) < 0x20 || ch > 0x7e) {
                 String s = "0000" + Integer.toString(ch, 16);
                 retval.append("\\u" + s.substring(s.length() - 4, s.length()));
              } else {
                 retval.append(ch);
              }
              continue;
        }
      }
      return retval.toString();
   }

}
```