The  MLP.zip zip file, which has

* the Math tagger JAR file (MLP.jar). It has all the software executables you need for tagging math.
* the README file related to the jar file
* the ReferenceData folder (which has the Lexicons), needed by the software to tag properly
* the Javadoc (software documentation) folder
* A "MLP.java" file that you can customize and start from
* PomTaggerParser.html (which is the grammar of the POM parser)
* This software usage guide

Here are some remarks to help you get started:

* Once you unzip MLP.zip, you can go to the javadoc, and click on the index.html to bring up the main page of the documentation and start learning about the various classes.
* To test (and use) the software, you need to:
  + create a "main" file that has the import statement "import mlp.\*;". You can use the attached MLP.java as your main file, but you need to do one thing to it first:
    - you should modify the first line of the "main" method, to set the variable "refDataDir" to the path where you the ReferenceData ended up after you unziped MLP.zip.

Then, inside the "try" block of the "main method", you can assign to the "String eq" whatever String that represents LaTeX math, and run the code.

* + make the MLP.jar file available to your application.
* There are several classes (in MLP.jar and documented the javadoc), but the key ones you need to know are:
  + MathTerm: this class models any single math term, capturing tags and named features that the tagger associates with the math term. The interface of this class familiarizes you with the various methods for querying MathTerm objects. For example, it allows you to determine the actual math term being modeled, what primary tag and secondary tags it has, what named features it has and and what their corresponding values are, and what alternative Feature-Sets it has. It also allows you to modify the contents of the object. For example, as you start to build new code on top of my software, you can add more (unnamed tags) and more named features and Feature-Sets to the MathTerm, remove existing features/tags, and decide which of the features/tags to make use of.
  + PomTaggedExpression: this class models a tagged math expression, and the parser returns a PomTaggedExpression. It has a large API. For brevity, I recommend you read the documentation (click on "PomTaggedExpression" in the left pane of the index.html page of the documentation).
  + PomParser: this is the actual parser. As you see in the attached sample MLP.java, you need first to create a PomParser object:

PomParser parser = new PomParser(refDataDir);

Also, if you want to use macros, you can create an array of macros (each macro is a String that has the LaTeX macro definition; that String can start with “\newcommand” but that is optional – if missing, it will be assumed re-inserted by default). Then you make those macros available to the parser by calling the addMacros(…) method of the PomParser. See the attached MLP.java for an example.

The main method of the PomParser is “parse(String equation)”. Use it to parse and tag math expressions. As mentioned above, the parse method returns a PomTaggedExpression object, which you can then query (using the PomTaggedExpression class API) to find out the math terms (in the input math expression) and their tags and features.

* + FeatureSet: it models a Feature Set, which is a set of related features. A Feature Set can optionally have a name, and has zero or more (feature-name, feature-value) pairs. For more details, refer to the documentation (click on " FeatureSet" in the left pane of the index.html page of the documentation).
  + Lexicon: this models a lexicon, where each entry is a (key,value) pair. A key is a string which can be a Latex command/macro, an alphabetic/alphanumeric symbol (including single Latin letters), or a non-alphanumeric symbol/string. The value part of an entry is a list of alternative FeatureSet objects.

The remaining classes play supportive roles.

* A note about lexicons:
  + When a user creates an instance of the PomParser, e.g.,

PomParser parser = new global-lexicon.txt (refDataDir);

The global-lexicon.txt is loaded to the internal lexicon of the PomParser, as the default lexicon. The parser then draws from the internal lexicon alternative feature sets to associate with various math terms.

* + The global-lexicon.txt can have too many alternative feature sets that may be irrelevant to a user.
  + Therefore, the PomParser class has methods for clearing its internal lexicon, and for adding smaller more specialized lexicons (either from the ones currently provided in the Lexicons subdirectory of the ReferenceData directory, or from user-created lexicons). The methods for doing so are:
    - clearLexicon( ); // for clearing the internal lexicon
    - addLexicons(String… lexicons); // for adding lexicon files
    - addLexicons(Lexicon… lexicons); // for addition Lexicon objects
  + The user can define their own lexicons, either dynamically using the Lexicon class and/or the LexiconFactory class, or statically by creating physical lexicon “.txt” files inside the Lexicons subdirectory.