CSC3010 Information Retrieval
Tutorial 05

Lucene 02

Peter CY YAU

WORLD CHANGING GLASGOW

THE TIMES THE SUNDAY TIMES GOOD UNIVERSITY GUIDE 2022 SCOTTISH UNIVERSITY OF THE YEAR

Agenda

- Parsing
- Tokenization
- Core Analysis

Parsing

- Applications that build their search capabilities upon Lucene may support documents in various formats – HTML, XML, PDF, Word – just to name a few. Lucene does not care about the Parsing of these and other document formats, and it is the responsibility of the application using Lucene to use an appropriate Parser to convert the original format into plain text before passing that plain text to Lucene.

Tokenization

- Plain text passed to Lucene for indexing goes through a process generally called tokenization. Tokenization is the process of breaking input text into small indexing elements – tokens. The way input text is broken into tokens heavily influences how people will then be able to search for that text. For instance, sentences beginnings and endings can be identified to provide for more accurate phrase and proximity searches (though sentence identification is not provided by Lucene).
- In some cases simply breaking the input text into tokens is not enough – a deeper *Analysis* may be needed. Lucene includes both pre- and post-tokenization analysis facilities.
- Pre-tokenization analysis can include (but is not limited to) stripping HTML markup, and transforming or removing text matching arbitrary patterns or sets of fixed strings.

Tokenization (Cont)

- Stemming – Replacing words with their stems. For instance with English stemming "bikes" is replaced with "bike"; now query "bike" can find both documents containing "bike" and those containing "bikes".
- Stop Words Filtering – Common words like "the", "and" and "a" rarely add any value to a search. Removing them shrinks the index size and increases performance. It may also reduce some "noise" and actually improve search quality.
- Text Normalization – Stripping accents and other character markings can make for better searching.
- Synonym Expansion – Adding in synonyms at the same token position as the current word can mean better matching when users search with words in the synonym set.

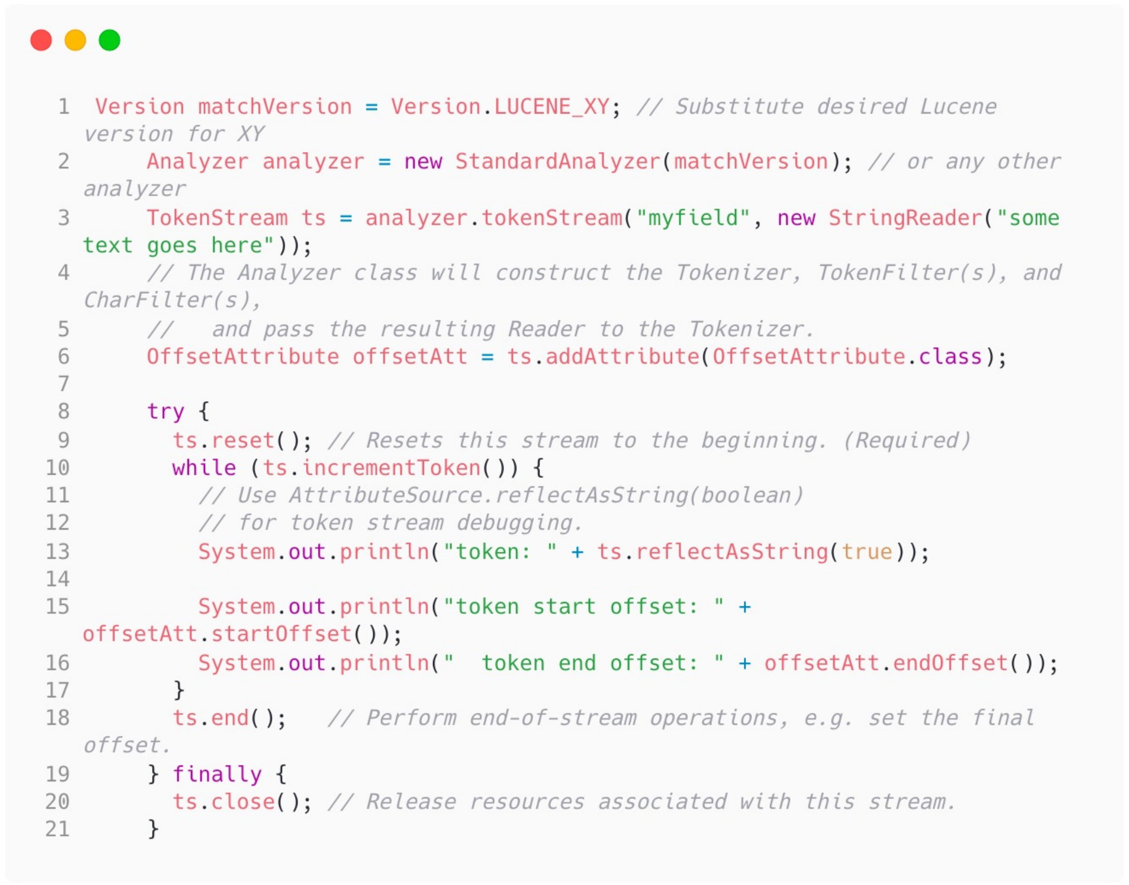
Core Analysis

- The analysis package provides the mechanism to convert Strings and Readers into tokens that can be indexed by Lucene. There are four main classes in the package from which all analysis processes are derived.

Core Analysis (Cont)

- Analyzer – An Analyzer is responsible for supplying a TokenStream which can be consumed by the indexing and searching processes. See below for more information on implementing your own Analyzer. Most of the time, you can use an anonymous subclass of Analyzer.
- CharFilter – CharFilter extends Reader to transform the text before it is tokenized, while providing corrected character offsets to account for these modifications. This capability allows highlighting to function over the original text when indexed tokens are created from CharFilter-modified text with offsets that are not the same as those in the original text. Tokenizer.setReader(java.io.Reader) accept CharFilters. CharFilters may be chained to perform multiple pre-tokenization modifications.
- Tokenizer – A Tokenizer is a TokenStream and is responsible for breaking up incoming text into tokens. In many cases, an Analyzer will use a Tokenizer as the first step in the analysis process. However, to modify text prior to tokenization, use a CharFilter subclass (see above).
- TokenFilter – A TokenFilter is a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer. Common modifications performed by a TokenFilter are: deletion, stemming, synonym injection, and case folding. Not all Analyzers require TokenFilters.

Invoking the Analyzer

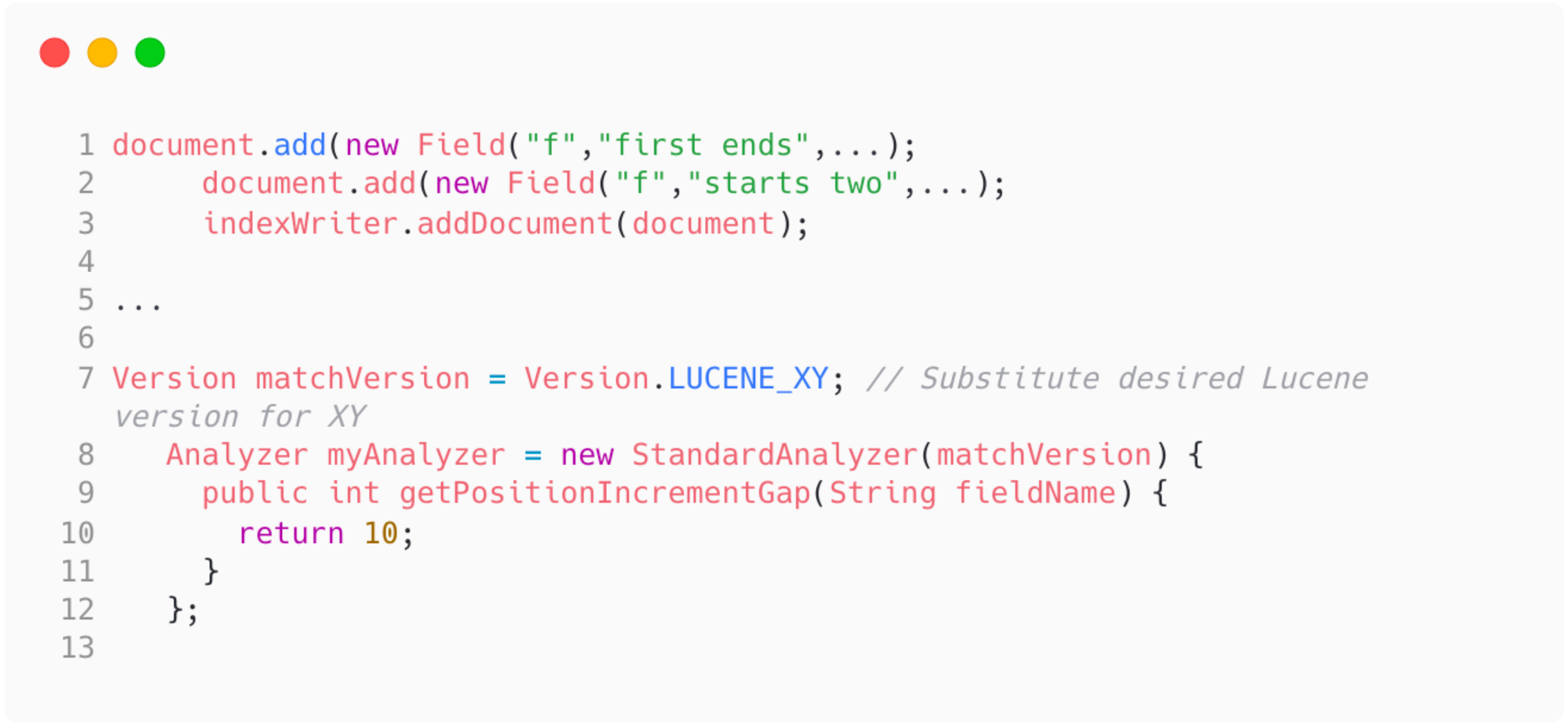


```java
Version matchVersion = Version.LUCENE_XY; // Substitute desired Lucene version for XY
    Analyzer analyzer = new StandardAnalyzer(matchVersion); // or any other analyzer
    TokenStream ts = analyzer.tokenStream("myfield", new StringReader("some text goes here"));
    // The Analyzer class will construct the Tokenizer, TokenFilter(s), and CharFilter(s),
    //   and pass the resulting Reader to the Tokenizer.
    OffsetAttribute offsetAtt = ts.addAttribute(OffsetAttribute.class);

    try {
      ts.reset(); // Resets this stream to the beginning. (Required)
      while (ts.incrementToken()) {
        // Use AttributeSource.reflectAsString(boolean)
        // for token stream debugging.
        System.out.println("token: " + ts.reflectAsString(true));

        System.out.println("token start offset: " + offsetAtt.startOffset());
        System.out.println("  token end offset: " + offsetAtt.endOffset());
      }
      ts.end();   // Perform end-of-stream operations, e.g. set the final offset.
    } finally {
      ts.close(); // Release resources associated with this stream.
    }
```

Implementing your own Analyzer and Analysis Components

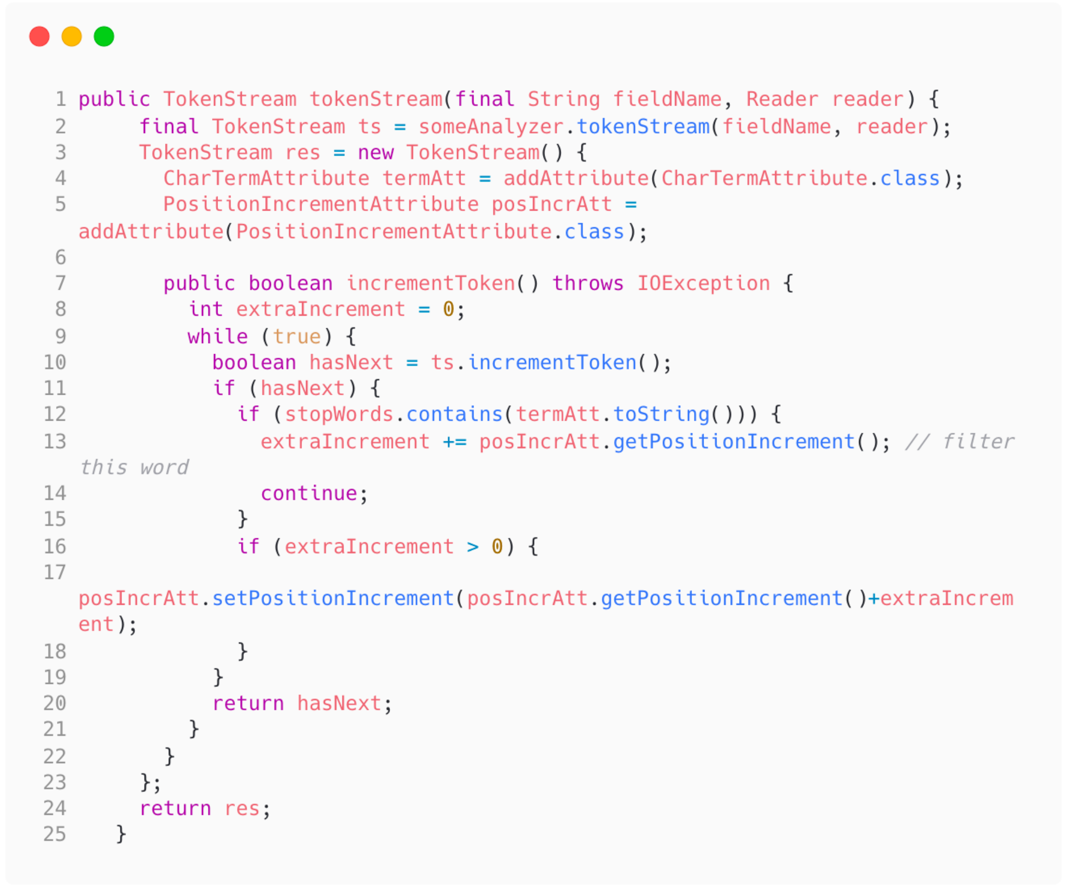


```
document.add(new Field("f","first ends",...);
     document.add(new Field("f","starts two",...);
     indexWriter.addDocument(document);

...

Version matchVersion = Version.LUCENE_XY; // Substitute desired Lucene version for XY
   Analyzer myAnalyzer = new StandardAnalyzer(matchVersion) {
     public int getPositionIncrementGap(String fieldName) {
       return 10;
     }
   };
```

Token Position Increments



```java
public TokenStream tokenStream(final String fieldName, Reader reader) {
     final TokenStream ts = someAnalyzer.tokenStream(fieldName, reader);
     TokenStream res = new TokenStream() {
       CharTermAttribute termAtt = addAttribute(CharTermAttribute.class);
       PositionIncrementAttribute posIncrAtt = addAttribute(PositionIncrementAttribute.class);

       public boolean incrementToken() throws IOException {
         int extraIncrement = 0;
         while (true) {
           boolean hasNext = ts.incrementToken();
           if (hasNext) {
             if (stopWords.contains(termAtt.toString())) {
               extraIncrement += posIncrAtt.getPositionIncrement(); // filter this word
               continue;
             }
             if (extraIncrement > 0) {
               posIncrAtt.setPositionIncrement(posIncrAtt.getPositionIncrement()+extraIncrement);
             }
           }
           return hasNext;
         }
       }
     };
     return res;
   }
```

References

Materials in this slides are obtained (included partially modify) from the following resources, including websites, content providers, printed materials, etc.:

- https://lucene.apache.org/
- https://www.lucenetutorial.com/