LUCENE - STANDARDANALYZER

Introduction

This is the most sofisticated analyzer and is capable of handling names, email address etc. It lowercases each token and removes common words and punctuation if any.

Class declaration

Following is the declaration for **org.apache.lucene.analysis.StandardAnalyzer** class:

```
public final class StandardAnalyzer
   extends StopwordAnalyzerBase
```

Fields

- **static int DEFAULT_MAX_TOKEN_LENGTH** - Default maximum allowed token length
- **static Set<?> STOP_WORDS_SET** - An unmodifiable set containing some common English words that are usually not useful for searching.

Class constructors

S.N.	Constructor & Description
1	**StandardAnalyzer*VersionmatchVersion*** Builds an analyzer with the default stop words $STOP_WORDS_SET$.
2	**StandardAnalyzer*VersionmatchVersion, Filestopwords*** Deprecated. Use StandardAnalyzer*Version, Reader* instead.
3	**StandardAnalyzer*VersionmatchVersion, Readerstopwords*** Builds an analyzer with the stop words from the given reader.
4	**StandardAnalyzer*VersionmatchVersion, Set < ? > stopWords*** Builds an analyzer with the given stop words.

Class methods

S.N.	Method & Description
1	**protected Reusable Analyzer Base. Token Stream Components create Components*StringfieldName, Readerreader*** Creates a new ReusableAnalyzerBase.TokenStreamComponents instance for this analyzer.

2	**int getMaxTokenLength**
3	**void setMaxTokenLength***intlength* Set maximum allowed token length.

Methods inherited

This class inherits methods from the following classes:

- org.apache.lucene.analysis.StopwordAnalyzerBase
- org.apache.lucene.analysis.ReusableAnalyzerBase
- org.apache.lucene.analysis.Analyzer
- java.lang.Object

Usage

```
private void displayTokenUsingStandardAnalyzer() throws IOException{
   String text
      = "Lucene is simple yet powerful java based search library.";
   Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_36);
   TokenStream tokenStream
      = analyzer.tokenStream(LuceneConstants.CONTENTS,
        new StringReader(text));
   TermAttribute term = tokenStream.addAttribute(TermAttribute.class);
   while(tokenStream.incrementToken()) {
      System.out.print("[" + term.term() + "] ");
   }
}
```

Example Application

Let us create a test Lucene application to test search using BooleanQuery.

Step	Description
1	Create a project with a name *LuceneFirstApplication* under a package *com.tutorialspoint.lucene* as explained in the *Lucene - First Application* chapter. You can also use the project created in *Lucene - First Application* chapter as such for this chapter to understand searching process.
2	Create *LuceneConstants.java* as explained in the *Lucene - First Application* chapter. Keep rest of the files unchanged.
3	Create *LuceneTester.java* as mentioned below.
4	Clean and Build the application to make sure business logic is working as per the requirements.

LuceneConstants.java

This class is used to provide various constants to be used across the sample application.

```
package com.tutorialspoint.lucene;
```

```
public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

LuceneTester.java

This class is used to test the searching capability of lucene library.

```
package com.tutorialspoint.lucene;

import java.io.IOException;
import java.io.StringReader;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.StandardAnalyzer;
import org.apache.lucene.analysis.tokenattributes.TermAttribute;
import org.apache.lucene.util.Version;

public class LuceneTester {

   public static void main(String[] args) {
      LuceneTester tester;

      tester = new LuceneTester();

      try {
         tester.displayTokenUsingStandardAnalyzer();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   private void displayTokenUsingStandardAnalyzer() throws IOException{
      String text
         = "Lucene is simple yet powerful java based search library.";
      Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_36);
      TokenStream tokenStream = analyzer.tokenStream(
         LuceneConstants.CONTENTS, new StringReader(text));
      TermAttribute term = tokenStream.addAttribute(TermAttribute.class);
      while(tokenStream.incrementToken()) {
         System.out.print("[" + term.term() + "] ");
      }
   }
}
```

Running the Program:

Once you are done with creating source, you are ready for this step which is compiling and running your program. To do this, Keep LuceneTester.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **LuceneTester** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

```
[lucene] [simple] [yet] [powerful] [java] [based] [search] [library]
```