Sahitya Khoz: Hindi Search Engine

- Primary focus on Hindi Stemmer

Impact This Project

- Hindi, the national language of India is widely spoken in the country and is the most preferred language after English.

- Hindi has one of the richest vocabularies, script, grammar, word collection, and parts of speech.

- Our search engine focuses on the major problems of Hindi text searching over the Hindi literature.

- This will help literature critics and reviewers to appreciate and contribute towards it.

- Literature search provides not only an opportunity to learn more about a given topic but provides insight on how the topic was studied by previous analysts.

- It helps to interpret ideas, detect shortcomings and recognise opportunities.

About The Project

- The primary focus of this project was to develop a Hindi stemmer and integrate it with the Hindi search engine based on lucene.

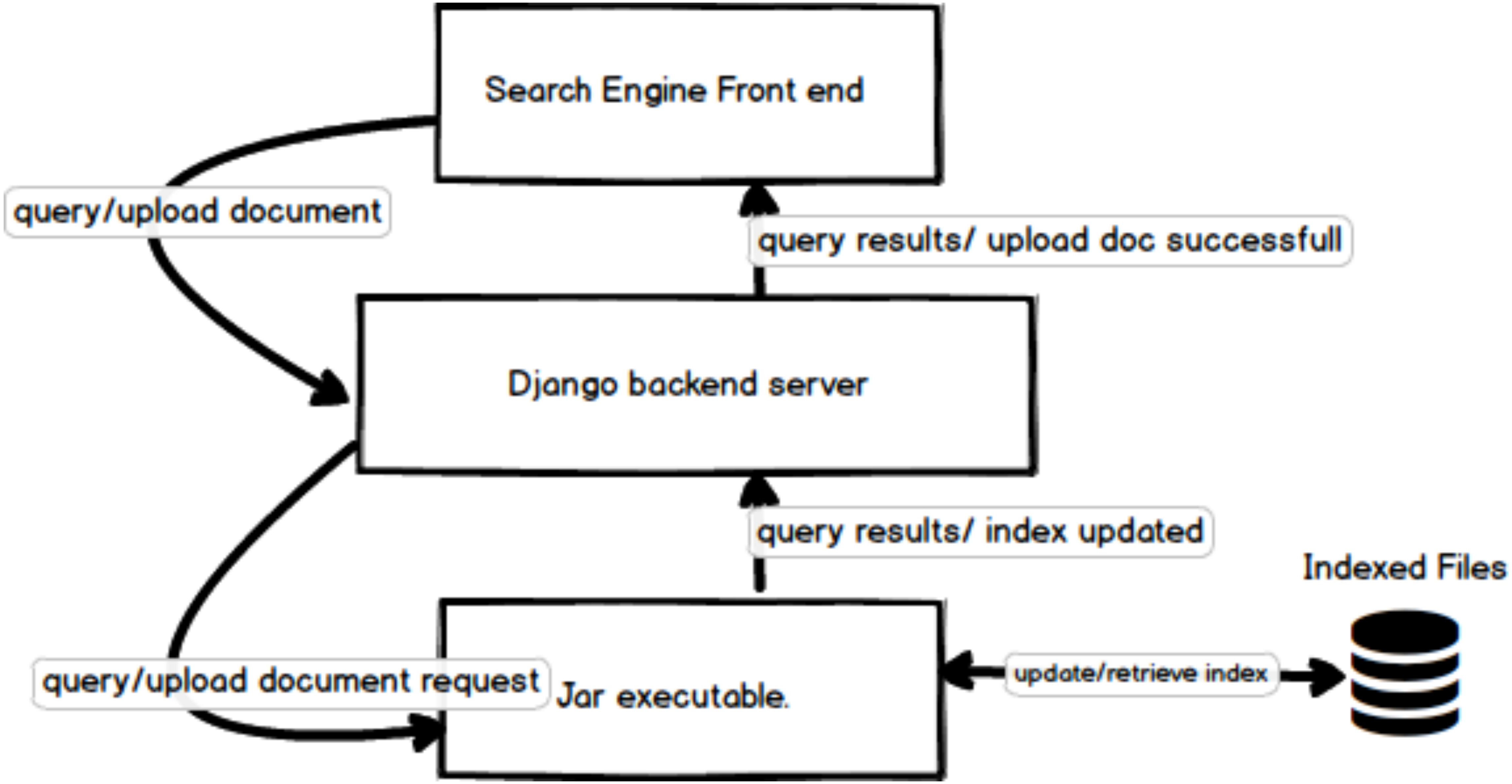
- The created search engine can be used to search in Hindi literature and texts.

- Search engine utilise a variety of Information retrieval techniques and one of the most important and common one being stemming the terms.

- Most often we take stemming for granted, however, for understanding the basic concept of stemming the best way is to implement it.

- For developing a stemmer for Hindi, we try to formulate various language rules for Hindi using web research and other sources, and then evaluate it on our own test corpus containing 20 documents with each document containing 60 terms each .

Server Architecture



Hosting the Search Engine

- We compile our search engine into a executable .jar file, and use it to fetch the results of the query.
- The .jar file is run from a python script, which is called from inside of a django-based server.
- The response from .jar file is shown to the user. User can upload documents, on which, .jar is called to update its index.

Common Character Encodings

ASCII: It is the most popular standard to represent character in computer. It uses 7 bits to code each character.

ISCII: It is standardized by Bureau of Indian Standard. It is an 8 bit code which encodes both English and Indian Script Alphabet.

Unicode: It has been standardized specifically to accommodate a large number of special symbols such as Greek characters, mathematical symbols and non-English characters. It uses 16 bits to represent each character.

UTF-8: It is suitable for texts that are mostly Latin alphabet letters. For example, English, French, and most web technology such as HTML, CSS, JS.

UTF-16: For Asian language containing lots of Chinese and Japanese characters, It create smaller size files.

WORKING OF Sahitya Khoz

- Pre-Processing

- We created a IndexWriterConfig which stores the configurations to create an index.
- We created a HindiAnalyser which uses our custom stemmer for stemming and other open sourced packages for normalization etc.
- We create the index in ADD_OR_APPEND mode as it is the most convenient one.
- We then initialize an IndexWriter object which takes the input corpus directory and IndexWriterConfig we created earlier as arguments.

WORKING OF Sahitya Khoz

- Pre-Processing

- Now, we call a indexDocs function which recursively iterates to all input files and directories.

- We then create a lucene Document for each file that we visit. Each document contains 3 fields: path of the input file, last modified time and the actual contents of the file.

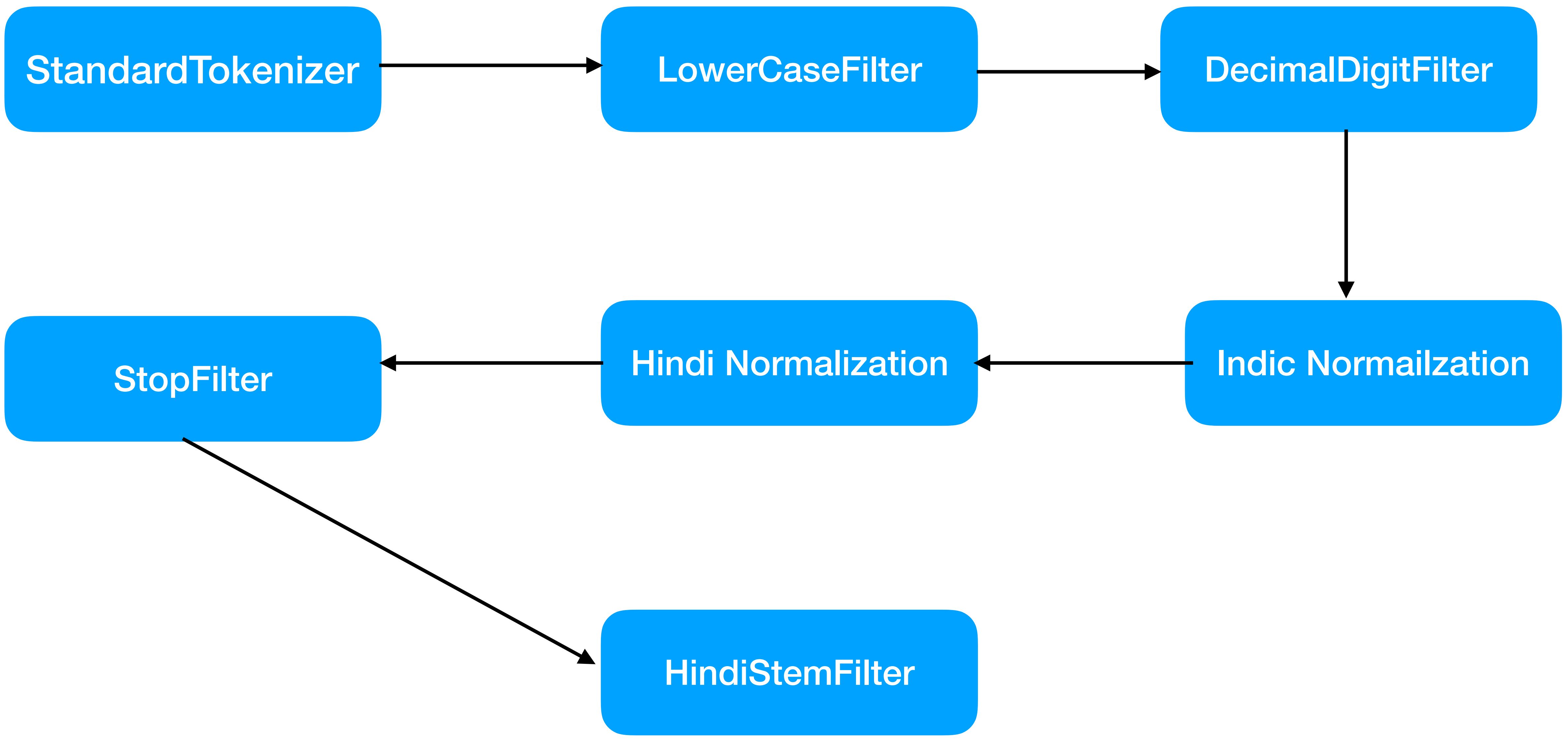
- We finally call the IndexWriter to further process the document and then write it into the index.

WORKING OF Sahitya Khoz

-Processing

- Further processing on the documents are StandardTokenizer, LowerCaseFilter, DecimalDigitFilter, IndicNormalization, HindiNormalization, StopFilter, and finally HindiStemFilter.
- When we query the items, each query is processed in the same fashion as each document is processed above, and then the TopDocs are fetched. The search is done in the contents field of the Documents and the top hits are returned with their scores.

Hindi Analyser Workflow:



StandardTokenizer

LowerCaseFilter

DecimalDigitFilter

Indic Normailzation

Hindi Normalization

StopFilter

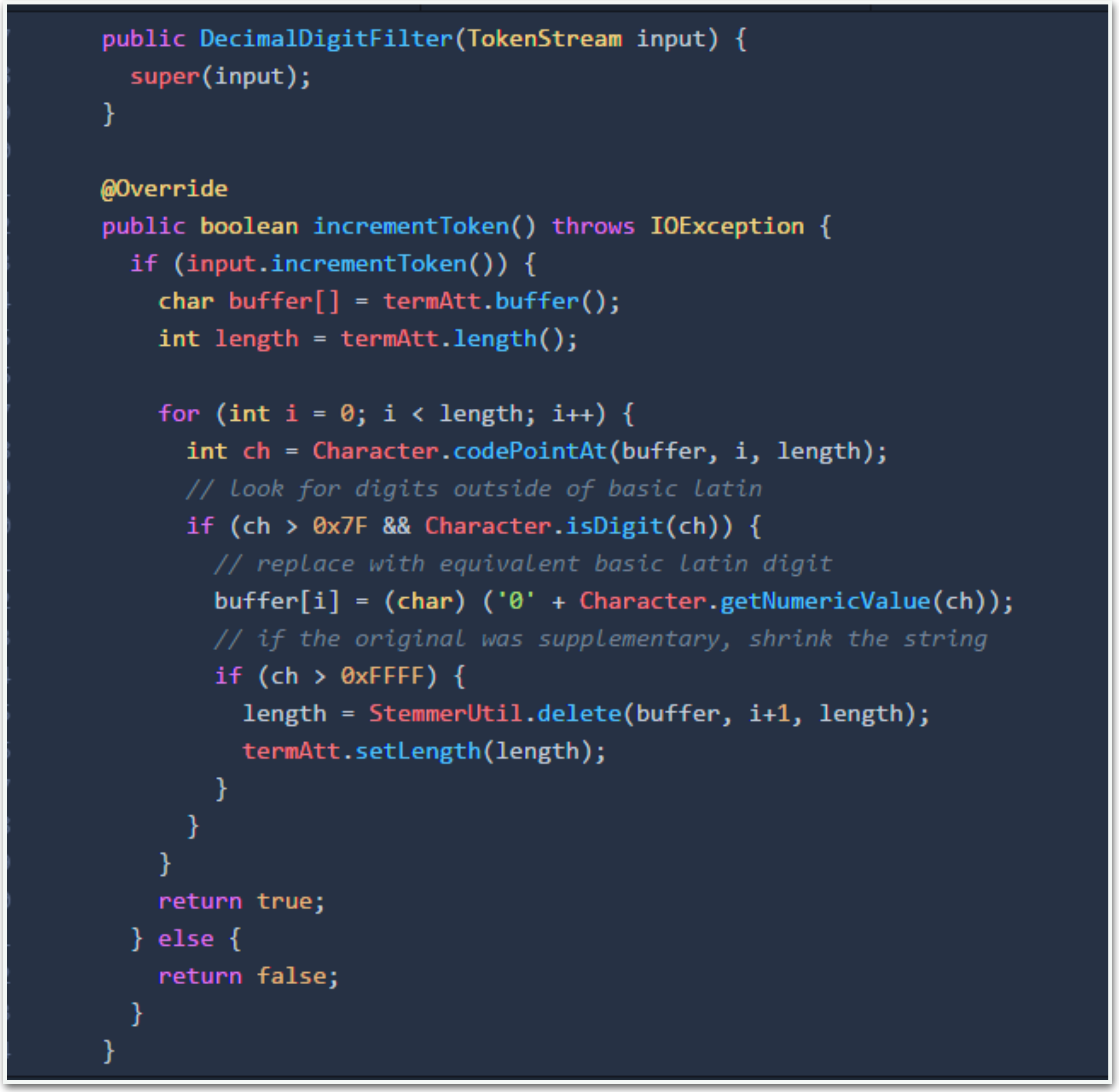
HindiStemFilter

Standard Tokenizer :

- Tokenisation is the task of converting documents unit into pieces, known as tokens while throwing away some characters such as spaces or punctuation marks.
- StandardTokenizer implements word break rules from the Unicode Text Segmentation algorithm and converts the content of our documents into tokens.
- This tokenizer was good enough to meet our requirements.

Lowercase Filter :

LowerCaseFilter normalizes token text to lowercase. There is no such thing as lower case in Hindi, but it’s better to use this to convert Latin text such as time, document name etc. present in the document to lowercase.



```
public DecimalDigitFilter(TokenStream input) {
  super(input);
}

@Override
public boolean incrementToken() throws IOException {
  if (input.incrementToken()) {
    char buffer[] = termAtt.buffer();
    int length = termAtt.length();

    for (int i = 0; i < length; i++) {
      int ch = Character.codePointAt(buffer, i, length);
      // look for digits outside of basic latin
      if (ch > 0x7F && Character.isDigit(ch)) {
        // replace with equivalent basic latin digit
        buffer[i] = (char) ('0' + Character.getNumericValue(ch));
        // if the original was supplementary, shrink the string
        if (ch > 0xFFFF) {
          length = StemmerUtil.delete(buffer, i+1, length);
          termAtt.setLength(length);
        }
      }
    }
    return true;
  } else {
    return false;
  }
}
```

Decimal Digit Filter

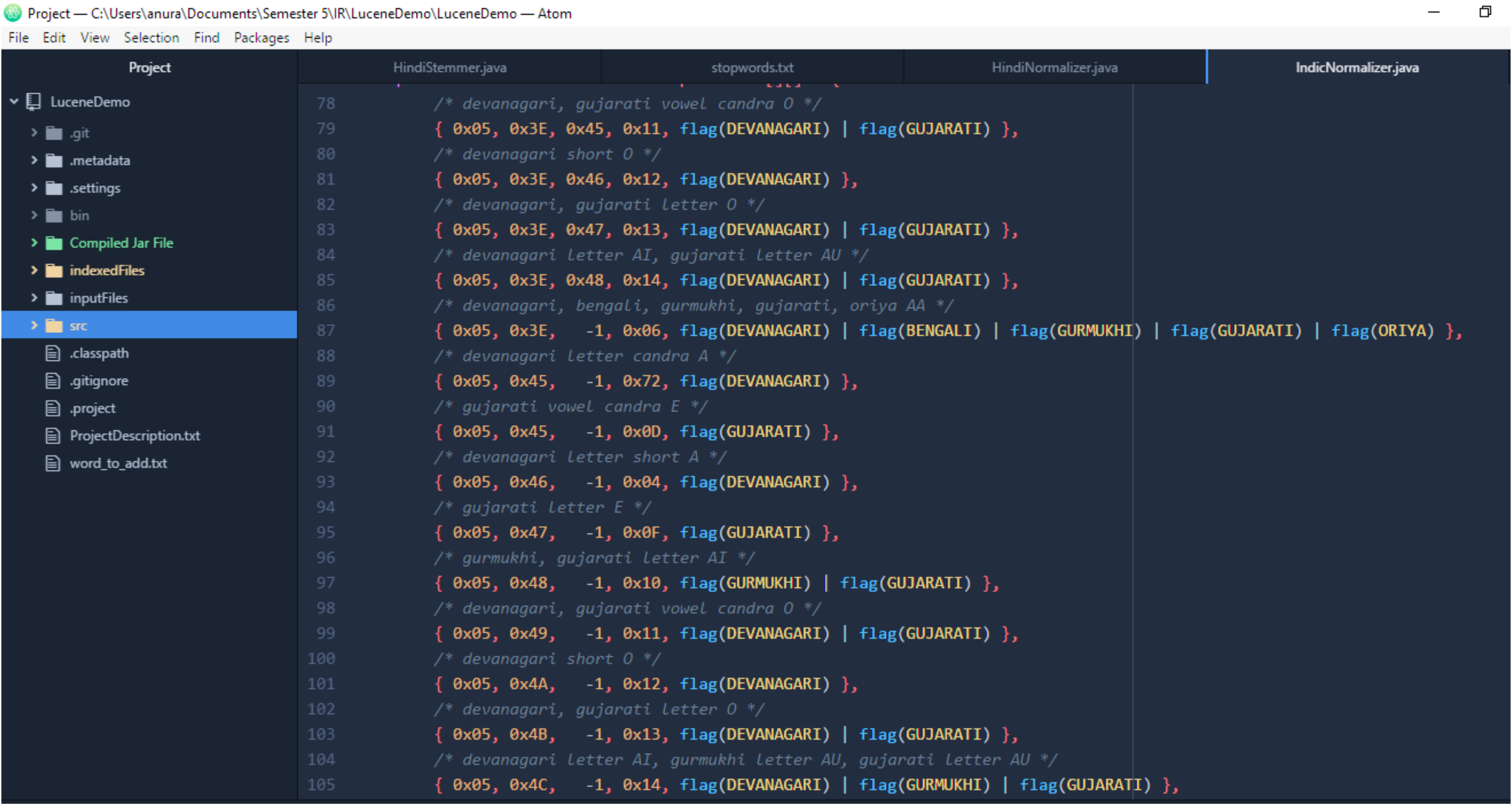
It looks for digits outside of basic latin and replace it with basic equivalent latin digit. So, this converts Hindi digits into their corresponding Latin equivalents.

Indic Normailzer

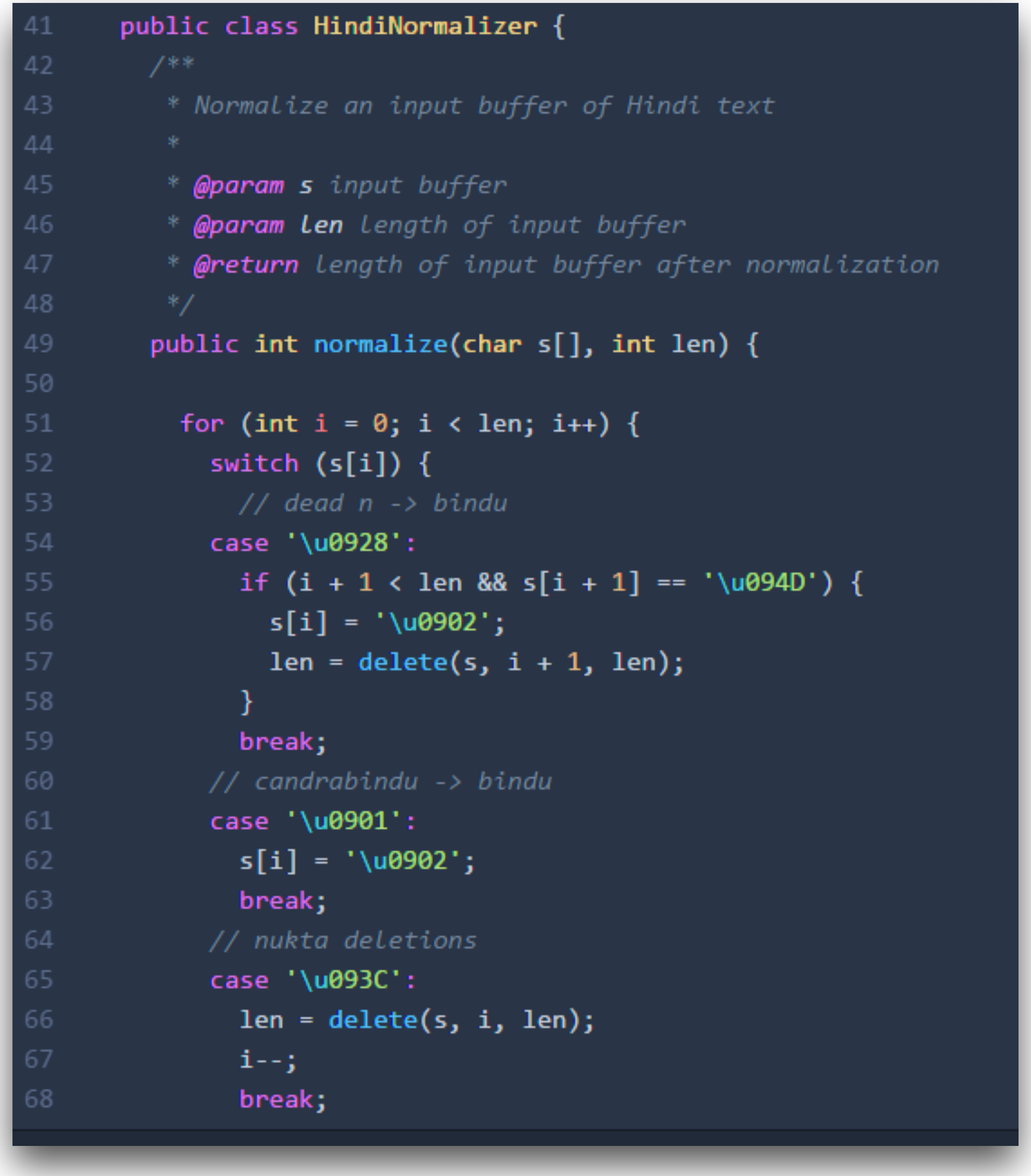
It normalizes the Unicode representation of text in Indian languages. It follows guidelines from Unicode 5.2, chapter 6, South Asian Scripts and graphical decomposition from Indian scripts and Unicode.

http://ldc.upenn.edu/myl/IndianScriptsUnicode.html

Indic Normailzer



```
    /* devanagari, gujarati vowel candra O */
    { 0x05, 0x3E, 0x45, 0x11, flag(DEVANAGARI) | flag(GUJARATI) },
    /* devanagari short O */
    { 0x05, 0x3E, 0x46, 0x12, flag(DEVANAGARI) },
    /* devanagari, gujarati letter O */
    { 0x05, 0x3E, 0x47, 0x13, flag(DEVANAGARI) | flag(GUJARATI) },
    /* devanagari letter AI, gujarati letter AU */
    { 0x05, 0x3E, 0x48, 0x14, flag(DEVANAGARI) | flag(GUJARATI) },
    /* devanagari, bengali, gurmukhi, gujarati, oriya AA */
    { 0x05, 0x3E,   -1, 0x06, flag(DEVANAGARI) | flag(BENGALI) | flag(GURMUKHI) | flag(GUJARATI) | flag(ORIYA) },
    /* devanagari letter candra A */
    { 0x05, 0x45,   -1, 0x72, flag(DEVANAGARI) },
    /* gujarati vowel candra E */
    { 0x05, 0x45,   -1, 0x0D, flag(GUJARATI) },
    /* devanagari letter short A */
    { 0x05, 0x46,   -1, 0x04, flag(DEVANAGARI) },
    /* gujarati letter E */
    { 0x05, 0x47,   -1, 0x0F, flag(GUJARATI) },
    /* gurmukhi, gujarati letter AI */
    { 0x05, 0x48,   -1, 0x10, flag(GURMUKHI) | flag(GUJARATI) },
    /* devanagari, gujarati vowel candra O */
    { 0x05, 0x49,   -1, 0x11, flag(DEVANAGARI) | flag(GUJARATI) },
    /* devanagari short O */
    { 0x05, 0x4A,   -1, 0x12, flag(DEVANAGARI) },
    /* devanagari, gujarati letter O */
    { 0x05, 0x4B,   -1, 0x13, flag(DEVANAGARI) | flag(GUJARATI) },
    /* devanagari letter AI, gurmukhi letter AU, gujarati letter AU */
    { 0x05, 0x4C,   -1, 0x14, flag(DEVANAGARI) | flag(GURMUKHI) | flag(GUJARATI) },
```

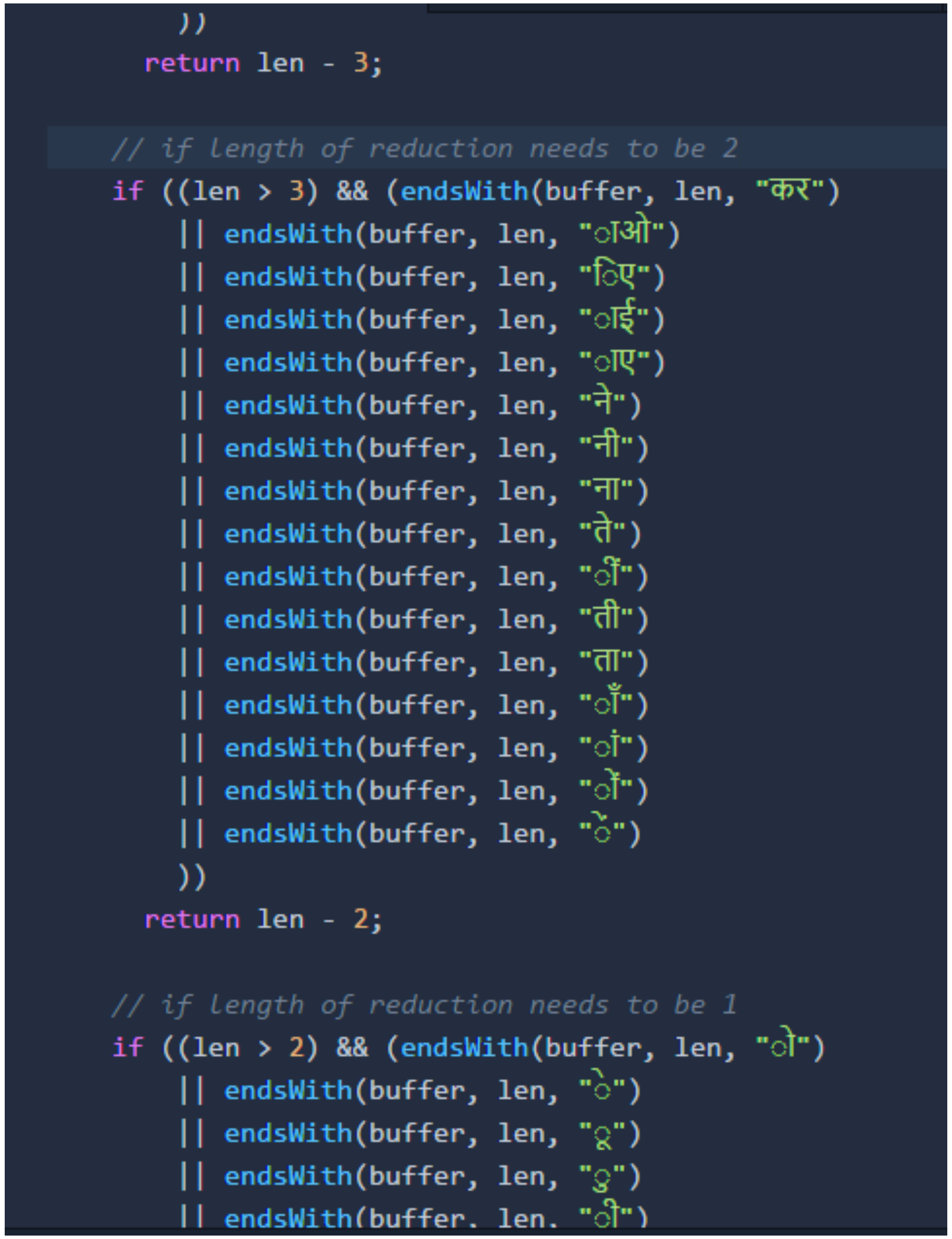


```
public class HindiNormalizer {
  /**
   * Normalize an input buffer of Hindi text
   *
   * @param s input buffer
   * @param len length of input buffer
   * @return length of input buffer after normalization
   */
  public int normalize(char s[], int len) {

    for (int i = 0; i < len; i++) {
      switch (s[i]) {
        // dead n -> bindu
      case '\u0928':
        if (i + 1 < len && s[i + 1] == '\u094D') {
          s[i] = '\u0902';
          len = delete(s, i + 1, len);
        }
        break;
      // candrabindu -> bindu
      case '\u0901':
        s[i] = '\u0902';
        break;
      // nukta deletions
      case '\u093C':
        len = delete(s, i, len);
        i--;
        break;
```

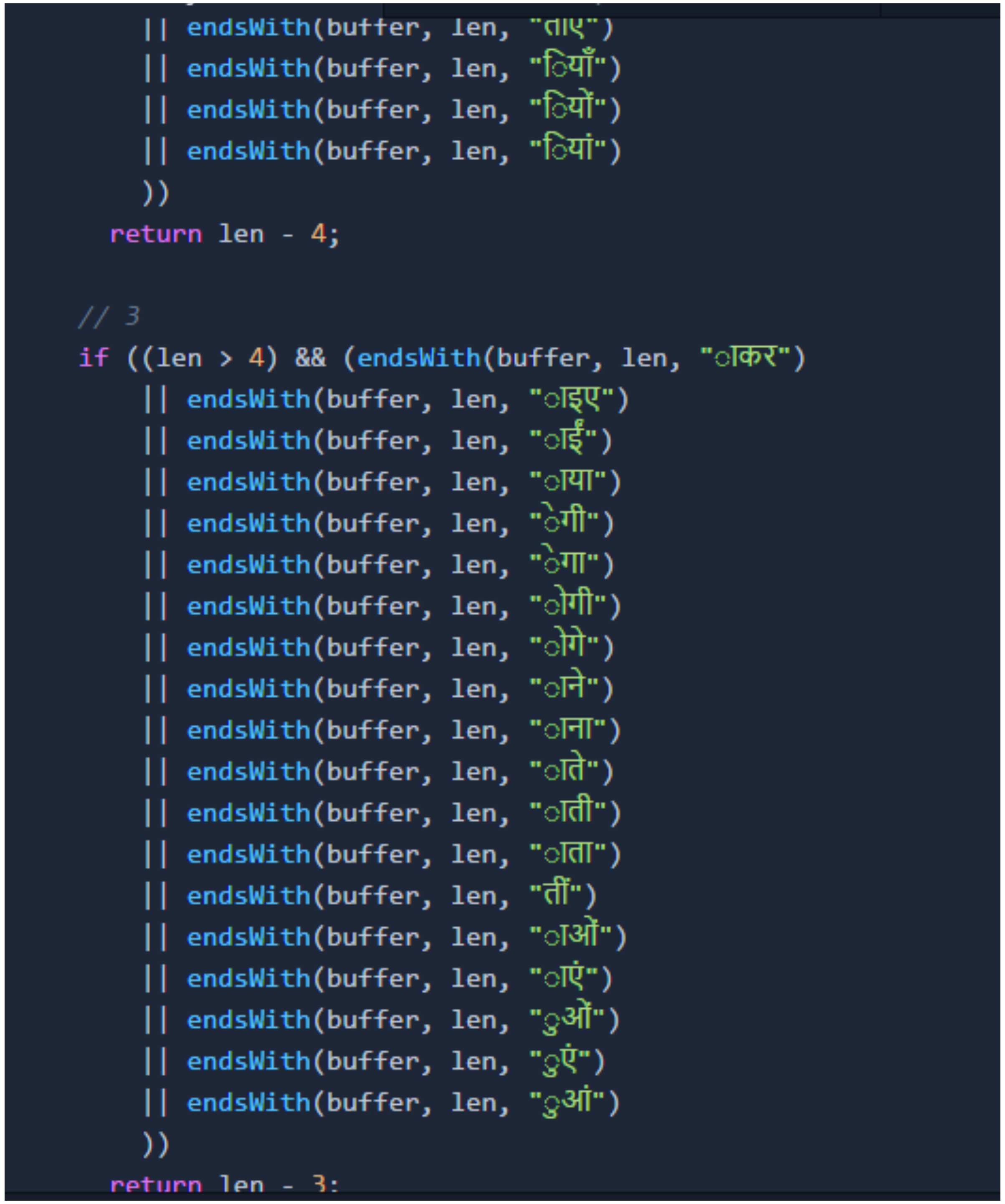
Hindi Normalizer

It normalizes the Hindi text to remove differences in spelling variations. It implements Hindi language specific algorithm specified in Word Normalization in Indian Languages along with certain additions from Hindi CLIR in Thirty Days.

For ex: ँ is normalized to anusvara, which is represented with a dot (bindu) above the letter (e.g. मं). virama(्) is deleted.



```
  ))
 return len - 3;

// if length of reduction needs to be 2
if ((len > 3) && (endsWith(buffer, len, "कर")
  || endsWith(buffer, len, "ाओ")
  || endsWith(buffer, len, "िए")
  || endsWith(buffer, len, "ाई")
  || endsWith(buffer, len, "ाए")
  || endsWith(buffer, len, "ने")
  || endsWith(buffer, len, "नी")
  || endsWith(buffer, len, "ना")
  || endsWith(buffer, len, "ते")
  || endsWith(buffer, len, "ीं")
  || endsWith(buffer, len, "ती")
  || endsWith(buffer, len, "ता")
  || endsWith(buffer, len, "ाँ")
  || endsWith(buffer, len, "ां")
  || endsWith(buffer, len, "ों")
  || endsWith(buffer, len, "ें")
  ))
 return len - 2;

// if length of reduction needs to be 1
if ((len > 2) && (endsWith(buffer, len, "ो")
  || endsWith(buffer, len, "े")
  || endsWith(buffer, len, "ू")
  || endsWith(buffer, len, "ु")
  || endsWith(buffer, len, "ी")
```



```
  || endsWith(buffer, len, "ाए")
  || endsWith(buffer, len, "ियाँ")
  || endsWith(buffer, len, "ियों")
  || endsWith(buffer, len, "ियां")
  ))
 return len - 4;

// 3
if ((len > 4) && (endsWith(buffer, len, "ाकर")
  || endsWith(buffer, len, "ाइए")
  || endsWith(buffer, len, "ाईं")
  || endsWith(buffer, len, "ाया")
  || endsWith(buffer, len, "ेगी")
  || endsWith(buffer, len, "ेगा")
  || endsWith(buffer, len, "ोगी")
  || endsWith(buffer, len, "ोगे")
  || endsWith(buffer, len, "ाने")
  || endsWith(buffer, len, "ाना")
  || endsWith(buffer, len, "ाते")
  || endsWith(buffer, len, "ाती")
  || endsWith(buffer, len, "ाता")
  || endsWith(buffer, len, "तीं")
  || endsWith(buffer, len, "ाओं")
  || endsWith(buffer, len, "ाएं")
  || endsWith(buffer, len, "ुओं")
  || endsWith(buffer, len, "ुएं")
  || endsWith(buffer, len, "ुआं")
  ))
 return len - 3;
```

HindiStemmer

Hindi Stemmer

HindiStemFilter is created by us, using the algorithm specified in a Lightweight Stemmer for Hindi.

We will discuss the rules of stemming in short below.

When the length of reduction needs to be 5. We remove the following 5 letter suffixes from our Hindi terms:
ाएंगी, ाएंगे , ाऊंगी , ाऊंगा , ाइयाँ , ाइयों , ाइयां

For ex: बनाएंगी stem form is बन .

When the length of reduction needs to be 4. We remove the following 4 letter suffixes from our Hindi terms:
ाएगी ,ाएगा ,ाओगी ,ाओगे ,एंगी ,ेंगी ,एंगे ,ेंगे ,ूंगी ,ूंगा ,ातीं ,नाओं, नाएं,
ताओं ,ताएं ,ियाँ ,ियों ,ियांयों ,”ियां

For ex: पढ़ाएंगे stem form is पढ़ .

Hindi Stemmer

When the length of reduction needs to be 3. We remove the following 3 letter suffixes from our Hindi terms:

ाकर,ाइए,ाई,ाया,ेगी,ेगा,ोगी,ोगे,ाने,ाना,ाते,ाती,ाता,तीं,ाओं,ाएं
,ुओं,ुएं,ुआं

For ex: कराते stem form is कर .

When the length of reduction needs to be 2. We remove the following 2 letter suffixes from our Hindi terms:

कर,ाओ,िए,ाई,ाए,ने,नी,ना,ते,ीं,ती,ता,ाँ,ां,ों,ें

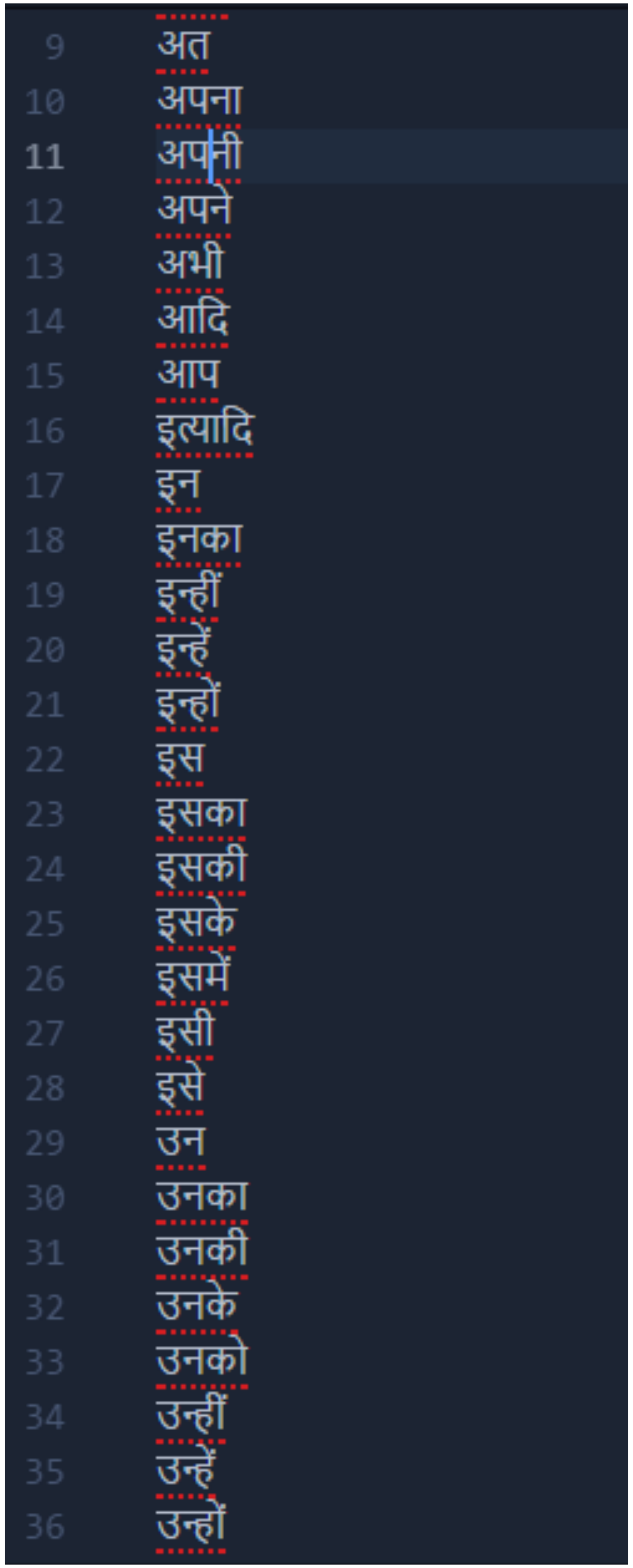
For ex: देखते stem form is देख .

When the length of reduction needs to be 1. We remove the following 1 letter suffixes from our Hindi terms:

ो,े,ू,ु,ी,ि,ा

For ex: घूमे stem form is घूम .

The rules of stemming are kept in if-else, with the rules of highest suffix being first. And only one stemming rule is applied on each term. So the rules of length 5 suffix are matched first, followed by rules of suffixes of length 4, 3, 2, 1 respectively. After Stemming, the stem form of the term is returned.



```
अत
अपना
अपनी
अपने
अभी
आदि
आप
इत्यादि
इन
इनका
इन्हीं
इन्हें
इन्हों
इस
इसका
इसकी
इसके
इसमें
इसी
इसे
उन
उनका
उनकी
उनके
उनको
उन्हीं
उन्हें
उन्हों
```

Stop Filter

StopFilter is the list of stop words containing 200+ words from the Hindi language. Words present in Stop Filter list are ignored while creating the index. The words list is taken from IR Multilingual resources at UniNE.

http://members.unine.ch/jacques.savoy/clef/index.html

Scoring

VSM
(Vector Space
Model)
+
Weight/
Score
(TFIDF)
=
Scoring

- Lucene scoring is the heart of why it is so popular.
- By default it uses Vector Space Model(VSM) and TFIDF weights to score the documents.
- Various state of the art ranking algorithms are used by Lucene to assign scores but by default it is figure out by similarity metric.
- Term Frequency (TF): no. of times term t repeat in a document.
- Inverse Document Frequency(IDF): ln(total no. of documents/ no. of documents with term t)

Similarity Metric

Docs.	Milk	Eggs	Bread	Butter	Wine	Vodka	Chicken	Chips
cons 1	1	1	1	1				
cons 2					1		1	1
cons 3	1		1	1				
cons 4						1	1	1
cons 5					1		1	1

- If you search for “milk” the system should return documents containing “milk”, “eggs”, and “bread”.

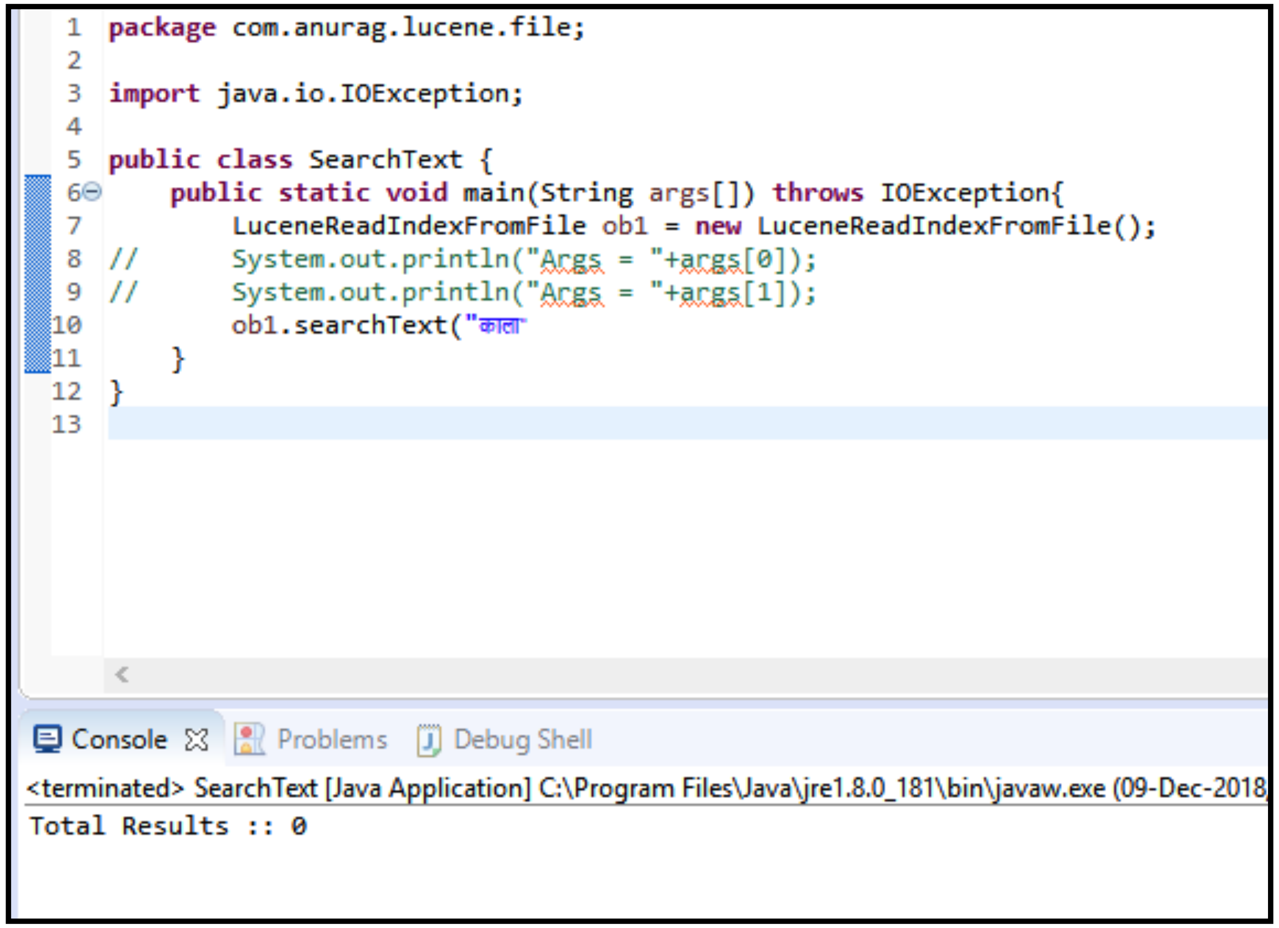
- If you search for “wine” the system should return documents containing “wine”, “vodka”, “chicken”, and “chips”.

Mathematical tools used to estimate the strength of the semantic relationship between units of documents.

Lucene Algorithms

TFIDF Similarity
BM25 Similarity
Multi Similarity
PerFieldSimilarity
SimilarityBase

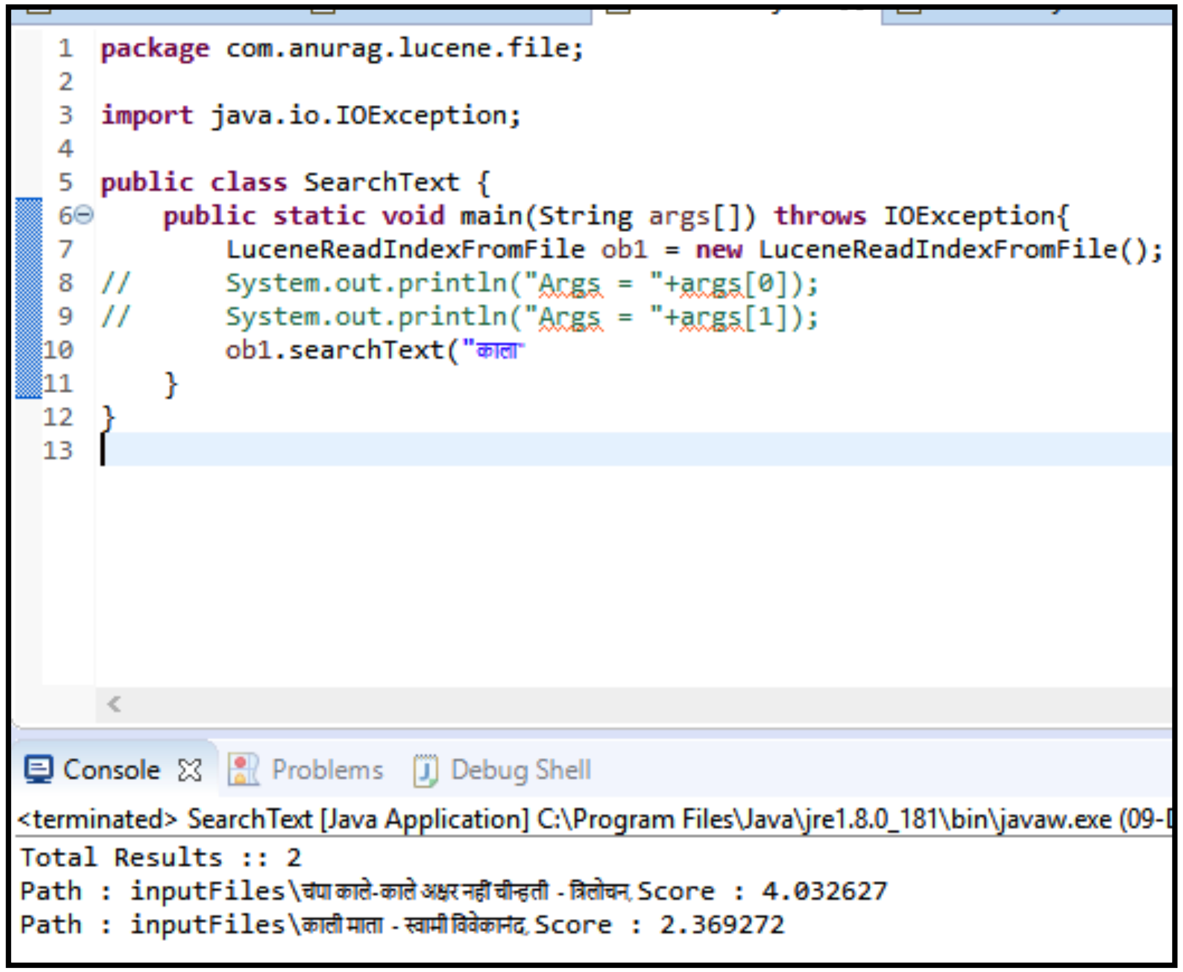
Without Stemming



```java
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("काला"
    }
}

```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018
Total Results :: 0
```

With Stemming

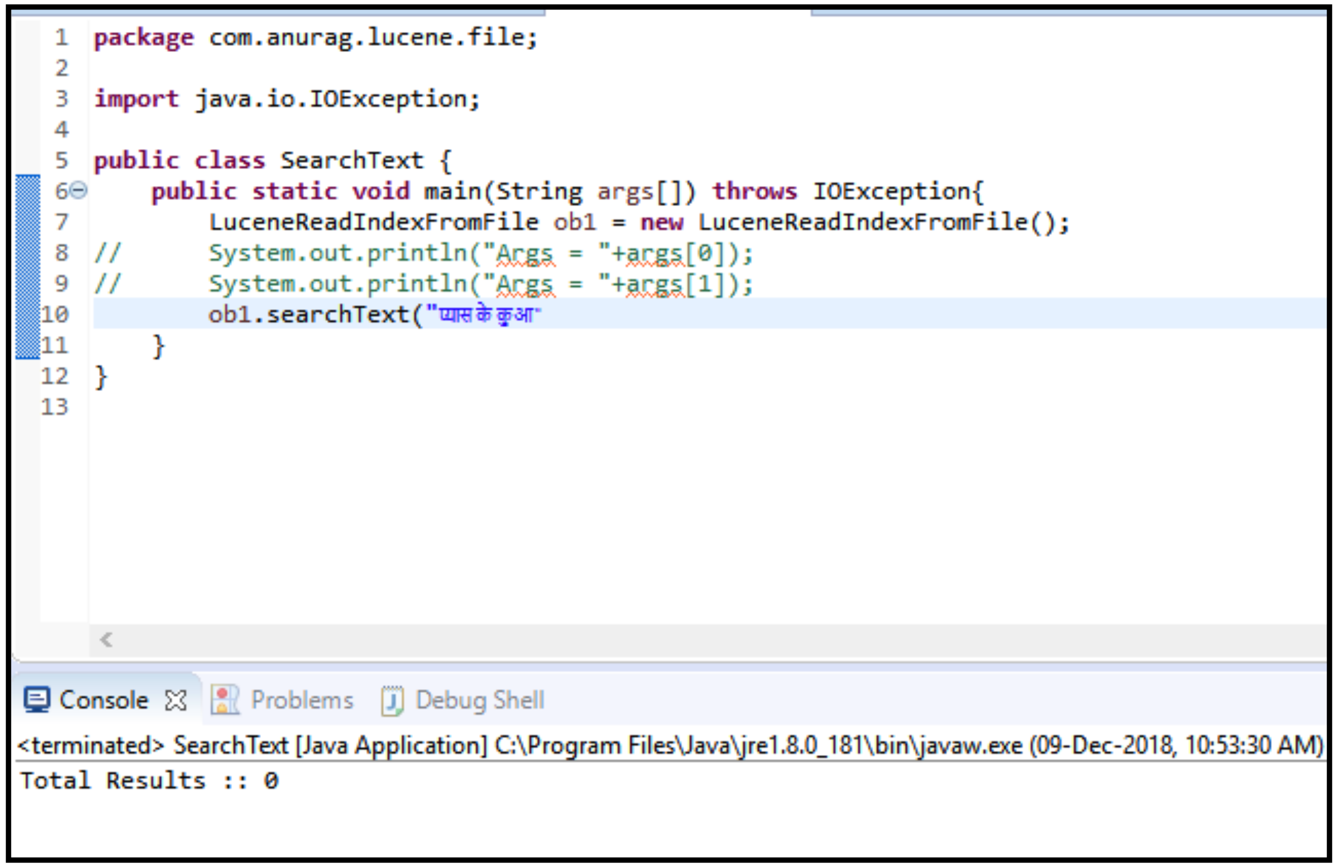


```java
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("काला"
    }
}

```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-I
Total Results :: 2
Path : inputFiles\चंपा काले-काले अक्षर नहीं चीन्हती - त्रिलोचन Score : 4.032627
Path : inputFiles\काली माता - स्वामी विवेकानंद Score : 2.369272
```

- **Document contains काले as a token.**
- **With stemming we reduce काले to काल and store it as a term for the index.**
- **We won't be able to find काला in the index without stemming.**

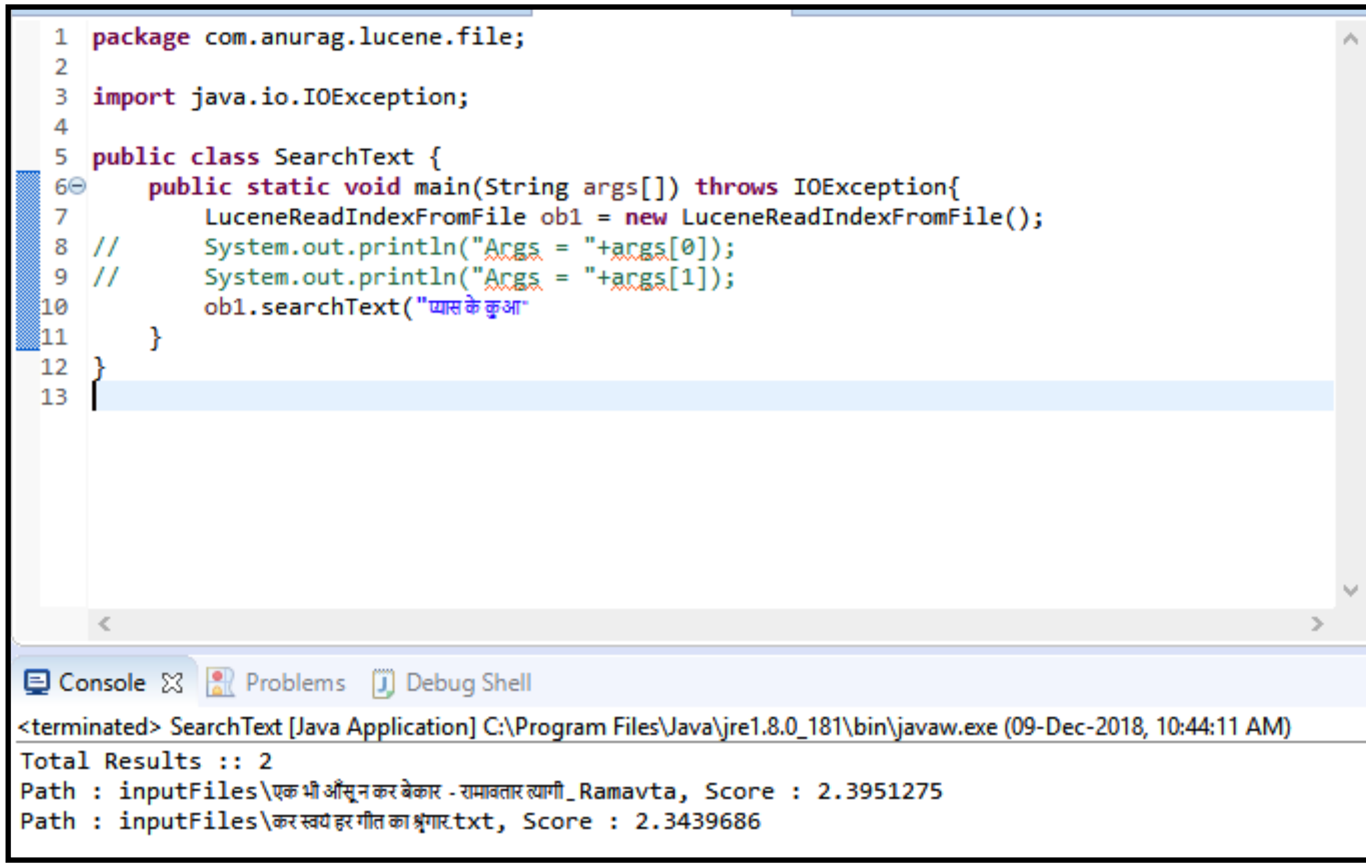
Without Stemming



```java
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("प्यास के कुआ"
    }
}
```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 10:53:30 AM)
Total Results :: 0
```

With Stemming

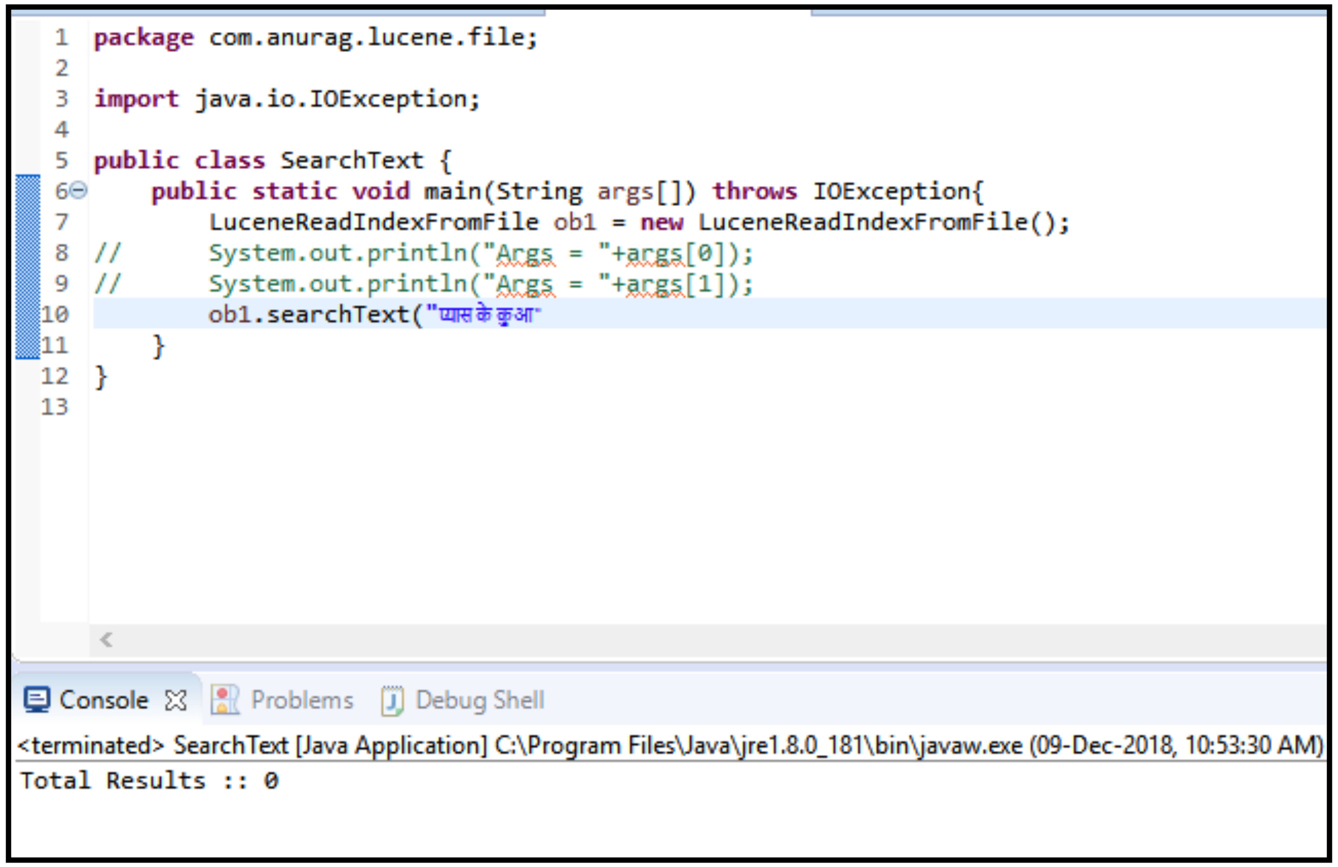


```java
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("प्यास के कुआ"
    }
}

```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 10:44:11 AM)
Total Results :: 2
Path : inputFiles\एक भी आँसू न कर बेकार - रामावतार त्यागी_Ramavta, Score : 2.3951275
Path : inputFiles\कर स्वयं हर गीत का श्रृंगार.txt, Score : 2.3439686
```

- **Two Documents contain “प्यासे के कुऔ”.**
- **We will have प्यास and कुअ as terms in the index.**
- **If the query is “प्यास के कुआ”.**
- **After query processing, we will search for documents containing प्यास and कुअ.**
- **We won’t be able to find प्यास and कुआ in the index without stemming as index has प्यासे and कुऔ as terms in the index .**

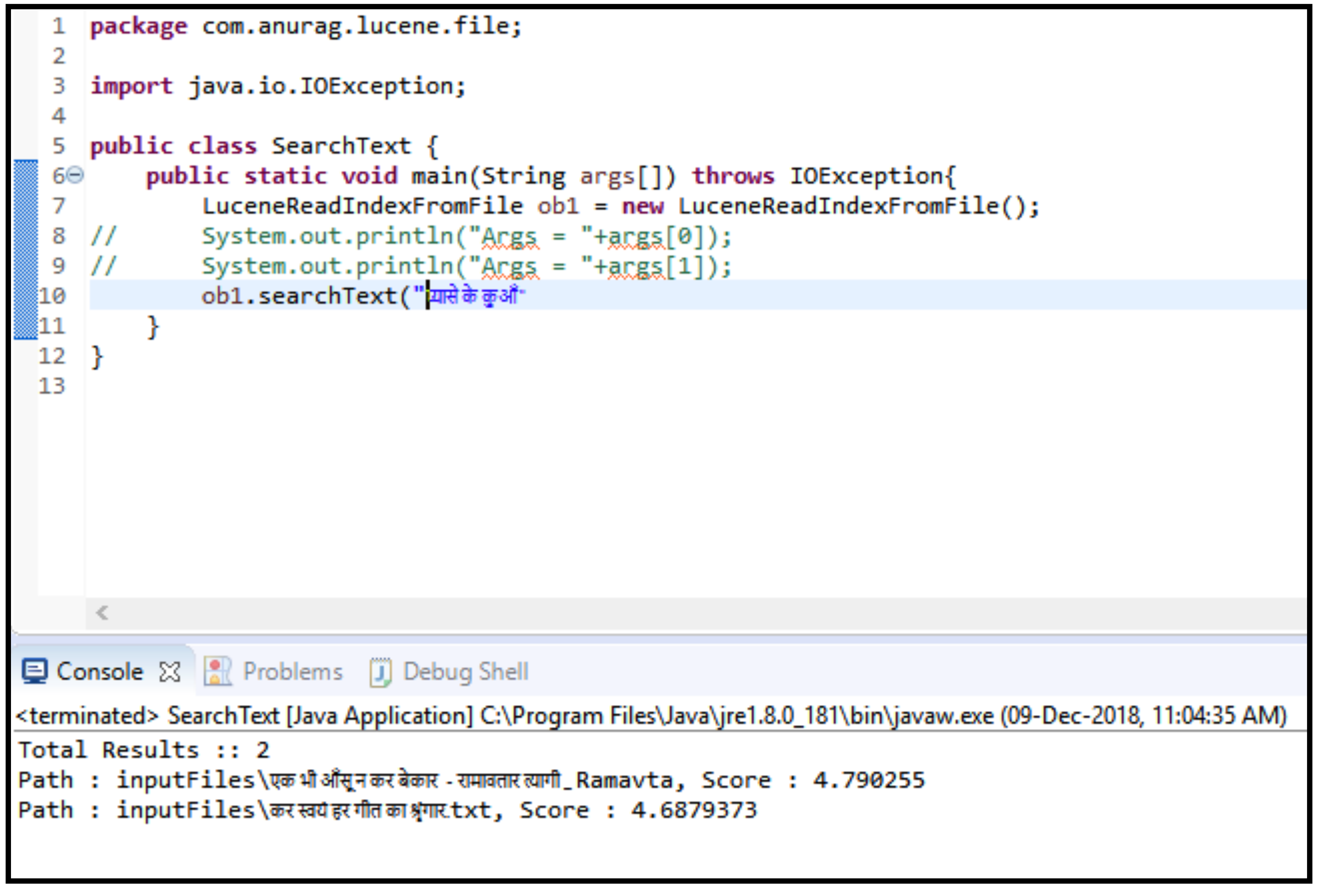
Without Stemming



```
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("प्यास के कुआ"
    }
}
```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 10:53:30 AM)
Total Results :: 0
```

Without Stemming

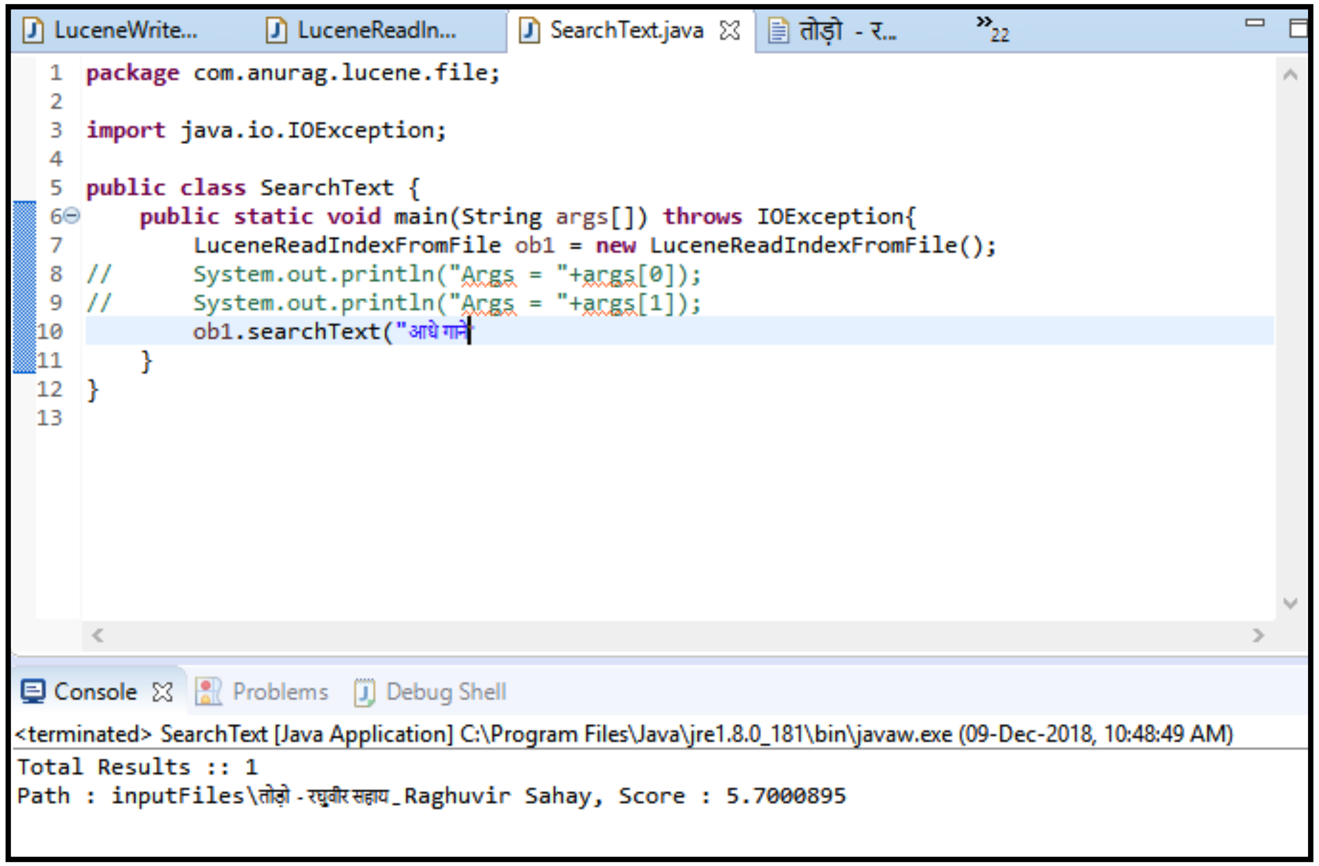


```
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("प्यासे के कुऔं"
    }
}
```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 11:04:35 AM)
Total Results :: 2
Path : inputFiles\एक भी आँसू न कर बेकार - रामावतार त्यागी_Ramavta, Score : 4.790255
Path : inputFiles\कर स्वयं हर गीत का श्रृंगार.txt, Score : 4.6879373
```

- **Two Documents contain “प्यासे के कुऔं” .**
- **We retrieve both the documents after the query “प्यासे के कुऔं” without stemming.**

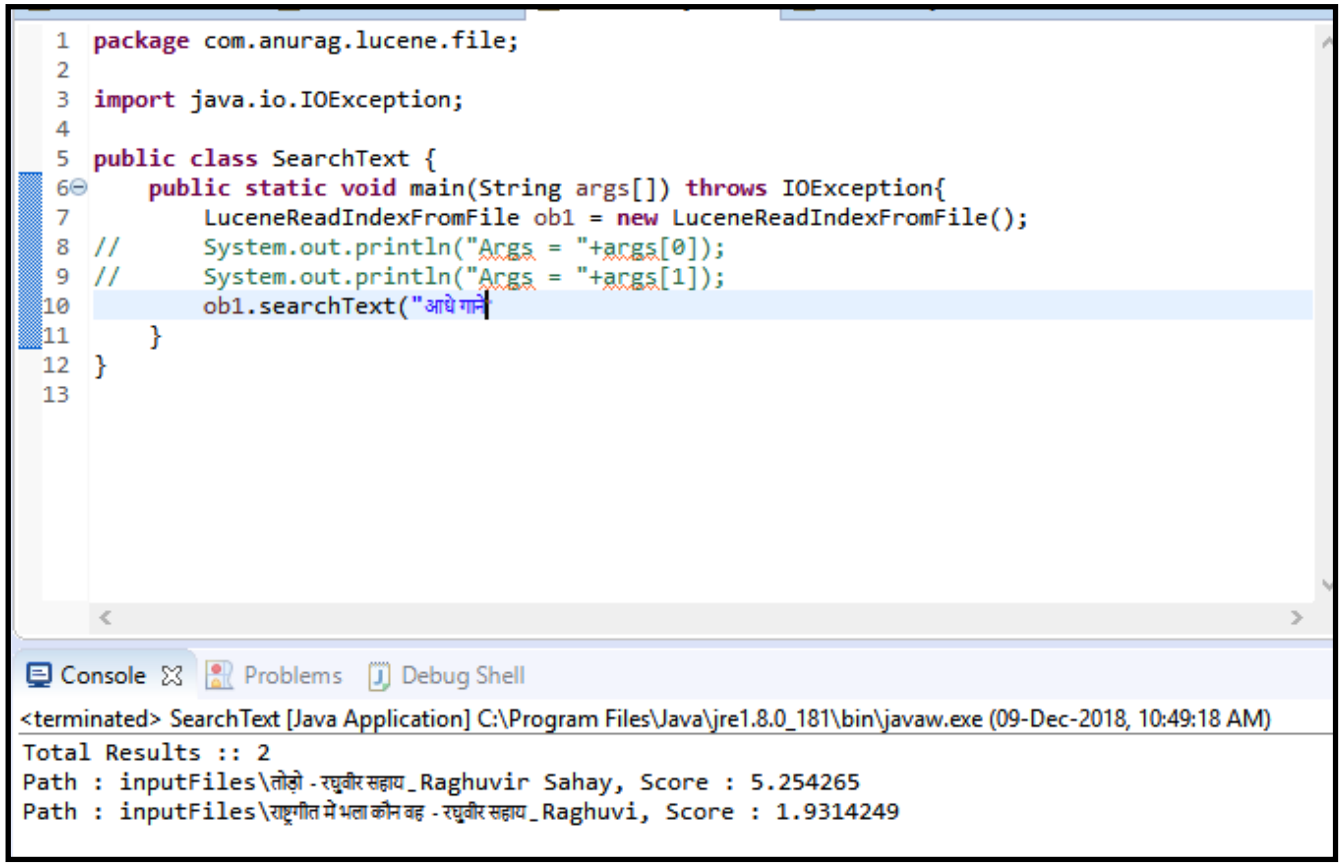
Without Stemming



```
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("आधे गाने
    }
}
```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 10:48:49 AM)
Total Results :: 1
Path : inputFiles\तोड़ो - रघुवीर सहाय_Raghuvir Sahay, Score : 5.7000895
```

With Stemming

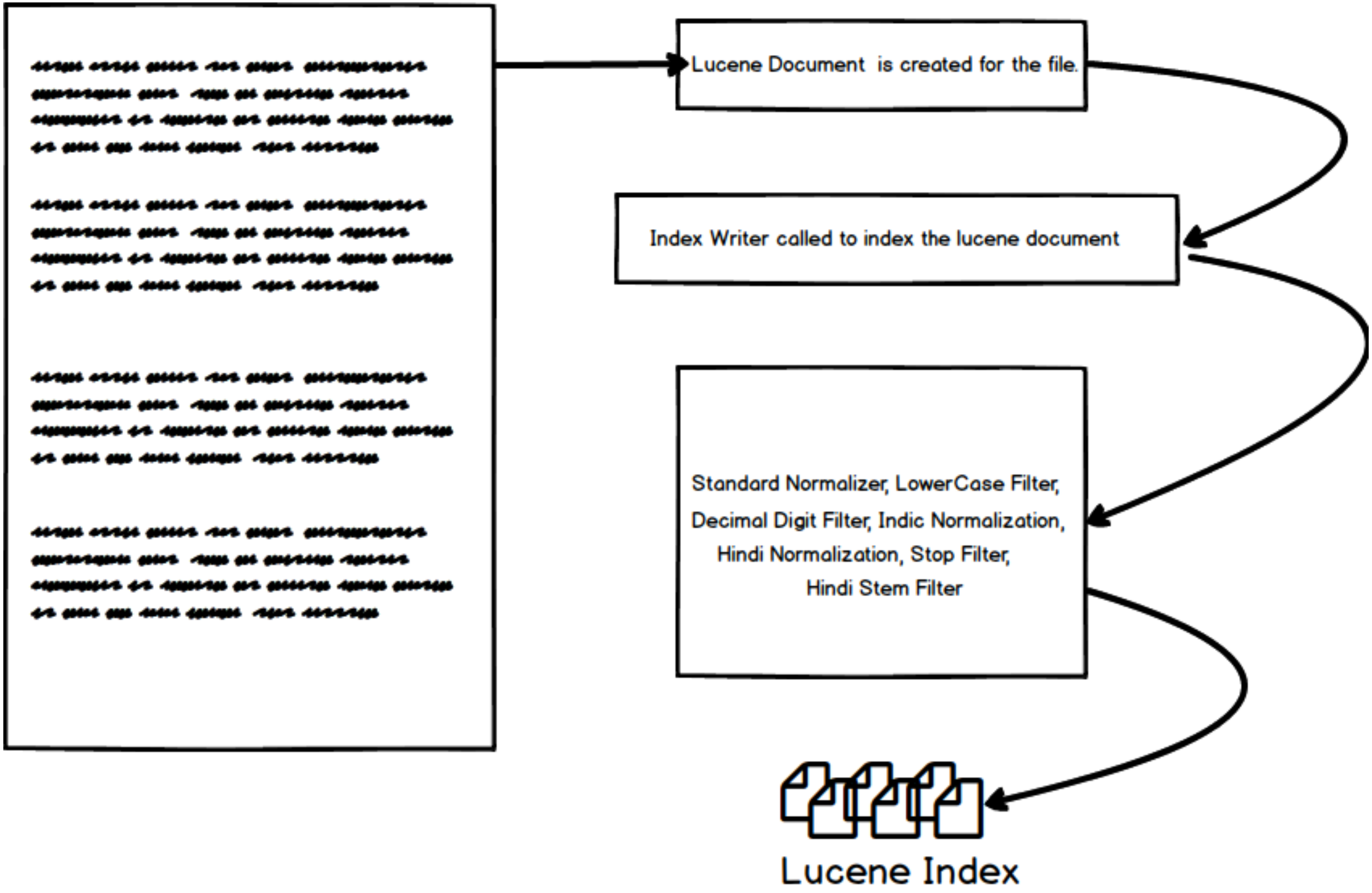


```
package com.anurag.lucene.file;

import java.io.IOException;

public class SearchText {
    public static void main(String args[]) throws IOException{
        LuceneReadIndexFromFile ob1 = new LuceneReadIndexFromFile();
//      System.out.println("Args = "+args[0]);
//      System.out.println("Args = "+args[1]);
        ob1.searchText("आधे गाने
    }
}
```

```
<terminated> SearchText [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (09-Dec-2018, 10:49:18 AM)
Total Results :: 2
Path : inputFiles\तोड़ो - रघुवीर सहाय_Raghuvir Sahay, Score : 5.254265
Path : inputFiles\राष्ट्रगीत में भला कौन वह - रघुवीर सहाय_Raghuvi, Score : 1.9314249
```

- **The score is more for result without stemming but we could retrieve 2 documents from the result with stemming.**
- **The query is "आधे गाने".**
- **After query processing, we will search for documents containing आध and गान.**
- **We will find आधे and गाने in the index without stemming as index has आधे and गाने are terms in the index so we get a high score document.**

Summary



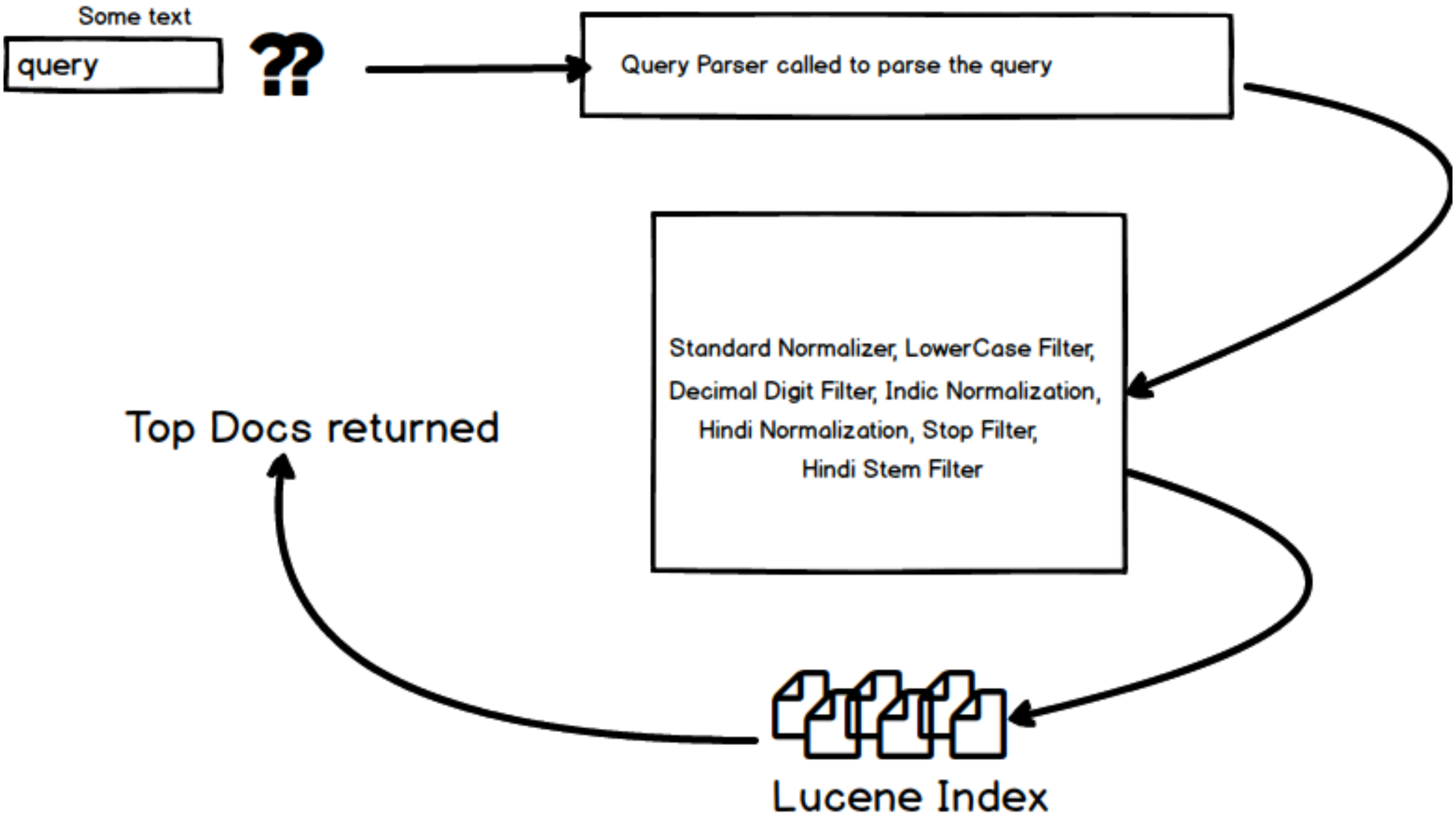
Lucene Document is created for the file.

Index Writer called to index the lucene document

Standard Normalizer, LowerCase Filter,
Decimal Digit Filter, Indic Normalization,
Hindi Normalization, Stop Filter,
Hindi Stem Filter

Lucene Index

Summary



Some text

query

??

Query Parser called to parse the query

Standard Normalizer, LowerCase Filter, Decimal Digit Filter, Indic Normalization, Hindi Normalization, Stop Filter, Hindi Stem Filter

Lucene Index

Top Docs returned

Thank You

Anubhav Ujjawal S20160010005
Anurag Gupta S20160010006
Garvit Kataria S20160010028