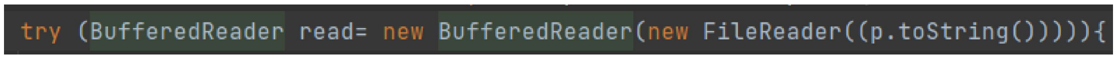
The index class uses the lucene API to find the target dataset based on the input path and to create the index file for these datasets so that subsequent search functions can be performed. There are three methods in this class. First is the indexDoc method, which creates an index file for a single data file. The second is the indexDocs method, which calls the indexDoc method with the input dataset and path as parameters to generate a series of index files. Finally, there is the createIndex method, which gets the path of the dataset and the path where the index file will be stored and then calls indexDocs with it as an argument to generate the index file.

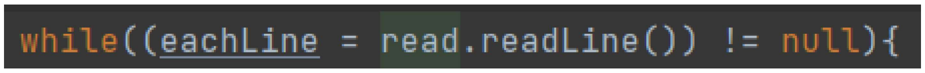
First create a single indexed document using the indexDoc method.

Inside the indexDoc method first use the BufferedReader object to read the input stream of data.



```
try (BufferedReader read= new BufferedReader(new FileReader((p.toString())))){
```

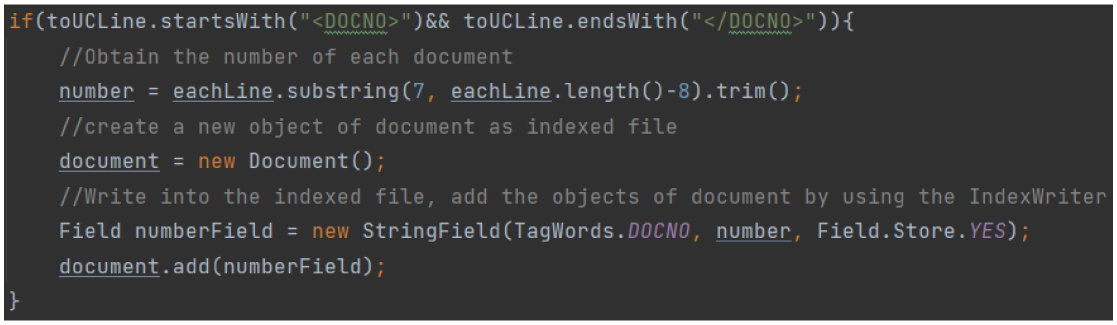
Then use a loop to iterate over each line in the input stream



```
while((eachLine = read.readLine()) != null){
```

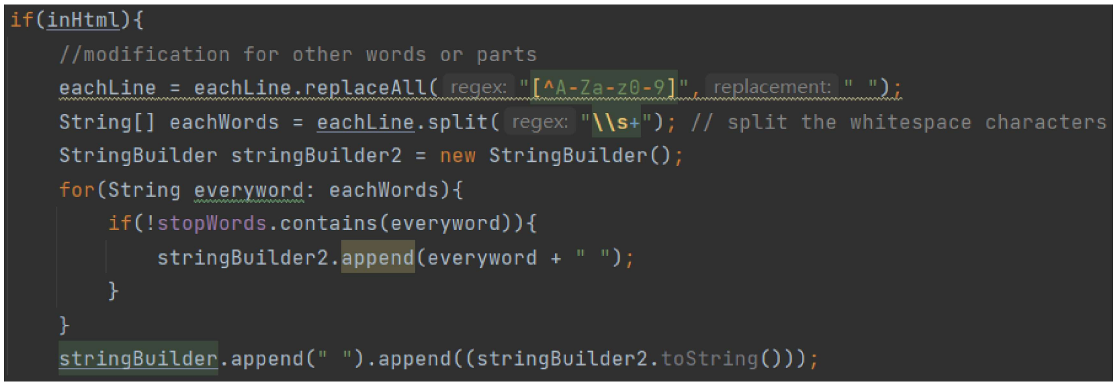
At each line readed, the program classifies the read string into three categories, for which there are different methods of processing.

The first category is lines that begin and end with the tags "<DOCNO>" and "</DOCNO>". When lines with these tags are read, the document number of the file is obtained and written to the index file. And only when a file number is read, will the index file be created and then the information in other lines in the input stream can be read.



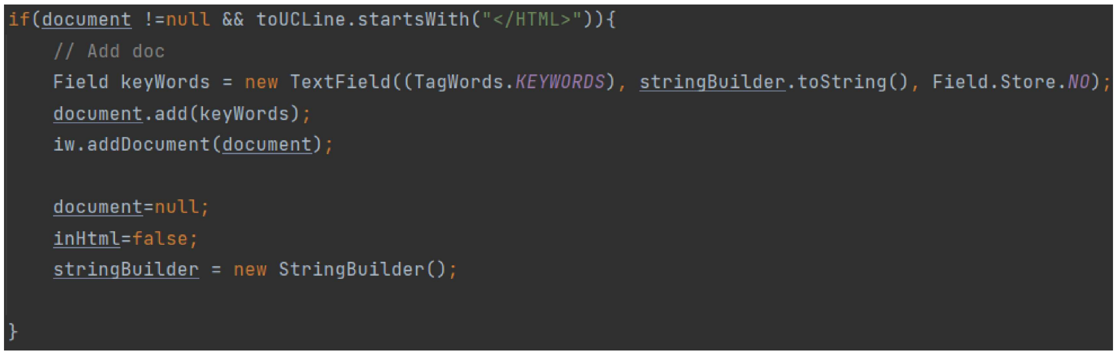
```
if(toUCLine.startsWith("<DOCNO>")&& toUCLine.endsWith("</DOCNO>")){
    //Obtain the number of each document
    number = eachLine.substring(7, eachLine.length()-8).trim();
    //create a new object of document as indexed file
    document = new Document();
    //Write into the indexed file, add the objects of document by using the IndexWriter
    Field numberField = new StringField(TagWords.DOCNO, number, Field.Store.YES);
    document.add(numberField);
}
```

The second category is lines that start with the "<HTML>" tag. When a line with this tag is read, it replaces each substring of this string that matches the given regular expression with the given replacement. Then convert the replaced substring of this line to a string with a StringBuilder object. After converting this line to a string, use the StringBuilder object again to save the strings of these lines together.



```
if(inHtml){
    //modification for other words or parts
    eachLine = eachLine.replaceAll( regex: "[^A-Za-z0-9]", replacement: " ");
    String[] eachWords = eachLine.split( regex: "\\s+"); // split the whitespace characters
    StringBuilder stringBuilder2 = new StringBuilder();
    for(String everyword: eachWords){
        if(!stopWords.contains(everyword)){
            stringBuilder2.append(everyword + " ");
        }
    }
    stringBuilder.append(" ").append((stringBuilder2.toString()));
```

The last category is the line starting with the "</HTML>" tag. When a line with this tag is read, the program will use the TextField object in the lucene API to edit each line of string stored in the stringBuilder into an indexed and tagged field line by line. Then write the edited fields to the index file. Finally, stop the next reading.

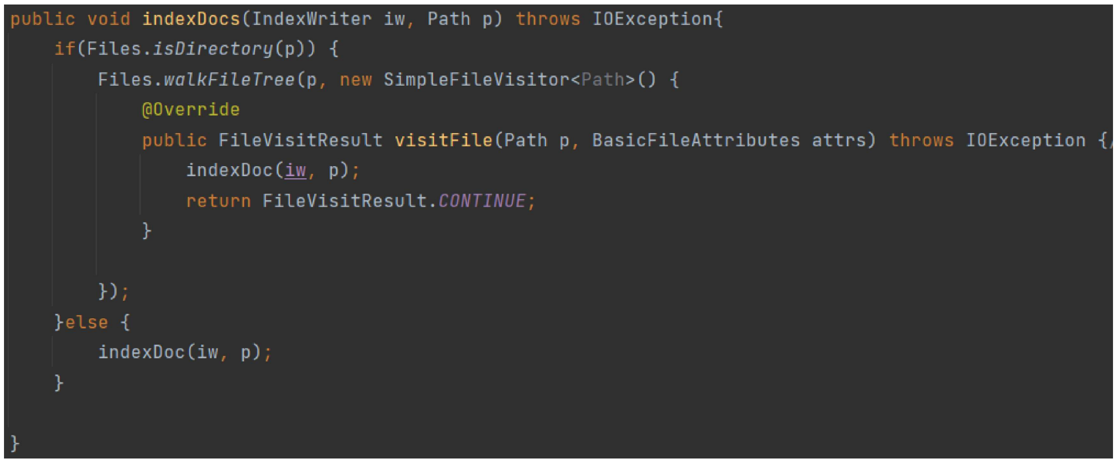


```java
if(document !=null && toUCLine.startsWith("</HTML>")){
    // Add doc
    Field keyWords = new TextField((TagWords.KEYWORDS), stringBuilder.toString(), Field.Store.NO);
    document.add(keyWords);
    iw.addDocument(document);

    document=null;
    inHtml=false;
    stringBuilder = new StringBuilder();

}
```

In order to generate a series of index documents at once we will use the indexDocs method. This method will call the indexDoc method with the obtained dataset and dataset path as parameters to generate index files one by one.

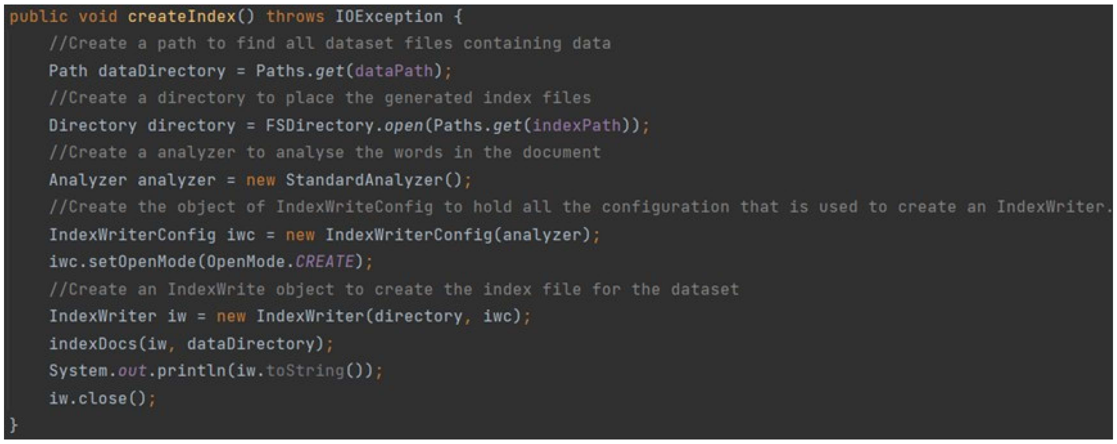


```java
public void indexDocs(IndexWriter iw, Path p) throws IOException{
    if(Files.isDirectory(p)) {
        Files.walkFileTree(p, new SimpleFileVisitor<Path>() {
            @Override
            public FileVisitResult visitFile(Path p, BasicFileAttributes attrs) throws IOException {
                indexDoc(iw, p);
                return FileVisitResult.CONTINUE;
            }

        });
    }else {
        indexDoc(iw, p);
    }

}
```

In order to generate the index file and input it to the specified path we will use the createIndex method. This method will read the path to the location of the dataset according to the input parameters, and generate an IndexWriter object that stores the path of the folder where the index file is stored. Finally, call the indexDocs method with the IndexWriter and the dataset path as parameters to generate the index file.



```java
public void createIndex() throws IOException {
    //Create a path to find all dataset files containing data
    Path dataDirectory = Paths.get(dataPath);
    //Create a directory to place the generated index files
    Directory directory = FSDirectory.open(Paths.get(indexPath));
    //Create a analyzer to analyse the words in the document
    Analyzer analyzer = new StandardAnalyzer();
    //Create the object of IndexWriteConfig to hold all the configuration that is used to create an IndexWriter.
    IndexWriterConfig iwc = new IndexWriterConfig(analyzer);
    iwc.setOpenMode(OpenMode.CREATE);
    //Create an IndexWrite object to create the index file for the dataset
    IndexWriter iw = new IndexWriter(directory, iwc);
    indexDocs(iw, dataDirectory);
    System.out.println(iw.toString());
    iw.close();
}
```