```
/**
* This is my code! Its goal is to read a file and convert it into a
* document object
* CS 312 - Assignment 9
* @author Mari Sisco
*/

import java.nio.file.Paths;
import java.nio.file.Path;
import java.nio.file.Files;
import java.io.BufferedReader;
import java.util.Scanner;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;

public class Document implements Iterable<String>
{
  protected String name;
  protected String content;
  protected static final String DELIMITERPATTERN = "[^a-zA-Z]+";

  /** Creates a Document based on the contents of a file
   * @param pathString, a String containing the path to reach the file
   */
  public Document(String pathString)
  {
    readContent(pathString);

    Path p = Paths.get(pathString);
    String filename = p.getFileName().toString();
    this.name = filename;
  }

  /** Reads content of a file and converts it into a String
   * @param filename, String of the path to file
   * complexity = O(n)
   */
  public void readContent(String filename)
  {
    String asRead = "";
    try
    {
      BufferedReader br;
      br = new BufferedReader(new FileReader(filename));
      asRead = new Scanner(br).useDelimiter("\\A").next();
      this.content = asRead;
      br.close();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
    this.content = asRead;
  }

  /** Iterates through the content of the Document using a specific delimiter pa
ttern
   * @return An Iterator<String> object
   * complexity = O(n)
   */
  public Iterator<String> iterator()
  {
    return new Scanner(content).useDelimiter(DELIMITERPATTERN);
  }

  /** Return the name of the document
   * @return A String, the name of the Document
   * complexity = O(1)
```

```
   */
  public String documentName()
  {
    return this.name;
  }

  /** Creates a human readable representation of the Document
   * @return A String
   */
  public String toString()
  {
    return name + ", content: \n " + content;
  }

}
```

```java
/**
* This is my code!  Its goal is to create a set of stoplist words from a file
* CS 312 - Assignment 9
* @author Mari Sisco
*/

import java.util.HashMap;
import java.util.Set;
import java.util.HashSet;
import java.nio.file.Paths;
import java.nio.file.Path;
import java.nio.file.Files;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;

public class Stoplist
{
  protected Set<String> stoplist;

  /** Creates a Stoplist, a set of stoplist words from a file
   * @param path, a String containing the path to the file
   */
  public Stoplist(String path)
  {
    try
    {
      List<String> asRead = Files.readAllLines(Paths.get(path));
      stoplist = new HashSet<>(asRead);
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  /** Returns wether a word is a stoplist word
   * @param w, a String containing the word to find in set
   * @return A Boolean, true if the word is a stopword
   * complexity = O(1)
   */
  public boolean hasStopWord(String w)
  {
    return stoplist.contains(w);
  }
}
```

```java
/**
 * This is my code!  Its goal is to create a SearchEngine to find the documents
a word query
 * is associated with
 * CS 312 - Assignment 8
 * @author Mari Sisco
 */

import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import java.util.ArrayList;
import java.lang.NullPointerException;

public class SearchEngine
{
  protected HashMap<String, Set<Document>> invertedIndex;
  protected Stoplist stoplist;
  protected boolean display;

  /** Creates a SearchEngine, containing an invertedIndex hashmap
   * @param stoplist, Stoplist with all stopwords
   * @param display, a Boolean deciding the way to display
   */
  public SearchEngine(Stoplist stoplist, boolean display)
  {
    invertedIndex = new HashMap<>();
    this.stoplist = stoplist;
    this.display = display;
  }

  /** Builds the Hashmap, with a String key (the word) that is associated to a s
et of
   * documents it is found in
   * @param doc, a Document to be analysed and added to invertedIndex
   * complexity = O(n)
   */
  public void buildIndex(Document doc)
  {
    Set<String> words = makeClean(doc);
    for (String w: words)
    {
      if(invertedIndex.containsKey(w))
      {
        // key of w exists
    invertedIndex.get(w).add(doc);
      }
      else
      {
        Set<Document> documents = new HashSet<Document>();
        documents.add(doc);
        invertedIndex.put(w, documents);
      }
    }
  }

  /** Removes all stop words and punctuation from the content of a document
   * @param doc, Document to be cleaned
   * @return a Set of Strings containing all the words in the document
   * complexity = O(n)
   */
  public Set<String> makeClean(Document doc)
  {
    String content = doc.iterator().toString();
    Set<String> words = new HashSet<String>();
    for(String s: doc)
    {
      if (!stoplist.hasStopWord(s) && !words.contains(s))
      {
        words.add(s);
      }
    }
    return words;
  }

  /** Finds query word in Hashmap
   * @param query, a String containing word to be found
   * complexity = O(1)
   */
  public void findQuery(String query)
  {
    Set<Document> querydocs = invertedIndex.get(query);
    try
    {
      display(query, querydocs);
    }
    catch(NullPointerException e)
    {
      System.out.print("Word not found in any \n");
    }
  }

  /** Displays invertedIndex Hashmap
   * complexity = O(1)
   */
  public void displayInvertedIndex()
  {
    invertedIndex.entrySet().forEach(entry -> {
    System.out.println("-> " + entry.getKey() + " FOUND IN: \n " + entry.getValue()
);
    });
  }

  /** Displays the documents query was found in
   * @param query, String to be found
   * @param querydocs, documents associated to the query
   * complexity = O(n)
   */
  public void display(String query, Set<Document> querydocs)
  {
    String result = "query '" + query + "' returned ";
    if(querydocs.isEmpty())
      result += "null";
    else
    {
      for (Document d: querydocs)
      {
        if (this.display == true)
          result += "\n" + d;
    else
          result += d.documentName() + " ";
      }
    }

    System.out.println(result);
    System.out.println("--- found in " + (querydocs == null ? 0 : querydocs.size())
                      + " documents");

  }

  /** Finds all documents that query words share
   * @param query, a String array with all the query words
   * @param querytofind, String query
   * complexity = O(n)
   */
  public void findMultiWord(String [] query, String querytofind)
  {
    ArrayList <String> words = makeClean(query);
```

```
    if (words.isEmpty())
    {
      System.out.println("No documents in common\n");
      return;
    }

    Set<Document> intersection = invertedIndex.get(words.get(0));

    for(int i = 1; i < words.size(); i++)
    {
      Set<Document> worddocs = invertedIndex.get(words.get(i));
      try
      {
        intersection.retainAll(worddocs);
      }
      catch(NullPointerException e)
      {
        System.out.print("No documents in common \n");
    return;
      }
    }
    try
    {
      display(querytofind, intersection);
    }
    catch(NullPointerException e)
    {
      System.out.print("No document contains all words\n");
    }
  }

  /** Cleans of stoplist words the string array with all the query words
   * @return An ArrayList of strings with the cleaned query words
   * complexity = O(n)
   */
  public ArrayList<String> makeClean(String [] query)
  {
    ArrayList<String> words = new ArrayList<String>();
    for(String s: query)
    {
      if (!stoplist.hasStopWord(s) && !words.contains(s))
      {
        words.add(s);
      }
    }
    return words;
  }
}
```

```java
/**
* This is my code!  Its goal is to read command line arguments and execute searc
h
* engine functions
* CS 312 - Assignment 9
* @author Mari Sisco
*/

import java.util.Scanner;
import java.util.Arrays;

public class CLI
{
  private String [] args;
  protected Boolean display;

  /** Creates a CLI object to read and interpret command line arguments
   * @param a, a String array containing command line arguments
   */
  public CLI(String [] a)
  {
    this.args = a;
    this.display = false;

    parse();

  }

  /** Prints a usage message
   */
  private void usage()
  {
    System.out.println("Usage: [-d] <Path to stoplist> <Path to document(s)>");
  }

  /** Interprets String arguments, parses arguments as parameters to specific fu
nctions
   */
  public void parse()
  {
    if (args.length == 0)
    {
      usage();
      return;
    }

    int i = 0;

    if ("-d".equals(args[i]))
    {
      this.display = true;
      i++;
    }

    if (args.length > i)
    {
      Stoplist sl = new Stoplist(args[i]);
      i++;
      SearchEngine se = new SearchEngine(sl, display);
      for( int j = i; j < args.length; j++)
      {
        Document doc = new Document(args[j]);
    se.buildIndex(doc);
      }

      Scanner sc = new Scanner(System.in);
      long startTime = System.currentTimeMillis();
      while (sc.hasNextLine())
      {
```

```java
    String query = sc.nextLine();
        if (query.equals("@@debug"))
    {
      se.displayInvertedIndex();
    }
    else
    {
      String [] words = query.split("\\W+");
      //System.out.println(Arrays.toString(words));
      if(words.length == 1)
            se.findQuery(query);
      else
            se.findMultiWord(words, query);
    }
      }
      long stopTime = System.currentTimeMillis();
      long elapsedTime = stopTime - startTime;
      System.out.println("@@ processing took " + elapsedTime + "ms");
    }
  }

  public static void main (String [] args)
  {
    CLI cli = new CLI(args);
    //cli.parse();

  }
}
```