```
/**
* This is my code! Its goal is to take command line arguments,
* create an inverted index,
* and then execute queries for words in that index
* CS 312 - Assignment 9
* @author Andrew Fallon
* @version 1.0 12/10/2018
*/

import java.io.*;
import java.util.*;

/**
 * Takes in command line arguments and with them creates an inverted index,
 * and also runs queries on the inverted index from standard in
 */
public class CLI
{
    /**
     * String value that would be entered from standard in
     * to print the entire inverted index
     * For debugging purpose
     */
    private static final String DEBUG = "@@debug";

    /**
     * Main method of the program,
     * creates an inverted index and conducts queries on it
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException
    {
        //Read in command line arguments
        String usage = "Usage: java CLI [-d] stopList Documents";
        if(args.length < 2 || (args.length < 3 && args[0].equals("-d")))
        {
            System.out.println("Improper input: " + usage);
            System.exit(-1);
        }

        boolean displayText = false;
        int start = 0;

        if("-d".equals(args[0]))
        {
            displayText = true;
            start = 1;
        }
        long startTime = System.currentTimeMillis();

        //manipulate args to create an index with the stop list and doc names
        List<String> argList = Arrays.asList(args);
        argList = argList.subList(start, argList.size());
        InvertedIndex index = new InvertedIndex(argList);

        //find how long building the index took
        long buildStop = System.currentTimeMillis();
        long buildTime = buildStop-startTime;
        System.out.println("@@ build time: " + buildTime + "ms");
```



```
        try
        {
            BufferedReader br = new BufferedReader
                                (new InputStreamReader(System.in));
            for(String line = br.readLine(); line != null; line=br.readLine())
            {
                if(DEBUG.equals(line))
                    index.printIndex();
                else
                {
                    //perform the search
                    Set<Document> docSet = index.query(line);

                    //print results of the search
                    System.out.println("---found in " +
                            (docSet == null ? 0 : docSet.size())+ " documents");
                    if(!docSet.isEmpty())
                    {
                        for(Document doc: docSet)
                            System.out.print(doc + " ");
                        System.out.println("");
                    }
                    else
                        System.out.println("null");

                    if(displayText)
                        for(Document doc: docSet)
                            doc.printFullDoc();
                }
            }
            br.close();
        }
        catch(Exception ex)
        {
            System.out.println("Received a " + ex);
            ex.printStackTrace();
        }

        //find how long it took to execute the queries
        long stopTime = System.currentTimeMillis();
        long queryTime = stopTime - buildStop;
        System.out.println("@@query time: " + queryTime + "ms");
    }
}
```

```java
/**
* This is my code! Its goal is to hold the contents of a file
* and manipulate that contents
* CS 312 - Assignment 9
* @author Andrew Fallon
* @version 1.0 12/10/2018
*/
import java.io.*;
import java.util.*;
import java.nio.file.Paths;
import java.nio.file.Path;

/**
 * Class that holds the contents of a documents
 */
public class Document implements Iterable<String>
{
    /**
     * File name of this document
     */
    protected String docName;

    /**
     * Entire String of the text of the document,
     * for -d output
     */
    protected String fullDocString;

    /**
     * Constructs a document, instantiates both instance variables
     * Expected complexity: O(1), constant time
     *
     * @param fileName- The name of the file to read from
     * @throws FileNotFoundException
     * @throws IOException
     * @throws NullPointerException
     */
    public Document(String fileName) throws FileNotFoundException,
                                IOException, NullPointerException
    {
        createDocString(fileName);
        Path p = Paths.get(fileName);
        docName = p.getFileName().toString();
    }

    /**
     * Instantiates fullDocString with the contents of a file
     * Expected complexity: O(1), constant time
     *
     * @param fileName - The name of the file to read from
     * @throws IOException
     * @throws FileNotFoundException
     * @throws NullPointerException
     */
    private void createDocString(String fileName) throws IOException,
                        FileNotFoundException, NullPointerException
    {
        try
        {
            BufferedReader br = new BufferedReader(new FileReader(fileName));
            fullDocString = new Scanner(br).useDelimiter("\\A").next();
            br.close();
        }
        catch(Exception ex)
        {
            System.out.println("Received a " + ex);
            ex.printStackTrace();
        }
    }
```



```java
    /**
     * Returns an Iterator of every word in the document
     * Implementation of a method declared in the Iterable interface
     * Expected complexity: O(1), constant time
     *
     * @return - An iterator over each word in the document
     */
    public Iterator<String> iterator()
    {
        return new Scanner(fullDocString).useDelimiter("[^a-zA-Z]+");
    }

    /**
     * Returns the document name, without the path
     * Expected complexity: O(1), constant time
     */
    @Override
    public String toString()
    {
        return docName;
    }

    /**
     * Prints the full text of the document to standard in
     * Expected complexity: O(1), constant time
     */
    public void printFullDoc()
    {
        System.out.println(fullDocString);
    }
}
```

```
/**
* This is my code! Its goal is to hold an inverted index of Documents,
* and be able to answer a query of word(s) with what documents they're in
* CS 312 - Assignment 9
* @author Andrew Fallon
* @version 1.0 12/10/2018
*/

import java.io.*;
import java.util.*;

/**
 * The main body of the search engine,
 * holds the inverted index and runs queries on it
 */
public class InvertedIndex
{
    /**
     * A set of common words that queries will ignore
     */
    protected Set<String> stopList;
    /**
     * Map that holds the actual inverted index,
     * with each word as a key and its value
     * being a Set of each Document it is in
     */
    protected Map<String, Set<Document>> wordIndex;

    /**
     * constructor, creates the stop list
     * and instantiates the instance variable,
     * reads the files, makes them into documents,
     * and fills out the inverted index with those documents
     * Expected complexity: O(n^2)
     *
     * @param argList - A list containing the stop list file name
     * and each document file name
     * @throws IOException
     */
    public InvertedIndex(List<String> argList) throws IOException
    {
        buildStopList(argList.get(0));
        Set<Document> docSet =
                createDocuments(argList.subList(1, argList.size()));
        buildIndex(docSet);
    }

    /**
     * Simple method to name searching through the stop list for a word
     * Expected complexity: O(1), constant time, searching through a HashSet
     *
     * @param oneWord - The word you are searching through the stop list for
     * @return - Whether or not the word is in the stop list
     */
    private boolean inStopList(String oneWord)
    {
        return stopList.contains(oneWord);
    }

    /**
     * Prints to standard out each key and its value in the inverted index
     * Expected complexity: O(n), n being the number of keys in wordIndex
     */
    public void printIndex()
    {
        System.out.println(wordIndex.entrySet());
    }
```



```
    /**
     * Reads in the stop list file and stores each word in stopList
     * Expected complexity: O(n), n being the number of words in the document
     *
     * @param stopListName - The name of the stop list file
     * @throws FileNotFoundException
     */
    private void buildStopList(String stopListName) throws FileNotFoundException
    {
        stopList = new HashSet<String>();
        try
        {
            File file = new File(stopListName);
            Scanner scan = new Scanner(file).useDelimiter("[^a-zA-Z]+");
            while(scan.hasNext())
            {
                String currentWord = scan.next();
                stopList.add(currentWord);
            }
            scan.close();
        }
        catch(FileNotFoundException fnf)
        {
            System.out.println("Stop list file name does not match an existing file");
        }
    }

    /**
     * fills out a HashMap with keys of each word
     * that appears in any of the documents,
     * provided it's not in the stop list,
     * and stores a set of each of the documents
     * the word appears in as the value of each word
     * Expected complexity: O(n^2), the n values being
     * the number of documents and the number of words in those documents
     *
     * @param docSet a set of all of the entered documents
     */
    private void buildIndex(Set<Document> docSet)
    {
        wordIndex = new HashMap<String, Set<Document>>();
        for(Document currentDoc: docSet)
        {
            for(String word: currentDoc)
            {
                if(!inStopList(word))
                {
                    //create a new key
                    if(!wordIndex.containsKey(word))
                    {
                        Set<Document> tempSet = new HashSet<Document>();
                        tempSet.add(currentDoc);
                        wordIndex.put(word, tempSet);
                    }

                    //add the value of the current document to the index
                    Set<Document> valueSet = new HashSet<Document>();
                    valueSet = wordIndex.get(word);
                    valueSet.add(currentDoc);
                    wordIndex.replace(word, valueSet);
                }
            }
        }
    }
```

```
    /**
     * Uses each file name in fileNames to create a Document
     * with the contents of the file,
     * and returns a set of those Documents
     * Expected complexity: O(n), n being the number of file names
     *
     * @param fileNames - the Strings of each of the files to read and store
     * @return - A set of each of the Documents
     * @throws IOException
     */
    private Set<Document> createDocuments(List<String> fileNames)
        throws IOException
    {
        Set<Document> docSet = new HashSet<Document>();
        for(int i=0; i<fileNames.size();i++)
        {
            docSet.add(new Document(fileNames.get(i)));
        }
        return docSet;
    }

    /**
     * Finds a set of documents that contains all of the entered words
     * Expected complexity: O(1), constant time, searching through a HashMap
     *
     * @param line - A String of each word to search for, separated by spaces
     * @return - A set of the Documents that contain all of the entered words
     * @throws IOException
     */
    public Set<Document> query(String line) throws IOException
    {
        Set<String> wordSet = new HashSet<String>();
        Set<Document> finalDocSet = new HashSet<Document>();

        StringTokenizer tokenizer = new StringTokenizer(line);
        while(tokenizer.hasMoreTokens())
        {
            String word = tokenizer.nextToken();
            wordSet.add(word);
        }

        boolean setInstantiated = false;
        for(String word: wordSet)
        {
            Set<Document> newDocs;
            newDocs = (wordIndex.get(word) == null) ?
                    new HashSet<Document>() : wordIndex.get(word);
            if(!inStopList(word))
            {
                //fill out the final set with the first word's documents
                if(!setInstantiated)
                {
                    finalDocSet = newDocs;
                    setInstantiated = true;
                }

                //find discrepancies in document sets
                Set<Document> removeSet = new HashSet<Document>();
                for(Document doc: finalDocSet)
                    if(!newDocs.contains(doc))
                        removeSet.add(doc);
                for(Document doc:removeSet)
                    finalDocSet.remove(doc);
            }
        }
        return finalDocSet;
    }
}
```