Assignment 1

Problem Statement: Implement Conflation Algorithm using File Handling

```
package First;
public class PorterStemmer {
        private char[] b;
        private int i,i_end,k;
        private static final int INC = 50;
        public PorterStemmer()
        {
                b = new char[INC];
                i=0;
                i_end=0;
        }
        public void setCurrent(String word)
        {
                i=word.length();
                b=word.toCharArray();
        }
        public void stem()
        {
                k=i-1;
                if(k>1)
                {
                        if(ends("ing")) k-=3;
                        else if(ends("ed")) k-=2;
                        else if(ends("es")) k-=2;
                        else if(ends("s") && k>0) k-=1;
                }
                i_end=k+1;
        }
        private boolean ends(String s)
        {
                int length = s.length();
                if(length>i)
                {
                        return false;
                }
                for(int c=0;c<length;c++)
                {
                        if(b[i-length+c]!=s.charAt(c))
                        {
                                return false;
                        }
                }
                return true;
        }

        public String getCurrent()
        {
```

```
        return new String(b,0,i_end);
    }
}
```

Simple Conflation:

```
package First;
import java.util.*;
public class SimpleConflation {
    private static final Set<String>STOP_WORDS = new HashSet<>(Arrays.asList("a",
                "an","the","and","or","but","is","are","was","were","in","on","that"
,
                "by","of","to"));
    private static final Map<String,String>LEMMA_DICT = new HashMap<>();
    static
    {
        LEMMA_DICT.put("methods","method");
        LEMMA_DICT.put("computable","compute");
        LEMMA_DICT.put("processing","process");
        LEMMA_DICT.put("systems","system");
        LEMMA_DICT.put("words","word");
        LEMMA_DICT.put("documents","document");
        LEMMA_DICT.put("generates","generate");
        LEMMA_DICT.put("means","mean");
    }
    public static void main(String[] args)
    {
        String text = "Ultimately one would like to develope a text processing system which
by means of computable methods with the minimum of human intervention will generate from the
input text a document representative adequate for use in an automatic information retrieval
system";

        String[] tokens = text.toLowerCase().replaceAll("[^a-z]"," ").split("\\s+");

        //Remove stopwords
        List<String> filteredTokens = new ArrayList<>();
        for(String token : tokens)
        {
            if(!STOP_WORDS.contains(token) && token.length()>0)
            {
                filteredTokens.add(token);
            }
        }
        //Apply Stemming
        PorterStemmer stemmer = new PorterStemmer();
        List<String> stemmedTokens = new ArrayList<>();
        for(String token : filteredTokens)
        {
            stemmer.setCurrent(token);
            stemmer.stem();
            stemmedTokens.add(stemmer.getCurrent());
        }
```

```
		//Apply Lemmatization
		List<String> lemmatizedTokens = new ArrayList<>();
		for(String token : filteredTokens)
		{
			String lemma = LEMMA_DICT.getOrDefault(token,token);

			lemmatizedTokens.add(lemma);
		}
		System.out.println("Original Tokens : ");
		System.out.println(Arrays.toString(tokens));

		System.out.println("\nFiltered tokens (without stopwords) : ");
		System.out.println(filteredTokens);

		System.out.println("\nLemmatized tokens (simple dictionary) : ");
		System.out.println(lemmatizedTokens);
	}
}
```

Output:

```
Original Tokens :
[ultimately, one, would, like, to, develope, a, text, processing, system, which, by, means, of,
computable, methods, with, the, minimum, of, human, intervention, will, generate, from, the, input,
text, a, document, representative, adequate, for, use, in, an, automatic, information, retrieval,
system]

Filtered tokens (without stopwords) :
[ultimately, one, would, like, develope, text, processing, system, which, means, computable,
methods, minimum, human, intervention, will, generate, from, input, text, document,
representative, adequate, for, use, automatic, information, retrieval, system]

Stemmed tokens :
[ultimately, one, would, like, develope, text, process, system, which, mean, computable, method,
minimum, human, intervention, will, generate, from, input, text, document, representative,
adequate, for, use, automatic, information, retrieval, system]

Lemmatized tokens (simple dictionary) :
[ultimately, one, would, like, develope, text, process, system, which, mean, compute, method,
minimum, human, intervention, will, generate, from, input, text, document, representative,
adequate, for, use, automatic, information, retrieval, system]
```