Mapper File

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;

public class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{
   IntWritable val = new IntWritable(1);
   private Text word = new Text();
   private Text fWord = new Text();
   Text chapter = new Text("CHAPTER");
   int chapterNo = 10;
   // Initializing stop words list array.
   String wordsSet[]=
{"a","about","above","all","am","an","and","any","are","arent","as","at","be","because","been","before","being","below
","between","both","but","by","cant","cannot","could","couldnt","did","didnt","do","does","doesnt","doing","dont","eac
h","few","for","from","further","had","hadnt","has","hasnt","have","havent","having","he","hed","hell","hes","her","her
e","heres","hers","herself","him","himself","his","how","hows","i","id","ill","im","ive","if","in","into","is","isnt","it","it
s","its","itself","lets","me","more","most","mustnt","my","myself","no","nor","not","of","off","on","once","only","or","
other","ought","our","ours
ourselves","out","over","own","shant","she","shed","shell","shes","should","shouldnt","so","some","such","than","that",
"thats","the","their","theirs","them","themselves","then","there","theres","these","they","theyd","theyll","theyre","theyv
e","this","those","through","to","too","under","until","up","very","was","wasnt","we","wed","well","were","weve","wer
e","werent","what","whats","when","whens","where","wheres","which","while","who","whos","whom","why","whys","
will","with","wont","would","wouldnt","you","youd","youll","youre","youve","your","yours","yourself","yourselve"};
   // Puting those words in Hash set.
   HashSet<String> stopWords = new HashSet<String>(Arrays.asList(wordsSet));
   public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {

     String line = value.toString();
     // Removing punctuation and space with empty.
     line = line.replaceAll("[^a-zA-Z ]", "");
     // Separating each work using tokenizer.
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
       String newWord = itr.nextToken();

         if(!newWord.equals("CHAPTER")){
            // Converting all word to lowercase to ease the counting job
            // except capital CHAPTER as it represent new chapter.
           newWord = newWord.toLowerCase();
         }

         word.set(newWord);

            // Checking and removing stop words from map.
               if(!stopWords.contains(word.toString())){
```

```
                if(word.equals(chapter) ){
                        chapterNo++;
                    }
                    if(chapterNo>10){
                val = new IntWritable(1);
                IntWritable chapNumber = new IntWritable(chapterNo);
                if(!word.equals(chapter)){
                    // Adding chapter number for counting words in each chapter alone.
                    fWord = new Text(word + chapNumber.toString());
                }
                else
                    fWord = new Text(word);
                context.write(fWord, val);
                   }
            }
        }
     }
}
```

Reducer file

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;

 public class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {

  private IntWritable result = new IntWritable();
  HashMap<String, Integer> map = new HashMap();
  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    // Iterating the keys and calculating the number of occurance of each words in lines.
    for (IntWritable val : values) {
      sum++;
    }
    result.set(sum);

    // Counting and putting the words in a separate Hashmap array for separate chapter.
    Integer wordCount = map.containsKey(key.toString()) ? map.get(key.toString()) : 0;
    wordCount += sum;
    map.put(key.toString(), wordCount);
  }

  // using run() for explicitly use last reducer to write output as expected.
  public void run(Context context) throws IOException, InterruptedException {
     setup(context);
     try {
       while (context.nextKey()) {
         reduce(context.getCurrentKey(), context.getValues(), context);
       }
     } finally {
                  Integer totalChap = map.get("CHAPTER");
                  HashMap<String, Integer> []chapterMap = new HashMap[totalChap];
                  HashMap<String, Integer> freqWords = new HashMap();
                  Integer mapIndex = 0;

                  // The following loop is to separate words into there according chapter as different index of map array.
                  for (Map.Entry<String, Integer> entry : map.entrySet()) {
                     String key = entry.getKey();
                     Integer value = entry.getValue();
                     String curWord = "";
                     String curKey = key.toString();
                     if(!curKey.equals("CHAPTER")){
                        curWord = curKey.substring(0, curKey.length()-2);
                        Integer curChap = Integer.parseInt(curKey.substring(curKey.length()-2));
                        mapIndex = curChap-11;
                           if(chapterMap[mapIndex]==null){
```

```
            chapterMap[mapIndex] = new HashMap();
          }
          chapterMap[mapIndex].put(curWord, value);
    }
}

// The following loop is to find out most frequent words in all chapters.
for(int i=0;i<1;i++){
       for (Map.Entry<String, Integer> entry : chapterMap[i].entrySet()) {
    String key = entry.getKey();
    Integer value = entry.getValue();
    int counter = 1;
    if(value>=5){
       counter = 1;
          for(int j=1;j<totalChap;j++){
             if (chapterMap[j].containsKey(key)) {
               //if(chapterMap[j].get(key)>=5)
               counter++;
             }
          }
    }

    if(counter==totalChap){
          for(int k=0;k<totalChap;k++){
             value = chapterMap[k].get(key);
           if(freqWords.get(key) == null){
              Integer newValue = value;
              freqWords.put(key, newValue);
           }
           else
           {
             Integer newValue = freqWords.get(key) + value;
             freqWords.put(key, newValue);
           }
          }
    }
       }
}
// Printing most frequent words in all chapters.
context.write(new Text("Most Frequent Word list with frequency >="), new IntWritable(5));
for(Map.Entry<String, Integer> entry : freqWords.entrySet()){
        String key = entry.getKey();
     Integer value = entry.getValue();
     Text finalWord = new Text(key);
     IntWritable finalCount = new IntWritable(value);
     context.write(finalWord, finalCount);
}
// Printing all words count from all chapters.
context.write(new Text("\nWords count for each chapter. Total Chapter="), new IntWritable(totalChap));
for(int i=0;i<totalChap;i++){
       context.write(new Text("\nCHAPTER:"), new IntWritable(i+1));
       for (Map.Entry<String, Integer> entry : chapterMap[i].entrySet()) {
    String key = entry.getKey();
    Integer value = entry.getValue();
```

```
                Text finalWord = new Text(key);
                IntWritable finalCount = new IntWritable(value);
                context.write(finalWord, finalCount);
                    }
                }
            }
        }
```