```
/* Edited 04.04.2017 08h00 */


                                        /******     MOVIE RATINGS REVIEW Design 2 - MapReduce    *****/

/* Import classes necessary for the movieReview job */
import java.io.*;
import java.io.BufferedReader; // Class which allows buffering of characters for efficient reading of characters
import java.io.FileReader;
import java.io.IOException;
import java.util.Collections;
import java.util.HashSet;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.Scanner;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;


/* The main movieReview class consisting of 'reviewMapper', 'reviewCombiner', 'reviewReducer', 'main' and 'run' methods*/
public class movieReview extends Configured implements Tool // A tool interface to manage custom arguments provided in the
command line (3 arguments are expected)
{
    /*  Data is read from the file into the reviewMapper and emitted to the context as ( Key : Value ) -> ( "reviewRating" :
    "reviewWord,1" ) -> ( "9" , "Amazing,1" ) */
    public static class reviewMapper extends Mapper < Object, Text, Text, Text > // reviewMapper input <Object, Text> and
    output as <Text, Text>
    {

        /* Declare/initialise variables for the reviewMapper */
        private BufferedReader myReader; // Declare myReader to use BufferedReader
        private Set<String> excludedWords = new HashSet<String>(); // Declare an empty HashSet in which to load the files or
        punctuation characters to be exluded from counts


                /* Use setup method to incorporate a third argument to point to the file location of words to be excluded  */
                @Override
                protected void setup(Context context) throws IOException, InterruptedException
```

```
48                  {
49                      try
50                      {
51                          Path[] excludedWordFilePath = new Path[0]; // Declare excludedWordFilePath as a path
52                          excludedWordFilePath = context.getLocalCacheFiles(); // Load the cache file location from the context
                            into variable excludedWordFilePath
53                          if (excludedWordFilePath != null && excludedWordFilePath.length > 0) // Ensure path exists (not null)
                            and the path length is a positive integer
54                          {
55                              for (Path excludedWordPath : excludedWordFilePath)
56                              {
57                                  readStopWordFile(excludedWordPath);
58                              }
59                          }
60                      }
61                      catch (IOException e)
62                      {
63                          System.err.println("Error reading the location of the excluded_word file: " + e);
64                      }
65                  }
66
67                  /* Method to load file contents (with words to be excluded) into HashSet (excludedWords) */
68                  private void readStopWordFile(Path excludedWordPath)
69                  {
70
71                      try
72                      {
73                          myReader = new BufferedReader(new FileReader(excludedWordPath.toString())); // Read the file provided
                            containing the words to be excluded, line by line
74                          String excludedWordLine = null;
75                          while ((excludedWordLine = myReader.readLine()) != null) // Continue reading line by line until there
                            are no more lines in the file
76                          {
77                              String wordsTobeExcluded[] = excludedWordLine.split(","); // Split the line being read, by the
                                (,) seperating the words to be excluded
78                              for (String excludedWord: wordsTobeExcluded) // Iterate over the comma seperated exclusion words
                                and punctation characters
79                                  {
80                                      excludedWords.add(excludedWord); // For each excludedWord found, add to the HashSet
                                        excludedWords
81                                  }
82                          }
83                      }
84                      catch (IOException ioe)
85                      {
86                          System.err.println("Unable to read contents of the file containing words to be excluded '" +
                            excludedWordPath + "' : " + ioe.toString());
87                      }
88                  }
89
90                  /* This map function takes a line of text from the file at a time, splits into a rewiew string and rating
```

```
                string, and emits a key and a word assosciated with that key */
91              @Override
92              public void map(Object key, Text value, Context context) throws IOException, InterruptedException
93              {
94                  String[] ratingAndReview = value.toString().toLowerCase().split("\t");// Line is of the format
                    <"[Review]"  [Rating {1-9}]> --> split this line by the tab (\t) into a string with the reviewRating and
                    the movie rating
95
96                  // Get the current line and remove unexpected characters
97
98                  String rating = ratingAndReview[0]; // Save the review words from the first part of the split, into
                    variable 'rating'
99                  rating = rating.replaceAll("'", ""); // Remove single quotes from the rating string
100                 rating = rating.replaceAll("[^a-zA-Z]", " "); // Remove other unnecessary punctuation from the rating
                    string
101                 String reviewRating = ratingAndReview[1]; // Save the review rating from the second part of the split,
                    into variable 'reviewRating'
102                 Scanner itr = new Scanner(rating); // Use scanner to run over the words in the 'rating' string
103                 while (itr.hasNext()) // Iterate over words until all the review words have been taken into account
104                 {
105                     String reviewWord = itr.next(); // Call next word in the sequence
106                     if (!excludedWords.contains(reviewWord) &&  reviewWord.length() > 1 ) // Check if the word being
                        iterated over is not in the file containing the words to be excluded
107                         {
108                             String compoundValue = reviewWord + ",1"; // Concatenate the word and a "1" to form a string
                            "word,1" as the value emitted from the mapper in the key-value pair
109                             context.write(new Text(reviewRating), new Text(compoundValue) ); // Emit all the
                            reviewRatings as the keys and values as "word,1" (for example)
110                         }
111                 }
112             }
113     }
114
115     /* The combiner class aims to aggregate data locally from each reviewMapper before sending the data to the reviewReducer
        */
116     public static class reviewCombiner extends Reducer < Text, Text ,Text, Text > // reviewCombiner input <Text, Text> and
        output as <Text, Text>
117     {
118         /* Declare/initialise variables for the reviewCombiner */
119         private HashMap <String, Integer> localCombinerCount = new HashMap <String, Integer>(); // Declare
            'localCombinerCount' HashMap to store the word (for a given review) and highest frequency found, nine HashMaps are
            produced - one for each reviewRating
120
121             public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
122             {
123                 for (Text value : values) // Iterate over the values read in from the context
124                 {
125                     StringTokenizer itr = new StringTokenizer(value.toString(),","); // Seperate the "word,1"
126                     String aWord = itr.nextToken(); // Pull out the word and store as a string variable aWord
127                     Integer count = Integer.parseInt(itr.nextToken()); // Parse the string, pull out the "1" and convert
                        the "1" to an integer value
```

```
128
129                     if (localCombinerCount.containsKey(aWord))
130                     {
131                         localCombinerCount.put(aWord, localCombinerCount.get(aWord) + 1); // Add 1 to the count/frequency
                            found in the HashMap if the given word is in the HashMap
132                     }
133                     else
134                     {
135                         localCombinerCount.put(aWord, count); // If not in the HashMap localCombinerCount, add the word
                            and give it a count of 1
136                     }
137                 }
138
139                 for (Map.Entry<String, Integer> entry : localCombinerCount.entrySet()) // Iterate over <String,Integer>
                    values in the HashMap entries
140                 {
141                     String aKeyWord = entry.getKey(); // Saves the key from the HashMap (which is a word) as aKeyWord
                        variable
142                     Integer aKeyWordCount = entry.getValue(); // Saves the count assosciated with the key in the HashMap
                        as aKeyWordCount variable
143                     String compoundKeyWithCount = aKeyWord + "," + Integer.toString(aKeyWordCount); // Coverts the count
                        to a string and concatenates aKeyWord and aKeyWordCount
144                     context.write(key,new Text(compoundKeyWithCount)); // Emits the reviewRating number (key) as "2" and
                        the compound (value) "word,count"
145                 }
146
147                 localCombinerCount.clear(); // Clear the HashMap
148             }
149     }
150
151     /*  Data is read into the reviewReducer, finally aggregated/collated and written out to a file as the final result */
152     public static class reviewReducer extends Reducer < Text, Text ,Text, Text >  // reviewReducer receives <Text, Text> and
        sends <Text, Text>
153     {
154
155         /* Declare variables for the reviewReducer */
156         Text frequentWord_Frequency = new Text(); // Declare variable frequentWord_Frequency to represent a word/s in the
            review for which we are counting its frequency
157         HashMap <String, Integer> reducerCount = new HashMap <String, Integer>(); // Declare 'reducerCount' HashMap to store
            the word (for a given review) and highest frequency found, nine HashMaps are produced - one for each reviewRating
158
159                 public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
160                 {
161
162                     StringBuffer wordWithHighestFrequency = new StringBuffer(); // Declare 'wordWithHighestFrequency'
                        variable to store word/s from each final HashMap with highest occurance
163
164                     for (Text value : values) // Iterate over the values read in from the context
165                     {
166                         StringTokenizer itr = new StringTokenizer(value.toString(),","); // Seperate the compoundValue (which
                            contains "word,12")
```

```
167                         String aWord = itr.nextToken(); // Pull the word out as a token called aWord
168                         Integer count = Integer.parseInt(itr.nextToken()); // Parse over the token string to pull out the
                            integer count
169
170                         if (reducerCount.containsKey(aWord))
171                         {
172                             reducerCount.put(aWord, reducerCount.get(aWord) + count); // Add the parsed integer count to the
                                count/frequency found in the HashMap if the given word is in the HashMap
173                         }
174                         else
175                         {
176                             reducerCount.put(aWord, count); // If not in the HashMap reducerCount, add the word and add its
                                count so far
177                         }
178                     }
179
180
181                     // Iterate through the reducerCount HashMaps and return the key assosciated with the highest count
182                     Integer maximumValue = Collections.max(reducerCount.values()); // Declare 'MaximumValue' as the variable
                        to hold the highest frequency
183                     for (Map.Entry<String, Integer> entry : reducerCount.entrySet()) // Iterate over values in the HashMap
184                     {
185                         if (entry.getValue() == maximumValue) // Compare the frequency of the values frequency being
                            considered, to the highest value stored so far in maximumValue
186                         {
187                             wordWithHighestFrequency.append(entry.getKey()+ "|"); // For the highest frequency found in the
                                HashMap, retrieve the assosciated word and add a '|'
188                         }
189                     }
190
191                     wordWithHighestFrequency.append(maximumValue); // Append/add the frequency for the word assosciated with
                        the highest frequency, to the end of the string
192                     frequentWord_Frequency.set(wordWithHighestFrequency.toString()); // Store this result in the variable
                        'frequentWord_Frequency'
193
194                     context.write(key, frequentWord_Frequency); // Write the final result to context
195                     reducerCount.clear();
196                 }
197     }
198
199     /* main() program initiates the program */
200     public static void main(String[] args) throws Exception
201     {
202         int exitCode = ToolRunner.run(new movieReview(), args);
203         System.exit(exitCode);
204     }
205
206     /* Configures and defines the jobs to be run in the main() program */
207     public int run(String[] args) throws Exception
208     {
209                 if (args.length != 3) // Ensure three arguments are provided to the command line
```

```
210                 {
211                     System.err.printf(" %s needs three arguments: <movie_data.txt> <results> <wordsTobeExcluded.txt> \n",
                        getClass().getSimpleName()); // Note the results directory must not exist, the program will create a new
                        directory
212                     return -1;
213                 }
214
215                 //Initialize the Hadoop job and set the jar as well as the name of the Job
216                 Job job = new Job(); // Create a new instance of the Job object
217                 job.setJarByClass(movieReview.class); // Set the jar class to use
218                 job.setJobName("reviewMR"); // Allocate the job a name for logging/tracking purposes
219
220                 //Add input and output file paths to job based on the arguments passed
221                 FileInputFormat.addInputPath(job, new Path(args[0])); // Assign the first argument provided as the input path
                    for the data to be considered in the MapReduce program
222                 FileOutputFormat.setOutputPath(job, new Path(args[1])); // Assign the second argument provided as the output
                    path for the results from the MapReduce program to be written
223
224                 // Use text objects to output the key (ideally the rating) and value (ideally the most frequent word
                    associated with the key)
225                 job.setOutputKeyClass(Text.class); // Use a text object for output of the key
226                 job.setOutputValueClass(Text.class); // Simillarly use a text object for output of the value
227
228
229                 //Set the reviewMapper and reviewReducer in the job
230                 job.setMapperClass(reviewMapper.class); // Set reviewMapper as the map class for the job
231                 job.setCombinerClass(reviewCombiner.class);
232                 job.setReducerClass(reviewReducer.class); // Set reviewReducer as the reduce class for the job
233                 //job.setNumReduceTasks(1); // Set the number of reducerReducers to be called - this number dictates how many
                    result files are created
234
235                 // Ensure the program excludes the words to be ommitted from the word count
236                 DistributedCache.addCacheFile(new Path(args[2]).toUri(), job.getConfiguration()); // Assign the second
                    argument provided as the location of the file containing words/punctuation to be excluded
237
238                 //Wait for the job to complete and print if the job was successful or not
239                 Integer returnValue = job.waitForCompletion(true) ? 0:1; // Unix notation to describe if a job is successful
                    (0) or unsuccessful (1)
240                 if(job.isSuccessful())
241                 {
242                     System.out.println("The movieReview MapReduce job was successful.");
243                 }
244                 else if(!job.isSuccessful())
245                 {
246                     System.out.println("The movieReview MapReduce job was unsuccessful...please review the program and
                        address errors.");
247                 }
248
249             return returnValue; // Return 0 or 1 depending on job success
250         }
251
```

```
252 }
```