Gavin Gunawardena

INFS 774 Big Data Analytics

Assignment 3

7/2/21

Due: 7/2/2021 + one-week grace period

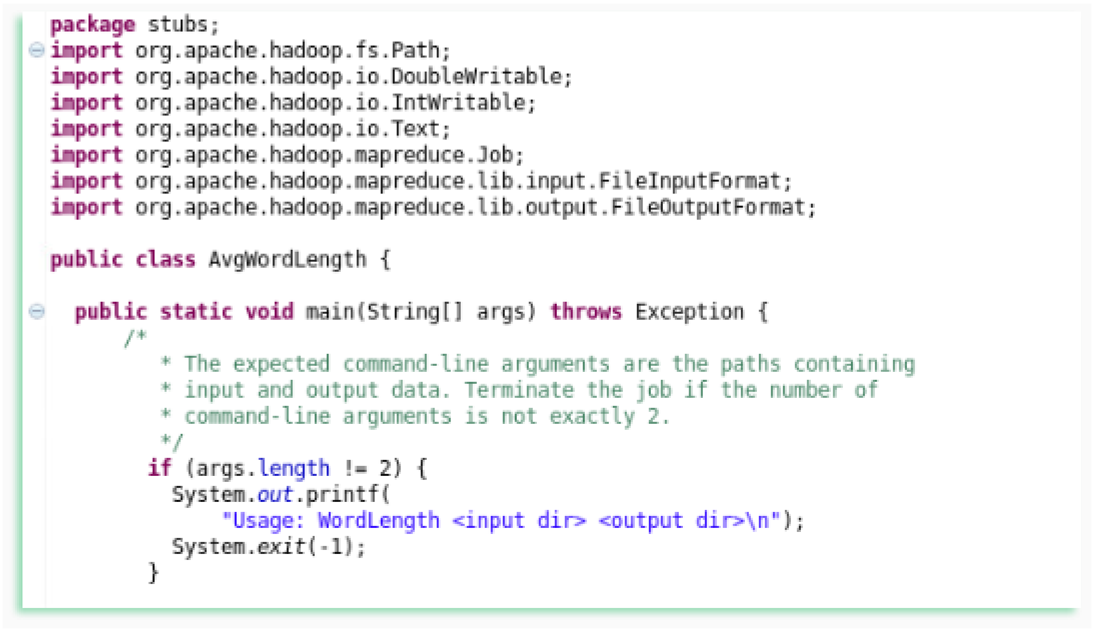
Please finish three tasks (Q1, Q2, and Q3) below. Please read the Common Errors section, if you encounter any problem. Quite possibly you can your answers there.

Q1. In this session, you will be working with the Cloudera hadoop environment. Please complete the Cloudera Homework Labs – Lecture #3

a. Please complete the tutorial "Lab: Writing a MapReduce Java Program" and submit screen shots of the following (only the first lab is required, pages 1-9):
   1. 'To do' section for AverageWordLength.java in Eclipse
   2. 'To do' section for LetterMapper.java in Eclipse
   3. 'To do' section for AverageReducer.java in Eclipse
   4. "Step 3: Test your program➔3."

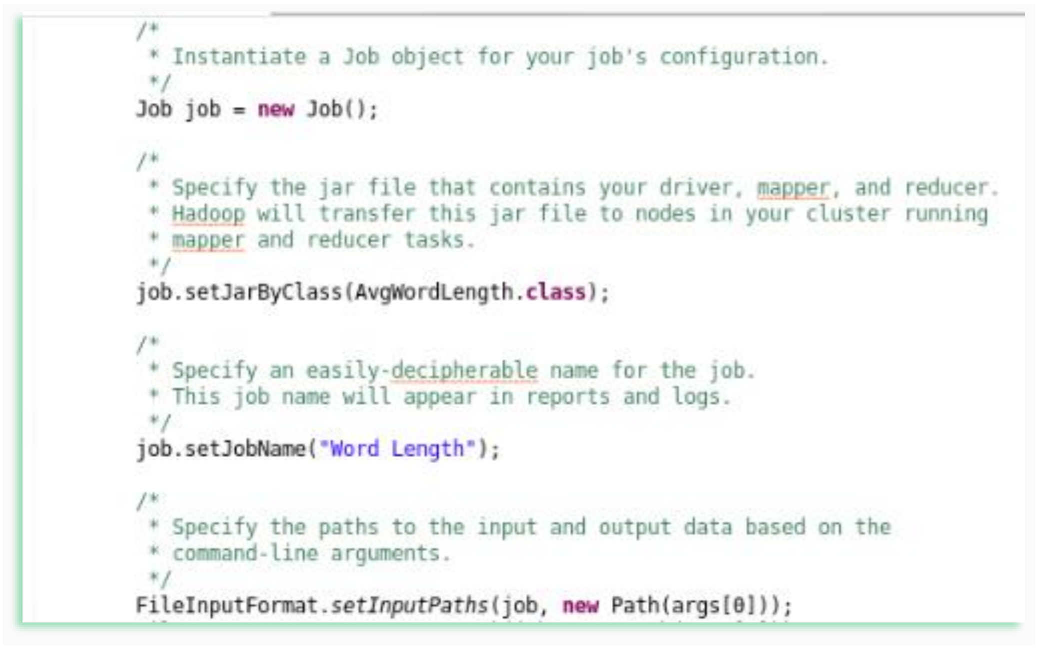
For 1, 2, and 3, you can submit a screenshot of the code; or you can submit the java code directly if you have trouble doing the screenshot.

1. **AverageWordLength.java**

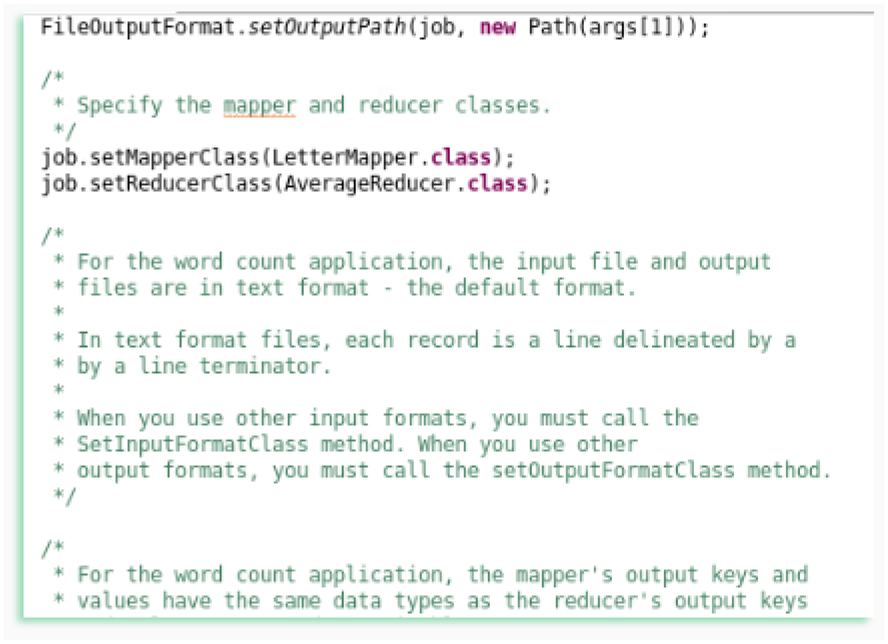


```java
package stubs;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AvgWordLength {

  public static void main(String[] args) throws Exception {
      /*
         * The expected command-line arguments are the paths containing
         * input and output data. Terminate the job if the number of
         * command-line arguments is not exactly 2.
         */
        if (args.length != 2) {
          System.out.printf(
              "Usage: WordLength <input dir> <output dir>\n");
          System.exit(-1);
        }
```

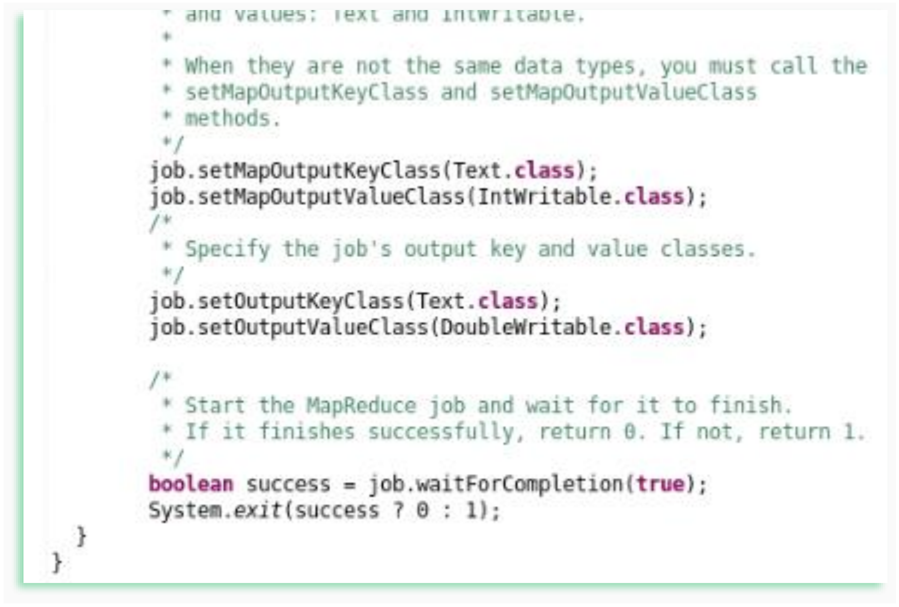
a.



```java
        /*
         * Instantiate a Job object for your job's configuration.
         */
        Job job = new Job();

        /*
         * Specify the jar file that contains your driver, mapper, and reducer.
         * Hadoop will transfer this jar file to nodes in your cluster running
         * mapper and reducer tasks.
         */
        job.setJarByClass(AvgWordLength.class);

        /*
         * Specify an easily-decipherable name for the job.
         * This job name will appear in reports and logs.
         */
        job.setJobName("Word Length");

        /*
         * Specify the paths to the input and output data based on the
         * command-line arguments.
         */
        FileInputFormat.setInputPaths(job, new Path(args[0]));
```

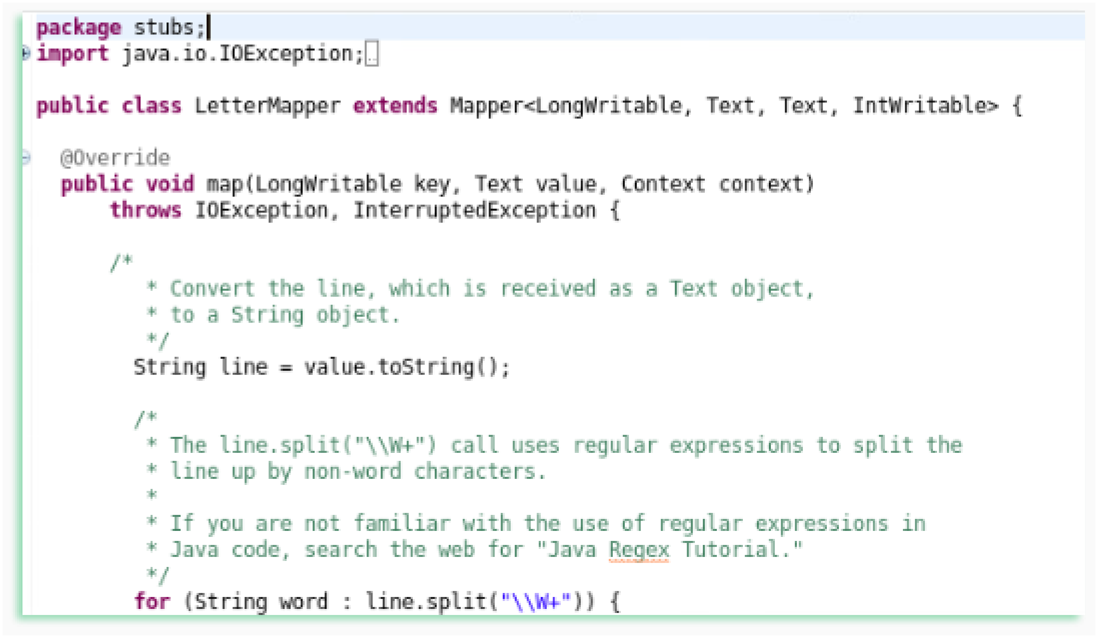


```
FileOutputFormat.setOutputPath(job, new Path(args[1]));

/*
 * Specify the mapper and reducer classes.
 */
job.setMapperClass(LetterMapper.class);
job.setReducerClass(AverageReducer.class);

/*
 * For the word count application, the input file and output
 * files are in text format - the default format.
 *
 * In text format files, each record is a line delineated by a
 * by a line terminator.
 *
 * When you use other input formats, you must call the
 * SetInputFormatClass method. When you use other
 * output formats, you must call the setOutputFormatClass method.
 */

/*
 * For the word count application, the mapper's output keys and
 * values have the same data types as the reducer's output keys
```



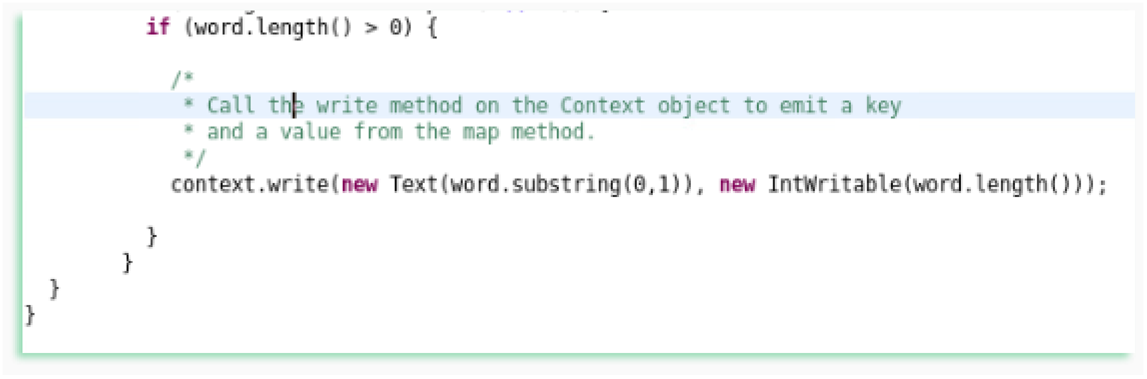
```
         * and values: Text and IntWritable.
         *
         * When they are not the same data types, you must call the
         * setMapOutputKeyClass and setMapOutputValueClass
         * methods.
         */
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        /*
         * Specify the job's output key and value classes.
         */
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        /*
         * Start the MapReduce job and wait for it to finish.
         * If it finishes successfully, return 0. If not, return 1.
         */
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}
```

2. LetterMapper.java



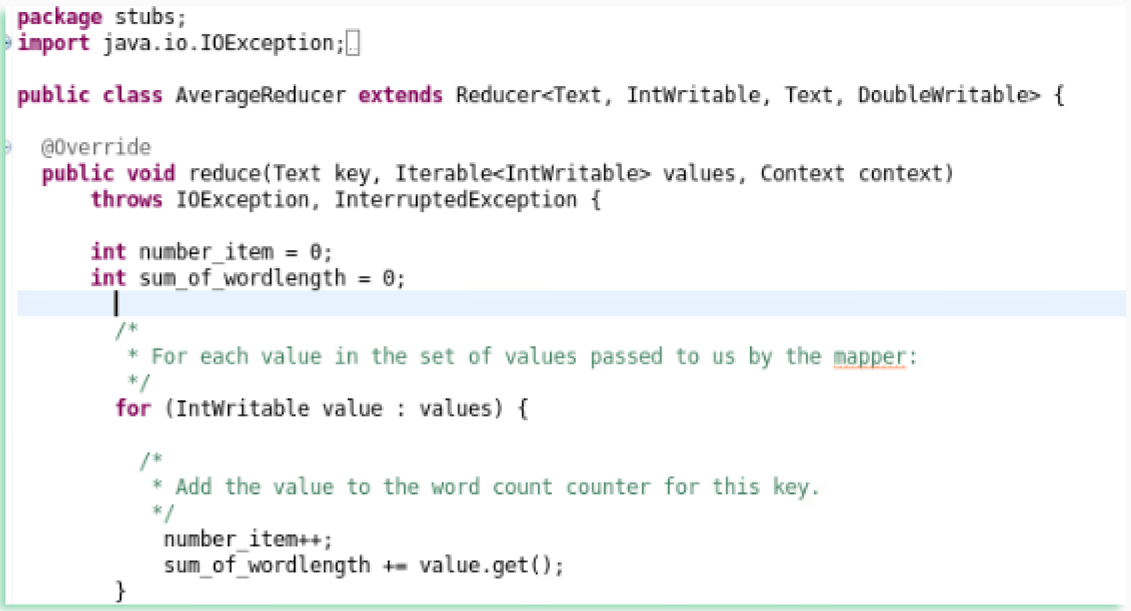
```
package stubs;
import java.io.IOException;

public class LetterMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      /*
       * Convert the line, which is received as a Text object,
       * to a String object.
       */
        String line = value.toString();

        /*
         * The line.split("\\W+") call uses regular expressions to split the
         * line up by non-word characters.
         *
         * If you are not familiar with the use of regular expressions in
         * Java code, search the web for "Java Regex Tutorial."
         */
        for (String word : line.split("\\W+")) {
```

a.

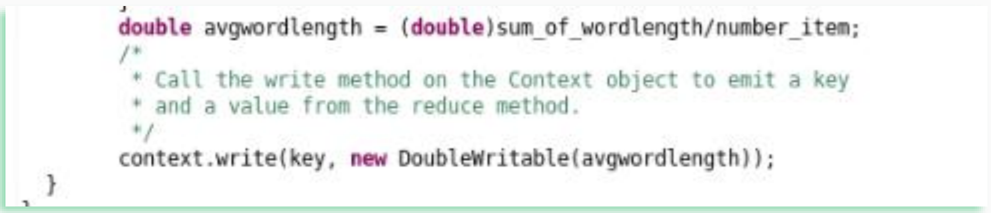


```
          if (word.length() > 0) {

            /*
             * Call the write method on the Context object to emit a key
             * and a value from the map method.
             */
            context.write(new Text(word.substring(0,1)), new IntWritable(word.length()));

          }
        }
  }
}
```

3. AverageReducer.java

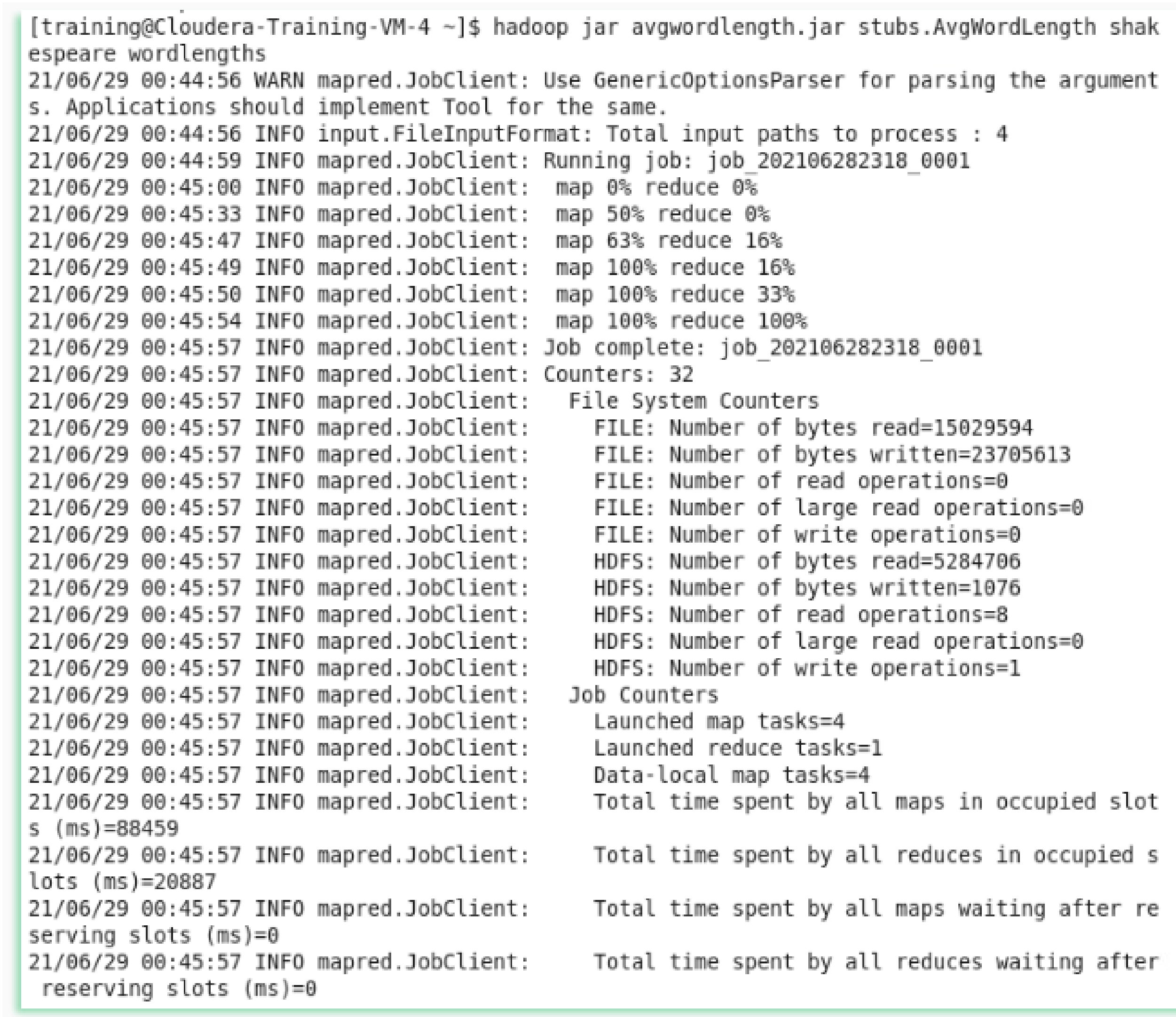
i. 

```
package stubs;
import java.io.IOException;

public class AverageReducer extends Reducer<Text, IntWritable, Text, DoubleWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

      int number_item = 0;
      int sum_of_wordlength = 0;

        /*
         * For each value in the set of values passed to us by the mapper:
         */
        for (IntWritable value : values) {

          /*
           * Add the value to the word count counter for this key.
           */
            number_item++;
            sum_of_wordlength += value.get();
        }
```



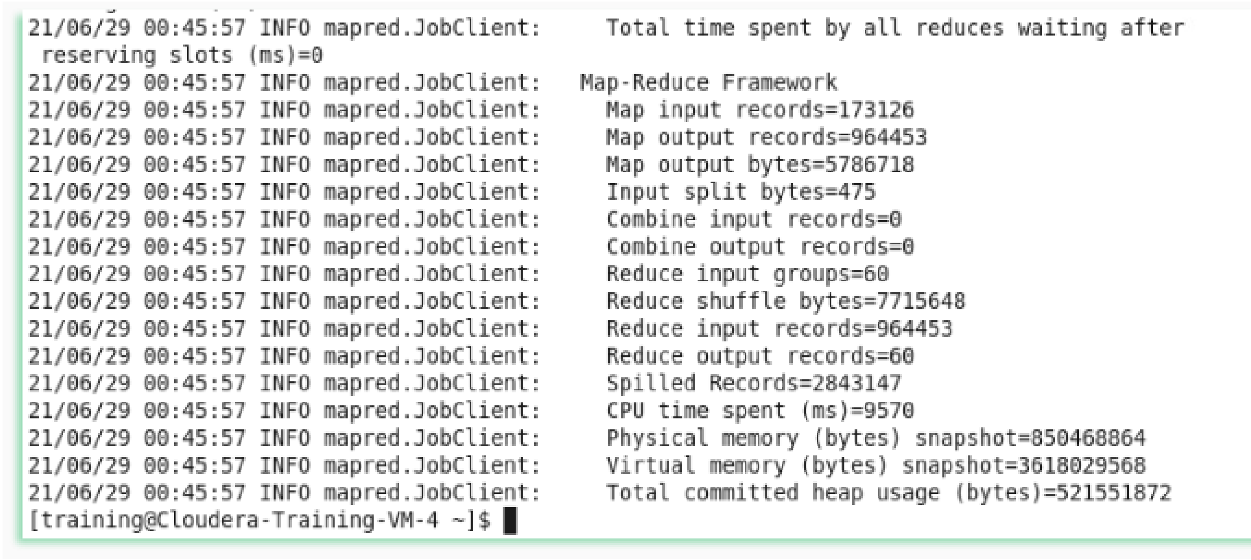
```
        double avgwordlength = (double)sum_of_wordlength/number_item;
        /*
         * Call the write method on the Context object to emit a key
         * and a value from the reduce method.
         */
        context.write(key, new DoubleWritable(avgwordlength));
  }
```

4. **Program Run**



```
[training@Cloudera-Training-VM-4 ~]$ hadoop jar avgwordlength.jar stubs.AvgWordLength shak
espeare wordlengths
21/06/29 00:44:56 WARN mapred.JobClient: Use GenericOptionsParser for parsing the argument
s. Applications should implement Tool for the same.
21/06/29 00:44:56 INFO input.FileInputFormat: Total input paths to process : 4
21/06/29 00:44:59 INFO mapred.JobClient: Running job: job_202106282318_0001
21/06/29 00:45:00 INFO mapred.JobClient:  map 0% reduce 0%
21/06/29 00:45:33 INFO mapred.JobClient:  map 50% reduce 0%
21/06/29 00:45:47 INFO mapred.JobClient:  map 63% reduce 16%
21/06/29 00:45:49 INFO mapred.JobClient:  map 100% reduce 16%
21/06/29 00:45:50 INFO mapred.JobClient:  map 100% reduce 33%
21/06/29 00:45:54 INFO mapred.JobClient:  map 100% reduce 100%
21/06/29 00:45:57 INFO mapred.JobClient: Job complete: job_202106282318_0001
21/06/29 00:45:57 INFO mapred.JobClient: Counters: 32
21/06/29 00:45:57 INFO mapred.JobClient:   File System Counters
21/06/29 00:45:57 INFO mapred.JobClient:     FILE: Number of bytes read=15029594
21/06/29 00:45:57 INFO mapred.JobClient:     FILE: Number of bytes written=23705613
21/06/29 00:45:57 INFO mapred.JobClient:     FILE: Number of read operations=0
21/06/29 00:45:57 INFO mapred.JobClient:     FILE: Number of large read operations=0
21/06/29 00:45:57 INFO mapred.JobClient:     FILE: Number of write operations=0
21/06/29 00:45:57 INFO mapred.JobClient:     HDFS: Number of bytes read=5284706
21/06/29 00:45:57 INFO mapred.JobClient:     HDFS: Number of bytes written=1076
21/06/29 00:45:57 INFO mapred.JobClient:     HDFS: Number of read operations=8
21/06/29 00:45:57 INFO mapred.JobClient:     HDFS: Number of large read operations=0
21/06/29 00:45:57 INFO mapred.JobClient:     HDFS: Number of write operations=1
21/06/29 00:45:57 INFO mapred.JobClient:   Job Counters
21/06/29 00:45:57 INFO mapred.JobClient:     Launched map tasks=4
21/06/29 00:45:57 INFO mapred.JobClient:     Launched reduce tasks=1
21/06/29 00:45:57 INFO mapred.JobClient:     Data-local map tasks=4
21/06/29 00:45:57 INFO mapred.JobClient:     Total time spent by all maps in occupied slot
s (ms)=88459
21/06/29 00:45:57 INFO mapred.JobClient:     Total time spent by all reduces in occupied s
lots (ms)=20887
21/06/29 00:45:57 INFO mapred.JobClient:     Total time spent by all maps waiting after re
serving slots (ms)=0
21/06/29 00:45:57 INFO mapred.JobClient:     Total time spent by all reduces waiting after
 reserving slots (ms)=0
```

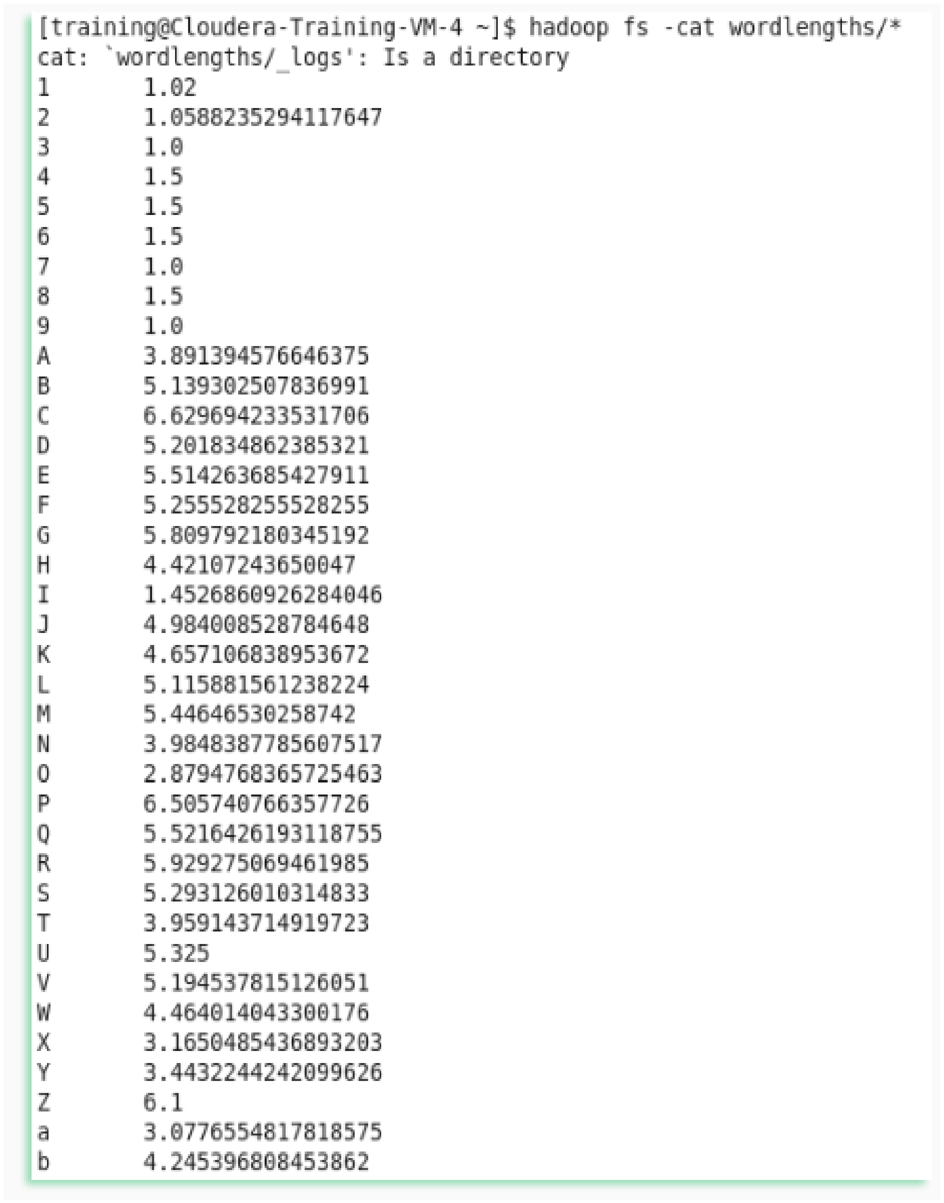
a.



```
21/06/29 00:45:57 INFO mapred.JobClient:     Total time spent by all reduces waiting after
 reserving slots (ms)=0
21/06/29 00:45:57 INFO mapred.JobClient:   Map-Reduce Framework
21/06/29 00:45:57 INFO mapred.JobClient:     Map input records=173126
21/06/29 00:45:57 INFO mapred.JobClient:     Map output records=964453
21/06/29 00:45:57 INFO mapred.JobClient:     Map output bytes=5786718
21/06/29 00:45:57 INFO mapred.JobClient:     Input split bytes=475
21/06/29 00:45:57 INFO mapred.JobClient:     Combine input records=0
21/06/29 00:45:57 INFO mapred.JobClient:     Combine output records=0
21/06/29 00:45:57 INFO mapred.JobClient:     Reduce input groups=60
21/06/29 00:45:57 INFO mapred.JobClient:     Reduce shuffle bytes=7715648
21/06/29 00:45:57 INFO mapred.JobClient:     Reduce input records=964453
21/06/29 00:45:57 INFO mapred.JobClient:     Reduce output records=60
21/06/29 00:45:57 INFO mapred.JobClient:     Spilled Records=2843147
21/06/29 00:45:57 INFO mapred.JobClient:     CPU time spent (ms)=9570
21/06/29 00:45:57 INFO mapred.JobClient:     Physical memory (bytes) snapshot=850468864
21/06/29 00:45:57 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=3618029568
21/06/29 00:45:57 INFO mapred.JobClient:     Total committed heap usage (bytes)=521551872
[training@Cloudera-Training-VM-4 ~]$ █
```

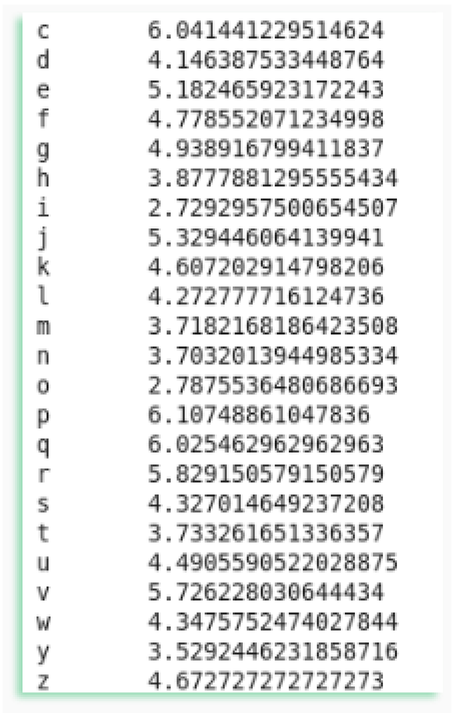
For 4. Results of Step 3.3 on page 8. You need to type hadoop fs -cat wordlengths/* and do a screenshot of the results.

1. **Program run Results**



```
[training@Cloudera-Training-VM-4 ~]$ hadoop fs -cat wordlengths/*
cat: `wordlengths/_logs': Is a directory
1       1.02
2       1.0588235294117647
3       1.0
4       1.5
5       1.5
6       1.5
7       1.0
8       1.5
9       1.0
A       3.891394576646375
B       5.139302507836991
C       6.629694233531706
D       5.201834862385321
E       5.514263685427911
F       5.255528255528255
G       5.809792180345192
H       4.42107243650047
I       1.4526860926284046
J       4.984008528784648
K       4.657106838953672
L       5.115881561238224
M       5.44646530258742
N       3.9848387785607517
O       2.8794768365725463
P       6.505740766357726
Q       5.5216426193118755
R       5.929275069461985
S       5.293126010314833
T       3.959143714919723
U       5.325
V       5.194537815126051
W       4.464014043300176
X       3.1650485436893203
Y       3.4432244242099626
Z       6.1
a       3.0776554817818575
b       4.245396808453862
```

a.



```
c       6.041441229514624
d       4.146387533448764
e       5.182465923172243
f       4.778552071234998
g       4.938916799411837
h       3.8777881295555434
i       2.7292957500654507
j       5.329446064139941
k       4.607202914798206
l       4.272777716124736
m       3.7182168186423508
n       3.7032013944985334
o       2.7875536480686693
p       6.10748861047836
q       6.025462962962963
r       5.829150579150579
s       4.327014649237208
t       3.733261651336357
u       4.4905590522028875
v       5.726228030644434
w       4.3475752474027844
y       3.5292446231858716
z       4.672727272727273
```

I think you should definitely try the second lab, but you don't need to submit anything.
Lab 3 "writing a MapReduce Streaming Program" is quite useful. Some of you may not be familiar with Java, the native language of Hadoop, but know other languages such as Python, Perl, Ruby, etc well. It's possible to write MapReduce code using these languages. Let's say you want use Python to write mapreduce code. You have two options. The first option is to write a streaming program in Python according to the instructions in the third lab. The second option is use Pydoop, a python interface to Hadoop. You are encouraged but not required to do this lab.
Lab 4 is about conducting unit tests, which is extremely important for Hadoop programming since it's very difficult to debug in the Hadoop environment. https://cwiki.apache.org/confluence/display/MRUNIT/MRUnit+Tutorial provides useful sample code for doing unit test. Please try this lab.

Q2. In this session, you will be working with the Cloudera hadoop environment. Please complete the Cloudera Homework Labs – Lecture #7

a. Please complete the tutorial "Lab: Creating an Inverted Index ".

   i. Provide a screen shots similar to those for the Homework for Lecture #3 above.

   ii. Hints:

      a. For the Driver:

```
/*
 * We are using a KeyValueText file as the input file.
 * Therefore, we must call setInputFormatClass.
 * There is no need to call setOutputFormatClass, because the
 * application uses a text file for output.
 */
job.setInputFormatClass(KeyValueTextInputFormat.class);

FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(IndexMapper.class);
job.setReducerClass(IndexReducer.class);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
```

      b. To import the input class needed for this assignment, add the following line to you import statements:

```
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
```

      c. For the Mapper:

```
import java.io.IOException;

    import org.apache.hadoop.fs.Path;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapreduce.lib.input.FileSplit;
    import org.apache.hadoop.mapreduce.Mapper;

    /*
     * Get the FileSplit for the input file, which provides access
     * to the file's path. You need the file's path because it
     * contains the name of the play.
     */
    FileSplit fileSplit = (FileSplit) context.getInputSplit();
    Path path = fileSplit.getPath();

    /*
```

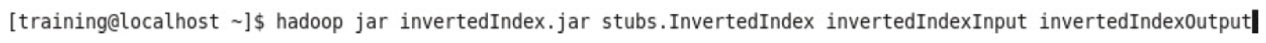
```
   * Call the getName method on the Path object to retrieve the
    * file's name, which is the name of the play. Then append
   * "@" and the line number to the play's name. The resulting
   * string is the location of the words on that line.
   */
  String wordPlace = path.getName() + "@" + key.toString();
  Text location = new Text(wordPlace);

  /*
   * Convert the line to lower case.
   */
  String lc_line = value.toString().toLowerCase();

  /*
   * Split the line into words. For each word on the line,
   * emit an output record that has the word as the key and
   * the location of the word as the value.
   */
  for (String word : lc_line.split("\\W+")) {
    if (word.length() > 0) {
      context.write(new Text(word), location);
    }
  }
```

d. For the Reducer, you need code to create a string from the list of input values created by the sort and shuffle. The code is:

```
/* Add the following line before the @Overide */
    private static final String SEP = ",";

/* The following code is added after the Public void reduce statement */
    StringBuilder valueList = new StringBuilder();
    boolean firstValue = true;
/*
     * For each "play name@line number" in the input value set:
     */
    for (Text value : values) {

      /*
       * If this is not the word's first location, add a comma to the
       * end of valueList.
       */
      if (!firstValue) {
        valueList.append(SEP);
      } else {
        firstValue = false;
      }

      /*
       * Convert the location to a String and append it to valueList.
       */
      valueList.append(value.toString());
    }

    /*
     * Emit the index entry.
     */
    context.write(key, new Text(valueList.toString()));
```

e. To create the jar file, follow the step 3 “Use Eclipse to Compile Your Solution” p. 5 from the previous lab (Lecture 3)

f. To run your application, follow the step 4 “Test your program”, p. 8 from the previous lab (Lecture 3).

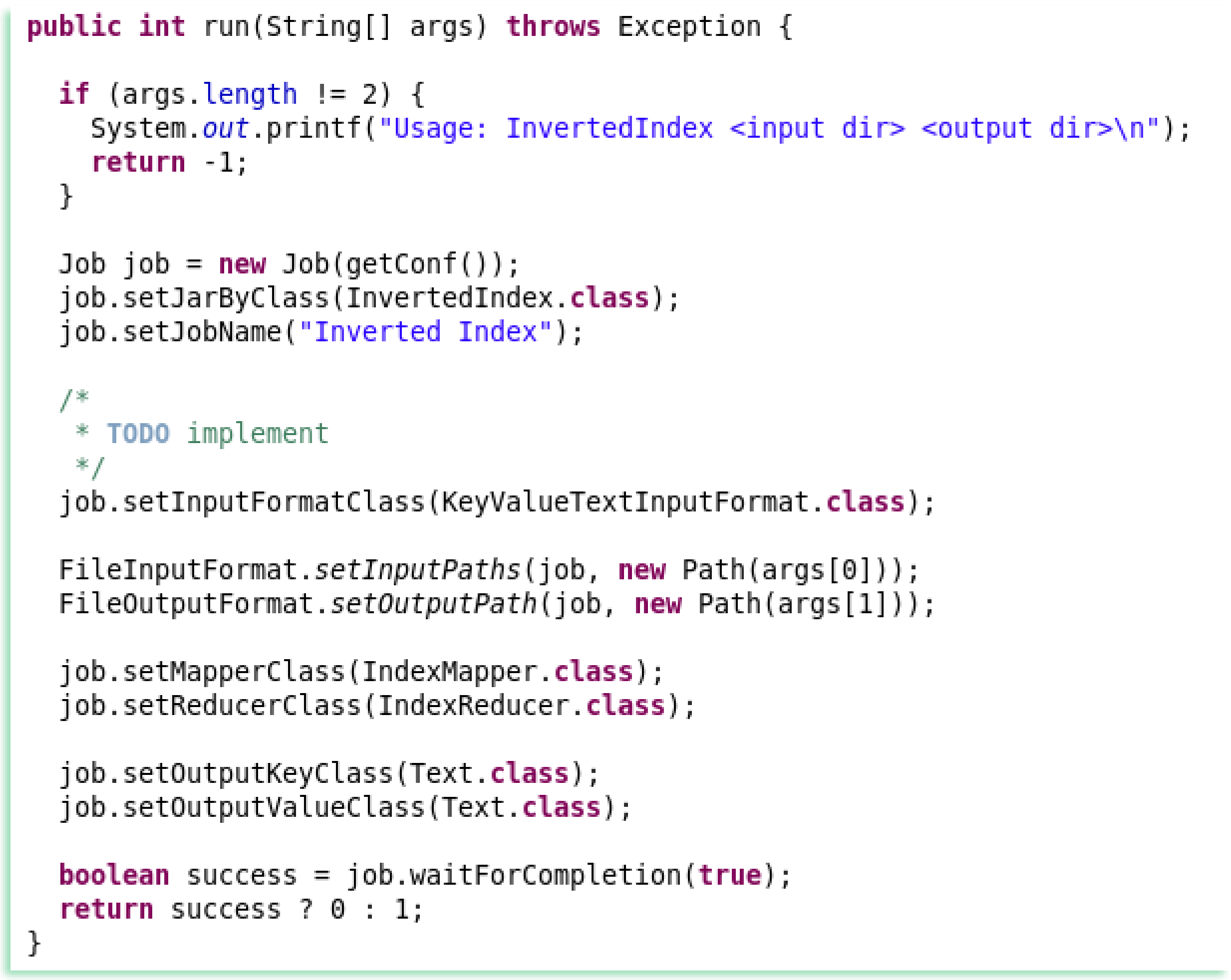


```
[training@localhost ~]$ hadoop jar invertedIndex.jar stubs.InvertedIndex invertedIndexInput invertedIndexOutput
```

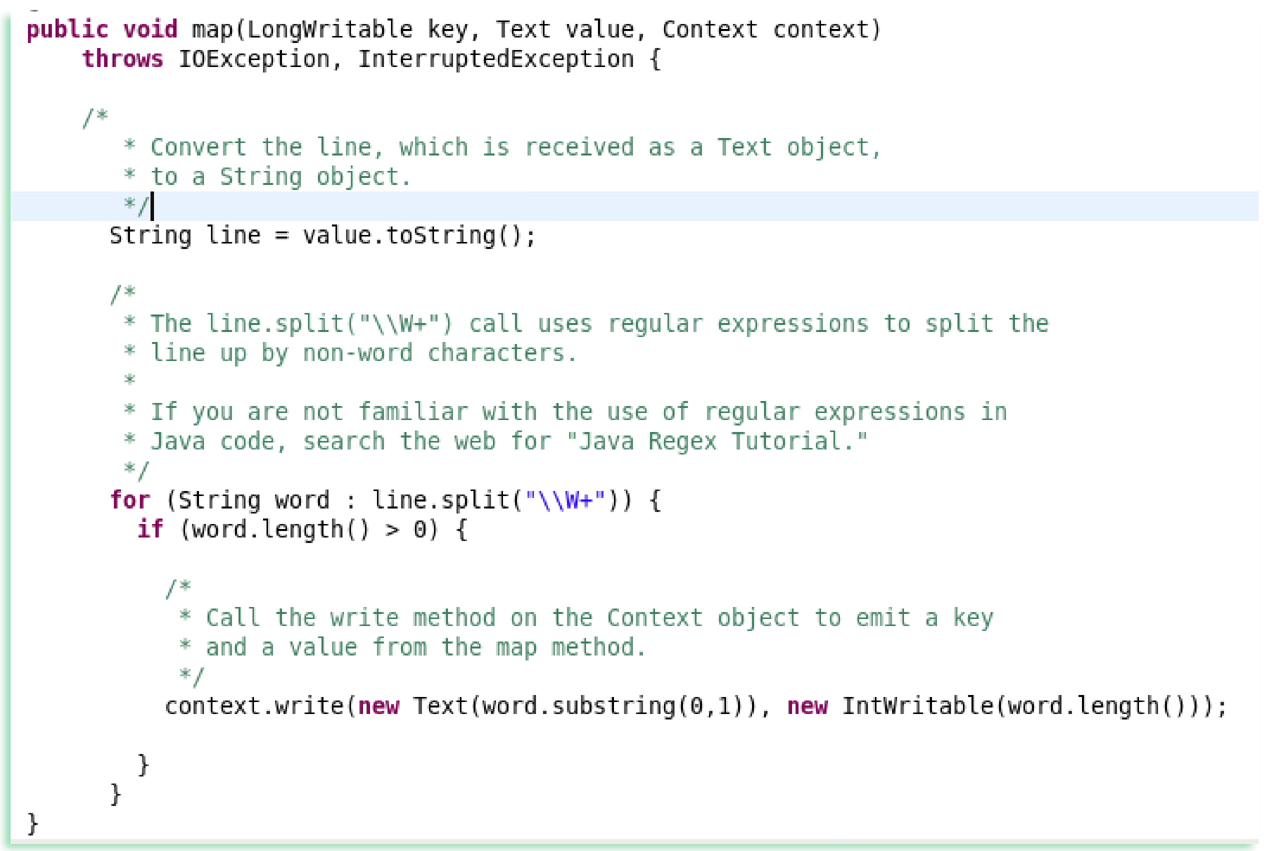
g. If you need to re-run your application follow the step outlined in last week’s session (Lecture 2) step 10 page 5

For Lec 7 lab (Q2), you need to submit:

1. The java code "InvertedIndex.java" or a screenshot of the code (just show me the run method. You can remove the comments when you make the screenshot)

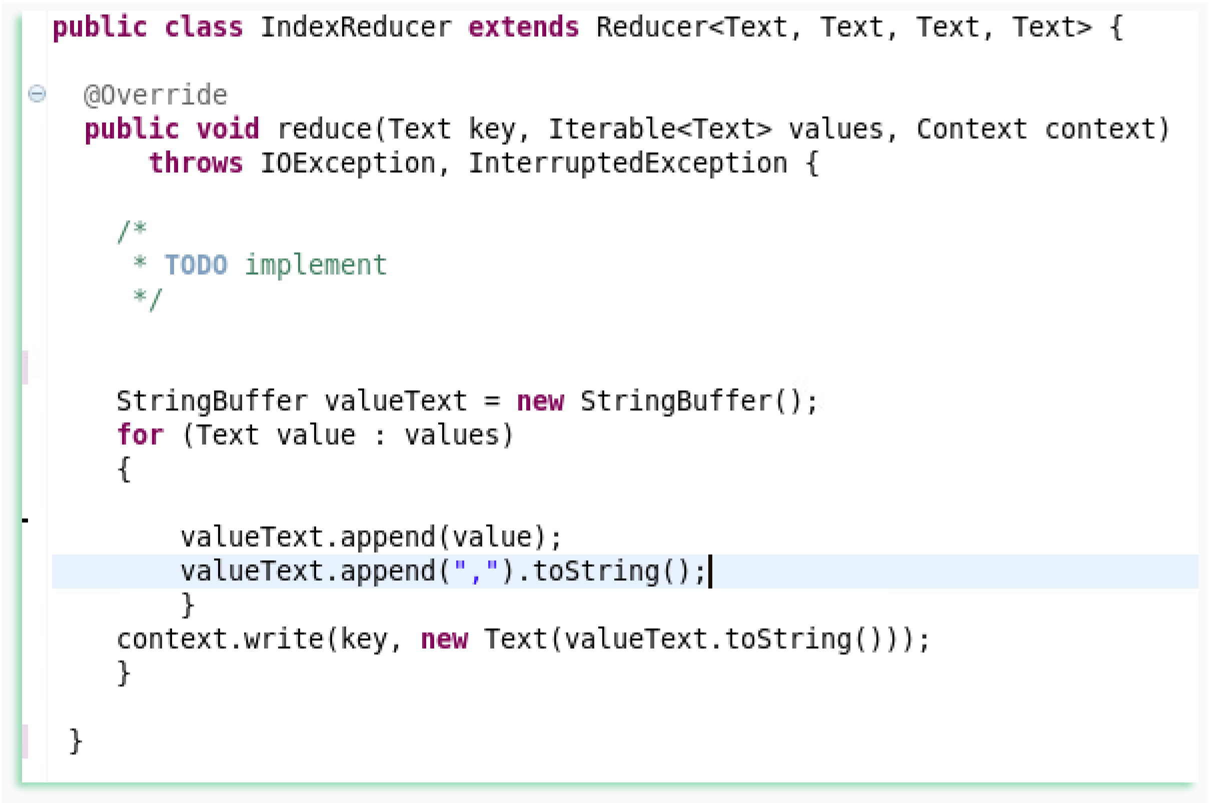

```java
public int run(String[] args) throws Exception {

  if (args.length != 2) {
    System.out.printf("Usage: InvertedIndex <input dir> <output dir>\n");
    return -1;
  }

  Job job = new Job(getConf());
  job.setJarByClass(InvertedIndex.class);
  job.setJobName("Inverted Index");

  /*
   * TODO implement
   */
  job.setInputFormatClass(KeyValueTextInputFormat.class);

  FileInputFormat.setInputPaths(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.setMapperClass(IndexMapper.class);
  job.setReducerClass(IndexReducer.class);

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);

  boolean success = job.waitForCompletion(true);
  return success ? 0 : 1;
}
```

2. The java code "IndexMapper.java" or a screenshot of the "map" method in the code.



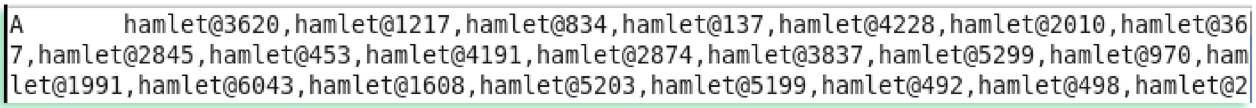
```java
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

    /*
       * Convert the line, which is received as a Text object,
       * to a String object.
       */
      String line = value.toString();

      /*
       * The line.split("\\W+") call uses regular expressions to split the
       * line up by non-word characters.
       *
       * If you are not familiar with the use of regular expressions in
       * Java code, search the web for "Java Regex Tutorial."
       */
      for (String word : line.split("\\W+")) {
        if (word.length() > 0) {

          /*
           * Call the write method on the Context object to emit a key
           * and a value from the map method.
           */
          context.write(new Text(word.substring(0,1)), new IntWritable(word.length()));

        }
      }
}
```

3. The java code "IndexReduce.java" or a screenshot of the "reduce" method in the code.



```
public class IndexReducer extends Reducer<Text, Text, Text, Text> {

  @Override
  public void reduce(Text key, Iterable<Text> values, Context context)
      throws IOException, InterruptedException {

    /*
     * TODO implement
     */


    StringBuffer valueText = new StringBuffer();
    for (Text value : values)
    {

        valueText.append(value);
        valueText.append(",").toString();
        }
    context.write(key, new Text(valueText.toString()));
    }

 }
```

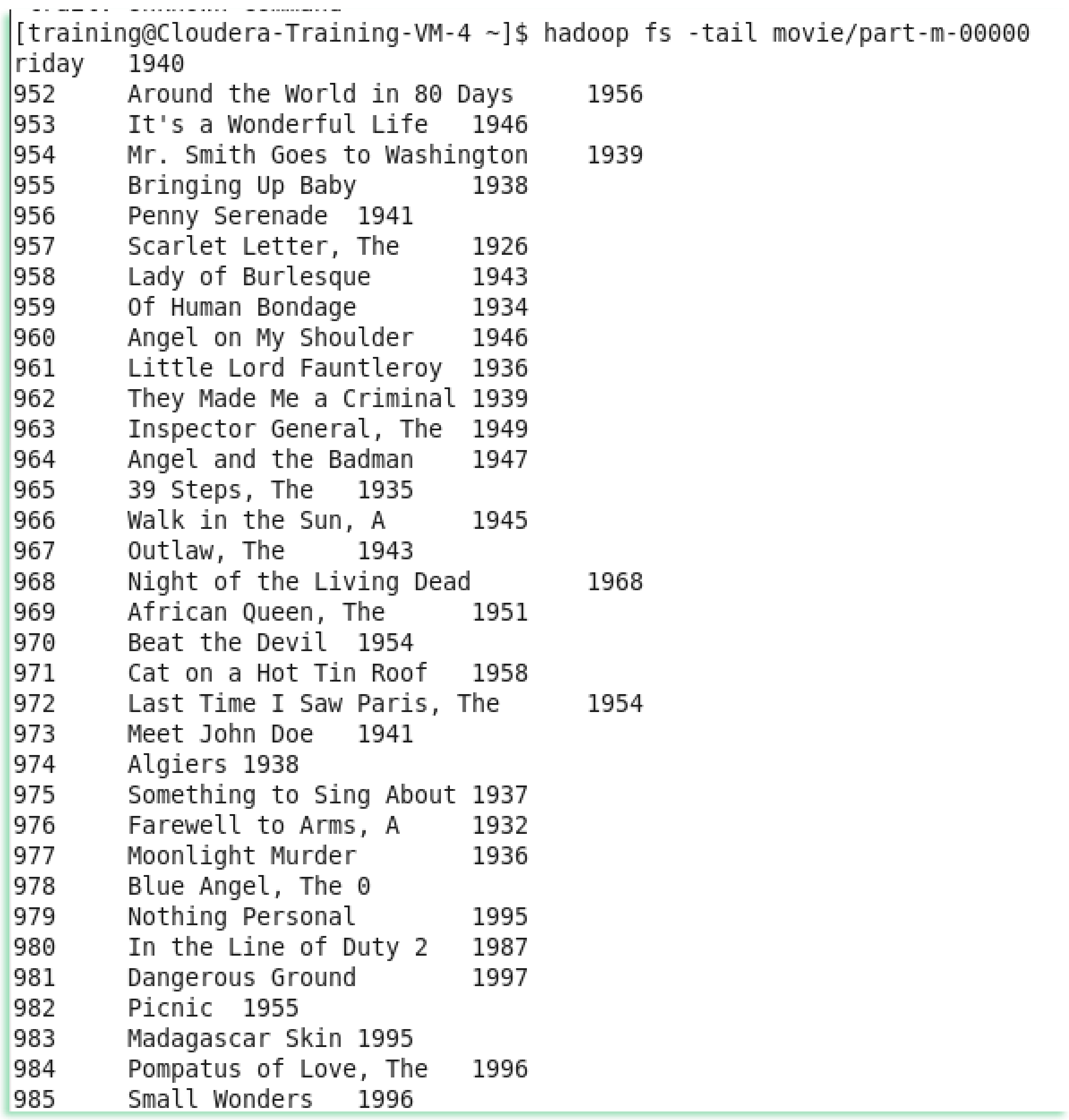
4. The screenshot of the results (I just need to see the first 3 lines of the results. you can do the following):
hadoop fs -cat invertedIndexOutput/* | head -n 3
invertedIndexOutput is the output folder name.



```
A       hamlet@3620,hamlet@1217,hamlet@834,hamlet@137,hamlet@4228,hamlet@2010,hamlet@36
7,hamlet@2845,hamlet@453,hamlet@4191,hamlet@2874,hamlet@3837,hamlet@5299,hamlet@970,ham
let@1991,hamlet@6043,hamlet@1608,hamlet@5203,hamlet@5199,hamlet@492,hamlet@498,hamlet@2
```

Q3. In this session, you will be working with the Cloudera hadoop environment. Please complete the Cloudera Homework Labs – Lecture #8

a. Please complete the tutorial “Lab: Importing Data with Sqoop”.
    1. Submit the screen shot of “Import with Sqoop➔ step 5”



```
[training@Cloudera-Training-VM-4 ~]$ hadoop fs -tail movie/part-m-00000
riday   1940
952     Around the World in 80 Days     1956
953     It's a Wonderful Life   1946
954     Mr. Smith Goes to Washington    1939
955     Bringing Up Baby        1938
956     Penny Serenade  1941
957     Scarlet Letter, The     1926
958     Lady of Burlesque       1943
959     Of Human Bondage        1934
960     Angel on My Shoulder    1946
961     Little Lord Fauntleroy  1936
962     They Made Me a Criminal 1939
963     Inspector General, The  1949
964     Angel and the Badman    1947
965     39 Steps, The   1935
966     Walk in the Sun, A      1945
967     Outlaw, The     1943
968     Night of the Living Dead        1968
969     African Queen, The      1951
970     Beat the Devil  1954
971     Cat on a Hot Tin Roof   1958
972     Last Time I Saw Paris, The      1954
973     Meet John Doe   1941
974     Algiers 1938
975     Something to Sing About 1937
976     Farewell to Arms, A     1932
977     Moonlight Murder        1936
978     Blue Angel, The 0
979     Nothing Personal        1995
980     In the Line of Duty 2   1987
981     Dangerous Ground        1997
982     Picnic  1955
983     Madagascar Skin 1995
984     Pompatus of Love, The   1996
985     Small Wonders   1996
```

Common Errors:

1. **NameNode in safe mode error**: If you see this error, please try "hdfs dfsadmin -safemode leave"
2. **Assignment 3 - "output directory not set"**: In lab 3, you need to work on three files including AverageReducer.java, LetterMap.java, and AvgWordLength.java. You need to write some java code. Primarily you need to copy the code from the wordcount project and modify the code a little. If you create the jar file without implementing the three java classes, you will see the "output directory not set" error.
3. **Assignment 3 - file/path does not exist error**: Whenever you see a input file (or path) does not exist error, you want to first make sure you have already uploaded your input dataset. If you are quite sure that you have uploaded the dataset (i.e., you can see the dataset by doing hadoop fs -ls)

and you still have such an error, then it is possible that you have uploaded the dataset more than once. I have discussed this error **previous in assignment 2**. The cause of the issue is that Hadoop does not allow us to overwrite a dataset. So if we upload the input dataset more than once, you will see the file/path not exist error. How to deal with the issue? For the input dataset not exist error, remove and re-upload the input dataset.

4. **Assignment 3 - Error opening jar:** You need to make sure that you are running the code where the jar file is located. Let's say the jar file you created is in the folder /home/training. You need to go to /home/training to wrong the jar code.
   If you really don't know how to solve the issue, you first cd ~/workspace/averagewordlength/src then use the javac command you used in assignment 2 to manually compile these files, and then use the jar cvf comman to construct the jar file manually.
5. **Assignment 3 - output directory already exist error**: When you see this error, please remove the output directory and re-run your code. The reason is that hadoop does not allow us to overwrite an existing dataset.
6. **Assignment 3 - XMLJAXBElementProvider$Text:** When you see this error, please refer to http://unmeshasreeveni.blogspot.com/2014/04/wrong-import-statements-in-hadoop.html
7. **Assignment 3 cannot resolve type error:** Whenever you see this error, quite possibly, it means a data type hasn't been imported. You can click the error icon (little red "x") and click import .... Eclipse is a very nice tool. It provides hints for resolving errors.