

CSCE 5300 Section 006 - Introduction to Big Data and Data Science

Assignment 2

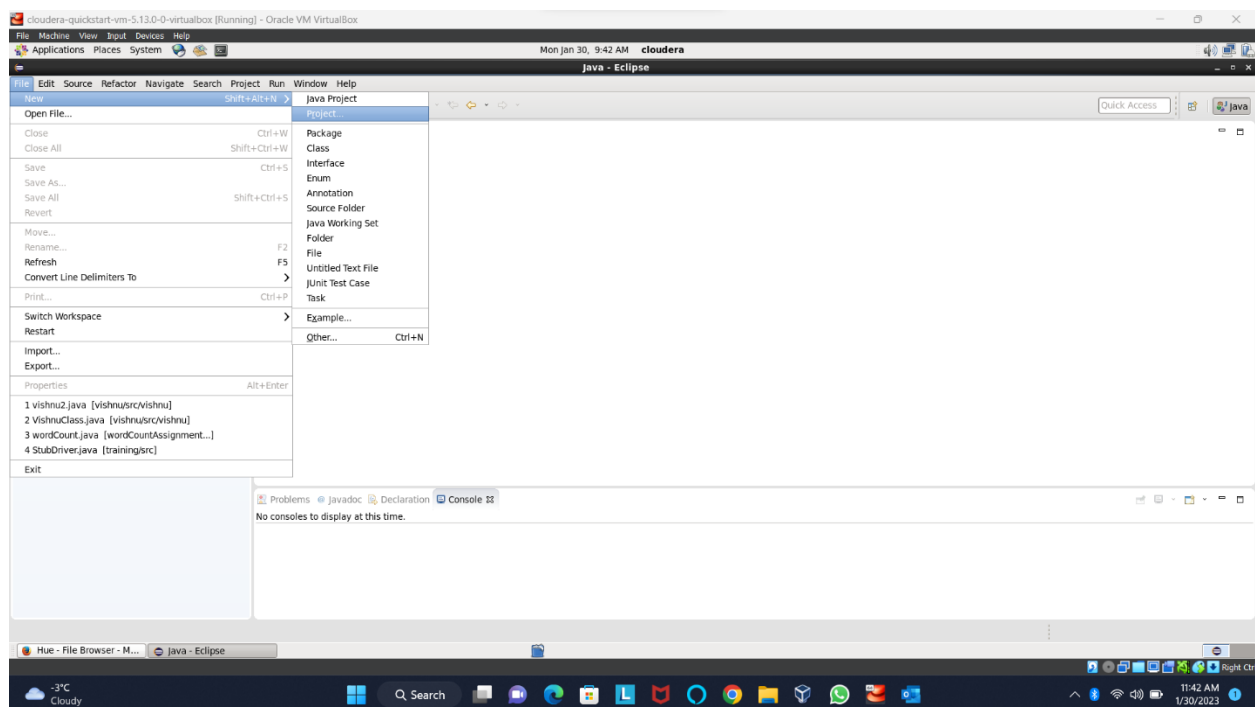
Downloaded the file from canvas and moved the file to hadoop by creating a folder named input using `hadoop fs -mkdir`

`Hadoop fs -mkdir <folder name>` is used to create a new directory in hadoop

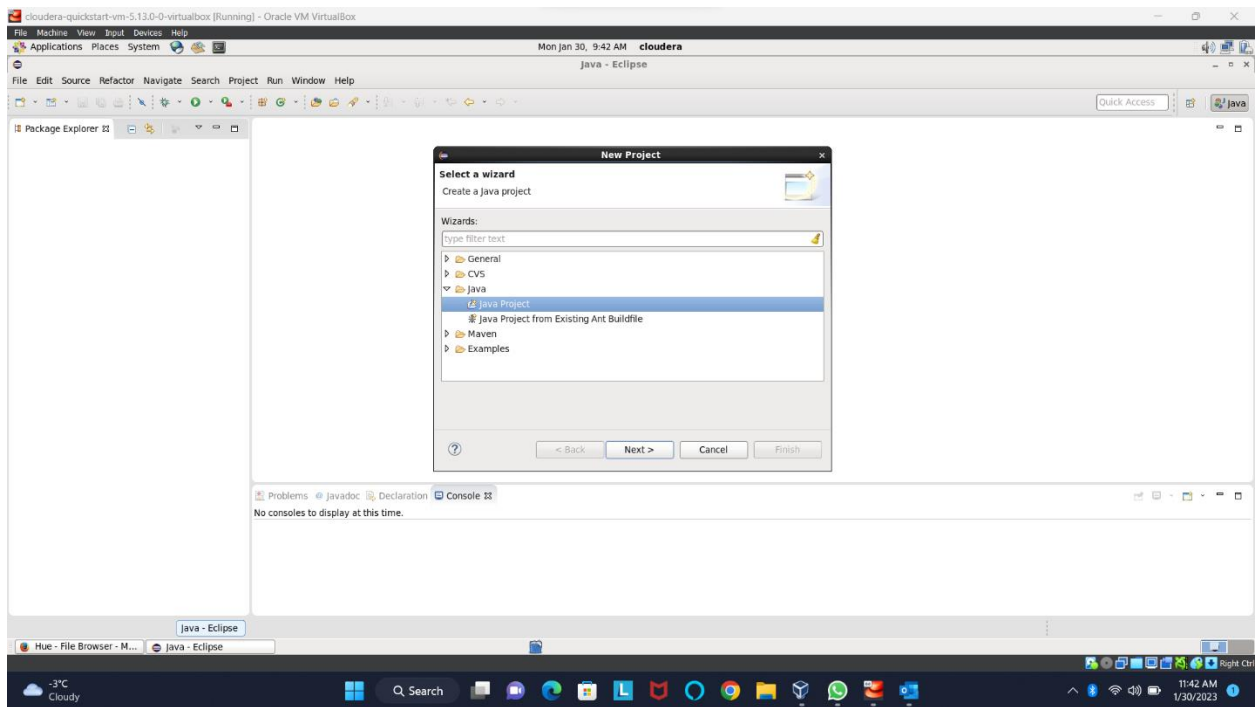
```
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ hadoop fs -mkdir input
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/sample.txt input/
[cloudera@quickstart ~]$ hadoop fs -ls input
Found 1 items
-rw-r--r-- 1 cloudera cloudera      2087 2023-01-28 12:53 input/sample.txt
[cloudera@quickstart ~]$
```

Creating a new project in Eclipse:

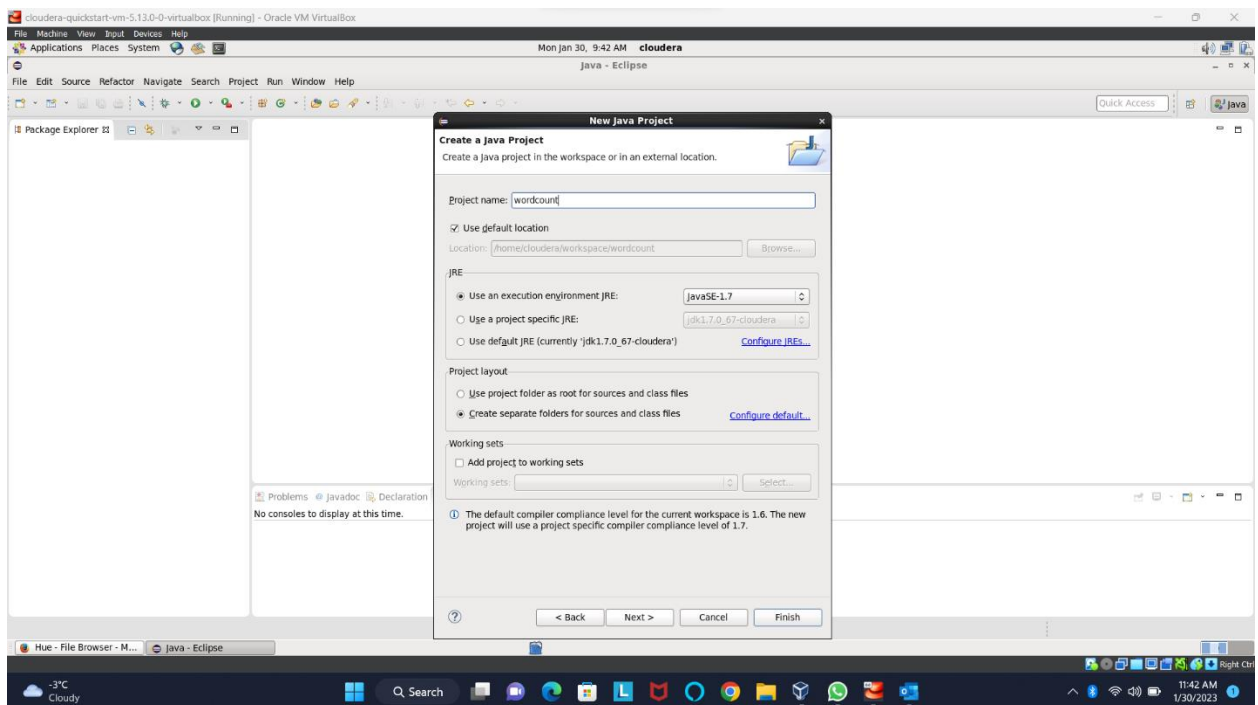
Created a new java project by clicking on file > new > Project



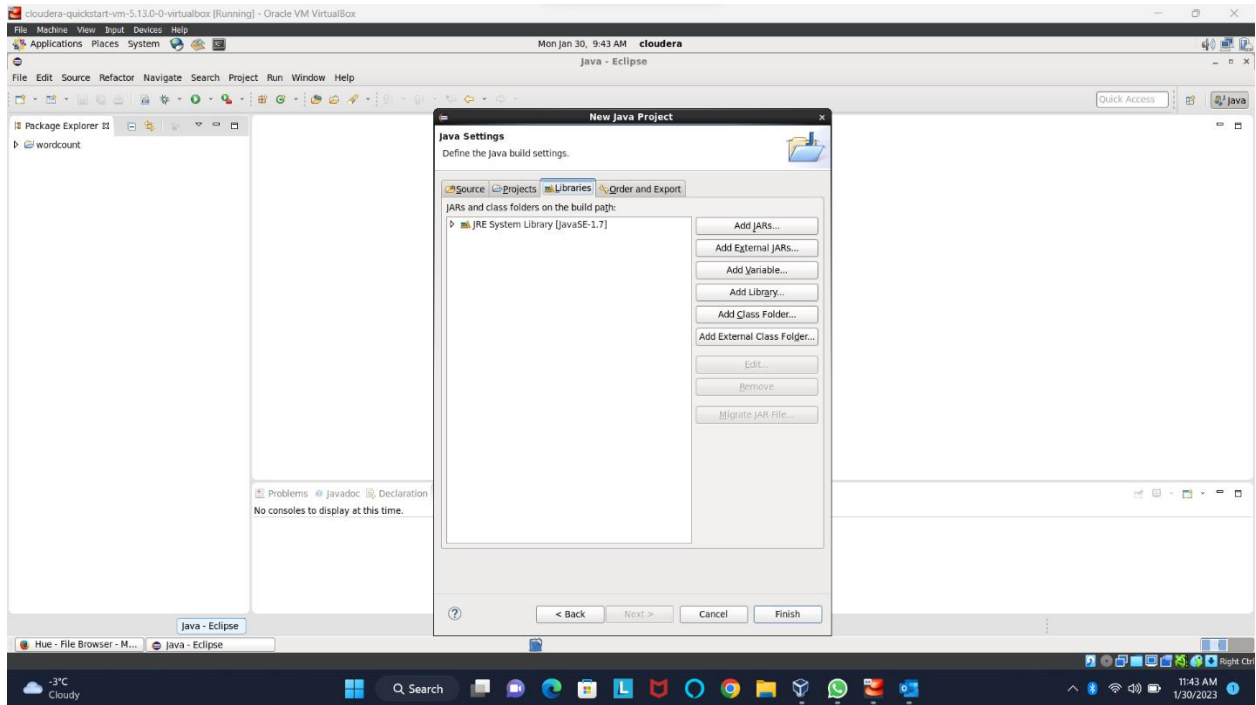
Select java project and click on next to the java project setup



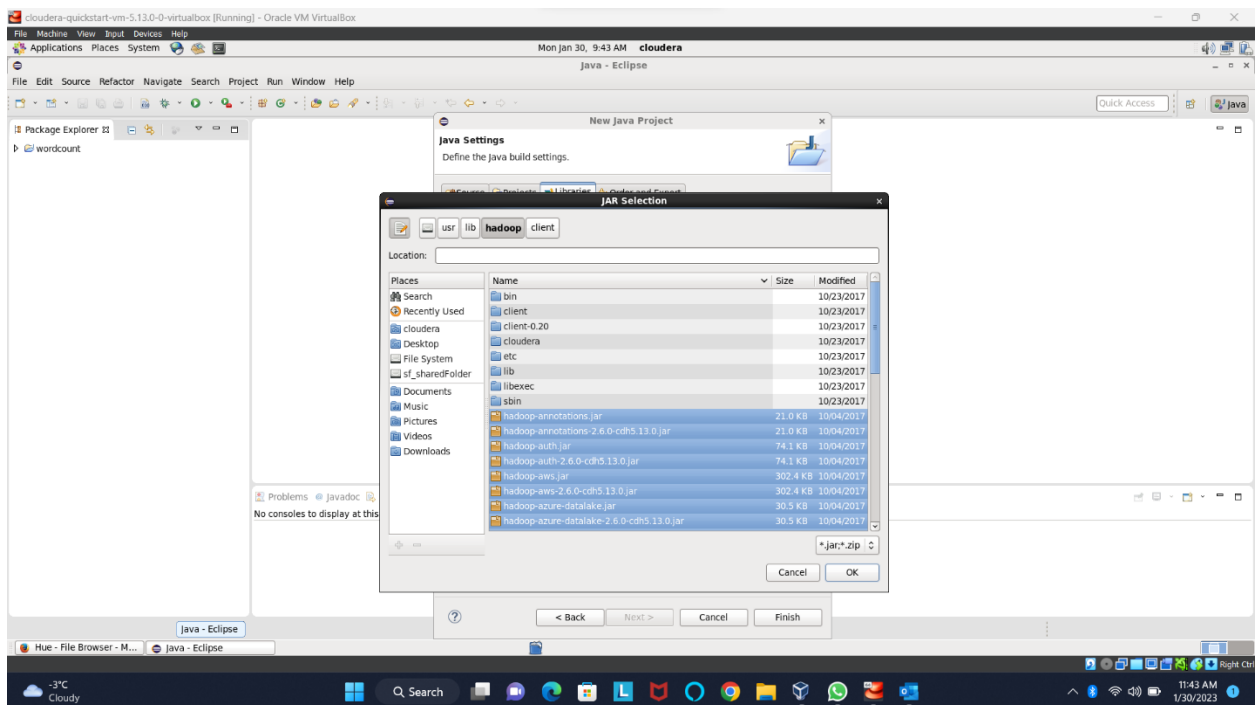
Give name of the project and clicked on next.



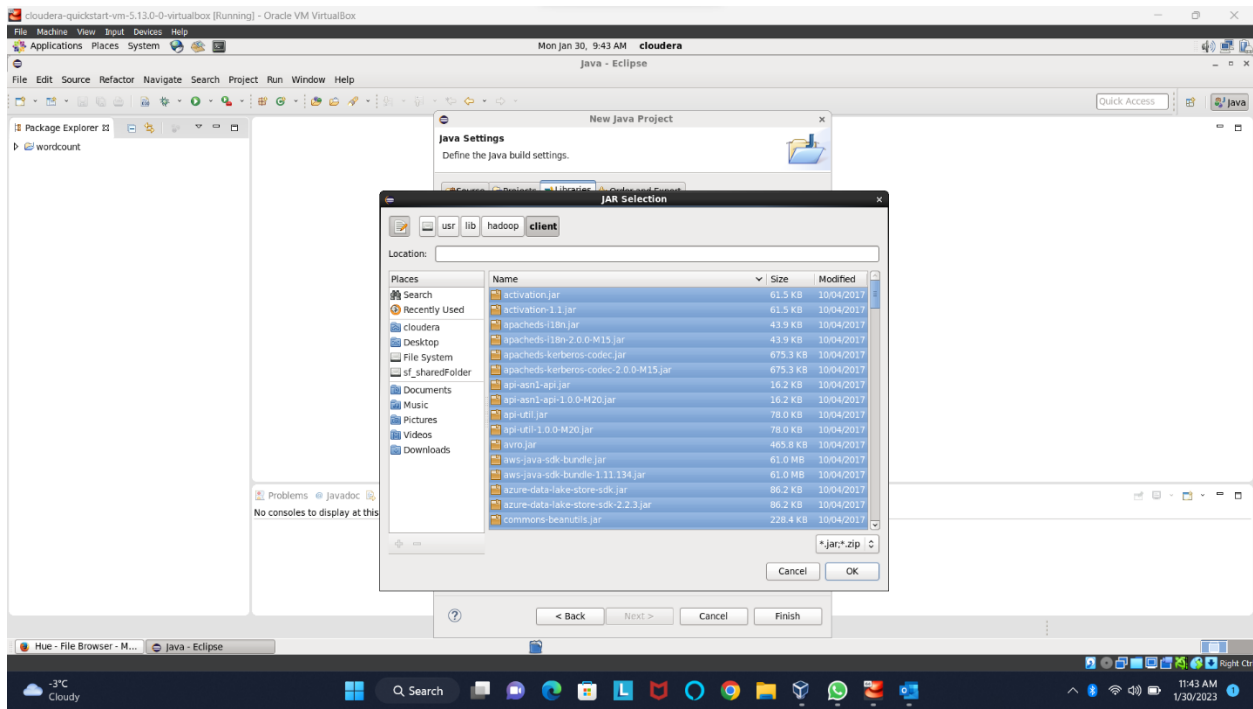
Select Libraries in the next pop up menu, New java Project Settings, and click on Add External Jars



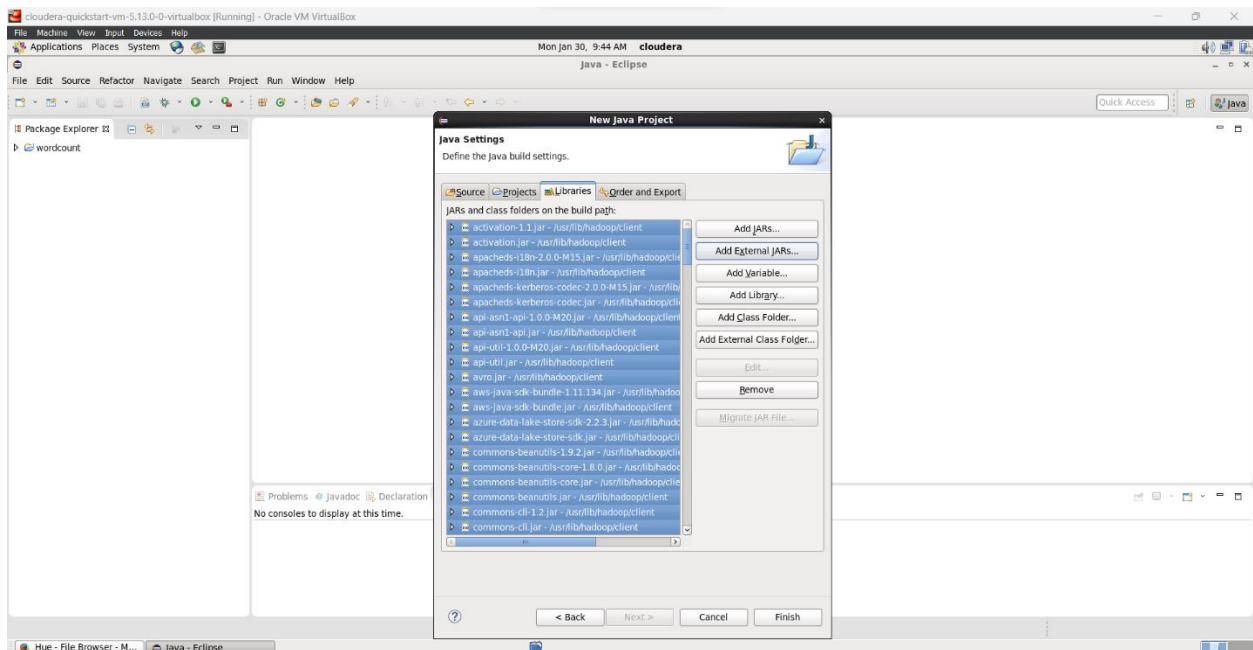
Select all the jars under hadoop folder and add them to jars by clicking on ok button.



Also select all the jars from the bin folder in hadoop and add them to the java jars by clicking on OK button.



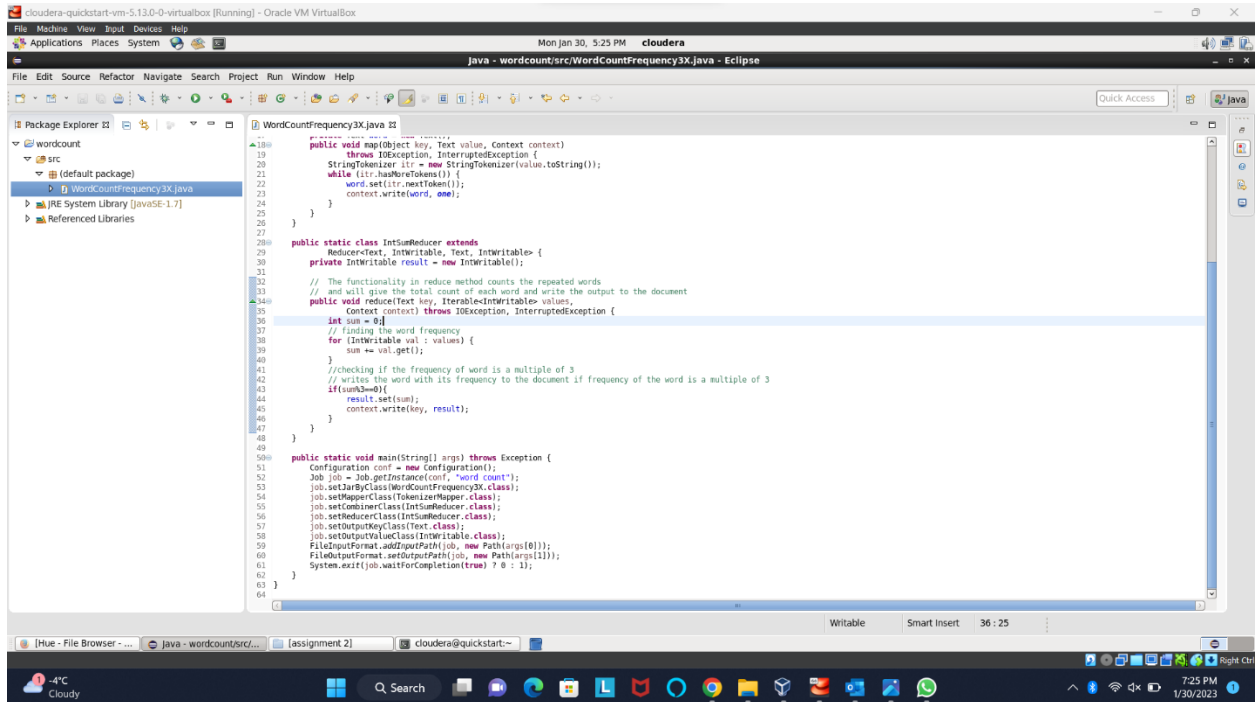
After selecting and adding all the necessary jar files to the java project, finish the project setup by clicking on finish button in the popup menu



Task 1:

Finding all the words that has the frequency of multiples of 3 in the given sample.txt file.

Got the sample code from canvas and add necessary logic to generate the file that has all the words that are repeated in multiples of 3s.

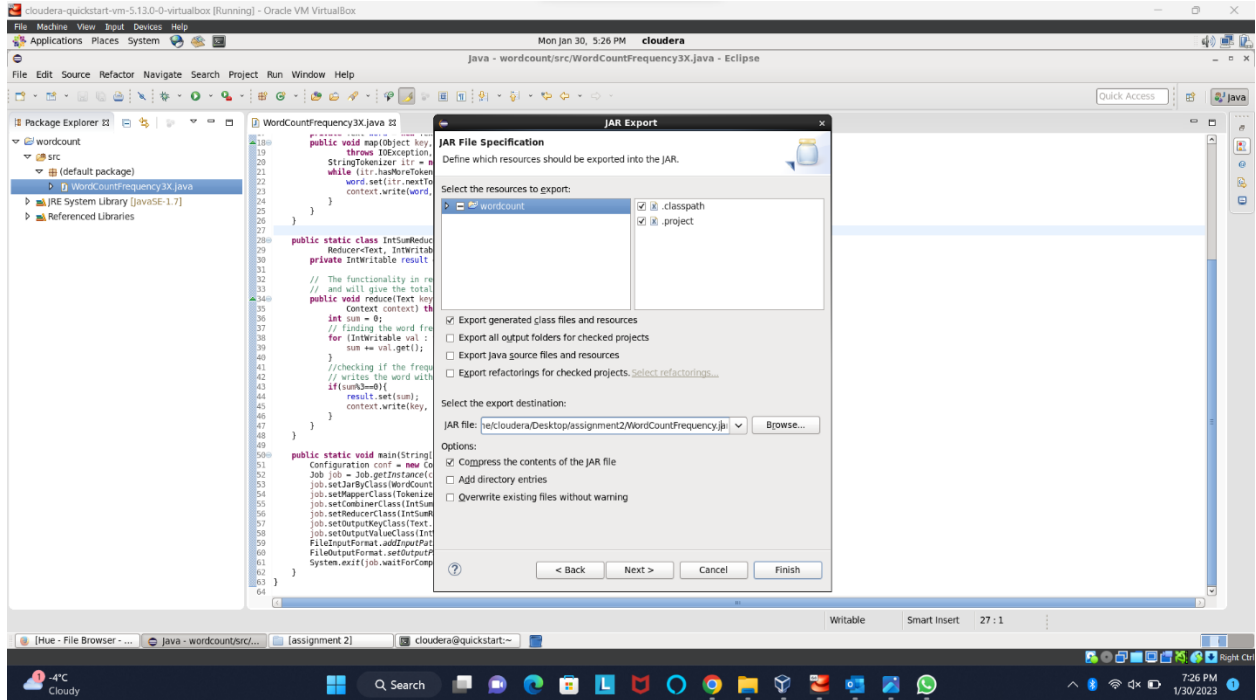


```
18 public void map(Object key, Text value, Context context)
19     throws IOException, InterruptedException {
20     StringTokenizer itr = new StringTokenizer(value.toString());
21     while (itr.hasMoreTokens()) {
22         word.set(itr.nextToken());
23         context.write(word, one);
24     }
25 }
26
27
28 public static class IntSumReducer extends
29     Reducer<Text, IntWritable, Text, IntWritable> {
30     private IntWritable result = new IntWritable();
31
32     // The functionality in reduce method counts the repeated words
33     // and will give the total count of each word and write the output to the document
34     public void reduce(Text key, Iterable<IntWritable> values,
35         Context context) throws IOException, InterruptedException {
36         int sum = 0;
37         // finding the word frequency
38         for (IntWritable val : values) {
39             sum += val.get();
40         }
41         //checking if the frequency of word is a multiple of 3
42         // writes the word with its frequency to the document if frequency of the word is a multiple of 3
43         if(sum%3==0){
44             result.set(sum);
45             context.write(key, result);
46         }
47     }
48 }
49
50 public static void main(String[] args) throws Exception {
51     Configuration conf = new Configuration();
52     Job job = Job.getInstance(conf, "word count");
53     job.setJarByClass(WordCountFrequency3X.class);
54     job.setMapperClass(TokenizerMapper.class);
55     job.setReducerClass(IntSumReducer.class);
56     job.setOutputKeyClass(Text.class);
57     job.setOutputValueClass(IntWritable.class);
58     FileOutputFormat.setOutputPath(job, new Path(args[0]));
59     FileOutputFormat.setOutputPath(job, new Path(args[1]));
60     System.exit(job.waitForCompletion(true) ? 0 : 1);
61 }
62 }
```

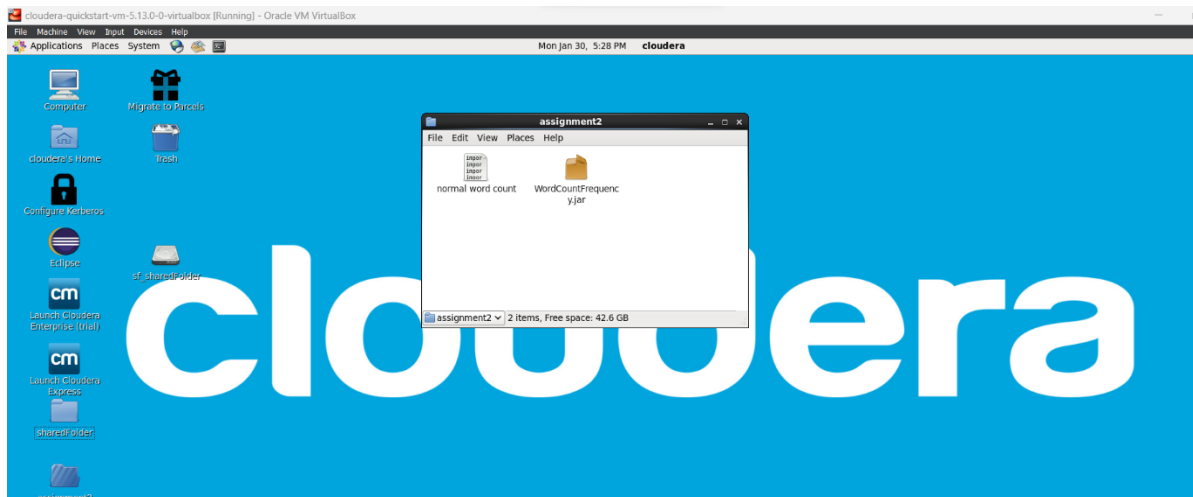
Added the login in IntSumReducer class. The purpose of this class is to count the number of occurrences of words in the file. Here an if condition is being added after 42 to check the count of word. If the count of the word is a multiple of 3, then the if condition in line 43 gets succeeded and the lines inside the if condition will gets executed. These lines will appended the word and its frequency to the the text file that will be generated at the end.

Exporting the project and creating a jar file

By right clicking on the project name on the right navigation and selecting properties in the menu, a new pop up for jar export will appear on the screen. Select java jar file export and then provide the location for with name of the jar file and click on finish.



A new jar file at the prescribed location is generated as shown in the below figure.

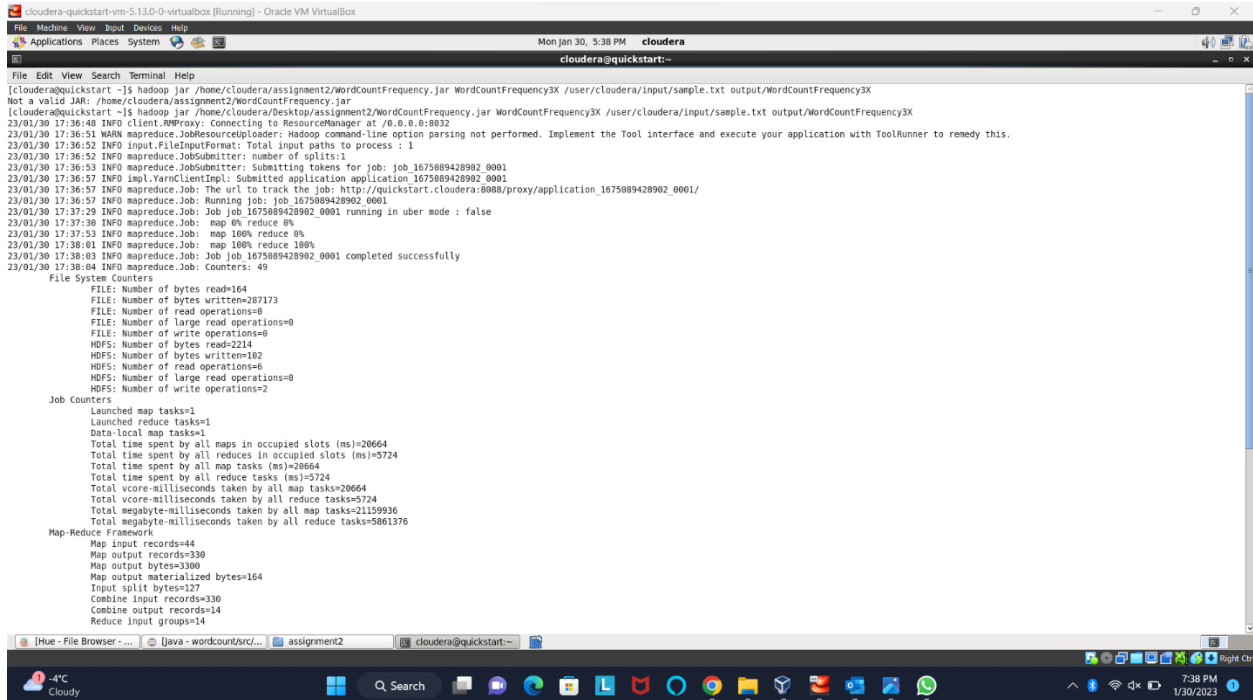


Running the Map Reduce job in the terminal.

To run the jar file in the terminal the following command is being used.

`hadoop jar <jar file location><space> <class name of the java file in jar><space><input file path><space><output file path>`

On giving the command in the above format, the java program along with hadoop will pick the input sample.txt file from the given input file path and runs the map reduce job in that file and generates an out file at the given output location.



```
cloudera@quickstart:~$ hadoop jar /home/cloudera/assignment2/WordCountFrequency.jar WordCountFrequency3X /user/cloudera/input/sample.txt output/WordCountFrequency3X
Not a valid JAR: /home/cloudera/assignment2/WordCountFrequency.jar
cloudera@quickstart:~$ hadoop jar /home/cloudera/Desktop/assignment2/WordCountFrequency.jar WordCountFrequency3X /user/cloudera/input/sample.txt output/WordCountFrequency3X
23/01/30 17:36:48 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/01/30 17:36:51 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
23/01/30 17:36:52 INFO Input.FileInputFormat: Total input paths to process : 1
23/01/30 17:36:52 INFO mapreduce.JobSubmitter: number of splits:1
23/01/30 17:36:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1675089428902_0001
23/01/30 17:36:57 INFO Impl.VanClientImpl: Submitted application application_1675089428902_0001
23/01/30 17:36:57 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1675089428902_0001/
23/01/30 17:36:57 INFO mapreduce.Job: Running job: job_1675089428902_0001
23/01/30 17:37:29 INFO mapreduce.Job: Job job_1675089428902_0001 running in uber mode : false
23/01/30 17:37:30 INFO mapreduce.Job: map 0% reduce 0%
23/01/30 17:37:53 INFO mapreduce.Job: map 100% reduce 0%
23/01/30 17:38:01 INFO mapreduce.Job: map 100% reduce 100%
23/01/30 17:38:03 INFO mapreduce.Job: Job job_1675089428902_0001 completed successfully
23/01/30 17:38:04 INFO mapreduce.Job: Counters: 49

File System Counters
  FILE: Number of bytes read=164
  FILE: Number of bytes written=287173
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=2214
  HDFS: Number of bytes written=182
  HDFS: Number of read operations=0
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2

Job Counters
  Launched map tasks=1
  Launched reduce tasks=1
  Data-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=20664
  Total time spent by all reduces in occupied slots (ms)=5724
  Total time spent by all map tasks (ms)=20664
  Total time spent by all reduce tasks (ms)=5724
  Total vcore-milliseconds taken by all map tasks=20664
  Total vcore-milliseconds taken by all reduce tasks=5724
  Total megabyte-milliseconds taken by all map tasks=21159936
  Total megabyte-milliseconds taken by all reduce tasks=5861376

Map-Reduce Framework
  Map input records=44
  Map output records=330
  Map output bytes=3306
  Map output materialized bytes=164
  Input split bytes=127
  Combine input records=330
  Combine output records=14
  Reduce input groups=14
```

The below picture is the visualization of the newly generated file in hue.

The screenshot displays the Hue File Browser interface within a Mozilla Firefox browser window. The address bar shows the URL: `quickstart.cloudera:8888/hue/filebrowser/view--user/cloudera/output/WordCountFrequency3X/part-r-00000`. The Hue interface includes a sidebar with navigation options like Apps, Editor, Dashboard, Scheduler, Browsers, Documents, Files, Tables, Indexes, Jobs, HBase, Security, and Sqoop. The main content area shows the file `WordCountFrequency3X / part-r-00000` with a list of actions: View as binary, Edit file, Download, View file location, and Refresh. Below these actions, a table displays the file's metadata and content.

File Name	Size	Last modified
WordCountFrequency3X / part-r-00000	102 B	01/31/2023 1:37 AM

The file content is displayed as a word frequency list:

Word	Count
Cooke	3
Dancer,	3
KILLER:	3
MOTIVE:	3
Nlight	3
WEAPON:	3
a	6
an	3
and	9
he	3
him	3
killed	3
of	9
who	3

The bottom of the screenshot shows the Windows taskbar with the system clock at 7:39 PM on 1/30/2023.

Task 2

Finding all the words which ends with letter 'S' and their frequency.

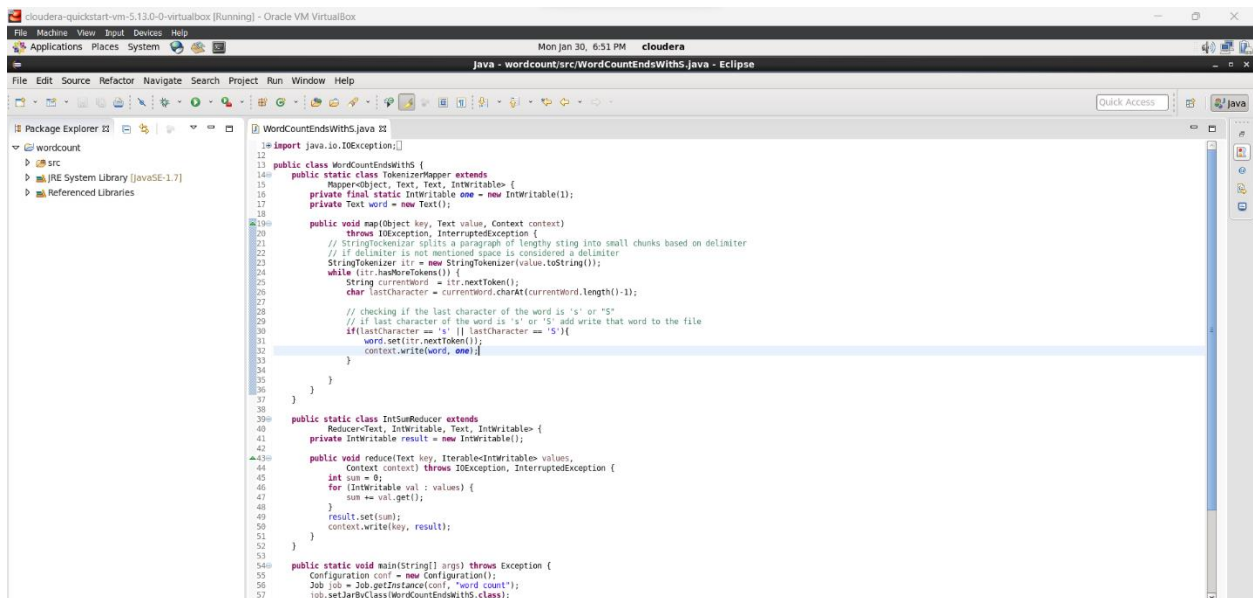
In the provided java code, all the words in sample.txt that are ending with letter words are found out and are added to a file and save in hadoop.

In the logic, StringTokenizer is used to break down the paragraphs in the file to words.

In line 25, checking availability of the word, if the word is available, the condition will turn true.

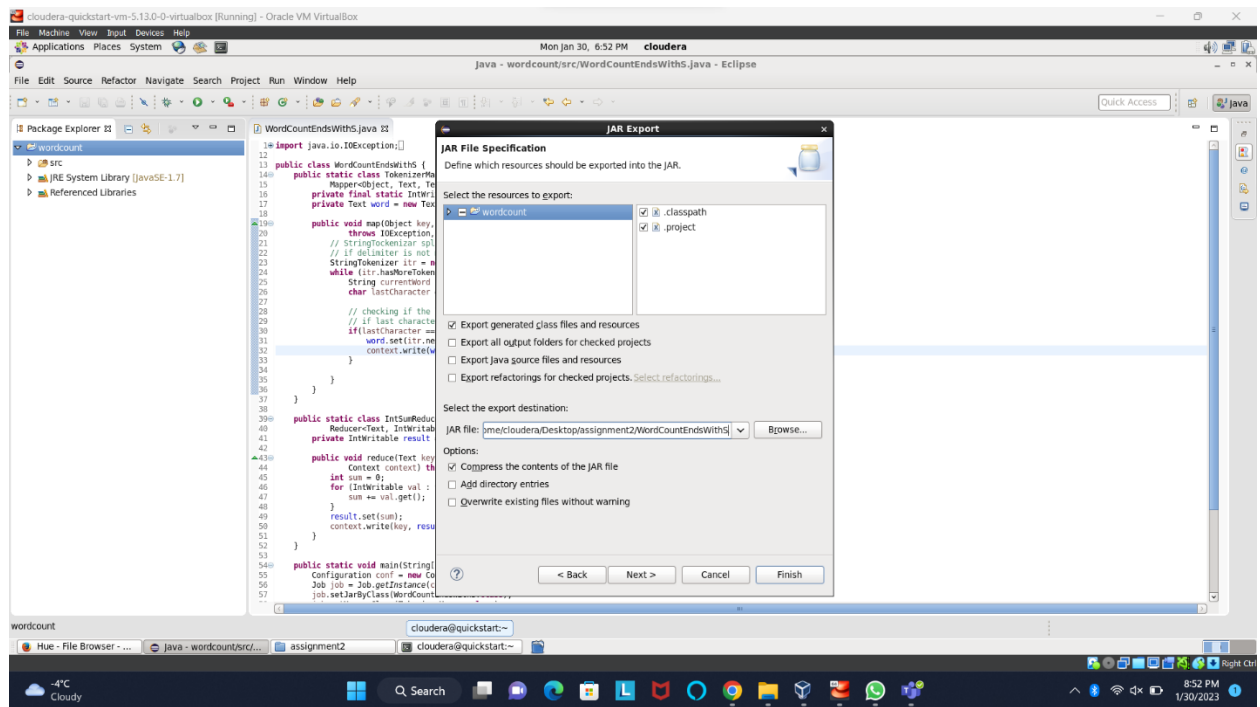
In line 26, getting the last letter in the word using `charAt(lengthOfWord -1)`

In line 30, checking if the letter is a 's' or 'S'. if they match, those words are sent to reduces and added to the file.



```
1 import java.io.IOException;
2
3 public class WordCountEndsWith {
4     public static class TokenizerMapper extends
5         Mapper<Object, Text, Text, IntWritable> {
6         private final static IntWritable one = new IntWritable(1);
7         private Text word = new Text();
8
9         public void map(Object key, Text value, Context context)
10             throws IOException, InterruptedException {
11             // StringTokenizer splits a paragraph of lengthy string into small chunks based on delimiter
12             // if delimiter is not mentioned space is considered a delimiter
13             StringTokenizer itr = new StringTokenizer(value.toString());
14             while (itr.hasMoreTokens()) {
15                 String currentword = itr.nextToken();
16                 char lastCharacter = currentword.charAt(currentword.length()-1);
17
18                 // checking if the last character of the word is 's' or 'S'
19                 // if last character of the word is 's' or 'S' add write that word to the file
20                 if(lastCharacter == 's' || lastCharacter == 'S'){
21                     word.set(itr.nextToken());
22                     context.write(word, one);
23                 }
24             }
25         }
26     }
27
28     public static class IntSumReducer extends
29         Reducer<Text, IntWritable, Text, IntWritable> {
30         private IntWritable result = new IntWritable();
31
32         public void reduce(Text key, Iterable<IntWritable> values,
33             Context context) throws IOException, InterruptedException {
34             int sum = 0;
35             for (IntWritable val : values) {
36                 sum += val.get();
37             }
38             result.set(sum);
39             context.write(key, result);
40         }
41     }
42
43     public static void main(String[] args) throws Exception {
44         Configuration conf = new Configuration();
45         Job job = Job.getInstance(conf, "word count");
46         job.setJarByClass(WordCountEndsWith.class);
47     }
48 }
```

Exported the jar file and ran in terminal.



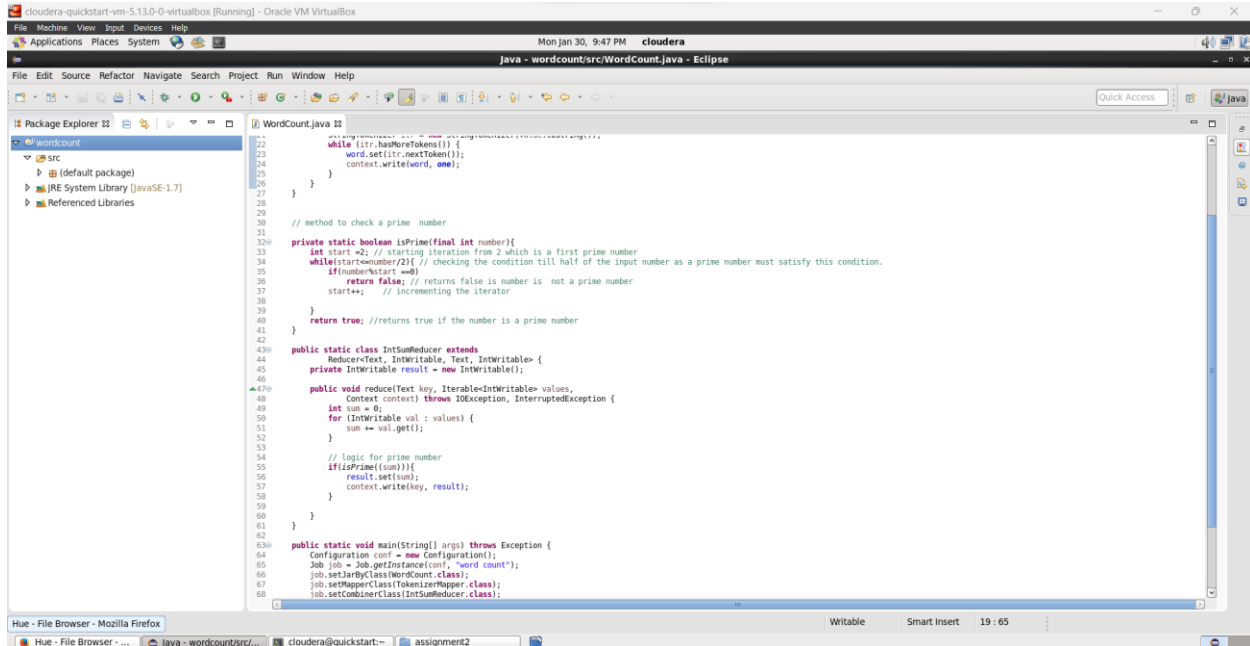
Task 3

Finding the words that are repeated prime numbers of times.

In the below program, logic is being added in the IntSumReducer class.

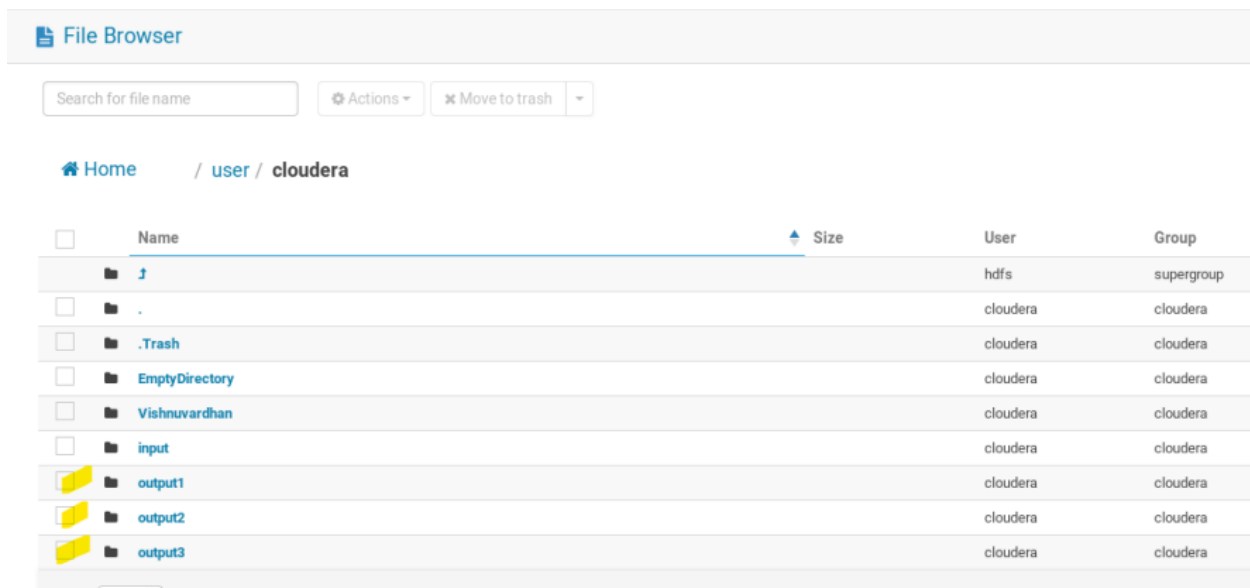
In line 55, checking if the sum is a prime number. If it is a prime number, the word and its frequency will be added to the output file.

This file is exported to a jar and ran in terminal to generate the output file in hadoop.



```
22 while (itr.hasMoreTokens()) {
23     word.set(itr.nextToken());
24     context.write(word, one);
25 }
26 }
27
28 // method to check a prime number
29
30 private static boolean isPrime(final int number){
31     int start=2; // starting iteration from 2 which is a first prime number
32     while(start<number/2){ // checking the condition till half of the input number as a prime number must satisfy this condition.
33         if(number%start ==0)
34             return false; // returns false is number is not a prime number
35         start++; // incrementing the iterator
36     }
37     return true; //returns true if the number is a prime number
38 }
39
40 public static class IntSumReducer extends
41     Reducer<Text, IntWritable, Text, IntWritable> {
42     private IntWritable result = new IntWritable();
43
44     public void reduce(Text key, Iterable<IntWritable> values,
45         Context context) throws IOException, InterruptedException {
46         int sum = 0;
47         for (IntWritable val : values) {
48             sum += val.get();
49         }
50         // logic for prime number
51         if(isPrime(sum)){
52             result.set(sum);
53             context.write(key, result);
54         }
55     }
56 }
57
58 public static void main(String[] args) throws Exception {
59     Configuration conf = new Configuration();
60     Job job = Job.getInstance(conf, "word count");
61     job.setJarByClass(WordCount.class);
62     job.setMapperClass(TokensMapper.class);
63     job.setCombinerClass(IntSumReducer.class);
64 }
```

Generated output files for all the 3 tasks.



<input type="checkbox"/>	Name	Size	User	Group
<input type="checkbox"/>	1		hdfs	supergroup
<input type="checkbox"/>	.		cloudera	cloudera
<input type="checkbox"/>	.Trash		cloudera	cloudera
<input type="checkbox"/>	EmptyDirectory		cloudera	cloudera
<input type="checkbox"/>	Vishnuvardhan		cloudera	cloudera
<input type="checkbox"/>	input		cloudera	cloudera
<input checked="" type="checkbox"/>	output1		cloudera	cloudera
<input checked="" type="checkbox"/>	output2		cloudera	cloudera
<input checked="" type="checkbox"/>	output3		cloudera	cloudera