Hadoop MapReduce program to count occurrences of a word and give Top N words.

Dataset Overview

Please download one dataset of your choice from Amazon review data using the following URL:

- http://jmcauley.ucsd.edu/data/amazon/Links to an external site.

Links to an external site.

Data source:
I choose All beauty product category from link below
https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/#complete-data

Data description:
It is the reviews provided by different customers for different beauty products on amazon platform. I am using 2018 version of this dataset. It has 371345 records i.e reviews. The reviews are stored in json file and the file size is 167 MB.

It has following attributes:

- overall : overall rating of product. Range is 1 to 5
- vote: number of votes
- verified: verified or not(values-True/False)
- reviewTime: time of the review
- reviewerID: reviewer ID
- asin: product ID
- reviewerName : reviewer name
- reviewText: review provided by the reviewer.
- summary: summary of review
- unixReviewTime: time of the review in unix time

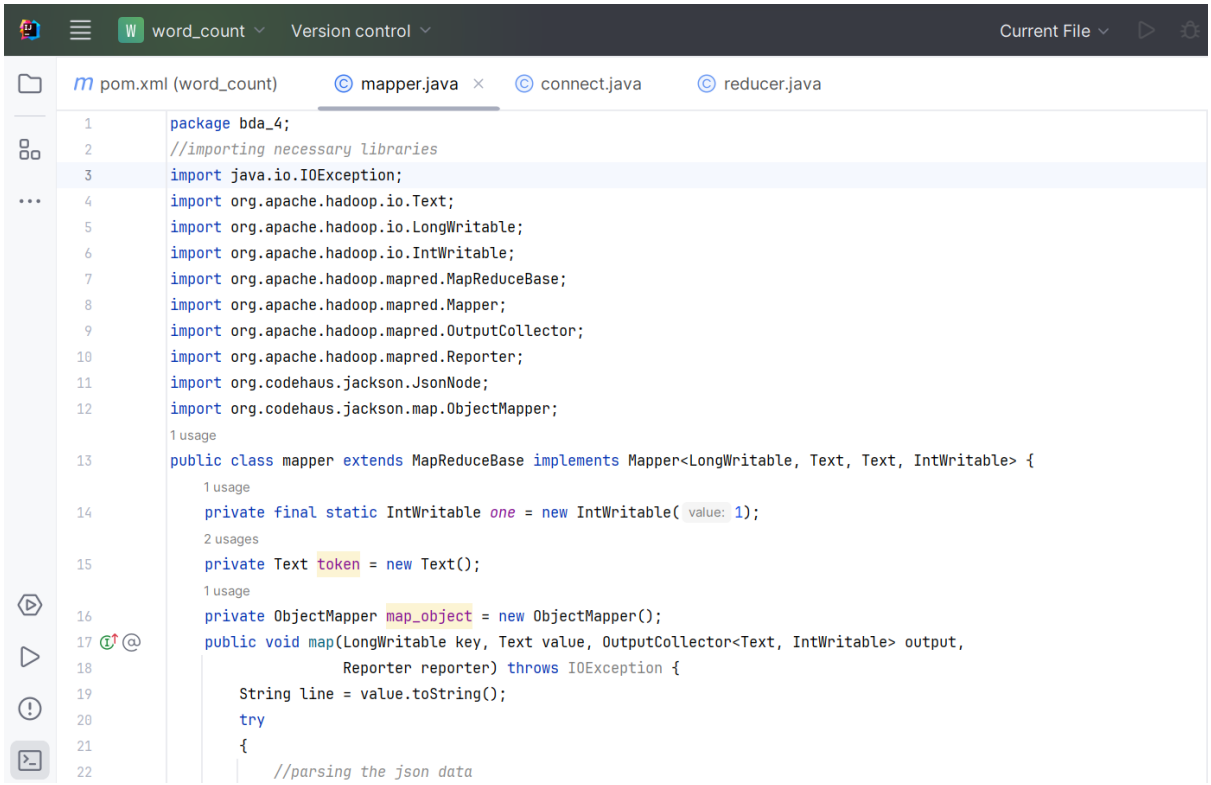
Task 1: Word Count

1. Implementing a MapReduce program to count the occurrences of each word in the dataset.
2. Ensuring that the program removes punctuation and converts all words to lowercase.
3. Output the word count in the format: <word> <count>.
4. And a brief explanation of the MapReduce workflow for the Word Count program.

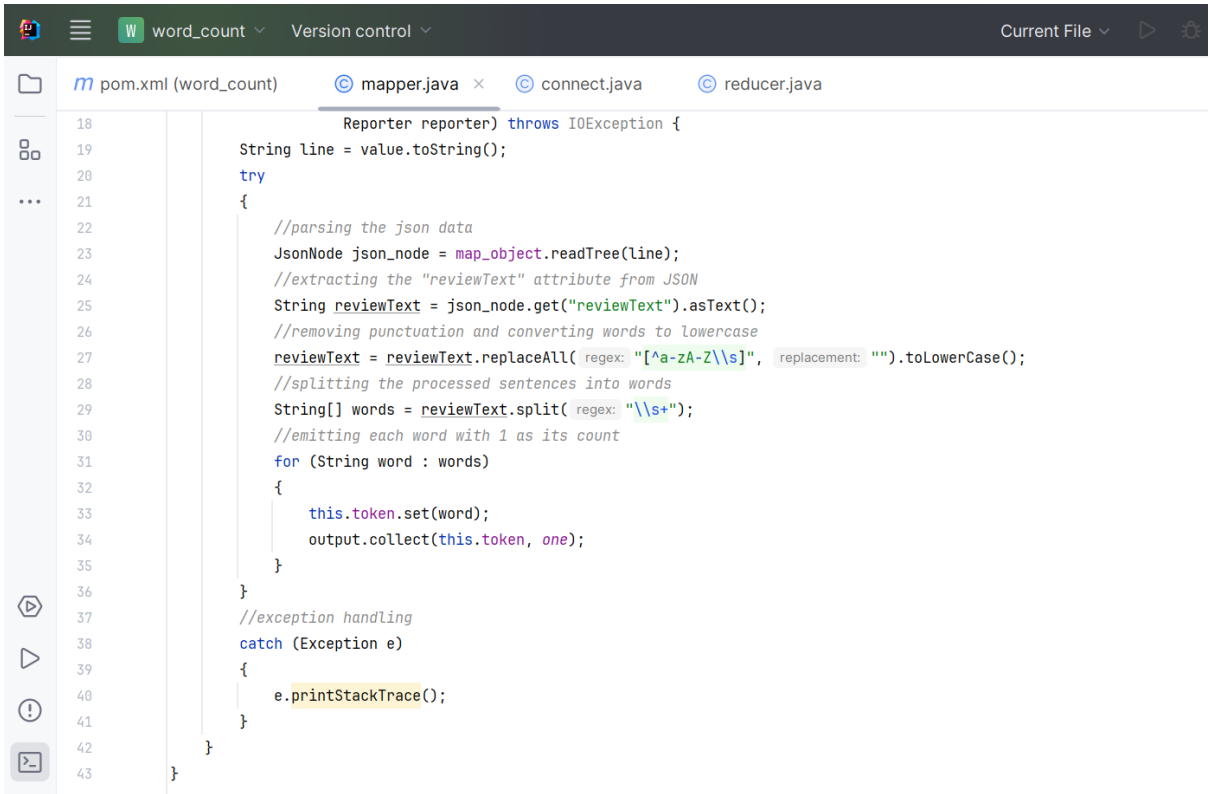
I used Intelli J IDEA IDE to achieve task 1 and task 2 as we can add dependencies directly using Maven.

Implementation

Mapper Class




```java
package bda_4;
//importing necessary libraries
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.ObjectMapper;
1 usage
public class mapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    1 usage
    private final static IntWritable one = new IntWritable( value: 1);
    2 usages
    private Text token = new Text();
    1 usage
    private ObjectMapper map_object = new ObjectMapper();
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
        String line = value.toString();
        try
        {
            //parsing the json data
```




```java
                    Reporter reporter) throws IOException {
        String line = value.toString();
        try
        {
            //parsing the json data
            JsonNode json_node = map_object.readTree(line);
            //extracting the "reviewText" attribute from JSON
            String reviewText = json_node.get("reviewText").asText();
            //removing punctuation and converting words to lowercase
            reviewText = reviewText.replaceAll( regex: "[^a-zA-Z\\s]", replacement: "").toLowerCase();
            //splitting the processed sentences into words
            String[] words = reviewText.split( regex: "\\s+");
            //emitting each word with 1 as its count
            for (String word : words)
            {
                this.token.set(word);
                output.collect(this.token, one);
            }
        }
        //exception handling
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Reducer Class

word_count Version control

pom.xml (word_count) | mapper.java | connect.java | reducer.java

```java
package bda_4;
////importing necessary libraries
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
1 usage
public class reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
                       Reporter reporter) throws IOException {
        //initial count of word is set to zero
        int frequency=0;
        //iterating and combining the values of the word to get frequency of that word in total words
        while (values.hasNext())
        {
            frequency+=values.next().get();
        }
        //word wits its frequency
        output.collect(key,new IntWritable(frequency));
    }
}
```

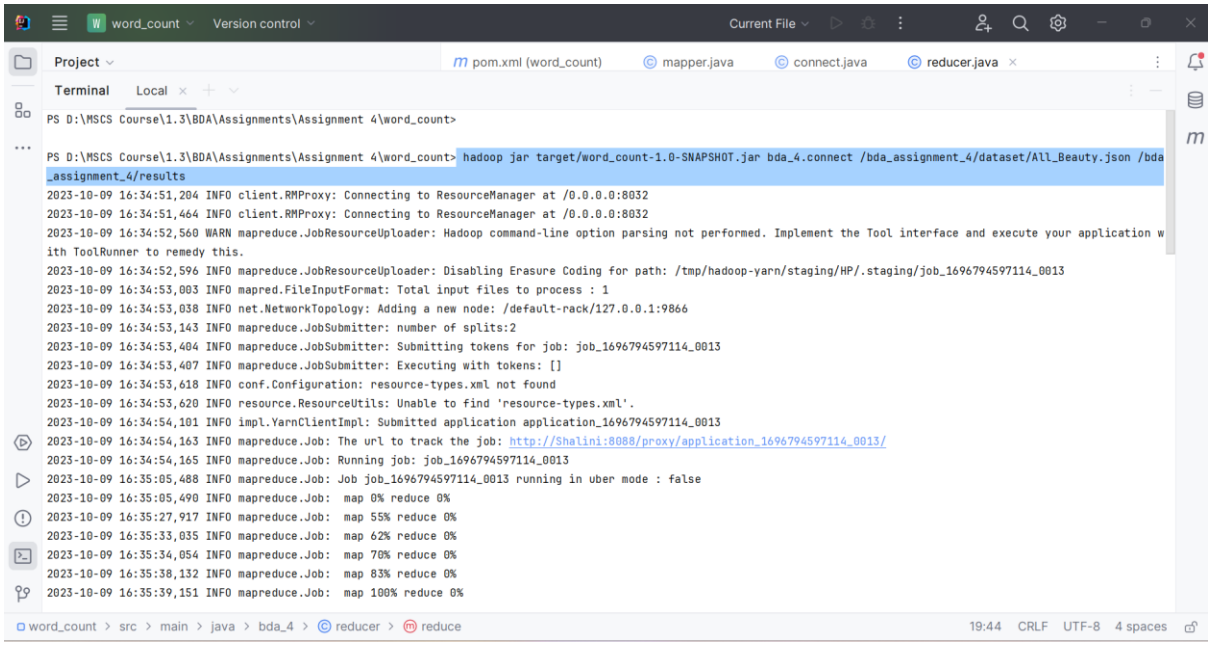
Connect Class

word_count Version control

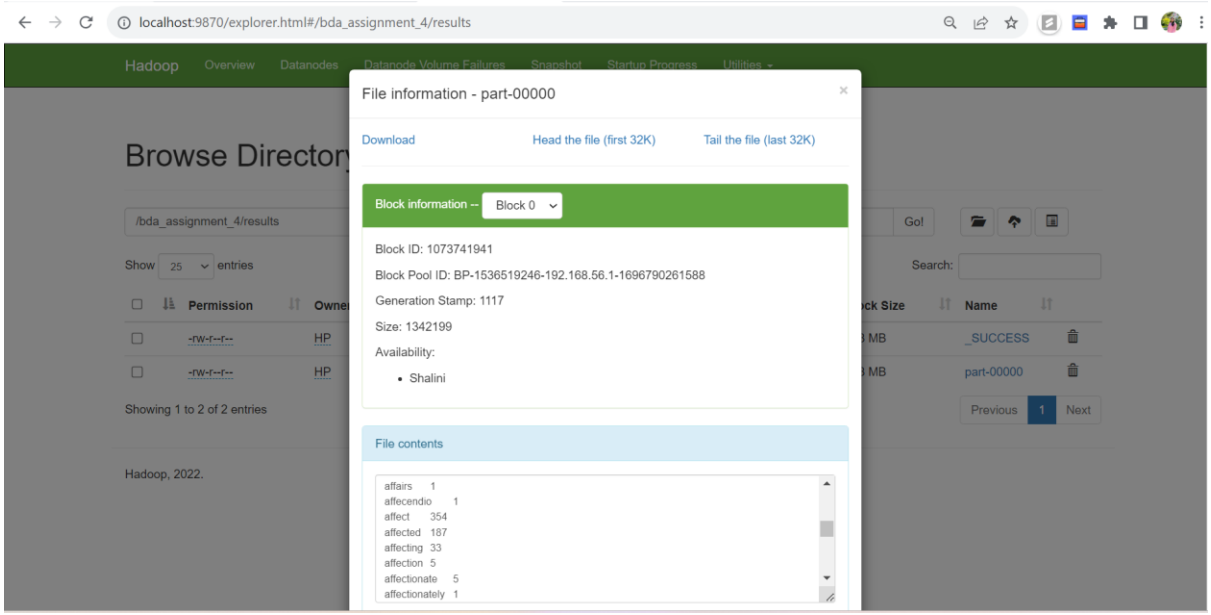
pom.xml (word_count) | mapper.java | connect.java | reducer.java

```java
package bda_4;
//importing necessary libraries
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class connect
{
    public static void main(String[] args) throws IOException
    {
        //configuring job
        JobConf job_conf = new JobConf(connect.class);
        job_conf.setJobName("word_count");
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);
        job_conf.setMapperClass(mapper.class);
        job_conf.setReducerClass(reducer.class);
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);
        //taking argument 1 as input
        FileInputFormat.setInputPaths(job_conf,new Path(args[0]));
        //taking argument 2 as output
        FileOutputFormat.setOutputPath(job_conf,new Path(args[1]));
        //running job
        JobClient.runJob(job_conf);
    }
}
```

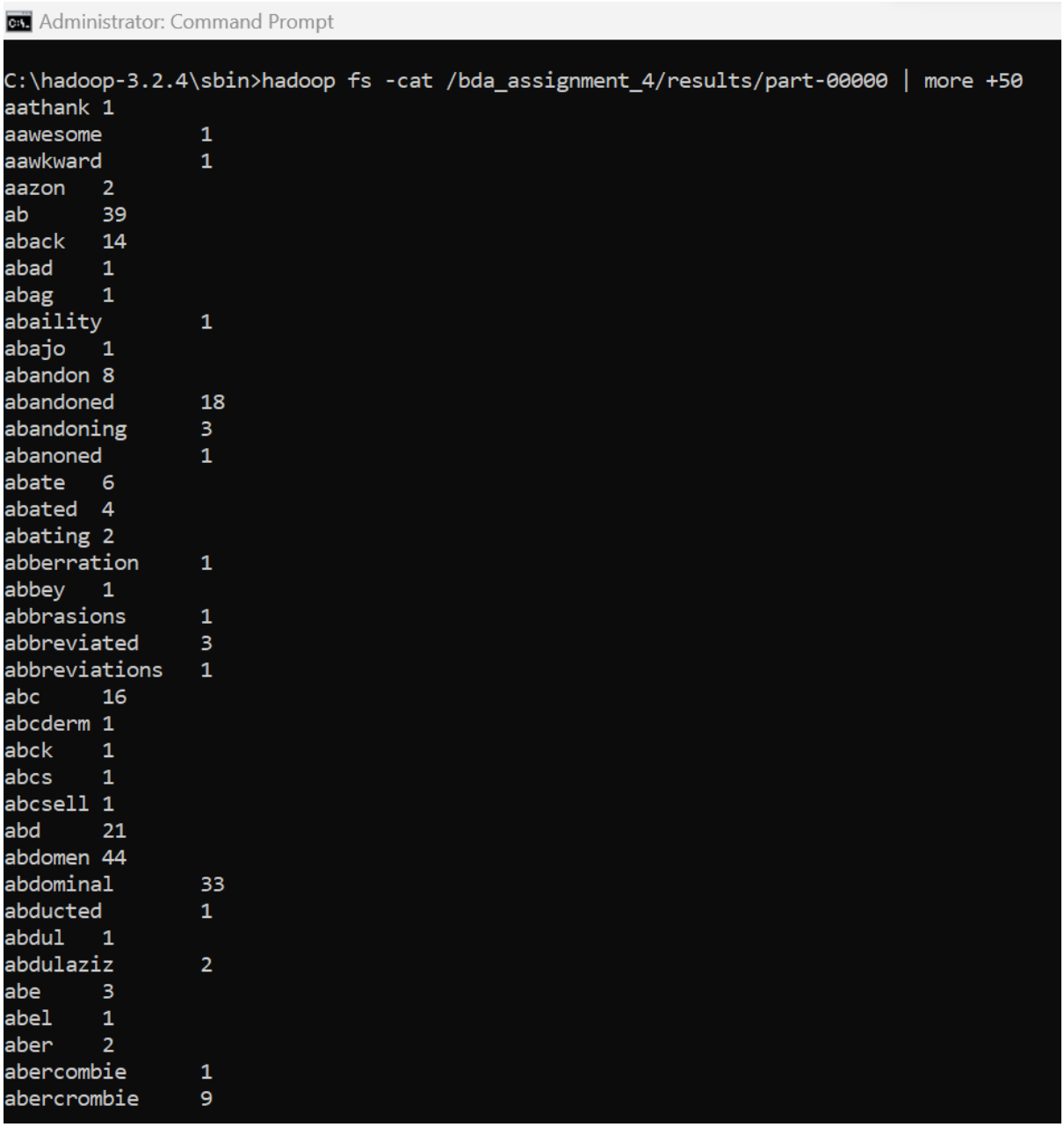
Output:



Seeing results in UI



Seeing results through cmd: I returned only first 50 due to huge size



Program workflow:

Prior to executing program: I have added dependencies in xml file. I have uploaded All_Beauty.json file to hdfs using commands.

Coming to program's workflow

Mapping phase(mapper java class): It takes All_Beauty.json file as input and extracts reviewText attribute from all records. We then process sentences in reviewText by removing punctuation and convert it lower cases. We then split sentences based on spaces. After splitting we have tokens. Each token is printed with its count as 1.

Shuffling phase: Hadoop groups and sort the data obtained from mappers and send it to reducer class.

Reducer phase(reducer class file). This program takes sorted tokens as input and combines the sum of all the values which has same key and result in key value pairs where word is the key and corresponding frequency is the value.

Connect class: In this class, I defined input to the program and the output it should obtain. It also can be seen as center of all classes. It specifies mapper and reducer class, formats. Basically, it configures the job to be executed by Hadoop.

Executing program:
Once all the class files are coded, I generated the jar file for my program and executed the project in terminal using following command:

Cmd to execute the word_count algorithm:

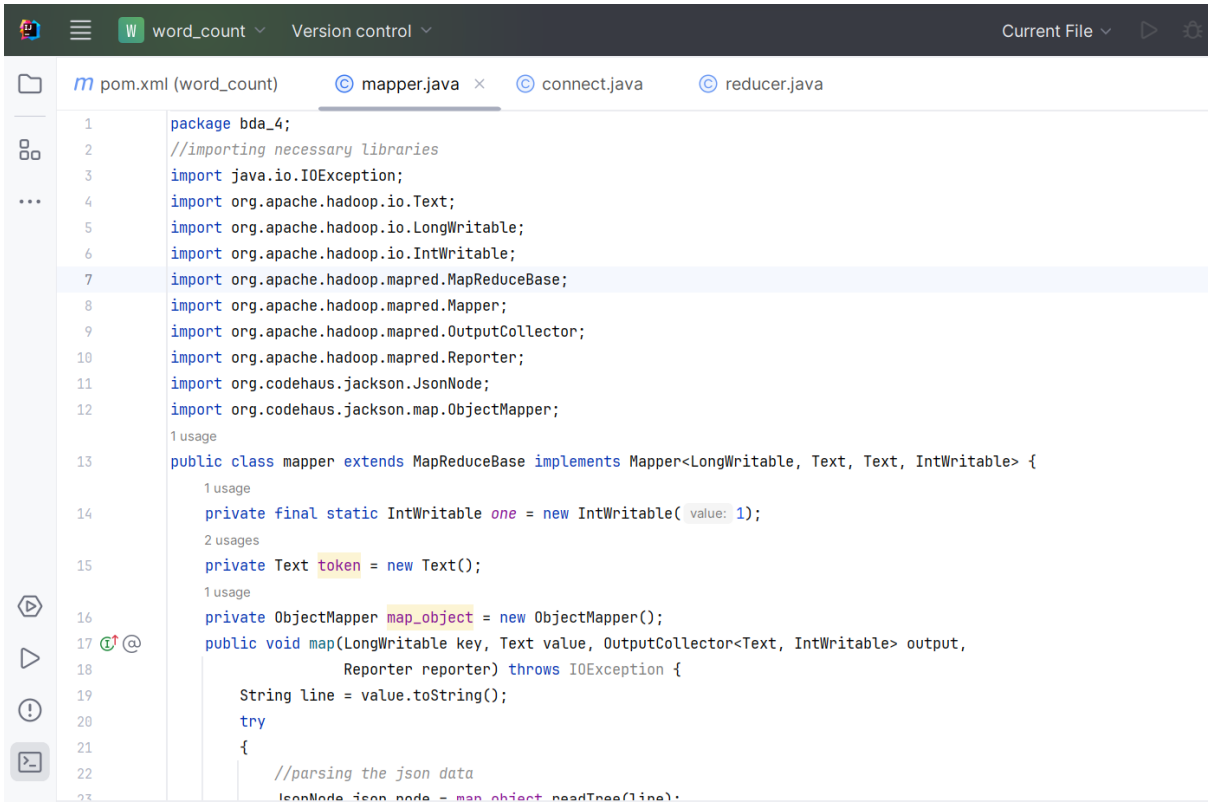
hadoop jar target/word_count-1.0-SNAPSHOT.jar bda_4.connect /bda_assignment_4/dataset/All_Beauty.json /bda_assignment_4/results

Task 2: Top N Words

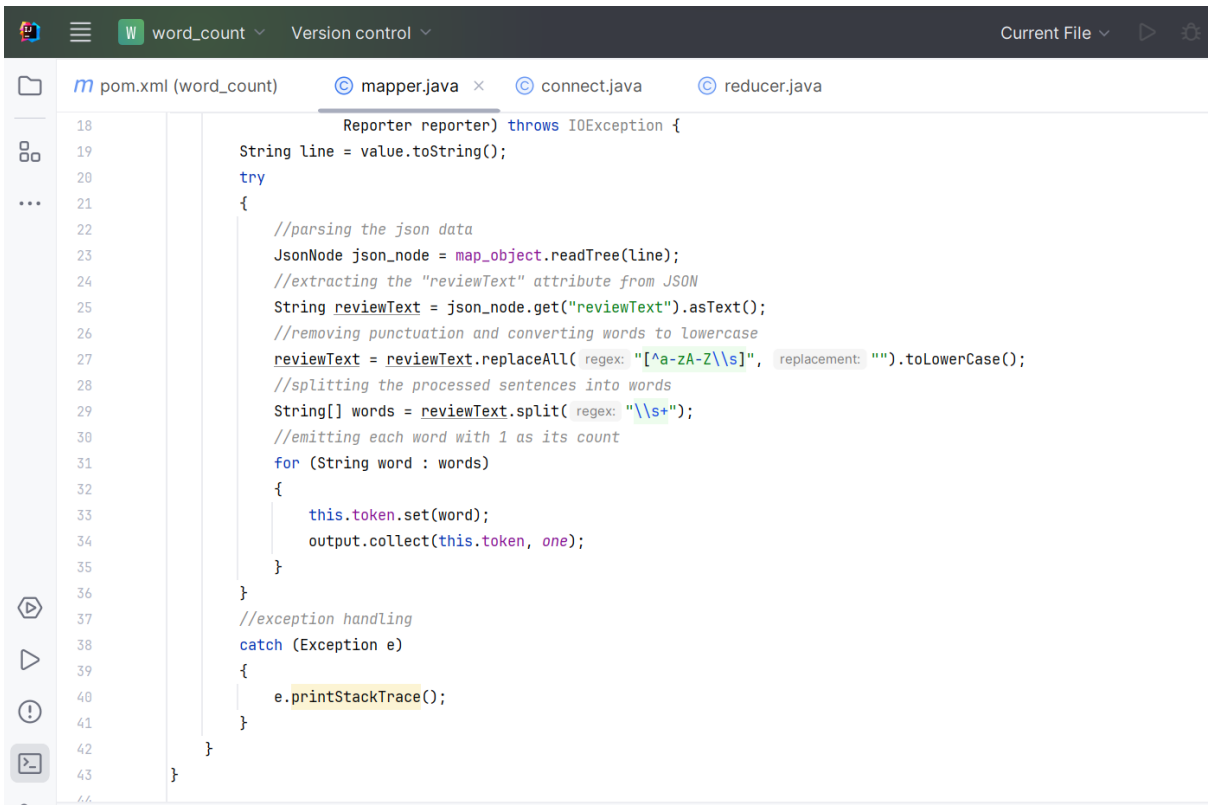
1. Extending the Word Count program to find the top N most frequently occurring words in the dataset.
2. Allowing the user to specify the value of N.
3. Output the top N words and their counts in descending order.
4. Repeat section 1 while using "combiner" in reducer and compare the results with the results in section 1. Does the "combiner" change the results in this case?
5. And a brief explanation of how you modified the Word Count program to accomplish this task.

Implementation

Mapper Class: The mapper class is same as task 1 and not altered.




```java
package bda_4;
//importing necessary libraries
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.ObjectMapper;
1 usage
public class mapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    1 usage
    private final static IntWritable one = new IntWritable( value: 1);
    2 usages
    private Text token = new Text();
    1 usage
    private ObjectMapper map_object = new ObjectMapper();
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
        String line = value.toString();
        try
        {
            //parsing the json data
            JsonNode json_node = map_object.readTree(line);
```




```java
                    Reporter reporter) throws IOException {
        String line = value.toString();
        try
        {
            //parsing the json data
            JsonNode json_node = map_object.readTree(line);
            //extracting the "reviewText" attribute from JSON
            String reviewText = json_node.get("reviewText").asText();
            //removing punctuation and converting words to lowercase
            reviewText = reviewText.replaceAll( regex: "[^a-zA-Z\\s]", replacement: "").toLowerCase();
            //splitting the processed sentences into words
            String[] words = reviewText.split( regex: "\\s+");
            //emitting each word with 1 as its count
            for (String word : words)
            {
                this.token.set(word);
                output.collect(this.token, one);
            }
        }
        //exception handling
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Reducer Class

word_count Version control Current File

pom.xml (word_count) mapper.java connect.java reducer.java

```java
package bda_4;
import java.io.IOException;
import java.util.Comparator;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.JobConf;
2 usages
public class reducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    2 usages
    private int N;
    //TreeMap to store word and its frequency
    5 usages
    private TreeMap<Integer, Text> top_N_words;
    2 usages
    private OutputCollector<Text, IntWritable> result;
    public void configure(JobConf job)
    {
        //getting n value from configuration, I gave default value as 3
        N = job.getInt( name: "top_N", defaultValue: 3);
        // new Treemap for the tokens
        top_N_words = new TreeMap<>(Comparator.reverseOrder());
    }

    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,
```

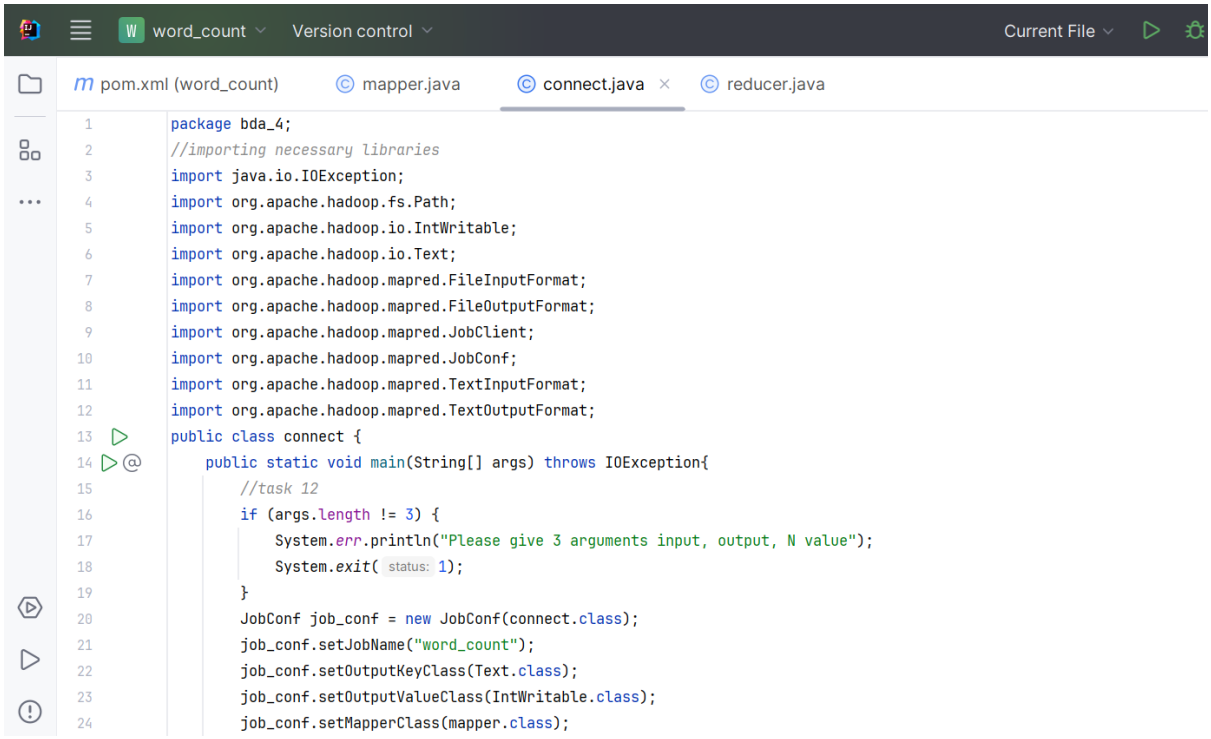
word_count Version control Current File

pom.xml (word_count) mapper.java connect.java reducer.java

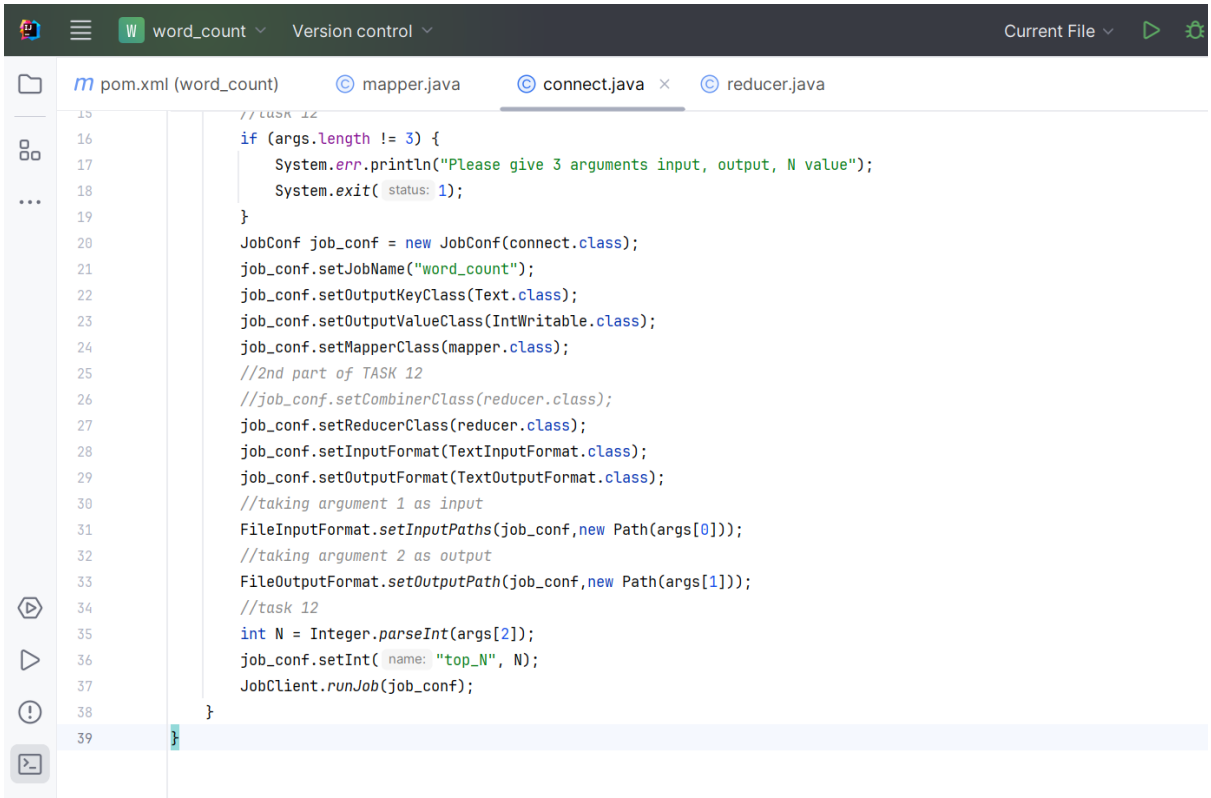
```java
        //getting n value from configuration, I gave default value as 3
        N = job.getInt( name: "top_N", defaultValue: 3);
        // new Treemap for the tokens
        top_N_words = new TreeMap<>(Comparator.reverseOrder());
    }

    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
        int frequency = 0;
        while (values.hasNext()) {
            frequency += values.next().get();
        }
        //adding word and its frequency to the TreeMap
        top_N_words.put(frequency, new Text(key.toString()));
        //keeping only top N words
        if (top_N_words.size() > N)
        {
            top_N_words.pollLastEntry();
        }
        //storing the OutputCollector for later use in the close method as I can't use output directly in close method
        result = output;
    }
    public void close() throws IOException
    {
        //emitting the top N words in descending order
        for (Map.Entry<Integer, Text> entry : top_N_words.entrySet())
        {
            result.collect(entry.getValue(), new IntWritable(entry.getKey()));
        }
    }
}
```

Connect Class

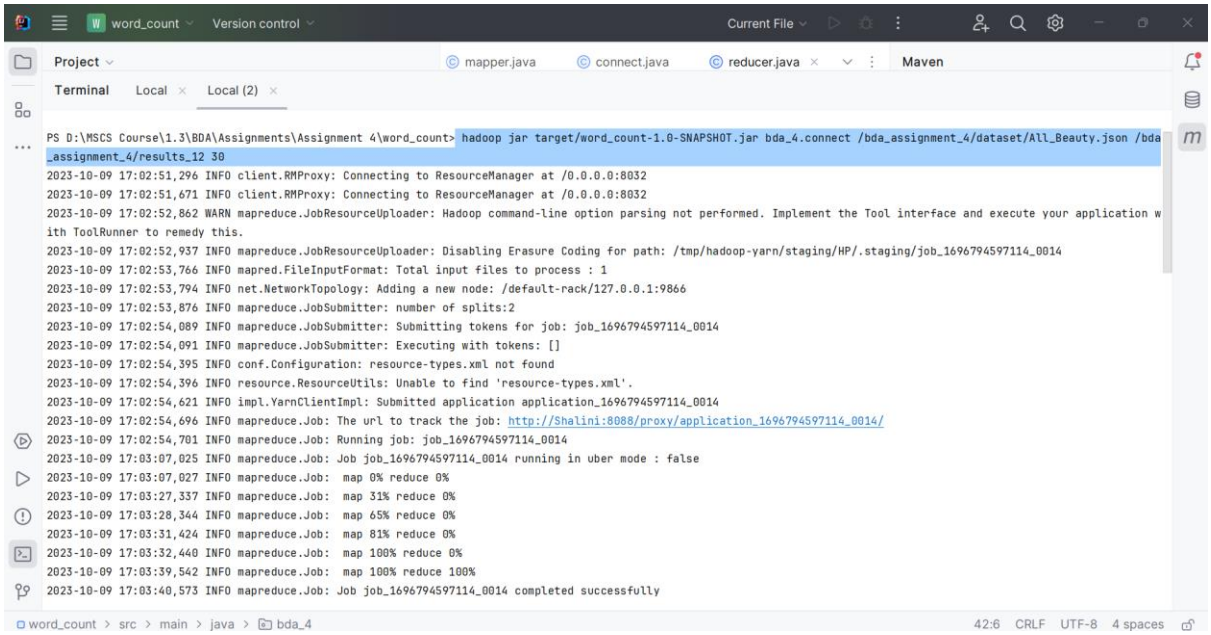



```java
package bda_4;
//importing necessary libraries
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class connect {
    public static void main(String[] args) throws IOException{
        //task 12
        if (args.length != 3) {
            System.err.println("Please give 3 arguments input, output, N value");
            System.exit( status: 1);
        }
        JobConf job_conf = new JobConf(connect.class);
        job_conf.setJobName("word_count");
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);
        job_conf.setMapperClass(mapper.class);
```

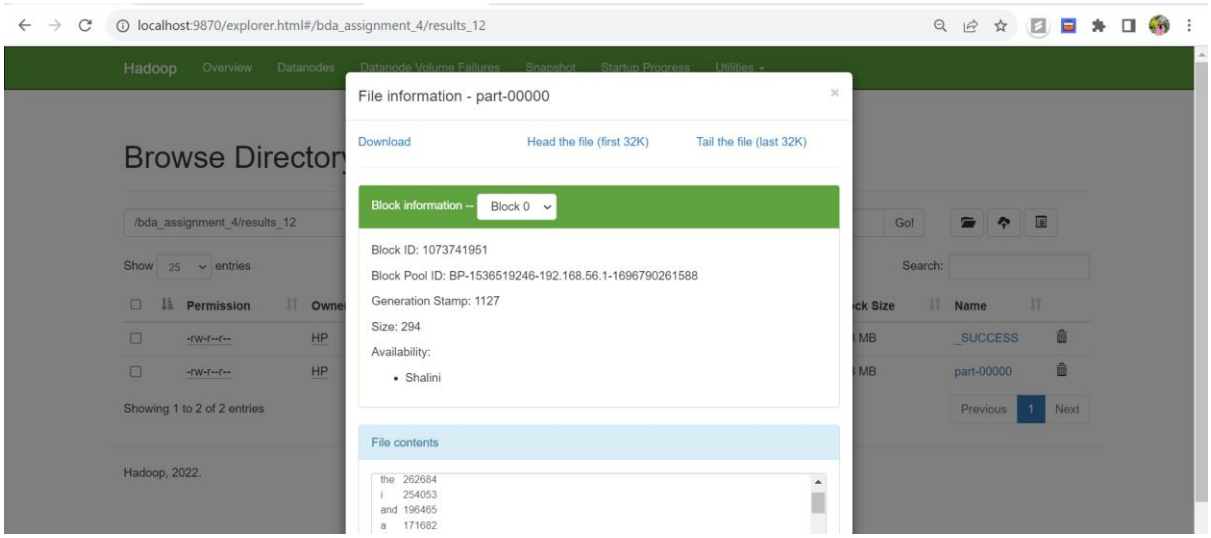



```java
        //task 12
        if (args.length != 3) {
            System.err.println("Please give 3 arguments input, output, N value");
            System.exit( status: 1);
        }
        JobConf job_conf = new JobConf(connect.class);
        job_conf.setJobName("word_count");
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);
        job_conf.setMapperClass(mapper.class);
        //2nd part of TASK 12
        //job_conf.setCombinerClass(reducer.class);
        job_conf.setReducerClass(reducer.class);
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);
        //taking argument 1 as input
        FileInputFormat.setInputPaths(job_conf,new Path(args[0]));
        //taking argument 2 as output
        FileOutputFormat.setOutputPath(job_conf,new Path(args[1]));
        //task 12
        int N = Integer.parseInt(args[2]);
        job_conf.setInt( name: "top_N", N);
        JobClient.runJob(job_conf);
    }
}
```

Output:



Since we were not asked to do complete text preprocessing of reviewText, results contain words like 'a', 'the' etc



Modifications on Word Count to achieve Task 2 - Top N Words:

I have modified the reducer and connect java class of word count - task 1 to achieve task 2.

- Connect class: It has N integer to take 3rd argument from the user input(input - 1st argument, output-2nd argument, N – 3rd argument)
- Reducer class: It has same functionality but in addition I used a TreeMap with a custom comparator that sorts the entries based on counts in descending order. This allows to efficiently keep track of the top N words and their frequencies. The close method emits these top N words in descending order of counts.

Cmd to execute the top N words algorithm: (30 – top 30 words)

hadoop jar target/word_count-1.0-SNAPSHOT.jar bda_4.connect /bda_assignment_4/dataset/All_Beauty.json /bda_assignment_4/results_12 30

Using Combiner:

The combiner runs on the output of the mapper class. It performs a local aggregation of word counts. It did not change the results but increases the efficiency of the algorithm.

We just add following line to connect java class file

Job_conf.setCombinerClass(reducer.class);