# Write a Map-reduce program to find the total amount of petrol in volume sold by every distributer.

package Asign1Of13;  
  
import java.io.IOException;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.mapreduce.Mapper;  
  
public class PetrolMapper extends Mapper<LongWritable, Text, Text, IntWritable>{  
  
    public void map(LongWritable key, Text value, Context context)  
    throws IOException, InterruptedException{  
        String values[] = value.toString().split(",");  
        int volume = Integer.parseInt(values[5]);  
        context.write(new Text(values[1]), new IntWritable(volume));  
    }  
      
}  
  
package Asign1Of13;  
  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
public class PetrolReducer extends Reducer<Text, IntWritable, Text, IntWritable>{  
      
    public void reduce(Text key, Iterable<IntWritable> value, Context context)  
    throws IOException, InterruptedException{  
        int sum =0;  
        for(IntWritable values: value){  
        sum = sum + values.get();  
        }  
        context.write(key, new IntWritable(sum));  
    }  
}  
  
  
package Asign1Of13;  
  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Partitioner;  
  
public class PetrolPartitioner extends Partitioner<Text, IntWritable> {  
      
    public int getPartition(Text key, IntWritable value, int numRedTasks){  
        String distributer = key.toString();  
        if(distributer.equalsIgnoreCase("Bharat"))  
            return 0;  
        if(distributer.equalsIgnoreCase("hindustan"))  
            return 1;  
        if(distributer.equalsIgnoreCase("reliance"))  
            return 2;  
        if(distributer.equalsIgnoreCase("shell"))  
            return 3;  
        else   
            return 4;  
    }  
}  
  
package Asign1Of13;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
import org.apache.hadoop.conf.Configured;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.io.compress.GzipCodec;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.fs.Path;  
  
public class PetrolDriver extends Configured implements Tool {  
    public static void main(String args[]) throws Exception{  
    ToolRunner.run(new Configuration(), new PetrolDriver(), args);  
    }  
    public int run(String args[]) throws Exception{  
        Configuration conf = new Configuration();  
        conf.set("mapreduce.map.output.compress", "true");  
        conf.set("mapreduce.map.output.compress.codec","org.apache.hadoop.io.compress.SnappyCodec");  
          
        Job job = Job.getInstance();  
        job.setJobName("Petrol Project");  
        job.setJarByClass(PetrolDriver.class);  
          
        job.setMapperClass(PetrolMapper.class);  
        job.setMapOutputKeyClass(Text.class);  
        job.setMapOutputValueClass(IntWritable.class);  
          
        job.setReducerClass(PetrolReducer.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
          
        job.setCombinerClass(PetrolReducer.class);  
        job.setPartitionerClass(PetrolPartitioner.class);  
        job.setNumReduceTasks(5);  
          
        FileOutputFormat.setCompressOutput(job, true);  
        FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);  
          
        job.setInputFormatClass(TextInputFormat.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
        return job.waitForCompletion(true)? 0 : 1;  
    }  
}

# Write a map-reduce program to find which are the top 10 distributers ID's for selling petrol and also display the amount of petrol sold in volume.

package Asign2Of13;  
  
import java.io.IOException;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.mapreduce.Mapper;  
  
public class PetrolMapper2 extends Mapper<LongWritable, Text, Text, IntWritable>{  
  
    public void map(LongWritable key, Text value, Context context)  
    throws IOException, InterruptedException{  
        String values[] = value.toString().split(",");  
        int volume = Integer.parseInt(values[5]);  
        context.write(new Text(values[0]), new IntWritable(volume));  
    }      
}  
  
package Asign2Of13;  
  
import java.io.IOException;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.HashMap;  
import java.util.LinkedHashMap;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Map;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
public class PetrolReducer2 extends Reducer<Text, IntWritable, Text, IntWritable>{  
    private Map<Text, IntWritable> countMap = new HashMap<>();  
    public void reduce(Text key, Iterable<IntWritable> value, Context context)  
    throws IOException, InterruptedException{  
        int sum =0;  
        for(IntWritable values: value){  
        sum = sum + values.get();  
        }  
        countMap.put(key, new IntWritable(sum));  
    }  
    protected void cleanup(Context context) throws IOException, InterruptedException{  
        Map<Text, IntWritable> sortedMap = sortByValues(countMap);  
        int counter = 0;  
        for (Text key : sortedMap.keySet()){  
            if(counter++ == 10){  
                break;  
            }  
            context.write(key, sortedMap.get(key));  
        }  
    }  
      
    private static <K extends Comparable, V extends Comparable> Map<K,V> sortByValues(Map<K,V> map){  
    List<Map.Entry<K,V>> entries = new LinkedList<Map.Entry<K,V>>(map.entrySet());  
    Collections.sort(entries, new Comparator<Map.Entry<K,V>>(){  
        public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2){  
            return o2.getValue().compareTo(o1.getValue());  
        }  
    });  
    Map<K,V> sortedMap = new LinkedHashMap<K,V>();  
    for(Map.Entry<K, V> entry : entries){  
        sortedMap.put(entry.getKey(), entry.getValue());  
    }  
    return sortedMap;  
    }  
}  
  
package Asign2Of13;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
import org.apache.hadoop.conf.Configured;  
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.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.fs.Path;  
  
public class PetrolDriver2 extends Configured implements Tool {  
    public static void main(String args[]) throws Exception{  
    ToolRunner.run(new Configuration(), new PetrolDriver2(), args);  
    }  
    public int run(String args[]) throws Exception{  
          
        Job job = Job.getInstance();  
        job.setJobName("Petrol Project 2");  
        job.setJarByClass(PetrolDriver2.class);  
          
        job.setMapperClass(PetrolMapper2.class);  
        job.setMapOutputKeyClass(Text.class);  
        job.setMapOutputValueClass(IntWritable.class);  
          
        job.setReducerClass(PetrolReducer2.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
          
        job.setNumReduceTasks(1);  
          
        job.setInputFormatClass(TextInputFormat.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
        return job.waitForCompletion(true)? 0 : 1;  
    }  
}