wc

```
import org.apache.hadoop.conf.Configured;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class wordcount extends Configured implements Tool {

 @Override
 public int run(String[] args) throws Exception {

 if(args.length<2)
 {
  System.out.println("Plz Give Input Output Directory Correctly");
  return -1;
 }

 JobConf conf = new JobConf(wordcount.class);

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

 conf.setMapperClass(wordmapper.class);
 conf.setReducerClass(wordreducer.class);

 conf.setMapOutputKeyClass(Text.class);
 conf.setMapOutputValueClass(IntWritable.class);
 conf.setOutputKeyClass(Text.class);
 conf.setOutputValueClass(IntWritable.class);




 JobClient.runJob(conf);
 return 0;
 }

 public static void main(String args[]) throws Exception
 {
  int exitcode = ToolRunner.run(new wordcount(), args);
  System.exit(exitcode);
 }

}
```

mpper

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;


public class wordmapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>
{
 public void map(LongWritable key, Text value,
   OutputCollector<Text, IntWritable> output, Reporter r) throws IOException
   {
    String s =value.toString();
    for(String word:s.split(" "))
    {
     if(word.length()>0)
     {
      output.collect(new Text(word), new IntWritable(1));
     }
    }
   }
}


reducer

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;


public class wordreducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable>
{
 public void reduce(Text key, Iterator<IntWritable> values,
   OutputCollector<Text, IntWritable> output, Reporter r)
   throws IOException {

 int count=0;
 while(values.hasNext())
 {
  IntWritable i= values.next();
  count+= i.get();
 }
 output.collect(key, new IntWritable(count));
 }
```

```
}
```

log

```
import java.io.*;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ProcessLogs extends Configured implements Tool {

 @Override
 public int run(String[] args) throws Exception {

 if(args.length<2)
 {
  System.out.println("Plz Give Input Output Directory Correctly");
  return -1;
 }

 JobConf conf = new JobConf(ProcessLogs.class);
 FileInputFormat.setInputPaths(conf,new Path(args[0]));
 FileOutputFormat.setOutputPath(conf, new Path(args[1]));
 conf.setMapperClass(LogMapper.class);
 conf.setReducerClass(LogReducer.class);
 conf.setMapOutputKeyClass(Text.class);
 conf.setMapOutputValueClass(IntWritable.class);
 conf.setOutputKeyClass(Text.class);
 conf.setOutputValueClass(IntWritable.class);
 JobClient.runJob(conf);
 return 0;
 }

 public static void main(String args[]) throws Exception
 {
  int exitcode = ToolRunner.run(new ProcessLogs(), args);
  System.exit(exitcode);
 }

}
```

mapper

```
import java.io.IOException;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;


public class LogMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>
{
 public void map(LongWritable key, Text value,
   OutputCollector<Text, IntWritable> output, Reporter r)
   throws IOException {

  String[] s = value.toString().split(" ");
  String ip = s[0];

  output.collect(new Text(ip), new IntWritable(1));
 }

}
```

reducer

```
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;


public class LogReducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable>
{


 public void reduce(Text key, Iterator<IntWritable> values,
   OutputCollector<Text, IntWritable> output, Reporter r)
   throws IOException {

 int count=0;
 while(values.hasNext())
 {
  IntWritable i= values.next();
  count+= i.get();
 }
 output.collect(key, new IntWritable(count));

 }

}
```

stock

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.io.FloatWritable;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class MaxClosePrice extends Configured implements Tool {

 public int run(String[] args) throws Exception {
  if(args.length<2)
  {
   System.out.println("Plz Give Input Output Directory Correctly");
   return -1;
  }
  JobConf conf = new JobConf(MaxClosePrice.class);
  FileInputFormat.setInputPaths(conf,new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));
  conf.setMapperClass(MaxClosePriceMapper.class);
  conf.setReducerClass(MaxClosePriceReducer.class);
  conf.setMapOutputKeyClass(Text.class);
  conf.setMapOutputValueClass(FloatWritable.class);
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(FloatWritable.class);
  JobClient.runJob(conf);
  return 0;
  }

 public static void main(String[] args) throws Exception {
  int exitcode = ToolRunner.run(new MaxClosePrice(), args);
  System.exit(exitcode);
  }
}
```

mapper

```
import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
```

```
import org.apache.hadoop.mapred.Mapper;

public class MaxClosePriceMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, Float
Writable>
{

 @Override
 public void map(LongWritable key, Text value,
   OutputCollector<Text, FloatWritable> output, Reporter r)
     throws IOException {

  String line = value.toString();
  String[] items = line.split(",");

  String stock = items[1];
  Float closePrice = Float.parseFloat(items[6]);

  output.collect(new Text(stock), new FloatWritable(closePrice));

 }
}
```

reducer

```
import java.io.IOException;
 import java.util.Iterator;

 import org.apache.hadoop.io.FloatWritable;
 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;

 public class MaxClosePriceReducer extends MapReduceBase implements Reducer<Text,FloatWritable,Text,Float
Writable>
 {

  @Override
  public void reduce(Text key, Iterator<FloatWritable> values,
    OutputCollector<Text, FloatWritable> output, Reporter r)
    throws IOException {

   float maxClosePrice = 0;

   //Iterate all and calculate maximum
   while (values.hasNext()) {
    FloatWritable i = values.next();
    maxClosePrice = Math.max(maxClosePrice, i.get());
   }

   //Write output
   output.collect(key, new FloatWritable(maxClosePrice));
  }
```

}