**import** java.io.IOException;

**import** java.util.regex.Pattern;

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.log4j.Logger;

**public** **class** WordCount **extends** Configured **implements** Tool {

**private** **static** **final** Logger ***LOG*** = Logger.*getLogger*(WordCount.**class**);

**public** **static** **void** main(String[] args) **throws** Exception {

**int** res = ToolRunner.*run*(**new** WordCount(), args);

System.*exit*(res);

}

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

Job job = Job.*getInstance*(getConf(), "wordcount");

job.setJarByClass(**this**.getClass());

// Use TextInputFormat, the default unless job.setInputFormatClass is used

FileInputFormat.*addInputPath*(job, **new** Path("input.txt"));

FileOutputFormat.*setOutputPath*(job, **new** Path("output"));

job.setMapperClass(Map.**class**);

job.setReducerClass(Reduce.**class**);

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

job.setOutputValueClass(IntWritable.**class**);

**return** job.waitForCompletion(**true**) ? 0 : 1;

}

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, IntWritable> {

**private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);

**private** Text word = **new** Text();

**private** **long** numRecords = 0;

**private** **static** **final** Pattern ***WORD\_BOUNDARY*** = Pattern.*compile*("\\s\*\\b\\s\*");

**public** **void** map(LongWritable offset, Text lineText, Context context)

**throws** IOException, InterruptedException {

String line = lineText.toString();

Text currentWord = **new** Text();

**for** (String word : ***WORD\_BOUNDARY***.split(line)) {

**if** (word.isEmpty()) {

**continue**;

}

currentWord = **new** Text(word);

context.write(currentWord,***one***);

}

}

}

**public** **static** **class** Reduce **extends** Reducer<Text, IntWritable, Text, IntWritable> {

@Override

**public** **void** reduce(Text word, Iterable<IntWritable> counts, Context context)

**throws** IOException, InterruptedException {

**int** sum = 0;

**for** (IntWritable count : counts) {

sum += count.get();

}

context.write(word, **new** IntWritable(sum));

}

}

}