Source Code – WordCount.java

```java
package org.myorg;

import java.io.*;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

```

```java
public class WordCount extends Configured implements Tool {

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

  static enum Counters { INPUT_WORDS }

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  private boolean caseSensitive = true;
  private Set<String> patternsToSkip = new HashSet<String>();

  private long numRecords = 0;
  private String inputFile;

  public void configure(JobConf job) {
   caseSensitive =
job.getBoolean("wordcount.case.sensitive", true);
   inputFile = job.get("map.input.file");

   if (job.getBoolean("wordcount.skip.patterns", false)) {
    Path[] patternsFiles = new Path[0];
    try {
     patternsFiles =
DistributedCache.getLocalCacheFiles(job);
    } catch (IOException ioe) {
      System.err.println("Caught exception while getting cached files: " + StringUtils.stringifyException(ioe));
    }
    for (Path patternsFile : patternsFiles) {
      parseSkipFile(patternsFile);
    }
   }
  }

  private void parseSkipFile(Path patternsFile) {
   try {
    BufferedReader fis = new BufferedReader(new FileReader(patternsFile.toString()));
    String pattern = null;
    while ((pattern = fis.readLine()) != null) {
      patternsToSkip.add(pattern);
    }
   } catch (IOException ioe) {
    System.err.println("Caught exception while parsing the cached file '" + patternsFile + "' : " + StringUtils.stringifyException(ioe));
   }
  }

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
   String line = (caseSensitive) ? value.toString() : value.toString().toLowerCase();

```

```java
    for (String pattern : patternsToSkip) {
     line = line.replaceAll(pattern, "");
    }

    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
     word.set(tokenizer.nextToken());
     output.collect(word, one);
     reporter.incrCounter(Counters.INPUT_WORDS, 1);
    }

    if ((++numRecords % 100) == 0) {
     reporter.setStatus("Finished processing " + numRecords
+ " records " + "from the input file: " + inputFile);
```

```
    }
   }
  }

 public static class Reduce extends MapReduceBase
implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable>
values, OutputCollector<Text, IntWritable> output,
Reporter reporter)
 throws IOException {
   int sum = 0;
   while (values.hasNext()) {
    sum += values.next().get();
   }
   output.collect(key, new IntWritable(sum));
  }
 }

 public int run(String[] args) throws Exception {
  JobConf conf = new JobConf(getConf(), WordCount.class);
  conf.setJobName("wordcount");

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

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

```

```java
  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  List<String> other_args = new ArrayList<String>();
  for (int i=0; i < args.length; ++i) {
   if ("-skip".equals(args[i])) {
     DistributedCache.addCacheFile(new
Path(args[++i]).toUri(), conf);
     conf.setBoolean("wordcount.skip.patterns", true);
   } else {
     other_args.add(args[i]);
   }
  }

  FileInputFormat.setInputPaths(conf, new Path
(other_args.get(0)));
  FileOutputFormat.setOutputPath(conf, new Path
(other_args.get(1)));

  JobClient.runJob(conf);
  return 0;
 }

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

Sample Runs

Sample text-files as input:

```
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World, Bye World!

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop, Goodbye to hadoop.
```

Run the application:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
/usr/joe/wordcount/input /usr/joe/wordcount/output
```

Output:

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop, 1
Hello 2
World! 1
World, 1
hadoop. 1
to 1
```

Notice that the inputs differ from the first version we looked at, and how they affect the outputs.

Now, lets plug-in a pattern-file which lists the word-patterns to be ignored, via the `DistributedCache`.

```
$ hadoop dfs -cat /user/joe/wordcount/patterns.txt
\.
\,
\!
to
```

Run it again, this time with more options:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
-Dwordcount.case.sensitive=true /usr/joe/wordcount/input
/usr/joe/wordcount/output -skip
/user/joe/wordcount/patterns.txt
```

As expected, the output:

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop 1
Hello 2
World 2
hadoop 1
```

Run it once more, this time switch-off case-sensitivity:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
-Dwordcount.case.sensitive=false /usr/joe/wordcount/input
/usr/joe/wordcount/output -skip
/user/joe/wordcount/patterns.txt
```

Sure enough, the output:

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
bye 1
goodbye 1
hadoop 2
hello 2
world 2
```

Highlights

The second version of `WordCount` improves upon the previous one by using some features offered by the MapReduce framework:

- Demonstrates how applications can access configuration parameters in the `configure` method of the `Mapper` (and `Reducer`) implementations (lines 28-43).
- Demonstrates how the `DistributedCache` can be used to distribute read-only data needed by the jobs. Here it allows the user to specify word-patterns to skip while counting (line 104).
- Demonstrates the utility of the `Tool` interface and the `GenericOptionsParser` to handle generic Hadoop command-line options (lines 87-116, 119).
- Demonstrates how applications can use `Counters` (line 68) and how they can set application-specific status information via the `Reporter` instance passed to the`map` (and `reduce`) method (line 72).