East West University
Department of Computer Science and Engineering

CSE 488: Big Data Analytics
LAB 01 (Handout)
Course Instructor: Dr. Mohammad Rezwanul Huq

Introduction to Hadoop and MapReduce Programming

Lab Objective

Familiarize students with the Hadoop Big Data Processing Platform and MaoReduce Programming Model.

Lab Outcome

After completing this lab successfully, students will be able to:
1. **Install** and **use** a virtual machine executing Hadoop Framework.
2. **Write** basic MapReduce programs and **execute** in Hadoop Framework.

Psychomotor Learning Levels

This lab involves activities that encompass the following learning levels in psychomotor domain.

Level	**Category**	**Meaning**	**Keywords**
P1	Imitation	Copy action of another; observe and replicate.	Relate, Repeat, Choose, Copy, Follow, Show, Identify, Isolate.
P2	Manipulation	Reproduce activity from instruction or memory	Copy, response, trace, Show, Start, Perform, Execute, Recreate.

Required Applications/Tools

- Cloudera Distribution of Hadoop
  - Cloudera is a big data platform for both IT and the business.
  - **Cloudera Distribution Hadoop (CDH)** has the ability to add new services to a running Hadoop cluster as well as it supports multi cluster management.
  - https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.5.0-0-virtualbox.zip

Lab Activities

1. **Setting up a Virtual Machine**
   - Download and install VirtualBox on your machine
   - Download the Cloudera Quickstart VM
   - Uncompress the VM archive.
   - Start VirtualBox and click Import Appliance in the File dropdown menu. Click the folder icon beside the location field. Browse to the uncompressed archive folder, select the .ovf file, and click the Open button. Click the Continue button. Click the Import button.
   - Your virtual machine should now appear in the left column. Select it and click on Start to launch it.

- To verify that the VM is running and you can access it, open a browser to the URL: http://localhost:8088. You should see the resource manager UI. The VM uses port forwarding for the common Hadoop ports, so when the VM is running, those ports on localhost will redirect to the VM.

2. Hadoop Framework

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
- **A framework for reliable, scalable, distributed computing.**

3. Running Hadoop Jobs

Generally, Hadoop can be run in three modes.

1. Standalone (or local) mode: There are no daemons used in this mode. Hadoop uses the local file system as an substitute for HDFS file system. The jobs will run as if there is 1 mapper and 1 reducer.

2. Pseudo-distributed mode: All the daemons run on a single machine and this setting mimics the behavior of a cluster. All the daemons run on your machine locally using the HDFS protocol. There can be multiple mappers and reducers.

3. Fully-distributed mode: This is how Hadoop runs on a real cluster.

   In this lab, we will show you how to run Hadoop jobs in Pseudo-distributed mode (to mimic the behavior of a cluster environment).

4. Creating a Hadoop project in Eclipse

1. Open Eclipse. If you just launched the VM, you may have to close the Firefox window to find the Eclipse icon on the desktop.
2. Right-click on the training node in the Package Explorer and select Copy. See Fig. 1.

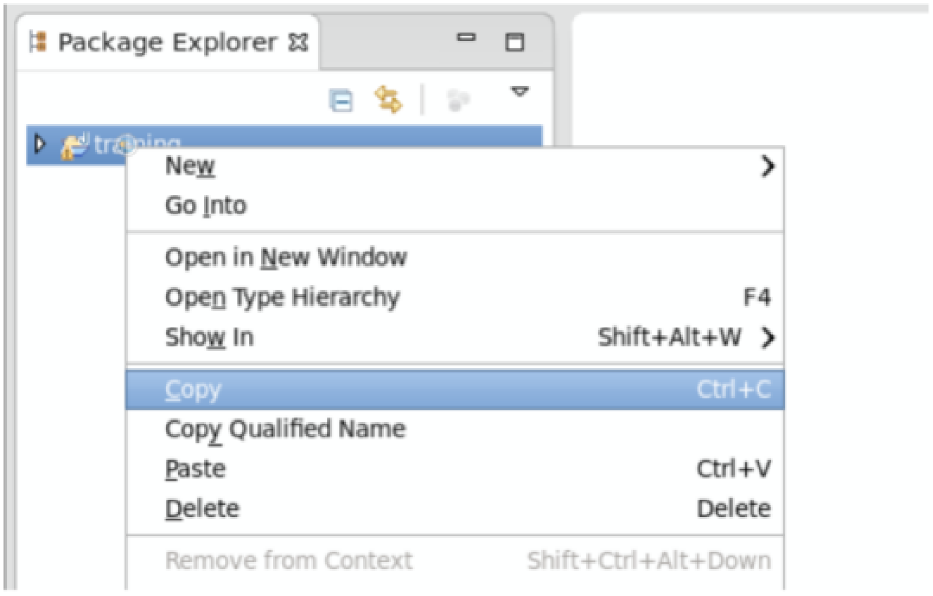
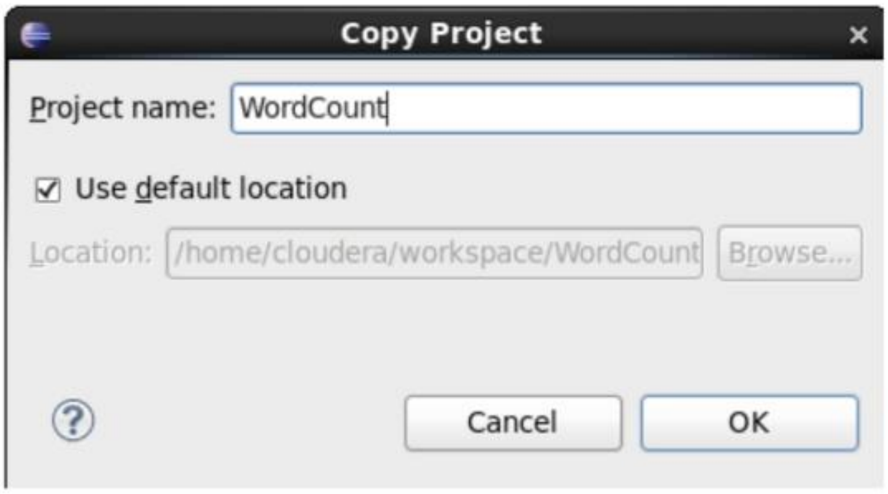


Fig. 1: Create a Hadoop Project

3. Now, hit Paste and enter the new project name in the Project Name field and click OK. See Fig. 2.



Project name: WordCount

☑ Use default location

Location: /home/cloudera/workspace/WordCount

Fig. 2: Create a Hadoop Project

4. Modify or replace the stub classes found in the src directory as needed.

5. MapReduce Programming Model

MapReduce is a style of programming designed for:

- Easy parallel programming
- Invisible management of hardware and software failures
- Easy management of very-large-scale data

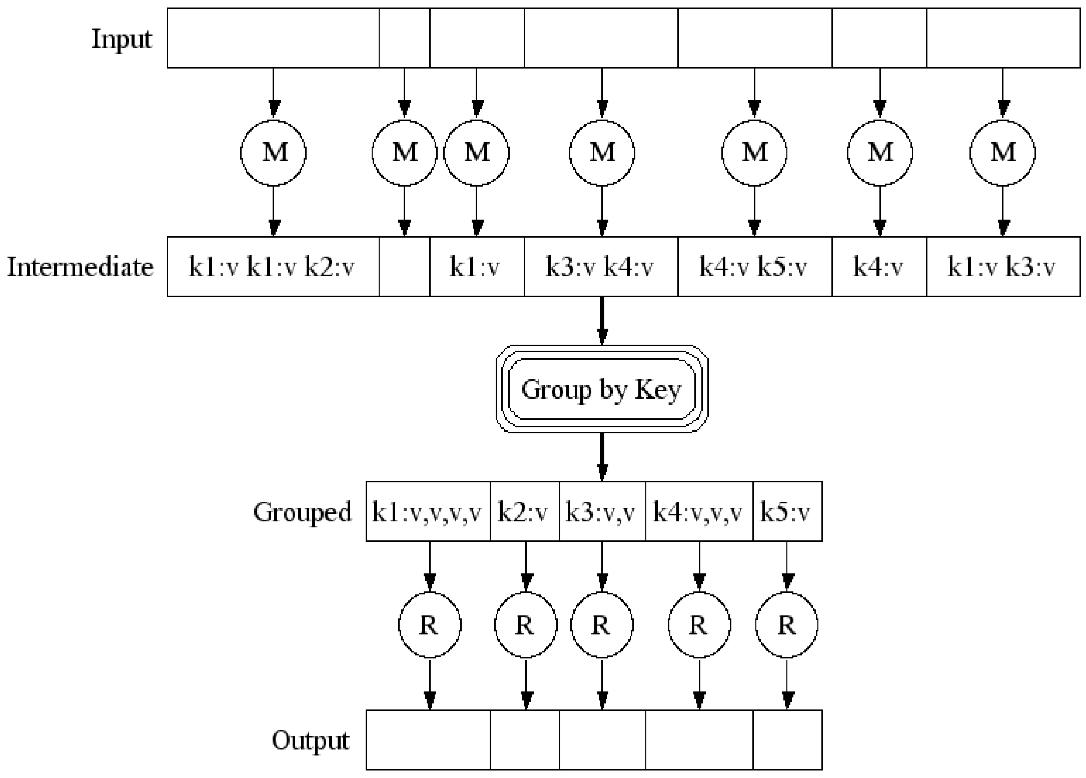
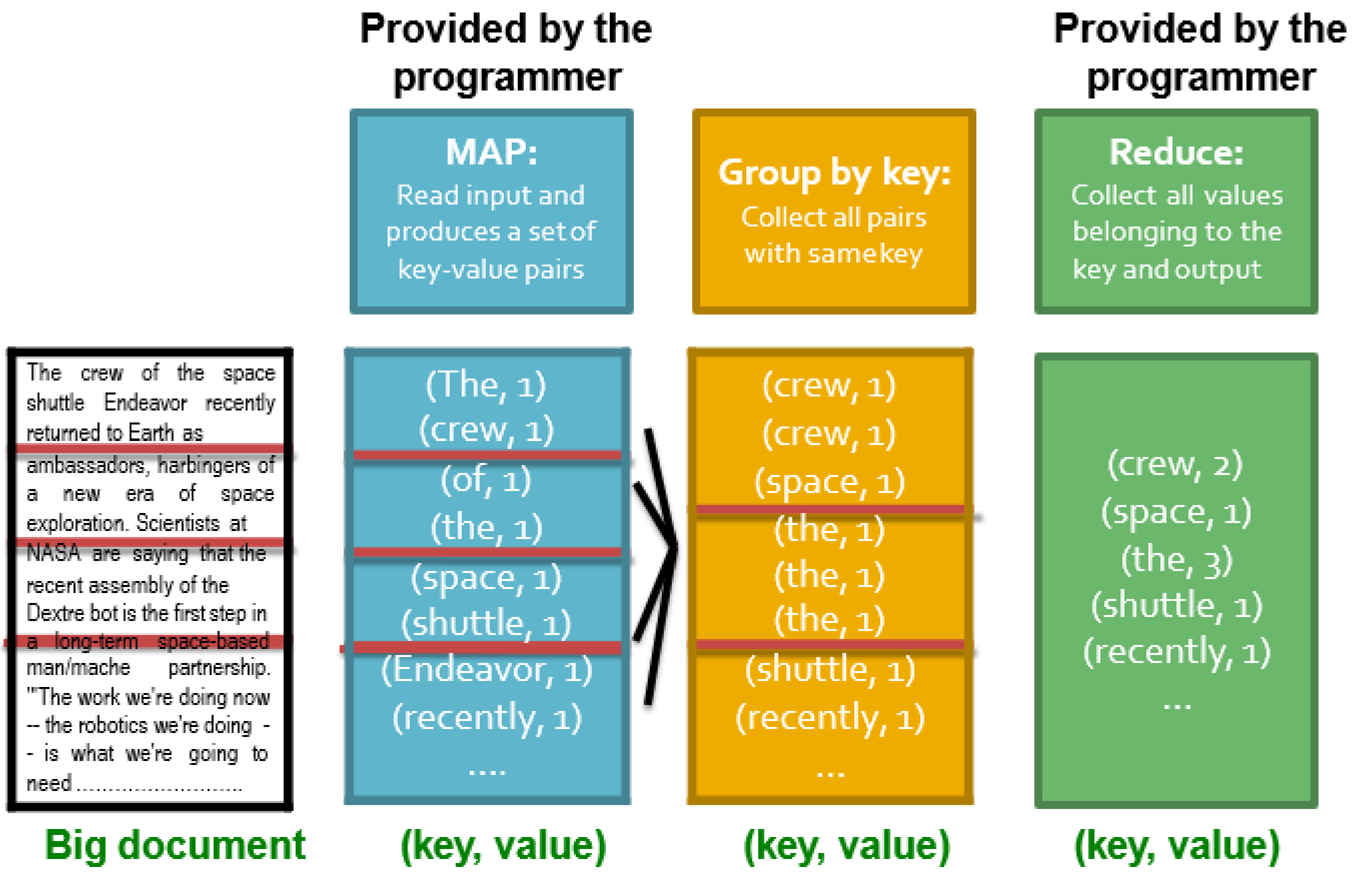


Fig. 3: MapReduce in Action



Provided by the programmer
Provided by the programmer
MAP: Read input and produces a set of key-value pairs
Group by key: Collect all pairs with same key
Reduce: Collect all values belonging to the key and output
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing - - is what we're going to need ....
(The, 1) (crew, 1) (of, 1) (the, 1) (space, 1) (shuttle, 1) (Endeavor, 1) (recently, 1) ....
(crew, 1) (crew, 1) (space, 1) (the, 1) (the, 1) (the, 1) (shuttle, 1) (recently, 1) …
(crew, 2) (space, 1) (the, 3) (shuttle, 1) (recently, 1) …
Big document
(key, value)
(key, value)
(key, value)

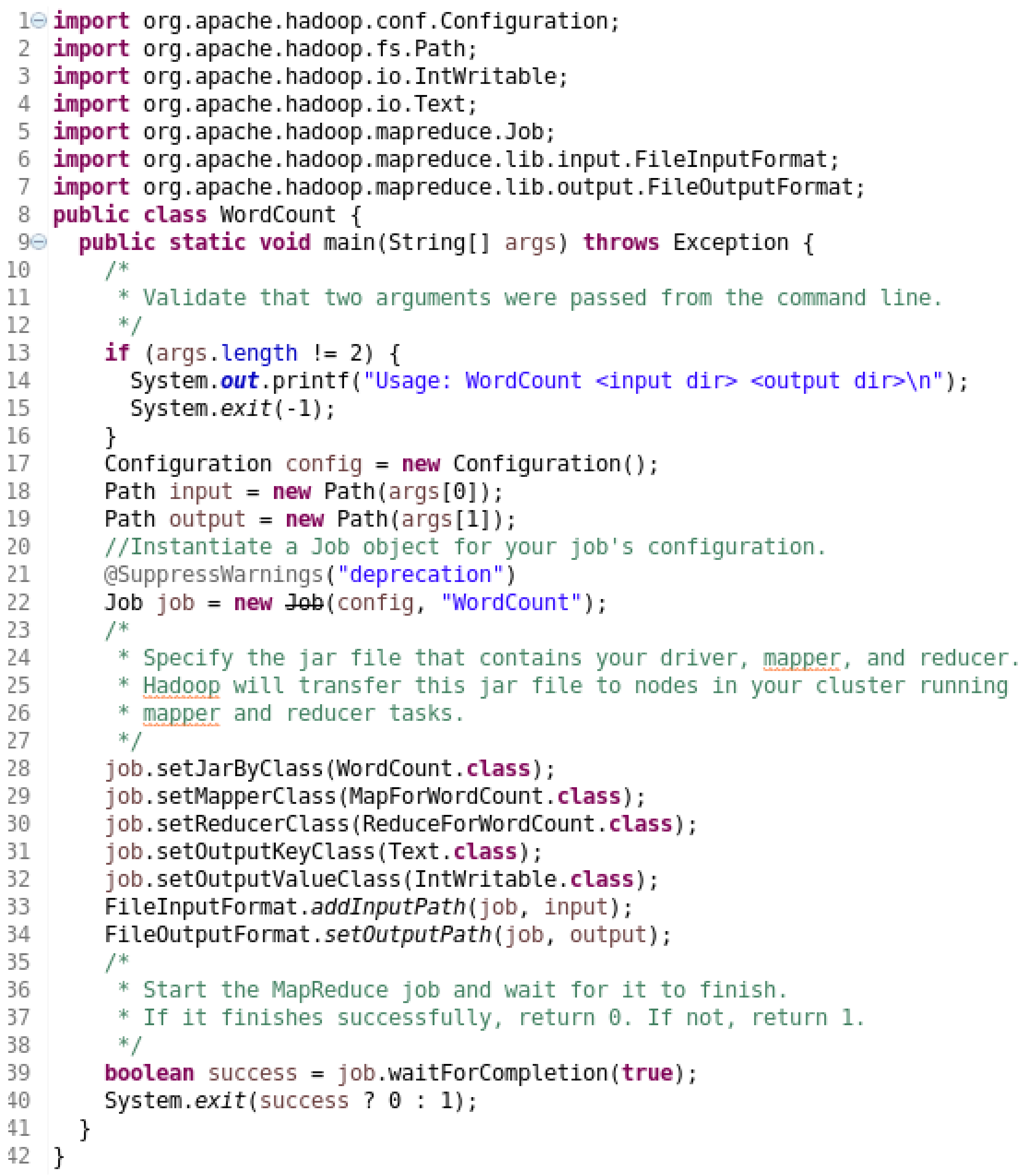
Fig. 4: MapReduce in Action with an example

6. Psuedocode of WordCount using MapReduce

```
map(key, value):
# key: document id; value: text of the document
     for each word w in value:
          emit(w, 1)

reduce(key, values):
# key: a word; value: an iterator over counts
     result = 0
     for each count v in values:
          result += v
          emit(key, result)
```

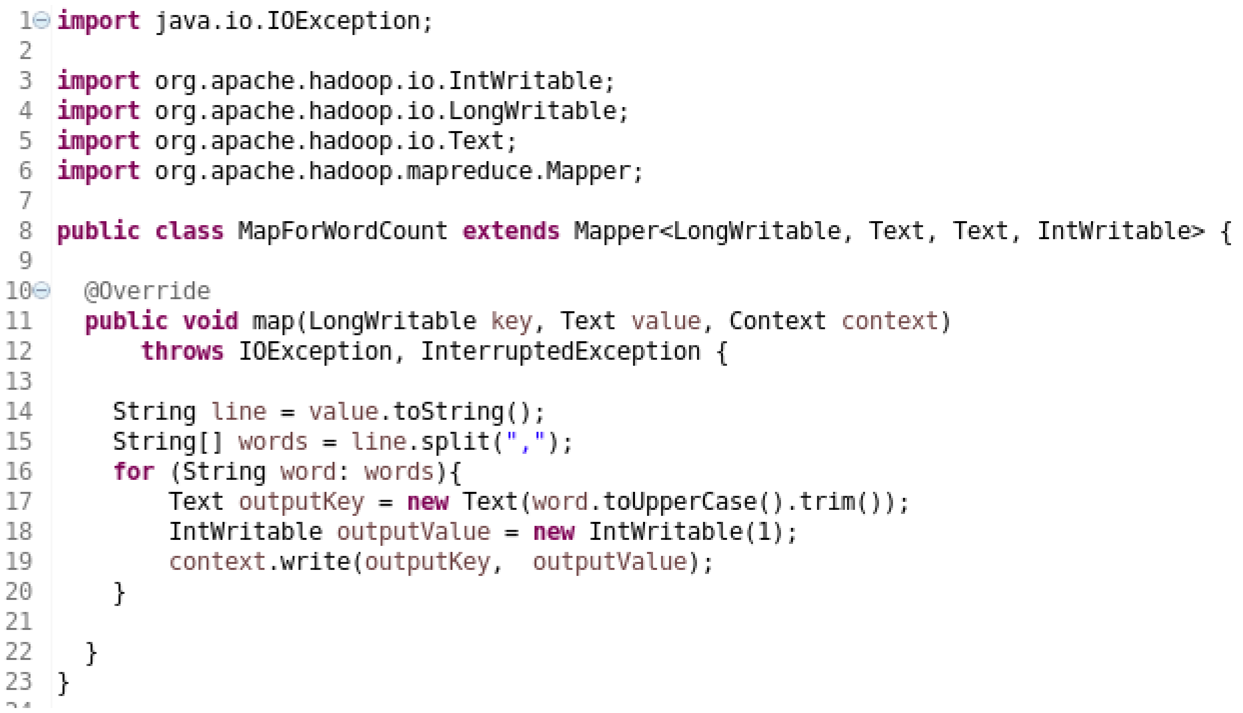
7. Codes need to be written

WordCount.java



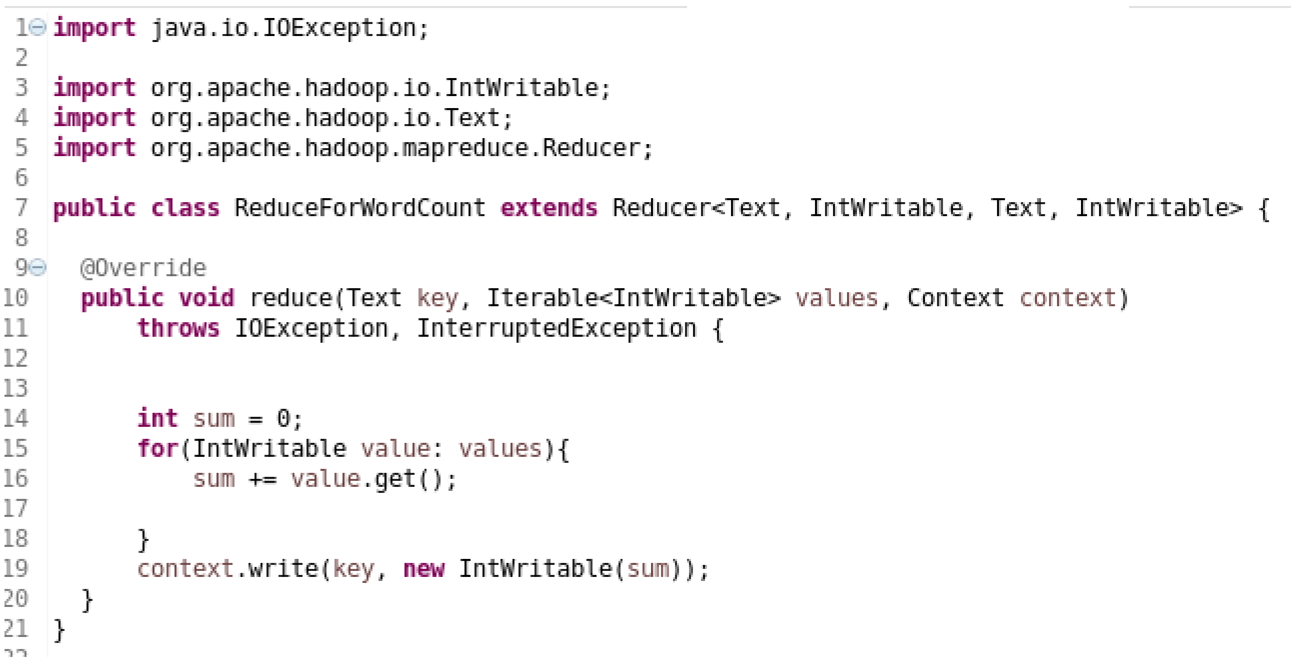
```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
  public static void main(String[] args) throws Exception {
    /*
     * Validate that two arguments were passed from the command line.
     */
    if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }
    Configuration config = new Configuration();
    Path input = new Path(args[0]);
    Path output = new Path(args[1]);
    //Instantiate a Job object for your job's configuration.
    @SuppressWarnings("deprecation")
    Job job = new Job(config, "WordCount");
    /*
     * Specify the jar file that contains your driver, mapper, and reducer.
     * Hadoop will transfer this jar file to nodes in your cluster running
     * mapper and reducer tasks.
     */
    job.setJarByClass(WordCount.class);
    job.setMapperClass(MapForWordCount.class);
    job.setReducerClass(ReduceForWordCount.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, input);
    FileOutputFormat.setOutputPath(job, output);
    /*
     * Start the MapReduce job and wait for it to finish.
     * If it finishes successfully, return 0. If not, return 1.
     */
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

MapForWordCount.java



```java
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.mapreduce.Mapper;

public class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String[] words = line.split(",");
    for (String word: words){
        Text outputKey = new Text(word.toUpperCase().trim());
        IntWritable outputValue = new IntWritable(1);
        context.write(outputKey,  outputValue);
    }

  }
}
```

ReduceForWordCount.java



```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {


      int sum = 0;
      for(IntWritable value: values){
          sum += value.get();

      }
      context.write(key, new IntWritable(sum));
  }
}
```