import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

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

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

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {

//Mapper

/\*\*

\*MaxTemperatureMapper class is static and extends Mapper abstract class

having four hadoop generics type LongWritable, Text, Text, Text.

\*/

public static class MaxTemperatureMapper extends

Mapper<LongWritable, Text, Text, Text> {

/\*\*

\* @method map

\* This method takes the input as text data type.

\* Now leaving the first five tokens,it takes 6th token is taken as temp\_max and

\* 7th token is taken as temp\_min. Now temp\_max > 35 and temp\_min < 10 are passed to the reducer.

\*/

@Override

public void map(LongWritable arg0, Text Value, Context context)

throws IOException, InterruptedException {

//Converting the record (single line) to String and storing it in a String variable line

String line = Value.toString();

//Checking if the line is not empty

if (!(line.length() == 0)) {

//date

String date = line.substring(6, 14);

//maximum temperature

//minimum temperature

float temp\_Min = Float

.parseFloat(line.substring(47, 53).trim());

//if maximum temperature is greater than 35 , its a hot day

//if minimum temperature is less than 10 , its a cold day

if (temp\_Min < 10) {

// Cold day

context.write(new Text("Cold Day " + date),

new Text(String.valueOf(temp\_Min)));

}

}

}

}

//Reducer

/\*\*

\*MaxTemperatureReducer class is static and extends Reducer abstract class

having four hadoop generics type Text, Text, Text, Text.

\*/

public static class MaxTemperatureReducer extends

Reducer<Text, Text, Text, Text> {

/\*\*

\* @method reduce

\* This method takes the input as key and list of values pair from mapper, it does aggregation

\* based on keys and produces the final context.

\*/

public void reduce(Text Key, Iterator<Text> Values, Context context)

throws IOException, InterruptedException {

//putting all the values in temperature variable of type String

String temperature = Values.next().toString();

context.write(Key, new Text(temperature));

}

}

/\*\*

\* @method main

\* This method is used for setting all the configuration properties.

\* It acts as a driver for map reduce code.

\*/

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

//reads the default configuration of cluster from the configuration xml files

Configuration conf = new Configuration();

//Initializing the job with the default configuration of the cluster

Job job = new Job(conf, "weather example");

//Assigning the driver class name

job.setJarByClass(MyMaxMin.class);

//Key type coming out of mapper

job.setMapOutputKeyClass(Text.class);

//value type coming out of mapper

job.setMapOutputValueClass(Text.class);

//Defining the mapper class name

job.setMapperClass(MaxTemperatureMapper.class);

//Defining the reducer class name

job.setReducerClass(MaxTemperatureReducer.class);

//Defining input Format class which is responsible to parse the dataset into a key value pair

job.setInputFormatClass(TextInputFormat.class);

//Defining output Format class which is responsible to parse the dataset into a key value pair

job.setOutputFormatClass(TextOutputFormat.class);

//setting the second argument as a path in a path variable

Path OutputPath = new Path(args[1]);

//Configuring the input path from the filesystem into the job

FileInputFormat.addInputPath(job, new Path(args[0]));

//Configuring the output path from the filesystem into the job

FileOutputFormat.setOutputPath(job, new Path(args[1]));

//deleting the context path automatically from hdfs so that we don't have delete it explicitly

OutputPath.getFileSystem(conf).delete(OutputPath);

//exiting the job only if the flag value becomes false

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}