MyMaxMin java class :

```
// importing Libraries
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 a 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 > 30 and temp_min < 15 are
* passed to the reducer.
*/

// the data in our data set with
// this value is inconsistent data
public static final int MISSING = 9999;

@Override
public void map(LongWritable arg0, Text Value, Context context)
throws IOException, InterruptedException {

// Convert the single row(Record) to
// String and store it in String
// variable name line

String line = Value.toString();

// Check for the empty line
if (!(line.length() == 0)) {

// from character 6 to 14 we have
// the date in our dataset
String date = line.substring(6, 14);

// similarly we have taken the maximum
// temperature from 39 to 45 characters
float temp_Max = Float.parseFloat(line.substring(39, 45).trim());
```

```
// similarly we have taken the minimum
// temperature from 47 to 53 characters

float temp_Min = Float.parseFloat(line.substring(47, 53).trim());

// if maximum temperature is
// greater than 30, it is a hot day
if (temp_Max > 30.0) {

// Hot day
context.write(new Text("The Day is Hot Day :" + date),
new Text(String.valueOf(temp_Max)));
}

// if the minimum temperature is
// less than 15, it is a cold day
if (temp_Min < 15) {

// Cold day
context.write(new Text("The Day is 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 the 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 the
// 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
// to 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);


}

}
```

Output:

```
The Day is Cold Day :20200101   -21.8
The Day is Cold Day :20200102   -23.4
The Day is Cold Day :20200103   -25.4
The Day is Cold Day :20200104   -26.8
The Day is Cold Day :20200105   -28.8
The Day is Cold Day :20200106   -30.0
The Day is Cold Day :20200107   -31.4
The Day is Cold Day :20200108   -33.6
The Day is Cold Day :20200109   -26.6
The Day is Cold Day :20200110   -24.3
```

In the above image, you can see the top 10 results showing the cold days. The second column is a day in yyyy/mm/dd format. For Example, 20200101 means

year = 2020
month = 01
Date = 01