Weather Program

```
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
    public static class MaxTemperatureMapper extends Mapper<LongWritable,
Text, Text, Text> {

        @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
                float temp_Max = Float.parseFloat(line.substring(39, 45).trim());
                // Minimum temperature
                float temp_Min = Float.parseFloat(line.substring(47, 53).trim());

                // If maximum temperature is greater than 35, it's a hot day
                if (temp_Max > 35.0) {
                    // Hot day
                    context.write(new Text("Hot Day " + date), new
Text(String.valueOf(temp_Max)));
                }

                // If minimum temperature is less than 10, it's a cold day
                if (temp_Min < 10) {
                    // Cold day
                    context.write(new Text("Cold Day " + date), new
Text(String.valueOf(temp_Min)));
                }
            }
        }
    }

    // Reducer
    public static class MaxTemperatureReducer extends Reducer<Text, Text,
Text, Text> {

        public void reduce(Text Key, Iterator<Text> Values, Context context)
throws IOException, InterruptedException {
            // Putting all the values in the temperature variable of type String
            String temperature = Values.next().toString();
            context.write(Key, new Text(temperature));
        }
    }

    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 output 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);
    }
}
```