<u>Map Reduce</u>

Structure of Map Reduce Class

```
// import packages
public class MainClass
{
public static MyMapperClass extends Mapper<input key , input value, output key, output value>
          {
            public void map(Input key, input value, context ) throws IOException, InterruptedExceptions
                    {
                              Functionality of map
                    }
public static MyReducerClass extends Reducer<input key , input value , ouput key, output value>
          {
            public void reduce(Input key, input value, context) throws IOException, InterruptedExceptions
                    {
                              Functionality of Reducer
                    }
          Public static void main(String args[])
                    {         Job Description, Configuration
                    }
```

<K,V> = K→Line offset , V→ entire line value

map() by taking that line separated required key and value <K,V>

<K,V> write(separated key, separated value) →map() function will automatically shuffles key, value pair

This shuffles data is sent to reduce functions.

We perform required Aggregation/Operations on value list.

Data Types in Map/Reduce:

Text→equivalent to string in java
Intwritable→Int in java
LongWritable→Long, float, double in Java

Import org.apache.hadoop.io.Text;
Import org.apache.hadoop.io.IntWritable;
Import org.apache.hadoop.io.LongWritable;

<u>Configuration:</u> to define/change configuration parameter run time

Import org.apache.hadoop.conf.configuration;
    Configuration c = new Configuration();

<u>Job:</u> to define a map reduce job

Import org.apache.hadoop.mapreduce.Job;

<u>Path:</u> to convert input path or output path into HDFS comfortable path.

Import org.apache.hadoop.fs.path;

<u>Generic Options Parser:</u> to parse arguments of CLI

String[] passed Arrays = new GenericOptionparser(J1, args);
    getRenamingParser();

import org.apache.hadoop.util.GenericOptionparser.

FileInputFormat→to specify input file path;
FileOutputFormat→to specify output file location.

Import org.apache.hadoop.mapreduce.lib.input.FileInputFormt.

Import org.apache.hadoop.mapreduce.lib.input.FileOutputFormt.
Example:
 FileInputFormat.addInputPath(J1, new path(parse[args[0]]);
FileOutputFormat.setOutPath(J1, new path(parse[args[1]]);

Mapper:

Import org.apache.hadoop.mapreduce.Mapper;
Import org.apache.hadoop.mapreduce.Reducer.

Temperature.txt

```
Xxxxx1950xxxxx21xxxx1xxxx
Xxxxx1950xxxxx27xxxx2xxxx
Xxxxx1950xxxxx39xxxx5xxxx
Xxxxx1950xxxxx19xxxx1xxxx
Xxxxx1951xxxxx27xxxx7xxxx
Xxxxx1951xxxxx22xxxx2xxxx
Xxxxx1951xxxxx 8xxxx2xxxx
```

Data collected from temperature recording machines (10,000) each machine reads temperature per each minute.
Data collected from 1950 to till date. Each year data is one separate file. All files are in HDFS directory /user/training/machines
Data Clues:
 year →6th column position→length -4
Temperature→15th column position →length -2 (some temperatures are in negative values)
If temperature is negative it's value is in 3 digits including sign
Quality→it indicated in 1,2,3→temperature collected from good conditioned machine.
Quality indicators→22nd column position→length: 1
Target Analytics:
Find the maximum temperature from each year
1950-27
1951-19 (don't consider bad records in terms of quality)
Package world.air.temperature

```
Do all imports
public class MaxTemperature
{
public static class MapforMaxTemperature extends Mapper <Longwritable, Text, Text, Intwritable>
            {
             Text mapkey = new Text();
             Intwritable mapval = new Intwritable();
public void map(Longwritabe key, Text val, ConteXt con) throws IO Exception, InterruptedExceptions
            {
            String line = val.toString();
             Int quality = Integet.parseInt(line.charAt(23));
             If(quality<4)
            Mapkey.set(line.subString(5,9));
            if(line.charAt(14)!="-")
             Mapval.set(Integer.parseInt(line.subString(14,16);
             else
             Mapval.set(Integer.parseInt(line.subString(14,17);
             con.write(mapkey, mapval);
             }
            }
public static class ReduceForMaxTemperature extends Reducer<Text, Intwritable, Text, Intwritable>
{
            Intwritable res= new Intwritable();
Public  void reduce(Text y, Interable<Intwritable> values, Context con) throws IOException,
                                                                                                    InterruptedException
            {           int m=0, m=values[0].get();//for min
                        for(Intwritable v:values)
                        {
                          m = math.max(m.v.get());
                        }
                        res.get(m);
                        con.write(y, res);
            }
}
Public static void main(String args[]) throws Exception
            {
            Configuration c = new Configuration();
            String[] parsed = new genericOptionparser(c, args).getRemaining args();
            Path p1 = new Path(parsed[0]);
            Path p2 = new Path(parsed[1]);
            Job j = new Job(c, "Temperature");
            j.setJarByClass(MaxTemperature.class);
            j.setMapperClass(MapForMaxTemperature.class);
            j.setCombinerClass(ReducerForMaxTemperature.class);
            j.setReducerClass(ReducerForMaxTemperature.class);
            j.setOutputKeyClass(Text.class);
            j.setOutPutValueClass(Intwritable.class);
            FileInputFormat.addInptPath(j, p1);
            FileOutputFormat .setOutputpath(j, p2);
            System.exit(j.waitForCompletion(true)?0:1);
            }
}
```

Commands to Execute:

```
$ hadoop jar /home/training/desktop/temp.jar  world.air.Temperature  MaxTemperature Temperature
Res
```

For quality of records change in the mapper class

```
Public void map(Longwritable, ------ )
{  int q = Integer.parseInt(line.charAt(21));
            If(q<4)
            status="good";
            else
            status = "bad";
            mapkey.set(status);
            con.write(mapkey, mapval);
```

Reducer:

```
{
            int sum= 0;
            for(Intwritable v:vals)
            {
                        sum+=v.get
            }
res.get(sum);
con.write(status, res);
}

Do all imports
Public class WordCount
{
Public class MaxTemperature
{
Public static class WordCountMapper extends Mapper <Longwritable, Text, Text, Intwritable>
            {
             Text mapkey = new Text();
              Intwritable mapval = new Intwritable();
              int cnt=1;
Public void map(Longwritabe key, Text val, ConteXt con) throws IO Exception, InterruptedExceptions
            {
            String line = val.toString();
             StringTokenizer  t = new StringTokenizer(line);
            while(t.hasmoretokens())
            {
             String word = t.nextToken();
             mapkey.set(word);
            mapval.set(cnt)
             con.write(word,cnt);
             }
            }
}
Public static class WordCountReducer extends Reducer<Text, Intwritable, Text, Intwritable>
{
            Intwritable res= new Intwritable();
Public  void reduce(Textokey, Interable<Intwritable> values, Context con) throws IOException,
                                                                                                                        InterruptedException
            {           int sum=0;
                        for(Intwritable v:values)
                        {
                                    sum +=v.get();
                        }
                        res.set(sum);
                        con.write(okey, res);
            }
}
Public static void main(String args[]) throws Exception
{
            Configuration c = new Configuration();
            String[] parsed = new GenericOptionparser(c, args).getRemaining args();
            Path p1 = new Path(parsed[0]);
            Path p2 = new Path(parsed[1]);
            Job j = new Job(c, “Temperature”);
            j.setJarByClass(MaxTemperature.class);
            j.setMapperClass(WordCountMapper.class);
            j.setCombinerClass(WordCountReducer.class);
            j.setReducerClass(WordCountReducer.class);
            j.setOutputKeyClass(Text.class);
            j.setOutPutValueClass(Intwritable.class);
            FileInputFormat.addInptPath(j, p1);
            FileOutputFormat .setOutputpath(j, p2);
            System.exit(j.waitForCompletion(true)?0:1);
            }
}
```

The classes Text, Intwritable, Longwritable, Path, Configuration, Job, FileInputFormat, FileOutputFormat , Mapper, Reducer are available in hadoop-core.jar (/usr/lib/hadoop-0.20/hadoop-core.jar)
And GenericOptionParser is available with commons – (usr/lib/hadoop-0.20/lib/commons-cli-1.2.jar)

Working with Delimited files:

```
101, Amar, 20000. 11, m
102, Amaresh, 30000, 11, m
104, Geeta, 40000, 12, f
104, pavani, 43000, 13, f
105, vikram, 54000, 14, m
```

```
Package myorg.hr.core;
Inport java.io.Exception;
Import java.util.StringTokenizer;
Import  map/reduce packages;
public class DeptAggregation
{
    Public static class MapForAggregation extends Mapper<LongWritable, Text, Text, IntWritable>
{
    String dname;
    int sal;
    Text mapkey = new Text();
     Intwritable mapval = new Intwritable();
     Public void map(Longwritable Key, Text val, Context con) throws IOException, InterruptedException
     {
     String line = val.toString();
     StringTokenizer t = new StringTokenizer(line);
     Int i=1;
            while(t.hasmoretokens())
            {
             String str = t.nextToken();
            If(i==3)
            sal = Integer.parseInt(str);
            if(i==4)
            i++;
            dname = str3;
            }
            mapkey.set(dname);
            mapval.set(sal);
            con.write(mapkey, mapval);
            }
}

public class ReduceForDeptAggregation extends Reduce<Text, IntWritable, Text, Intwritable>
{
String dept;
int totalsal;
 public void reduce(Text dno, Iterable<IntWritable> vals, context con) throws IOexception, Interrupted
Exception
    {
     totalsal = 0;
for(Intwritable v: vals)
{
totalsal+=v.get();
}
dept=dno.toString();
if(dept.matches("11"))
   dept="marketing";
else if(dept.matches("12"))
    dept ="hr";
else if(dept.matches("13"))
  dept="finance";
else
  dept="other";
con.write(new Text(dept), new Intwritable(totalsal));
}
}
public static void main(String args[]) throws Exception
{
{
            Configuration c = new Configuration();
            String[] files = new GenericOptionparser(c, args).getRemaining args();
            Path p1 = new Path(files[0]);
            Path p2 = new Path(files[1]);
            Job j = new Job(c, "AggrJob");
            j.setJarByClass(DeptAggragation.class);
            j.setMapperClass(MapforDeptAggregation.class);
            j.setCombinerClass(ReduceForAggregation.class);
            j.setReducerClass(ReduceForAggragation.class);
            j.setOutputKeyClass(Text.class);
            j.setOutPutValueClass(Intwritable.class);
            FileInputFormat.addInptPath(j, p1);
            FileOutputFormat .setOutputpath(j, p2);
            System.exit(j.waitForCompletion(true)?0:1);
            }
}
```