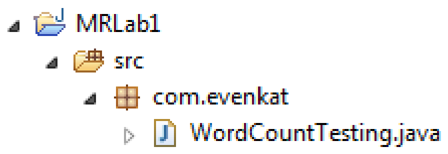
I. Map Reduce – Word Count

A. Switch the workspace to a known folder.

B. Open eclipse and create a new Java Project called **MRLab**

C. Hint : ***File-> New-> Others-> Java Project***

D. Create a package **com.evenkat** under *src* folder under project *MRLab*



E. Add the Hadoop jar files to the project

Hint : Right click on **MRLab->Properties->Java Build Path-> Libraries - Add External Jars**

Jar files from hadoop 2.7.2 directory.

3 jar files from → share\hadoop\common

9 jar files from → share\hadoop\mapreduce

All jars from → share\hadoop\mapreduce\lib

F. Create a class called WordCountTesting

G. The packages to be imported are

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.StringTokenizer;
```

H. **Map Reduce Program code**

Note: This would be inside the class WordCountTesting

```java
public static class MyMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {

        private Text word = new Text();

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

          String line = value.toString();

          StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write( word, new IntWritable( 1)  );
          }

        }
      } //end of MyMapper class

      public static class MyReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context
context ) throws IOException, InterruptedException {

          int sum = 0;

          for (IntWritable val : values) {

              sum += val.get();

          }

          context.write( key, new IntWritable(sum) );
           }
        } //end of MyReducer class
```

I. Driver Code [Note that the driver

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

                Configuration conf = new Configuration();

              Job job = new Job(conf, "Word Counter");
          //Ignore the warning as new techniques for getting handle to Job is
available now.

              job.setJarByClass( WordCountTesting.class );
              job.setMapperClass( MyMapper.class );
              job.setReducerClass( MyReducer.class );

              job.setMapOutputKeyClass( Text.class );
              job.setMapOutputValueClass( IntWritable.class );

              job.setOutputKeyClass( Text.class );
              job.setOutputValueClass( IntWritable.class );

              FileInputFormat.addInputPath( job, new Path( args[0] ) );
              FileOutputFormat.setOutputPath( job, new Path( args[1] ) );

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

J. Create the jar file

Right Click on **MRLab-> export-> java -> jar file** (Give name WordCount.jar). Click Finish

Transfer the jar file to VM under **/ home/notroot/lab/programs**

Hint : Use WinSCP software to ftp the jar file to linux VM

K. Create a words file under/home/notroot/lab/data directory and write few line of text in the file. >

L. Then move the file to HDFS using hdfs dfs –copyFromLocal and put it inside the input directory in HDFS

M. Go to /home/notroot/lab/programs and run the job

N. hadoop jar WordCount.jar com.evenkat.WordCountTesting   /input/words /output/wcount

O. Check output

Hadoop fs   -cat /output/wcount/part-r-00000 OR you can view this also through the Browse the File System through the 50070 port no.

Techniques for location System.out.println statements

Purpose: Write a line called System.out.println(key+" "+value); in the first line of the map function and to check where it will be present in the logs ?

Technique 1 → Winscp

1) Go into your Hadoop_Installation directory, then into "logs/userlogs".
2) Open your job_id directory.
3) Check directories with _m_ if you want the mapper output or _r_ if you're looking for reducers.

Example: In Hadoop-20.2.0:

```
> ls ~/hadoop-0.20.2/logs/userlogs/attempt_201209031127_0002_m_000000_0/
log.index stderr stdout syslog
```

The above means:
Hadoop_Installation: ~/hadoop-0.20.2
job_id: job_201209031127_0002
_m_: map task , "map number": _000000_

4) open stdout if you used "system.out.println" or stderr if you used "system.err.append".
PS. other hadoop versions might have a sight different hierarchy but they're all should be under $Hadoop_Installtion/logs/userlogs.

Working with Temperature MR Example

Create a new class in the same project earlier called MaxTemp [for Driver] and the name of the Inner Mapper class would be MaxMapper and the Inner Reducer class would be MaxReducer

Map Logic

```
 String line = value.toString();

 String year = line.substring(15, 19);

 int airTemperature;

 if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs

   airTemperature = Integer.parseInt(line.substring(88, 92));

 } else {

   airTemperature = Integer.parseInt(line.substring(87, 92));    }

   context.write(new Text(year), new IntWritable(airTemperature));
```

Reduce Logic

```
 int maxValue = Integer.MIN_VALUE;

 while (values.iterator().hasNext()) {

   maxValue = Math.max(maxValue, values.iterator().next().get());

 }    context.write (key, new IntWritable(maxValue));

} // end of class
```

Working with Combiner in the MR Example

job.setCombinerClass(MaxReducer.class); → Driver class after job.setMapperClass method.

Working with Transaction Example→ SumTransaction.java

[please have @override before the map and reduce function]

Driver class name-> SumDriver

Mapper-> SumMapper

Reducer ->SumReducer

Map Logic

```
                String txnString = value.toString();

                String[] txnData = txnString.split( "," );

                double amount = Double.parseDouble( txnData[3] );

           context.write( new Text( txnData[5].trim().toUpperCase() ), new DoubleWritable(
amount ) );
```

Reduce Logic

```
               Double sum = 0.00;

               for (DoubleWritable val : values) {

      sum += val.get();

    }

               DecimalFormat formatter = new DecimalFormat( "0.##" );

    context.write( key, new Text( formatter.format( sum ) ) );
```

1. Comment the combiner line for this example
2. Change the Value of the output of the mapper signature from IntWritable to DoubleWritable.
3. change the IntWritable to DoubleWritable in the context.write in the map function
4. Import DoubleWritable class
5. Change the Value of the input value of the Reducer to DoubleWritable
6. Change the Value of the output value of the Reducer to Text
7. Change the IntWritable to Text in the context.write in the reduce function.
8. Change the class to DoubleWritable for the job.setMapOutputValueClass.
9. Change the class to Text for the job.setOutputValueClass

Working with Unit Testing on the Transaction Example

Unit Testing for the above Transaction example [Include All jars provided in the build path.

```
package com.evenkat;


import org.apache.hadoop.io.*;

import org.apache.hadoop.mrunit.mapreduce.MapDriver;

import org.apache.hadoop.mrunit.mapreduce.MapReduceDriver;

import org.apache.hadoop.mrunit.mapreduce.ReduceDriver;

import org.junit.Before;

import org.junit.Test;

import java.util.*;


public class MRUnitTestCase {


        MapReduceDriver<LongWritable, Text, Text, DoubleWritable, Text, Text>
mapReduceDriver;

        MapDriver<LongWritable, Text, Text, DoubleWritable> mapDriver;

        ReduceDriver<Text, DoubleWritable, Text, Text> reduceDriver;


        @Before

        public void setUp() {

          SumMapper mapper = new    SumMapper ();

           SumReducer reducer = new SumReducer();


           mapDriver = new MapDriver<LongWritable, Text, Text, DoubleWritable>();

           mapDriver.setMapper(mapper);
        reduceDriver = new ReduceDriver<Text, DoubleWritable, Text, Text>();

        reduceDriver.setReducer(reducer);



        mapReduceDriver = new MapReduceDriver<LongWritable, Text, Text, DoubleWritable,
Text, Text>();

        mapReduceDriver.setMapper(mapper);

        mapReduceDriver.setReducer(reducer);

      }




      @Test

      public void testMapper() {

        mapDriver.withInput( new LongWritable(0), new Text("00000000,01-03-
2011,4006236,045.28,Outdoor Play Equipment,Sandboxe,New York,New York,credit") );

        mapDriver.withOutput( new Text("SANDBOXES"), new DoubleWritable( 045.28 ) );

        mapDriver.runTest();

      }


      @Test

      public void testReducer() {

        List<DoubleWritable> values = new ArrayList<DoubleWritable>();

        values.add( new DoubleWritable( 10.00 ) );

        values.add( new DoubleWritable( 15.00 ) );

        reduceDriver.withInput( new Text( "SANDBOXES" ), values );

        reduceDriver.withOutput( new Text("SANDBOXES"), new Text( "25" ) );

        reduceDriver.runTest();

      }
```

```
	@Test
	public void testMapReduceSingleProduct() {
	  mapReduceDriver.withInput( new LongWritable(0), new Text("00000000,01-03-2011,4006236,045.00,Outdoor Play Equipment,Sandboxes,New York,New York,credit") );
	  mapReduceDriver.withInput( new LongWritable(1), new Text("00000000,01-03-2011,4006236,045.00,Outdoor Play Equipment,Sandboxes,New York,New York,credit") );
	  mapReduceDriver.addOutput( new Text("SANDBOXES"), new Text( "90" ) );
	  mapReduceDriver.runTest();
	}

	@Test
	public void testMapReduceMultipleProducts() {
	  mapReduceDriver.withInput( new LongWritable(0), new Text("00000000,01-03-2011,4006236,045.00,Outdoor Play Equipment,Sandboxes,New York,New York,credit") );
	  mapReduceDriver.withInput( new LongWritable(0), new Text("00000000,01-03-2011,4006236,045.00,Outdoor Play Equipment,Skating,New York,New York,credit") );
	  mapReduceDriver.addOutput( new Text("SANDBOXES"), new Text( "45" ) );
	  mapReduceDriver.addOutput( new Text("SKATING"), new Text( "45" ) );
	  mapReduceDriver.runTest();
	}
}
```

```
package com.evenkat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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;
import org.apache.hadoop.util.GenericOptionsParser;
```

```
/*

Input Format: name      age      gender            score

Problem Statement:

We will use custom partitioning in MapReduce program to find the maximum scorer in each gender
and three age categories: less than 20, 20 to 50, greater than 50.

output:

Partition - 0: (this partition contains the maximum

scorers for each gender whose age is less than 20)

Partition - 1: (this partition contains the maximum

scorers for each gender whose age is between 20 and 50)

Partition - 2: (this partition contains the maximum

scorers for each gender whose age is greater than 50)


 */


public class SamplePartitioner{


/* The data belonging to the same partition go to the same reducer.

 * In a particular partition, all the values with the same key are iterated

 * and the person with the maximum score is found. Therefore the output

 * of the reducer will contain the male and female maximum scorers in

 * each of the 3 age categories.

*/



        public static void main(String[] args) {

                try{
```

```
            Configuration conf = new Configuration();
              Job job = new Job(conf, "Partitioning Sample");
              job.setJarByClass( SamplePartitioner.class );
        job.setMapperClass( PartitionMapper.class );
        job.setReducerClass( PartitionReducer.class );
        job.setPartitionerClass(AgePartitioner.class);
        job.setNumReduceTasks(3);

        job.setMapOutputKeyClass( Text.class );
        job.setMapOutputValueClass( Text.class );

        job.setOutputKeyClass( Text.class );
        job.setOutputValueClass( Text.class );

        FileInputFormat.addInputPath( job, new Path( args[0] ) );
        FileOutputFormat.setOutputPath( job, new Path( args[1] ) );

        System.exit( job.waitForCompletion( true ) ? 0 : 1 );
            }//end of try
            catch(Exception e){e.printStackTrace();}
            }//end of main
      }//end of class
```

```
package com. evenkat;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;


/* PartitionMapper parses the input records and emits the key,
 * value pairs suitable for the partitioner and the reducer.
 * Mapper output format : gender is the key, the value is formed
 * by concatenating the name, age and the score
*/
        public class PartitionMapper extends
                        Mapper<LongWritable, Text, Text, Text> {


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


                        String[] tokens = value.toString().split("\t");


                        String gender = tokens[2].toString();
                        String nameAgeScore = tokens[0]+"\t"+tokens[1]+"\t"+tokens[3];


                        context.write(new Text(gender), new Text(nameAgeScore));
                }//end of map
        }//end of mapper
```

```
package com. evenkat;


import java.io.IOException;


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

```
public   class PartitionReducer extends Reducer<Text, Text, Text, Text> {
         @Override
         public void reduce(Text key, Iterable<Text> values, Context context)
                           throws IOException, InterruptedException {

                  int maxScore = Integer.MIN_VALUE;

                  String name = " ";
                  String age = " ";
                  String gender = " ";
                  int score = 0;
//iterating through the values corresponding to a particular key
                  for(Text val: values){

                           String [] valTokens = val.toString().split("\t");
                           score = Integer.parseInt(valTokens[2]);

/*if the new score is greater than the current maximum score,
* update the fields as they will be the output of the reducer
* after all the values are processed for a particular key
*/
                             if(score > maxScore){
                                    name = valTokens[0];
                                    age = valTokens[1];
                                    gender = key.toString();
          maxScore = score;
```

```
                    } //end of if
                } // end of for
                context.write(new Text(name), new Text("age- "+age+"\t"+
                gender+"\tscore-"+maxScore));
            }// end of Reduce
} //end of Reducer
```

```
package com. evenkat;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

/* AgePartitioner is a custom Partitioner to partition the data according to age.
 * The age is a part of the value from the input file.
 * The data is partitioned based on the range of the age.
 * The use case needs 3 partitions, the first partition contains the information
 * where the age is less than 20. The second partition contains data with age
 * ranging between 20 and 50 and the third partition contains data
 * where the age is >50.
*/
        public class AgePartitioner extends Partitioner<Text, Text> {

                @Override
                public int getPartition(Text key, Text value, int numReduceTasks) {

                        String [] nameAgeScore = value.toString().split("\t");
                        String age = nameAgeScore[1];
                        int ageInt = Integer.parseInt(age);
                        //this is done to avoid performing mod with 0

                        if(numReduceTasks == 0)

                                return 0;


                        //if the age is <20, assign partition 0

                        if(ageInt <=20){

                                return 0;

                        }

                        //else if the age is between 20 and 50, assign partition 1

                        if(ageInt >20 && ageInt <=50){


                                return 1;

                        }

                        //otherwise assign partition 2

                        else

                                return 2;

                } //end of partition function

        } //end of Partitioner
```