mkdir grade

nano StudentGrade.java

nano students.txt

javac -classpath `hadoop classpath` -d output StudentGrade.java

jar -cvf studentgrade.jar -C output .

hdfs dfs -mkdir /user/hadoop/output_studentgrade

hdfs dfs -put students.txt /user/hadoop/input

hadoop jar studentgrade.jar StudentGrade /user/hadoop/input /user/hadoop/output_studentgrade

hdfs dfs -cat /user/hadoop/output_studentgrade/part-r-00000

Exp 2

Java file :

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class WordCount {

  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
```

```
            context.write(word, one);
        }
      }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
      private IntWritable result = new IntWritable();

      public void reduce(Text key, Iterable<IntWritable> values, Context context)
          throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
          sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
      }
    }

    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "word count");

      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.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);
    }
}
```

Exp 3

Java :

```
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.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;

public class StudentGrade {

  public static class GradeMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] parts = value.toString().split(",");
      if (parts.length == 2) {
        String studentName = parts[0].trim();
        double score = Double.parseDouble(parts[1].trim());
        String grade;

        if (score >= 90) {
          grade = "A";
        } else if (score >= 80) {
          grade = "B";
        } else if (score >= 70) {
          grade = "C";
        } else if (score >= 60) {
          grade = "D";
        } else {
          grade = "F";
        }

        context.write(new Text(studentName), new Text(grade));
      }
    }
```

```
    }

    public static class GradeReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            for (Text val : values) {
                context.write(key, val);
                break; // Only one value per student expected
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: StudentGrade <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Student Grade Calculation");

        job.setJarByClass(StudentGrade.class);
        job.setMapperClass(GradeMapper.class);
        job.setReducerClass(GradeReducer.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);
    }
}
```

Exp 4

Java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
```

```
import org.apache.hadoop.io.IntWritable;
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;

// Main class
public class MaxTemperature {

  // Mapper Class
  public static class TemperatureMapper extends Mapper<Object, Text, IntWritable, DoubleWritable> {
    private IntWritable year = new IntWritable();
    private DoubleWritable temperature = new DoubleWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] fields = value.toString().split(","); // Assuming CSV format
      if (fields.length < 3) return; // Skip malformed lines

      try {
        year.set(Integer.parseInt(fields[0].trim()));         // Extract Year
        temperature.set(Double.parseDouble(fields[2].trim())); // Extract Temperature
        context.write(year, temperature);
      } catch (NumberFormatException e) {
        // Skip invalid data
      }
    }
  }

  // Reducer Class
  public static class TemperatureReducer extends Reducer<IntWritable, DoubleWritable, IntWritable, DoubleWritable> {
    public void reduce(IntWritable key, Iterable<DoubleWritable> values, Context context)
        throws IOException, InterruptedException {
      double maxTemp = Double.MIN_VALUE;
      for (DoubleWritable val : values) {
```

```
            maxTemp = Math.max(maxTemp, val.get());
        }
        context.write(key, new DoubleWritable(maxTemp));
    }
  }

  // Driver Code
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Max Temperature Finder");

    job.setJarByClass(MaxTemperature.class);
    job.setMapperClass(TemperatureMapper.class);
    job.setReducerClass(TemperatureReducer.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(DoubleWritable.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(DoubleWritable.class);

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

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

Exp 5

Java:

```
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.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.ArrayList;
import java.util.List;

public class MatrixMultiplication {

  // Mapper Class
  public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] parts = value.toString().split("\\s+");
      if (parts.length < 4) return;

      String matrixName = parts[0]; // 'A' or 'B'
      int i = Integer.parseInt(parts[1]);
      int j = Integer.parseInt(parts[2]);
      double val = Double.parseDouble(parts[3]);

      Configuration conf = context.getConfiguration();
      int p = Integer.parseInt(conf.get("p")); // Columns of B

      if (matrixName.equals("A")) {
        for (int k = 0; k < p; k++) {
          context.write(new Text(i + "," + k), new Text("A," + j + "," + val));
        }
      } else if (matrixName.equals("B")) {
        int m = Integer.parseInt(conf.get("m")); // Rows of A
        for (int k = 0; k < m; k++) {
          context.write(new Text(k + "," + j), new Text("B," + i + "," + val));
        }
      }
    }
  }

  // Reducer Class
  public static class MatrixReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
      List<String> aValues = new ArrayList<>();
      List<String> bValues = new ArrayList<>();
```

```
        for (Text val : values) {
            String valueStr = val.toString();
            if (valueStr.startsWith("A")) {
                aValues.add(valueStr);
            } else {
                bValues.add(valueStr);
            }
        }

        double sum = 0;
        for (String a : aValues) {
            String[] aParts = a.split(",");
            int aCol = Integer.parseInt(aParts[1]);
            double aVal = Double.parseDouble(aParts[2]);

            for (String b : bValues) {
                String[] bParts = b.split(",");
                int bRow = Integer.parseInt(bParts[1]);
                double bVal = Double.parseDouble(bParts[2]);

                if (aCol == bRow) {
                    sum += aVal * bVal;
                }
            }
        }

        context.write(key, new Text(Double.toString(sum)));
    }
}

// Main Method
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("m", "2"); // Rows of A
    conf.set("n", "3"); // Columns of A / Rows of B
    conf.set("p", "2"); // Columns of B

    Job job = Job.getInstance(conf, "Matrix Multiplication");
```

```
        job.setJarByClass(MatrixMultiplication.class);
        job.setMapperClass(MatrixMapper.class);
        job.setReducerClass(MatrixReducer.class);

        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);
    }
}
```

Exp 6

Java:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class MaxElectricityConsumption {

    // Mapper Class
    public static class MaxConsumptionMapper extends Mapper<Object, Text, Text, IntWritable> {
        private Text year = new Text();
        private IntWritable consumption = new IntWritable();

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

```
        String[] fields = value.toString().split(",");
        if (fields.length == 3) {
            year.set(fields[0].trim()); // Extract Year
            consumption.set(Integer.parseInt(fields[2].trim())); // Extract Consumption
            context.write(year, consumption);
        }
    }
}

// Reducer Class
public static class MaxConsumptionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int maxConsumption = Integer.MIN_VALUE;
        for (IntWritable val : values) {
            maxConsumption = Math.max(maxConsumption, val.get());
        }
        context.write(key, new IntWritable(maxConsumption));
    }
}

// Driver Code
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Max Electricity Consumption");

    job.setJarByClass(MaxElectricityConsumption.class);
    job.setMapperClass(MaxConsumptionMapper.class);
    job.setReducerClass(MaxConsumptionReducer.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);
```

```
    }
}
```

Exp 7

Java:

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

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class MaxElectricalConsumption {

  // Mapper class to extract month and consumption data
  public static class ConsumptionMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text month = new Text();
    private IntWritable consumption = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      // Expected format: "Month Consumption"
      if (itr.hasMoreTokens()) {
        String monthString = itr.nextToken(); // Month (e.g., "January")
        if (itr.hasMoreTokens()) {
          String consumptionString = itr.nextToken(); // Consumption value
          try {
            month.set(monthString);
            consumption.set(Integer.parseInt(consumptionString));
            context.write(month, consumption);
          } catch (NumberFormatException e) {
            System.err.println("Skipping invalid record: " + value.toString());
```

```
            }
        }
    }
}

// Reducer class to calculate the maximum consumption across all months
public static class MaxConsumptionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private Text resultMonth = new Text();
    private IntWritable resultConsumption = new IntWritable();
    private String maxMonth = "";
    private int maxConsumption = Integer.MIN_VALUE;

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        for (IntWritable val : values) {
            if (val.get() > maxConsumption) {
                maxConsumption = val.get();
                maxMonth = key.toString();
            }
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        resultMonth.set(maxMonth);
        resultConsumption.set(maxConsumption);
        context.write(resultMonth, resultConsumption);
    }
}

// Driver code
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Maximum Electrical Consumption");

    job.setJarByClass(MaxElectricalConsumption.class);
    job.setMapperClass(ConsumptionMapper.class);
    job.setReducerClass(MaxConsumptionReducer.class);
```

```
        job.setOutputKeyClass(Text.class); // Month
        job.setOutputValueClass(IntWritable.class); // Consumption

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

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

Exp 8

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MovieTagsAnalysis {

    // Mapper Class
    public static class TagsMapper extends Mapper<Object, Text, Text, Text> {
        private Text movieID = new Text();
        private Text tag = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] fields = value.toString().split(",");
            if (fields.length >= 3) {
                movieID.set(fields[1]); // Movie ID is in the second column
```

```
            tag.set(fields[2]);     // Tag is in the third column
            context.write(movieID, tag);
        }
    }
}

// Reducer Class
public static class TagsReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        List<String> tagsList = new ArrayList<>();
        for (Text val : values) {
            tagsList.add(val.toString());
        }
        context.write(key, new Text(String.join(", ", tagsList)));
    }
}

// Driver Code
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Movie Tags Analysis");

    job.setJarByClass(MovieTagsAnalysis.class);
    job.setMapperClass(TagsMapper.class);
    job.setReducerClass(TagsReducer.class);

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

    FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path (e.g., tags.csv)
    FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output path

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

Exp 9

```
import java.io.IOException;
```

```
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class UberTripsAnalysis {

  // Mapper Class
  public static class UberMapper extends Mapper<Object, Text, Text, Text> {
    private Text baseId = new Text();
    private Text dateAndTrips = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] tokens = value.toString().split("\\s+"); // Split by whitespace
      if (tokens.length == 4) { // Expected: base_id date vehicles trips
        String base = tokens[0];    // Base ID
        String date = tokens[1];    // Date
        String trips = tokens[3];   // Trips (assumed at 4th column)

        baseId.set(base);
        dateAndTrips.set(date + "," + trips);
        context.write(baseId, dateAndTrips);
      }
    }
  }

  // Reducer Class
  public static class UberReducer extends Reducer<Text, Text, Text, Text> {
    private Text maxTripDate = new Text();

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
      int maxTripsValue = Integer.MIN_VALUE;
      String maxDay = "";
```

```
        for (Text val : values) {
            String[] parts = val.toString().split(",");
            if (parts.length == 2) {
                String date = parts[0];
                int trips = Integer.parseInt(parts[1]);

                if (trips > maxTripsValue) {
                    maxTripsValue = trips;
                    maxDay = date;
                }
            }
        }

        maxTripDate.set(maxDay + " " + maxTripsValue);
        context.write(key, maxTripDate);
    }
}

// Driver Code
public static void main(String[] args) throws Exception {
    if (args.length < 2) {
        System.err.println("Usage: UberTripsAnalysis <input file> <output dir>");
        System.exit(1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Uber Trips Analysis");

    job.setJarByClass(UberTripsAnalysis.class);
    job.setMapperClass(UberMapper.class);
    job.setReducerClass(UberReducer.class);

    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);
  }
}
```

Exp 10

```
import java.io.IOException;

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TitanicAnalysis {

  // Mapper Class
  public static class TitanicMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] columns = value.toString().split(",");

      // Skip header
      if (columns[0].equals("PassengerId")) {
        return;
      }

      try {
        int survived = Integer.parseInt(columns[1]); // 0 or 1
        int pclass = Integer.parseInt(columns[2]);    // 1, 2, or 3
        String sex = columns[5];                  // male/female
        String ageStr = columns[6];               // Age (can be empty)

        // Emit age of deceased passengers grouped by gender
```

```
            if (survived == 0 && !ageStr.isEmpty()) {
                context.write(new Text("Age_" + sex), new Text(ageStr + ",1"));
            }

            // Emit survivors grouped by class
            if (survived == 1) {
                context.write(new Text("Class_" + pclass), new Text("1,1"));
            }

        } catch (Exception e) {
            System.err.println("Skipping invalid record: " + value.toString());
        }
    }
}

// Reducer Class
public static class TitanicReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double totalAge = 0;
        int count = 0;

        for (Text val : values) {
            String[] parts = val.toString().split(",");
            if (parts.length == 2) {
                double value = Double.parseDouble(parts[0]);
                int occurrences = Integer.parseInt(parts[1]);

                totalAge += value;
                count += occurrences;
            }
        }

        // Output average age for deceased passengers
        if (key.toString().startsWith("Age_")) {
            double avgAge = (count == 0) ? 0 : totalAge / count;
            context.write(key, new Text(String.format("Average Age: %.2f", avgAge)));
        }
        // Output survivor count per class
```

```
        else if (key.toString().startsWith("Class_")) {
            context.write(key, new Text("Total Survivors: " + count));
        }
    }
}

// Main Method
public static void main(String[] args) throws Exception {
    if (args.length < 2) {
        System.err.println("Usage: TitanicAnalysis <input file> <output dir>");
        System.exit(1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Titanic Data Analysis");

    job.setJarByClass(TitanicAnalysis.class);
    job.setMapperClass(TitanicMapper.class);
    job.setReducerClass(TitanicReducer.class);

    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);
}
}
```

Exp 11

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
```

```
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;

public class MaxTemperature {

  // Mapper Class
  public static class MaxTempMapper extends Mapper<Object, Text, Text, Text> {

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      String[] columns = value.toString().split("\\s+"); // Split by whitespace
      if (columns.length != 3) return; // Skip invalid lines

      String year = columns[0];   // Extract year
      String month = columns[1];  // Extract month
      String temp = columns[2];   // Extract temperature

      // Emit (year, month_temperature)
      context.write(new Text(year), new Text(month + "_" + temp));
    }
  }

  // Reducer Class
  public static class MaxTempReducer extends Reducer<Text, Text, Text, IntWritable> {

    public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
      int maxTemp = Integer.MIN_VALUE;
      String maxMonth = "";

      // Find the month with the highest temperature for each year
      for (Text val : values) {
        String[] parts = val.toString().split("_");
        String month = parts[0];
        int temperature = Integer.parseInt(parts[1]);
```

```
            if (temperature > maxTemp) {
                maxTemp = temperature;
                maxMonth = month;
            }
        }

        // Emit (Year, Month MaxTemperature)
        context.write(new Text(key.toString() + " " + maxMonth), new IntWritable(maxTemp));
    }
  }

  // Main Method to configure and run the MapReduce job
  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
        System.err.println("Usage: MaxTemperature <input file> <output dir>");
        System.exit(1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Max Temperature Finder");

    job.setJarByClass(MaxTemperature.class);
    job.setMapperClass(MaxTempMapper.class);
    job.setReducerClass(MaxTempReducer.class);

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

    FileInputFormat.addInputPath(job, new Path(args[0]));  // Input file path
    FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

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

Exp 12

```
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 org.apache.hadoop.io.Writable;
import java.io.DataInput;
import java.io.DataOutput;

public class Aggregate {

  // Writable class to aggregate Titanic data
  public static class TitanicData implements Writable {
    int totalPassengers;
    int survived;
    double totalAge;

    public TitanicData() {
      this.totalPassengers = 0;
      this.survived = 0;
      this.totalAge = 0.0;
    }

    // Add passenger data to the aggregate
    public void addPassenger(int survived, double age) {
      totalPassengers++;
      if (survived == 1) {
        this.survived++;
      }
      if (age > 0) {
```

```
            this.totalAge += age;
        }
    }

    // Calculate the survival rate
    public double getSurvivalRate() {
        if (totalPassengers == 0) return 0.0;
        return (double) survived / totalPassengers * 100;
    }

    // Calculate the average age of passengers
    public double getAverageAge() {
        if (totalPassengers == 0) return 0.0;
        return totalAge / totalPassengers;
    }

    // Get the total number of passengers
    public int getTotalPassengers() {
        return totalPassengers;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(totalPassengers);
        out.writeInt(survived);
        out.writeDouble(totalAge);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        totalPassengers = in.readInt();
        survived = in.readInt();
        totalAge = in.readDouble();
    }
}

// Mapper class to process Titanic dataset and map data by passenger class
public static class TitanicMapper extends Mapper<LongWritable, Text, Text, TitanicData> {
```

```
    private Text classKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] tokens = line.split(",");
        try {
            String pclass = tokens[2];
            int survived = Integer.parseInt(tokens[1]);
            double age = tokens[5].equals("") ? 0 : Double.parseDouble(tokens[5]);

            classKey.set(pclass);
            TitanicData data = new TitanicData();
            data.addPassenger(survived, age);
            context.write(classKey, data);
        } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
            // Handle invalid records gracefully
        }
    }
}

// Reducer class to aggregate data by passenger class
public static class TitanicReducer extends Reducer<Text, TitanicData, Text, Text> {
    private Text result = new Text();

    public void reduce(Text key, Iterable<TitanicData> values, Context context) throws IOException, InterruptedException {
        TitanicData aggregatedData = new TitanicData();
        for (TitanicData data : values) {
            aggregatedData.totalPassengers += data.getTotalPassengers();
            aggregatedData.survived += data.survived;
            aggregatedData.totalAge += data.totalAge;
        }

        double survivalRate = aggregatedData.getSurvivalRate();
        double avgAge = aggregatedData.getAverageAge();
        int totalPassengers = aggregatedData.getTotalPassengers();

        String resultString = String.format("Total Passengers: %d, Survival Rate: %.2f%%, Average Age: %.2f",
                                    totalPassengers, survivalRate, avgAge);
```

```
            result.set(resultString);
            context.write(key, result);
        }
    }

    // Main method to configure and run the MapReduce job
    public static void main(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("Usage: Aggregate <input file> <output dir>");
            System.exit(1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Titanic Aggregator");

        job.setJarByClass(Aggregate.class);
        job.setMapperClass(TitanicMapper.class);
        job.setReducerClass(TitanicReducer.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input file path
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

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