Implementation Steps:

1. Create the CustomerMapper, TransactionMapper, JoinReducer, CustomerCountMapper, CustomerCountReducer and CustomerTransactionDriver classes and generate a .jar file. The complete implementation and code are explained in the section 2.3. A separate file 'MapReduce_Code.docx' is submitted along with the coursework.

2. Start the HDFS file system as shown below and confirm if all the services are running or not.

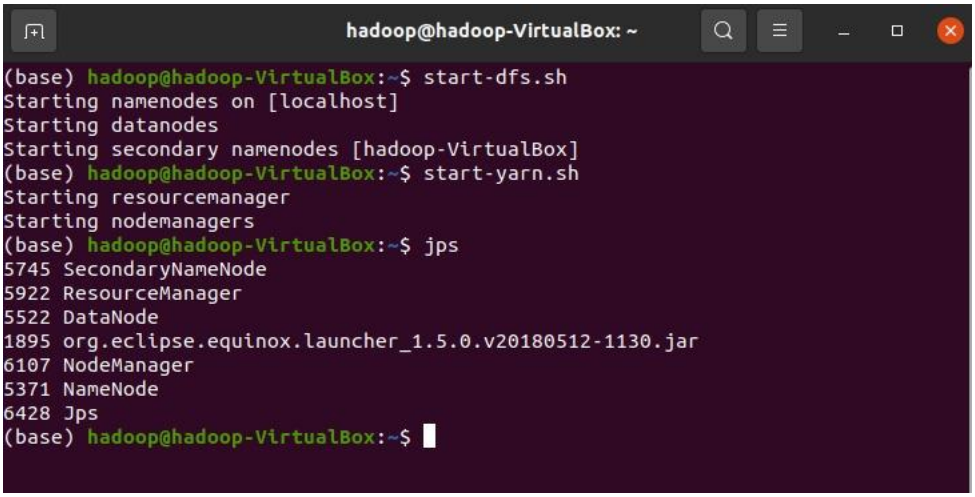


Fig-2

3. Create the input1 and input2 directories in the HDFS file system as shown below.

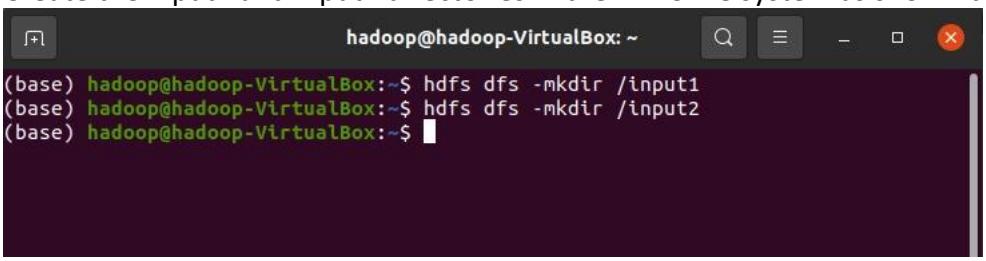


Fig-3

4. Copy and place the input files 'CustomerDemographic.csv' and 'Transactions.csv' in the folders 'input1' and 'input2' respectively using the commands as shown below.

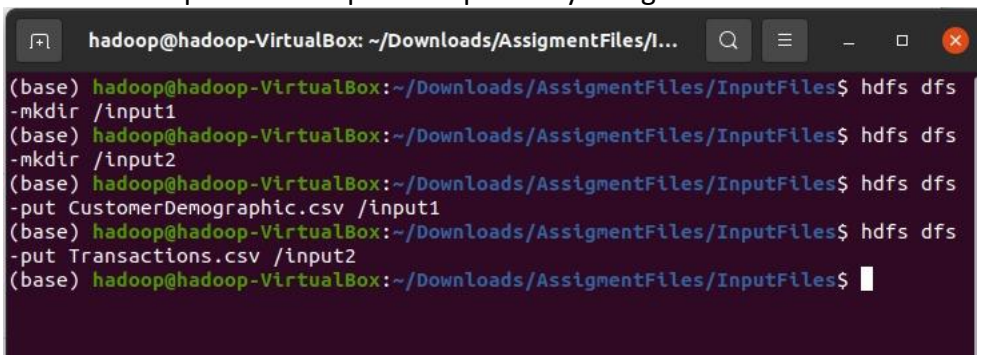


Fig-4

5. Run the jar file using the below command.
   hadoop jar <jar file name> <input1 path> <input2 path> <output path>

Example: hadoop jar CustomerDemographicsAnalysis.jar /input1 /input2 /output

Note: There is no need to provide the driver class name, as the jar file already consists of those details. It is a good practice to avoid using driver class name, when there is only one driver class. So that, it would be easier for the second person to run the job as they don't need to worry about the driver class name.

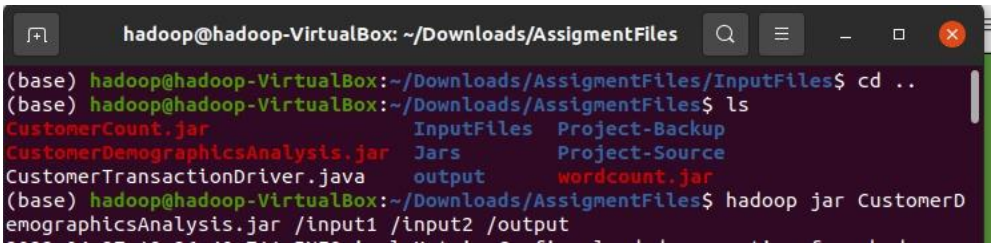


Fig-5

6. When the jobs are run, the end of the run will display the success message as shown below.

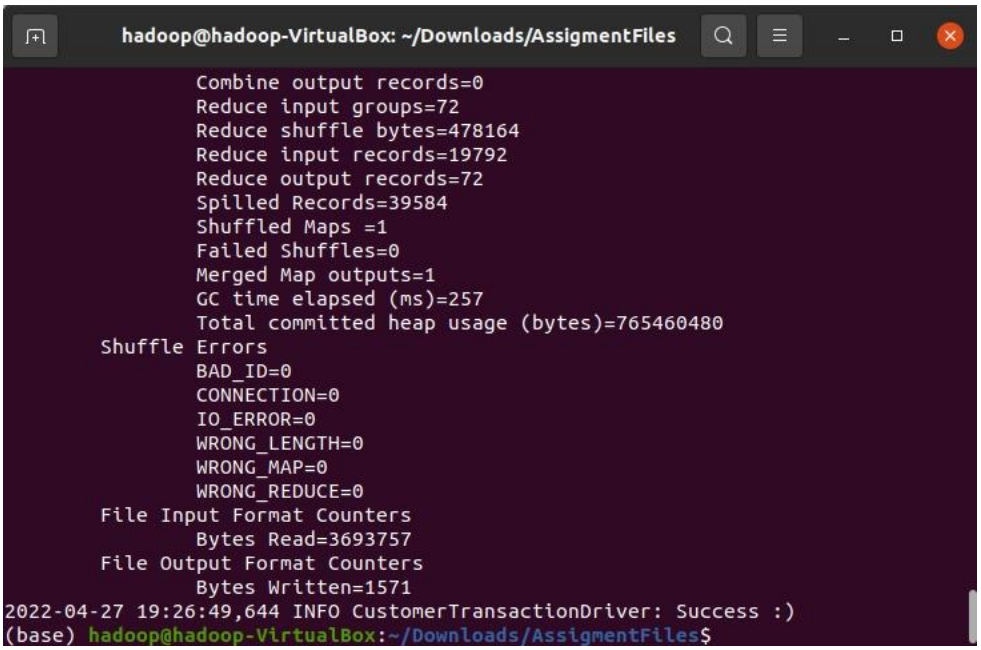
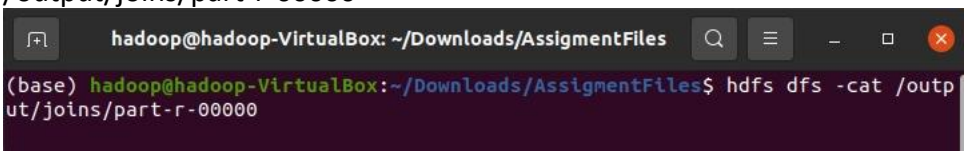


Fig-6

7. The code is developed to put the output of the inner join MapReduce job (first MR job) in following location.
/output/joins/part-r-00000



```
hadoop@hadoop-VirtualBox: ~/Downloads/AssigmentFiles
,$507.58,38750
3499,Shelton Tewkesberrie,Male,29,1979-06-17,42.00 ,NA,Manufacturing,Mass Custo
mer,N,Yes,7,18403,67,09/11/2017,TRUE,Approved,Norco Bicycles,Road,medium,medium
,544.05,$376.84,38859
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,6309,69,09/01/2017,TRUE,Approved,Giant Bicycles,Road,medium,medium,79
2.9,$594.68,33879
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,8562,33,08/08/2017,FALSE,Approved,Giant Bicycles,Standard,medium,smal
l,1311.44,"$1,167.18",33888
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,2463,2,25/04/2017,FALSE,Approved,Solex,Standard,medium,medium,71.49,$
53.62,41245
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,8416,74,16/02/2017,FALSE,Approved,WeareA2B,Standard,medium,medium,122
8.07,$400.91,36668
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,14870,22,11/01/2017,FALSE,Approved,WeareA2B,Standard,medium,medium,60
.34,$45.26,34165
3500,Josy Fleeman,Female,71,1967-07-21,54.00 ,NA,Entertainment,Affluent Custome
r,N,No,17,19836,40,14/03/2017,TRUE,Approved,OHM Cycles,Standard,high,medium,145
8.17,$874.90,38750
(base) hadoop@hadoop-VirtualBox:~/Downloads/AssigmentFiles$
```

Fig-7

8. The output of the final job is placed in the following location. /output/output/part-r-00000

```
hadoop@hadoop-VirtualBox: ~/Downloads/AssigmentFiles
(base) hadoop@hadoop-VirtualBox:~/Downloads/AssigmentFiles$ hdfs dfs -cat /outpu
t/output/part-r-00000
01/2017,large,no,175
01/2017,large,yes,167
01/2017,medium,no,561
01/2017,medium,yes,525
01/2017,small,no,117
01/2017,small,yes,118
02/2017,large,no,150
02/2017,large,yes,147
02/2017,medium,no,536
02/2017,medium,yes,535
02/2017,small,no,117
02/2017,small,yes,127
03/2017,large,no,168
03/2017,large,yes,159
03/2017,medium,no,533
03/2017,medium,yes,536
03/2017,small,no,115
03/2017,small,yes,118
04/2017,large,no,151
04/2017,large,yes,163
04/2017,medium,no,526
04/2017,medium,yes,570
```

Fig-8

2.3. JAVA Classes Implementation

1. Create the project as shown below figure -9. Four additional jars were used in the development of this project.
    a. hadoop-common-3.0.3.jar – available at </usr/local/hadoop/share/haddoop/common>
    b. hadoop-mapreduct-client-core-3.0.3.jar – available at </usr/local/hadoop/share/hadoop/mapreduce>

c. hadoop-mapreduct-client-jobclient-3.0.3.jar – available at </usr/local/hadoop/share/hadoop/mapreduce>
d. log4j-1.2.17.jar – available at </home/Hadoop/Downloads/AssignmentFiles/Jars>

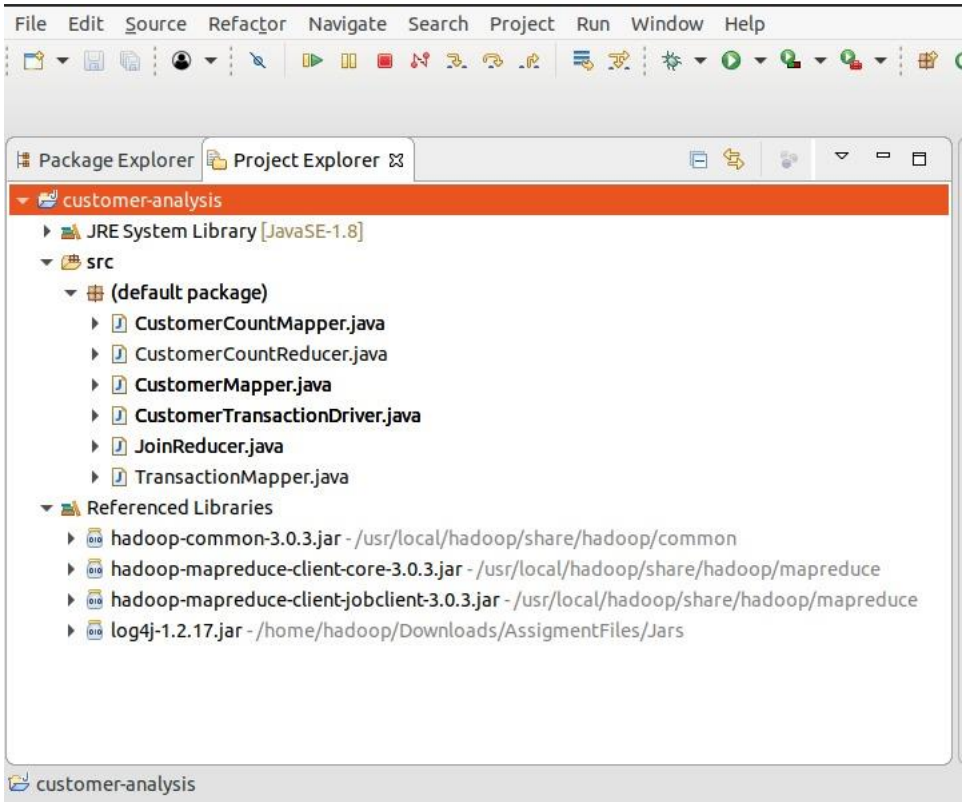


Fig.9

2. CustomerMapper

CustomerMapper.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

/**
 *
 * @author Uma
 * CustomerMapper class is used read the customer related data from file and
 * generate key, value pairs
 *
 */
public class CustomerMapper extends Mapper<LongWritable, Text, IntWritable, Text> {

    private static final Logger _log = LogManager.getLogger(CustomerMapper.class);

    private final Pattern separator = Pattern.compile(",");

    private final IntWritable custIdAsKey = new IntWritable();
    private final Text custRecordAsValue = new Text();
```

```java
    /**
     * Process input line and write city and count in output
     * @param offset  input line offset
     * @param line    input line
     * @param context mapper context
     */
    @Override
    protected void map(LongWritable offset, Text line, Context context) throws IOException, InterruptedException {
        try {
            // Split input line to get customer id
            final List<String> columns = Arrays.stream(separator.split(line.toString(), 4)).collect(Collectors.toList());

            try {
                // Extract customer id from customer record and parse it
                int customerId = Integer.parseInt(columns.remove(0));

                // Add customer record indicator (will use in reducer)
                columns.add(0, "C");

                // Write customer id as key and customer record as value
                custIdAsKey.set(customerId);
                custRecordAsValue.set(String.join(",", columns));
                context.write(custIdAsKey, custRecordAsValue);

            } catch (NumberFormatException ignore) {
            }
        } catch (Exception e) {
            _log.error("Failed to process record: " + line.toString(), e);
            throw e;
        }
    }
}
```

3. TransactionMapper

TransactionMapper.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

/**
 *
 * @author Uma
 * TransactionMapper class is used read the Transaction related data from file and
 * generate key, value pairs
 */
public class TransactionMapper extends Mapper<LongWritable, Text, IntWritable, Text> {

    private static final Logger _log = LogManager.getLogger(TransactionMapper.class);

    private final Pattern separator = Pattern.compile(",");

    private final IntWritable custIdAsKey = new IntWritable();
    private final Text transactionAValue = new Text();

    /**
     * Process input line and write city and count in output
     *
     * @param offset  input line offset
     * @param line    input line
```

TransactionMapper.java

```java
     * @param context mapper context
     */
    @Override
    protected void map(LongWritable offset, Text line, Context context) throws IOException, InterruptedException {

        try {
            // Split input line to get customer id
            final List<String> columns = Arrays.stream(separator.split(line.toString(), 4)).collect(Collectors.toList());

            try {
                // Extract customer id from transaction record and convert to integer
                int customerId = Integer.parseInt(columns.remove(2));

                // Add transaction record indicator (will use in reducer)
                columns.add(0, "T");

                // Write customer id as key and transaction record as value
                custIdAsKey.set(customerId);
                // Converting transaction list values as comma separated string and
                // setting to transaction value
                transactionAValue.set(String.join(",", columns));
                // Writing customer id key and transaction value to output
                context.write(custIdAsKey, transactionAValue);

            }catch (NumberFormatException ignore) {
            }

        } catch (Exception e) {
            _log.error("Failed to process record: " + line.toString(), e);
            throw e;
        }
    }
}
```

4. JoinReducer

JoinReducer.java

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

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;

/**
 *
 * @author Uma
 * JoinReducer class is used to join the output of the CustomerMapper and TransactionMapper
 *
 */
public class JoinReducer extends Reducer<IntWritable, Text, IntWritable, Text> {

    private final Pattern separator = Pattern.compile(",");
    private final Text joinedRecordAsValue = new Text();


    @Override
    protected void reduce(IntWritable custId, Iterable<Text> records, Context context) throws IOException, InterruptedException {

        String customerRecord = null;
        List<String> customerTransactions = new ArrayList<>();

        // Collect customer records and transactions separately based on flag 'C' & 'T'
        // respectively at first position of each value
        for(Text record: records) {
            String[] typeRecord = separator.split(record.toString(), 2);
            if("C".equals(typeRecord[0])) {
                customerRecord= typeRecord[1];
            } else if("T".equals(typeRecord[0])) {
                customerTransactions.add(typeRecord[1]);
            }
        }

        if (customerRecord != null) {
            // Joining customer records with each transaction record
            for (String transaction : customerTransactions) {
                joinedRecordAsValue.set(customerRecord + "," + transaction);
                // Writing joined data along with customer id to output
                context.write(custId, joinedRecordAsValue);
            }
        }
    }
}
```

5. CustomerCountMapper

CustomerCountMapper.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import java.io.IOException;
import java.util.regex.Pattern;

/**
 *
 * @author Uma
 * CustomerCountMapper class is used to obtain the joined output from first MR job
 * then create key value pairs for each record
 *
 */
public class CustomerCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private static final Logger _log = LogManager.getLogger(CustomerCountMapper.class);

    private final Pattern separator = Pattern.compile(",");

    private final Text KEY = new Text();
    private final IntWritable VALUE = new IntWritable(1);

    @Override
    protected void map(LongWritable offset, Text joinedRecord, Context context) throws IOException, InterruptedException {

        try {
            // Split input line to get customer id
            final String[] columns = separator.split(joinedRecord.toString().toLowerCase());
            // Getting Date, Product Size and Car owned information using their positions in file
            if (!columns[14].isEmpty() && !columns[20].isEmpty() && !columns[10].isEmpty()) {
```


```java
            // Getting Date, Product Size and Car owned information using their positions in file
            if (!columns[14].isEmpty() && !columns[20].isEmpty() && !columns[10].isEmpty()) {
```


```java
                // Extracting month and year from date
                String date = columns[14].split("/", 2)[1];
                String prodSize = columns[20];
                String ownsCar = columns[10];
                //Setting extracted date, product size and owns car as key
                KEY.set(date + "," + prodSize + "," + ownsCar);

                // Write date, product size and owns car as key and 1 as value
                context.write(KEY, VALUE);
            }
        } catch (Exception e) {
            _log.error("Failed to process record: " + joinedRecord.toString(), e);
            try {
                throw e;
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    }

}
```

6. CustomerCountReducer

CustomerCountReducer.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

/**
 *
 * @author Uma
 * CustomerCountReducer class is used to aggregate the output obtained from CustomerCount Mapper
 *
 */
public class CustomerCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private final IntWritable COUNT = new IntWritable();

    @Override
    protected void reduce(Text dateProdTypeOwnsCar, Iterable<IntWritable> counts, Context context) throws IOException, InterruptedException {

        // Count number of customer transactions
        int sum = 0;
        for (IntWritable c : counts) {
            sum += c.get();
        }
        COUNT.set(sum);

        // Writing date, product size and owns car along with count to output
        context.write(dateProdTypeOwnsCar, COUNT);
    }
}
```

7. CustomerTransactionDriver

CustomerTransactionDriver.java

```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.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

/**
 *
 * @author Uma
 * CustomerTransactionDriver class is the main driving class used to run the two map reduce jobs
 * sequentially by taking input files
 *
 */
public class CustomerTransactionDriver {

    private static final Logger _log = LogManager.getLogger(CustomerTransactionDriver.class);

    public static void main(String[] args) {

        // Displaying error message if user did not pass 3 arguments which are 2 input
        // paths and 1 output path
        if (args.length < 3) {
            _log.error(String.format("Usage: %s <customer-file-path> <transaction-file-path> <output-path>",
                    CustomerTransactionDriver.class.getSimpleName()));
            System.exit(-1);
```

CustomerTransactionDriver.java

```java
        try {
            // Creating configuration instance and setting comma as mapreduce output separator
            Configuration conf = new Configuration();
            conf.set("mapreduce.output.textoutputformat.separator", ",");

            // Creating job configuration instance
            JobConf jobConf = new JobConf(conf);
            // Creating job instance with job name
            Job job = Job.getInstance(jobConf, "Customer Inner Join Transaction Job");
            job.setJarByClass(CustomerTransactionDriver.class);

            // Setting input paths and map corresponding mapper classes along with inputo format
            // CustomerDemographic file will be processed by CustomerMapper
            // Transactions file will be processed by TransactionMapper
            MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, CustomerMapper.class);
            MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, TransactionMapper.class);

            // Appending / at the end of output path if it is not given by user
            String outputPath = args[2];
            if (!outputPath.endsWith("/") && !outputPath.endsWith("\\")) {
                outputPath += '/';
            }

            // Setting output path
            FileOutputFormat.setOutputPath(job, new Path(outputPath + "joins/"));

            // Setting output type, format classes and reducer class
            job.setMapOutputKeyClass(IntWritable.class);
            job.setMapOutputValueClass(Text.class);
            job.setOutputKeyClass(IntWritable.class);
            job.setOutputValueClass(Text.class);
            job.setOutputFormatClass(TextOutputFormat.class);
            job.setReducerClass(JoinReducer.class);
```

CustomerTransactionDriver.java

```java

            // Submit first job
            if (job.waitForCompletion(true)) {

                // Creating configuration instance and setting comma as mapreduce output separator
                conf = new Configuration();
                conf.set("mapreduce.output.textoutputformat.separator", ",");

                // Prepare job instance
                jobConf = new JobConf(conf);
                // Creating job instance with job names
                job = Job.getInstance(jobConf, "Count Customers By Month, ProductType, OwnsCar");
                job.setJarByClass(CustomerTransactionDriver.class);

                // Setting output from first MR job as input for second MR job
                FileInputFormat.addInputPath(job, new Path(outputPath + "joins/"));
                FileOutputFormat.setOutputPath(job, new Path(outputPath + "output/"));

                // Setting output type and format classes
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(IntWritable.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                // Setting mapper and reduce implementation classes
                job.setMapperClass(CustomerCountMapper.class);
                job.setReducerClass(CustomerCountReducer.class);

                // Start job and wait for complete
                if (job.waitForCompletion(true)) {
                    // Showing Success message
                    _log.info("Success :)");
                } else {
                    // Showing fail message
                    _log.error("Fail :(");
                }
            } else {
                // Showing fail message
                _log.error("Fail :(");
            }
            // Exit the process once job is completed
            System.exit(job.isSuccessful() ? 0 : 1);
        } catch (Exception e) {
            _log.error("Error", e);
            System.exit(1);
        }
    }
}
```