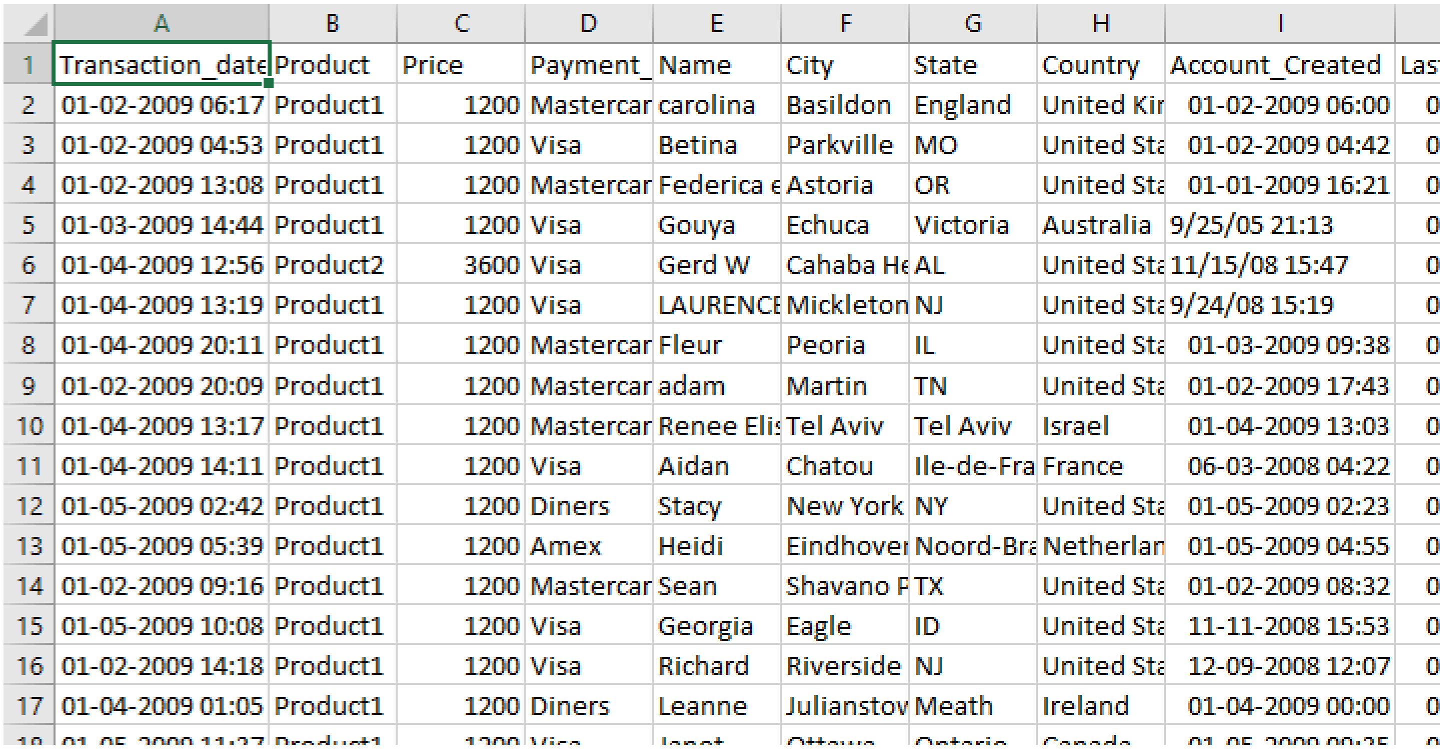
Experiment No:14

Aim: Design a distributed application using Map Reduce which processes a log file of a system.

Theory: In this tutorial, you will learn to use Hadoop with MapReduce Examples. The input data used is SalesJan2009.csv. It contains Sales related information like Product name, price, payment mode, city, country of client etc. The goal is to **Find out Number of Products Sold in Each Country.**

First Hadoop MapReduce Program

Now in this MapReduce tutorial, we will create our first Java MapReduce program:

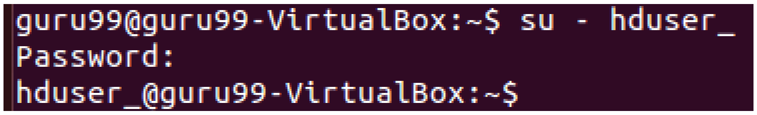


	A	B	C	D	E	F	G	H	I	
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Las
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	0
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	0
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	0
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	0
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	0
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	0
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	0
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	0
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	0
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	0
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	0
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	0
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	0
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	0
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	0
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	0
18	01-05-2009 11:37	Product1	1200	Visa	Janet	Ottawa	Ontario	Canada	01-05-2009 09:25	0

Data of SalesJan2009

Ensure you have Hadoop installed. Before you start with the actual process, change user to 'hduser' (id used while Hadoop configuration, you can switch to the userid used during your Hadoop programming config).

su - hduser_

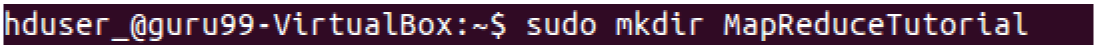


```
guru99@guru99-VirtualBox:~$ su - hduser_
Password:
hduser_@guru99-VirtualBox:~$
```

Step 1)

Create a new directory with name **MapReduceTutorial** as shwon in the below MapReduce example

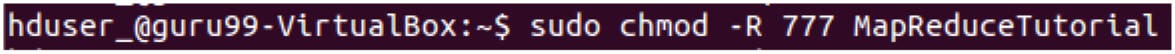
```
sudo mkdir MapReduceTutorial
```



```
hduser_@guru99-VirtualBox:~$ sudo mkdir MapReduceTutorial
```

Give permissions

```
sudo chmod -R 777 MapReduceTutorial
```



```
hduser_@guru99-VirtualBox:~$ sudo chmod -R 777 MapReduceTutorial
```

SalesMapper.java

```
package SalesCountry;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements Mapper <LongWritable, Text,
Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable>
output, Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] SingleCountryData = valueString.split(",");
                output.collect(new Text(SingleCountryData[7]), one);
        }
}
```

SalesCountryReducer.java

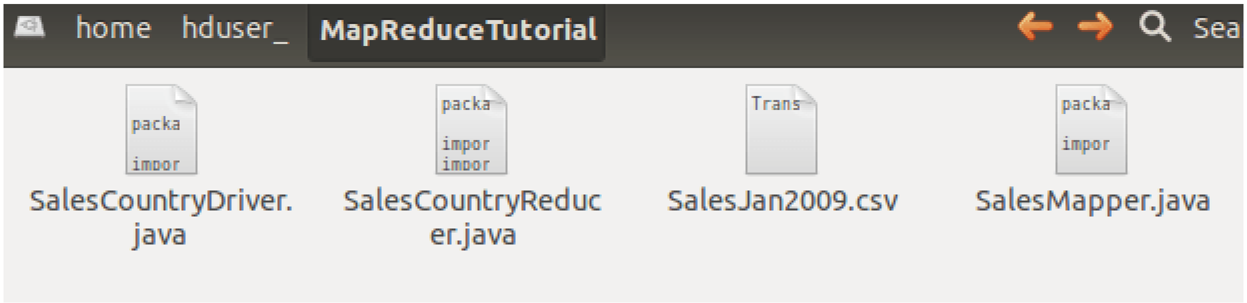
```
package SalesCountry;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
```

```
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

        public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
                Text key = t_key;
                int frequencyForCountry = 0;
                while (values.hasNext()) {
                        // replace type of value with the actual type of our value
                        IntWritable value = (IntWritable) values.next();
                        frequencyForCountry += value.get();

                }
                output.collect(key, new IntWritable(frequencyForCountry));
        }
}
```

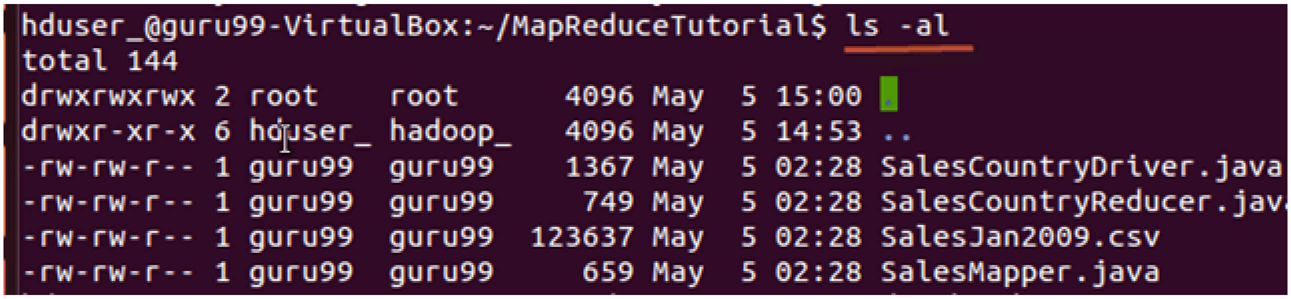
SalesCountryDriver.java

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SalesCountryDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        // Create a configuration object for the job
        JobConf job_conf = new JobConf(SalesCountryDriver.class);

        // Set a name of the Job
        job_conf.setJobName("SalePerCountry");

        // Specify data type of output key and value
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);

        // Specify names of Mapper and Reducer Class
        job_conf.setMapperClass(SalesCountry.SalesMapper.class);
        job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

        // Specify formats of the data type of Input and output
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);
        // Set input and output directories using command line arguments,
        //arg[0] = name of input directory on HDFS, and arg[1] =  name of output directory to be
created to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Download Files Here

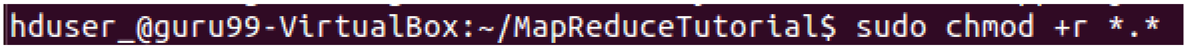


Check the file permissions of all these files



```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ ls -al
total 144
drwxrwxrwx 2 root    root      4096 May  5 15:00 .
drwxr-xr-x 6 hduser_ hadoop_   4096 May  5 14:53 ..
-rw-rw-r-- 1 guru99  guru99    1367 May  5 02:28 SalesCountryDriver.java
-rw-rw-r-- 1 guru99  guru99     749 May  5 02:28 SalesCountryReducer.jav
-rw-rw-r-- 1 guru99  guru99  123637 May  5 02:28 SalesJan2009.csv
-rw-rw-r-- 1 guru99  guru99     659 May  5 02:28 SalesMapper.java
```

and if ‘read’ permissions are missing then grant the same-

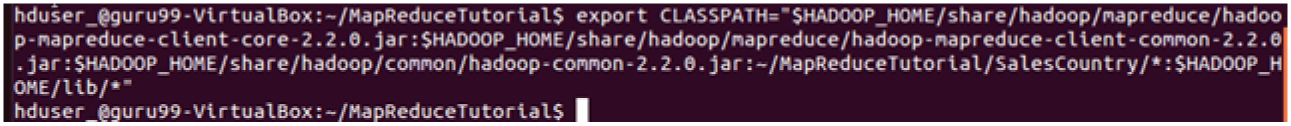


```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ sudo chmod +r *.*
```

Step 2)

Export classpath as shown in the below Hadoop example

```
export
CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_HOME/lib/*"
```

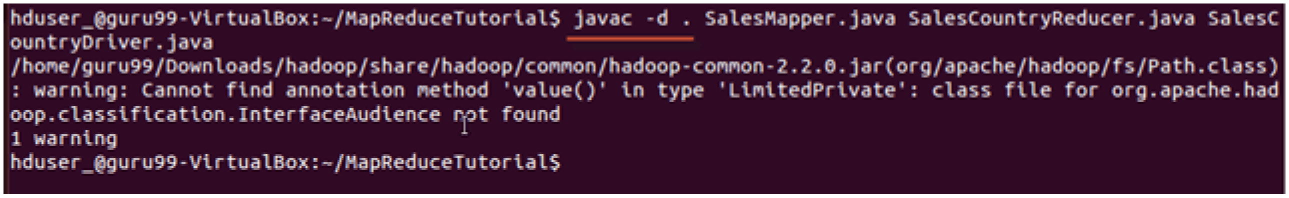


```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoo
p-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0
.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_H
OME/lib/*"
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

Step 3)

Compile Java files (these files are present in directory **Final-MapReduceHandsOn**). Its class files will be put in the package directory

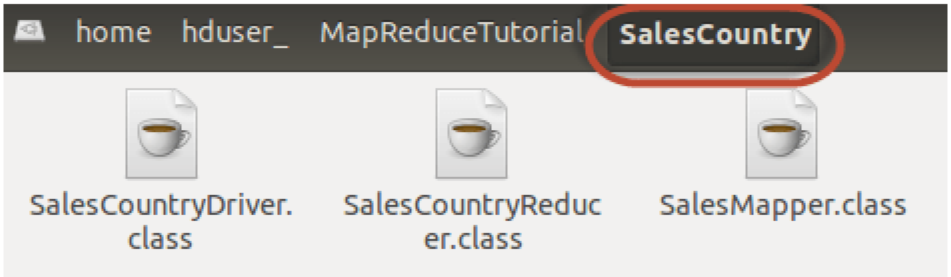
```
javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java
```



```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ javac -d . SalesMapper.java SalesCountryReducer.java SalesC
ountryDriver.java
/home/guru99/Downloads/hadoop/share/hadoop/common/hadoop-common-2.2.0.jar(org/apache/hadoop/fs/Path.class)
: warning: Cannot find annotation method 'value()' in type 'LimitedPrivate': class file for org.apache.had
oop.classification.InterfaceAudience not found
1 warning
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

This warning can be safely ignored.

This compilation will create a directory in a current directory named with package name specified in the java source file (i.e. **SalesCountry** in our case) and put all compiled class files in it.



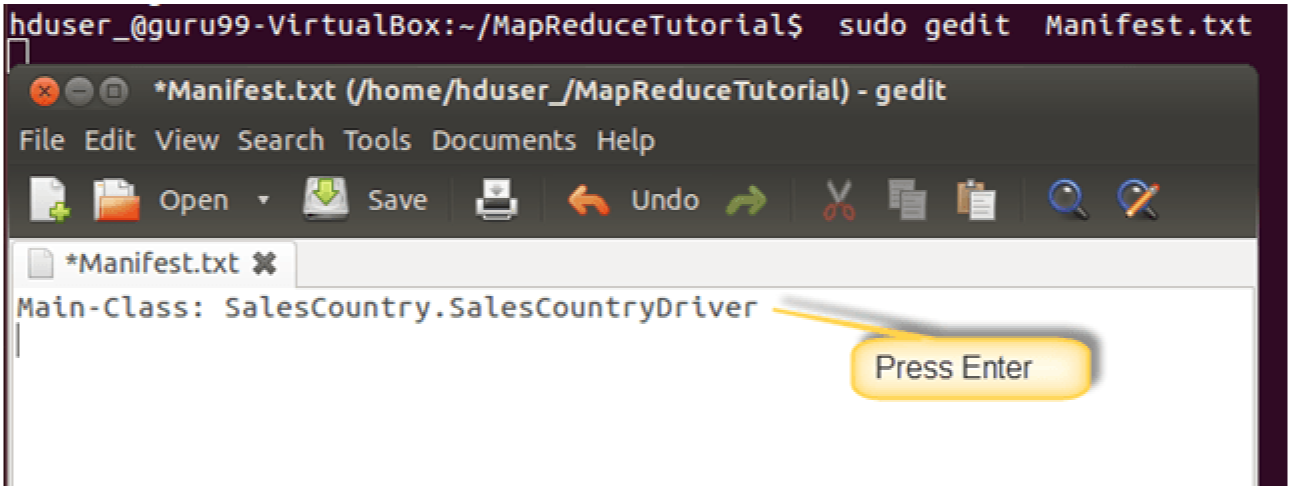
Step 4)

Create a new file **Manifest.txt**

```
sudo gedit Manifest.txt
```

add following lines to it,

```
Main-Class: SalesCountry.SalesCountryDriver
```



```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$  sudo gedit  Manifest.txt
```

*Manifest.txt (/home/hduser_/MapReduceTutorial) - gedit

```
Main-Class: SalesCountry.SalesCountryDriver
```

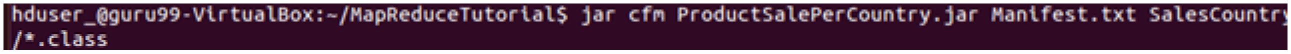
Press Enter

SalesCountry.SalesCountryDriver is the name of main class. Please note that you have to hit enter key at end of this line.

Step 5)

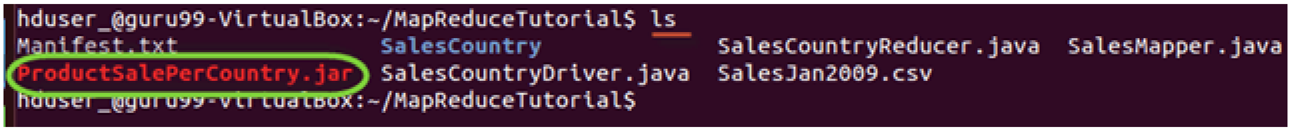
Create a Jar file

jar cfm ProductSalePerCountry.jar Manifest.txt SalesCountry/*.class



```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ jar cfm ProductSalePerCountry.jar Manifest.txt SalesCountry
/*.class
```

Check that the jar file is created



```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ ls
Manifest.txt               SalesCountry             SalesCountryReducer.java  SalesMapper.java
ProductSalePerCountry.jar  SalesCountryDriver.java  SalesJan2009.csv
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

Step 6)

Start Hadoop

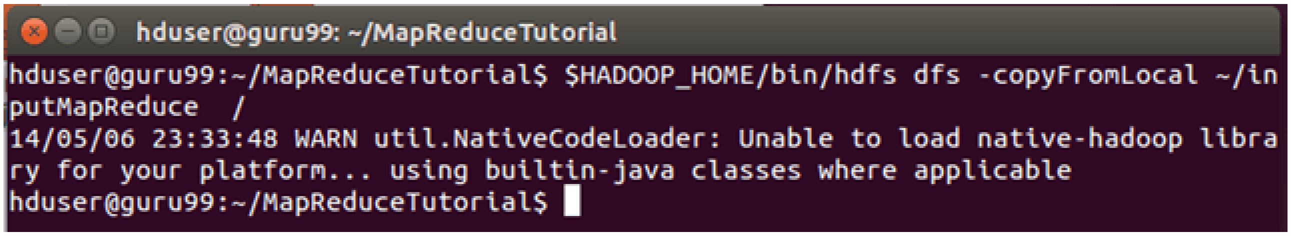
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh

Step 7)

Copy the File **SalesJan2009.csv** into **~/inputMapReduce**

Now Use below command to copy **~/inputMapReduce** to HDFS.

$HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/inputMapReduce /

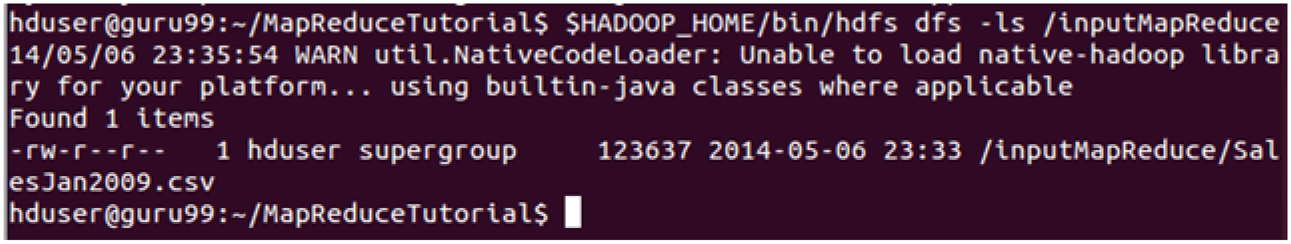


```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/in
putMapReduce  /
14/05/06 23:33:48 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hduser@guru99:~/MapReduceTutorial$ 
```

We can safely ignore this warning.

Verify whether a file is actually copied or not.

$HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce

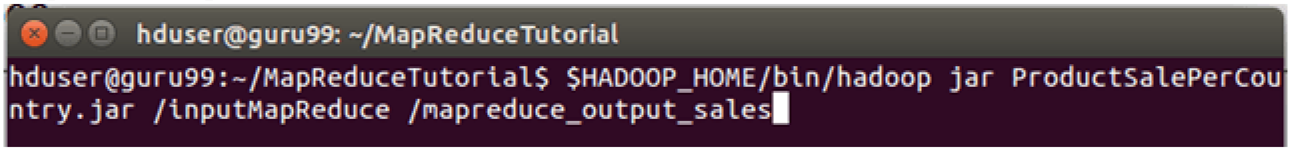


```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce
14/05/06 23:35:54 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 hduser supergroup     123637 2014-05-06 23:33 /inputMapReduce/Sal
esJan2009.csv
hduser@guru99:~/MapReduceTutorial$ 
```

Step 8)

Run MapReduce job

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales



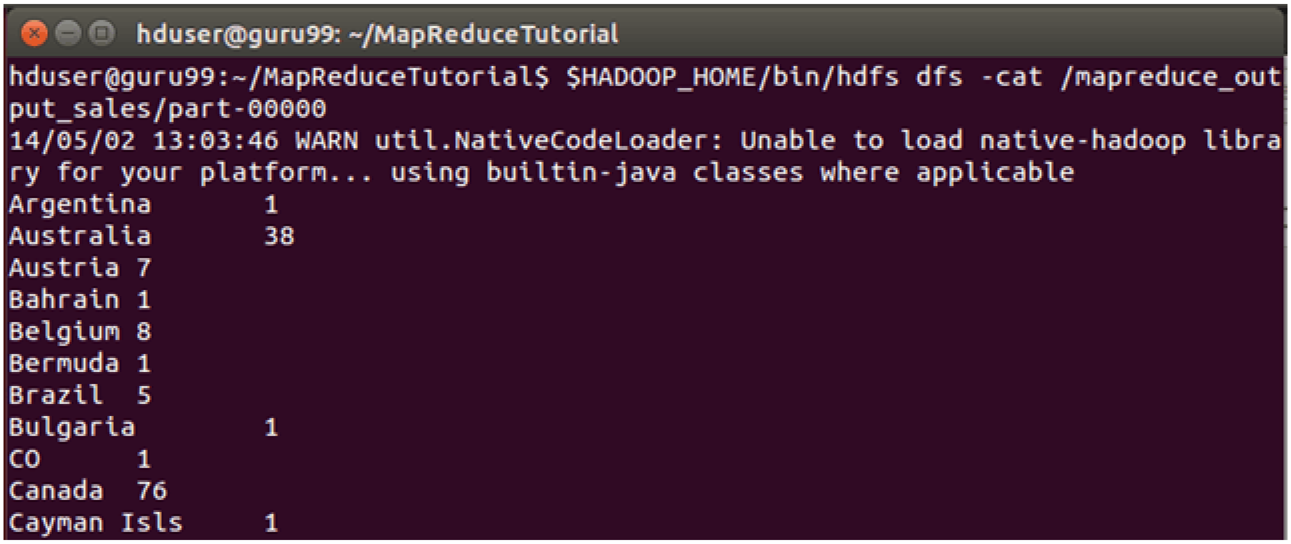
```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hadoop jar ProductSalePerCou
ntry.jar /inputMapReduce /mapreduce_output_sales
```

This will create an output directory named mapreduce_output_sales on HDFS. Contents of this directory will be a file containing product sales per country.

Step 9)

The result can be seen through command interface as,

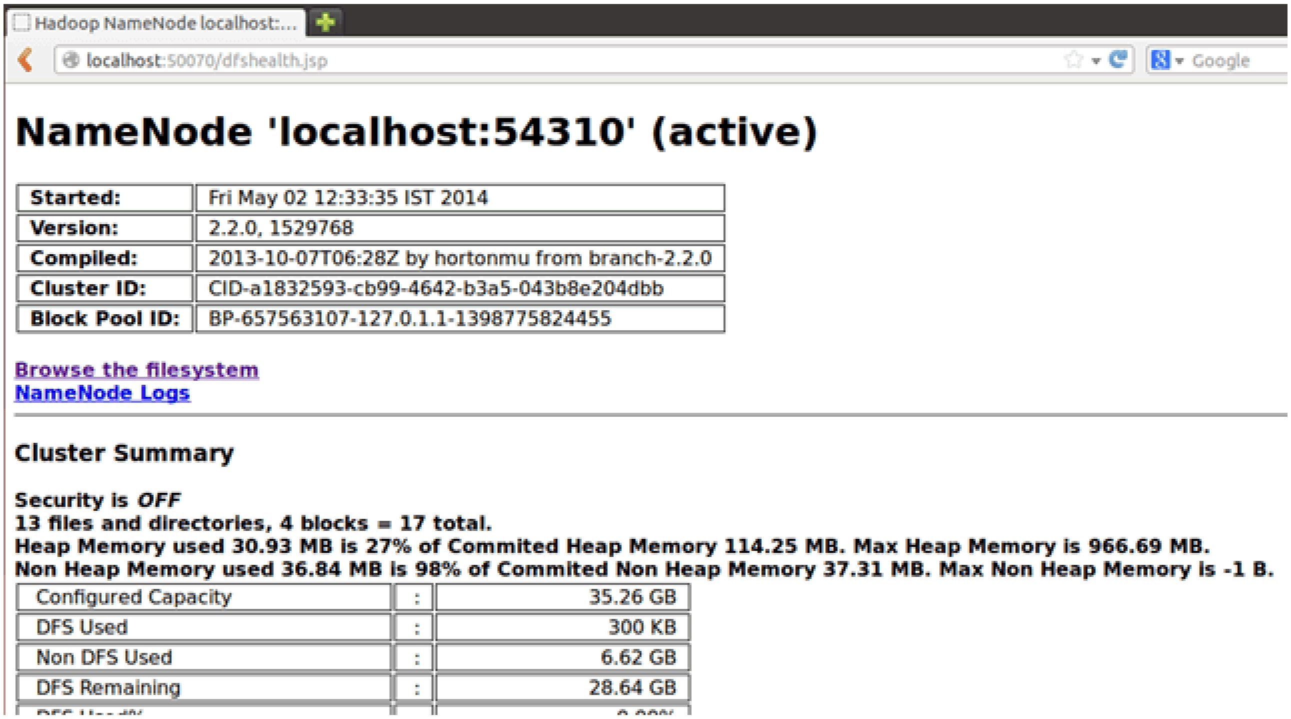
$HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_output_sales/part-00000



```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_out
put_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
```

Results can also be seen via a web interface as-

Open r in a web browser.



NameNode 'localhost:54310' (active)

Started:	Fri May 02 12:33:35 IST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-a1832593-cb99-4642-b3a5-043b8e204dbb
Block Pool ID:	BP-657563107-127.0.1.1-1398775824455

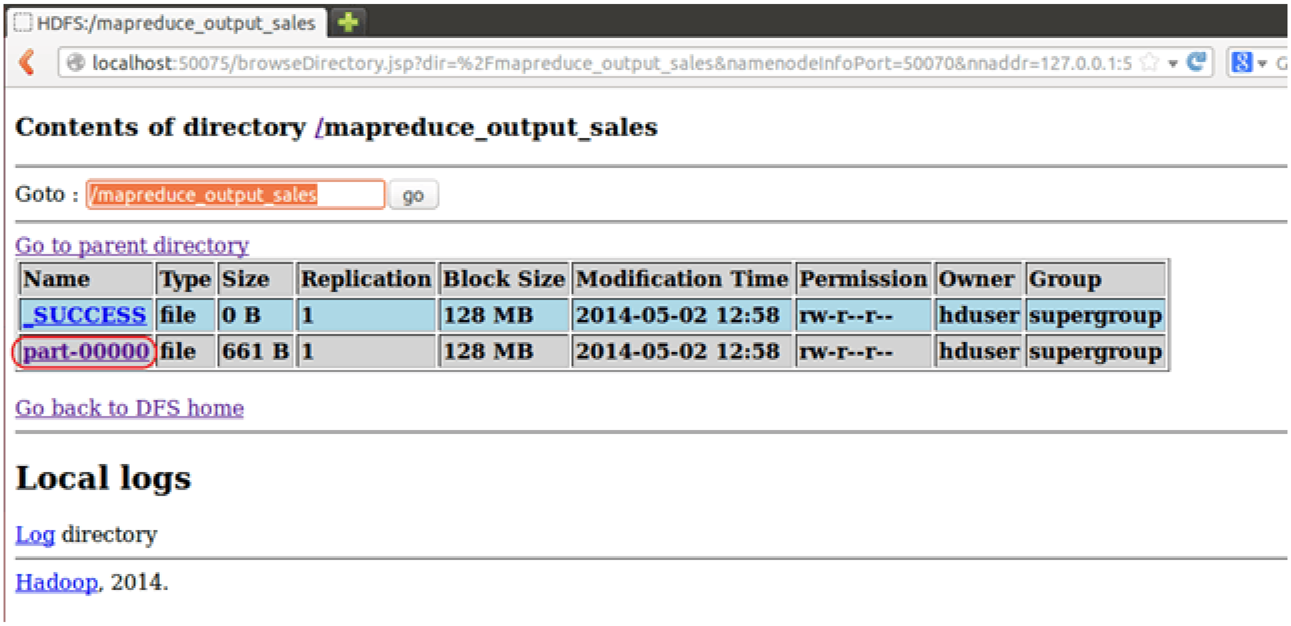
Browse the filesystem
NameNode Logs

Cluster Summary

Security is *OFF*
13 files and directories, 4 blocks = 17 total.
Heap Memory used 30.93 MB is 27% of Commited Heap Memory 114.25 MB. Max Heap Memory is 966.69 MB.
Non Heap Memory used 36.84 MB is 98% of Commited Non Heap Memory 37.31 MB. Max Non Heap Memory is -1 B.

Configured Capacity	:	35.26 GB
DFS Used	:	300 KB
Non DFS Used	:	6.62 GB
DFS Remaining	:	28.64 GB

Now select **'Browse the filesystem'** and navigate to **/mapreduce_output_sales**



HDFS:/mapreduce_output_sales
localhost:50075/browseDirectory.jsp?dir=%2Fmapreduce_output_sales&namenodeInfoPort=50070&nnaddr=127.0.0.1:5

Contents of directory /mapreduce_output_sales

Goto : /mapreduce_output_sales go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	1	128 MB	2014-05-02 12:58	rw-r--r--	hduser	supergroup
part-00000	file	661 B	1	128 MB	2014-05-02 12:58	rw-r--r--	hduser	supergroup

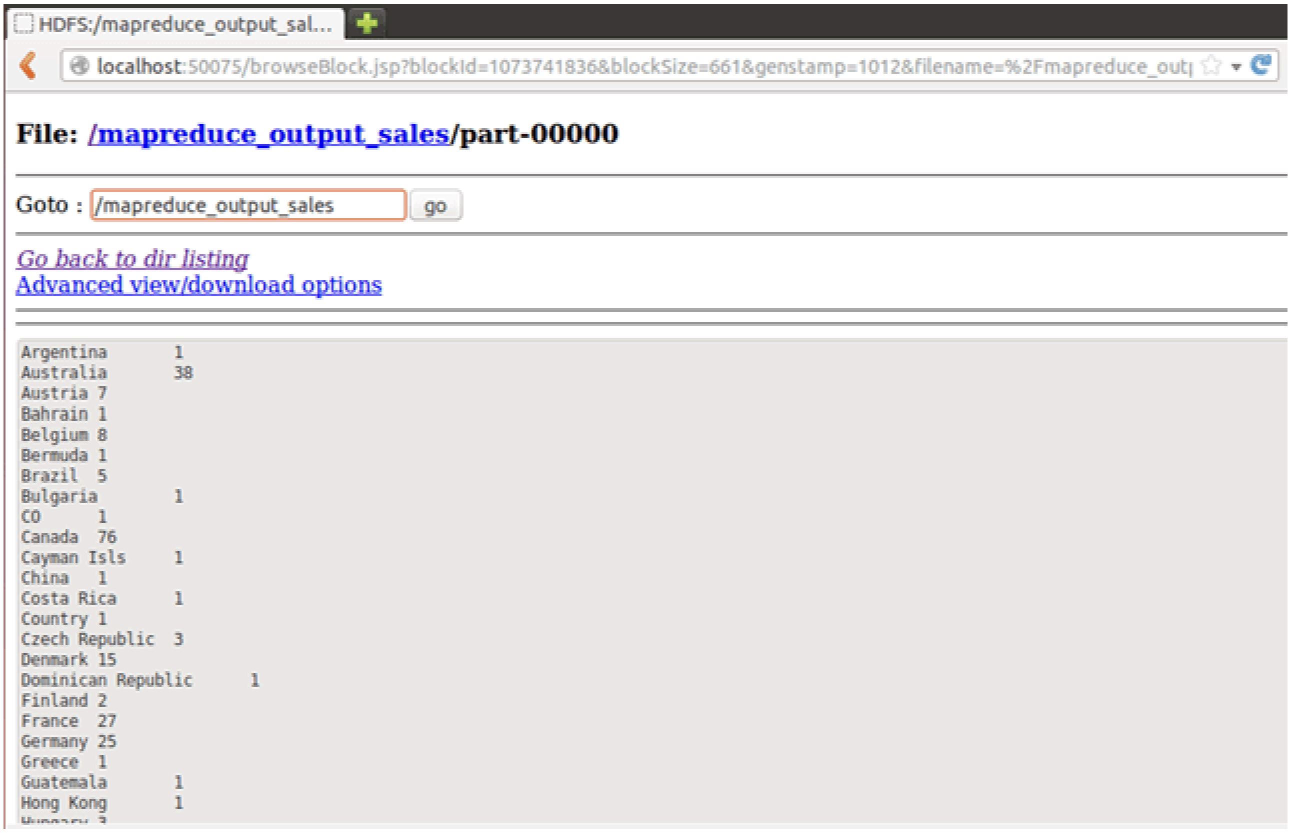
Go back to DFS home

Local logs

Log directory

Hadoop, 2014.

Open **part-r-00000**



HDFS:/mapreduce_output_sal...
localhost:50075/browseBlock.jsp?blockId=1073741836&blockSize=661&genstamp=1012&filename=%2Fmapreduce_out

File: /mapreduce_output_sales/part-00000

Goto : /mapreduce_output_sales go

Go back to dir listing
Advanced view/download options

```
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
China   1
Costa Rica      1
Country 1
Czech Republic  3
Denmark 15
Dominican Republic      1
Finland 2
France  27
Germany 25
Greece  1
Guatemala       1
Hong Kong       1
Hungary 3
```

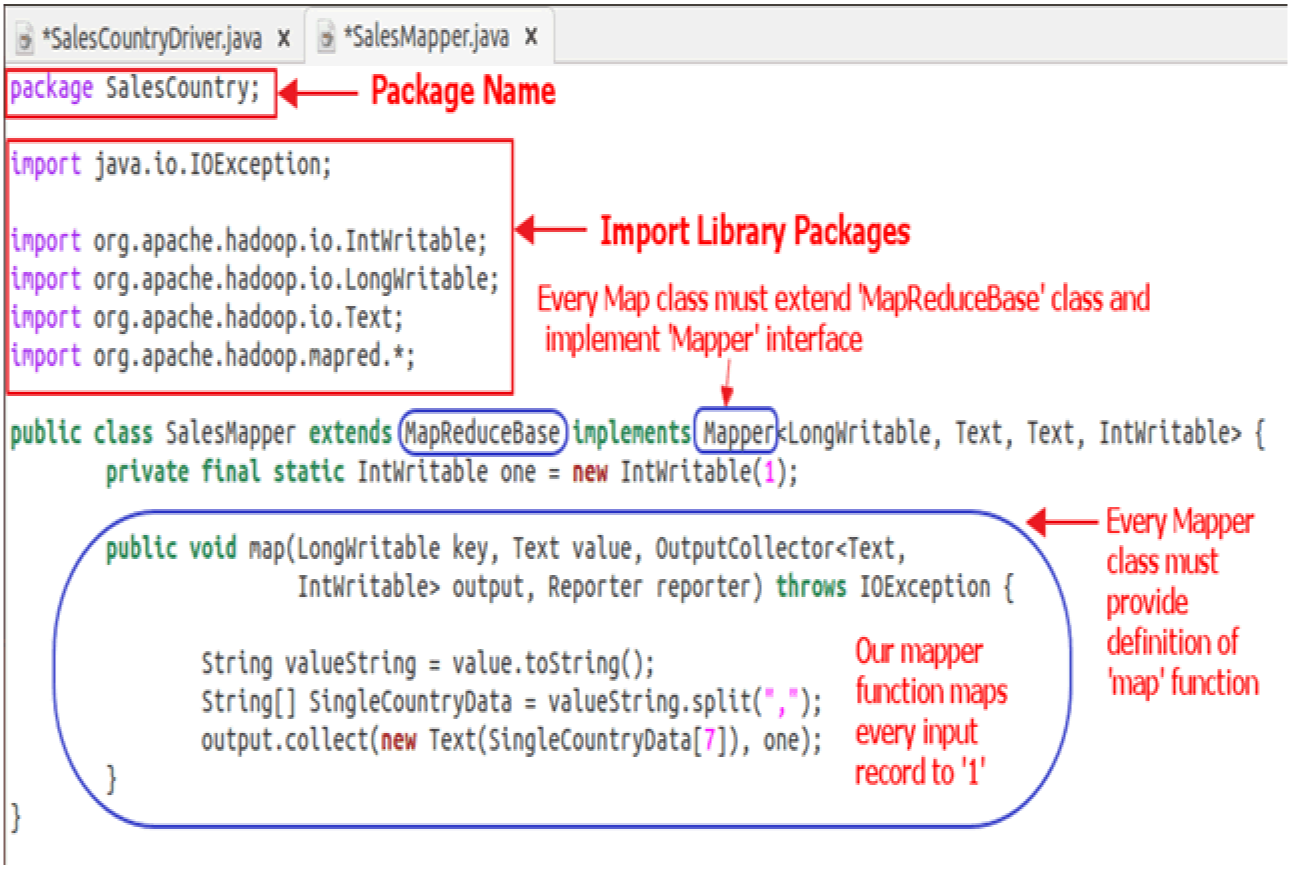
Explanation of SalesMapper Class

In this section, we will understand the implementation of **SalesMapper** class.

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of our package. Please note that output of compilation, **SalesMapper.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesMapper** class-


*SalesCountryDriver.java
*SalesMapper.java
Package Name
Import Library Packages
Every Map class must extend 'MapReduceBase' class and implement 'Mapper' interface
Every Mapper class must provide definition of 'map' function
Our mapper function maps every input record to '1'

```
package SalesCountry;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector<Text,
                        IntWritable> output, Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] SingleCountryData = valueString.split(",");
                output.collect(new Text(SingleCountryData[7]), one);
        }
}
```

Sample Code Explanation:

1. SalesMapper Class Definition-

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

Every mapper class must be extended from **MapReduceBase** class and it must implement **Mapper** interface.

2. Defining 'map' function-

public void map(LongWritable key,
        Text value,

```
OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException
```

The main part of Mapper class is a **‘map()’** method which accepts four arguments.

At every call to **‘map()’** method, a **key-value** pair (**‘key’** and **‘value’** in this code) is passed.

‘map()’ method begins by splitting input text which is received as an argument. It uses the tokenizer to split these lines into words.

```
String valueString = value.toString();
String[] SingleCountryData = valueString.split(",");
```

Here, **‘,’** is used as a delimiter.

After this, a pair is formed using a record at 7th index of array **‘SingleCountryData’** and a value **‘1’**.

```
output.collect(new Text(SingleCountryData[7]), one);
```

We are choosing record at 7th index because we need **Country** data and it is located at 7th index in array **‘SingleCountryData’**.

Please note that our input data is in the below format (where **Country** is at 7th index, with 0 as a starting index)-

Transaction_date,Product,Price,Payment_Type,Name,City,State,**Country**,Account_Created,Last_Login,Latitude,Longitude

An output of mapper is again a **key-value** pair which is outputted using **‘collect()’** method of **‘OutputCollector’**.

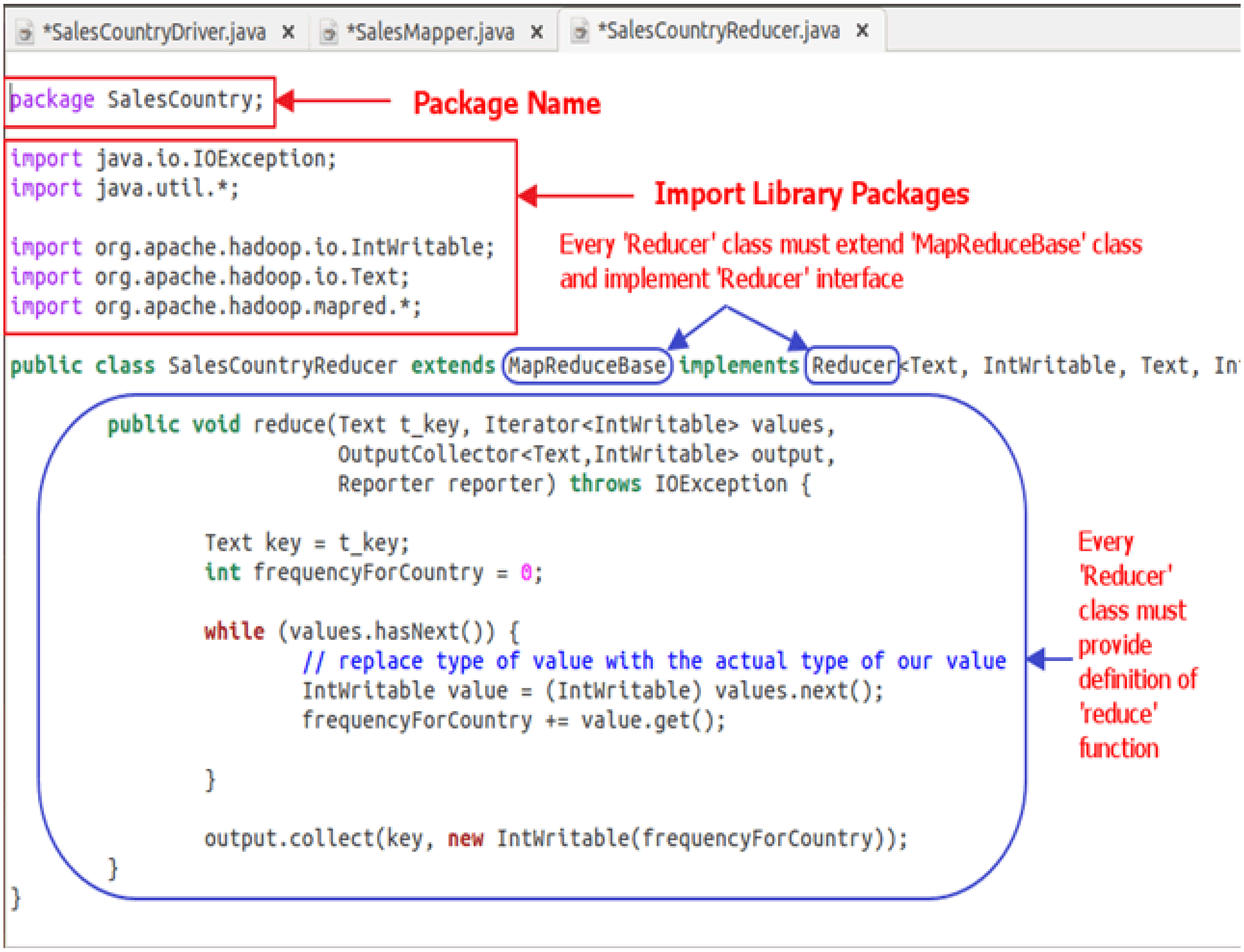
Explanation of SalesCountryReducer Class

In this section, we will understand the implementation of **SalesCountryReducer** class.

1. We begin by specifying a name of the package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryReducer.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesCountryReducer** class-



```
package SalesCountry;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, In
    public void reduce(Text t_key, Iterator<IntWritable> values,
                        OutputCollector<Text,IntWritable> output,
                        Reporter reporter) throws IOException {

            Text key = t_key;
            int frequencyForCountry = 0;

            while (values.hasNext()) {
                    // replace type of value with the actual type of our value
                    IntWritable value = (IntWritable) values.next();
                    frequencyForCountry += value.get();

            }

            output.collect(key, new IntWritable(frequencyForCountry));
    }
}
```

Every 'Reducer' class must extend 'MapReduceBase' class and implement 'Reducer' interface

Every 'Reducer' class must provide definition of 'reduce' function

Code Explanation:

1. SalesCountryReducer Class Definition-

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

Here, the first two data types, **'Text'** and **'IntWritable'** are data type of input key-value to the reducer.

Output of mapper is in the form of <CountryName1, 1>, <CountryName2, 1>. This output of mapper becomes input to the reducer. So, to align with its data type, **Text** and **IntWritable** are used as data type here.

The last two data types, 'Text' and 'IntWritable' are data type of output generated by reducer in the form of key-value pair.

Every reducer class must be extended from **MapReduceBase** class and it must implement **Reducer** interface.

2. Defining 'reduce' function-

```
public void reduce( Text t_key,
            Iterator<IntWritable> values,
            OutputCollector<Text,IntWritable> output,
            Reporter reporter) throws IOException {
```

An input to the **reduce()** method is a key with a list of multiple values.

For example, in our case, it will be-

<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>,<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>.

This is given to reducer as **<United Arab Emirates, {1,1,1,1,1,1}>**

So, to accept arguments of this form, first two data types are used, viz., **Text** and **Iterator<IntWritable>**. **Text** is a data type of key and **Iterator<IntWritable>** is a data type for list of values for that key.

The next argument is of type **OutputCollector<Text,IntWritable>** which collects the output of reducer phase.

reduce() method begins by copying key value and initializing frequency count to 0.

```
Text key = t_key;
int frequencyForCountry = 0;
```

Then, using '**while**' loop, we iterate through the list of values associated with the key and calculate the final frequency by summing up all the values.

```
 while (values.hasNext()) {
           // replace type of value with the actual type of our value
           IntWritable value = (IntWritable) values.next();
           frequencyForCountry += value.get();

       }
```

Now, we push the result to the output collector in the form of **key** and obtained **frequency count**.

Below code does this-

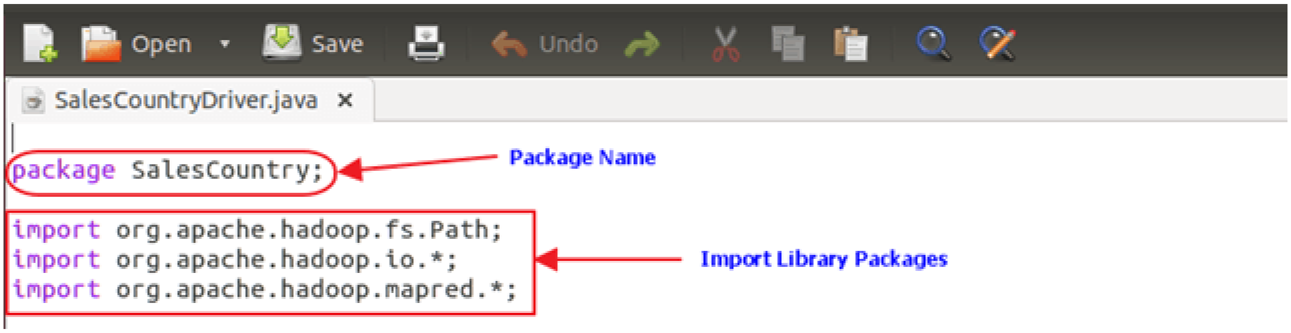
```
output.collect(key, new IntWritable(frequencyForCountry));
```

Explanation of SalesCountryDriver Class

In this section, we will understand the implementation of **SalesCountryDriver** class

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryDriver.class** will go into directory named by this package name: **SalesCountry**.

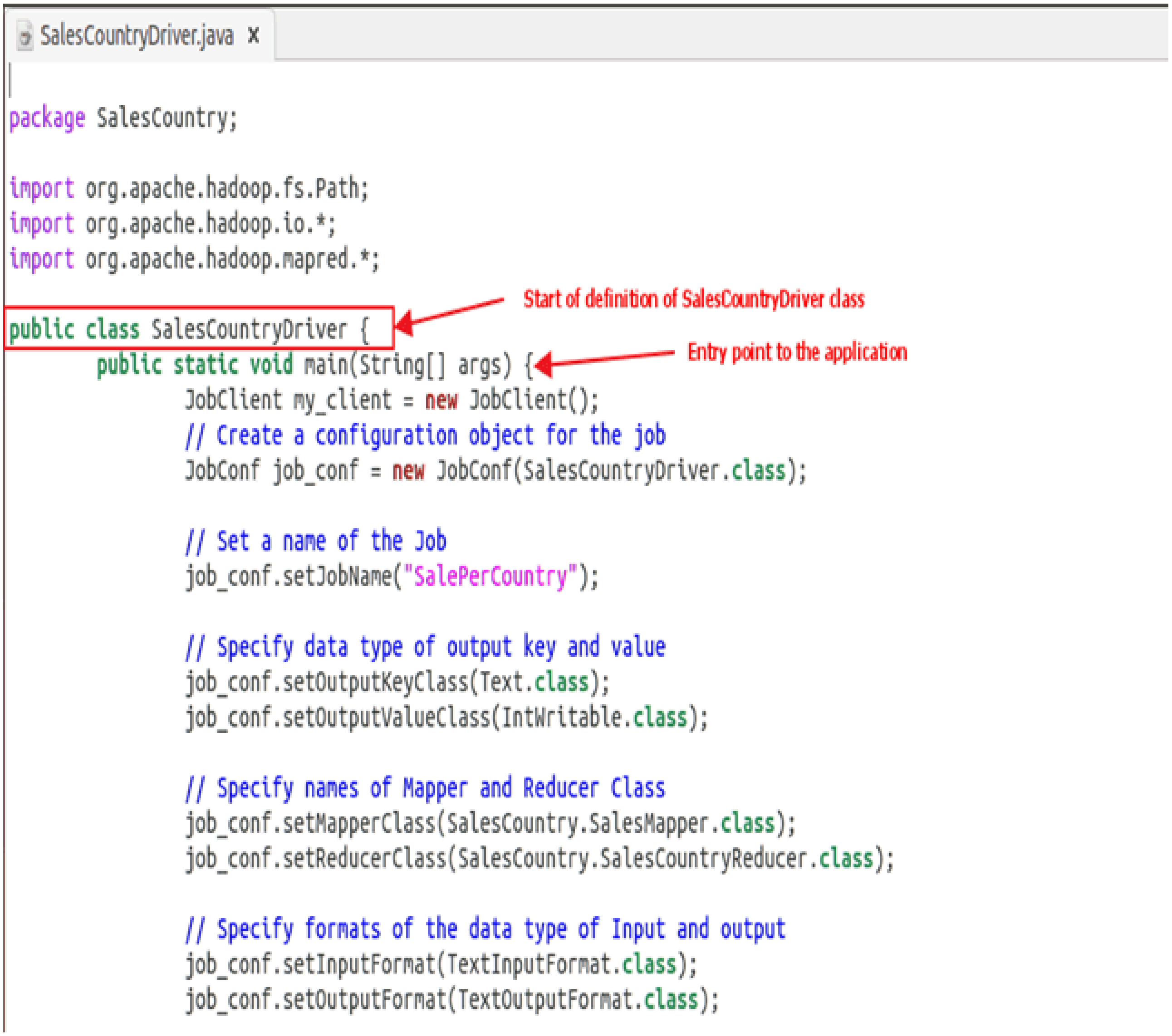
Here is a line specifying package name followed by code to import library packages.



```
package SalesCountry;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
```

2. Define a driver class which will create a new client job, configuration object and advertise Mapper and Reducer classes.

The driver class is responsible for setting our MapReduce job to run in Hadoop. In this class, we specify **job name, data type of input/output and names of mapper and reducer classes**.



SalesCountryDriver.java

```java
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SalesCountryDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        // Create a configuration object for the job
        JobConf job_conf = new JobConf(SalesCountryDriver.class);

        // Set a name of the Job
        job_conf.setJobName("SalePerCountry");

        // Specify data type of output key and value
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);

        // Specify names of Mapper and Reducer Class
        job_conf.setMapperClass(SalesCountry.SalesMapper.class);
        job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

        // Specify formats of the data type of Input and output
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);
```

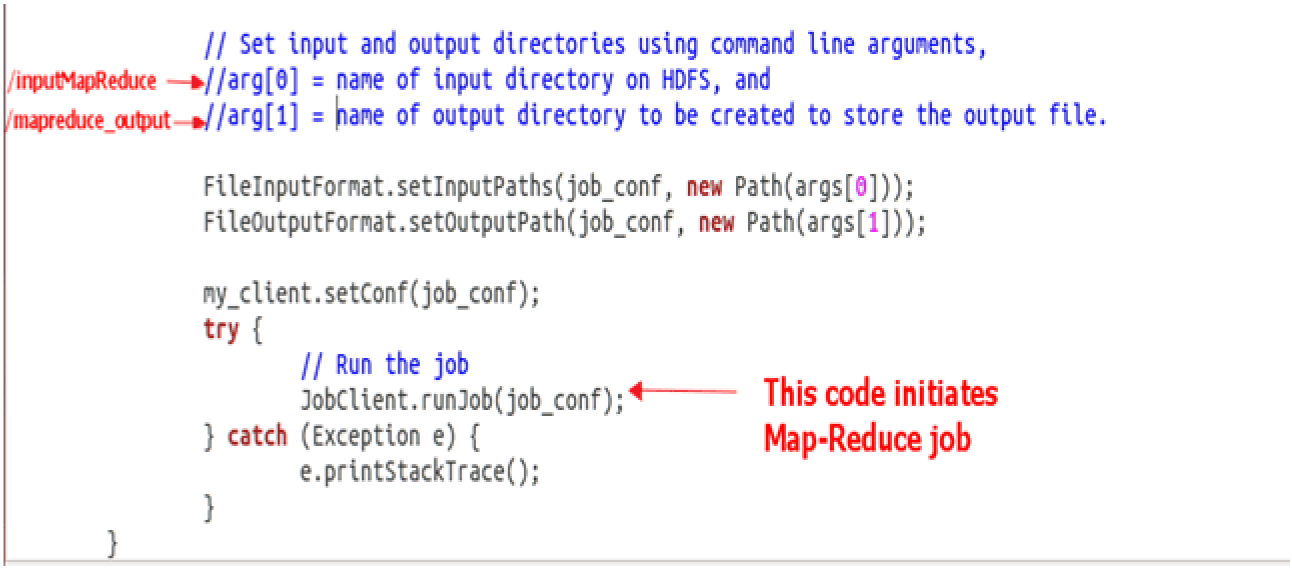
Start of definition of SalesCountryDriver class

Entry point to the application

3. In below code snippet, we set input and output directories which are used to consume input dataset and produce output, respectively.

arg[0] and **arg[1]** are the command-line arguments passed with a command given in MapReduce hands-on, i.e.,

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales



```
        // Set input and output directories using command line arguments,
        //arg[0] = name of input directory on HDFS, and
        //arg[1] = name of output directory to be created to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

/inputMapReduce

/mapreduce_output

This code initiates Map-Reduce job

4. Trigger our job

Below code start execution of MapReduce job-

```
try {
    // Run the job
    JobClient.runJob(job_conf);
} catch (Exception e) {
    e.printStackTrace();
}
```

Conclusion: Hence we have thourouly studied how to design a distributed application using Map Reduce which processes a log file of a system.