Hadoop & Mapreduce Examples: Create your First

Problem Statement:

Find out Number of Products Sold in Each Cou

SalesJan2009.csv

Prerequisites:

This tutorial is developed on **Linux - Ubuntu** c
You should have **Hadoop** (**version 2.2.0** used
already installed.
You should have **Java** (**version 1.8.0** used for t
installed on the system.

Before we start with the actual process, change user to 'hduser' (user used for Hadoop).

su - hduser_

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

Transaction date	Product	Price	Payment Type	Name	City	State	Country	Account Created	Last in
01-02-2009 6:17	Product1	1200	Mastercard	carolina	Basildon	England	United Kingdom	01-02-2009 6:00	01-02 09 6:
01-02-2009 4:53	Product1	1200	Visa	Betina	Parkville	MO	United States	01-02-2009 4:42	01-02 09 7:
01-02-2009 13:08	Product1	1200	Mastercard	Federica e Andrea	Astoria	OR	United States	01-01-2009 16:21	01-03 09 12

```
01-03-20  Pro   1   Visa    Gouya      Echuca           Vic   Austral  9/25/05   01-03
09 14:44  duc   2                                       tor   ia        21:13    09 14
          t1    0                                       ia
                0
01-04-20  Pro   3   Visa    Gerd W     Cahaba Heights   AL    United   11/15/08  01-04
09 12:56  duc   6                                              States   15:47    09 12
          t2    0
                0
01-04-20  Pro   1   Visa    LAURENCE   Mickleton        NJ    United   9/24/08   01-04
09 13:19  duc   2                                              States   15:19    09 13
          t1    0
                0
```

Steps: 1

Create a new directory with name **MapReduceTutorial**

sudo mkdir MapReduceTutorial

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

Give permissions

sudo chmod -R 777 MapReduceTutorial

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

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, Te
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable
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, IntWrita
able> {

        public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Tex
put, 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));
        }
}
```

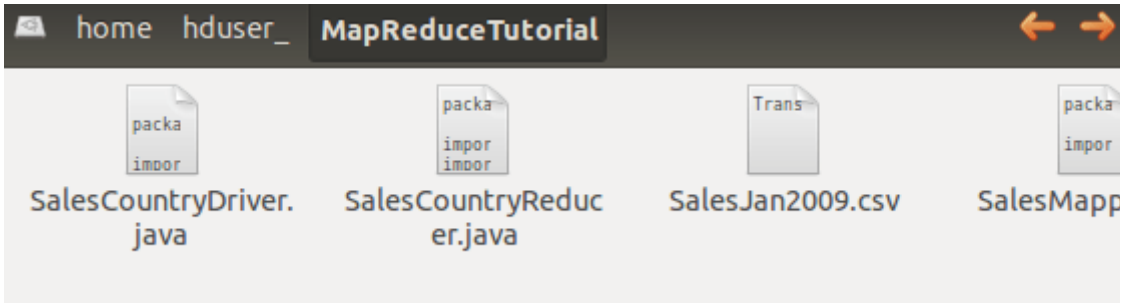
SalesCountryDriver.java

```
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);
```

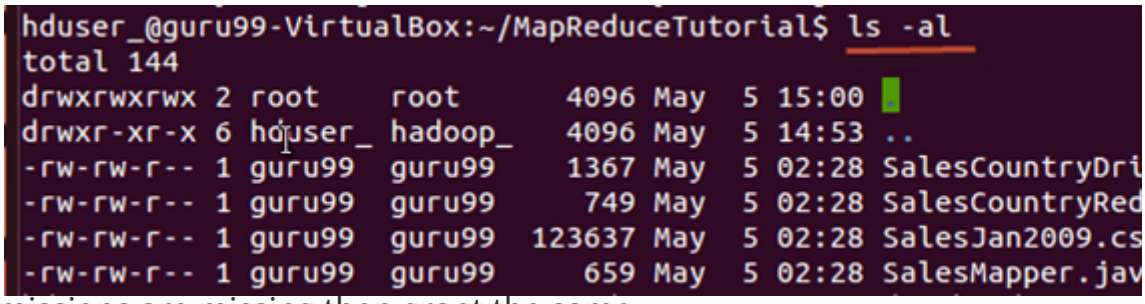
```
                job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

                // 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 outpu
reated 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

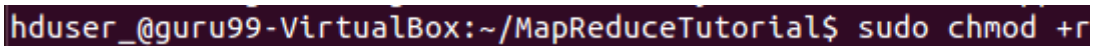
If you want to understand the code in these files refer this Guide



Check the file permissions of all these files



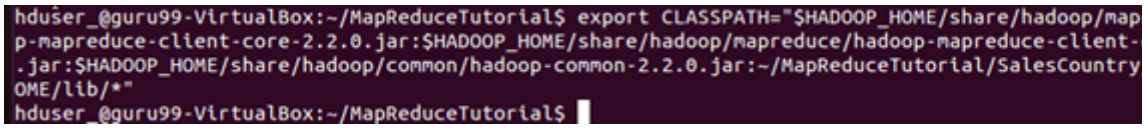
and if 'read' permissions are missing then grant the same-



Steps: 2

Export classpath

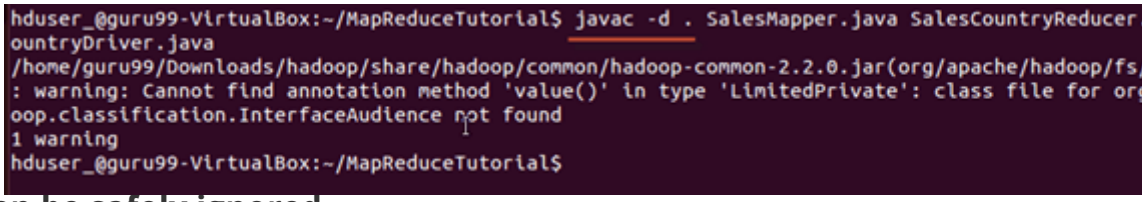
```
export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0
share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0.jar:$HADOOP_HOME/share/hadoo
mmon-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_HOME/lib/*"
```



Steps: 3

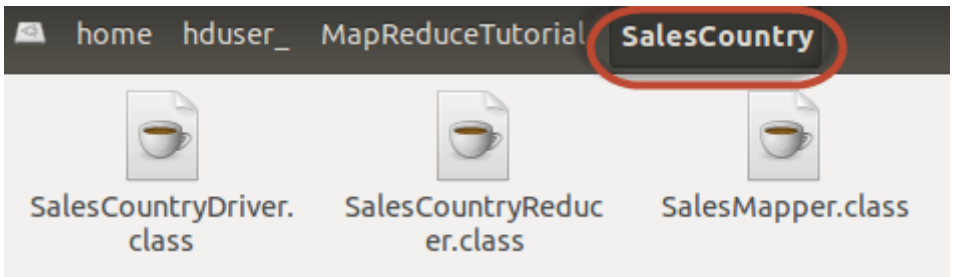
Compile Java files (these files are present in directory **Final-MapReduceHandsOn**). Its clas
the package directory

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



This warning can be safely ignored.

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



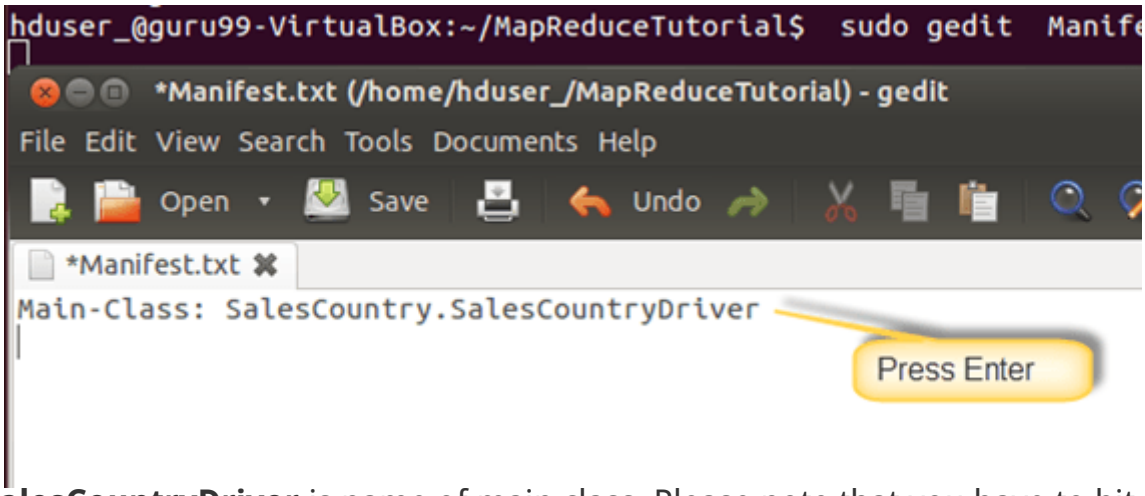
Steps: 4

Create a new file **Manifest.txt**

sudo gedit Manifest.txt

add following lines to it,

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

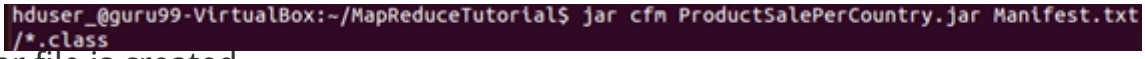


SalesCountry.SalesCountryDriver is name of main class. Please note that you have to hit this line.

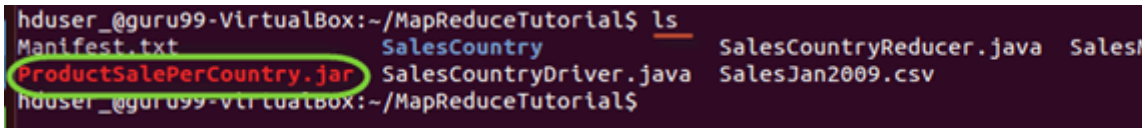
Steps: 5

Create a Jar file

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



Check that the jar file is created



Steps: 6

Start Hadoop

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

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

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

We can safely ignore this warning.

Verify whether file is actually copied or not.

```
$HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce
```

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

Steps: 8

Run MapReduce job

```
$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_
```

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

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

Steps: 9

Result can be seen through command interface as,

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

```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -cat /mapr
put_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-had
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
```

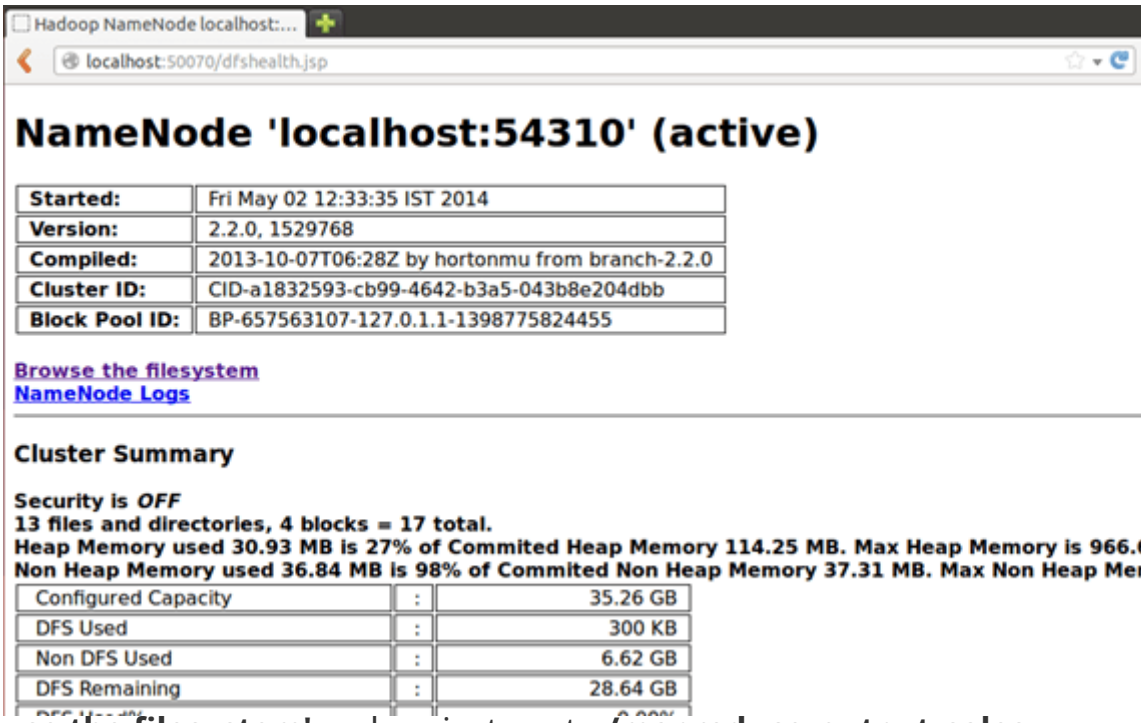
o/p of above

OR

Results can also be seen via web interface as-

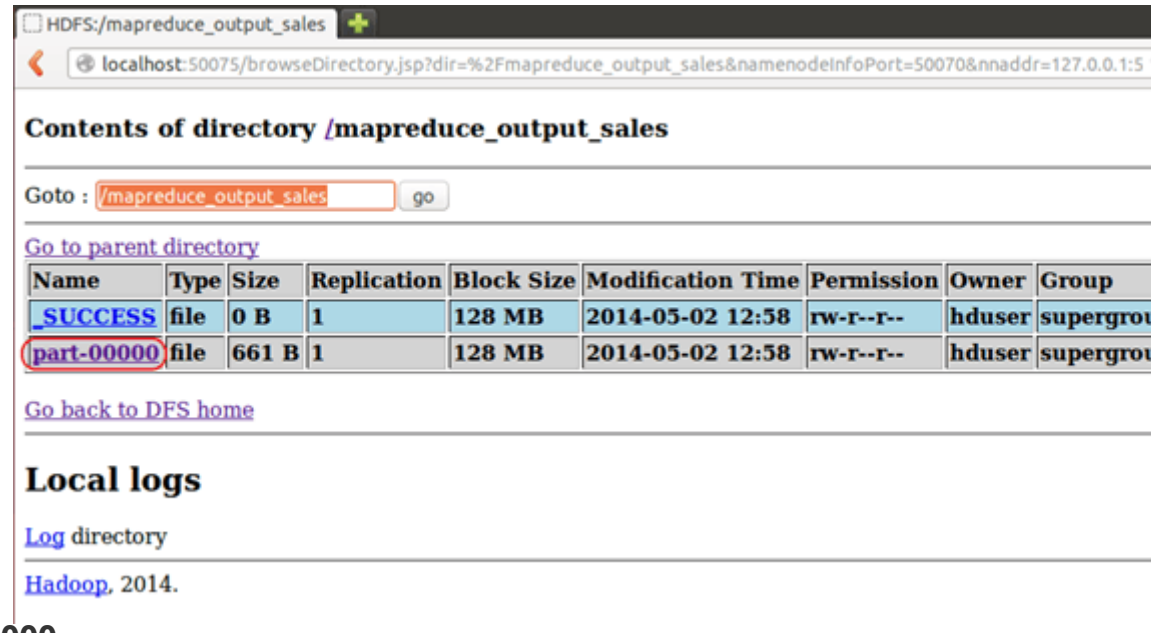
Results through web interface-

Open r in web browser.



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

o/p of above



Open **part-r-00000**

Understanding MapReducer Code

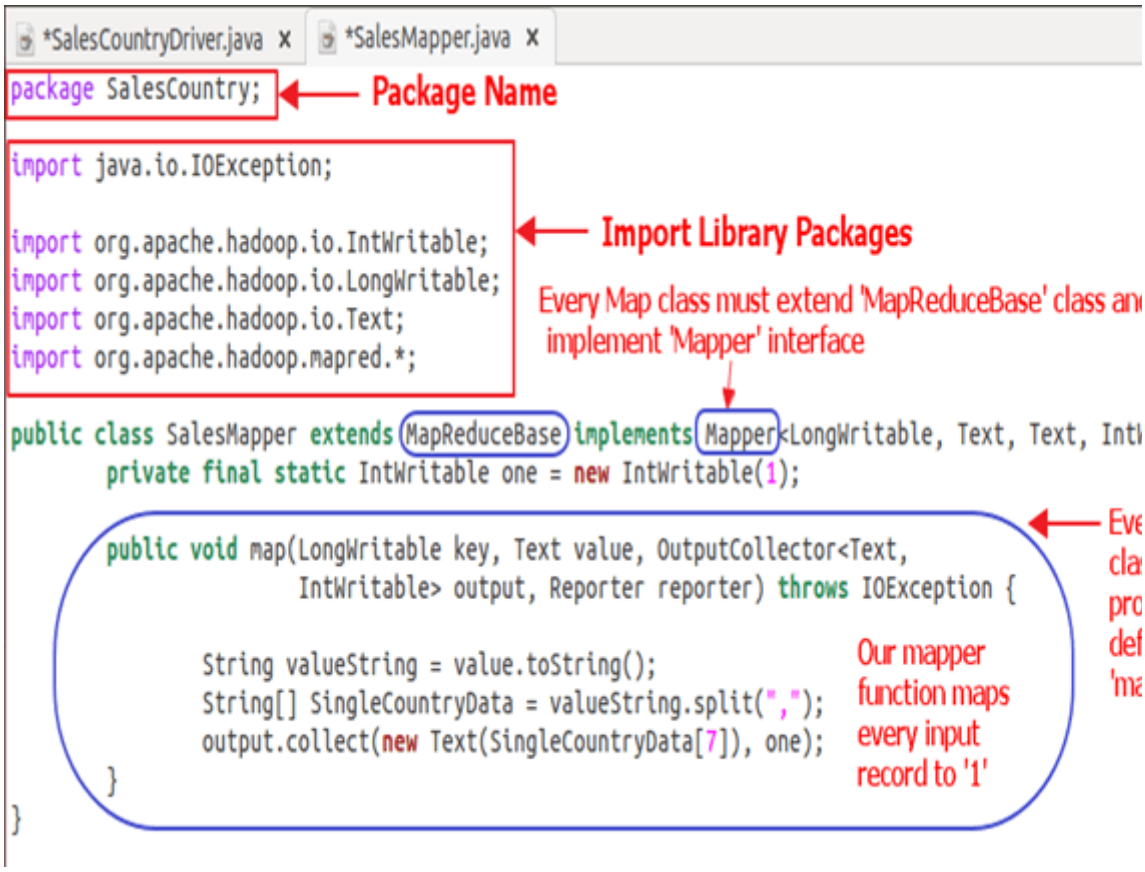
Explanation of SalesMapper Class

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

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

Followed by this, we import library packages.

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



Code Explanation:

1. SalesMapper Class Definition-

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text

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

2. Defining 'map' function-

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

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 toke 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

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 array **'SingleCountryData'**.

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

Transaction_date,Product,Price,Payment_Type,Name,City,State,**Country**,Account_Created,L

Output of mapper is again a **key-value** pair which is outputted using **'collect()'** method of

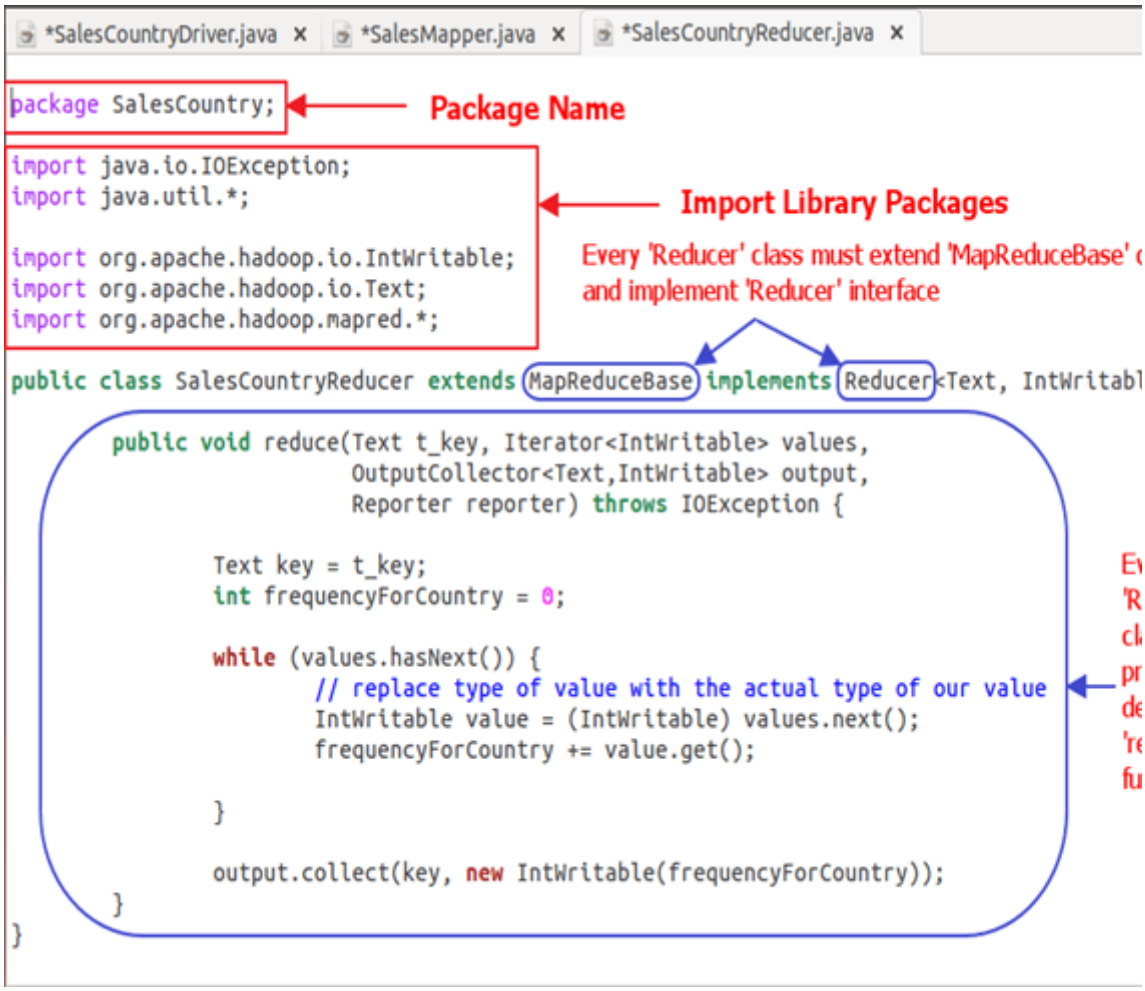
Explanation of SalesCountryReducer Class

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

1. We begin by specifying name of package for our class. **SalesCountry** is name of out pack that output of compilation, **SalesCountryReducer.class** will go into directory named by th name: **SalesCountry**.

Followed by this, we import library packages.

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



Code Explanation:

1. SalesCountryReducer Class Definition-

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

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

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

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

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

2. Defining 'reduce' function-

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

Input to the **reduce()** method is a key with 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 A
<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**
list of values for that key.

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

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 ca
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**

Below code does this-

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

Explanation of SalesCountryDriver Class

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

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

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

```
SalesCountryDriver.java ×

package SalesCountry;

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

Package Name

Import Library Packages

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

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

```
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);
```

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 ir produce output, respectively.

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

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

```
                // Set input and output directories using command line arguments,
/inputMapReduce ->//arg[0] = name of input directory on HDFS, and
/mapreduce_output->//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);  <-- This code initiates Map-Reduce job
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

4. Trigger our job

Below code start execution of MapReduce job-

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

Prev

4 Comments **Guru99**

Recommend 3 **Share**

Join the discussion…

souradeep misra • 4 months ago

thanks for this tutorial.this program run successfully.can u give me some other program? it is really helpful to me.

Reply • Share ›

FredPret • 8 months ago

Step 3: I updated the classpath to reflect my version of the jars (2.7.2). I also downloaded ha since it wasn't in the path from step 2.

Here are my error messages:

SalesMapper.java:5: error: package org.apache.hadoop.io does not exist
import org.apache.hadoop.io.IntWritable;
^
SalesMapper.java:6: error: package org.apache.hadoop.io does not exist
import org.apache.hadoop.io.LongWritable;
^
SalesMapper.java:7: error: package org.apache.hadoop.io does not exist
import org.apache.hadoop.io.Text;
^
SalesMapper.java:8: error: package org.apache.hadoop.mapred does not exist
import org.apache.hadoop.mapred.*;
^
SalesMapper.java:10: error: cannot find symbol
public class SalesMapper extends MapReduceBase implements Mapper...

etc etc.

How do I fix this?

Reply • Share ›

Gopalvq • 9 months ago

In step 7, where is ~/inputMapReduce. I am getting no such file or directory error. Help woul

Reply • Share ›

Gopalvq → Gopalvq • 9 months ago

I created one, there is no issue now. Thanks.

Reply • Share ›

Subscribe **Add Disqus to your site Add Disqus Add** **Privacy**