

Hadoop & Mapreduce Examples: Create your First



Artificial Intelligence

Hands-on Training and Workshops for
Students, Data Scientists and Developers.

Intel® Student Ambassador



Problem Statement:

Find out Number of Products Sold in Each Country

[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 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:~$
```

Transact ion date	Pro duc t	P r i c e	Payme nt Ty pe	Name	City	Sta te	Country	Account Created	Last in
01-02-20 09 6:17	Pro duc t1	1 2 0	Maste rcard	carolina	Basildon	Eng lan d	United Kingdo m	01-02-20 09 6:00	01-02 09 6:
01-02-20 09 4:53	Pro duc t1	1 2 0	Visa	Betina	Parkville	MO	United States	01-02-20 09 4:42	01-02 09 7:
01-02-20 09 13:08	Pro duc t1	1 2 0	Maste rcard	Federica e Andre a	Astoria	OR	United States	01-01-20 09 16:21	01-03 09 12

01-03-2009 14:44	Pro duc t1	1 2 0 0	Visa	Gouya	Echuca	Vic tor ia	Austral ia	9/25/05 21:13	01-03 09 14
01-04-2009 12:56	Pro duc t2	3 6 0 0	Visa	Gerd W	Cahaba Heights	AL	United States	11/15/08 15:47	01-04 09 12
01-04-2009 13:19	Pro duc t1	1 2 0 0	Visa	LAURENCE	Mickleton	NJ	United States	9/24/08 15:19	01-04 09 13

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

SalesMapper.java

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

```
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> {
    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, IntWritable> {

    public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Text> 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
reated to store the output file.

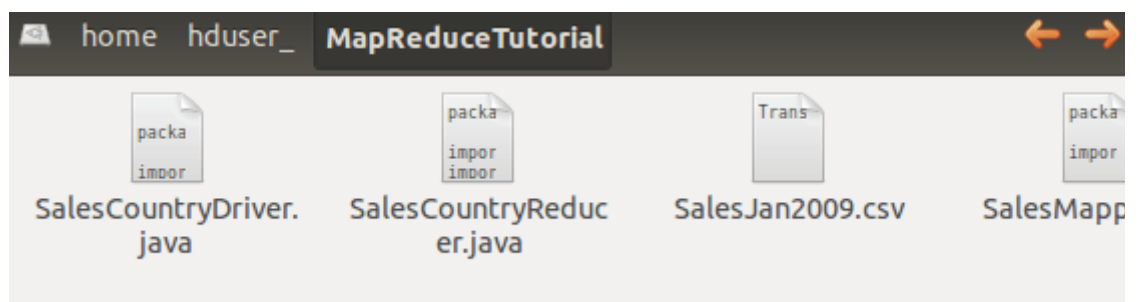
FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

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

```

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 SalesCountryDri
-rw-rw-r-- 1 guru99  guru99     749 May  5 02:28 SalesCountryRed
-rw-rw-r-- 1 guru99  guru99  123637 May  5 02:28 SalesJan2009.cs
-rw-rw-r-- 1 guru99  guru99     659 May  5 02:28 SalesMapper.jav

```

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

```

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

```

HPE Flexible Capacity.

The speed and agility of the public cloud and the control of on-premise IT are no longer apples and oranges.

Accelerating next



Hewlett Packard
Enterprise

Steps: 2

Export classpath

```
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="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:$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$
```

Steps: 3

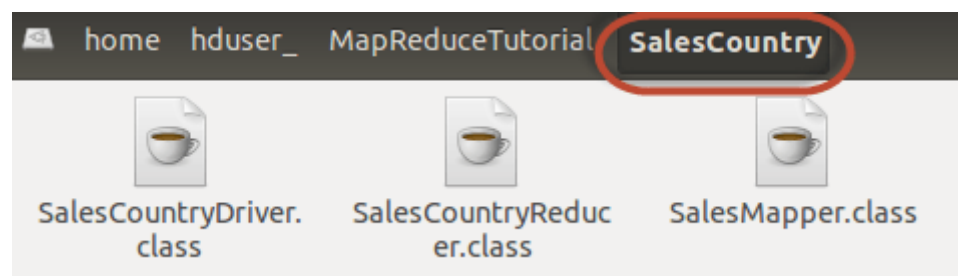
Compile **Java** files (these files are present in directory **Final-MapReduceHandsOn**). Its class files will be put in the package directory

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

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java
SalesCountryDriver.java:10: warning: Cannot find annotation method 'value()' in type 'LimitedPrivate': class file for org.apache.hadoop.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 **sp** (i.e. **SalesCountry** in our case) and put all compiled class files in it.



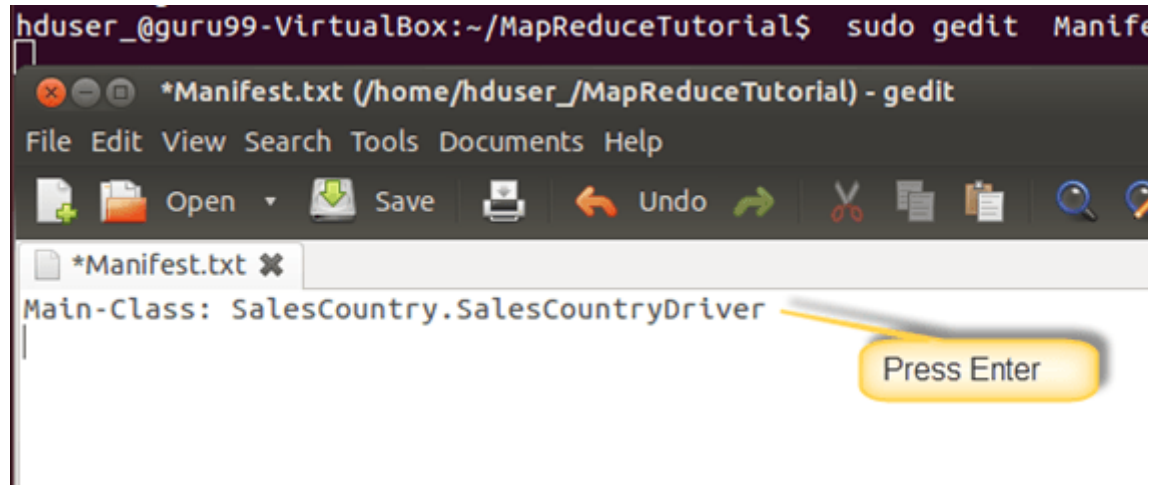
Steps: 4

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
```

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

Check that the jar file is created

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

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.

HPE Flexible Capacity.

The speed and agility of the public cloud and the control of on-premise IT are no longer apples and oranges.

Accelerating next



Hewlett Packard
Enterprise

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

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

We can safely ignore this warning.

Verify whether file is actually copied or not.

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

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

Steps: 8

Run MapReduce job

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

```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.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.

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 /mapreduce_output_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library 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.

The screenshot shows the Hadoop NameNode web interface for 'localhost:54310' (active). The browser address bar shows 'localhost:50070/dfshealth.jsp'. The page title is 'NameNode 'localhost:54310' (active)'. Below the title is a table with the following information:

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

Below the table are links for [Browse the filesystem](#) and [NameNode Logs](#). The 'Cluster Summary' section indicates that security is OFF and provides details about the cluster's state:

13 files and directories, 4 blocks = 17 total.
 Heap Memory used 30.93 MB is 27% of Committed Heap Memory 114.25 MB. Max Heap Memory is 966.1 MB.
 Non Heap Memory used 36.84 MB is 98% of Committed Non Heap Memory 37.31 MB. Max Non Heap Memory is 37.31 MB.

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 upto /mapreduce_output_sales

o/p of above

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 : 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	supergrou
part-00000	file	661 B	1	128 MB	2014-05-02 12:58	rw-r--r--	hduser	supergrou

[Go back to DFS home](#)

Local logs

[Log directory](#)

[Hadoop](#), 2014.

Open **part-r-00000**

Why choose?

With HPE Flexible Capacity,
you can have it all.



Hewlett Packard
Enterprise

HDFS:/mapreduce_output_sal...

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

File: /mapreduce_output_sales/part-00000

Goto : 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

```

Understanding MapReducer Code

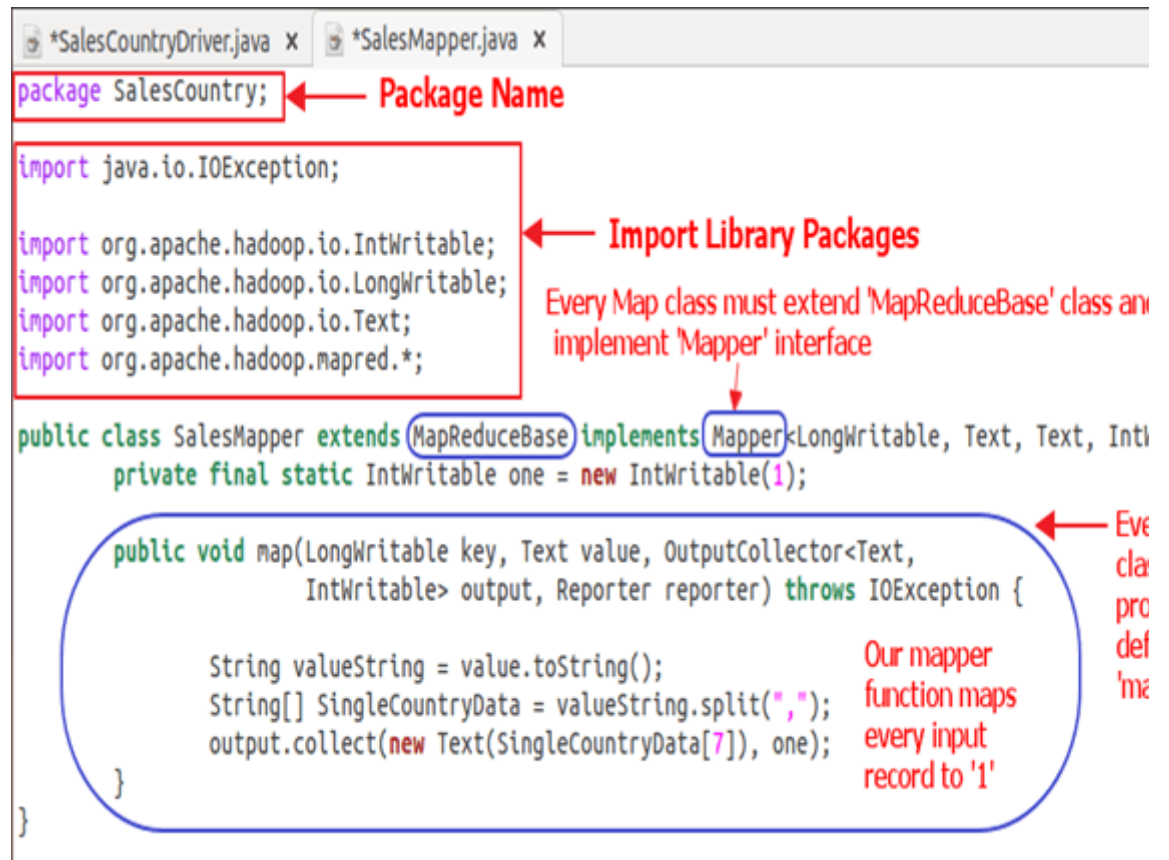
Explanation of SalesMapper Class

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 our package. The output of compilation, **SalesMapper.class** will go into directory named by this package 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, Text, IntWritable>`

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 tokens to split 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 0 index)-

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

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

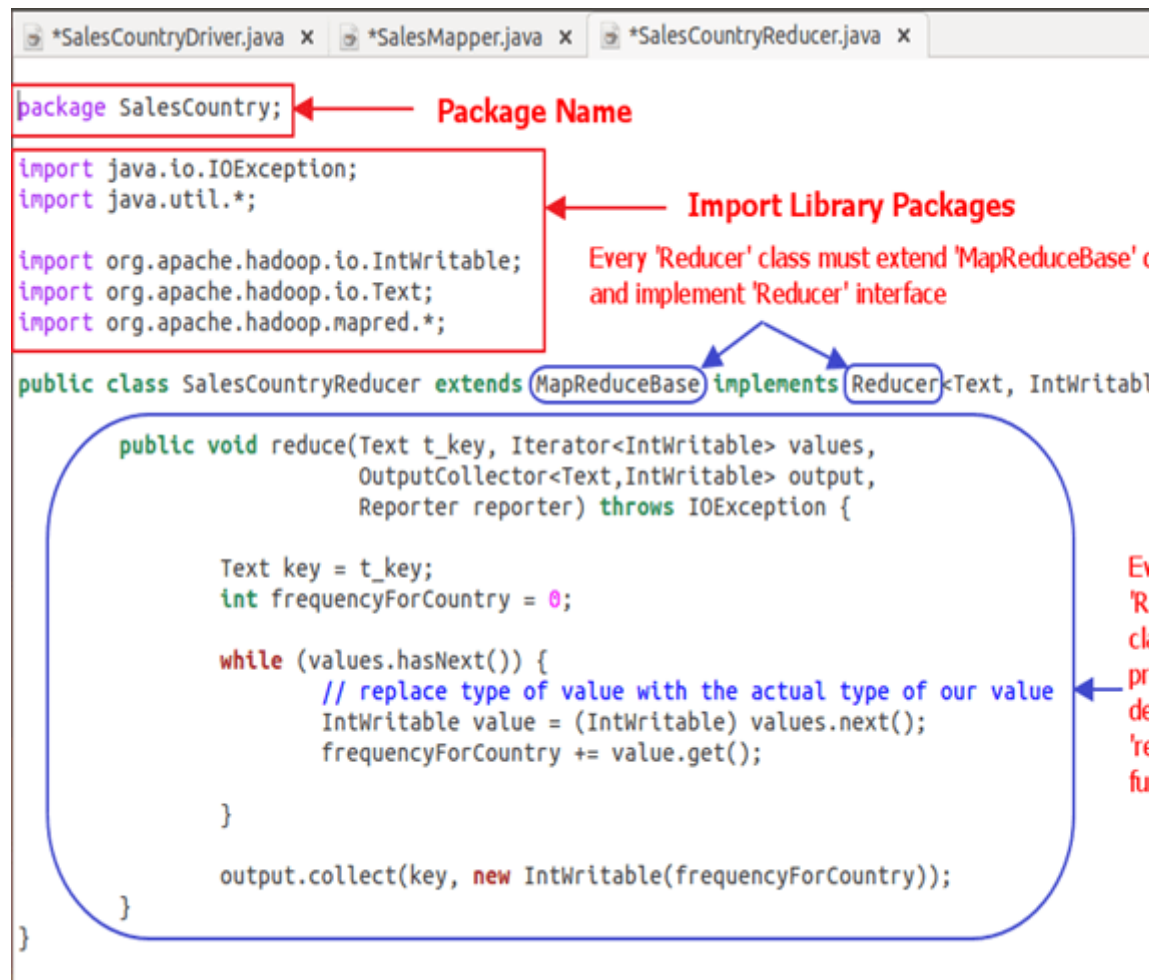
Explanation of SalesCountryReducer Class

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 our package that output of compilation, **SalesCountryReducer.class** will go into directory named by this 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, 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 calculate 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 our package 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 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.**

```

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);
    }
}

```

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

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

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

```

// Set input and output directories using command line arguments,


```

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

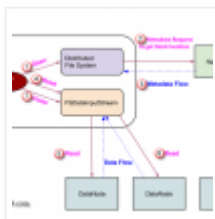
YOU MIGHT LIKE:

HBASE



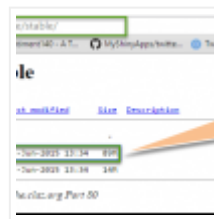
HBase Tutorials for Beginners

BIGDATA



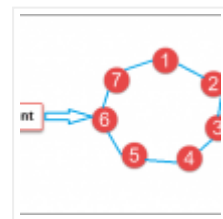
HDFS Tutorial: Read & Write Commands using Java API

HIVE



Installation and Configuration of HIVE and MYSQL

CASSANDRA



Cassandra Database Tutorial for Beginners: Learn in 3 Days

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.

^ | v • 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 h
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?

^ | v • Reply • Share ›



Gopalvq • 9 months ago

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


^ | v • Reply • Share ›



Gopalvq → Gopalvq • 9 months ago

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

^ | v • Reply • Share ›

 Subscribe  Add Disqus to your site Add Disqus Add  Privacy

Go on the Roadtrip of a Lifetime in New Zealand.

Drive your way through our beautiful country and explore stunning landscapes wherever you go.

[Learn More](#)

Sponsored by **Tourism New Zealand**

About

[About us](#)
[Advertise with Us](#)
[Jobs](#)
[Privacy Policy](#)

Contact Us

[Contact us](#)
[FAQ](#)
[Write For Us](#)

Follow Us

Certificat

[ISTQB Cer](#)
[MySQL Ce](#)
[QTP Certif](#)
[Testing Ce](#)
[CTAL Exar](#)

© Copyright - Guru99 2017