SalesReducer Class 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-

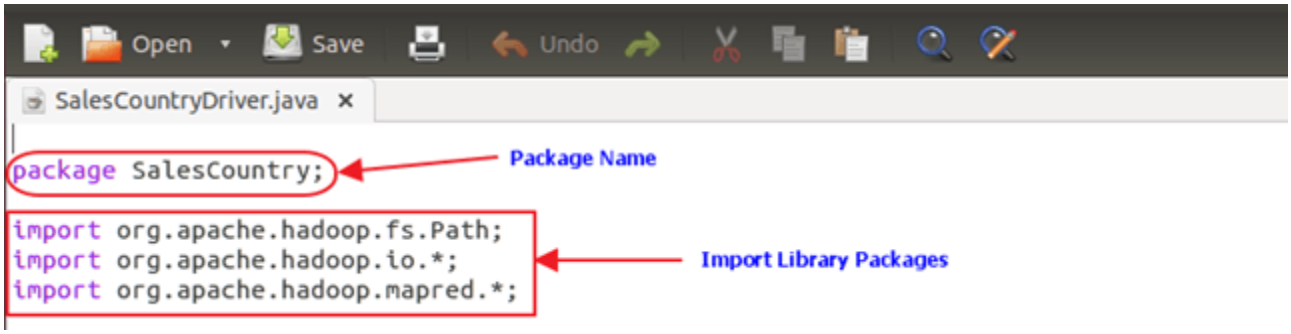
```
output.collect(key, new IntWritable(frequencyForCountry));
```

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.



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

```
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);
```

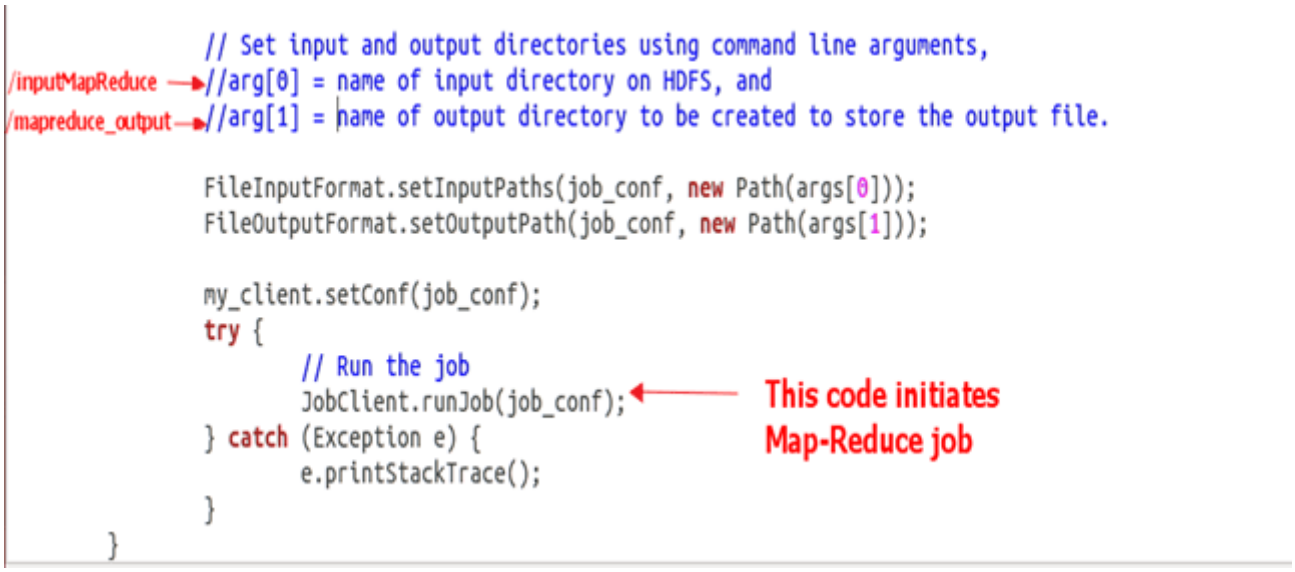
Start of definition of SalesCountryDriver class

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



4. Trigger our job

Below code start execution of MapReduce job-

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