# Big Data Hadoop Training

Session 5 Assignment 1 Solution:

Task 4:

Q) **Write a Map Reduce program to modify Task 2 (refer session 4, assignment 2) to use a custom partitioner with 4 reducers.**

**Make sure that all records whose company name starts with A-F (upper or lower case) should go to 1st reducer, those starting with G-L to 2nd reducer, those starting with M-R to 3rd reducer and others to 4th reducer.**

A) **Java Program**: Takes HDFS Path for **television.txt** as input and filters-out the invalid records and stores the result in output path as a **part-file.**

**Note:** Input file kept into HDFS in **/tmp/access\_folder/ folder**

Output path provided: **/tmp/access\_folder/television**

**import** java.io.IOException;

**import** org.apache.hadoop.conf.Configuration;

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

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

**import** org.apache.hadoop.io.LongWritable;

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

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**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.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Partitioner;

**import** org.apache.hadoop.mapreduce.Reducer;

**public** **class** custom\_partitioner\_combiner{

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, IntWritable> {

**private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);

**private** Text word = **new** Text();

**public** **void** map(LongWritable key, Text value, Context context) **throws** IOException,InterruptedException {

String line = value.toString();

String[] elements = line.split("\\|"); // Delimiter "|" to separate words

word.set(elements[0]);

context.write(word, ***one***);

}

}

**public** **static** **class** CustomPartitioner **extends** Partitioner<Text, IntWritable> {

String partitionkey;

@Override

**public** **int** getPartition(Text key, IntWritable value, **int** numReduceTasks) {

partitionkey = key.toString();

**char** firstChar = partitionkey.charAt(0);

//if company name starts with A-F (upper or lower case), assign partition 0 i.e. Reducer-1

**if** ((firstChar >= 'a' && firstChar <= 'f')||(firstChar >= 'A' && firstChar <= 'F')) {

**return** 0;

}

//if company name starts with G-L (upper or lower case), assign partition 1 i.e. Reducer-2

**if** ((firstChar >= 'g' && firstChar <= 'l')||(firstChar >= 'G' && firstChar <= 'L')) {

**return** 1;

}

//if company name starts with M-R (upper or lower case), assign partition 2 i.e. Reducer-3

**if** ((firstChar >= 'm' && firstChar <= 'r')||(firstChar >= 'M' && firstChar <= 'R')) {

**return** 2;

}

//Rest to partition 3 i.e. Reducer-4

**else**

**return** 3;

}

}

**public** **static** **class** Reduce **extends** Reducer<Text, IntWritable, Text, IntWritable> {

**public** **void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException,InterruptedException {

**int** sum = 0;

**for** (IntWritable val : values) {

sum += val.get();

}

context.write(key, **new** IntWritable(sum));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

@SuppressWarnings("deprecation")

Job job = **new** ~~Job~~(conf, "custom\_part\_comb");

job.setJarByClass(custom\_partitioner\_combiner.**class**);

// We require 4 Reducers in this task

job.setNumReduceTasks(4);

job.setMapperClass(Map.**class**);

job.setPartitionerClass(CustomPartitioner.**class**);

job.setReducerClass(Reduce.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(IntWritable.**class**);

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

job.waitForCompletion(**true**);

}

}

**Input format: “**hdfs://<hostname>:<port\_no>/<HDFS\_Input\_File\_Path>**” “**hdfs://<hostname>:<port\_no>/<HDFS\_Output\_File\_Path>**”**

**Running in Eclipse:**

* Input given in Run Configurations Argument Tab as : "hdfs://pavan:8020/tmp/access\_folder/television.txt" "hdfs://pavan:8020/tmp/access\_folder/television"

**Note: Here Hostname: pavan and Port no:8020 is required since files are in HDFS**

* Filtered Output can be seen with the help of :

**hadoop fs –cat /tmp/access\_folder/television/part-00000** command (Become a hdfs user to use this command)

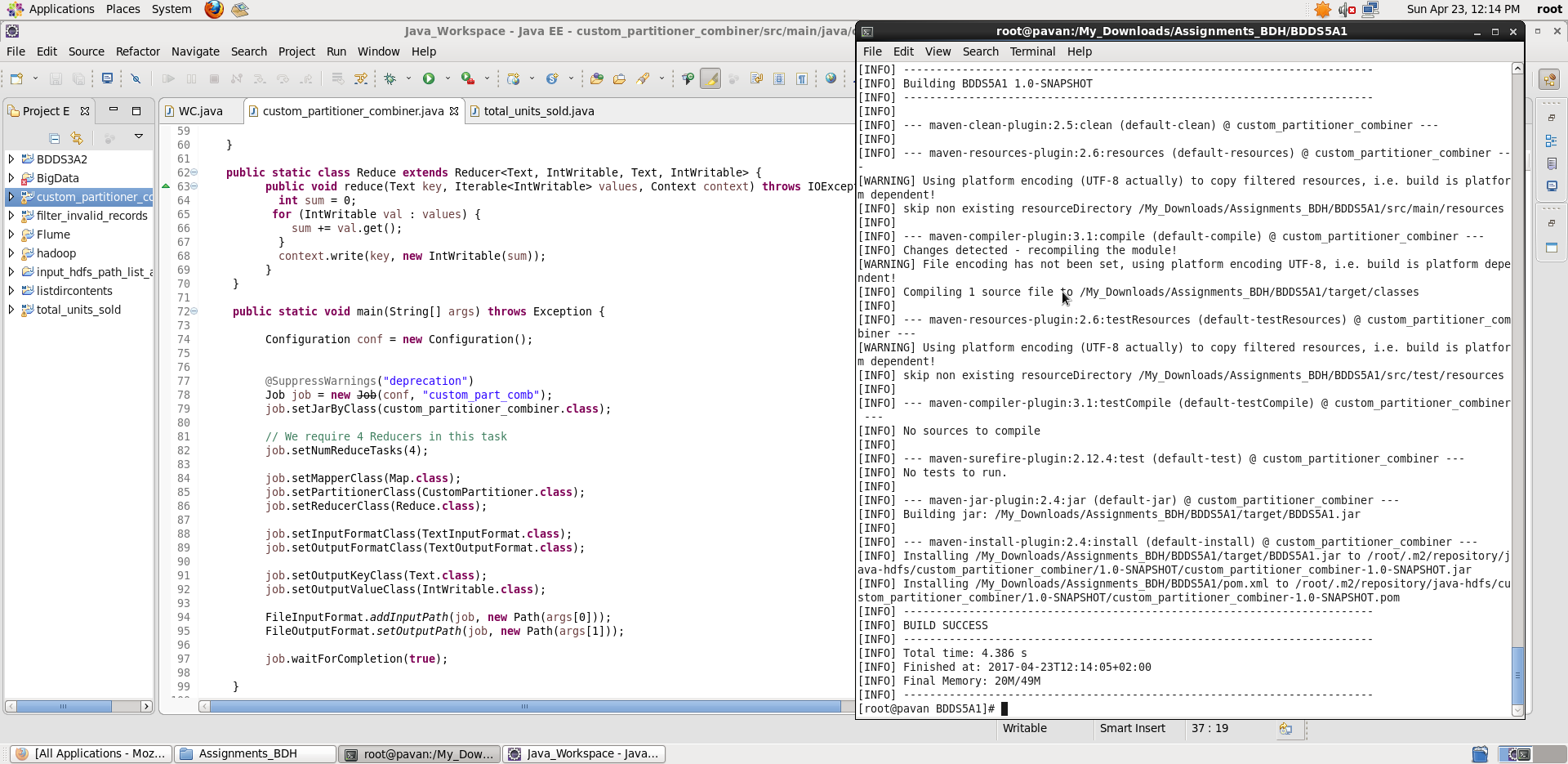
**Running in Command Line:**

* Make the folder structure BDDS4A1/src/main/java/filter\_invalid\_records.java
* Then in BDDS4A1 folder, open terminal
* Create the Jar file target/BDDS4A1.jar using Maven as **root user** # mvn clean install
* Now , become HDFS user # su hdfs
* Run the Hadoop Job : “hadoop jar <jar\_file\_path> <class\_name> <input\_hdfs\_file\_path> <output\_hdfs\_file\_path>”
* In our case : **hadoop jar target/BDDS4A1.jar filter\_invalid\_records /tmp/access\_folder/television.txt /tmp/access\_folder/television**
* **Note: Here both input and output present in HDFS, so no Hostname and Port No specification required.**
* Filtered Output can be seen with the help of :

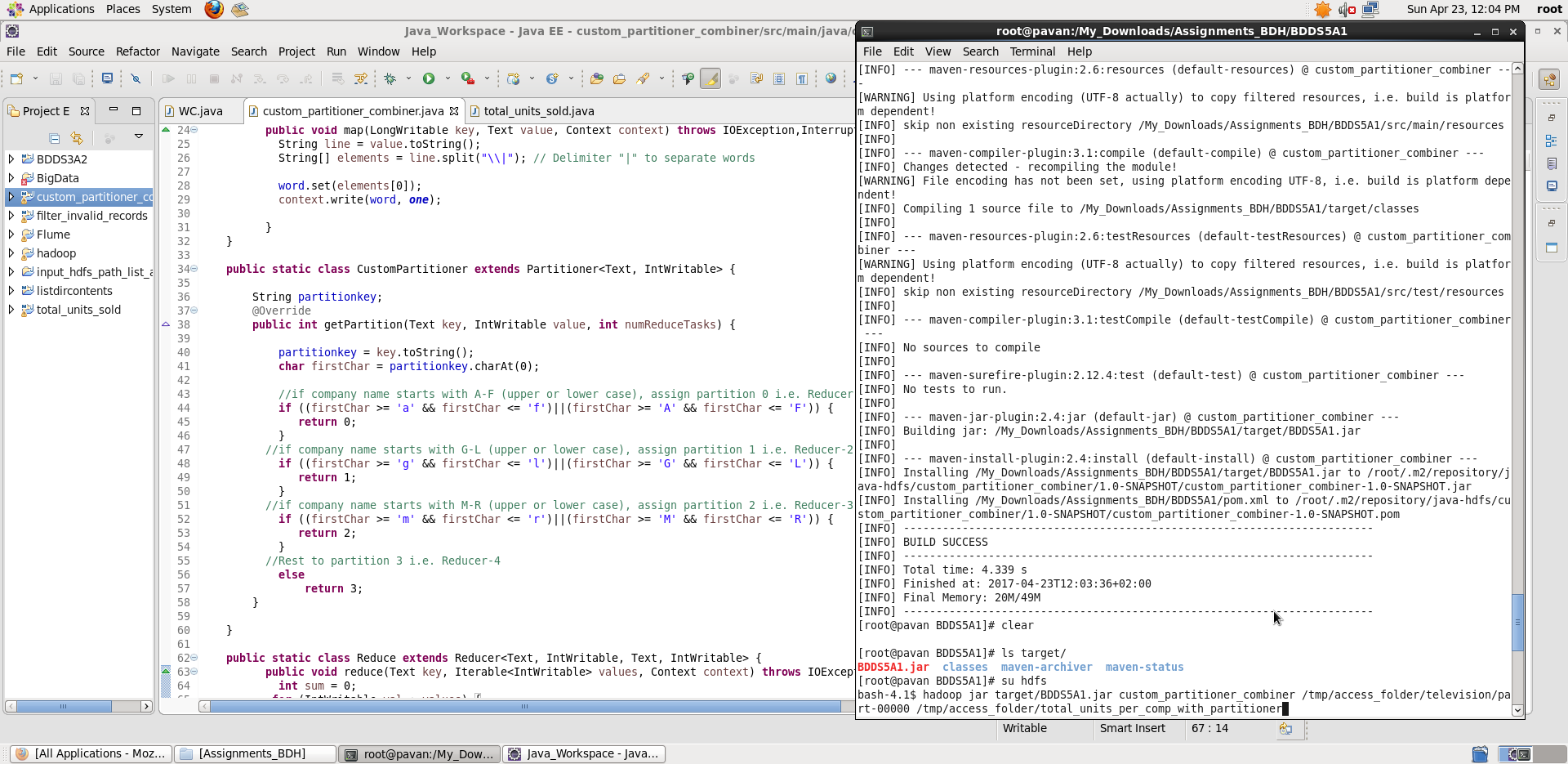
**hadoop fs –cat /tmp/access\_folder/television/part-00000** command (Become a hdfs user to use this command)

Screenshots

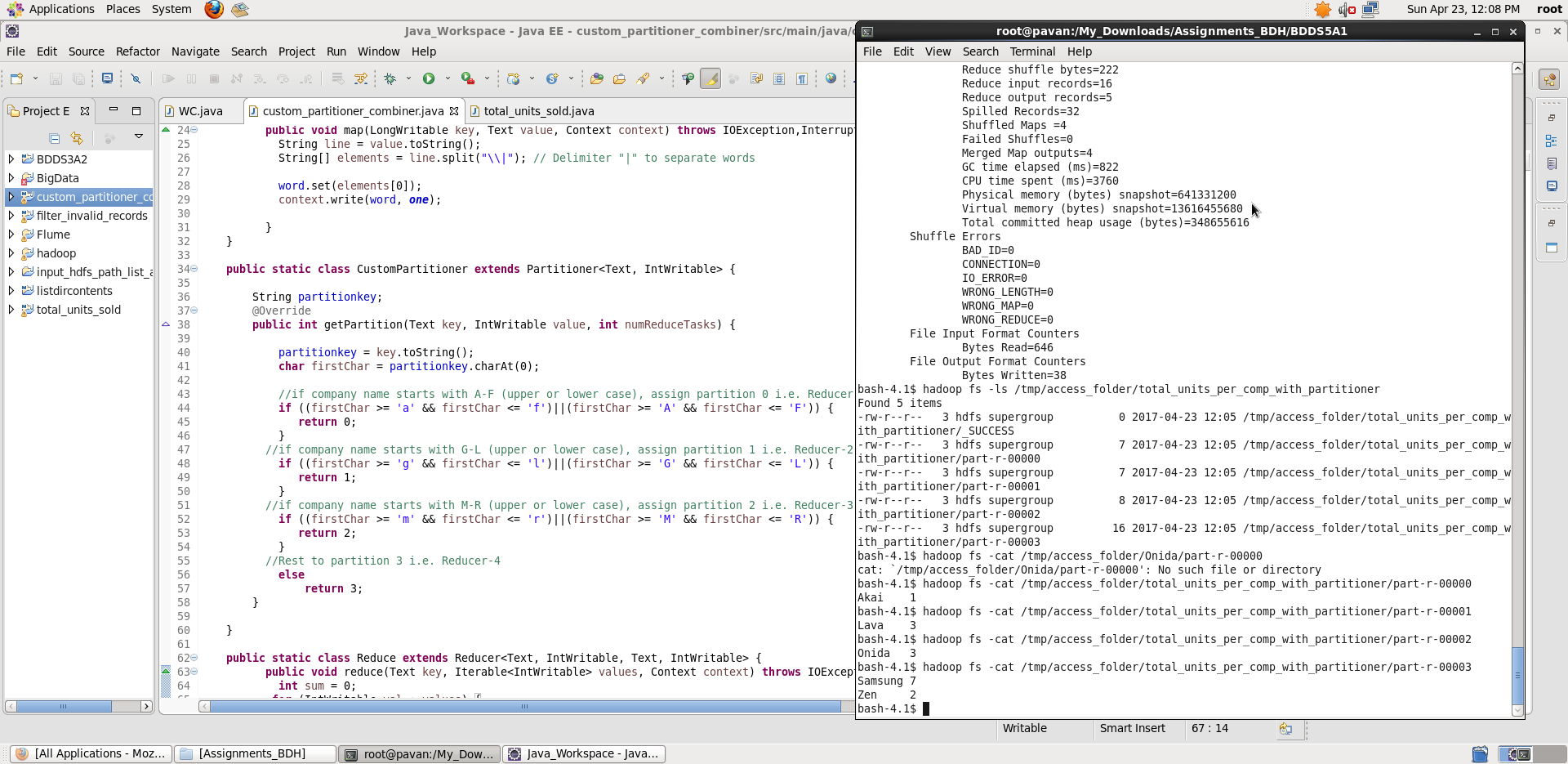
Eclipse Java code and Building it as Maven Project in cmd line :



**Submitting Job as HDFS user:**

****

Once we **Run the Hadoop Job**, we see 4 **part-files generated each corresponding to each reducer.** Displaying **output** using hadoop fs –cat command:

****