**EXPERIMENT - 3**

**AIM**:

To run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

1. Find the number of occurrences of each word appearing in an input file.
2. Performing a Map Reduce Job for word search count (look for specific keywords in a file).

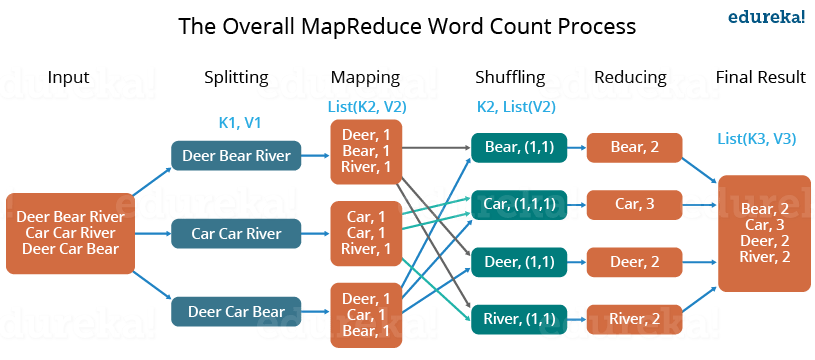
**THEORY**:

Let us understand, how a MapReduce works by taking an example considering a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

The aim is to find the unique words in the text file and the number of occurrences of those unique words.

1) First, the input is divided in three splits as shown in the below figure. This will distribute the work among all the map nodes.



**Fig. 1. The Overall MapReduce WordCount Process**

2)Then each of the mapper the words are tokenized and given a value(1)to each of the tokens or words.the idea behind giving a value equal to 1is that every word ,in itself,well occur once.

3) Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.

4) After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.

5) After the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1]; etc.

6) Now, each Reducer counts the values which are present in that list of values. As shown in the Fig. 1, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.

7) Finally, all the output key/value pairs are then collected and written in the output file.

**Importance of MapReduce:**

**1. Parallel Processing**

• In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously.

• MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines.

• As the data is processed by multiple machine instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount as shown in the figure below (2).

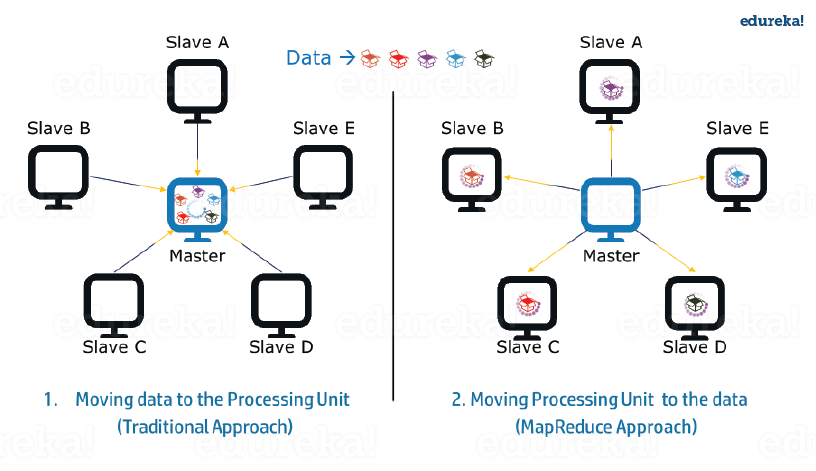


Fig. 2 Traditional Approach vs MapReduce Approach

**2. Data Locality**

• Instead of moving data to the processing unit, we are moving processing unit to the data in the MapReduce Framework.

• In the traditional system, data is brought to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed following issues:

− Moving huge data to processing is costly and deteriorates the network performance.

− Processing takes time as the data is processed by a single unit which becomes the bottleneck.

− Master node can get over-burdened and may fail.

• Now, MapReduce allows to overcome above issues by bringing the processing unit to the data. The data is distributed among multiple nodes where each node processes the part of the data residing on it. It has the following advantages:

− It is very cost effective to move processing unit to the data.

− The processing time is reduced as all the nodes are working with their part of the data in parallel.

− Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

i) **Find the number of occurrences of each word appearing in the input file(s) (USING JAVA)**

**Map Reduce Word Count Program:**

**WordCount.java**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount

{

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>

{

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

private Text word = new Text();

public void map(Object key, Text value, Context context) throws IOException

InterruptedException

{

StringTokenizer itr = new StringTokenizer(value.toString());

while (itr.hasMoreTokens())

{

word.set(itr.nextToken());

context.write(word, one);

}

}

}

public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>

{

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throw

IOException, InterruptedException

{

int sum = 0;

for (IntWritable val : values)

{

sum += val.get();

}

result.set(sum); context.write(key, result);

}

}

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

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[1])); System.exit(job.waitForCompletion(true) ? 0 : 1);

}

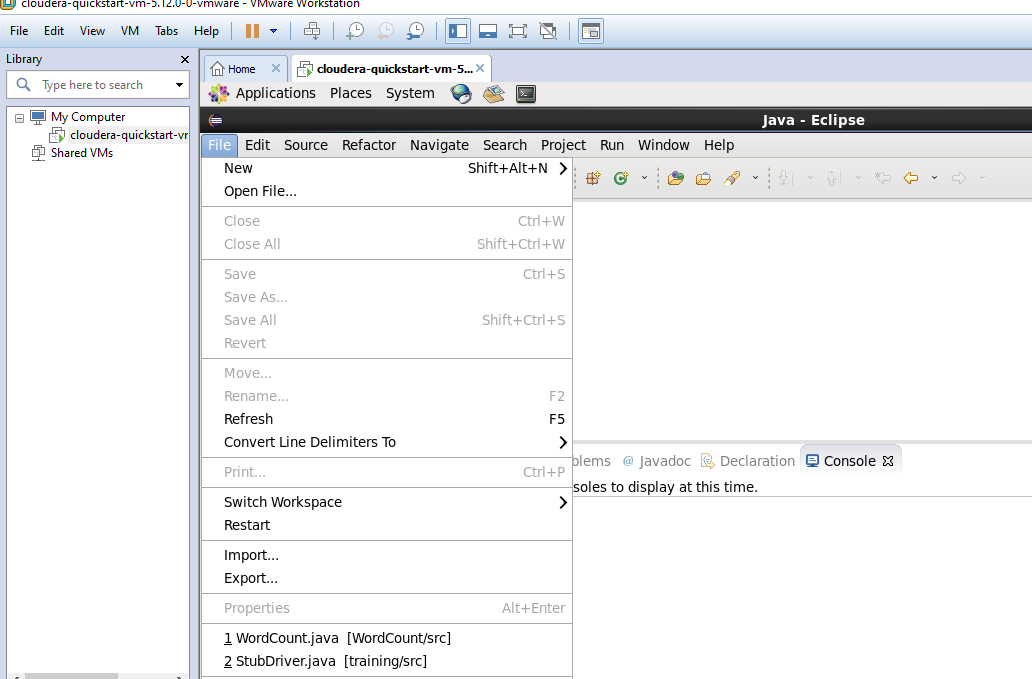
}

**Running the Word Count Map Reduce Program in Eclipse:**

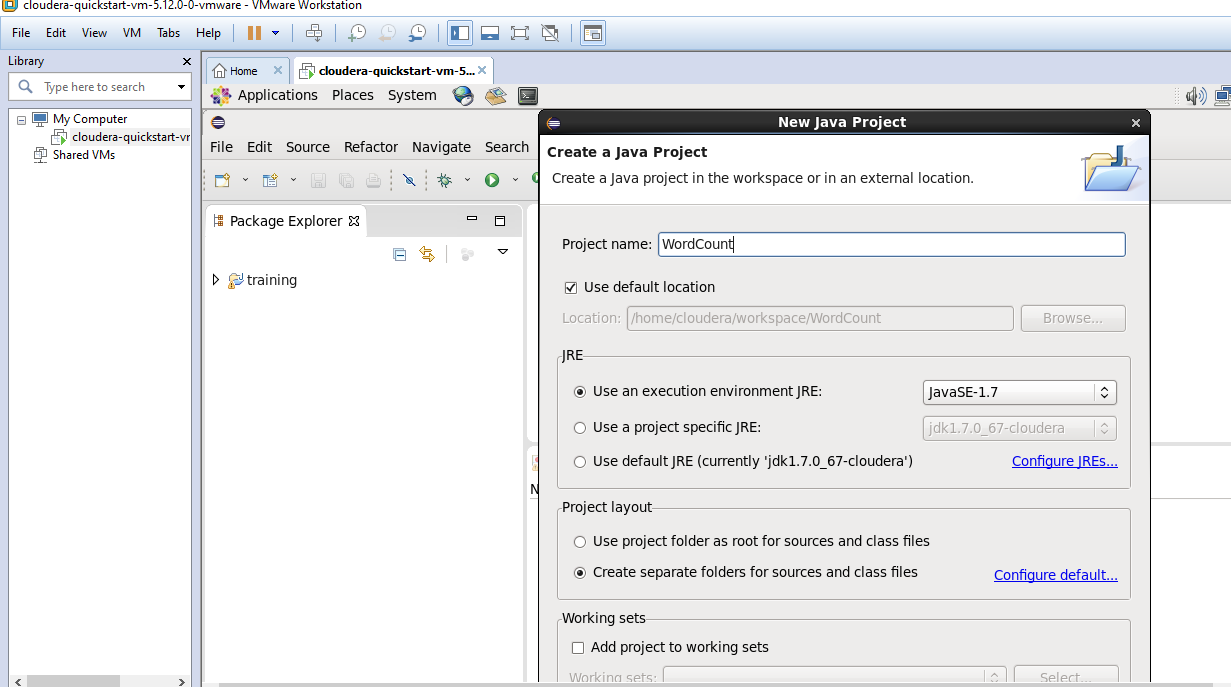
**STEP1)**- open cloudera

**STEP2)-**Open eclipse

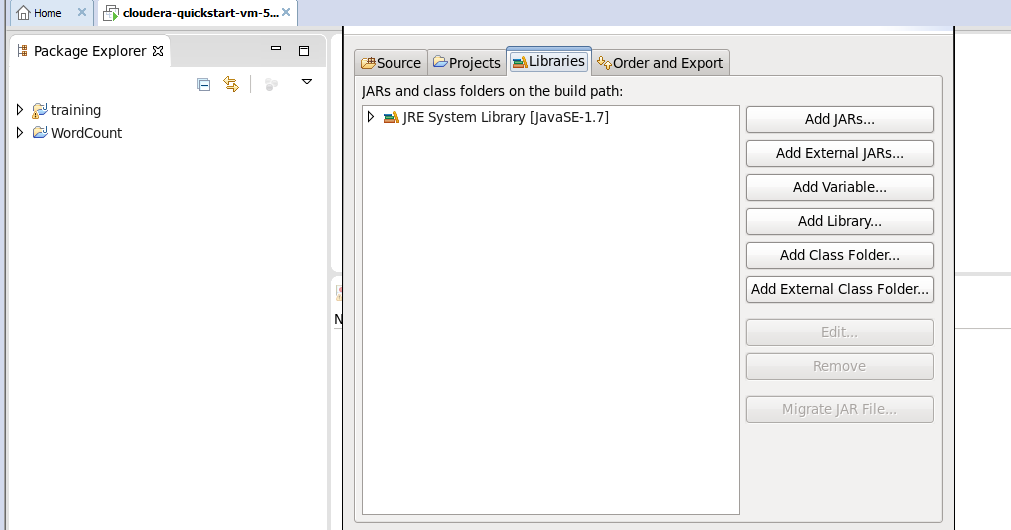
**STEP3)**-Click on file and select new file.

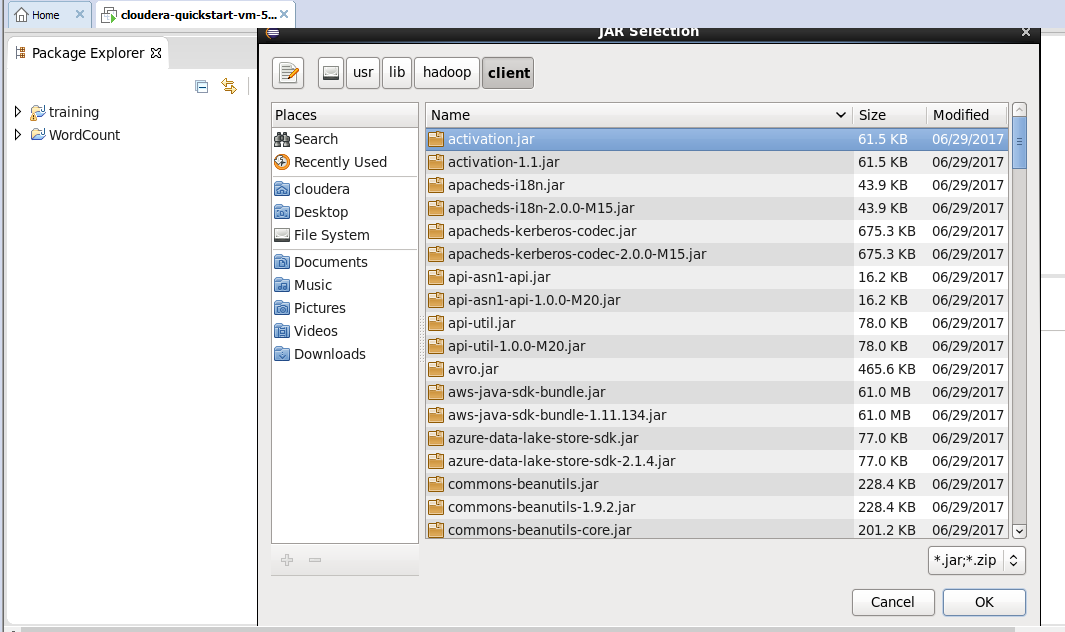
****

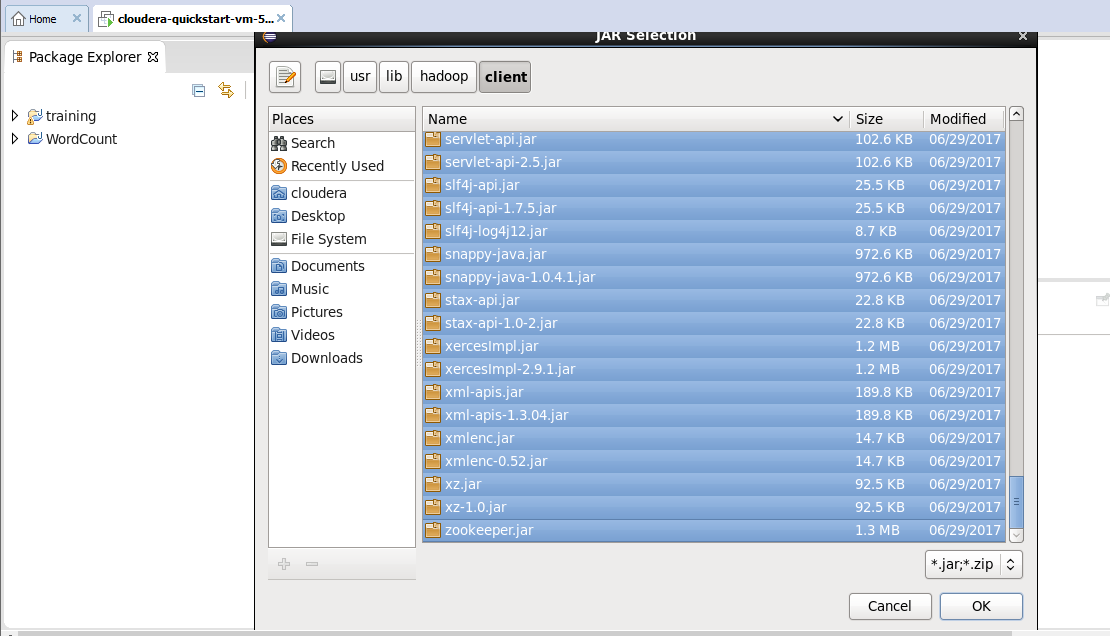
**STEP4)-**create a project name(WordCount)

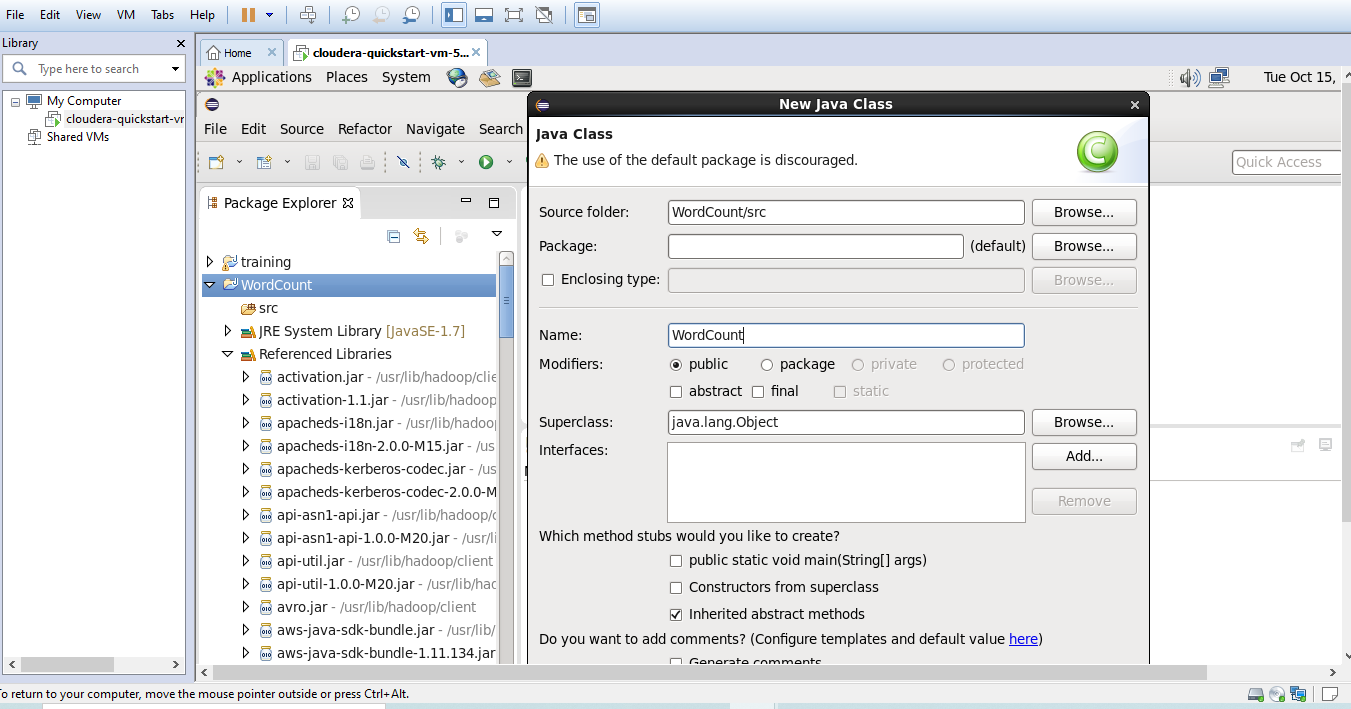


**STEP5)-**Click on libraries and add external jar file.

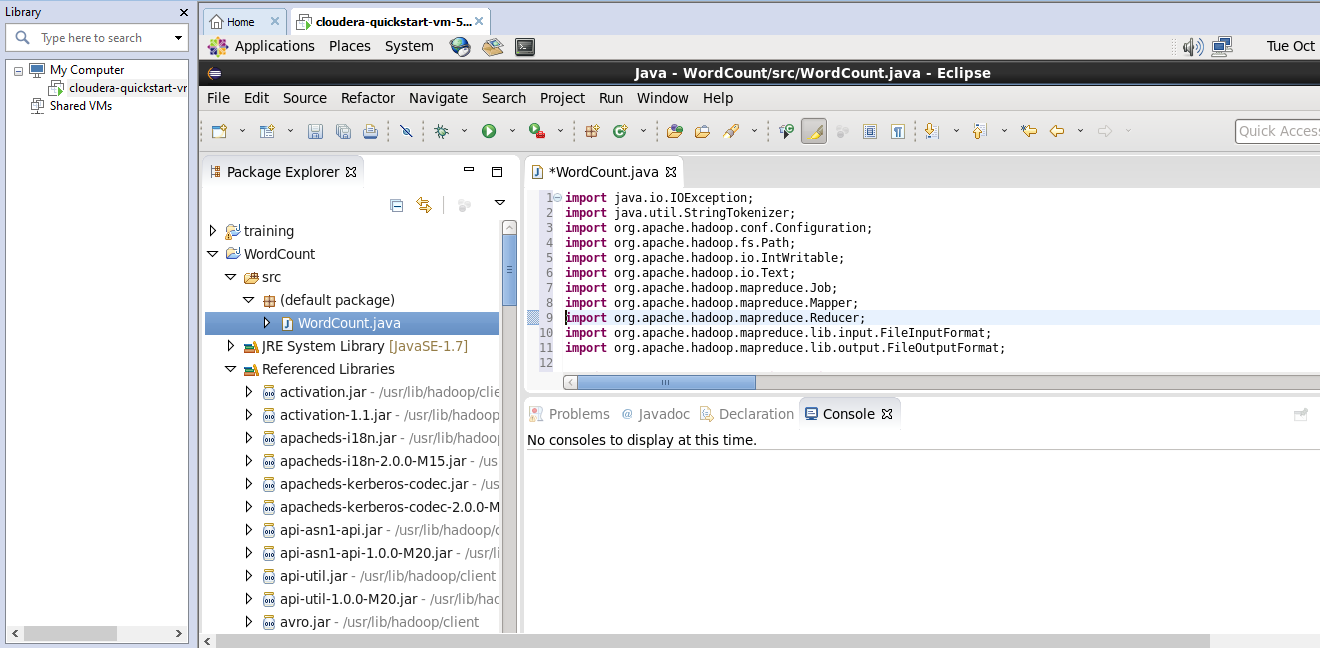


**STEP6)-** Select all jar files listed as shoen in below figure.****

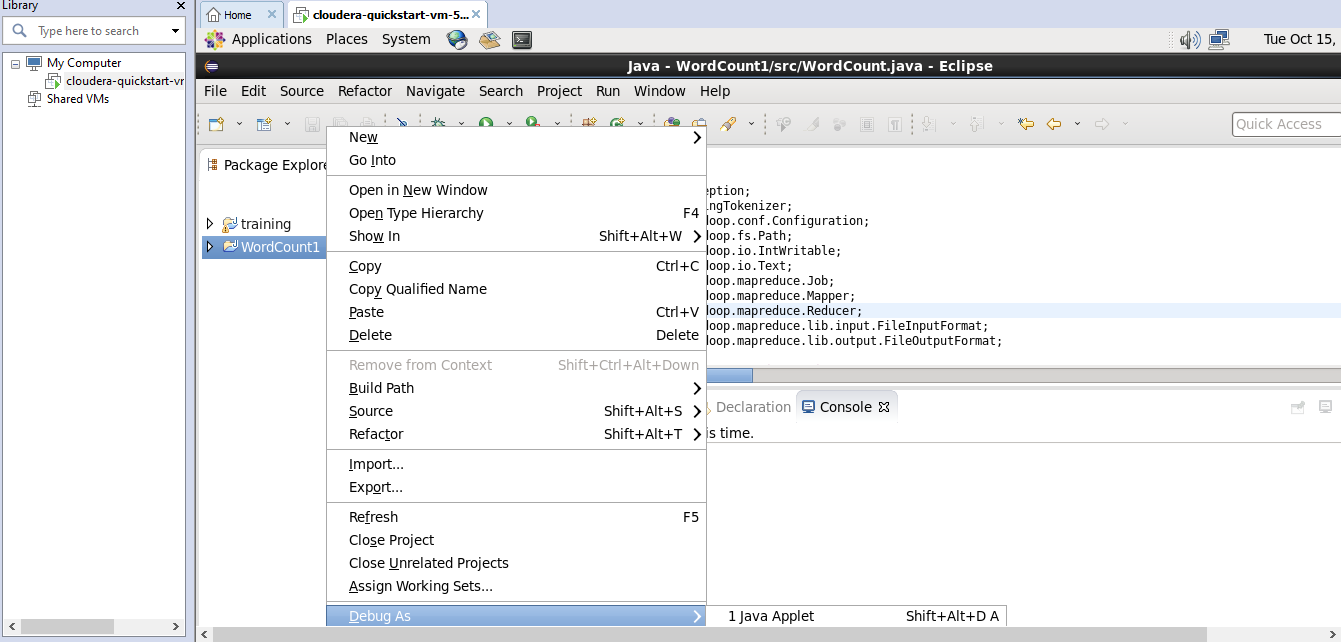
**STEP7-)** selectall jar file under client****

**STEP8)-** C**lick on wordcount and create class file named as (WordCount)**

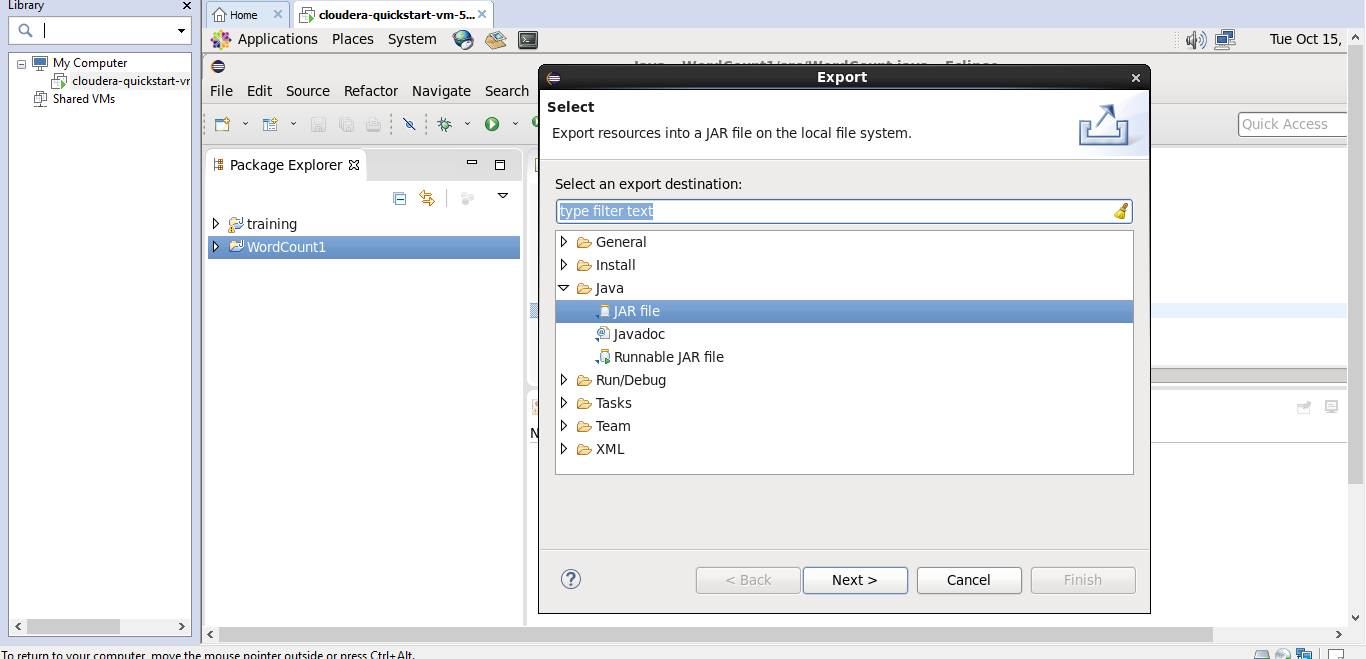
**STEP9)-write java mapreducer code on wordcount.java.**

****

**STEP10)-Now click on wordcount and import the java jar file to the cloudera.**

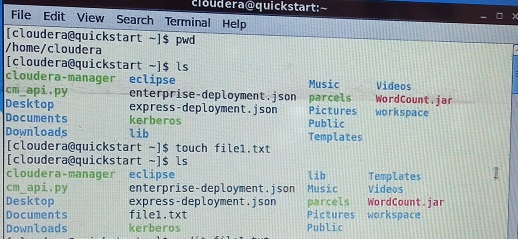
****

**Step11)-then click on export then java.**

****

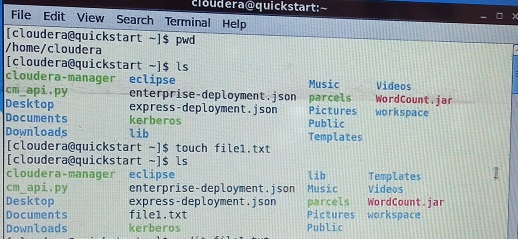
**OPEN CLOUDERA TERMINAL**

**STEP1>**to check jar file imported type(ls)

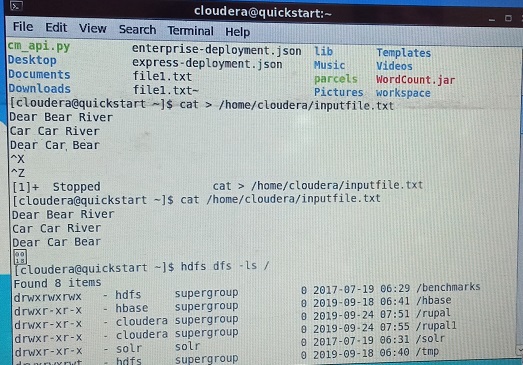


Above figure shows jar file .

**STEP2>**use touch<file name.txt> to store data.



STEP3>use cat<path><.txt>to see the stored data



**STEP3>** use –mkdir /<new folder>

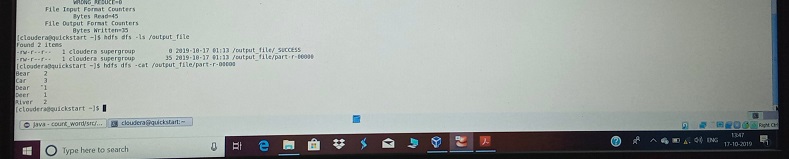
**STEP4>** use hdfs dfs –put<folder path><.txtfile><newfolder> to import text document local to remote system.

**STEP5>** use -cat <newfolder><.txtfile> to see the ouput of text file written in local system.

**STEP6>** use hadoop jar/wordcount.jar WordCount/<new folder name><.txt file>/output\_new to run wordcount program.

**STEP7>** hdfs dfs –cat/output\_new part-r-00000 to check the output of wordcount.

**Output**

****