**Practical - 5 Matrix Multiplication**

MapReduce is a technique in which a huge program is subdivided into small tasks and run parallel to make computation faster, save time, and mostly used in distributed systems. It has 2 important parts:

* **Mapper:** It takes raw data input and organizes into key, value pairs. For example, In a dictionary, you search for the word “Data” and its associated meaning is “facts and statistics collected together for reference or analysis”. Here the Key is *Data* and the **Value** associated with is *facts and statistics collected together for reference or analysis.*
* **Reducer:** It is responsible for processing data in parallel and produce final output.

**Matrix Multiplication:**

Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing.

**Matrix Data Model for MapReduce**

We represent matrix M as a relation M(I,J,V), with tuples  and matrix N as a relation N(J,K,W), with tuples . Most matrices are sparse so large amount of cells have value zero. When we represent matrices in this form, we do not need to keep entries for the cells that have values of zero to save large amount of disk space. As input data files, we store matrix M and N on HDFS







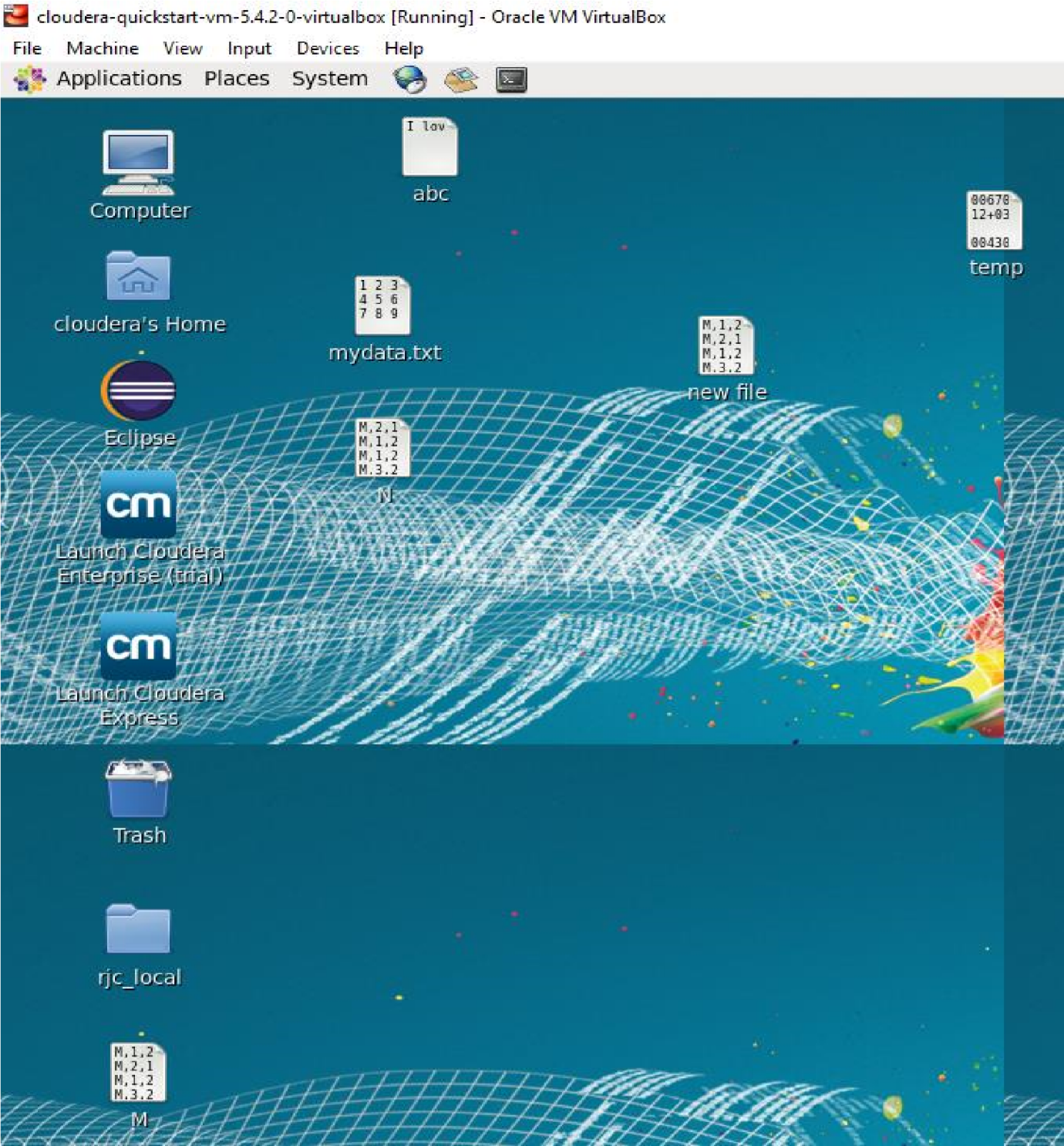


**Map Reduce:**

We will write Map and Reduce functions to process input files. Map function will produce key,value pairs from the input data as it is described in Algorithm 1. Reduce function uses the output of the Map function and performs the calculations and produces key,value pairs as described in Algorithm 2. All outputs are written to HDFS.

**Steps for MatrixMultiplication :**

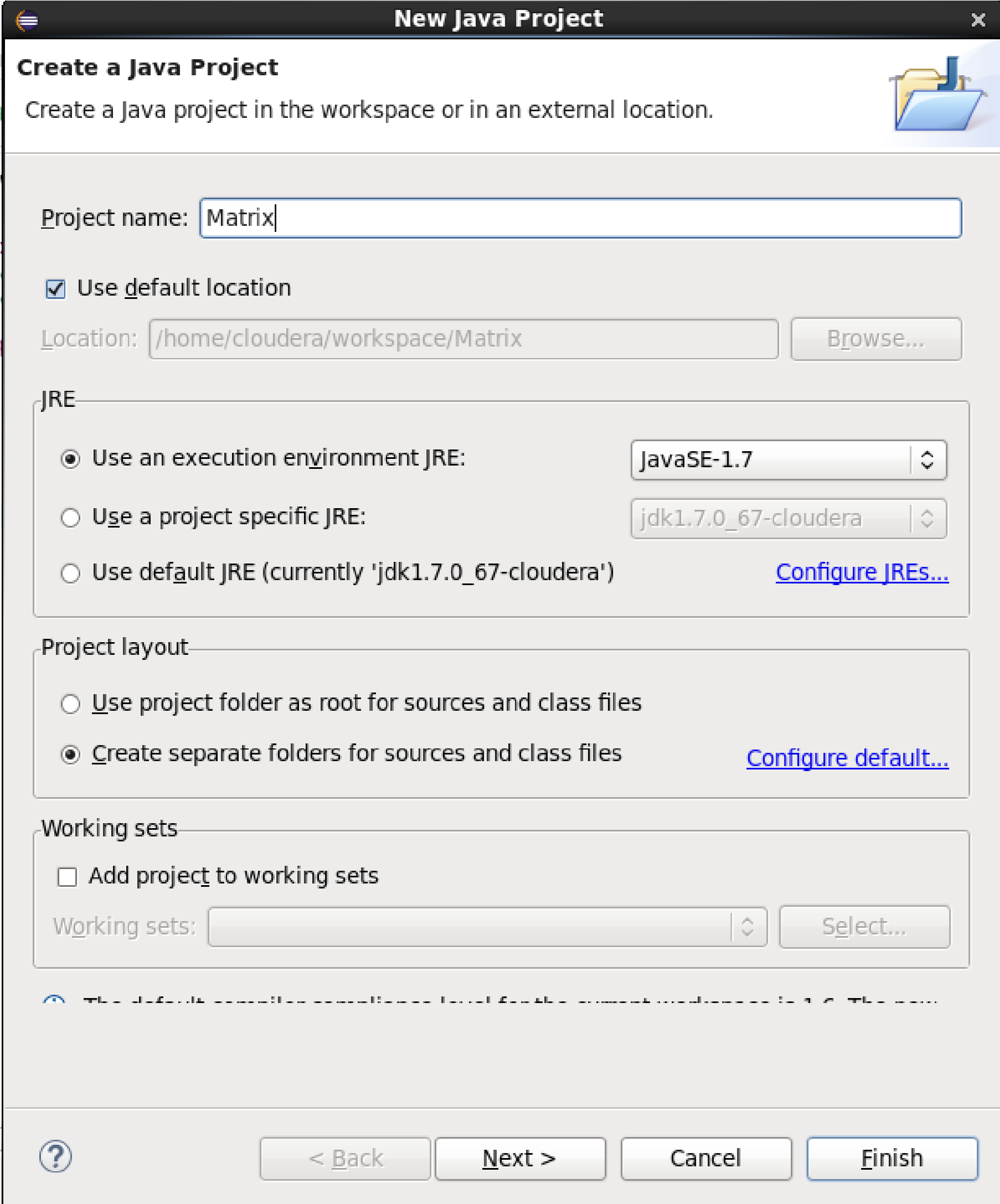
1. Open virtual box and then start cloudera quickstart



1. Open Eclipse present on the cloudera desktop

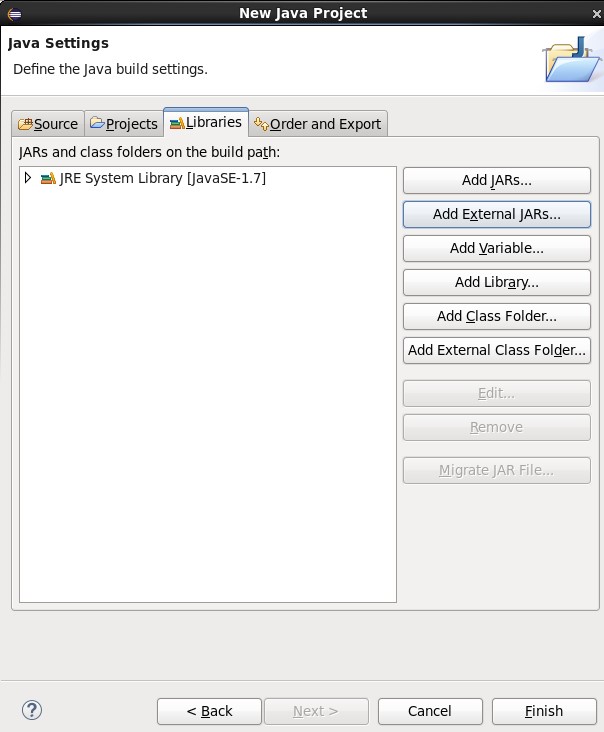


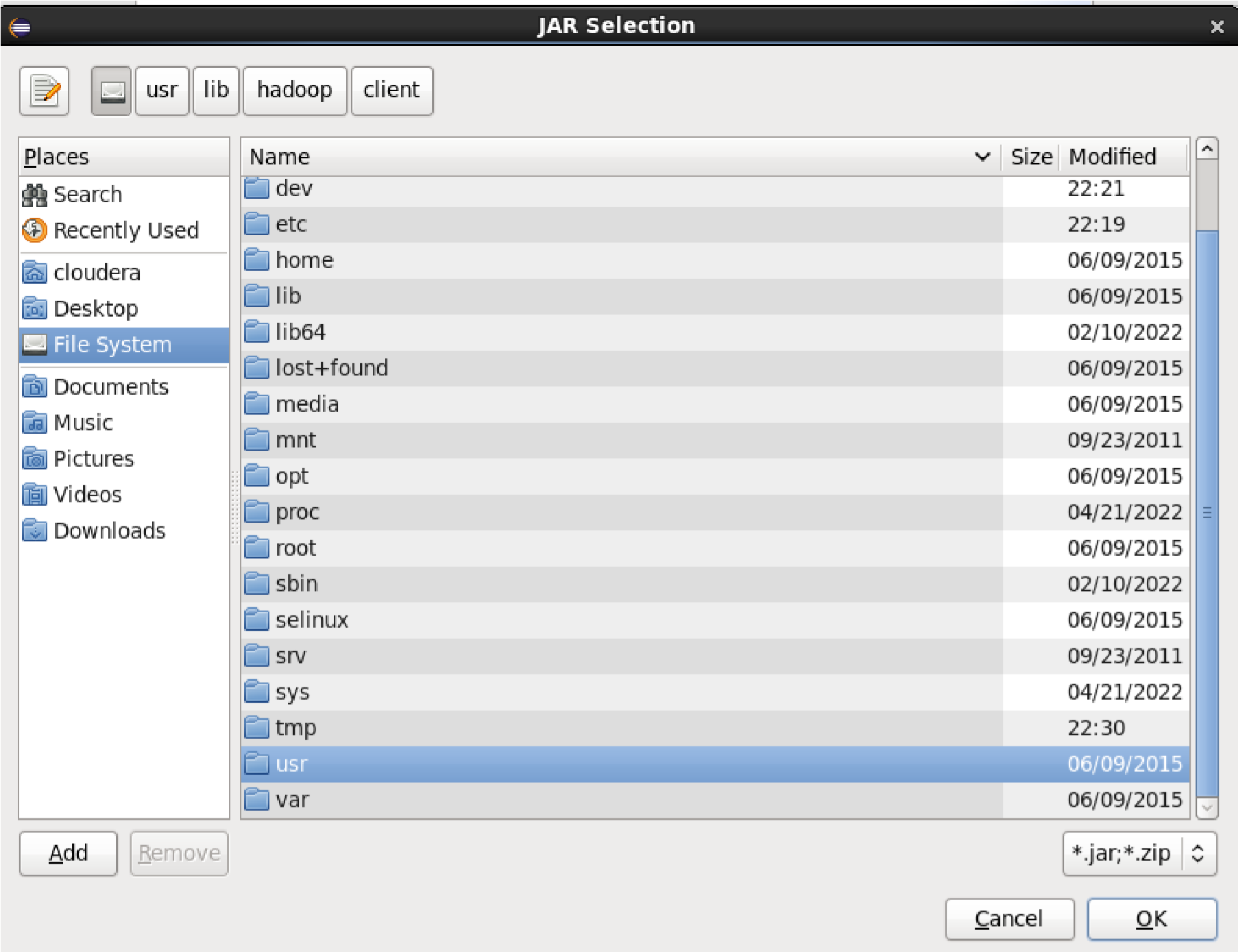
1. Create a new Java project clicking: File -> New -> Project -> Java Project -> Next (“Matrix” is the project name).

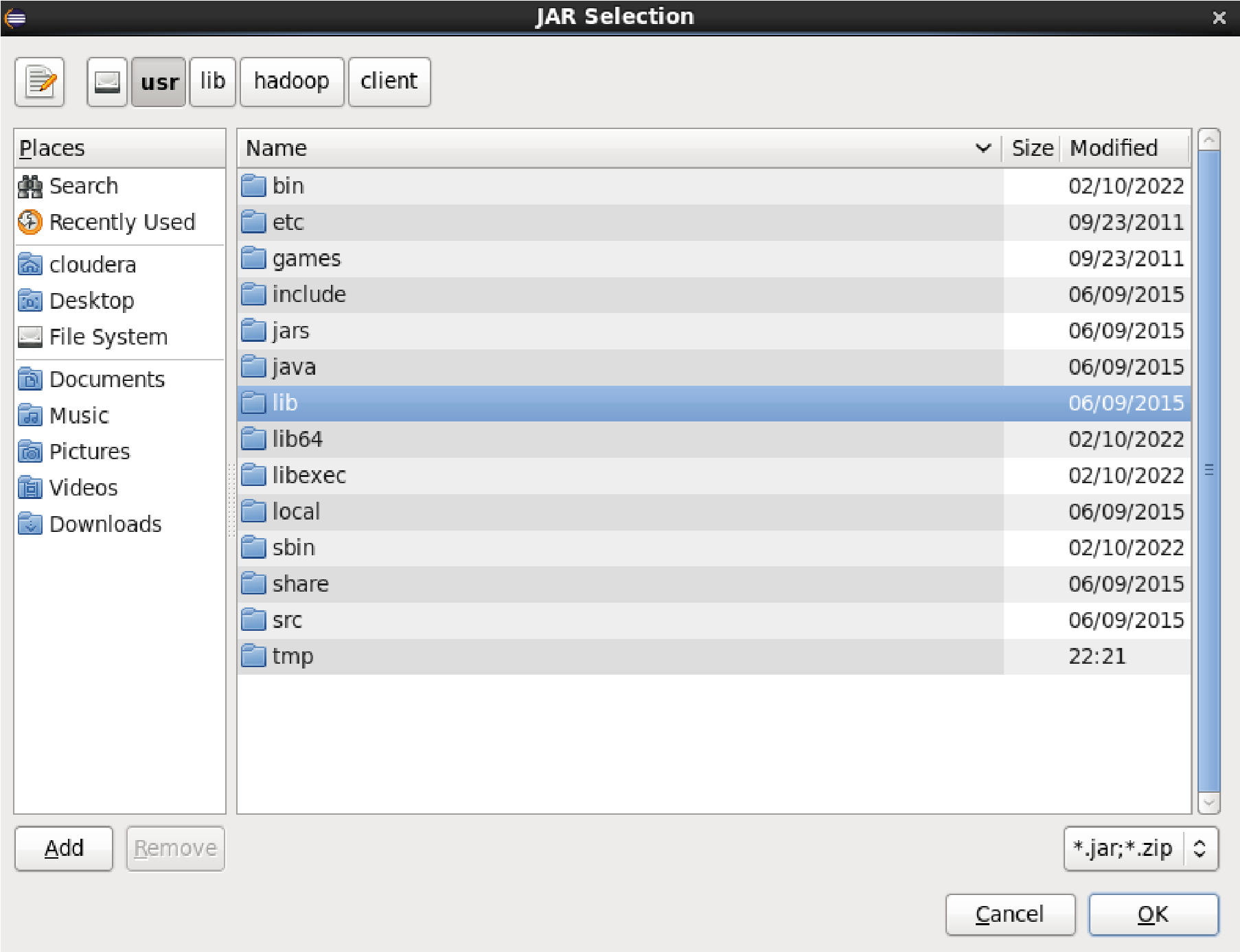


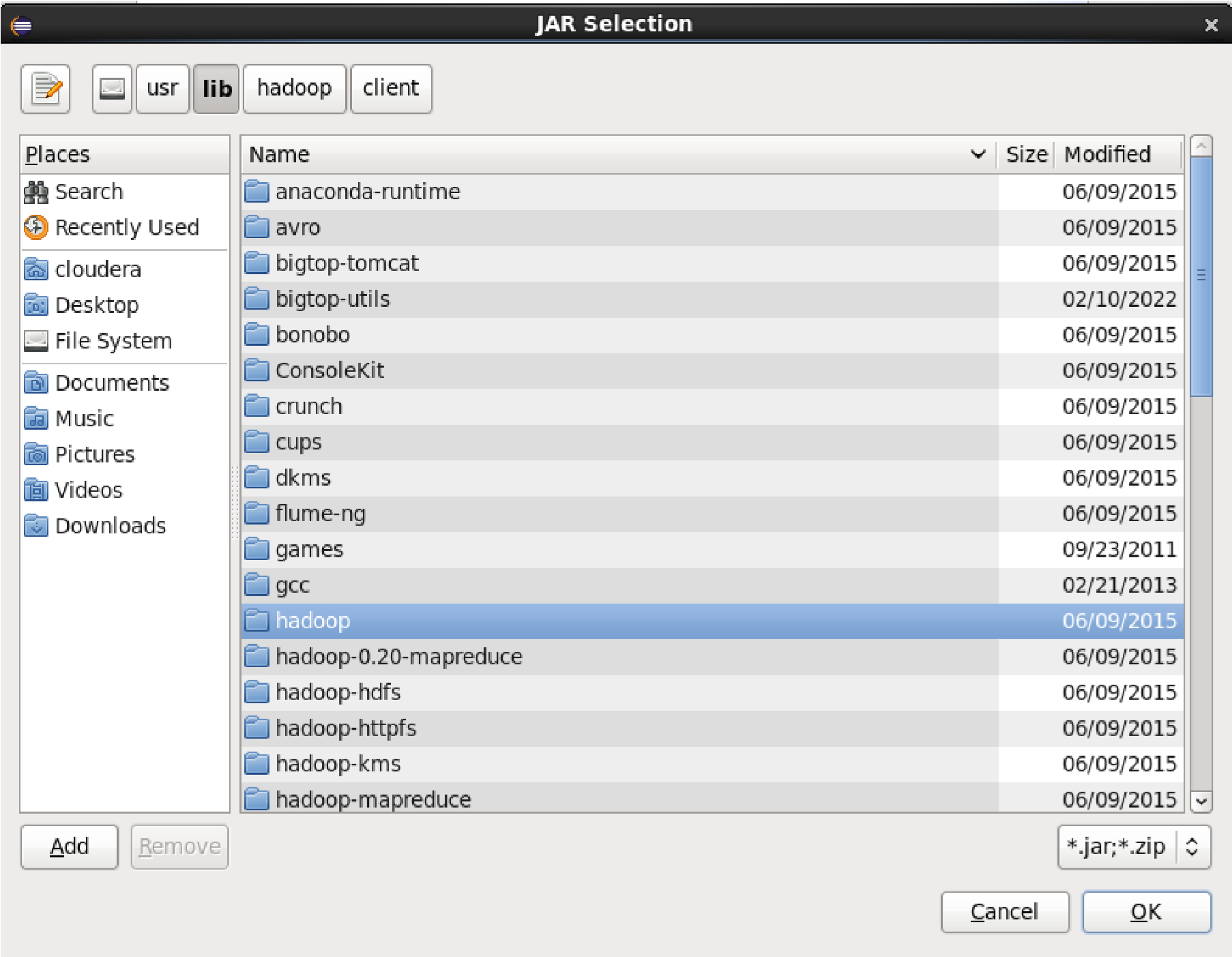
1. Adding the Hadoop libraries to the project Click on Libraries -> Add External JARs Click on

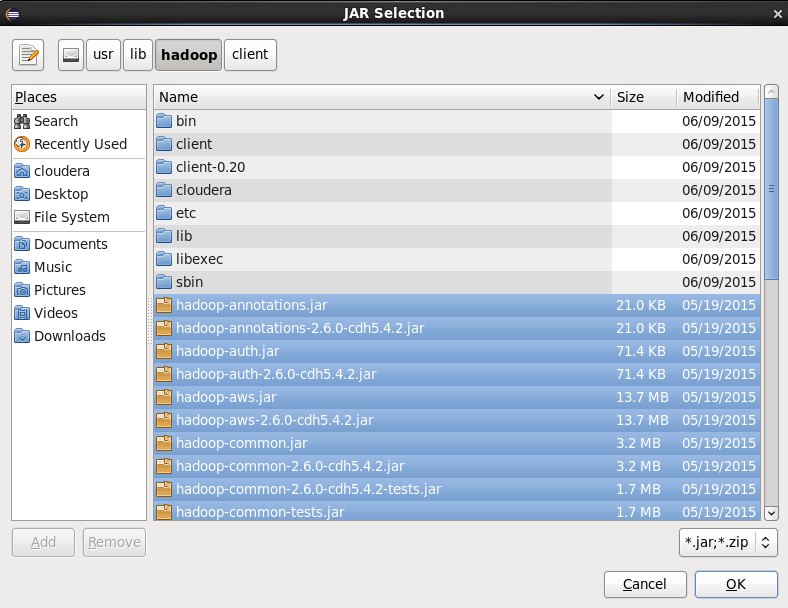
File System -> usr -> lib -> Hadoop, Select all the libraries (JAR Files) -> click OK Click on Add External jars, -> client -> select all jar files -> ok -> Finish



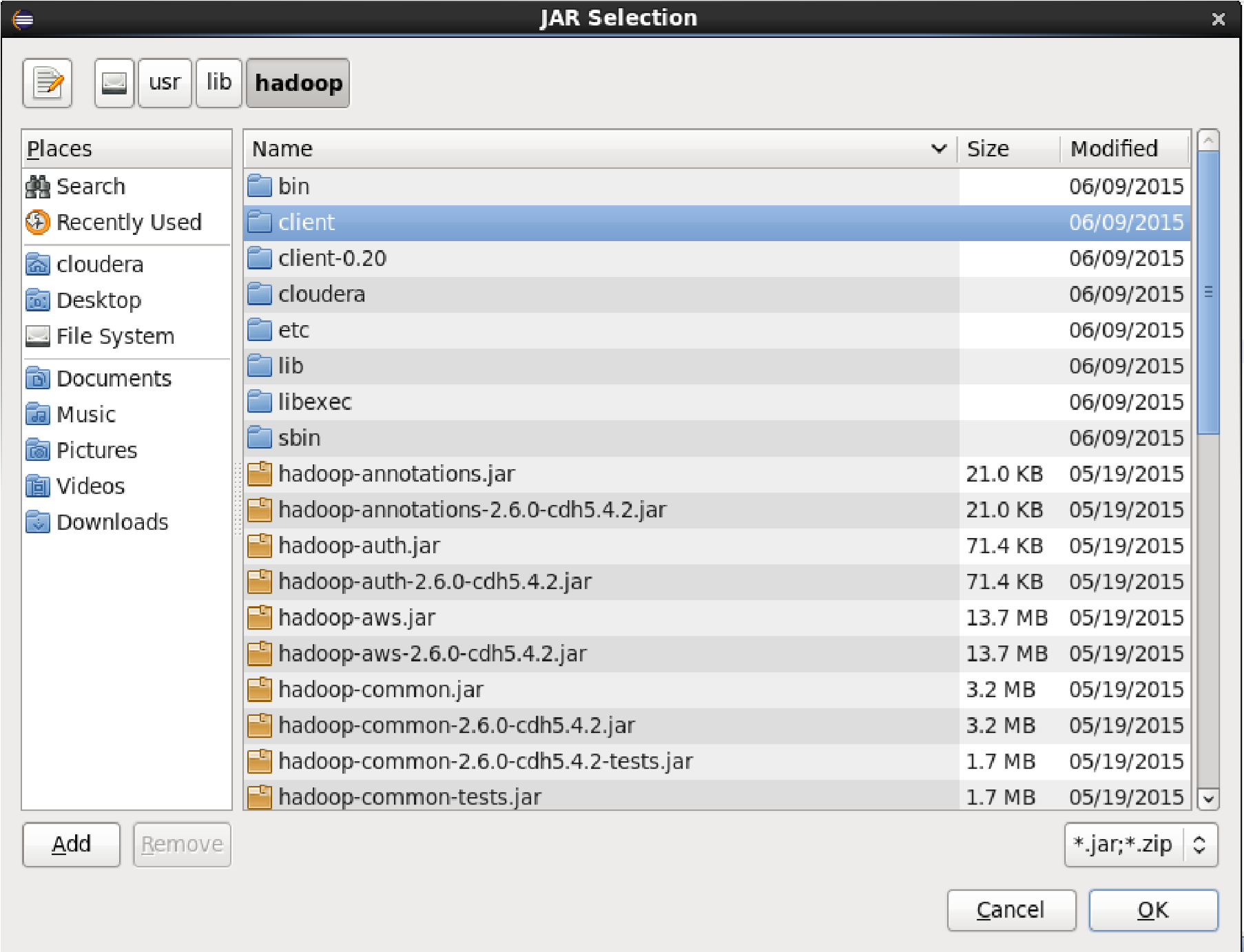


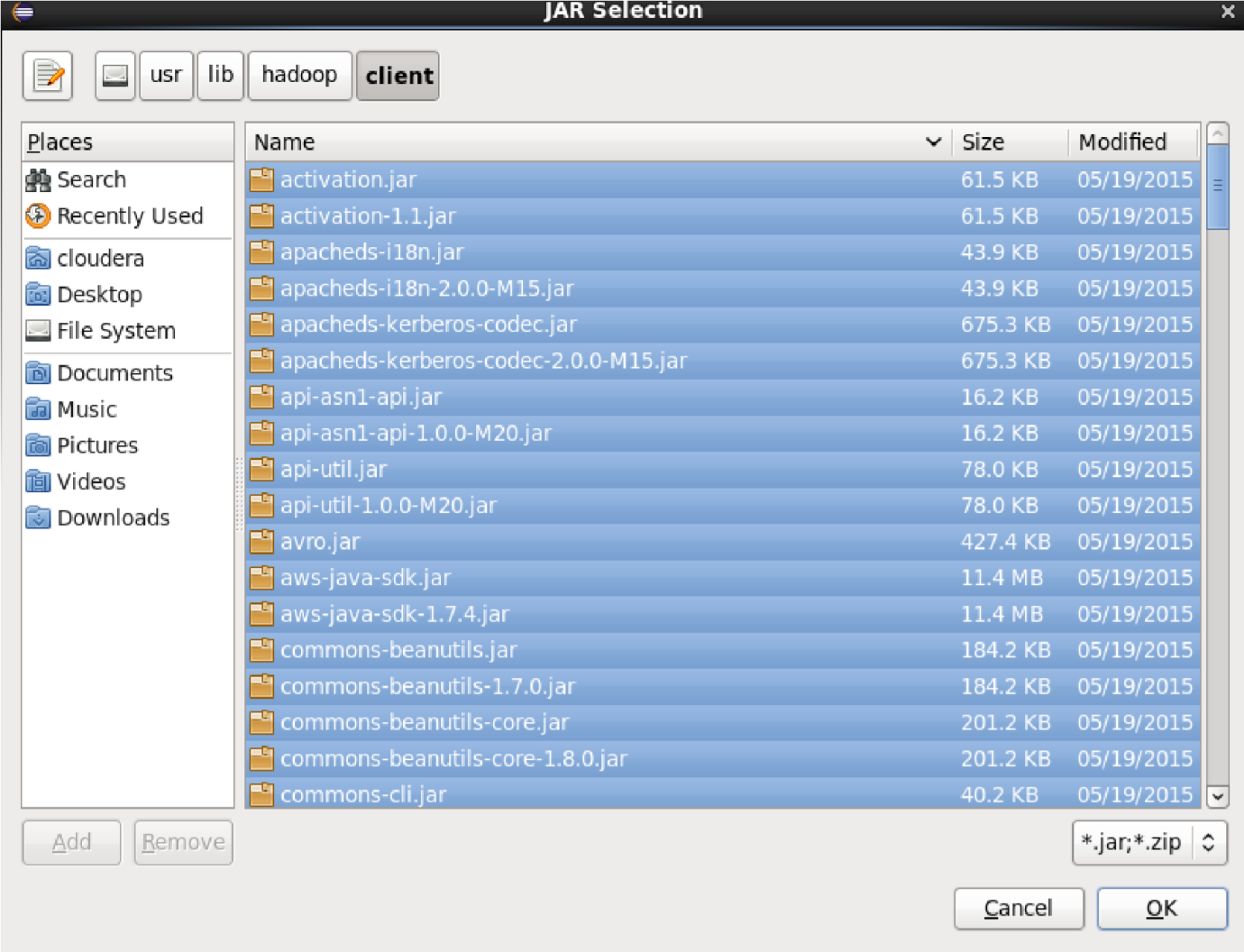




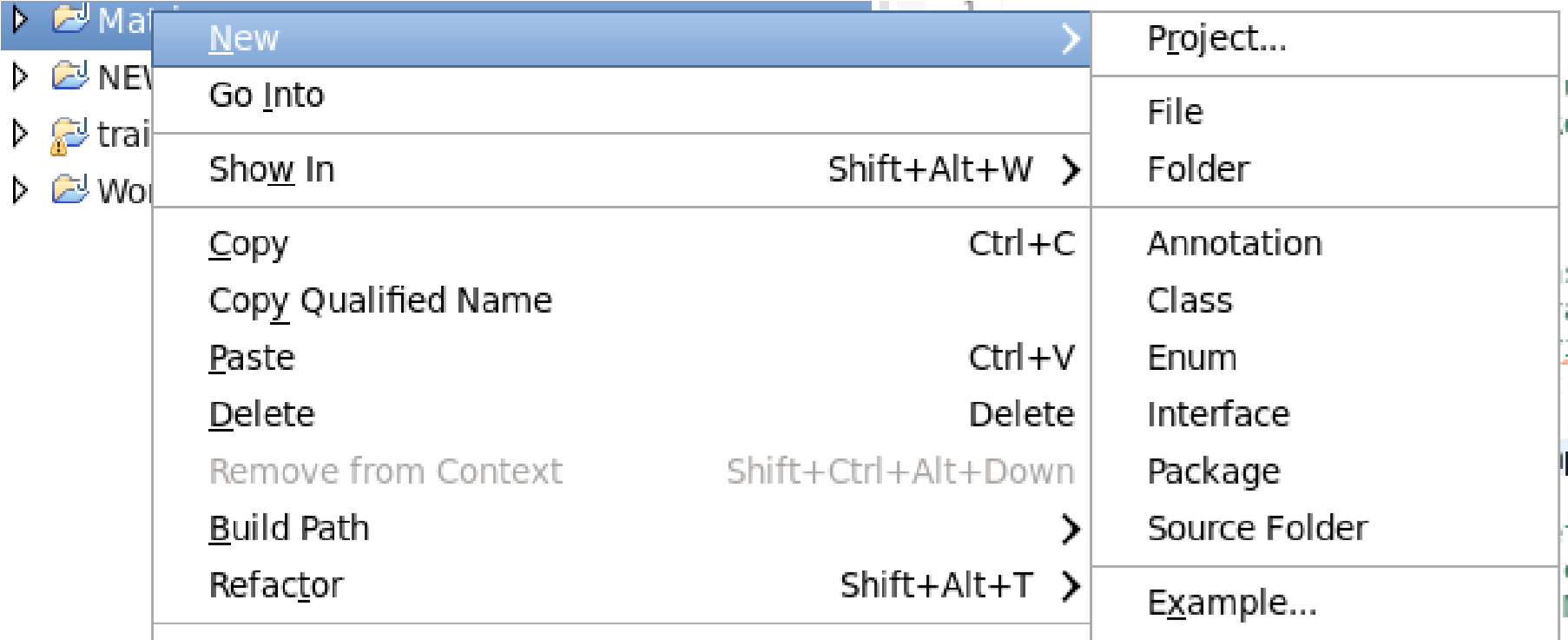


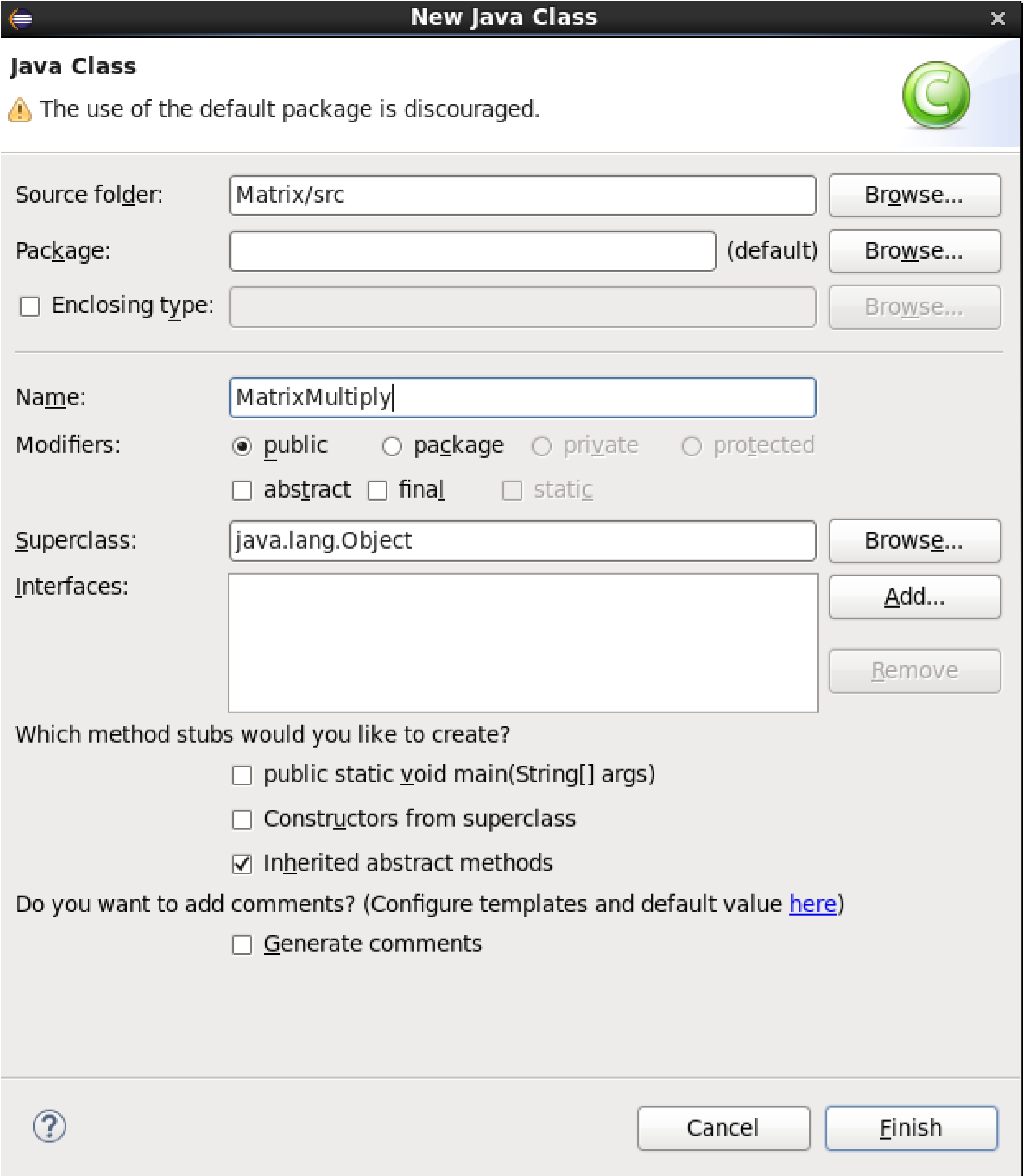






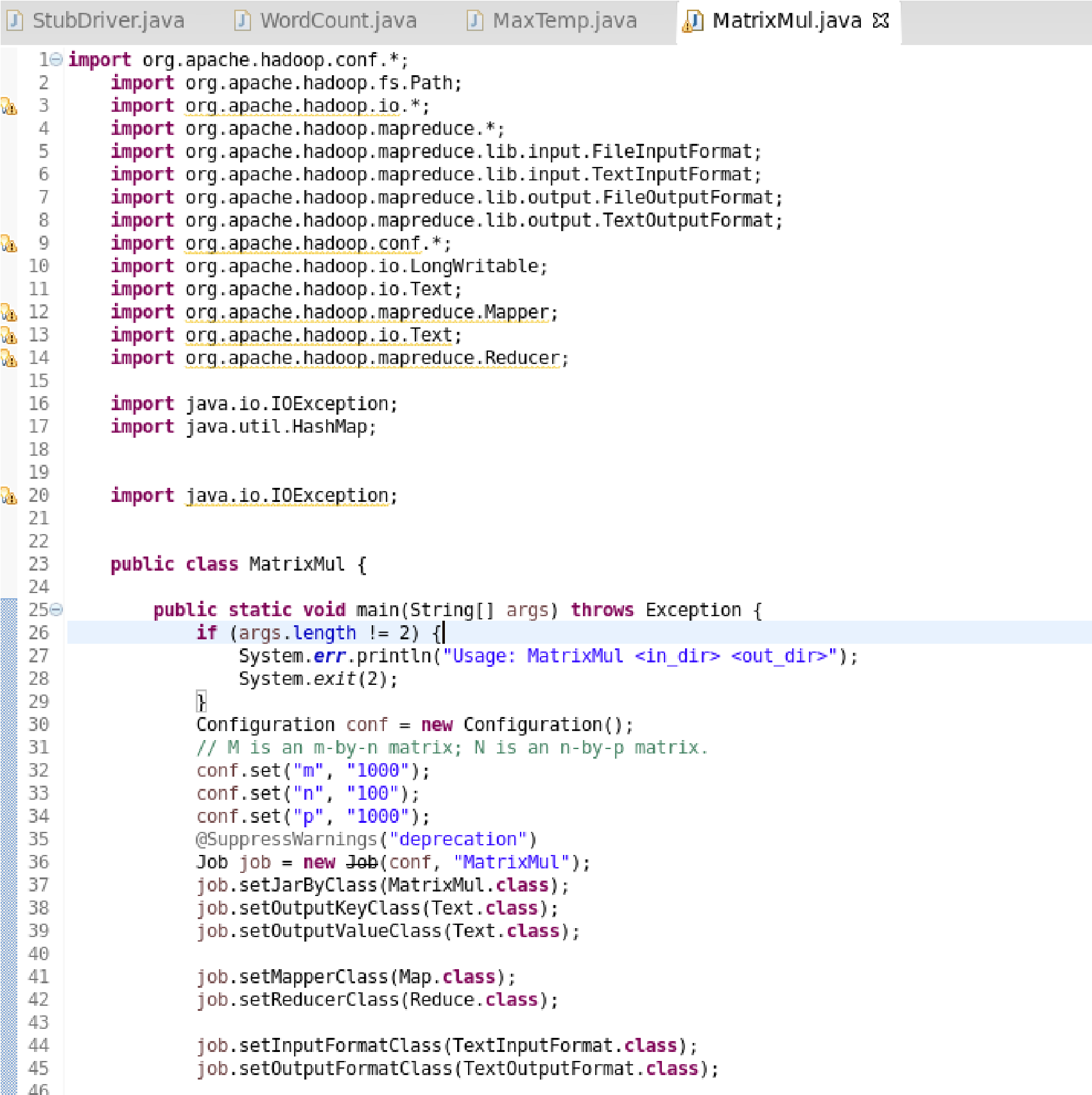
1. Right Click on the name of Project “Matrix” -> New -> class Don’t write anything for package Write Name Textbox write “Matrix” -> Finish Then MatrixMulltiply.java window will pop up

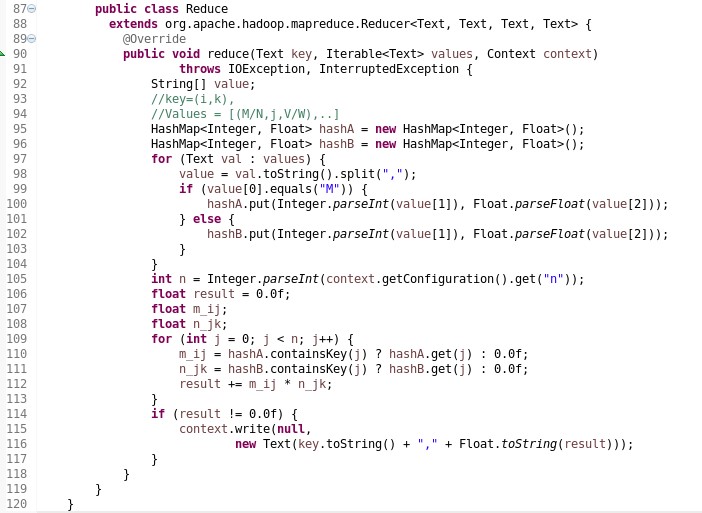


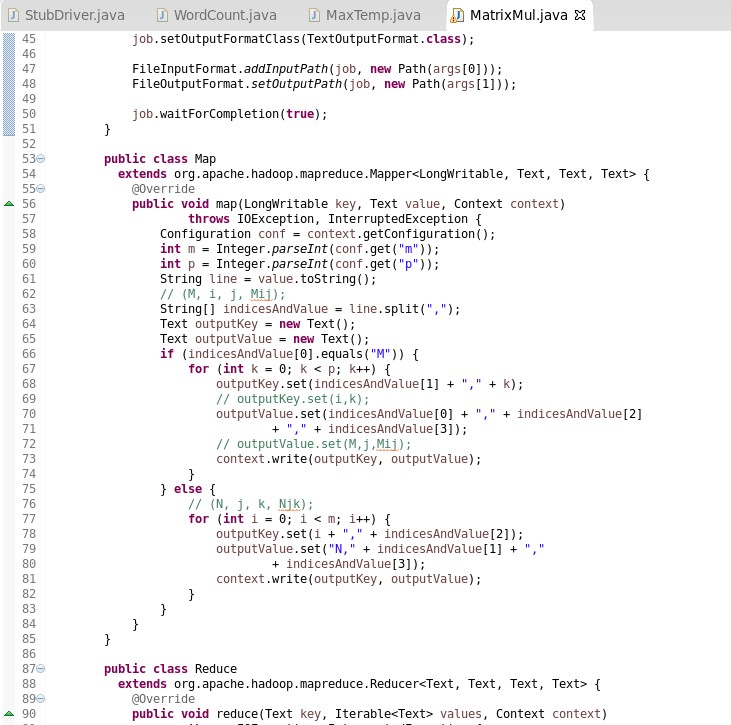


**Source Code:**

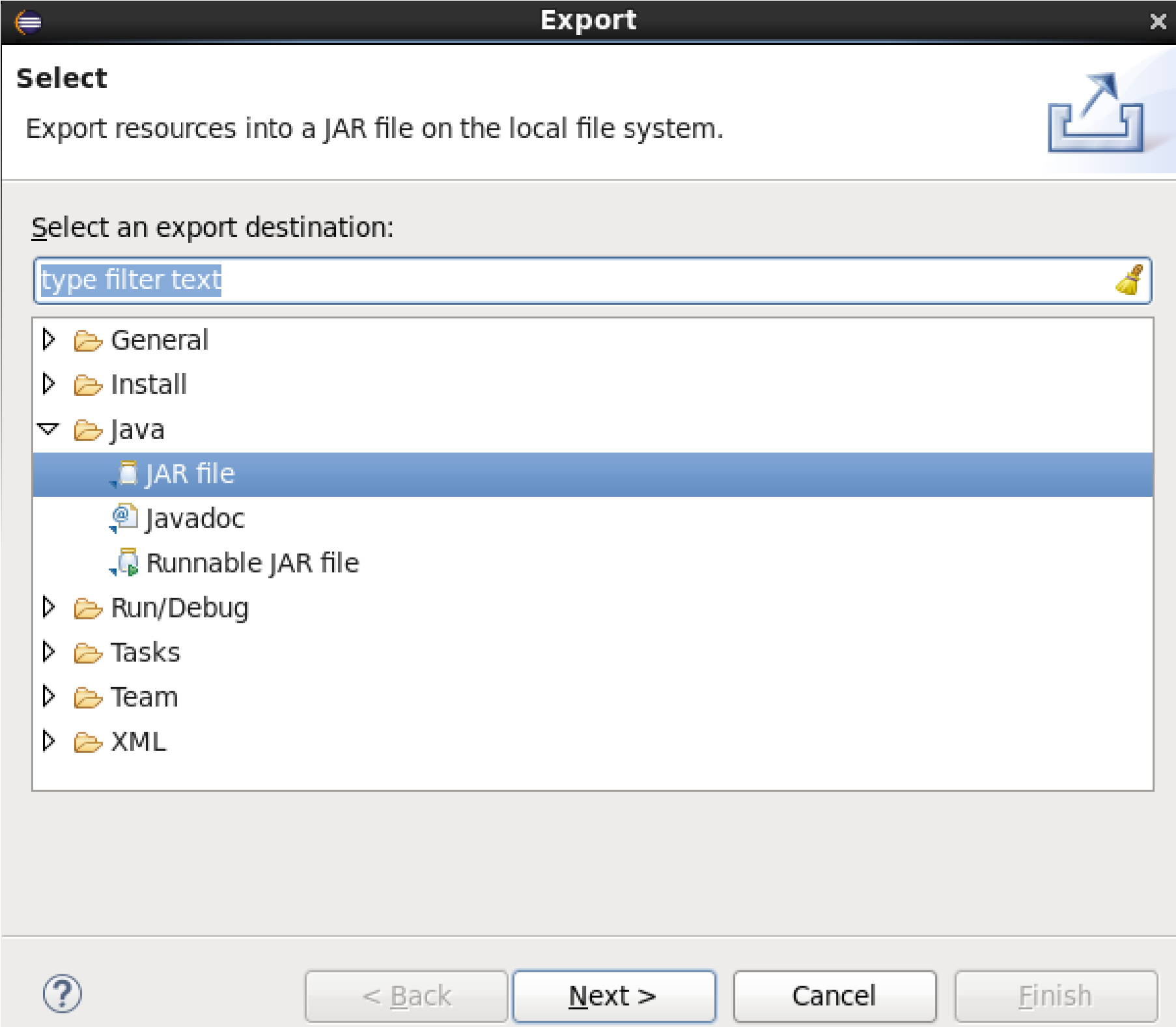
import java.io.IOException;  
import java.util.HashMap;  
  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.\*;  
import org.apache.hadoop.mapreduce.\*;  
import org.apache.hadoop.mapreduce.Mapper.Context;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
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.conf.\*;  
import [org.apache.hadoop.io](http://org.apache.hadoop.io/" \t "_blank).LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
  
import java.io.IOException;  
import java.util.HashMap;  
  
        public class MatrixMultiply {  
  
                public static void main(String[] args) throws Exception {  
                if (args.length != 2) {  
                    System.err.println("Usage: MatrixMultiply <in\_dir> <out\_dir>");  
                    System.exit(2);  
                }  
                Configuration conf = new Configuration();  
                // M is an m-by-n matrix; N is an n-by-p matrix.  
                conf.set("m", "1000");  
                conf.set("n", "100");  
                conf.set("p", "1000");  
                @SuppressWarnings("deprecation")  
                        Job job = new Job(conf, "MatrixMultiply");  
                job.setJarByClass(MatrixMultiply.class);  
                job.setOutputKeyClass(Text.class);  
                job.setOutputValueClass(Text.class);  
  
                job.setMapperClass(Map.class);  
                job.setReducerClass(Reduce.class);  
  
                job.setInputFormatClass(TextInputFormat.class);  
                job.setOutputFormatClass(TextOutputFormat.class);  
  
                FileInputFormat.addInputPath(job, new Path(args[0]));  
                FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
                job.waitForCompletion(true);  
                }  
                public class Map  
                  extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text,  
Text, Text> {  
                        @Override  
                        public void map(LongWritable key, Text value, Context context)  
                                        throws IOException, InterruptedException {  
                                Configuration conf = context.getConfiguration();  
                                int m = Integer.parseInt(conf.get("m"));  
                                int p = Integer.parseInt(conf.get("p"));  
                                String line = value.toString();  
                                // (M, i, j, Mij);  
                                String[] indicesAndValue = line.split(",");  
                                Text outputKey = new Text();  
                                Text outputValue = new Text();  
                                if (indicesAndValue[0].equals("M")) {  
                                        for (int k = 0; k < p; k++) {  
                                                outputKey.set(indicesAndValue[1] + "," + k);  
                                                // outputKey.set(i,k);  
                                                outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]  
                                                                + "," + indicesAndValue[3]);  
                                                // outputValue.set(M,j,Mij);  
                                                context.write(outputKey, outputValue);  
                                        }  
                                } else {  
                                        // (N, j, k, Njk);  
                                        for (int i = 0; i < m; i++) {  
                                                outputKey.set(i + "," + indicesAndValue[2]);  
                                                outputValue.set("N," + indicesAndValue[1] + ","  
                                                                + indicesAndValue[3]);  
                                                context.write(outputKey, outputValue);  
                                        }  
                                }  
                        }  
                }  
                        public class Reduce  
                  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {  
                        @Override  
                        public void reduce(Text key, Iterable<Text> values, Context context)  
                                        throws IOException, InterruptedException {  
                                String[] value;  
                                //key=(i,k),  
                                //Values = [(M/N,j,V/W),..]  
                                HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();  
                                HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();  
                                for (Text val : values) {  
                                        value = val.toString().split(",");  
                                        if (value[0].equals("M")) {  
                                                hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));  
                                        } else {  
                                                hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));  
                                        }  
                                }  
                                int n = Integer.parseInt(context.getConfiguration().get("n"));  
                                float result = 0.0f;  
                                float m\_ij;  
                                float n\_jk;  
                                for (int j = 0; j < n; j++) {  
                                        m\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;  
                                        n\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;  
                                        result += m\_ij \* n\_jk;  
                                }  
                                if (result != 0.0f) {  
                                        context.write(null,  
                                                        new Text(key.toString() + "," + Float.toString(result)));  
                                }  
                        }  
                }  
                }

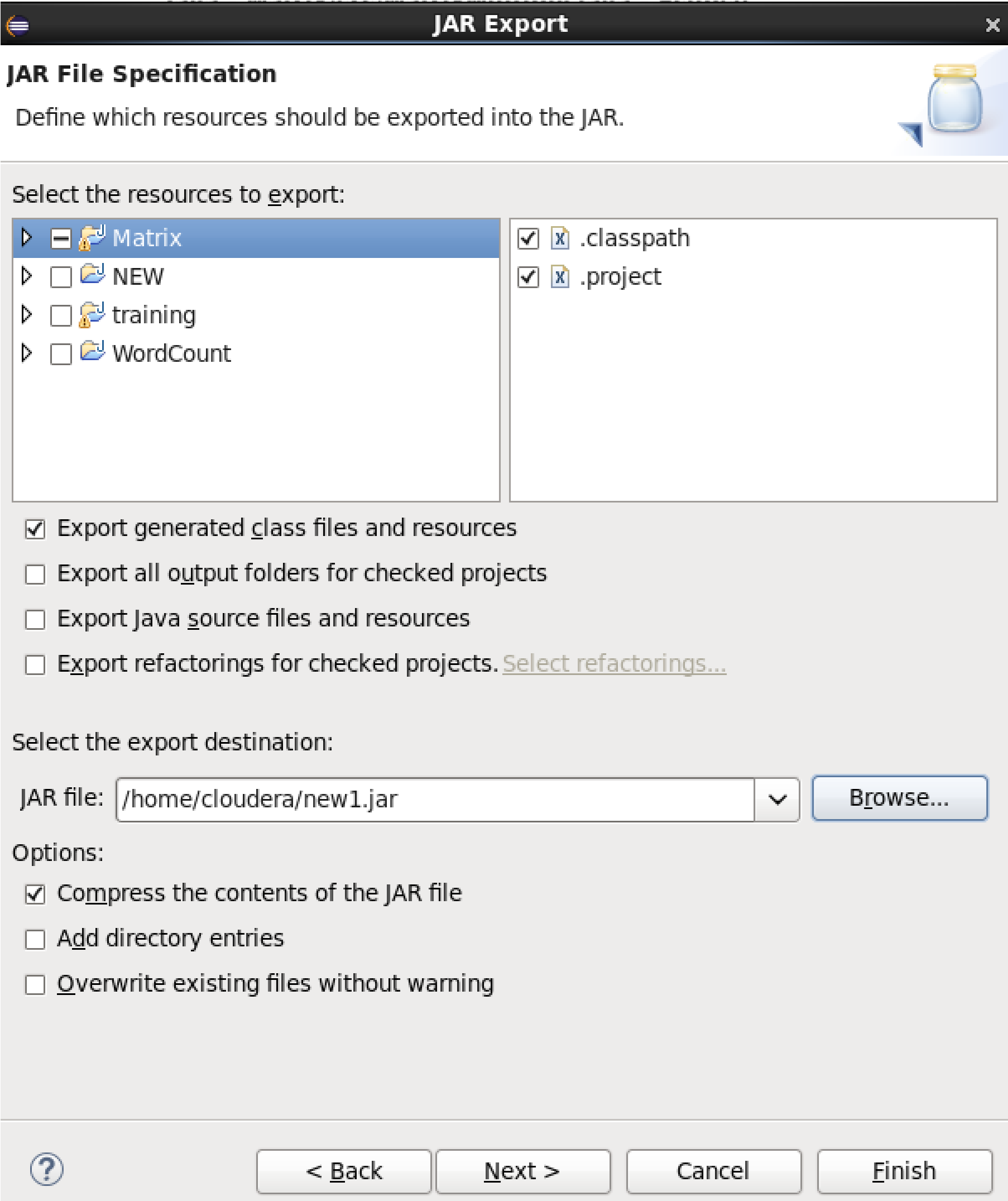


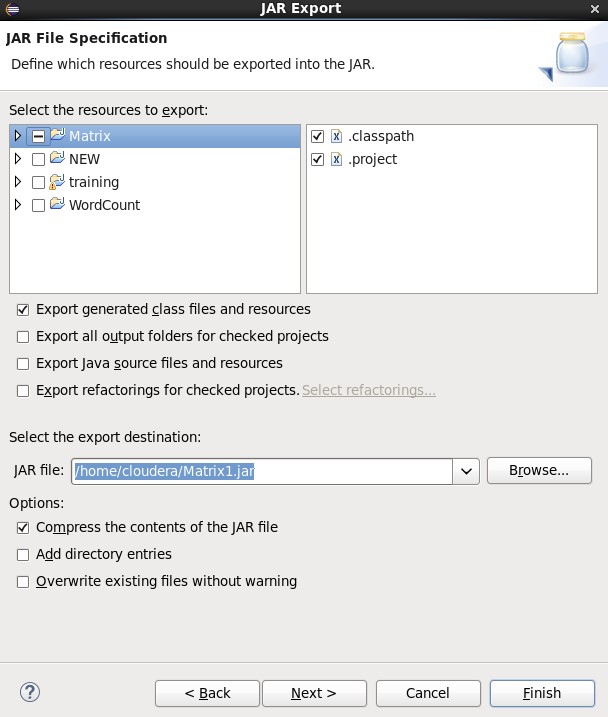
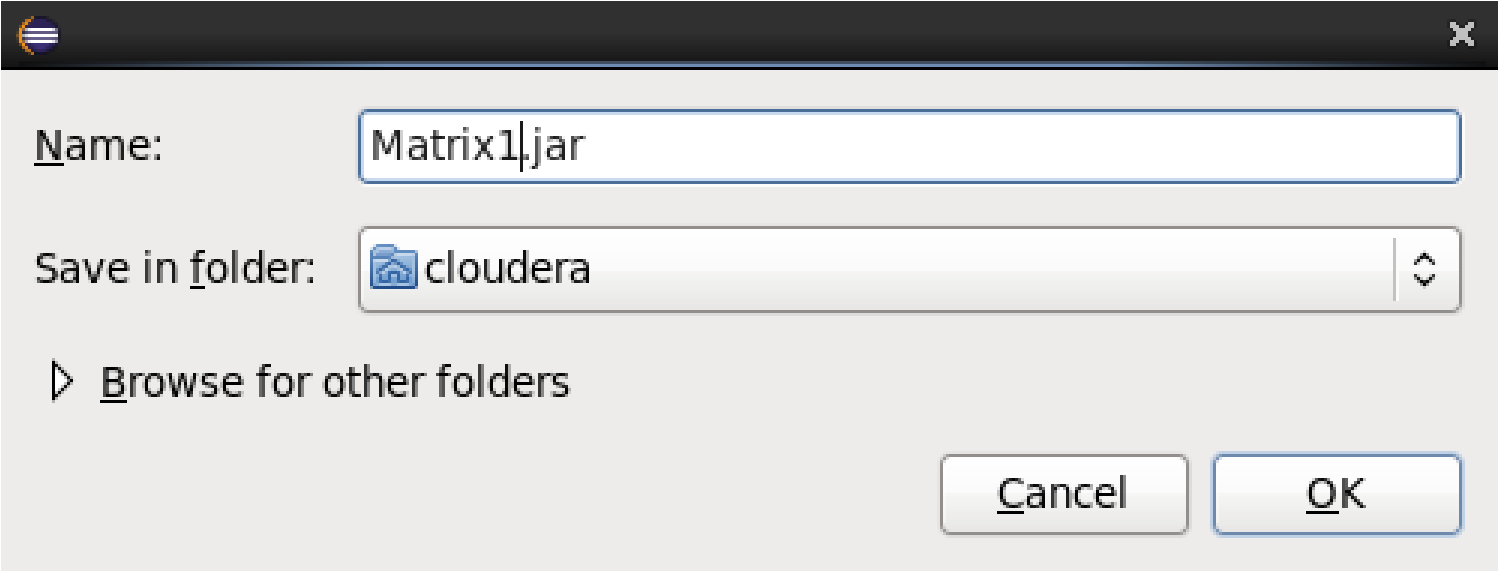




6) Right Click on the project name Matrix -> Export -> Java -> JAR File -> Next -> for select the export destination for JAR file: browse -> Name : Matrix1.jar -> save in folder -> cloudera -> Finish -> OK

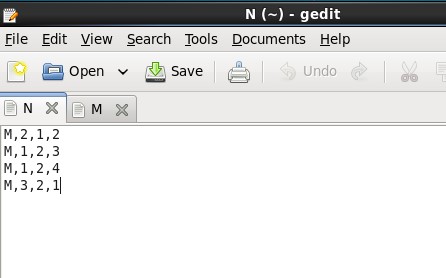






7) We need to create an input file in local file system

Creating an input file named as “M” and “N”

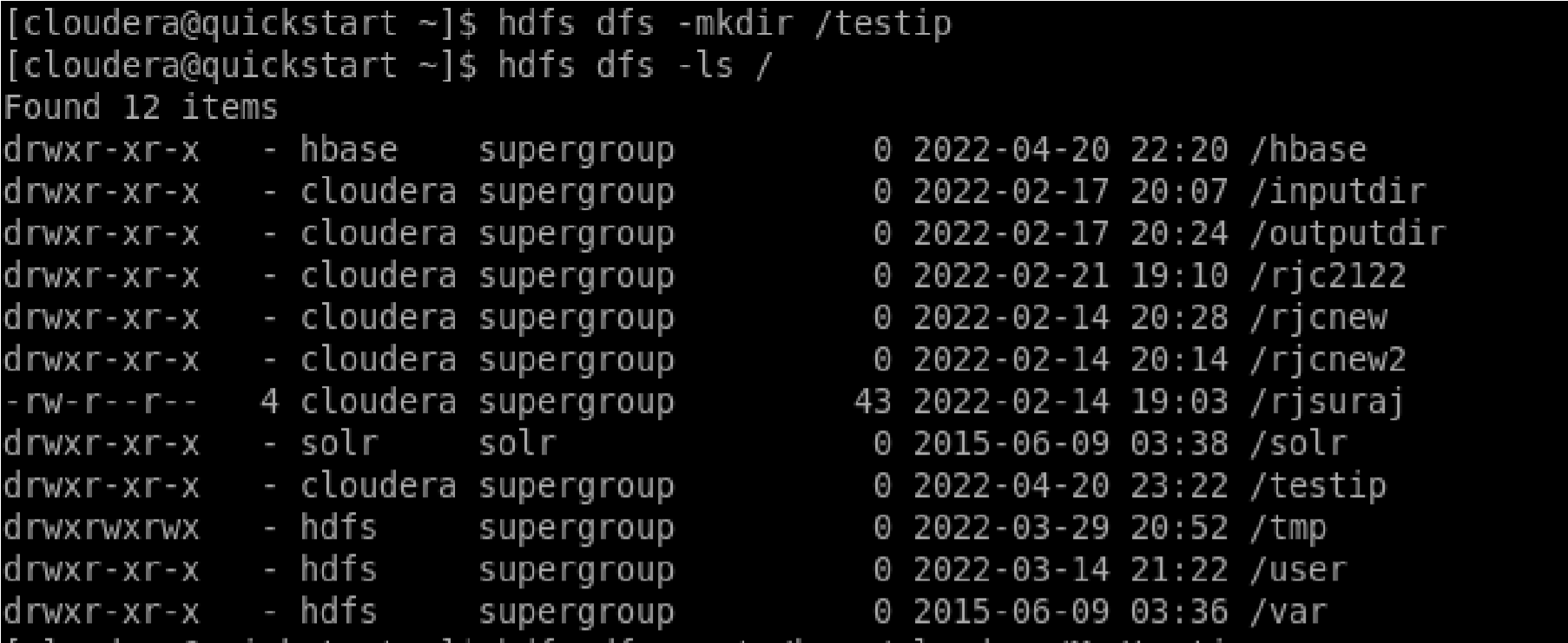


8) Open Terminal

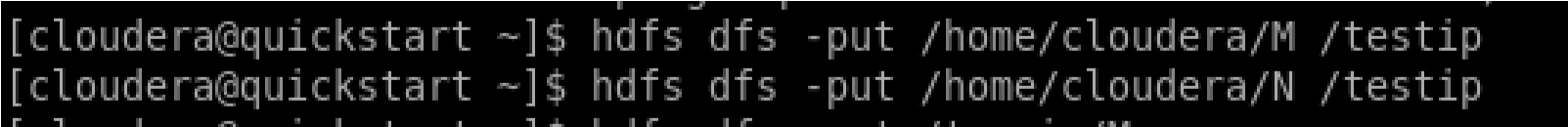
Now we have to move this input file to **hdfs**. For this we create a direcory on **hdfs** using command **hdfs dfs -mkdir /testip**.

1. Here listing all the directory present in **hdfs** using **hdfs dfs -ls /** command

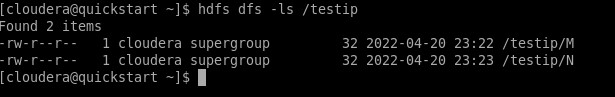
Then we can verify whether this directory is created or not using ls command **hdfs dfs -ls /**



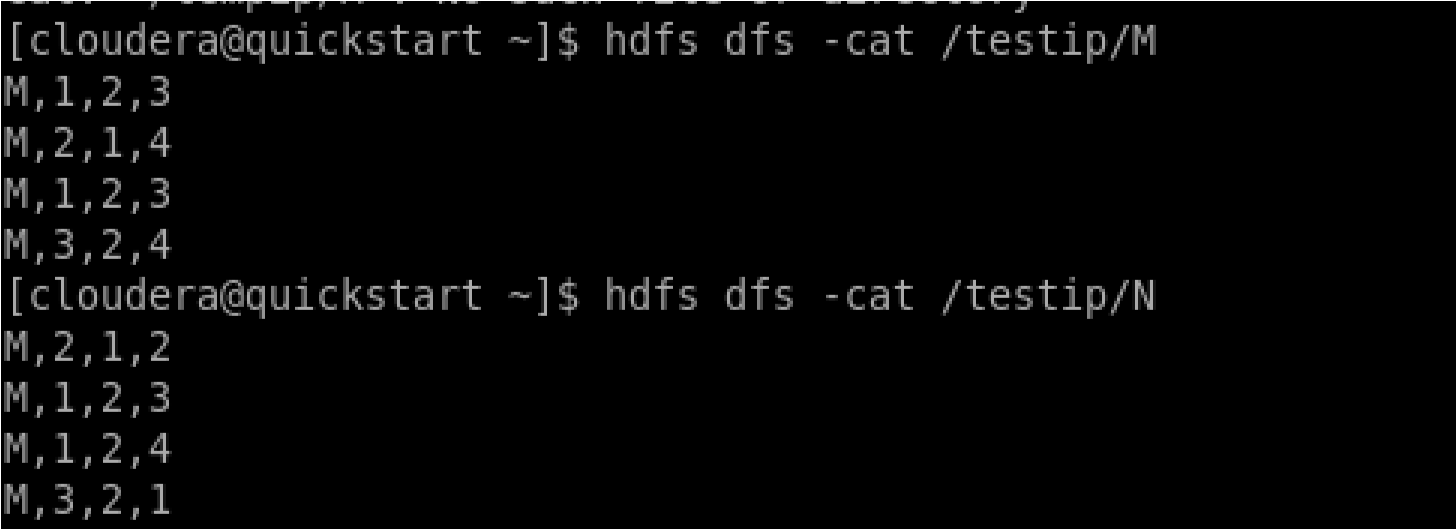
10) Move the input file to this directory created in **hdfs** by using either put command or **copyFromLocal** command.



11) Now checking whether the “M” and “N” present in **/testip** directory of **hdfs** or not using **hdfs dfs -ls /testip** command.

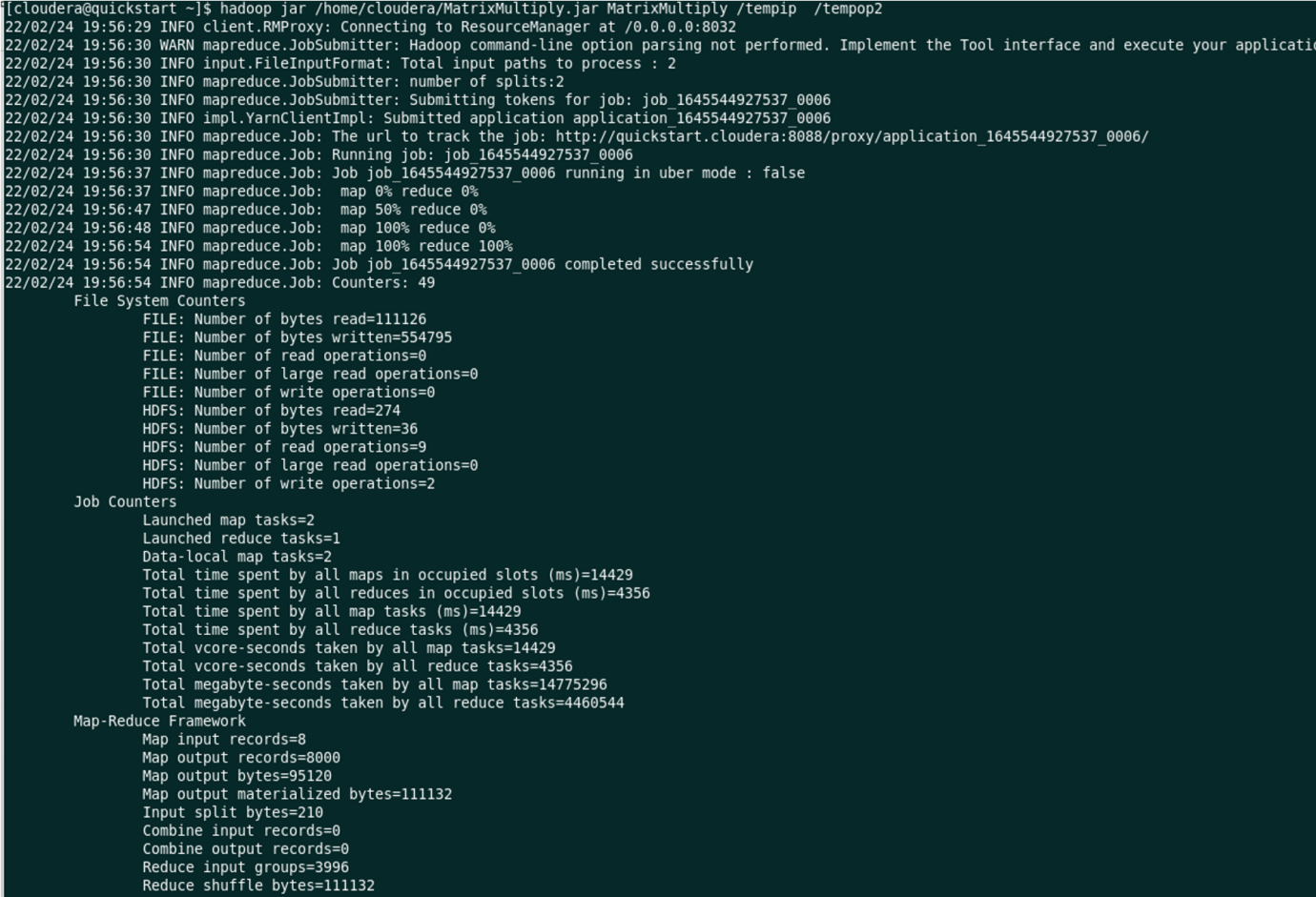


12) As we can see “M” and “N” file is present in /testip directory of hdfs. Now we will see the content of this file using **hdfs dfs –cat /testip/M** command

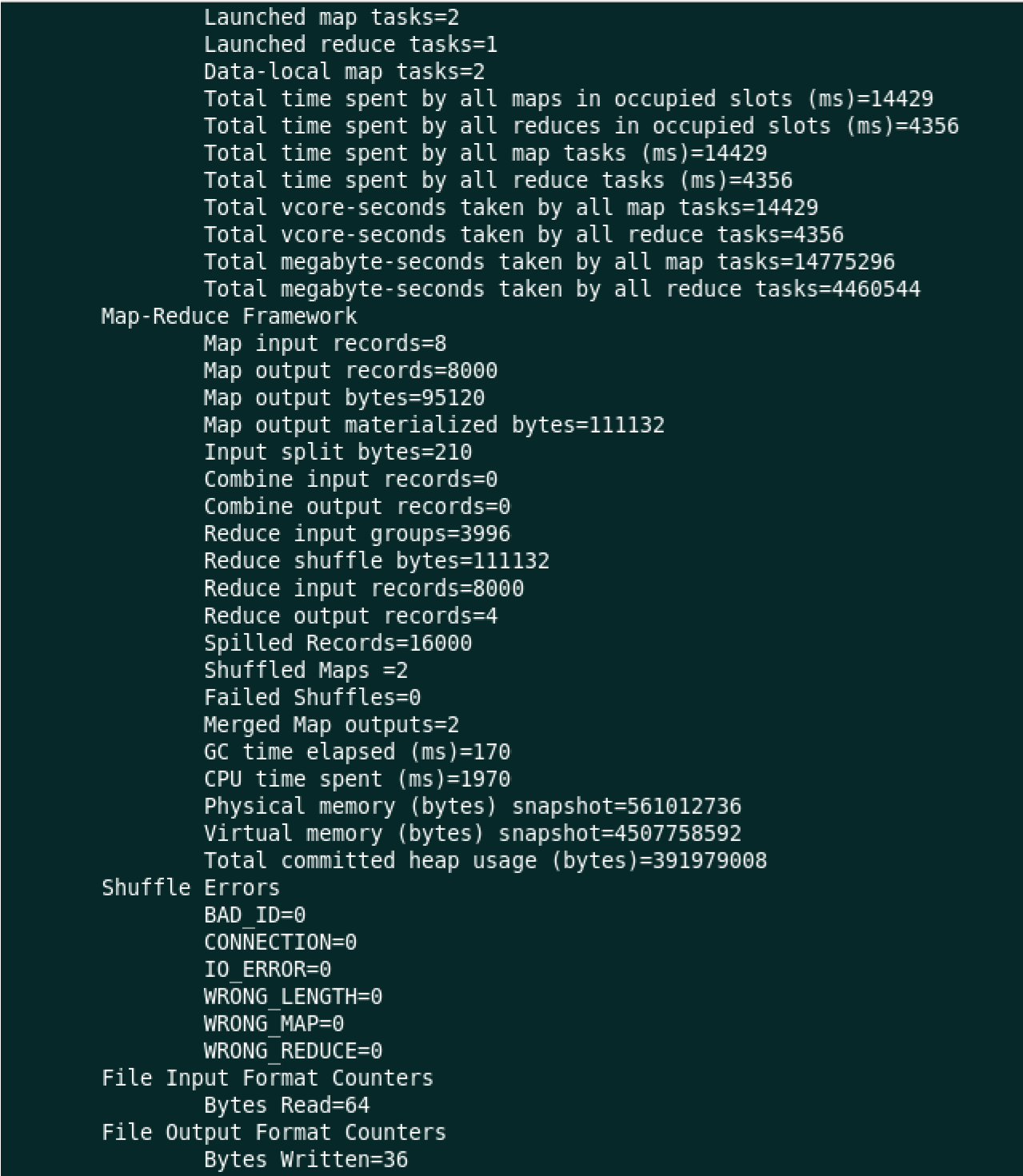


13) Running Mapreduce Program on Hadoop, syntax is hadoop jar jarFileName.jar ClassName/InputFileAddress /outputdir.

i.e. **hadoop jar /home/cloudera/Matrix1.jar MatrixMultiply /testip/\* /testip\_out**



**Map Reduce Framework**



14) Now we can check what we have inside this **tempop2** directory using command as **hdfs dfs -ls /tempop2.**

