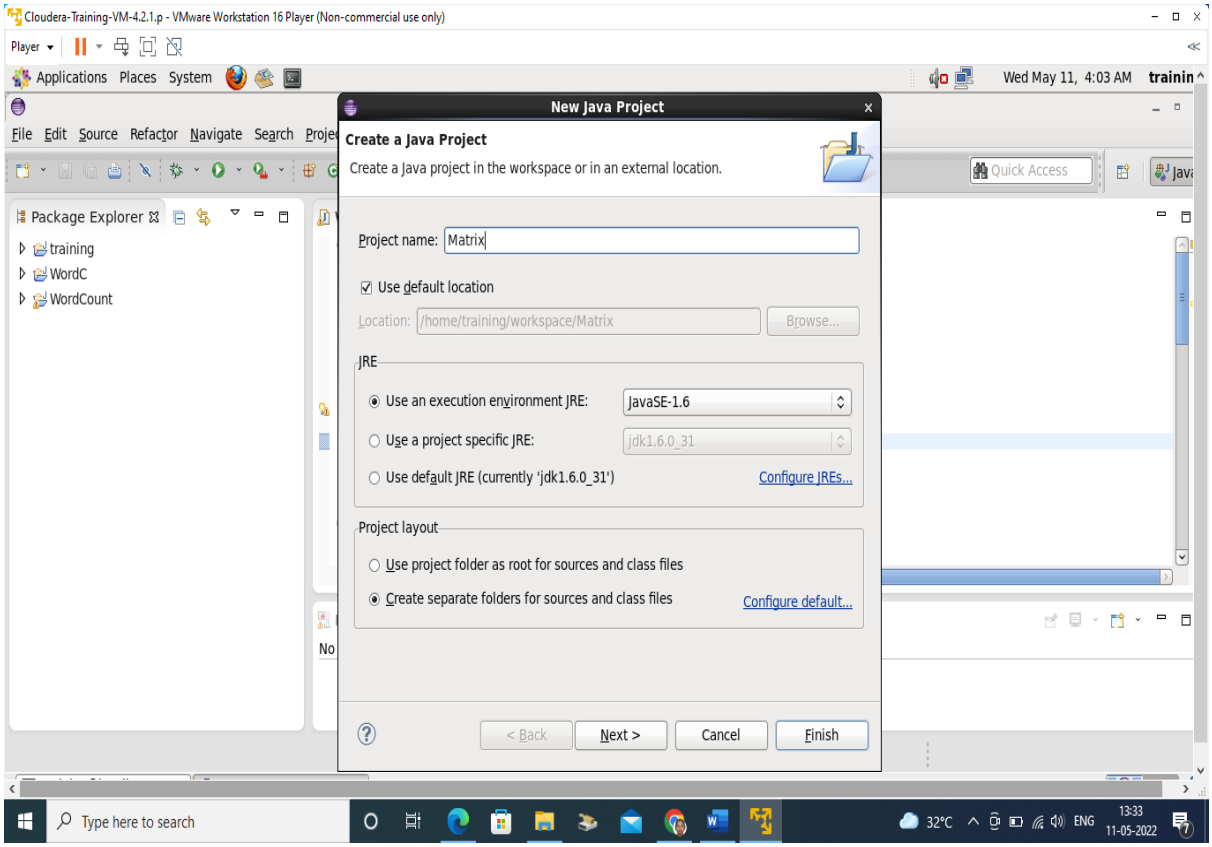
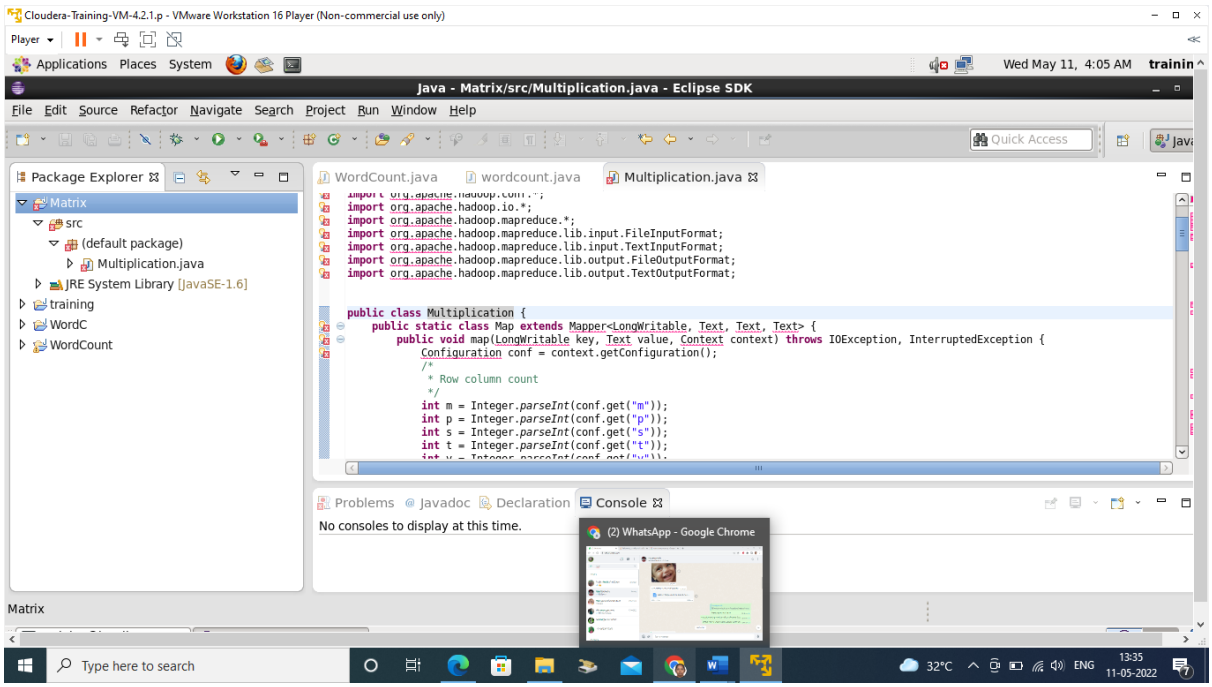


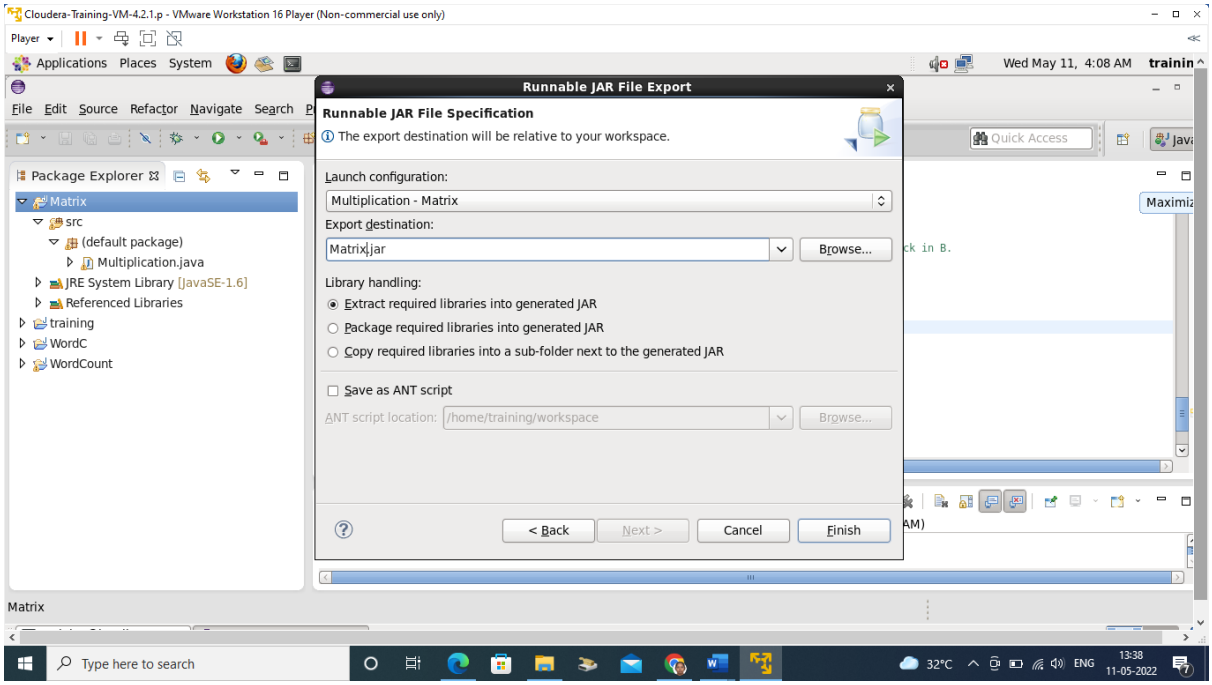

```
import java.io.IOException;
import java.util.*;
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
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;

public class MatrixMultiplication {

    public static class Map extends Mapper<LongWritable, Text, Text, Text> {
        public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
            Configuration conf = context.getConfiguration();
            /*
             * Row column count
             */
            int m = Integer.parseInt(conf.get("m"));
            int p = Integer.parseInt(conf.get("p"));
            int s = Integer.parseInt(conf.get("s"));
```

```
        int t = Integer.parseInt(conf.get("t"));
        int v = Integer.parseInt(conf.get("v"));

        int mPerS = m/s; // Number of blocks in each column of A.
        int pPerV = p/v; // Number of blocks in each row of B.

        String line = value.toString();
        String[] indicesAndValue = line.split(",");
        Text outputKey = new Text();
        Text outputValue = new Text();

        if (indicesAndValue[0].equals("A")) {
           int i = Integer.parseInt(indicesAndValue[1]);
           int j = Integer.parseInt(indicesAndValue[2]);
           for (int kPerV = 0; kPerV < pPerV; kPerV++) {
              outputKey.set(Integer.toString(i/s) + "," + Integer.toString(j/t) + "," +
Integer.toString(kPerV));
              outputValue.set("A," + Integer.toString(i%s) + "," + Integer.toString(j%t) + "," +
indicesAndValue[3]);
              context.write(outputKey, outputValue);
           }
        } else {
           int j = Integer.parseInt(indicesAndValue[1]);
           int k = Integer.parseInt(indicesAndValue[2]);
           for (int iPerS = 0; iPerS < mPerS; iPerS++) {
              outputKey.set(Integer.toString(iPerS) + "," + Integer.toString(j/t) + "," +
Integer.toString(k/v));
              outputValue.set("B," + Integer.toString(j%t) + "," + Integer.toString(k%v) + ","
+ indicesAndValue[3]);
              context.write(outputKey, outputValue);
           }
        }
     }
  }

  public static class Reduce extends Reducer<Text, Text, Text, Text> {
     public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
        String[] value;
        ArrayList<Entry<String, Float>> listA = new ArrayList<Entry<String, Float>>();
        ArrayList<Entry<String, Float>> listB = new ArrayList<Entry<String, Float>>();
        for (Text val : values) {
           value = val.toString().split(",");
           if (value[0].equals("A")) {
              listA.add(new SimpleEntry<String, Float>(value[1] + "," + value[2],
Float.parseFloat(value[3])));
           } else {
              listB.add(new SimpleEntry<String, Float>(value[1] + "," + value[2],
Float.parseFloat(value[3])));
           }
```

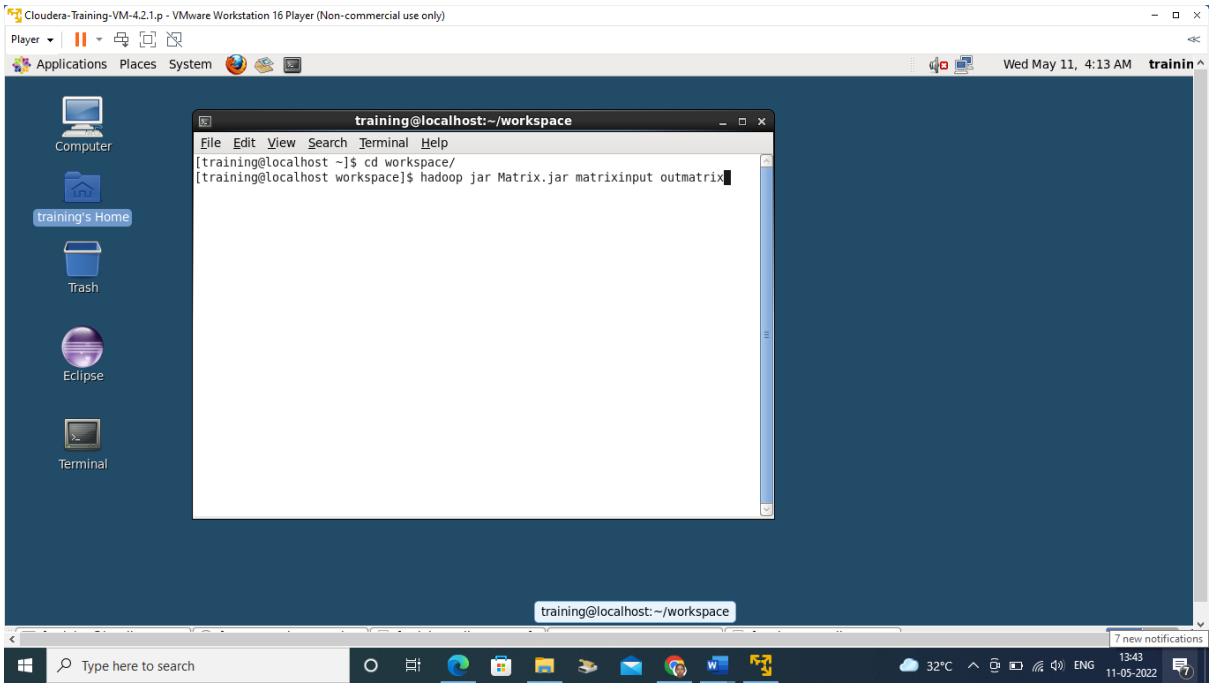
```
        }
        String[] iModSAndJModT;
        String[] jModTAndKModV;
        float a_ij;
        float b_jk;
        String hashKey;
        HashMap<String, Float> hash = new HashMap<String, Float>();
        for (Entry<String, Float> a : listA) {
           iModSAndJModT = a.getKey().split(",");
           a_ij = a.getValue();
           for (Entry<String, Float> b : listB) {
              jModTAndKModV = b.getKey().split(",");
              b_jk = b.getValue();
              if (iModSAndJModT[1].equals(jModTAndKModV[0])) {
                 hashKey = iModSAndJModT[0] + "," + jModTAndKModV[1];
                 if (hash.containsKey(hashKey)) {
                    hash.put(hashKey, hash.get(hashKey) + a_ij*b_jk);
                 } else {
                    hash.put(hashKey, a_ij*b_jk);
                 }
              }
           }
        }
        String[] blockIndices = key.toString().split(",");
        String[] indices;
        String i;
        String k;
        Configuration conf = context.getConfiguration();
        int s = Integer.parseInt(conf.get("s"));
        int v = Integer.parseInt(conf.get("v"));
        Text outputValue = new Text();
        for (Entry<String, Float> entry : hash.entrySet()) {
           indices = entry.getKey().split(",");
           i = Integer.toString(Integer.parseInt(blockIndices[0])*s +
Integer.parseInt(indices[0]));
           k = Integer.toString(Integer.parseInt(blockIndices[2])*v +
Integer.parseInt(indices[1]));
           outputValue.set(i + "," + k + "," + Float.toString(entry.getValue()));
           context.write(null, outputValue);
        }
     }
  }

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     // A is an m-by-n matrix; B is an n-by-p matrix.
     conf.set("m", "2");
     conf.set("n", "5");
     conf.set("p", "3");
     conf.set("s", "2"); // Number of rows in a block in A.
```

```
        conf.set("t", "5"); // Number of columns in a block in A = number of rows in a block in
B.
        conf.set("v", "3"); // Number of columns in a block in B.

        Job job = new Job(conf, "Multiplication");
        job.setJarByClass(MatrixMultiplication.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);
    }
}
```





hadoop jar jarfilename input_file_name output_file_name



matrixinput
A,0,2,2.0
A,0,3,3.0
A,0,4,4.0
B,3,1,10.0
B,3,2,11.0
A,1,0,5.0
A,1,1,6.0

A,1,2,7.0
A,1,3,8.0
A,1,4,9.0
B,0,1,1.0
B,0,2,2.0
B,1,0,3.0
B,1,1,4.0
B,1,2,5.0
B,2,0,6.0
B,2,1,7.0
B,2,2,8.0
B,3,0,9.0
B,4,0,12.0
B,4,1,13.0
B,4,2,14.0

hdfs dfs -copyFromLocal matrixinput /user/training/matrixinput