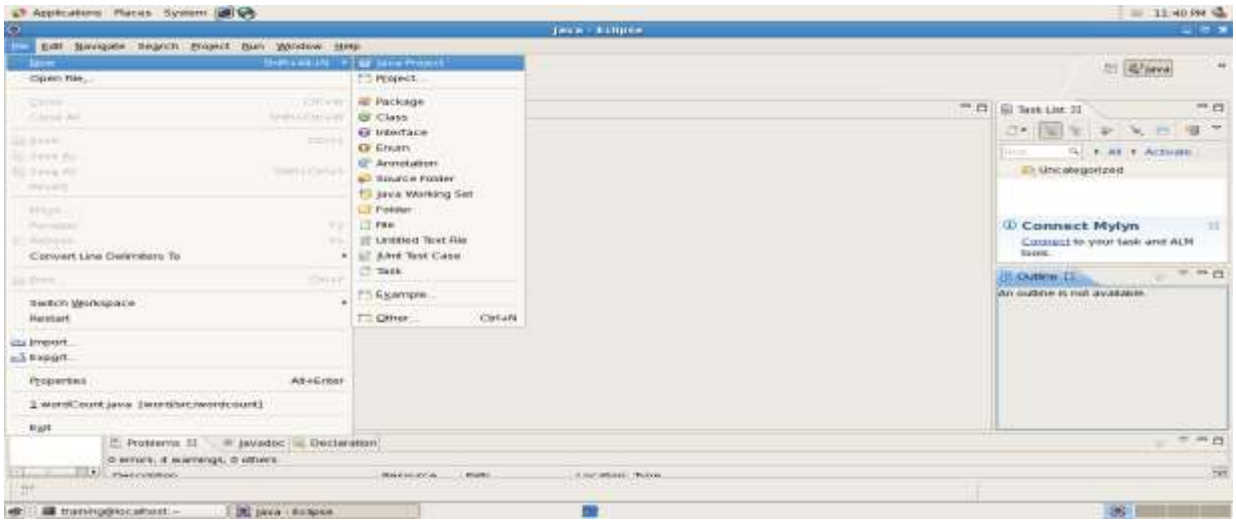
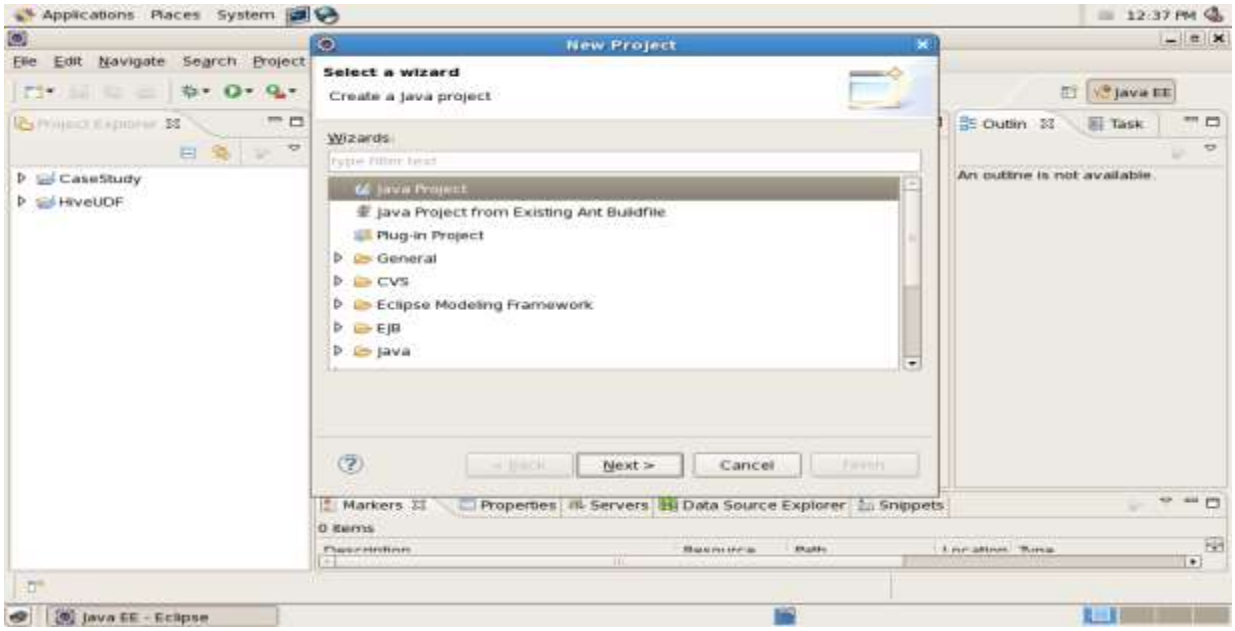
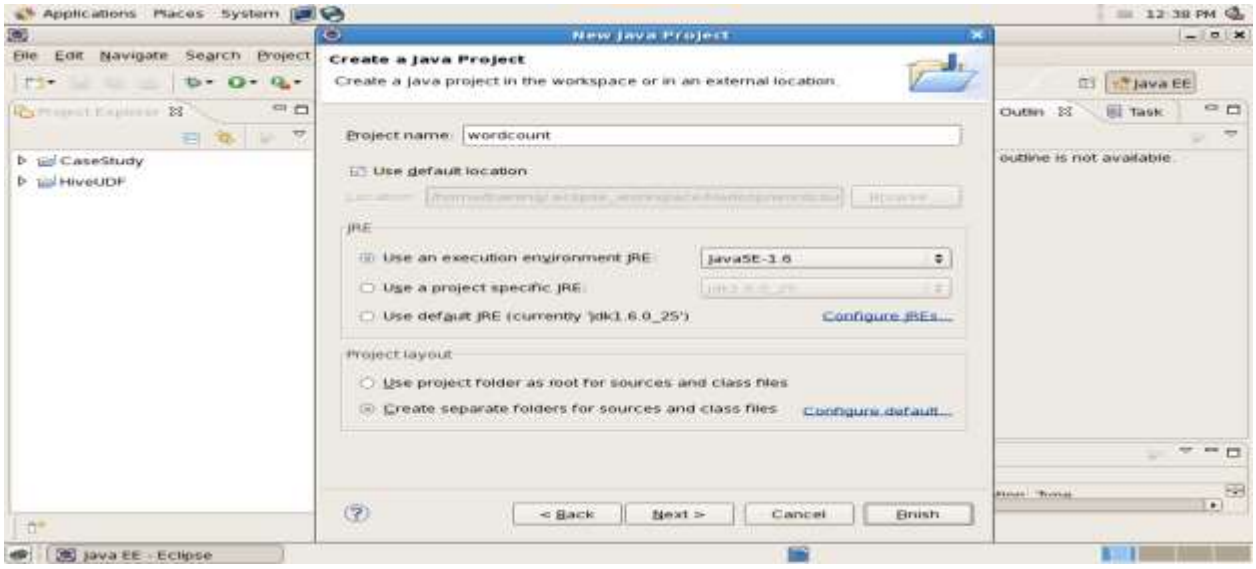
Map-reduce program for Word count

In this practical single node hadoop cluster have been used. The hadoop cluster with pre-installed eclipse on Cent OS is going to used for running Map-reduce program. The steps to run word count program using map-reduce framework are as follows

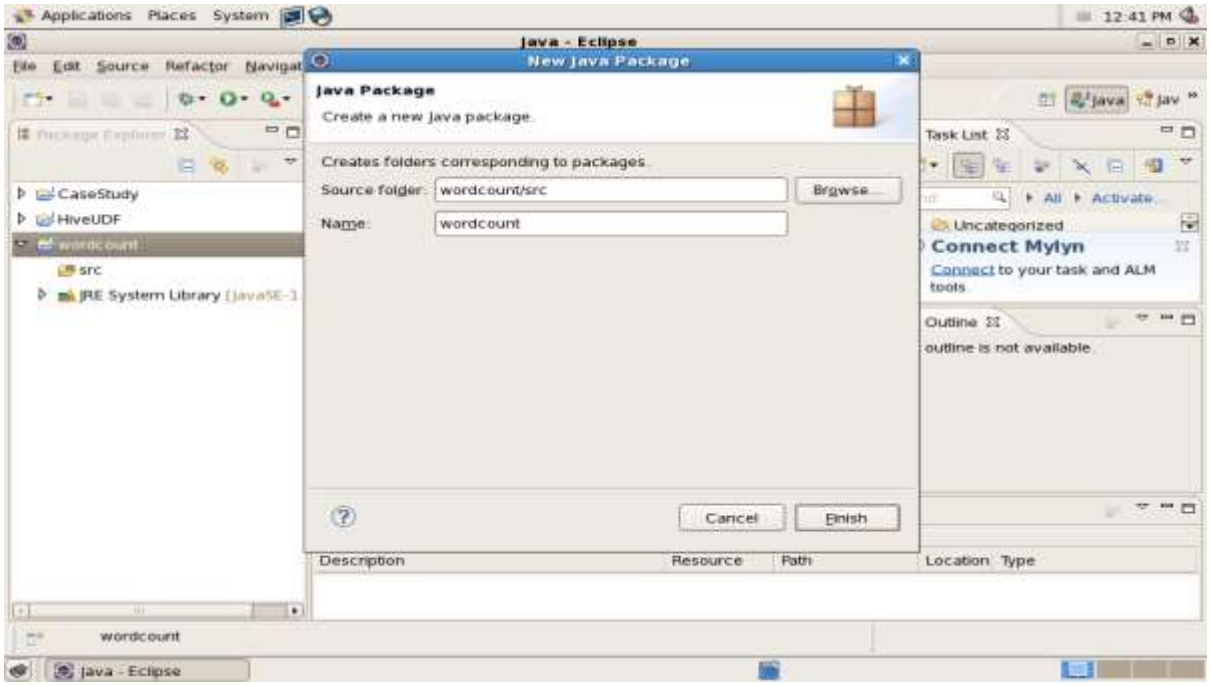
Step 1-: Open Eclipse and create new Java project specify name and click on finish



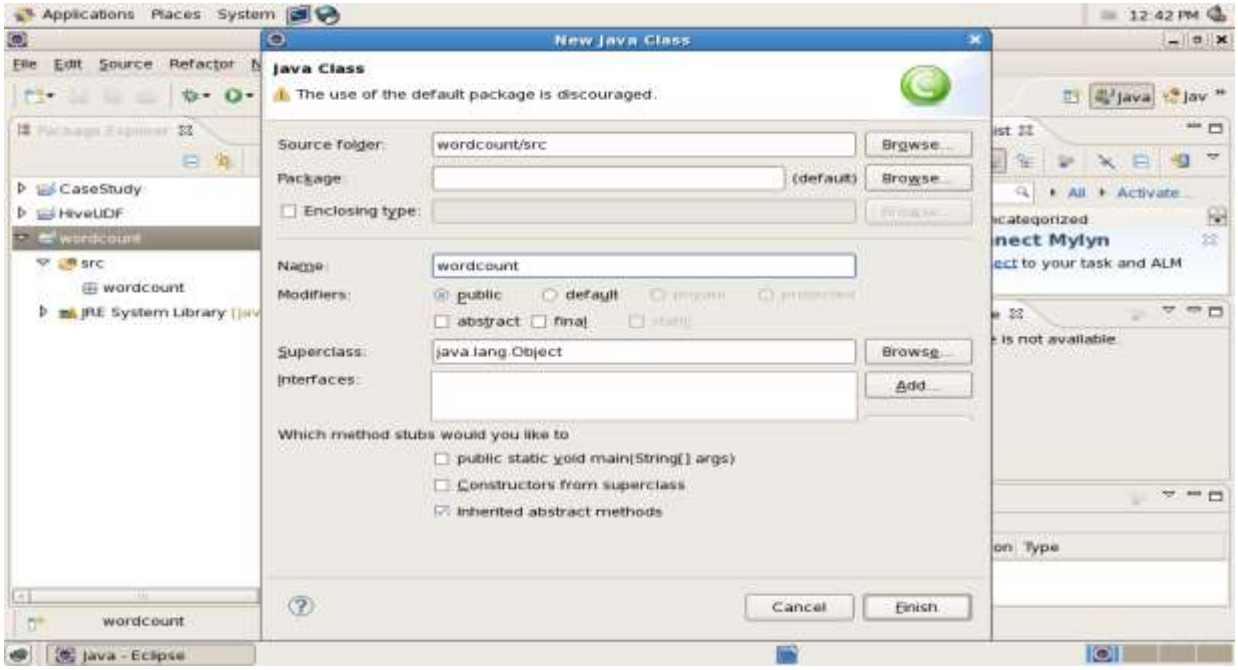




Step 2-: Right click on project and Create new package wordcount



Step 3-:Right click on Package name wordcount and create new class in it and assign name wordcount



Step 4-: Write mapreduce program for wordcount with in that class

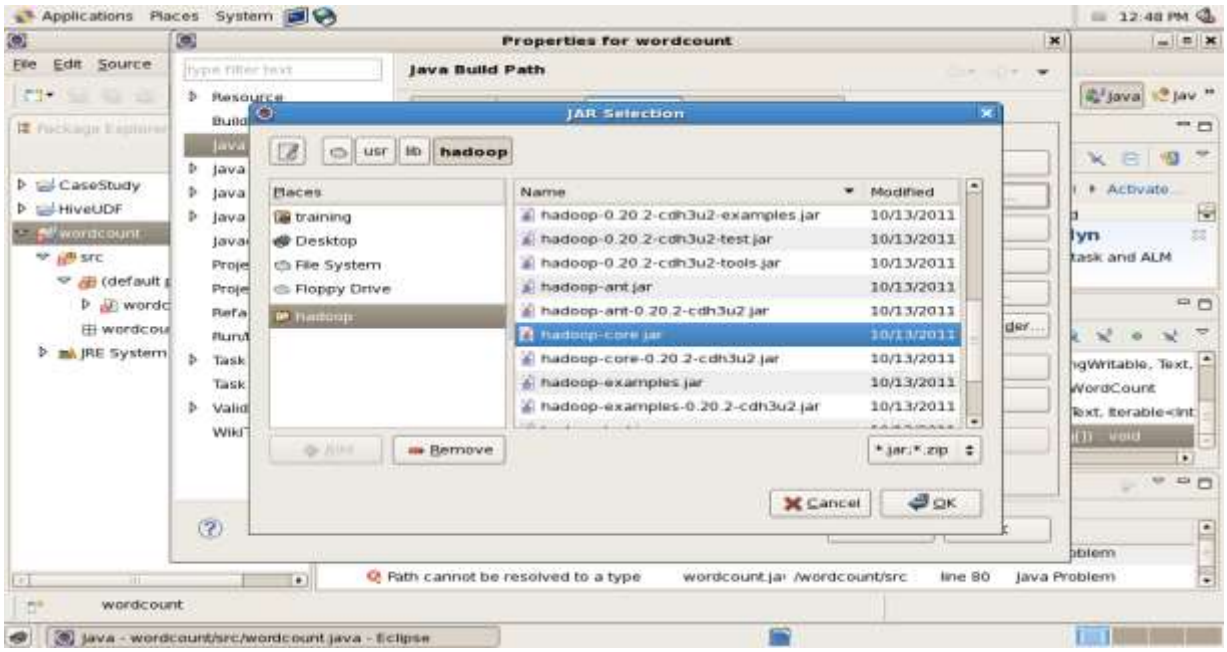
```
package wordcount;

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.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class wordcount
{
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>
{
        public void map(LongWritable key, Text value, Context con) throws IOException,
InterruptedException
            {
                String line = value.toString();

                StringTokenizer token = new StringTokenizer(line);

              while(token.hasMoreTokens())
                {
                        String status = new String();
                        String word = token.nextToken();
                        Text outputKey = new Text(word);
                        IntWritable outputValue = new IntWritable(1);
                        con.write(outputKey, outputValue);
```

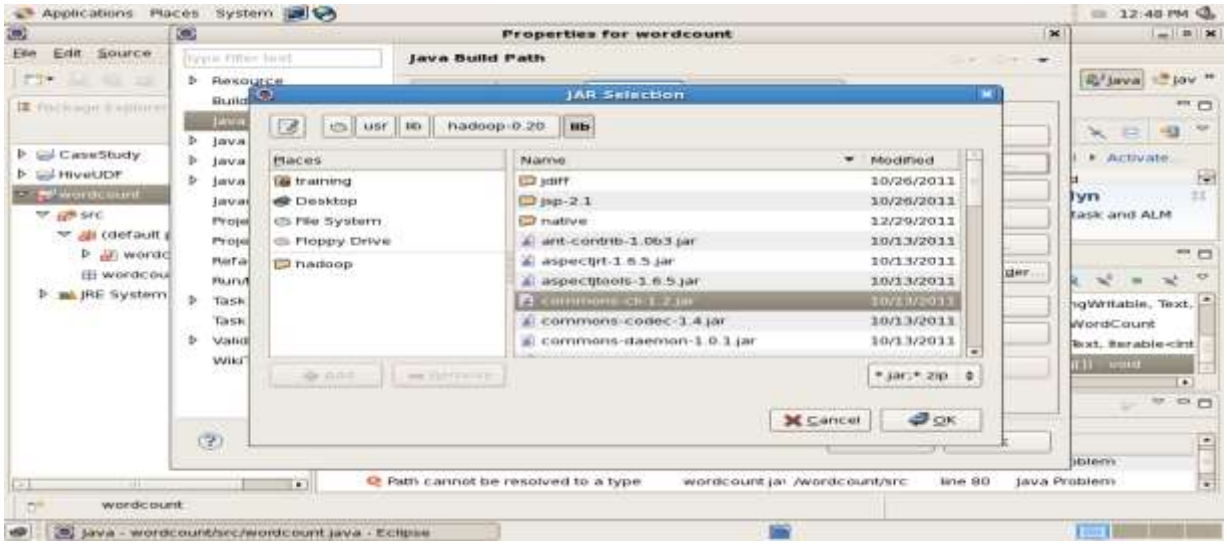
```
                    }
                } // end of map()
} //end of Mapper Class

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
        public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException,
InterruptedException
        {

                int sum = 0;

                for(IntWritable value : values)
                {
                        sum += value.get();
                }

                con.write(word, new IntWritable(sum));

        } // end of reduce()
} // end of Reducer class
/*



*/

// job definition

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

        Configuration c = new Configuration();
        String[] files = new GenericOptionsParser(c, args).getRemainingArgs();
        Path input = new Path(files[0]);
        Path output = new Path(files[1]);
```

```
        Job j = new Job(c, "wordcount");
        j.setJarByClass(wordcount.class);
        j.setMapperClass(MapForWordCount.class);
        j.setReducerClass(ReduceForWordCount.class);
        j.setOutputKeyClass(Text.class);
        j.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(j, input);
        FileOutputFormat.setOutputPath(j, output);
        System.exit(j.waitForCompletion(true) ? 0:1);

} // end of main()

} //end of main class
```

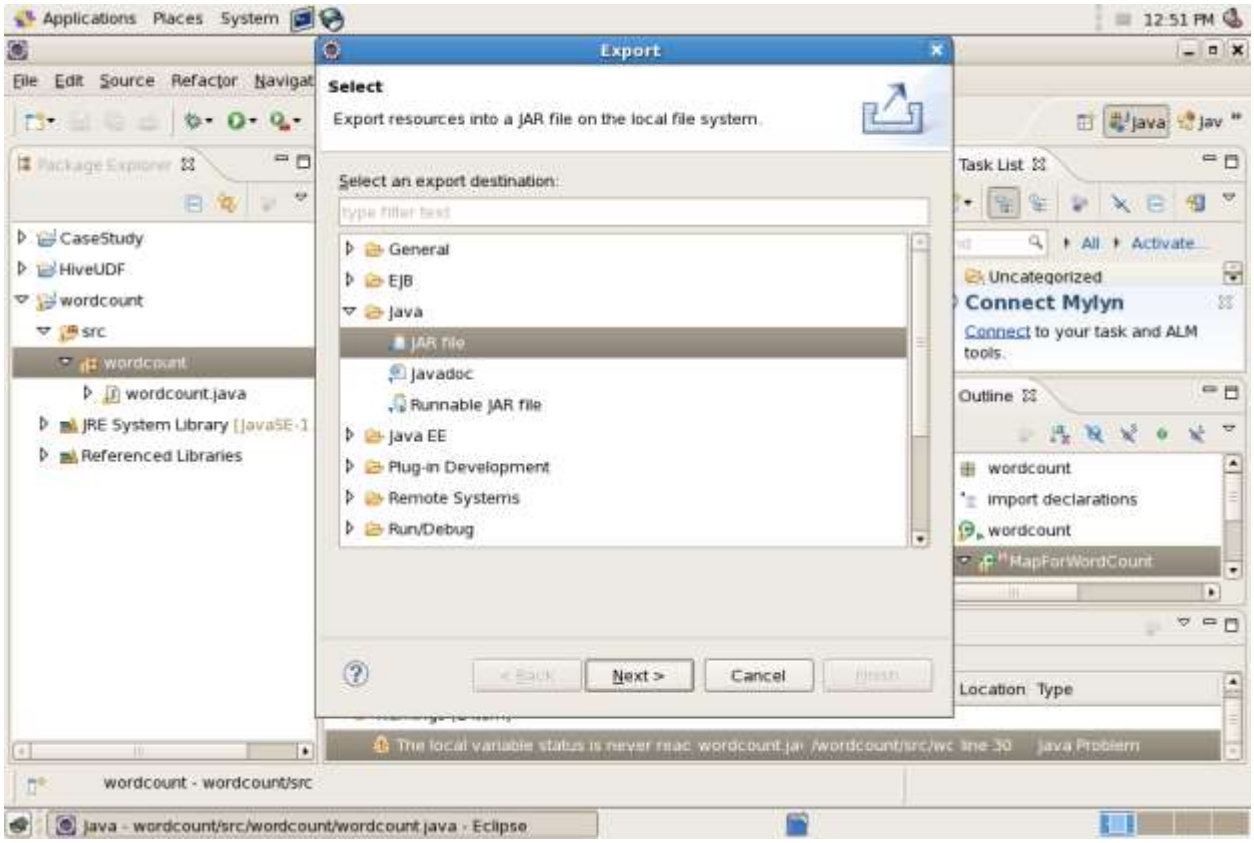
Step 5-: Add required jar files to resolve errors

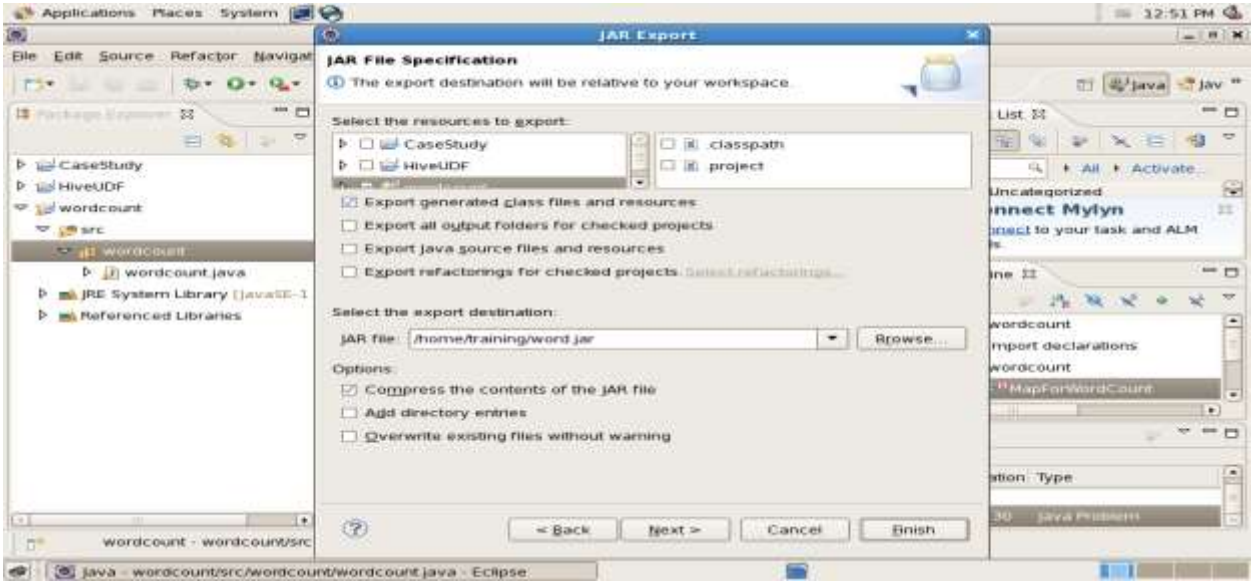
To add jar files right click on class file then select build path option then open configure build path window. To add essential libraries click on add external jars butoon and add three jar files one by one .Here we need three jar files namely hadoop-core.jar,common-cli-1.2.jar and core-3.1.1.jar





Step 6 -: Once all the errors have been resolved then right click on project and select export jar files,specify name to it and click on finish.





Step 7-: Create input text file and copy both input and jar files it to hadoop directory

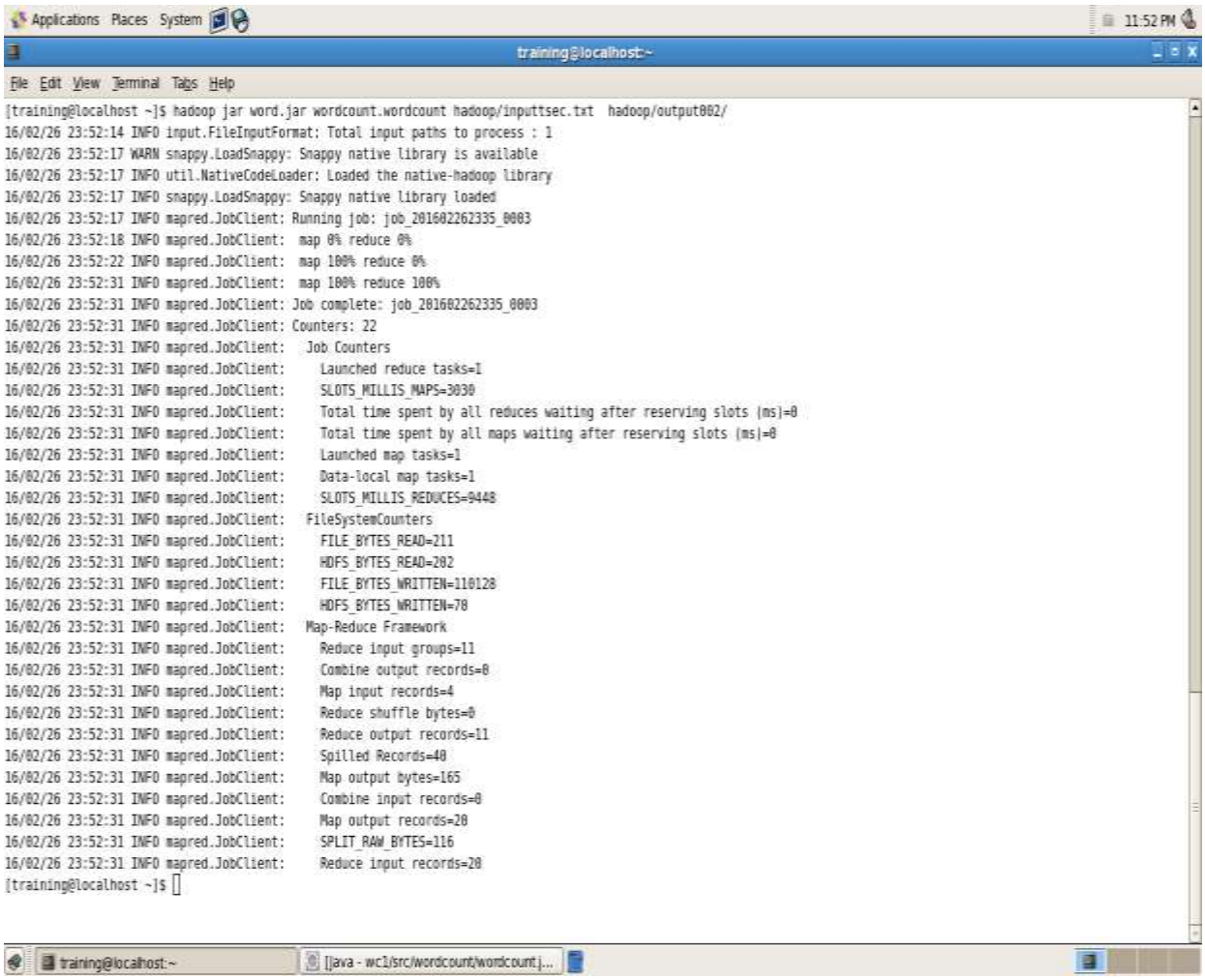
```
[training@localhost ~]$ cat > inputtsec
this is a demo program on Map reduce
the the is is a a demo mo
map map
reduce reduce

[1]+  Stopped                 cat > inputtsec
[training@localhost ~]$ hadoop fs -copyFromLocal /home/training/inputtsec hadoop/
[training@localhost ~]$ hadoop fs -copyFromLocal /home/training/word.jar hadoop/
[training@localhost ~]$
```

Step 8 -: Run the program using following command

$ hadoop jar jar-name.jar package.class input-file(s) output-directory

In our program jar file name is word.jar, package name is wordcount, class name is wordcount and input file name is inputtsec. So command will be

$hadoop jar word.jar wordcount.wordcount hadoop/inputtsec.txt hadoop/output002/



Step 9-: check the output

To see the output open part file which lies inside output002 directory

```
[training@localhost ~]$ hadoop fs -cat hadoop/output002/part-r-00000
Map     1
a       3
demo    2
is      3
map     2
mo      1
on      1
program 1
reduce  3
the     2
this    1
[training@localhost ~]$
```

Example 2-: Map-reduce program for Matrix Multiplication

```
package Matrixmulti;
import java.io.IOException;
import java.util.*;

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 Matrix{

   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();
         int m = Integer.parseInt(conf.get("m"));
         int p = Integer.parseInt(conf.get("p"));
```

```
            String line = value.toString();
            String[] indicesAndValue = line.split(",");
            Text outputKey = new Text();
            Text outputValue = new Text();
            if (indicesAndValue[0].equals("A")) {
                for (int k = 0; k < p; k++) {
                    outputKey.set(indicesAndValue[1] + "," + k);
                    outputValue.set("A," + indicesAndValue[2] + "," + indicesAndValue[3]);
                    context.write(outputKey, outputValue);
                }
            } else {
                for (int i = 0; i < m; i++) {
                    outputKey.set(i + "," + indicesAndValue[2]);
                    outputValue.set("B," + indicesAndValue[1] + "," + 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;
            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("A")) {
                    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 a_ij;
        float b_jk;
        for (int j = 0; j < n; j++) {
            a_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
            b_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
            result += a_ij * b_jk;
        }
        if (result != 0.0f) {
            context.write(null, new Text(key.toString() + "," + Float.toString(result)));
        }
    }
}

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");

    Job job = new Job(conf, "MatrixMatrixMultiplicationOneStep");
    job.setJarByClass(Matrix.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);
    }
}
```

Input

A,0,0,1.0

A,0,1,2.0

A,1,0,3.0

A,1,1,4.0

B,0,0,1.0

B,0,1,2.0

B,1,0,3.0

B,1,1,4.0

Output

0,0,7.0

0,1,10.0

1,0,15.0

1,1,22.0

Example 3-: Map-reduce program for K-means Clustering

```
package cl;
import java.io.IOException;
import java.util.*;
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
```

```
import org.apache.hadoop.mapred.Reducer;

//@SuppressWarnings("deprecation")
public class Clustering1 {
	public static String OUT = "outfile";
	public static String IN = "inputlarger";
	public static String CENTROID_FILE_NAME = "/centroid.txt";
	public static String OUTPUT_FILE_NAME = "/part-00000";
	public static String DATA_FILE_NAME = "/data.txt";
	public static String JOB_NAME = "KMeans";
	public static String SPLITTER = "\t| ";
	public static List<Double> mCenters = new ArrayList<Double>();

	/*
	 * In Mapper class we are overriding configure function. In this we are
	 * reading file from Distributed Cache and then storing that into instance
	 * variable "mCenters"
	 */
	public static class Map extends MapReduceBase implements
			Mapper<LongWritable, Text, DoubleWritable, DoubleWritable> {
		@Override
		public void configure(JobConf job) {
			try {
				// Fetch the file from Distributed Cache Read it and store the
				// centroid in the ArrayList
				Path[] cacheFiles = DistributedCache.getLocalCacheFiles(job);
				if (cacheFiles != null && cacheFiles.length > 0) {
					String line;
					mCenters.clear();
					BufferedReader cacheReader = new BufferedReader(
							new FileReader(cacheFiles[0].toString()));
					try {
						// Read the file split by the splitter and store it in
						// the list
						while ((line = cacheReader.readLine()) != null) {
							String[] temp = line.split(SPLITTER);
							mCenters.add(Double.parseDouble(temp[0]));
```

```
                                        }
                                } finally {
                                        cacheReader.close();
                                }
                        }
                } catch (IOException e) {
                        System.err.println("Exception reading DistribtuedCache: " + e);
                }
        }

        /*
         * Map function will find the minimum center of the point and emit it to
         * the reducer
         */
        @Override
        public void map(LongWritable key, Text value,
                        OutputCollector<DoubleWritable, DoubleWritable> output,
                        Reporter reporter) throws IOException {
                String line = value.toString();
                double point = Double.parseDouble(line);
                double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters
                                .get(0);
                // Find the minimum center from a point
                for (double c : mCenters) {
                        min1 = c - point;
                        if (Math.abs(min1) < Math.abs(min2)) {
                                nearest_center = c;
                                min2 = min1;
                        }
                }
                // Emit the nearest center and the point
                output.collect(new DoubleWritable(nearest_center),
                                new DoubleWritable(point));
        }
}

public static class Reduce extends MapReduceBase implements
```

```
                Reducer<DoubleWritable, DoubleWritable, DoubleWritable, Text> {

        /*
         * Reduce function will emit all the points to that center and calculate
         * the next center for these points
         */
        @Override
        public void reduce(DoubleWritable key, Iterator<DoubleWritable> values,
                        OutputCollector<DoubleWritable, Text> output, Reporter reporter)
                        throws IOException {
                double newCenter;
                double sum = 0;
                int no_elements = 0;
                String points = "";
                while (values.hasNext()) {
                        double d = values.next().get();
                        points = points + " " + Double.toString(d);
                        sum = sum + d;
                        ++no_elements;
                }

                // We have new center now
                newCenter = sum / no_elements;

                // Emit new center and point
                output.collect(new DoubleWritable(newCenter), new Text(points));
        }
}

public static void main(String[] args) throws Exception {
        run(args);
}

public static void run(String[] args) throws Exception {
        IN = args[0];
        OUT = args[1];
        String input = IN;
```

```
String output = OUT + System.nanoTime();
String again_input = output;

// Reiterating till the convergence
int iteration = 0;
boolean isdone = false;
while (isdone == false) {
        JobConf conf = new JobConf(Kpack1.class);
        if (iteration == 0) {
                Path hdfsPath = new Path(input + CENTROID_FILE_NAME);
                // upload the file to hdfs. Overwrite any existing copy.
                DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
        } else {
                Path hdfsPath = new Path(again_input + OUTPUT_FILE_NAME);
                // upload the file to hdfs. Overwrite any existing copy.
                DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
        }

        conf.setJobName(JOB_NAME);
        conf.setMapOutputKeyClass(DoubleWritable.class);
        conf.setMapOutputValueClass(DoubleWritable.class);
        conf.setOutputKeyClass(DoubleWritable.class);
        conf.setOutputValueClass(Text.class);
        conf.setMapperClass(Map.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf,
                        new Path(input + DATA_FILE_NAME));
        FileOutputFormat.setOutputPath(conf, new Path(output));

        JobClient.runJob(conf);

        Path ofile = new Path(output + OUTPUT_FILE_NAME);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br = new BufferedReader(new InputStreamReader(
```

```
                        fs.open(ofile)));
List<Double> centers_next = new ArrayList<Double>();
String line = br.readLine();
while (line != null) {
        String[] sp = line.split("\t| ");
        double c = Double.parseDouble(sp[0]);
        centers_next.add(c);
        line = br.readLine();
}
br.close();

String prev;
if (iteration == 0) {
        prev = input + CENTROID_FILE_NAME;
} else {
        prev = again_input + OUTPUT_FILE_NAME;
}
Path prevfile = new Path(prev);
FileSystem fs1 = FileSystem.get(new Configuration());
BufferedReader br1 = new BufferedReader(new InputStreamReader(
                        fs1.open(prevfile)));
List<Double> centers_prev = new ArrayList<Double>();
String l = br1.readLine();
while (l != null) {
        String[] sp1 = l.split(SPLITTER);
        double d = Double.parseDouble(sp1[0]);
        centers_prev.add(d);
        l = br1.readLine();
}
br1.close();

// Sort the old centroid and new centroid and check for convergence
// condition
Collections.sort(centers_next);
Collections.sort(centers_prev);

Iterator<Double> it = centers_prev.iterator();
```

```
                        for (double d : centers_next) {
                                double temp = it.next();
                                if (Math.abs(temp - d) <= 0.1) {
                                        isdone = true;
                                } else {
                                        isdone = false;
                                        break;
                                }
                        }
                        ++iteration;
                        again_input = output;
                        output = OUT + System.nanoTime();
                }
        }
}
```

Input

```
[training@localhost ~]$ nano data.txt
1
2
3
4
5
6
7
8
9
10
[training@localhost ~]$ nano centroid.txt
1
2
3
[training@localhost ~]$ hadoop fs -copyFromLocal data.txt hadoop/
[training@localhost ~]$ hadoop fs -copyFromLocal centroid.txt hadoop/
```

Output

```
[training@localhost ~]$ hadoop jar cluster.jar cl.clustering1 hadoop/ hadoop/ hadoop/
1.0     1.0
2.0     2.0
```

```
6.5     3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0

1.0     1.0
3.0     2.0 3.0 4.0
7.5     5.0 6.0 7.0 8.0 9.0 10.0

1.5     1.0 2.0
4.0     3.0 4.0 5.0
8.0     6.0 7.0 8.0 9.0 10.0

1.5     1.0 2.0
4.5     3.0 4.0 5.0 6.0
8.5     7.0 8.0 9.0 10.0

2.0     1.0 2.0 3.0
5.0     4.0 5.0 6.0
8.5     7.0 8.0 9.0 10.0

2.0     1.0 2.0 3.0
5.0     4.0 5.0 6.0
8.5     7.0 8.0 9.0 10.0
```

Example 4-: Map-reduce program for frequent item set

```
package frequent_itemset;
import java.io.IOException;
import java.util.*;
import java.util.Map.Entry;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;

import org.apache.commons.lang.text.StrBuilder;

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 Frequentitemset {

   public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {
      public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
         String line = value.toString();
         String[] userAndFriends = line.split("\t");
         if (userAndFriends.length == 2) {
            String user = userAndFriends[0];
            IntWritable userKey = new IntWritable(Integer.parseInt(user));
            String[] friends = userAndFriends[1].split(",");
            String friend1;
            IntWritable friend1Key = new IntWritable();
            Text friend1Value = new Text();
            String friend2;
            IntWritable friend2Key = new IntWritable();
            Text friend2Value = new Text();
            for (int i = 0; i < friends.length; i++) {
               friend1 = friends[i];
               friend1Value.set("1," + friend1);
               context.write(userKey, friend1Value);   // Paths of length 1.
               friend1Key.set(Integer.parseInt(friend1));
               friend1Value.set("2," + friend1);
               for (int j = i+1; j < friends.length; j++) {
                  friend2 = friends[j];
                  friend2Key.set(Integer.parseInt(friend2));
                  friend2Value.set("2," + friend2);
                  context.write(friend1Key, friend2Value);   // Paths of length 2.
                  context.write(friend2Key, friend1Value);   // Paths of length 2.
               }
            }
         }
      }
   }

   public static class Reduce extends Reducer<IntWritable, Text, IntWritable, Text> {
      public void reduce(IntWritable key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
         String[] value;
         HashMap<String, Integer> hash = new HashMap<String, Integer>();
         for (Text val : values) {
            value = (val.toString()).split(",");
            if (value[0].equals("1")) { // Paths of length 1.
```

```
            hash.put(value[1], -1);
        } else if (value[0].equals("2")) {  // Paths of length 2.
            if (hash.containsKey(value[1])) {
                if (hash.get(value[1]) != -1) {
                    hash.put(value[1], hash.get(value[1]) + 1);
                }
            } else {
                hash.put(value[1], 1);
            }
        }
    }
    // Convert hash to list and remove paths of length 1.
    ArrayList<Entry<String, Integer>> list = new ArrayList<Entry<String, Integer>>();
    for (Entry<String, Integer> entry : hash.entrySet()) {
        if (entry.getValue() != -1) {   // Exclude paths of length 1.
            list.add(entry);
        }
    }
    // Sort key-value pairs in the list by values (number of common friends).
    Collections.sort(list, new Comparator<Entry<String, Integer>>() {
        public int compare(Entry<String, Integer> e1, Entry<String, Integer> e2) {
            return e2.getValue().compareTo(e1.getValue());
        }
    });
    int MAX_RECOMMENDATION_COUNT = 10;
    if (MAX_RECOMMENDATION_COUNT < 1) {
        // Output all key-value pairs in the list.
      String A1="";
      String separator=",";
      if (list == null) {
            A1=null;
        }

      int startIndex=0;
      int endIndex=list.size();
      int bufSize = (endIndex - startIndex);
       if (bufSize <= 0) {
            A1="";
        }

        //bufSize *= ((list.get(startIndex) == null ? 16 : list.get(startIndex).toString().length()) + 1);
        //StrBuilder buf = new StrBuilder(bufSize);

        for (int i = startIndex; i < endIndex; i++) {
            if (i > startIndex) {
                    A1= A1.concat(separator);
            }
            if (list.get(i) != null) {
```

```
                    A1= A1.concat(list.get(i).toString());
                }
            }
            context.write(key, new Text(A1));
        } else {
            // Output at most MAX_RECOMMENDATION_COUNT keys with the highest values (number of
common friends).
            ArrayList<String> top = new ArrayList<String>();
            for (int i = 0; i < Math.min(MAX_RECOMMENDATION_COUNT, list.size()); i++) {
                top.add(list.get(i).getKey());
            }
           String A2="";
           String separator=",";
           if (top == null) {
               A2=null;
            }

           int startIndex=0;
           int endIndex=top.size();
           int bufSize = (endIndex - startIndex);
            if (bufSize <= 0) {
               A2="";
            }

          //  bufSize *= ((list.get(startIndex) == null ? 16 : list.get(startIndex).toString().length()) + 1);
           // StrBuilder buf = new StrBuilder(bufSize);

            for (int i = startIndex; i < endIndex; i++) {
               if (i > startIndex) {
                 // buf.append(separator);
                     A2= A2.concat(separator);
               }
               if (top.get(i) != null) {
                //  buf.append(list.get(i));
                     A2= A2.concat(top.get(i).toString());
               }
            }
            context.write(key, new Text(A2));
        }
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "FriendshipRecommender");
    job.setJarByClass(Frequentitemset.class);
    job.setOutputKeyClass(IntWritable.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);
  }
}
```

Input:

1 2,3,4

2 1,4,3

3 4,5,6

4 1,2,3

5 3,4,6

Output

1

2

3 2,1

4 6,5

5

6 4,3,5