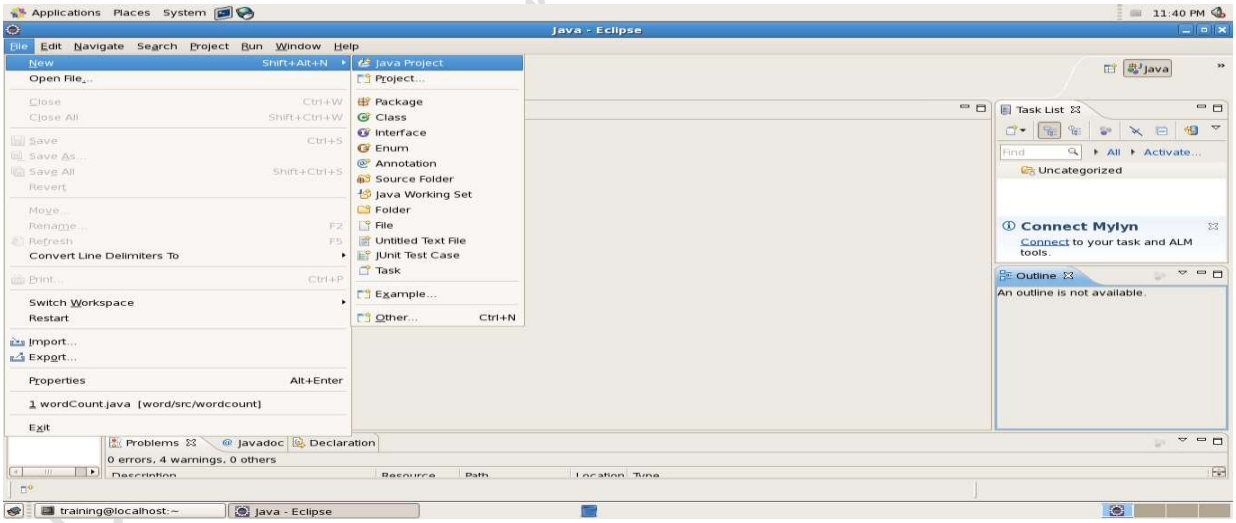
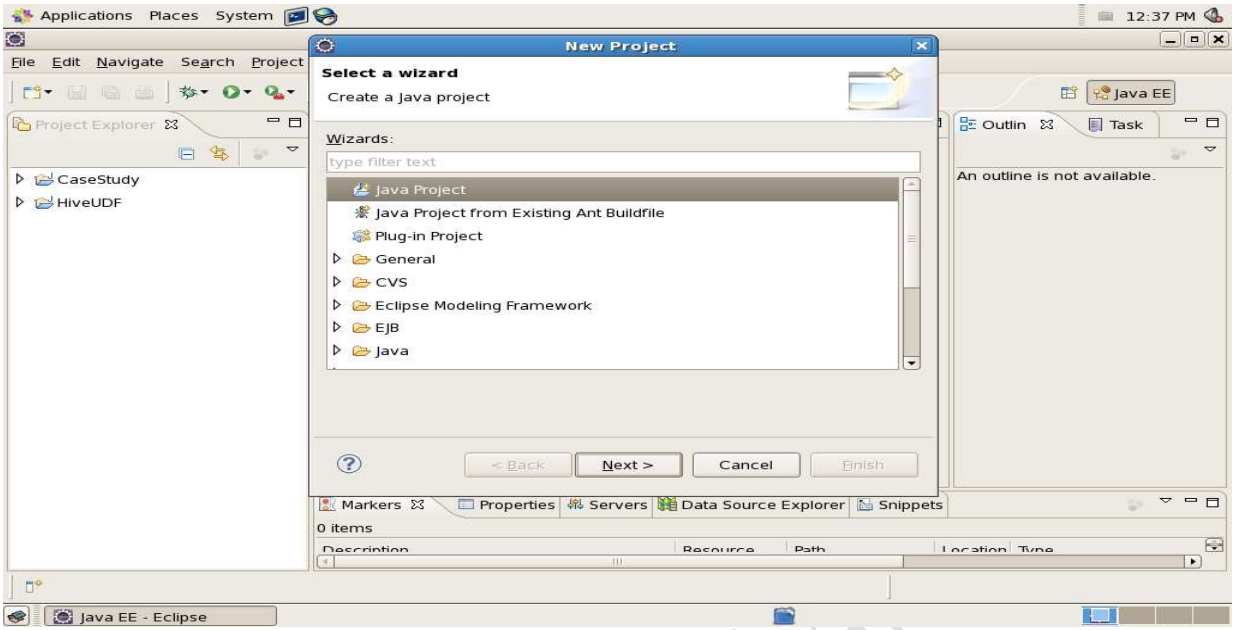
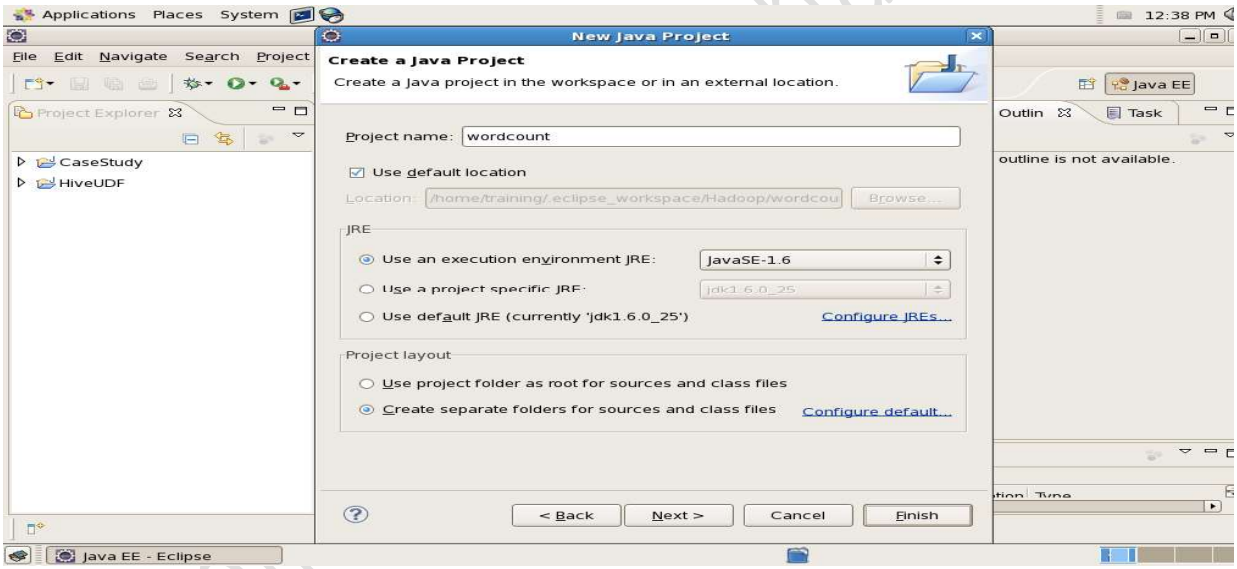
Lab 2 - Write a Map-reduce program for to demonstrate Word count application

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

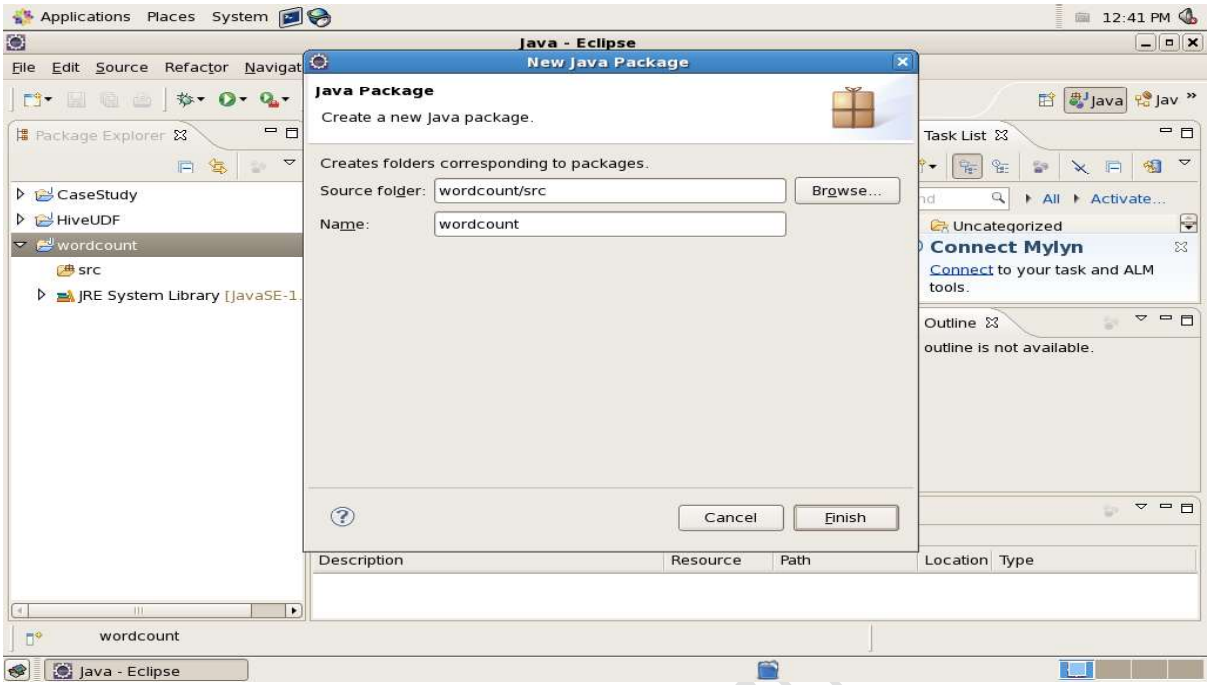
Step 1-: Open Eclipse and create new Java project specify name and click on finish



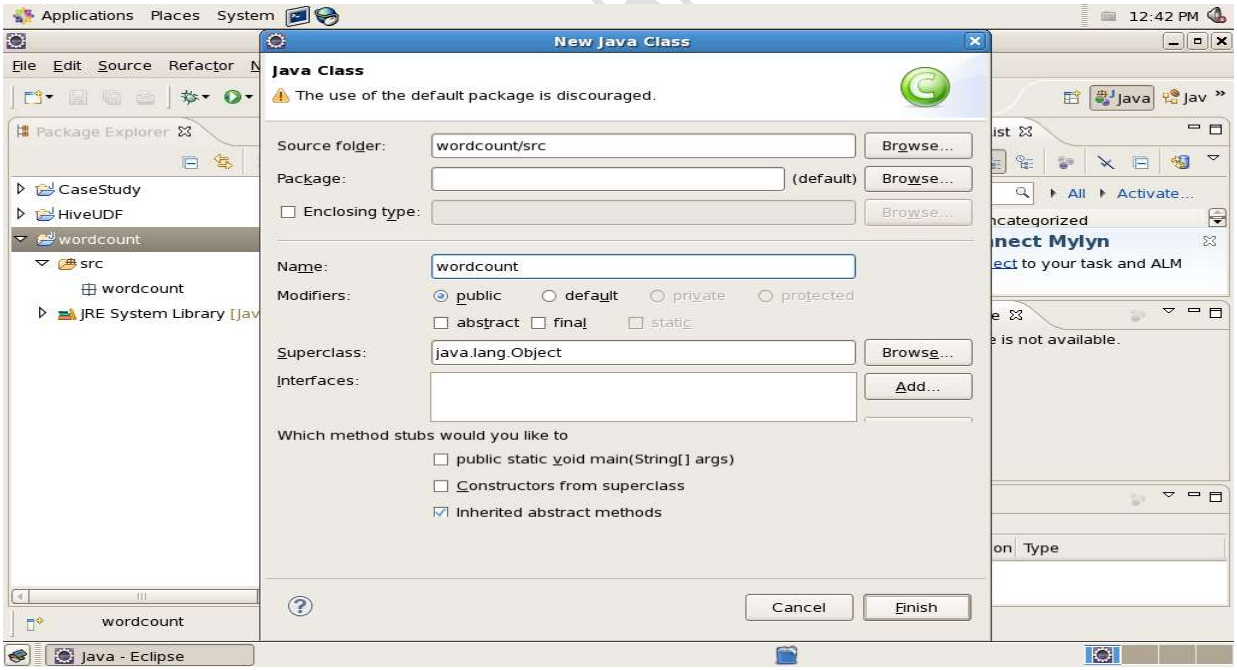




Step 2-: Right click on project and Create new package wordcount



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

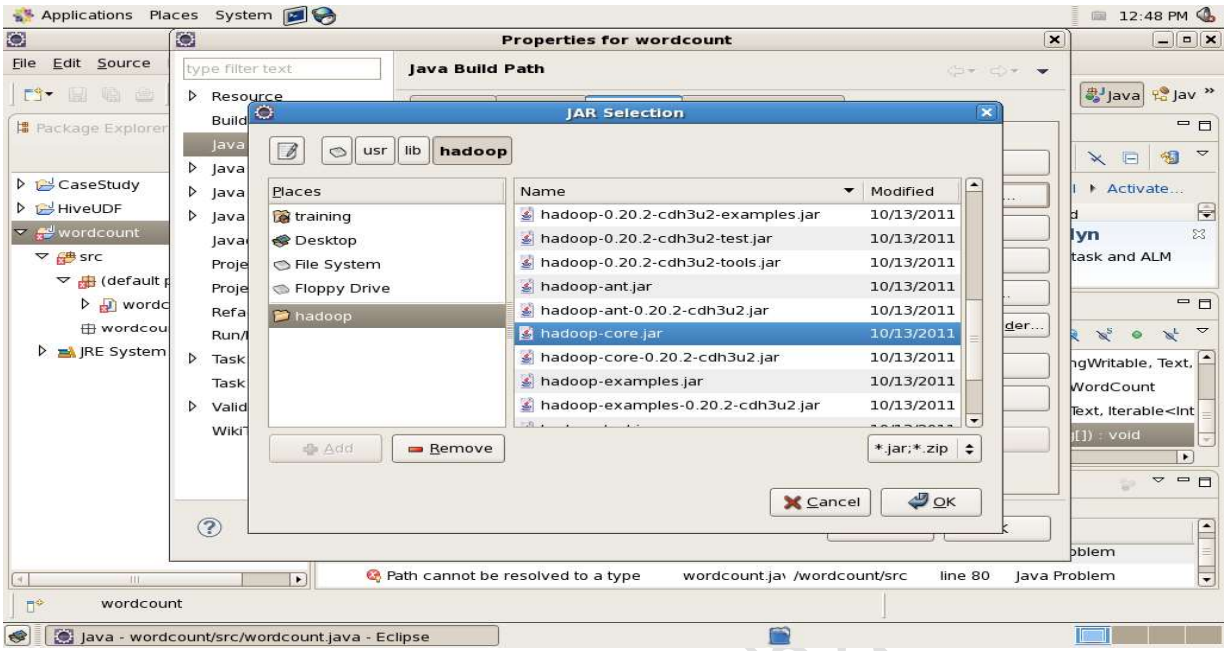
```
package wordcount;

import java.io.IOException;
import java.util.StringTokenizer;
```

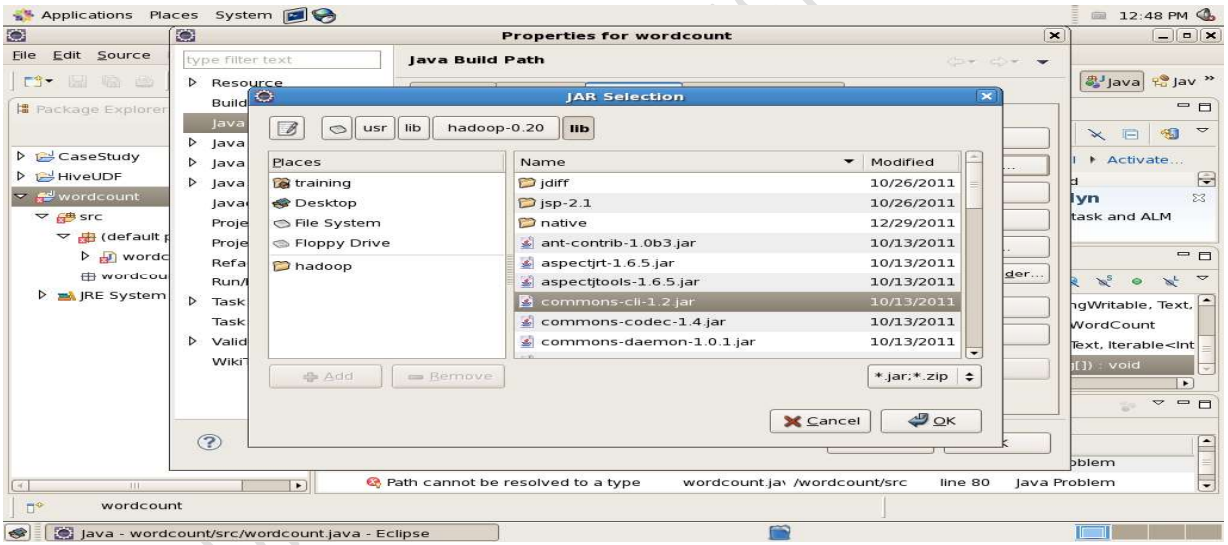
```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
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
```

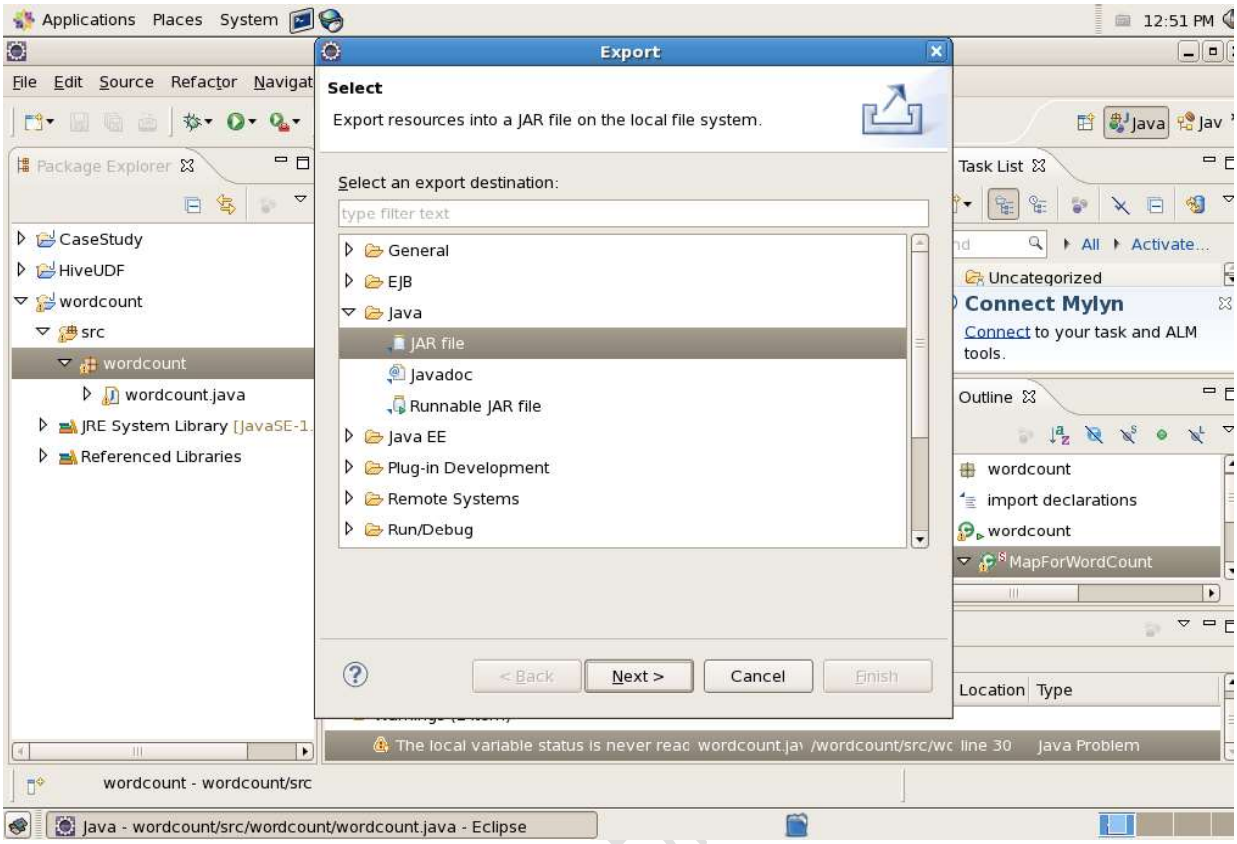
Step 5-: Add required jar files to resolve errors

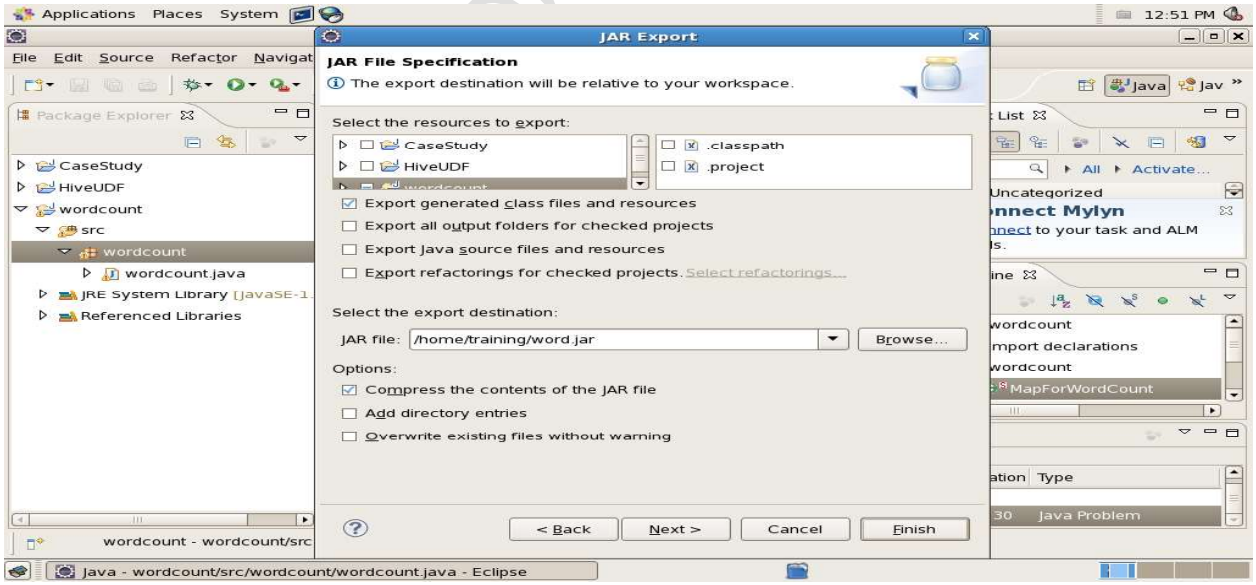
To add jar files right click on class file then select build path option then open configure build path window. To add essential libraries click on add external jars butoon and add three jar files one by one .Here we need three jar files namely hadoop-core.jar,common-cli-1.2.jar and core-3.1.1.jar





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/

```
Applications  Places  System                                                    11:52 PM
training@localhost:~
File  Edit  View  Terminal  Tabs  Help
[training@localhost ~]$ hadoop jar word.jar wordcount.wordcount hadoop/inputtsec.txt  hadoop/output002/
16/02/26 23:52:14 INFO input.FileInputFormat: Total input paths to process : 1
16/02/26 23:52:17 WARN snappy.LoadSnappy: Snappy native library is available
16/02/26 23:52:17 INFO util.NativeCodeLoader: Loaded the native-hadoop library
16/02/26 23:52:17 INFO snappy.LoadSnappy: Snappy native library loaded
16/02/26 23:52:17 INFO mapred.JobClient: Running job: job_201602262335_0003
16/02/26 23:52:18 INFO mapred.JobClient:  map 0% reduce 0%
16/02/26 23:52:22 INFO mapred.JobClient:  map 100% reduce 0%
16/02/26 23:52:31 INFO mapred.JobClient:  map 100% reduce 100%
16/02/26 23:52:31 INFO mapred.JobClient: Job complete: job_201602262335_0003
16/02/26 23:52:31 INFO mapred.JobClient: Counters: 22
16/02/26 23:52:31 INFO mapred.JobClient:   Job Counters
16/02/26 23:52:31 INFO mapred.JobClient:     Launched reduce tasks=1
16/02/26 23:52:31 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=3030
16/02/26 23:52:31 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
16/02/26 23:52:31 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
16/02/26 23:52:31 INFO mapred.JobClient:     Launched map tasks=1
16/02/26 23:52:31 INFO mapred.JobClient:     Data-local map tasks=1
16/02/26 23:52:31 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=9448
16/02/26 23:52:31 INFO mapred.JobClient:   FileSystemCounters
16/02/26 23:52:31 INFO mapred.JobClient:     FILE_BYTES_READ=211
16/02/26 23:52:31 INFO mapred.JobClient:     HDFS_BYTES_READ=202
16/02/26 23:52:31 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=110128
16/02/26 23:52:31 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=70
16/02/26 23:52:31 INFO mapred.JobClient:   Map-Reduce Framework
16/02/26 23:52:31 INFO mapred.JobClient:     Reduce input groups=11
16/02/26 23:52:31 INFO mapred.JobClient:     Combine output records=0
16/02/26 23:52:31 INFO mapred.JobClient:     Map input records=4
16/02/26 23:52:31 INFO mapred.JobClient:     Reduce shuffle bytes=0
16/02/26 23:52:31 INFO mapred.JobClient:     Reduce output records=11
16/02/26 23:52:31 INFO mapred.JobClient:     Spilled Records=40
16/02/26 23:52:31 INFO mapred.JobClient:     Map output bytes=165
16/02/26 23:52:31 INFO mapred.JobClient:     Combine input records=0
16/02/26 23:52:31 INFO mapred.JobClient:     Map output records=20
16/02/26 23:52:31 INFO mapred.JobClient:     SPLIT_RAW_BYTES=116
16/02/26 23:52:31 INFO mapred.JobClient:     Reduce input records=20
[training@localhost ~]$
training@localhost:~   [Java - wc1/src/wordcount/wordcount.j...
```

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 ~]$
```