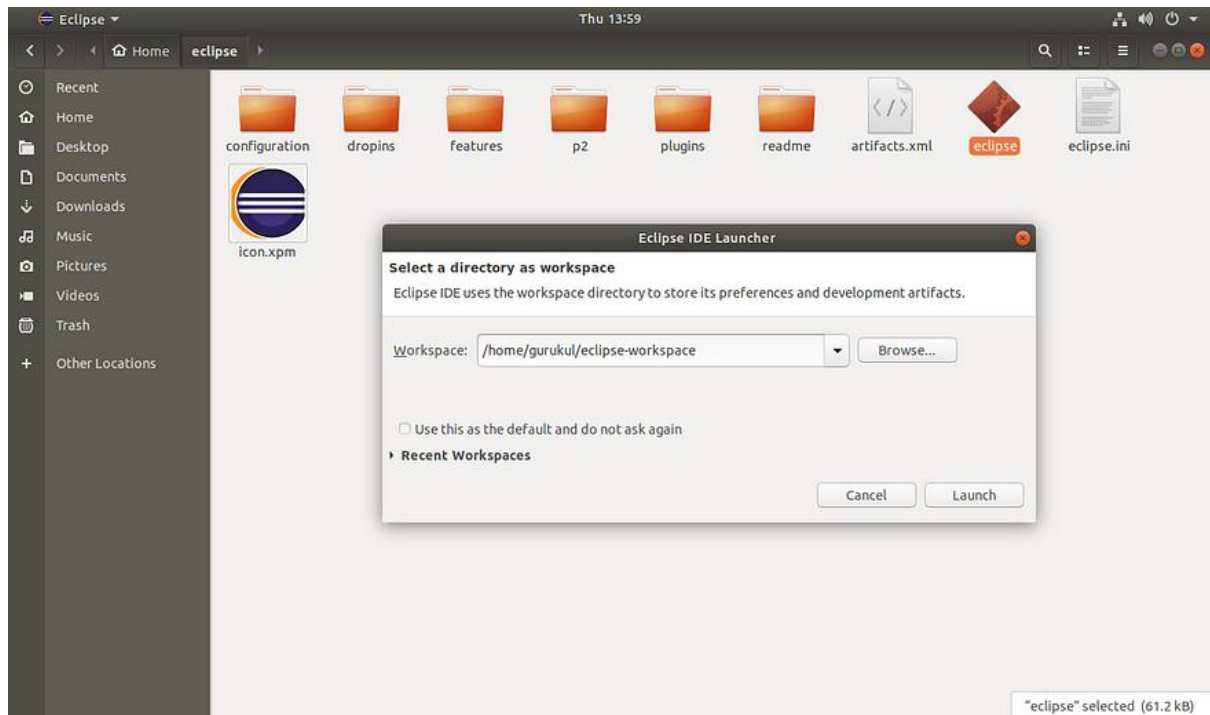


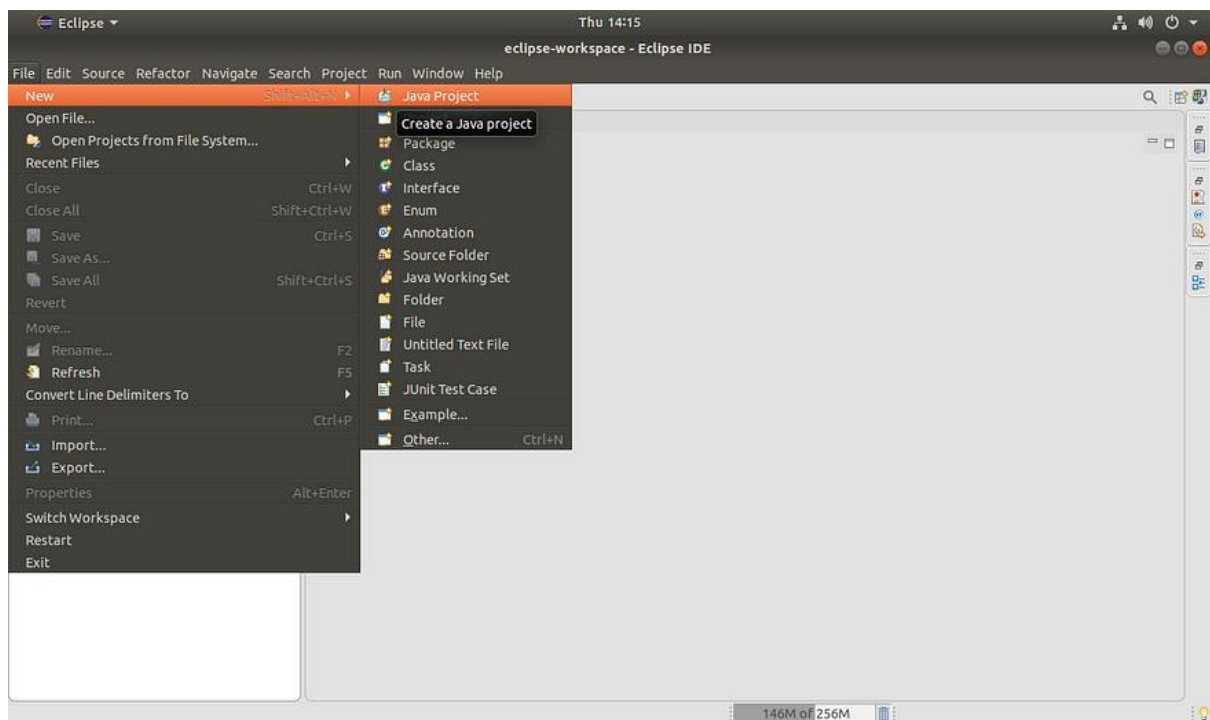
## Hadoop MapReduce in Java With Eclipse

Here are the steps to create the Hadoop MapReduce Project in Java with Eclipse:

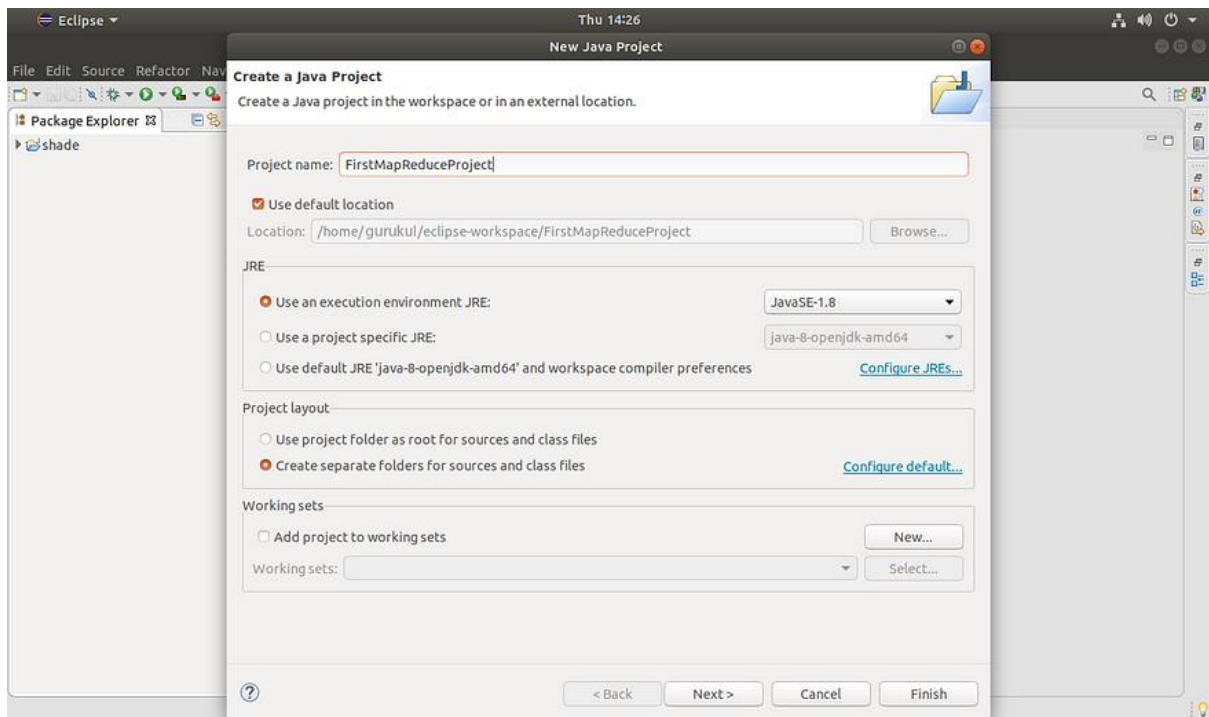
**Step 1. Launch Eclipse and set the Eclipse Workspace.**



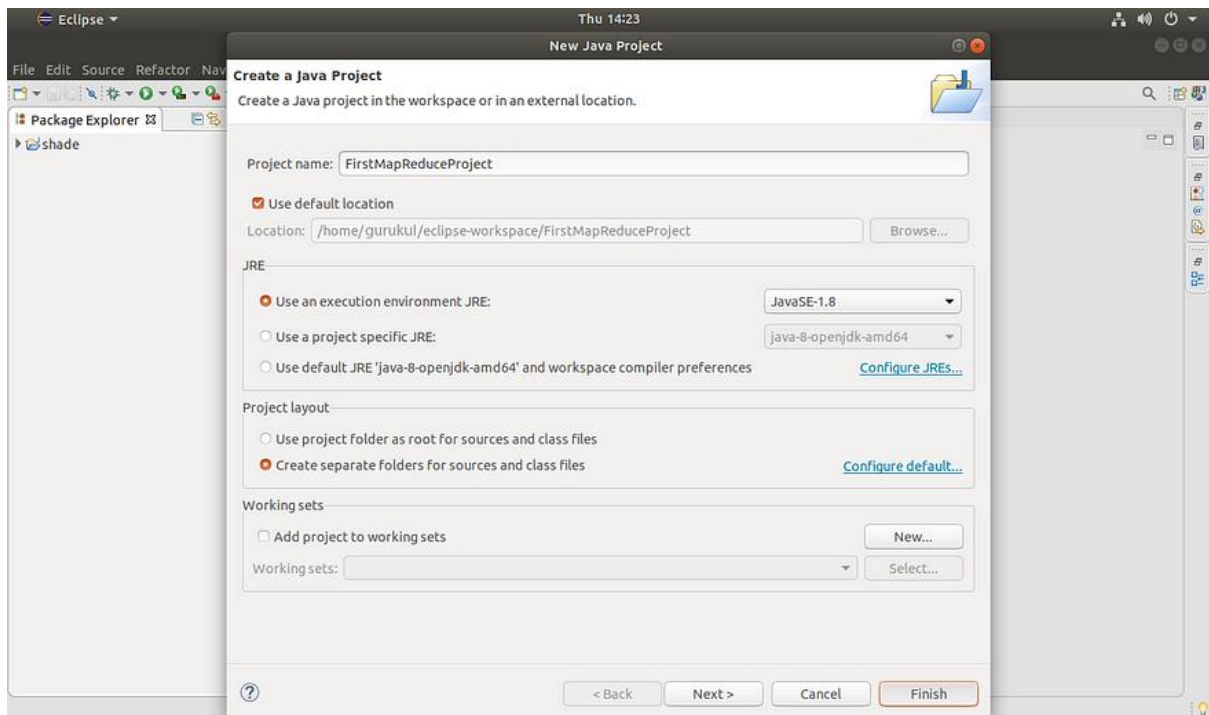
**Step 2. To create the Hadoop MapReduce Project, click on File >> New >> Java Project.**



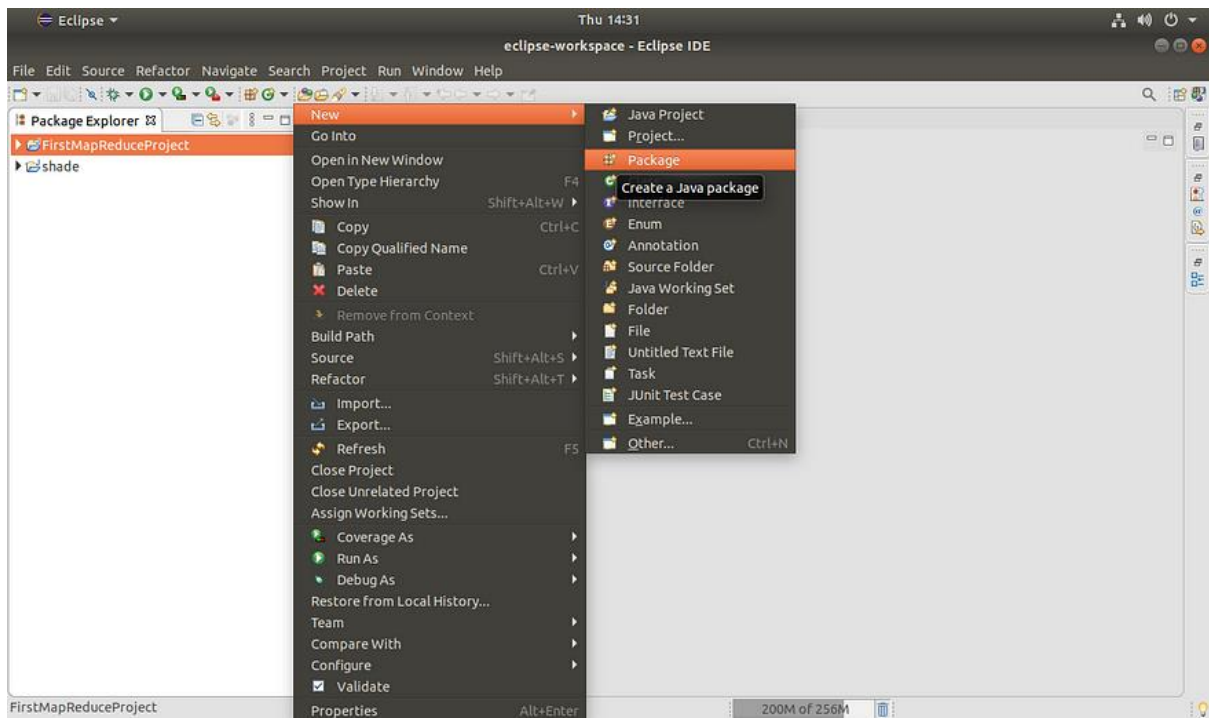
**Provide the Project Name:**



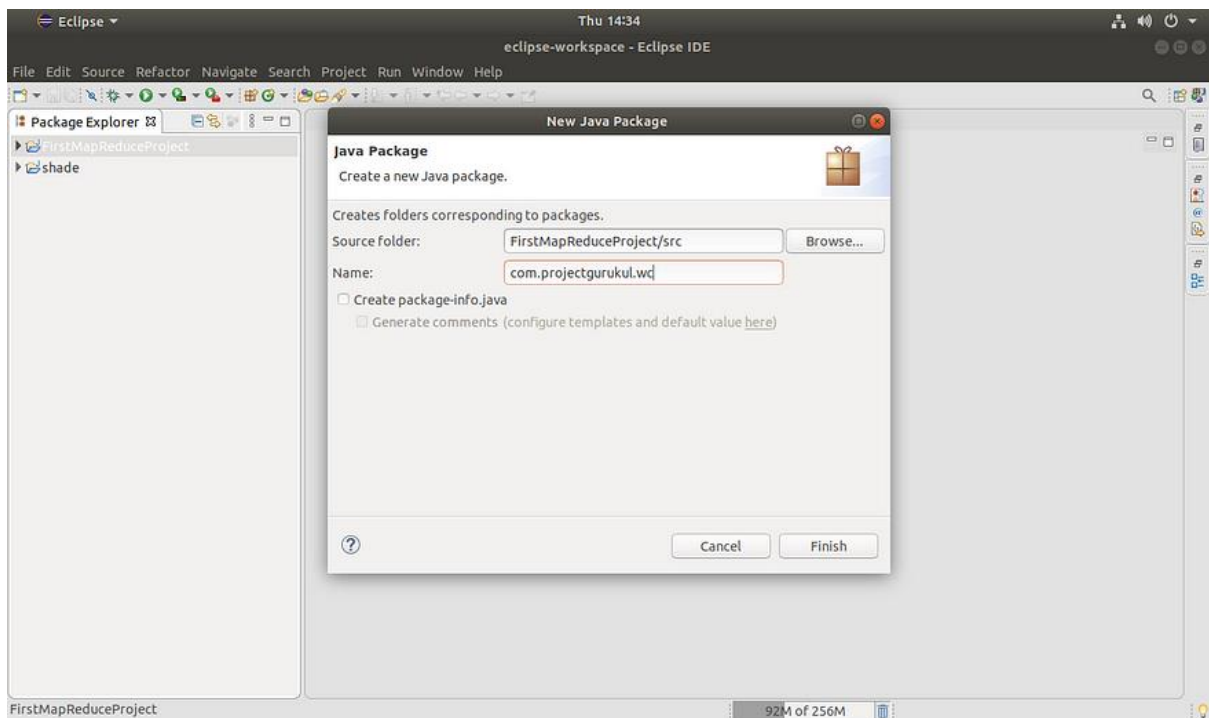
**Click Finish to create the project.**



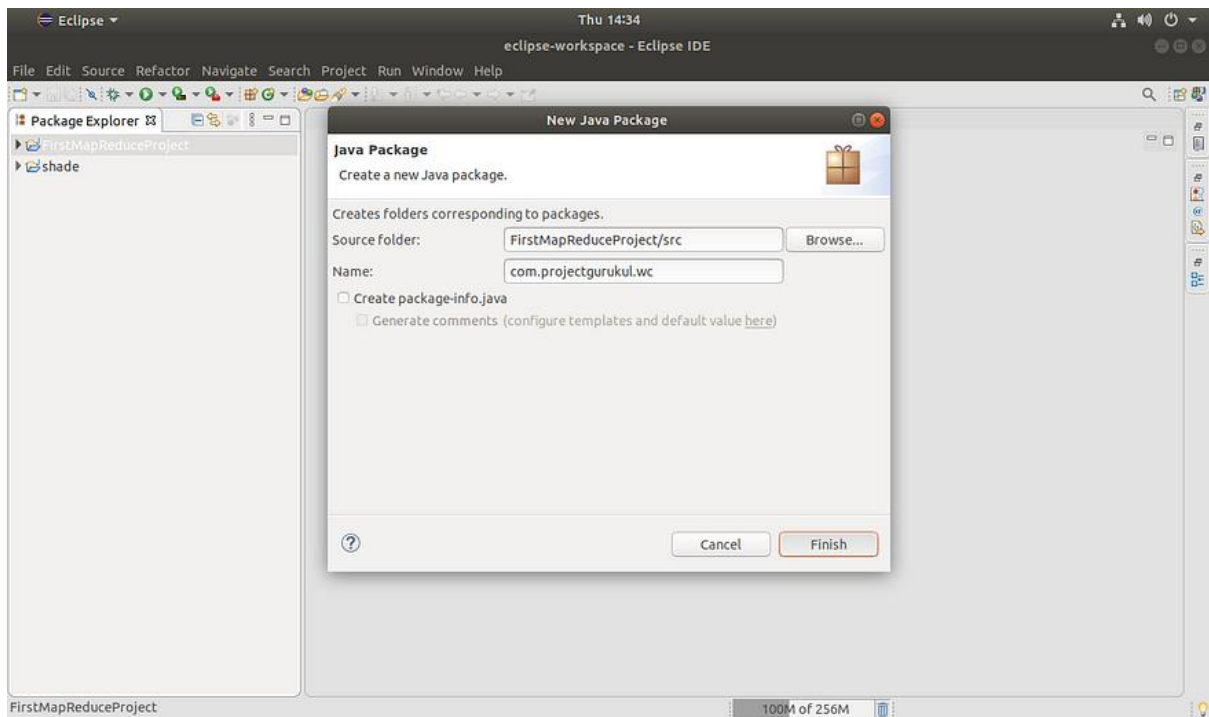
**Step 3. Create a new Package right-click on the Project Name >> New >> Package.**



**Provide the package name:**

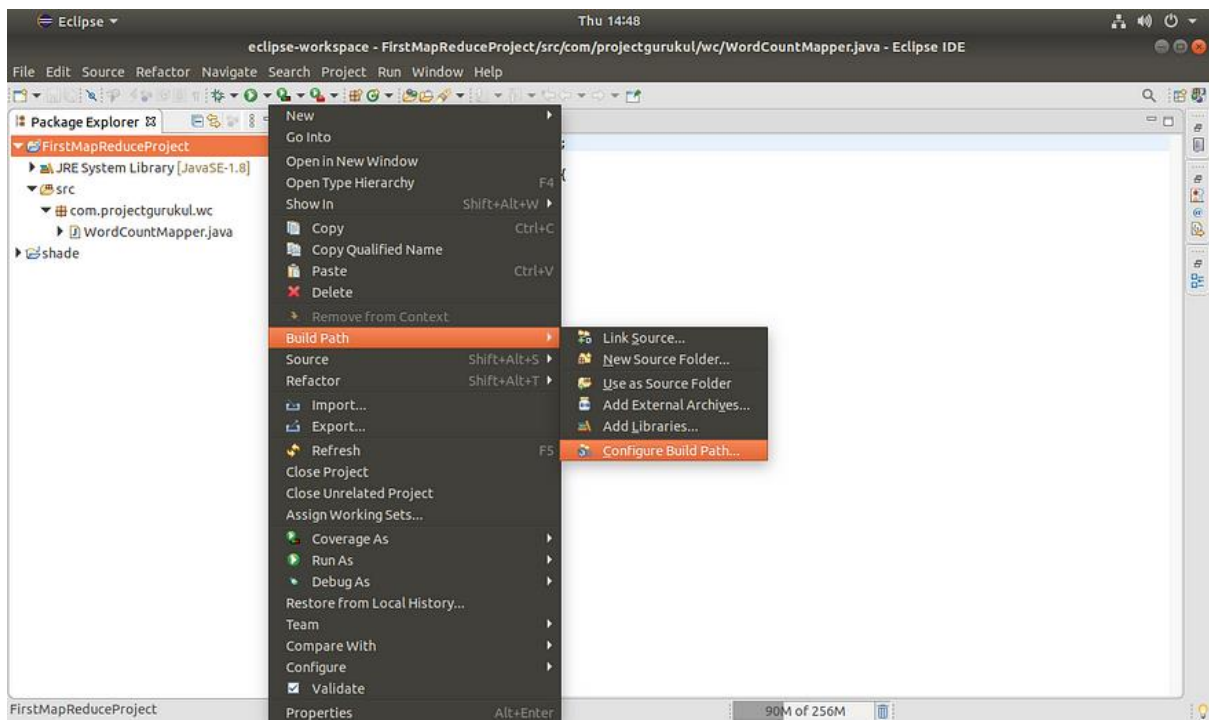


Click Finish to create the package.

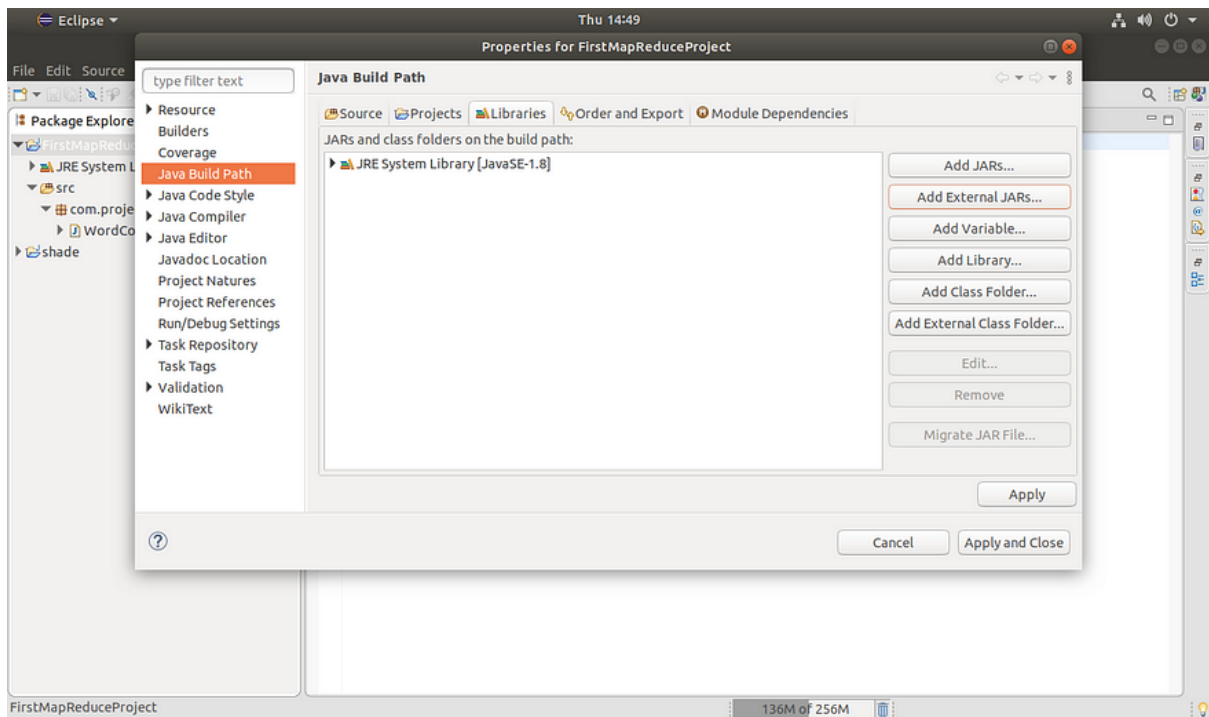


**Step 4. Add the Hadoop libraries (jars).**

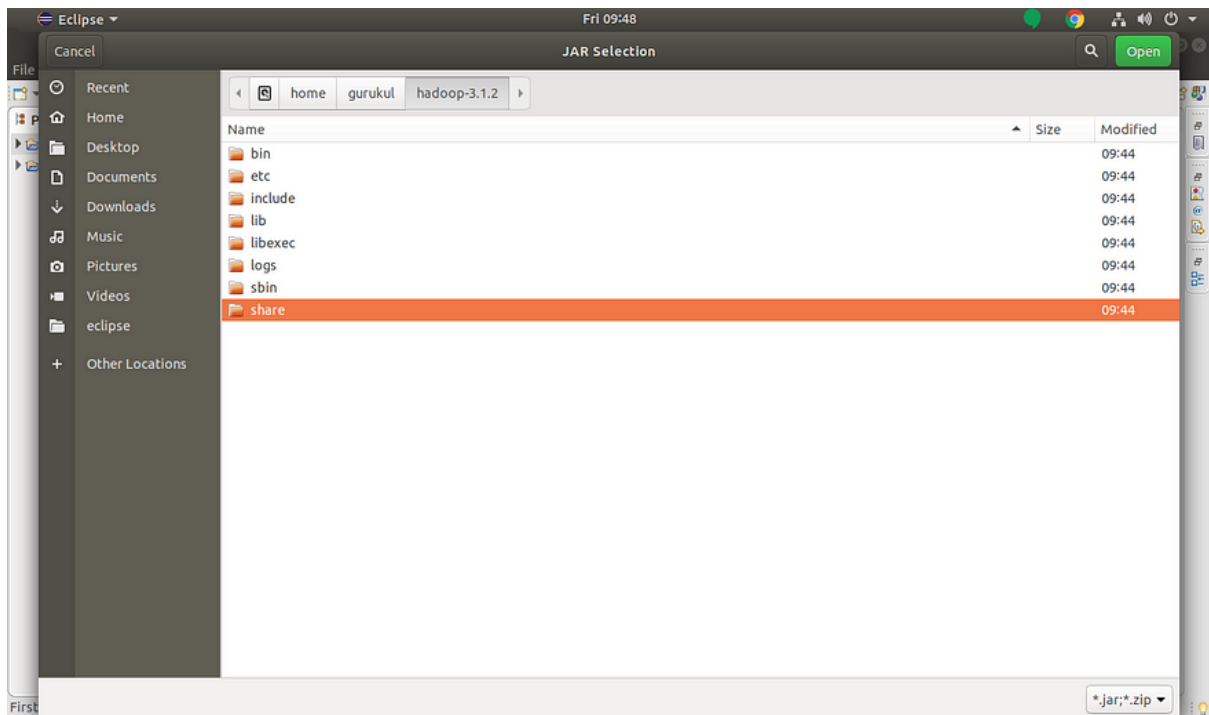
To do so Right-Click on Project Name >>Build Path>> configure Build Path.



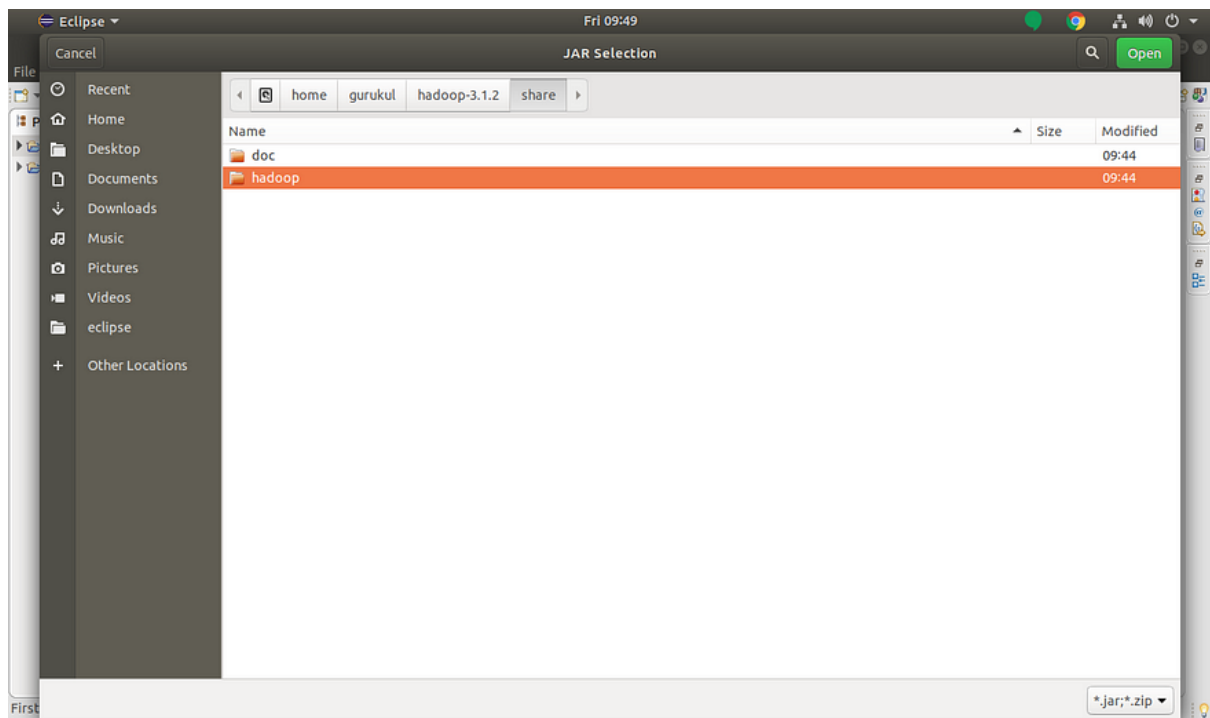
**Add the External jars.**



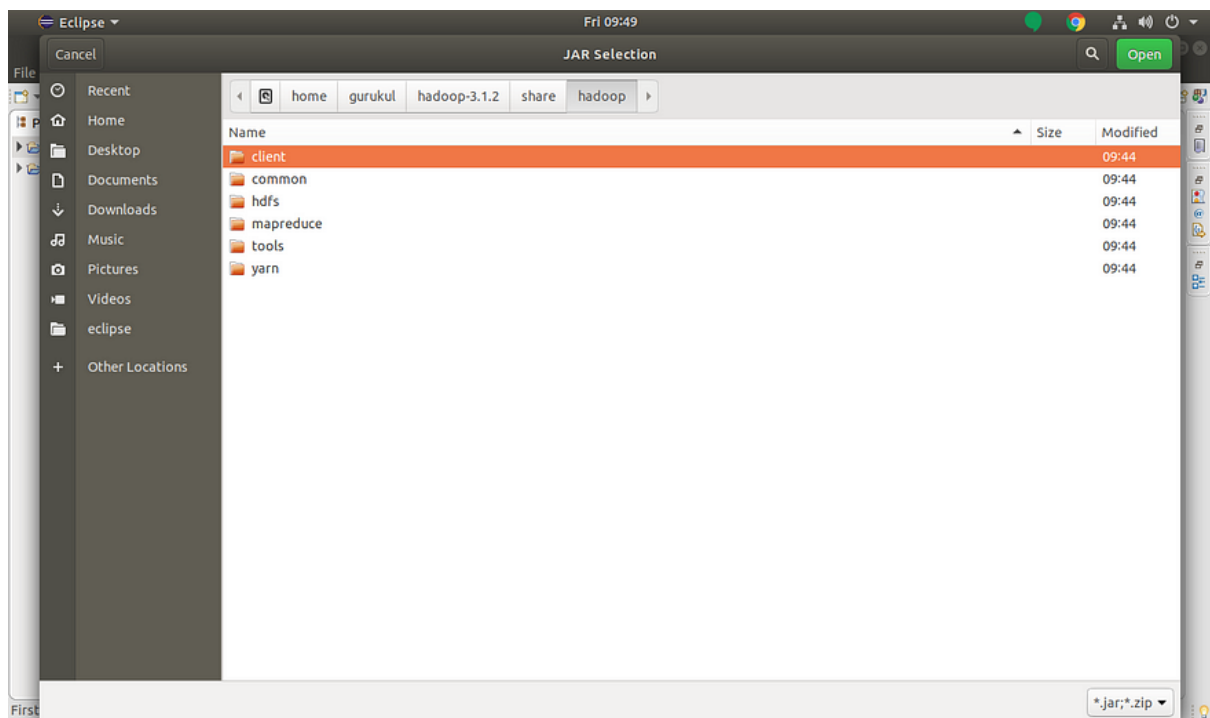
**For this go to `hadoop-3.1.2>> share >> hadoop`.**



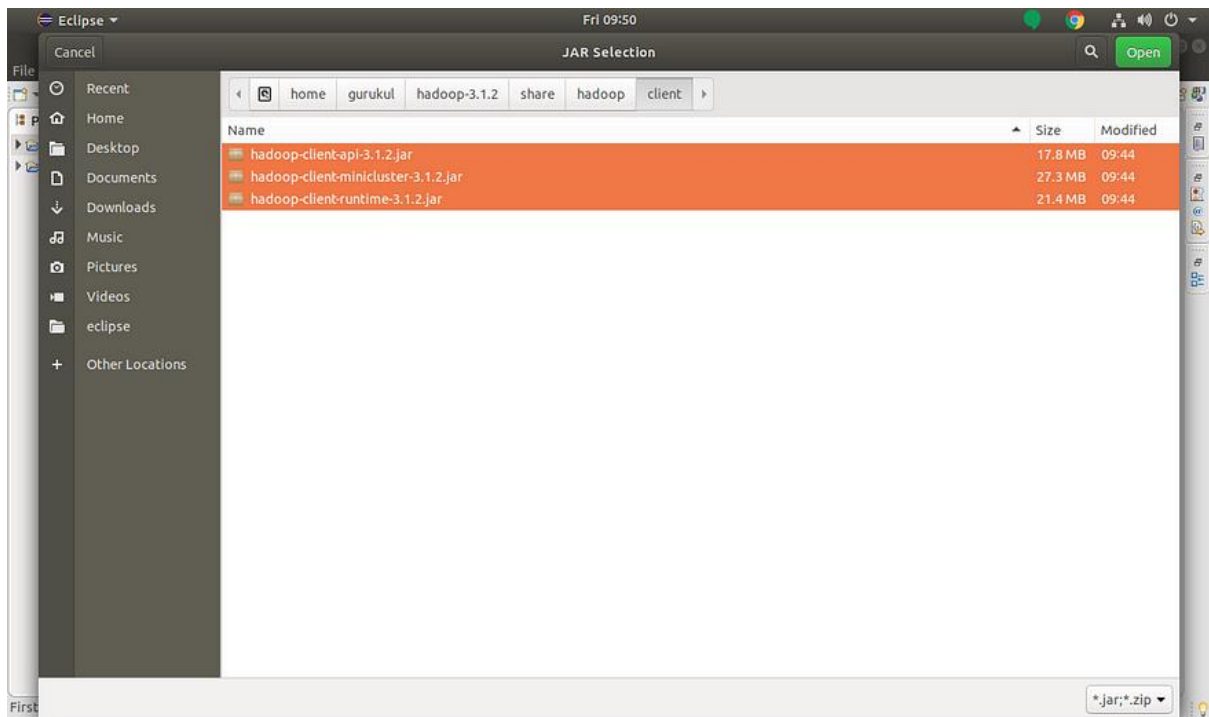
**Now we will move to `share >> Hadoop` in Hadoop MapReduce Project.**



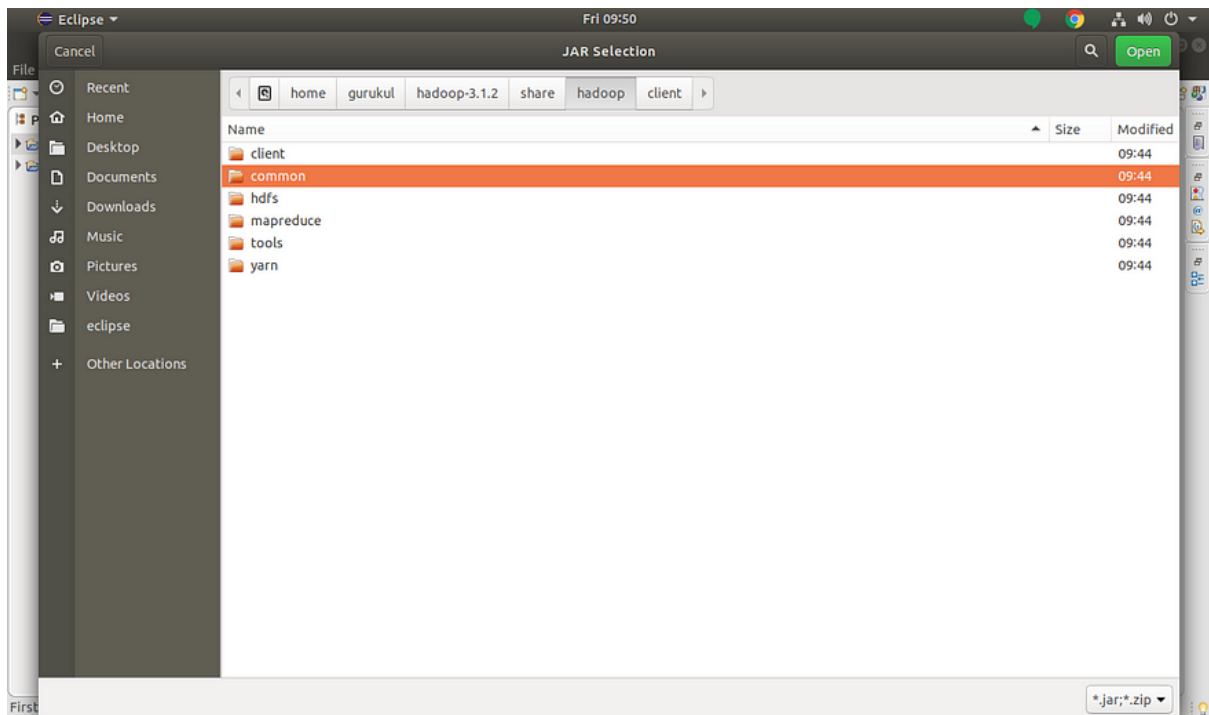
**A. Add the client jar files.**



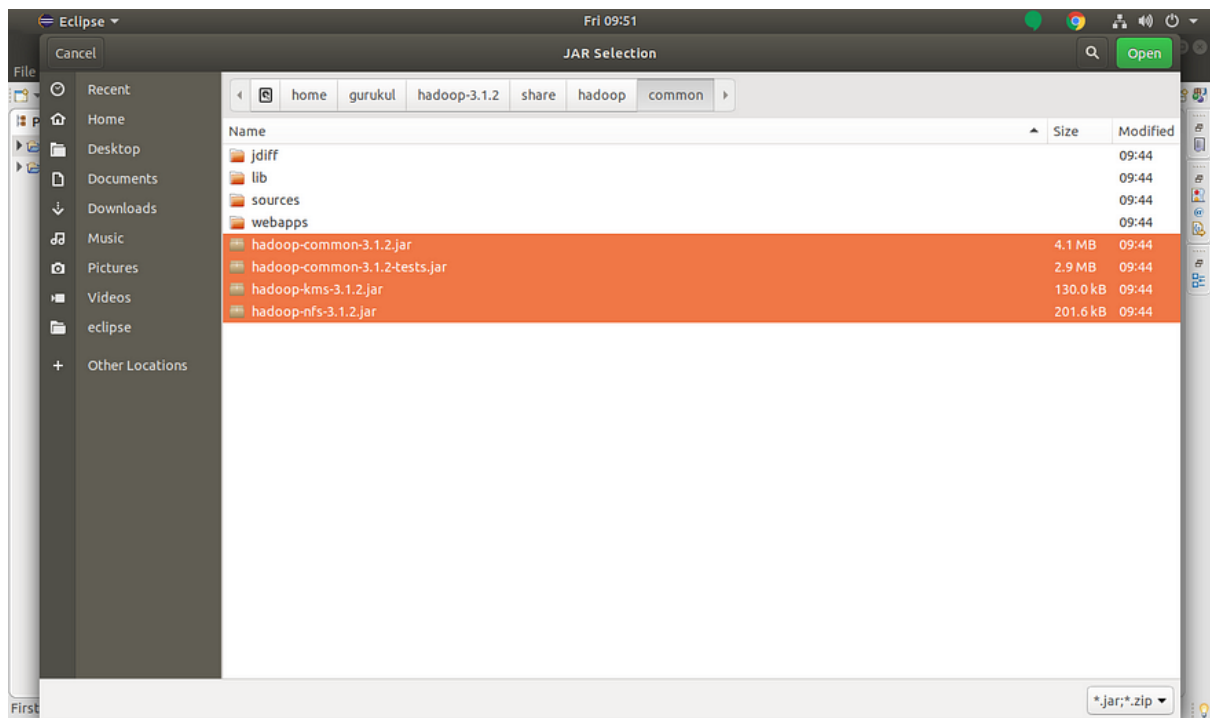
Select client jar files and click on Open.



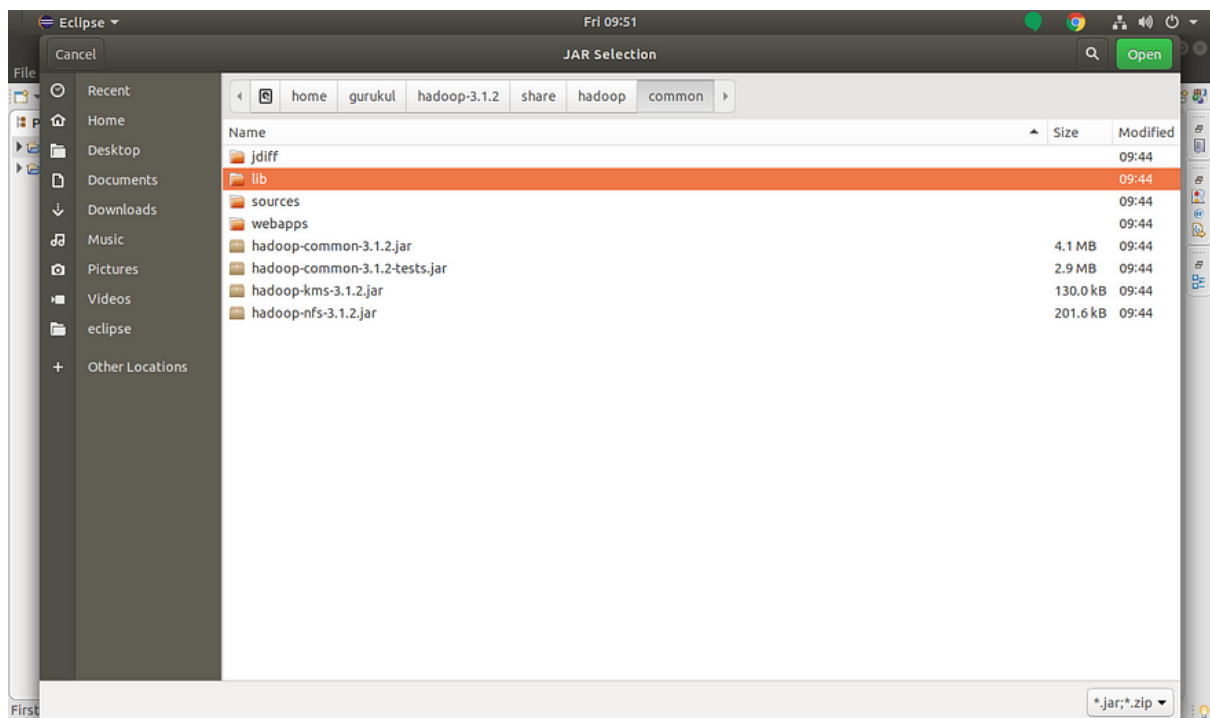
B. Add common jar files.



Select common jar files and Open.

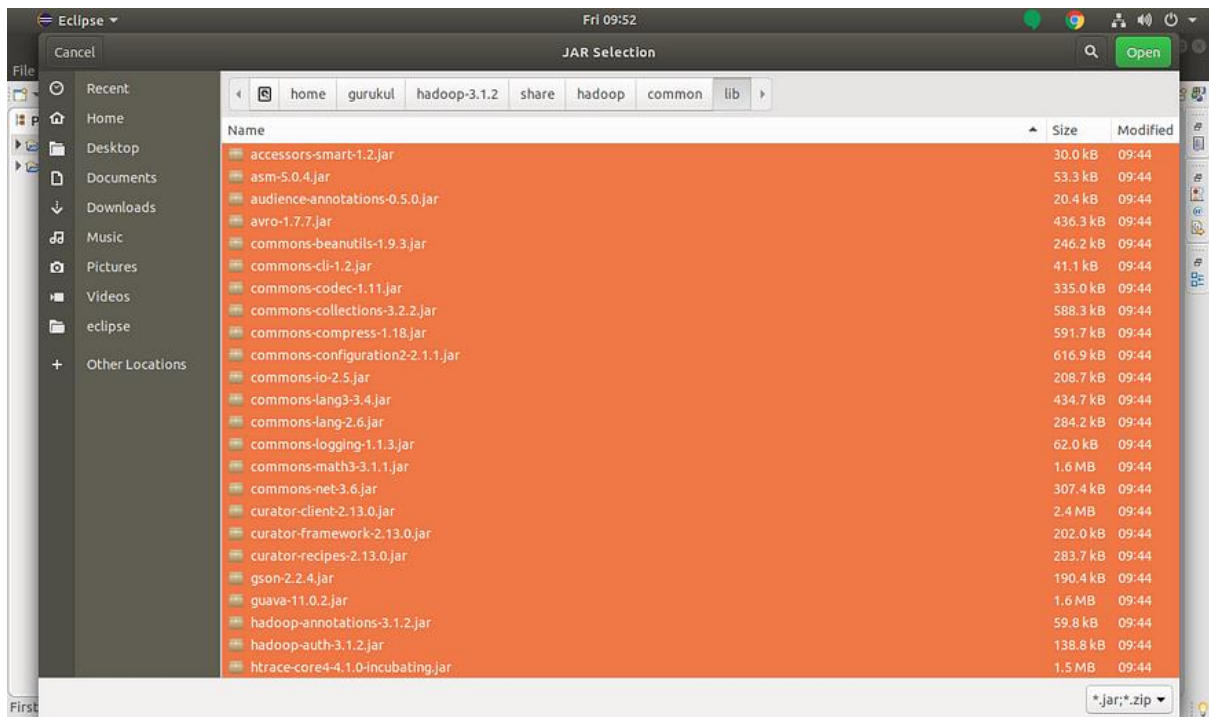


**Also, add common/lib libraries.**

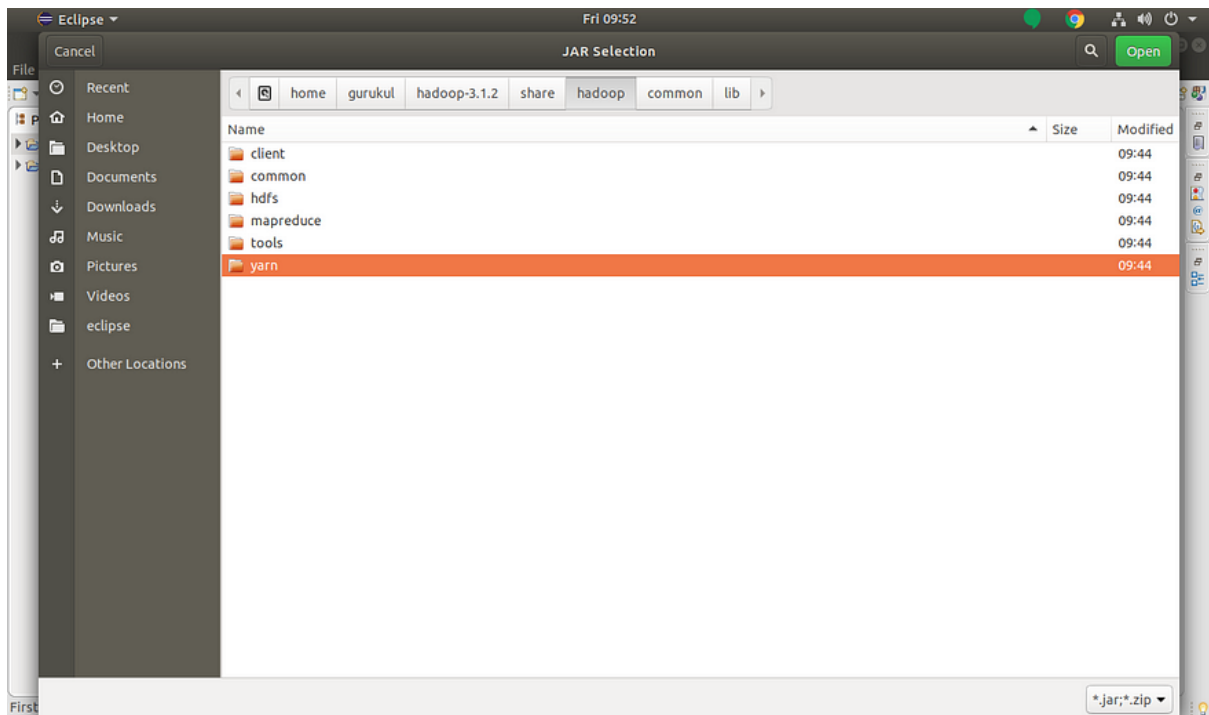




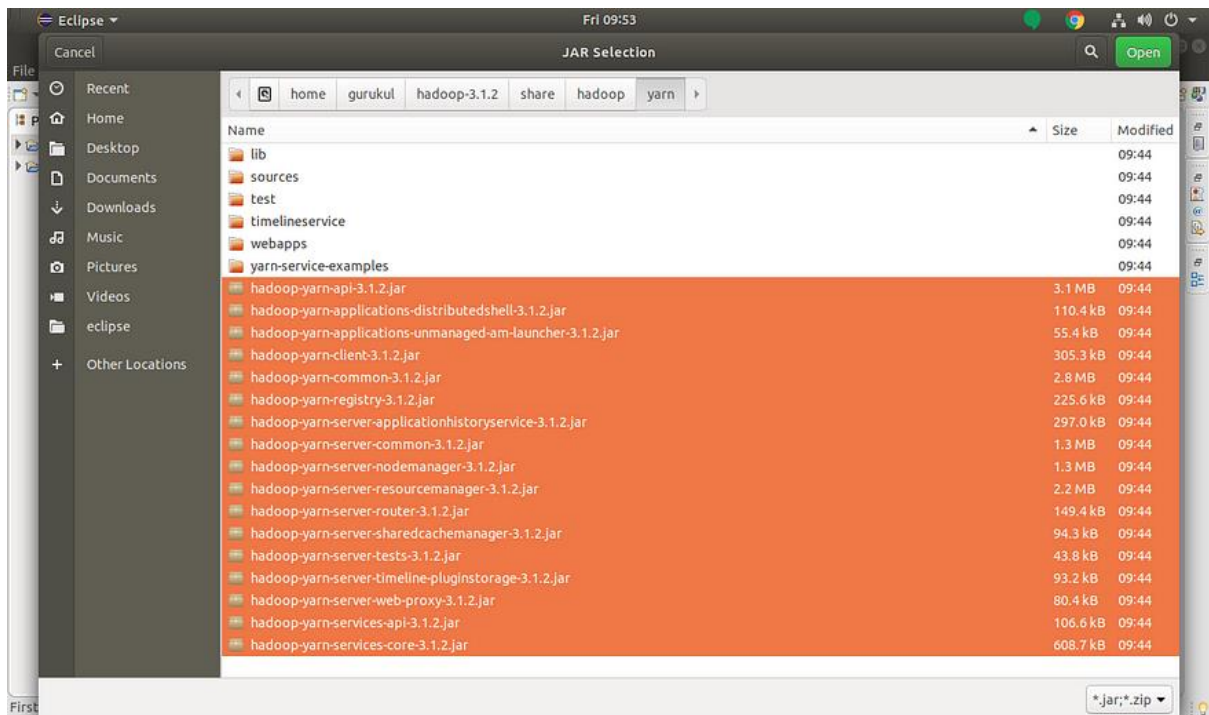
Select all common/lib jars and click Open.



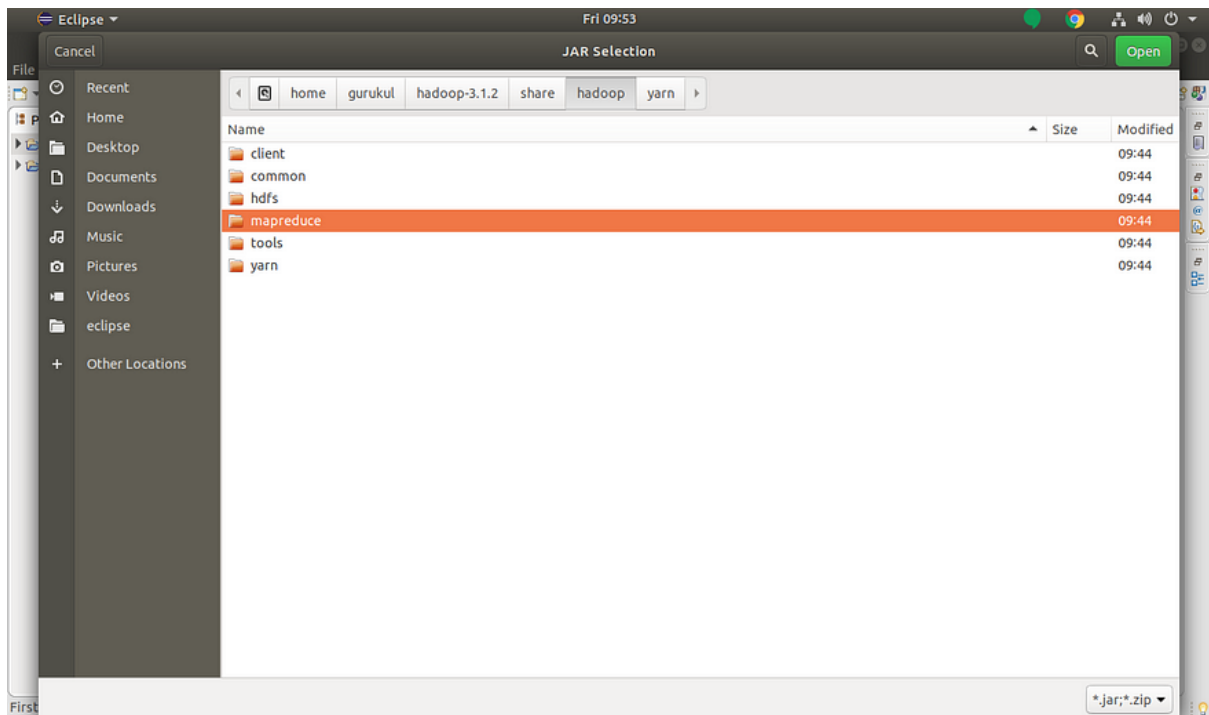
C. Add yarn jar files.



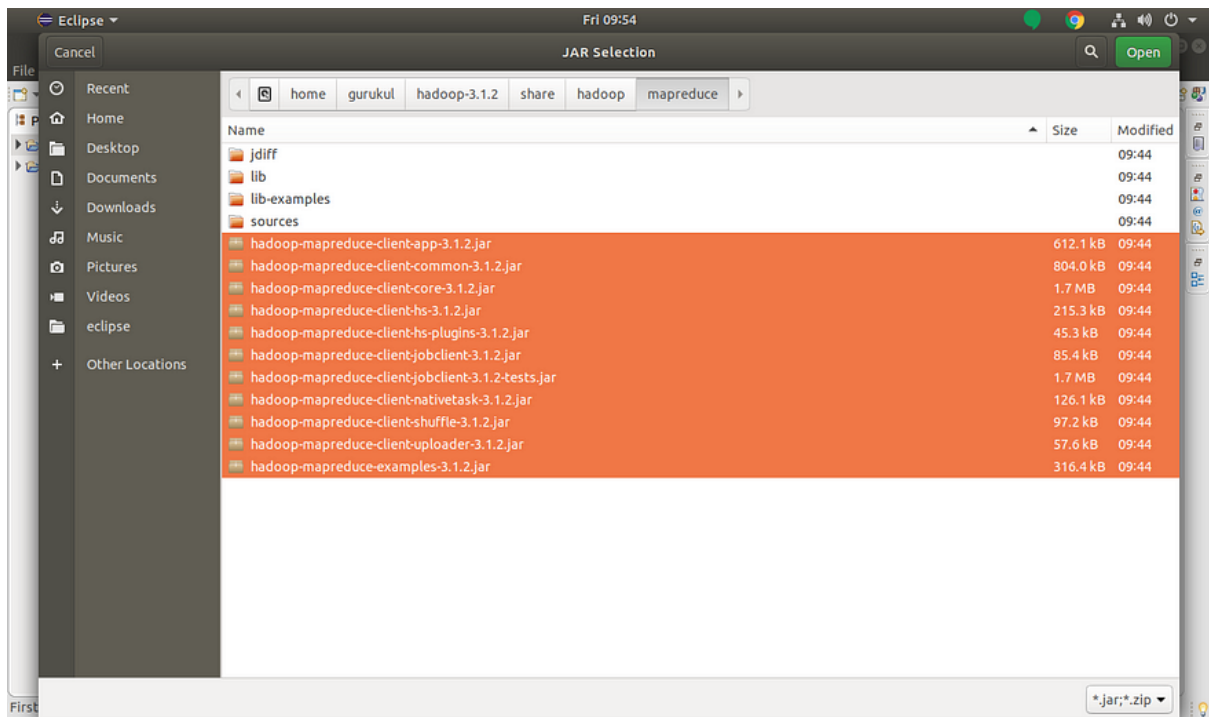
Select yarn jar files and then select Open.



D. Add MapReduce jar files.

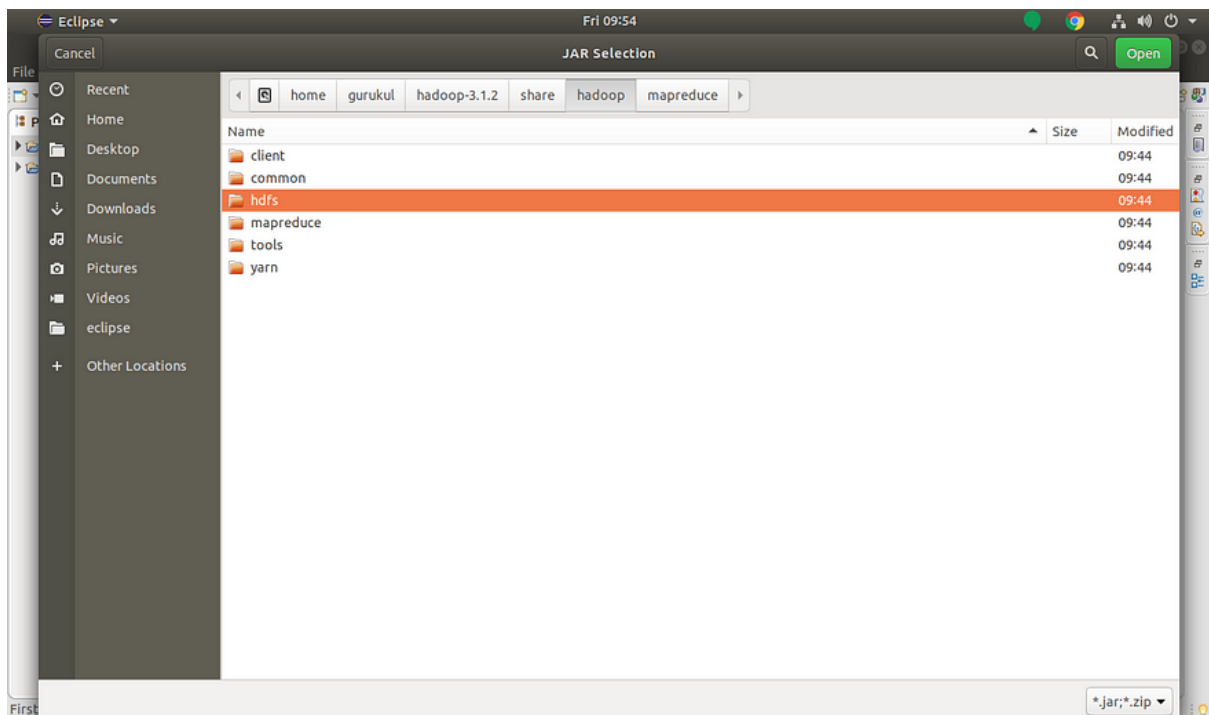


Select [MapReduce](#) jar files.

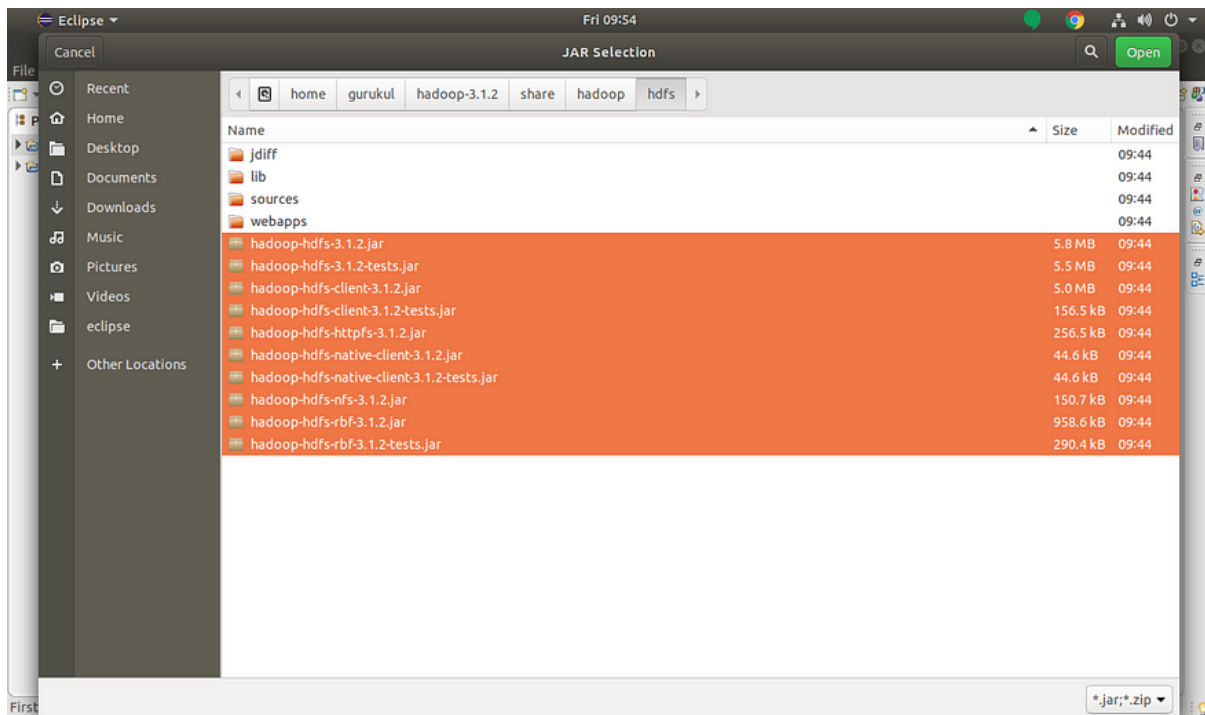


Click Open.

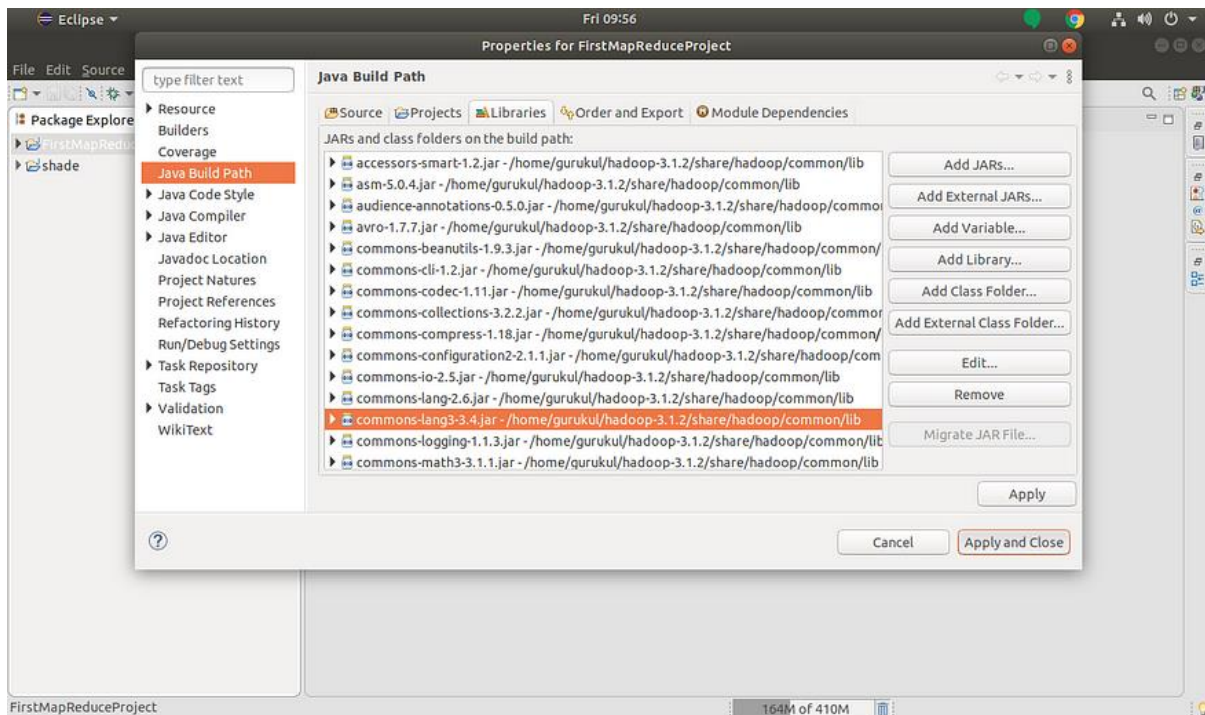
E. Add HDFS jar files.



Select HDFS jar files and click Open.



Click on Apply and Close to add all the Hadoop jar files.

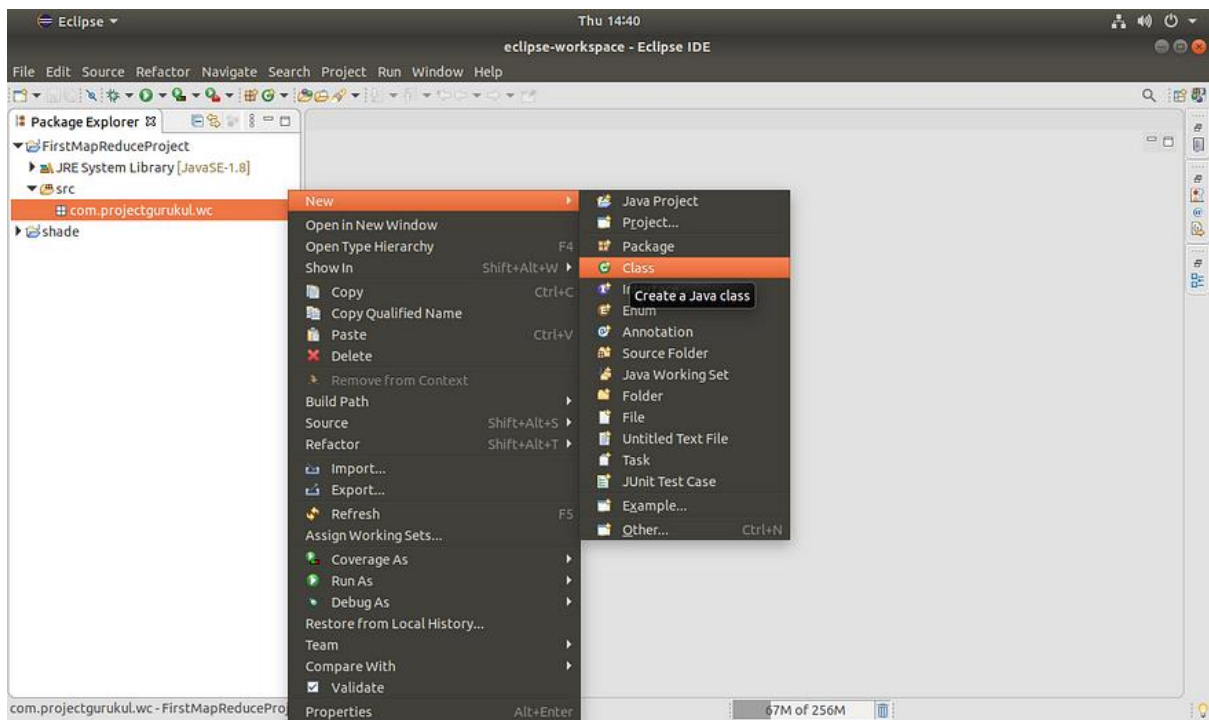


Now, we have added all required jar files in our project.

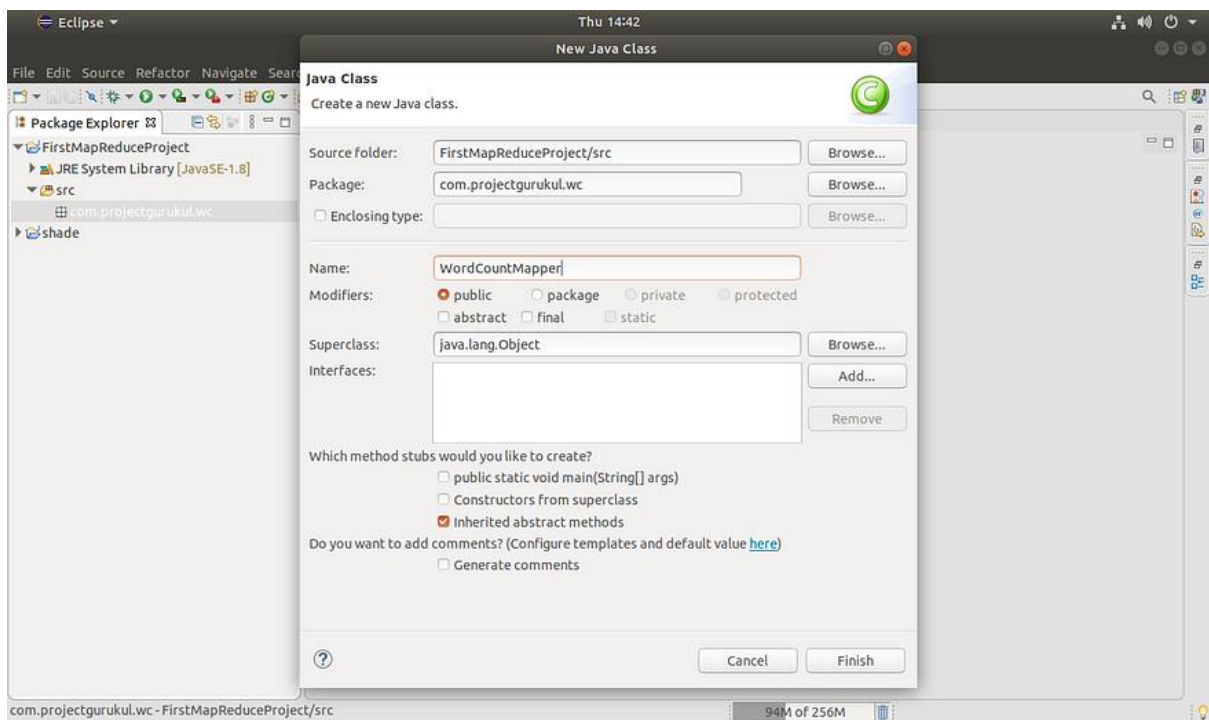
Step 5. Now create a new class that performs the map job.

Here in this article, WordCountMapper is the class for performing the mapping task.

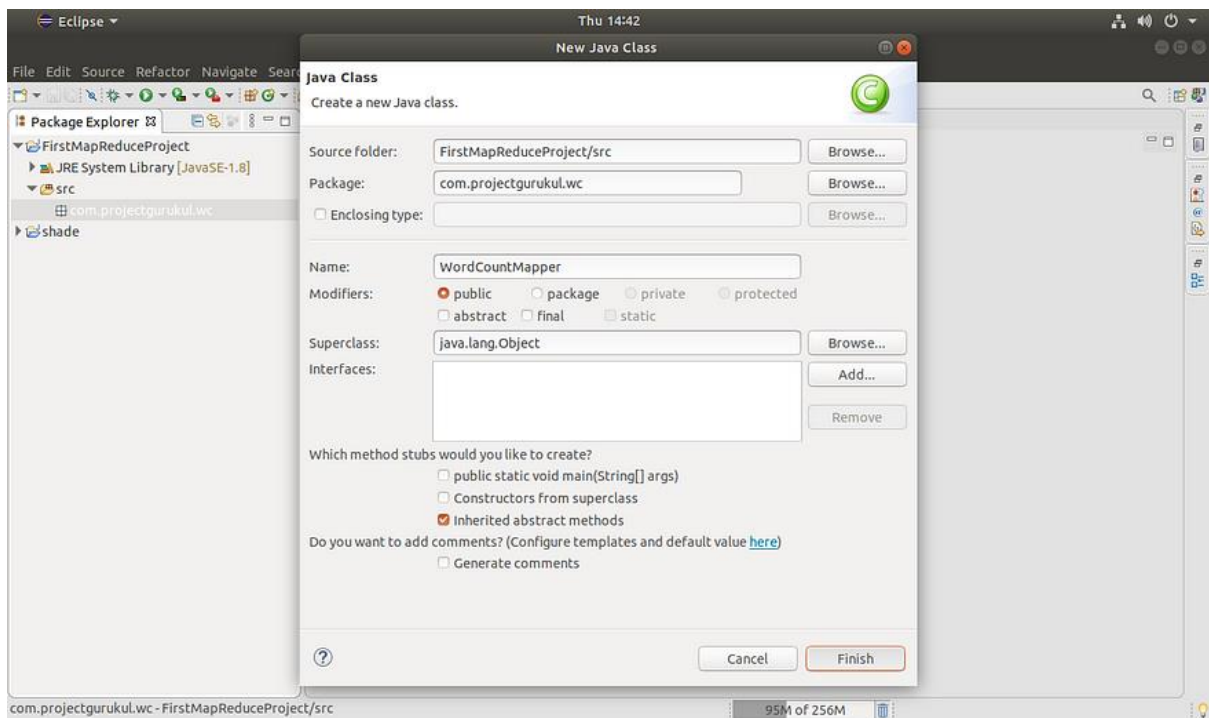
Right-Click on Package Name >> New >> Class



Provide the class name:



Click Finish.



**Step 6. Copy the below code in your class created above for the mapper.**

```
package com.projectgurukul.wc;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.LongWritable;

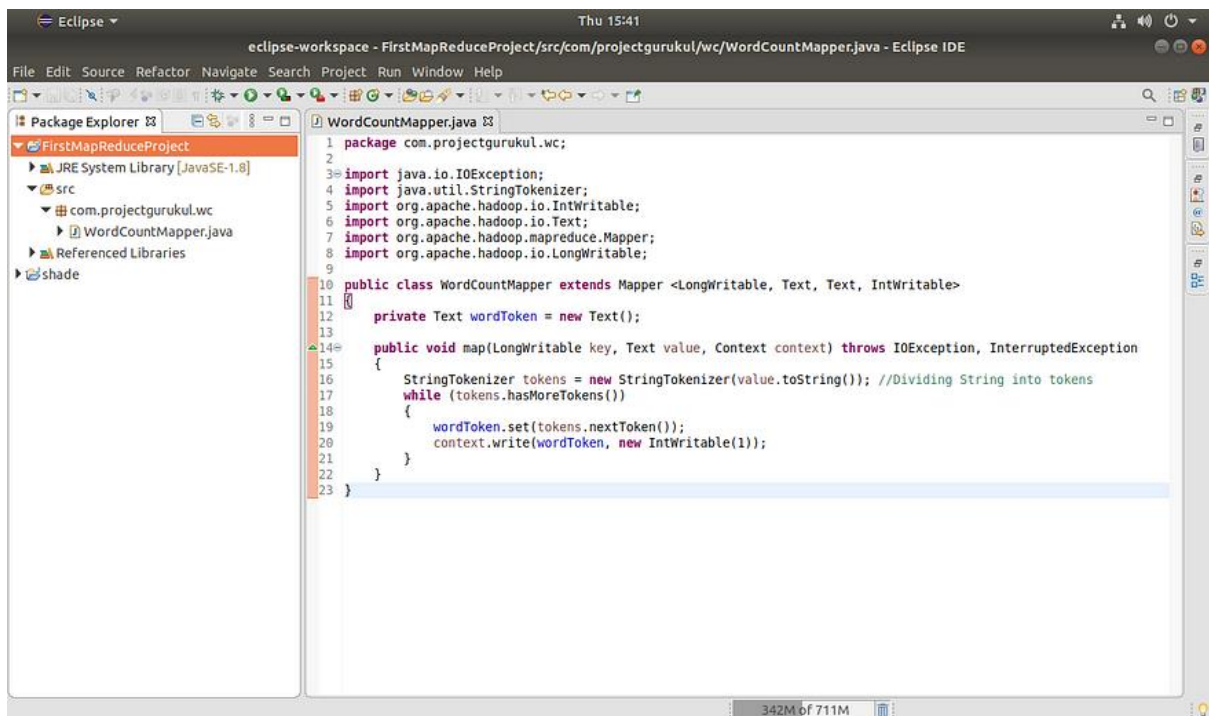
public class WordCountMapper extends Mapper <LongWritable, Text, Text, IntWritable>{
    private Text wordToken = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException{

        StringTokenizer tokens = new StringTokenizer(value.toString()); //Dividing String into tokenswhile
        (tokens.hasMoreTokens()){

            wordToken.set(tokens.nextToken());context.write(wordToken, new IntWritable(1));}}}
```

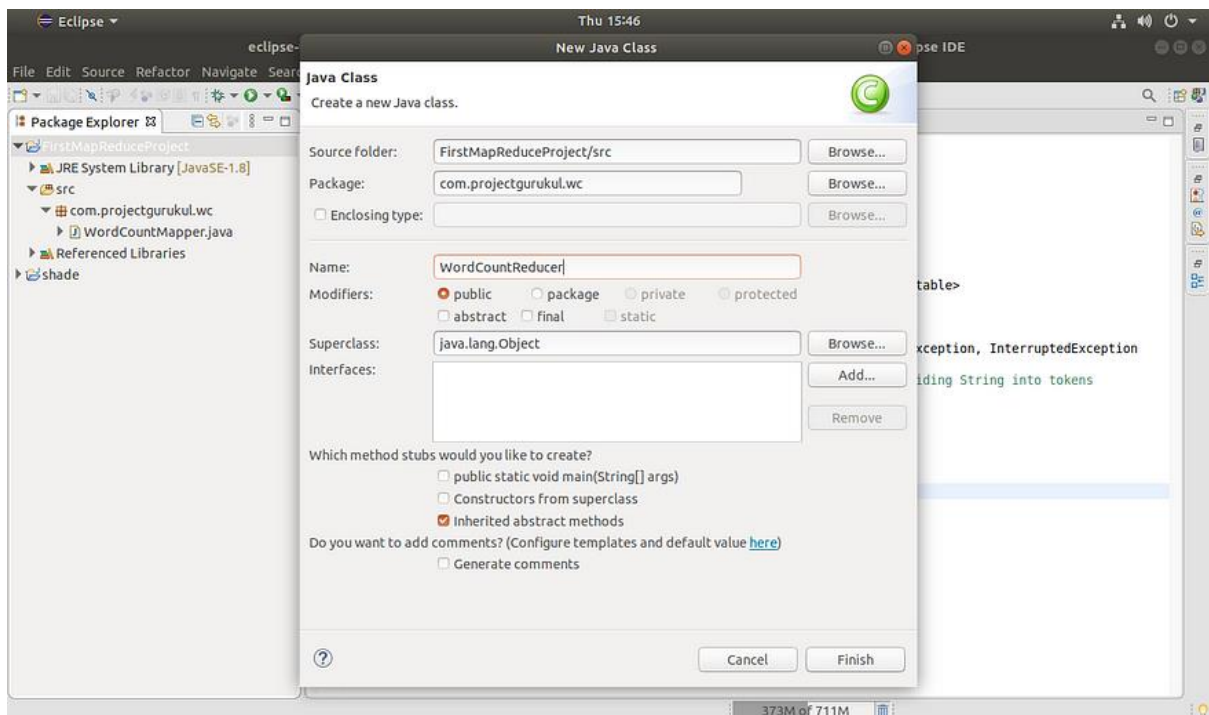




Press Ctrl+S to save the code.

**Step 7. Now create another class (in the same way as we used above), for creating a class that performs the reduce job.**

Here in this article, WordCountReducer is the class to perform the reduce task.



Click Finish.

**Step 8. Copy the below code in your class created above for the reducer.**

```

package com.projectgurukul.wc;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;import org.apache.hadoop.mapreduce.Reducer;

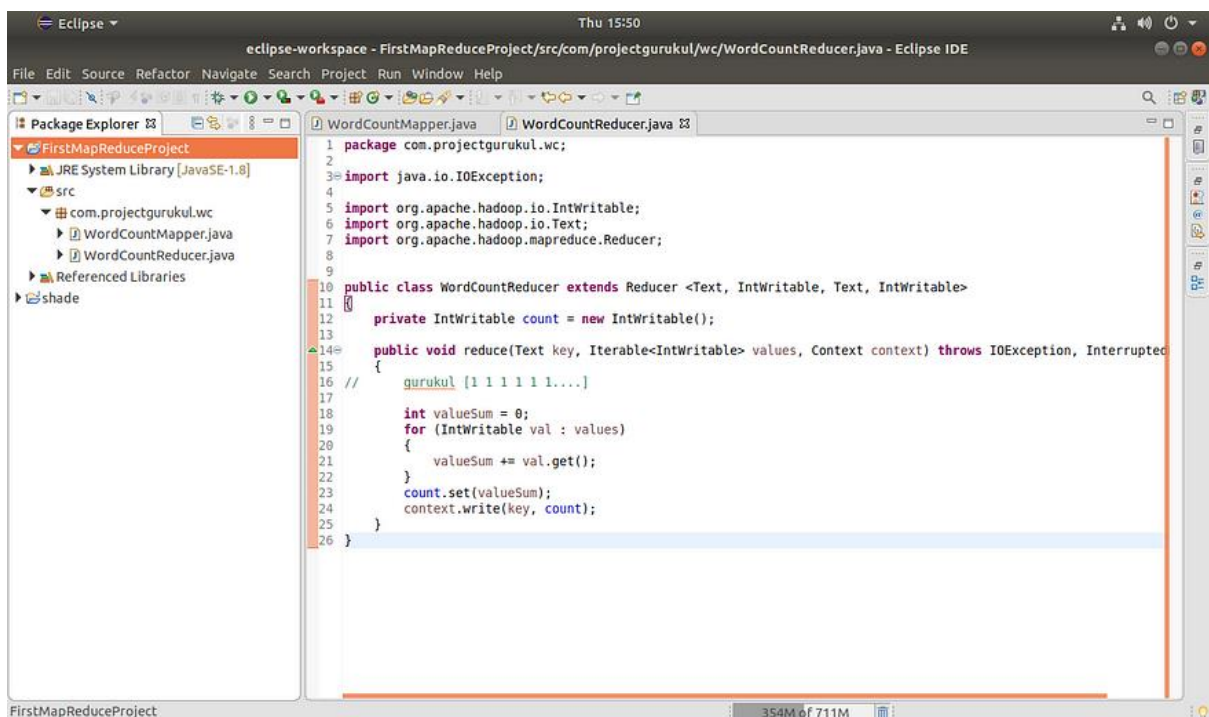
public class WordCountReducer extends Reducer <Text, IntWritable, Text, IntWritable>{

private IntWritable count = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException{

// gurukul [1 1 1 1 1 1....]int valueSum = 0;for (IntWritable val : values){
valueSum += val.get();}count.set(valueSum);context.write(key, count);}}

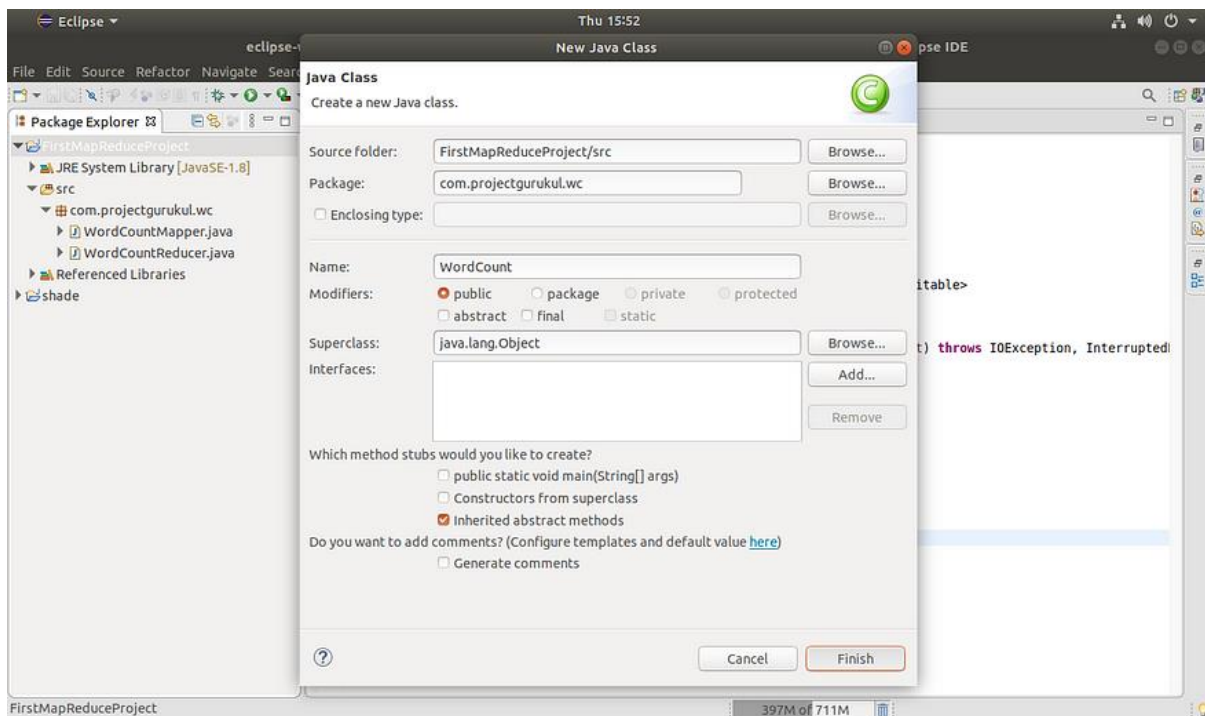
```



Press Ctrl+S to save the code.

**Step 9. Now create the driver class, which contains the main method. Here in this article, the driver class for the project is named “WordCount”.**





Click Finish.

**Step 10. Copy the below code in your driver class, which contains the main method.**

```
package com.projectgurukul.wc;

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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

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

        Configuration conf = new Configuration();
        String[] pathArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (pathArgs.length < 2) {

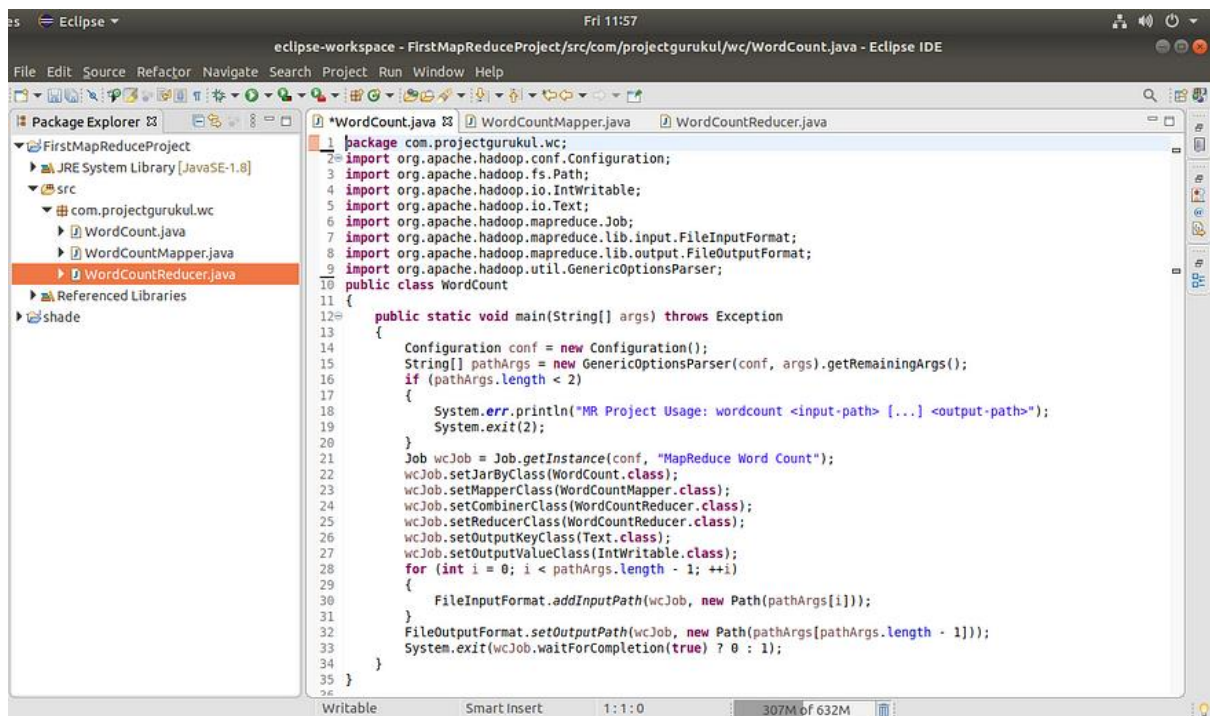
            System.err.println("MR Project Usage: wordcount <input-path> [...] <output-path>");
            System.exit(2);
        }

        Job wcJob = Job.getInstance(conf, "MapReduce WordCount");
        wcJob.setJarByClass(WordCount.class);
        wcJob.setMapperClass(WordCountMapper.class);
        wcJob.setCombinerClass(WordCountReducer.class);
        wcJob.setReducerClass(WordCountReducer.class);
        wcJob.setOutputKeyClass(Text.class);
        wcJob.setOutputValueClass(IntWritable.class);

        for (int i = 0; i < pathArgs.length - 1; ++i) {

            FileInputFormat.addInputPath(wcJob, new Path(pathArgs[i]));
        }

        FileOutputFormat.setOutputPath(wcJob, new Path(pathArgs[pathArgs.length - 1]));
        System.exit(wcJob.waitForCompletion(true) ? 0 : 1);
    }
}
```

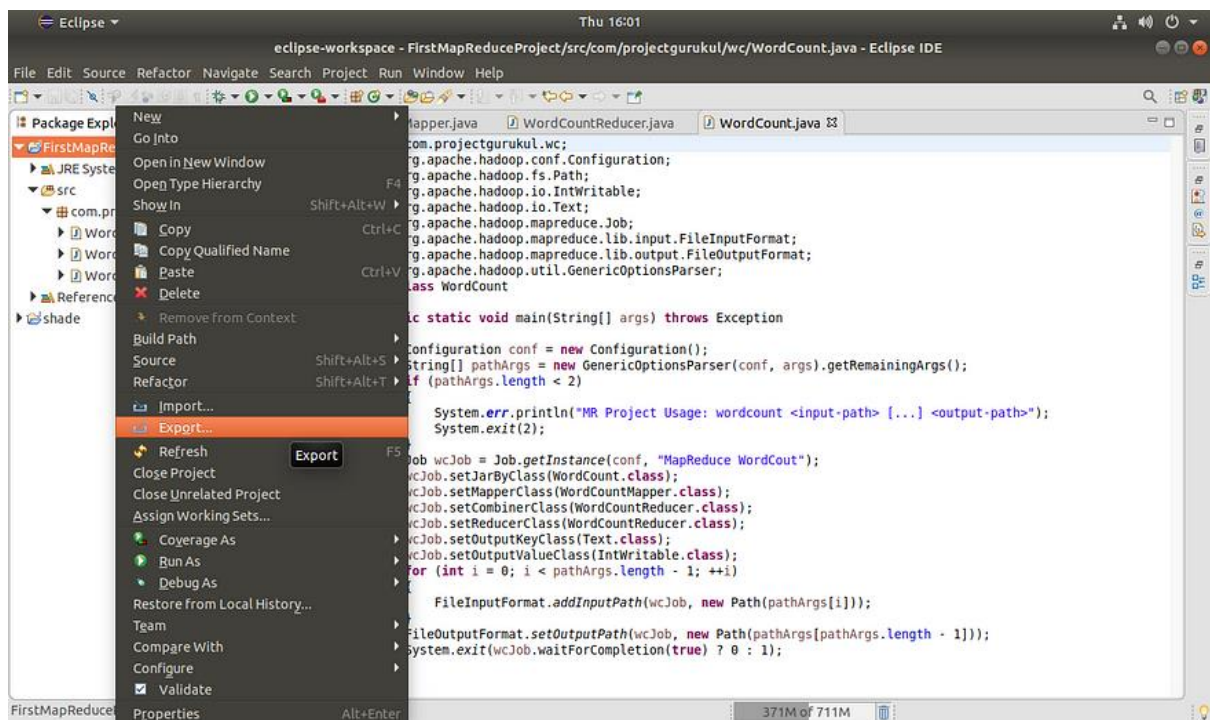


Press **Ctrl+S** to save the Code.

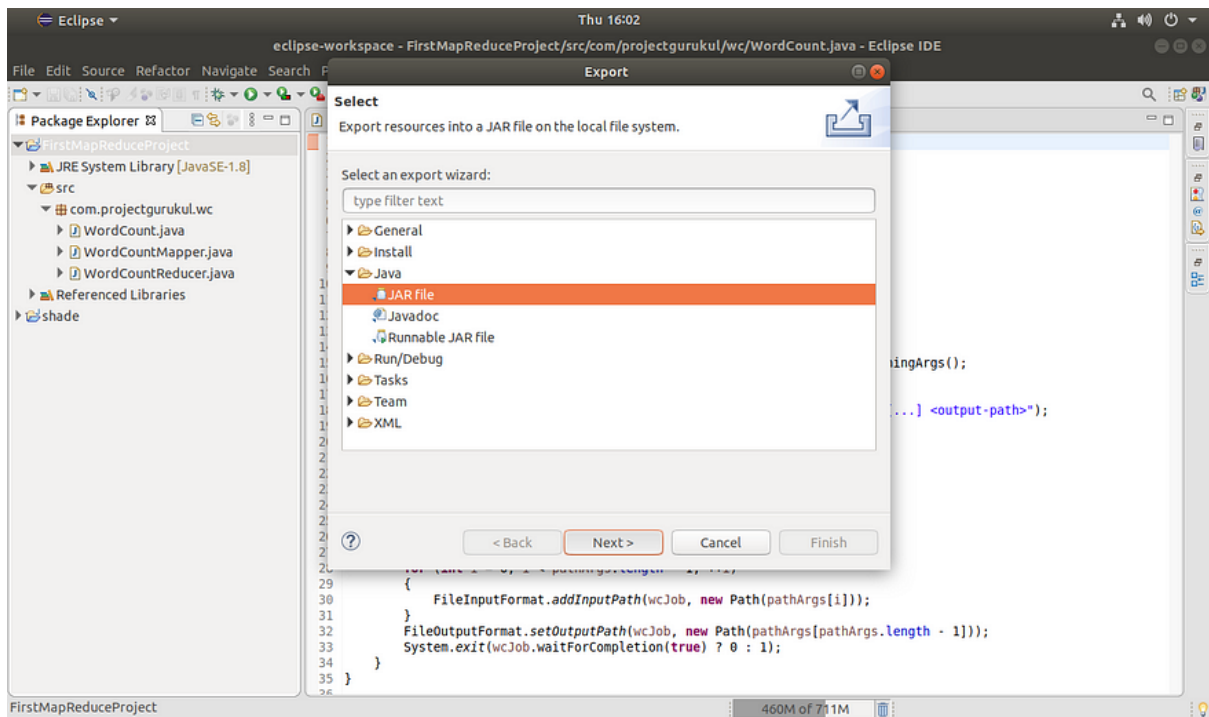
## Step 11. Creating the Jar File of the Project

Before running created Hadoop MapReduce word count application, we have to create a jar file.

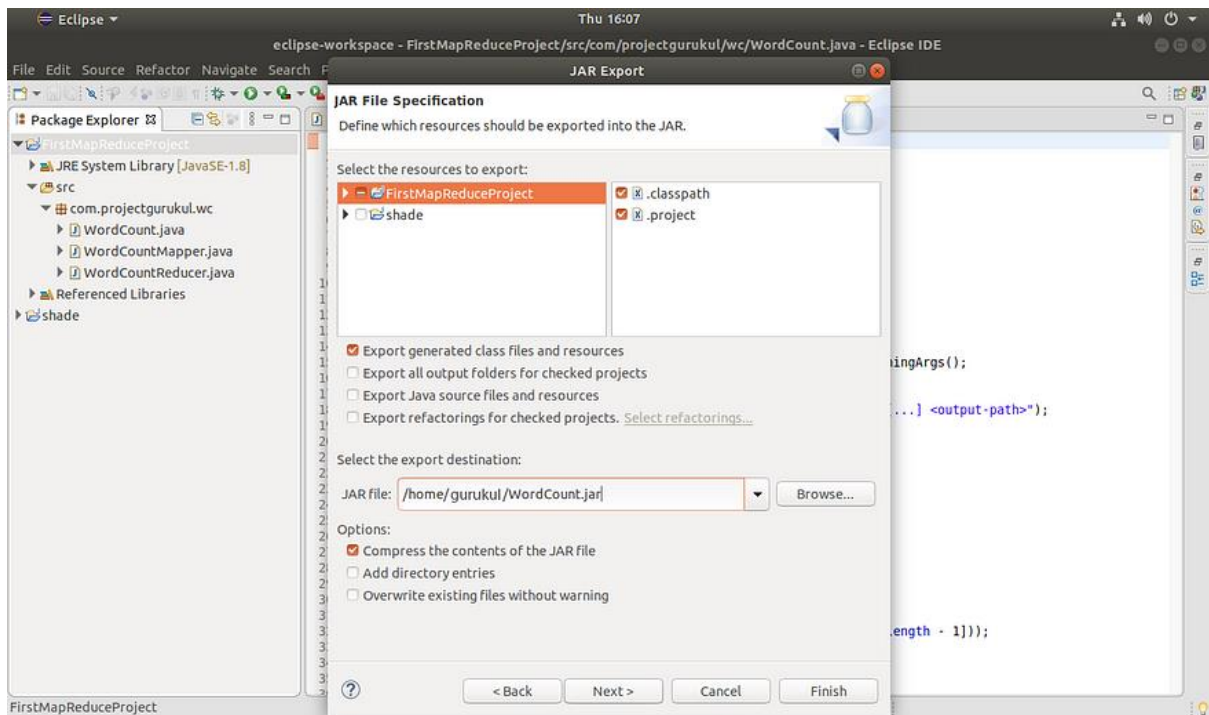
To do so Right-click on project name >> Export.



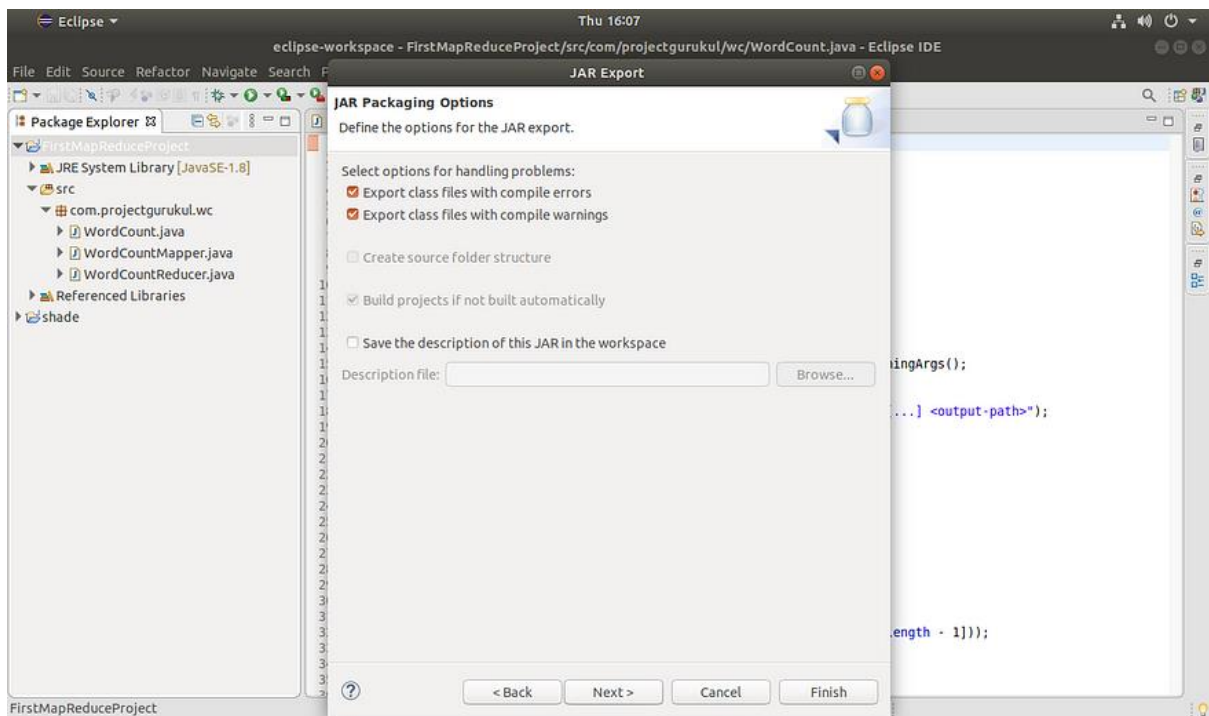
Select the JAR file option. Click Next.



Provide the Jar file name:



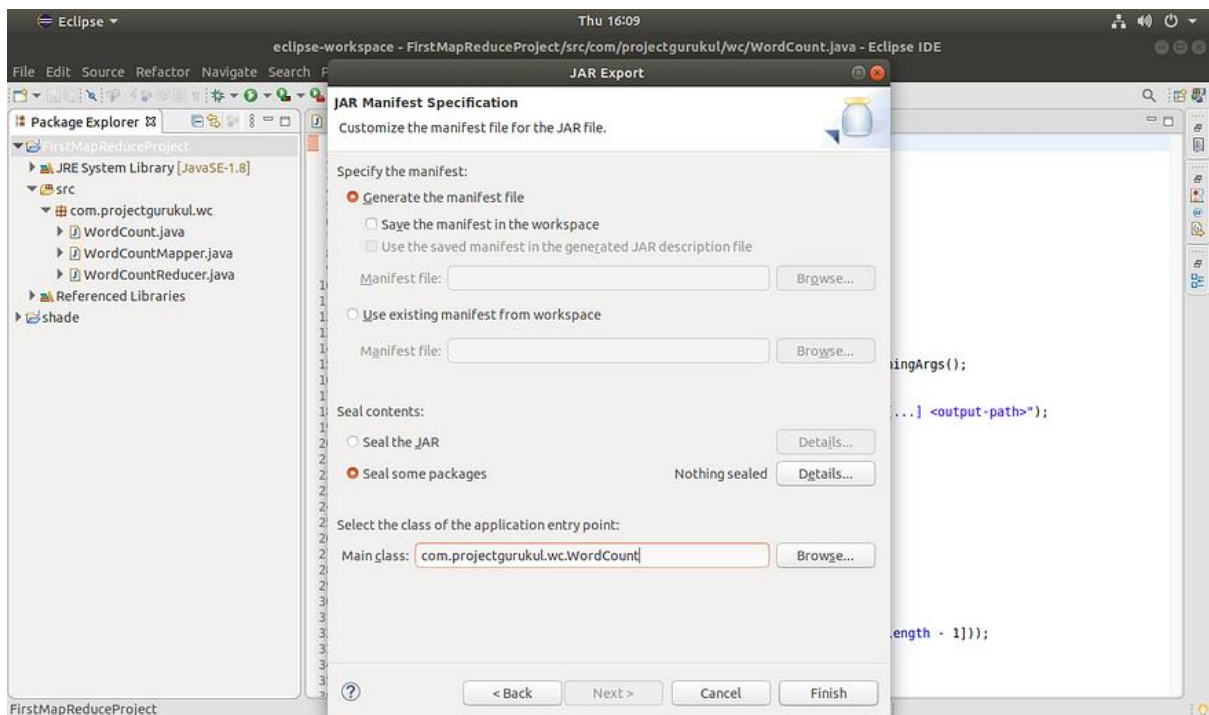
Click Next.



Click Next.

Now select the class of the application entry point.

Here in this Hadoop MapReduce Project article, the class for the application entry point is the WordCount class.



Click Finish.

**Step 12. Execute the Hadoop MapReduce word count application using the below execution command.**

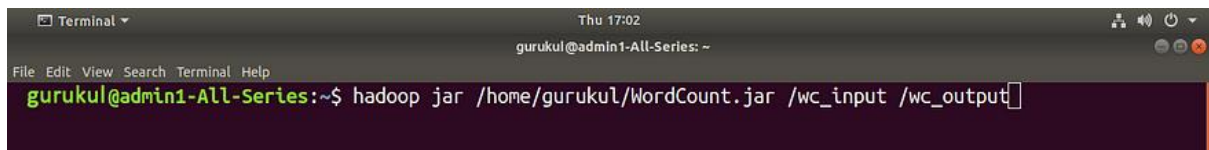


hadoop jar <project jar file path> <input file path> <output directory>

hadoop jar /home/gurukul/WordCount.jar /wc\_input /wc\_output

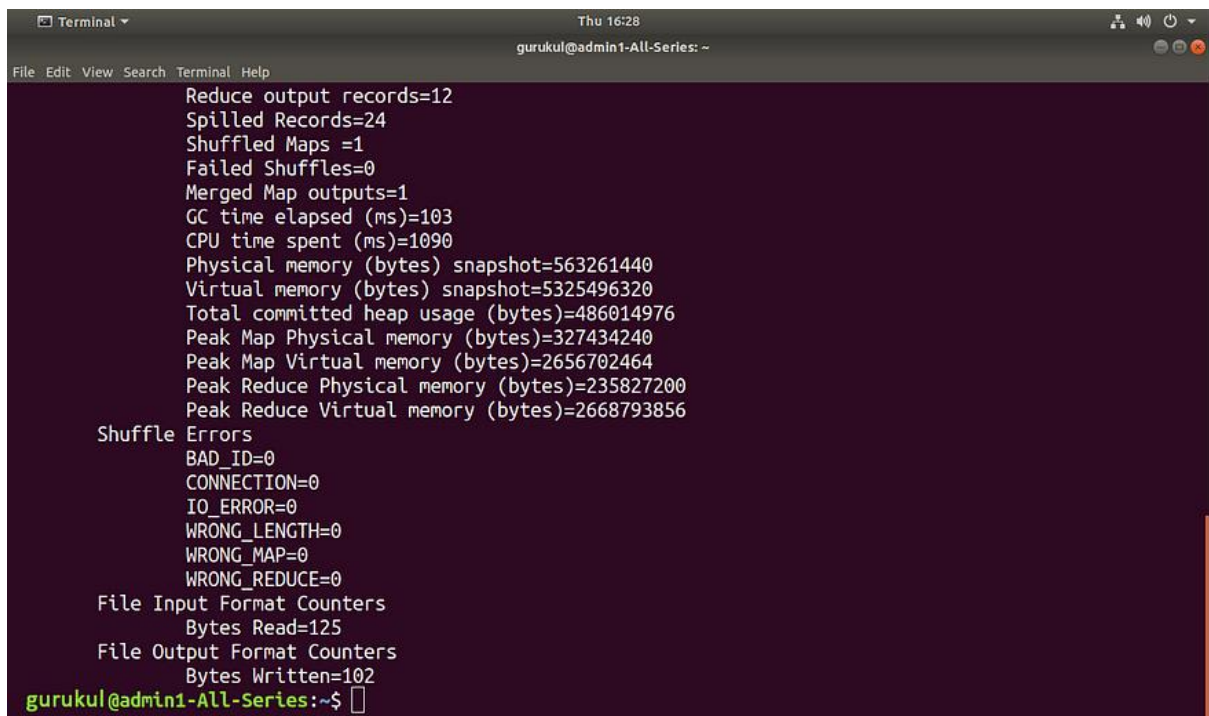
Here in this command,

- <project jar file path> is the path of the jar file of the project created above.
- <input file path> is the file in HDFS, which is input to the Hadoop MapReduce Word Count Project.
- <output directory> is the directory where the output of the Hadoop MapReduce WordCount program is going to be stored.

A terminal window titled 'Terminal' with a dark background. The prompt is 'gurukul@admin1-All-Series: ~'. The command 'hadoop jar /home/gurukul/WordCount.jar /wc\_input /wc\_output' has been entered and is followed by a cursor.

```
Terminal Thu 17:02
gurukul@admin1-All-Series: ~
File Edit View Search Terminal Help
gurukul@admin1-All-Series:~$ hadoop jar /home/gurukul/WordCount.jar /wc_input /wc_output
```

This will start the execution of MapReduce job

A terminal window titled 'Terminal' with a dark background. The prompt is 'gurukul@admin1-All-Series: ~'. The output of the Hadoop MapReduce job is displayed, showing various statistics and counters.

```
Terminal Thu 16:28
gurukul@admin1-All-Series: ~
File Edit View Search Terminal Help
Reduce output records=12
Spilled Records=24
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=103
CPU time spent (ms)=1090
Physical memory (bytes) snapshot=563261440
Virtual memory (bytes) snapshot=5325496320
Total committed heap usage (bytes)=486014976
Peak Map Physical memory (bytes)=327434240
Peak Map Virtual memory (bytes)=2656702464
Peak Reduce Physical memory (bytes)=235827200
Peak Reduce Virtual memory (bytes)=2668793856
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=125
File Output Format Counters
Bytes Written=102
gurukul@admin1-All-Series:~$
```

Now we have run the Map Reduce job successfully. Let us now check the result.

### Step 13. Browse the Hadoop MapReduce Word Count Project Output.

The output directory of the Project in HDFS contains two files: \_SUCCESS and part-r-00000

The output is present in the /part-r-00000 file.

You can browse the result using the below command.

hadoop fs -cat <output directory/part-r-00000>hadoop fs -cat /wc\_output/part-r-00000

```
Terminal Thu 16:29
gurukul@admin1-All-Series: ~
File Edit View Search Terminal Help
gurukul@admin1-All-Series:~$ hadoop fs -ls /wc_output
Found 2 items
-rw-r--r-- 1 dataflair supergroup 0 2020-02-20 16:27 /wc_output/_SUCCESS
-rw-r--r-- 1 dataflair supergroup 102 2020-02-20 16:27 /wc_output/part-r-00000
gurukul@admin1-All-Series:~$ hadoop fs -cat /wc_output/part-r-00000
Hadoop 1
MapReduce 2
This 2
article 2
best 1
first 1
gurukul 3
is 2
learn 1
project 1
provides 1
to 1
gurukul@admin1-All-Series:~$
```