

Hadoop MapReduce

Dr Amin Karami

a.karami@uel.ac.uk

www.aminkarami.com

CN7022 – Week 3
18 October 2019

Outline

- MapReduce Programming
- Develop Hadoop applications with Driver, Mapper, and Reducer classes using Java
- Optimizing MapReduce Performance by Partitioner and Combiner
- Exercise several MapReduce Programming with Java

Learning Outcomes

- Be able to explain MapReduce framework
- Understand MapReduce Programming
- Be able to write MapReduce Programs by three main classes: Driver, Mapper, and Reducer
- Understand optimization techniques for MapReduce by Partitioner and Combiner
- Be able to write MapReduce Programs with Java language in Eclipse IDE



MapReduce Concept

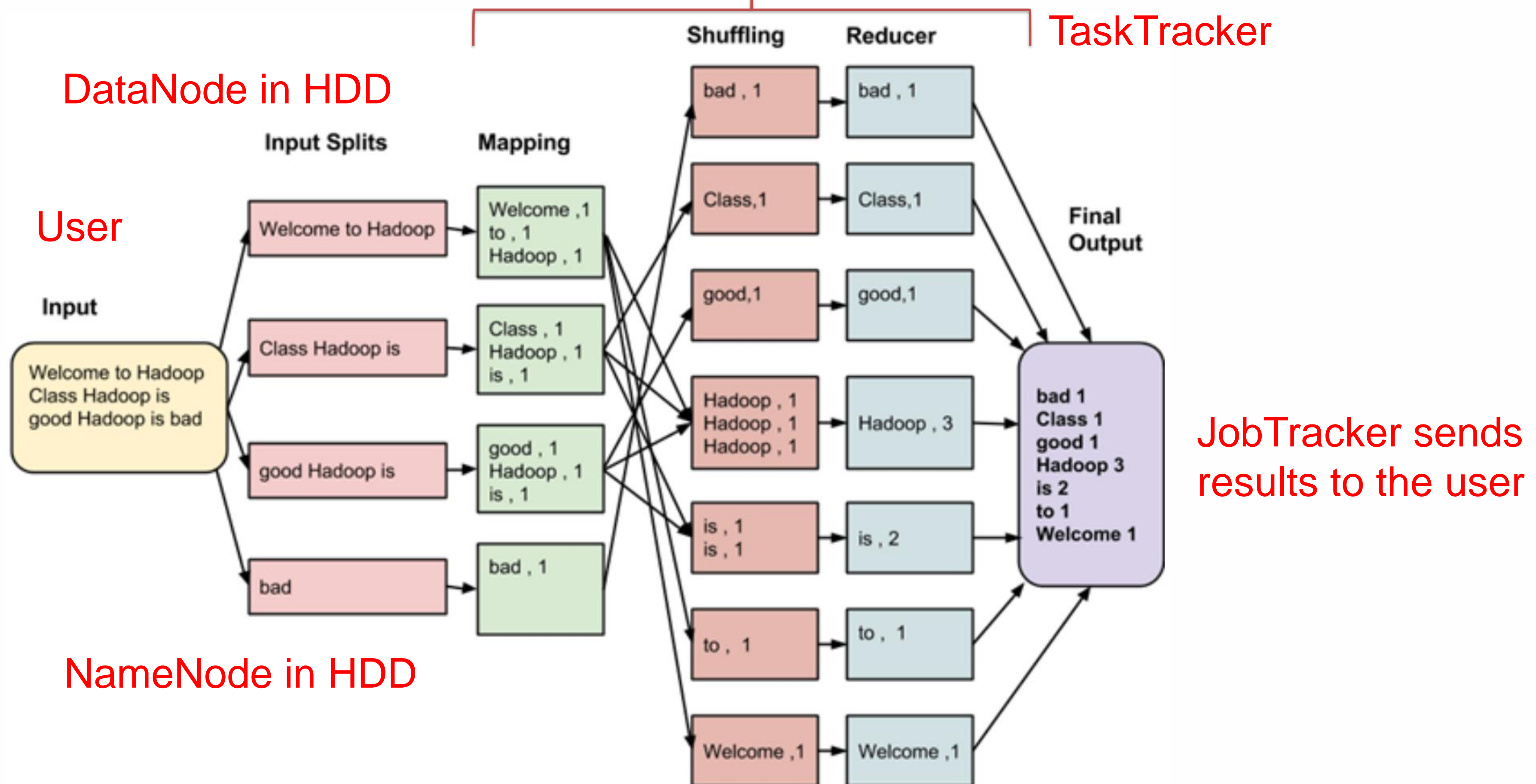
MapReduce is a programming concept for processing big data sets using parallel and distributed algorithm running on several machines on a Hadoop cluster. MapReduce is a combination of two tasks: **map and reduce**.

The map task simply **labels** thus producing groups that can be processed in a somewhat independent manner. Based on the labelling the data is then processed by worker nodes.

Reducer is a aggregator for the similar groups of labels created by map phase as key-value pairs.



MapReduce example



- **Word Count** example: **key-value** pairs are **<word, frequency>**



MapReduce phases

Mapper: Hadoop reads the split of the mapper line by line. Hadoop calls the method `map()` of the mapper for every line passing it as the key/value parameters.

Shuffle and sort: Hadoop collects all the different partitions received from the mappers and sort them by key.

Reducer: Hadoop reads the aggregated partitions line by line. locally, Hadoop calls the `reduce()` method on the reducer for every line of the input. The reducer computes its application logic and aggregate key/value pairs. Finally, Hadoop writes the pairs output to HDFS.

Output of MapReduce

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

hadoop

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores
1	0	0	1	0	0 B	8 GB	0 B	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Reboot
1	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% Queue
application_1508247403025_0002	root	WordCount.jar	MAPREDUCE	default	0	Tue Oct 17 06:38:24 -0700 2017	Tue Oct 17 06:38:47 -0700 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A	0.0

Showing 1 to 1 of 1 entries

localhost:50070/explorer.html#/user/root/Hamlet

Search

☆

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

/user/root/Hamlet

Go!

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Oct 17 06:38	1	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	82.02 KB	Oct 17 06:38	1	128 MB	part-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2017.



University of
East London

Data type in MapReduce: Writable

- Hadoop uses simple and efficient serialization protocol to serialize data between map and reduce phase and these are called **Writable(s)**.

Java primitive	Writable implementation
boolean	BooleanWritable
byte	ByteWritable
short	ShortWritable
int	IntWritable VIntWritable
float	FloatWritable
long	LongWritable VLongWritable
double	DoubleWritable

Java collection	Writable implementation
<i>array</i>	ArrayWritable ArrayPrimitiveWritable TwoDArrayWritable
Map	MapWritable
SortedMap	SortedMapWritable
<i>enum</i>	EnumSetWritable

Java class	Writable implementation
String	Text
byte[]	BytesWritable
Object	ObjectWritable
<i>null</i>	NullWritable



MapReduce in Java: requirements

- We are going to start MapReduce programming by Eclipse:

```
sudo apt-get install eclipse
```

- We need at least three java classes: **Driver**, **Mapper** and **Reducer**
- Then, we will optimize MapReduce performance by **Partitioner** and **Combiner** techniques.



The probable error in eclipse installation

- You may face with the following error:

E: Could not get lock /var/lib/dpkg/lock - open (11: Resource temporarily unavailable)

E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?

- Solution? Delete the lock files from cache

```
sudo rm /var/lib/apt/lists/lock
```

```
sudo rm /var/cache/apt/archives/lock
```

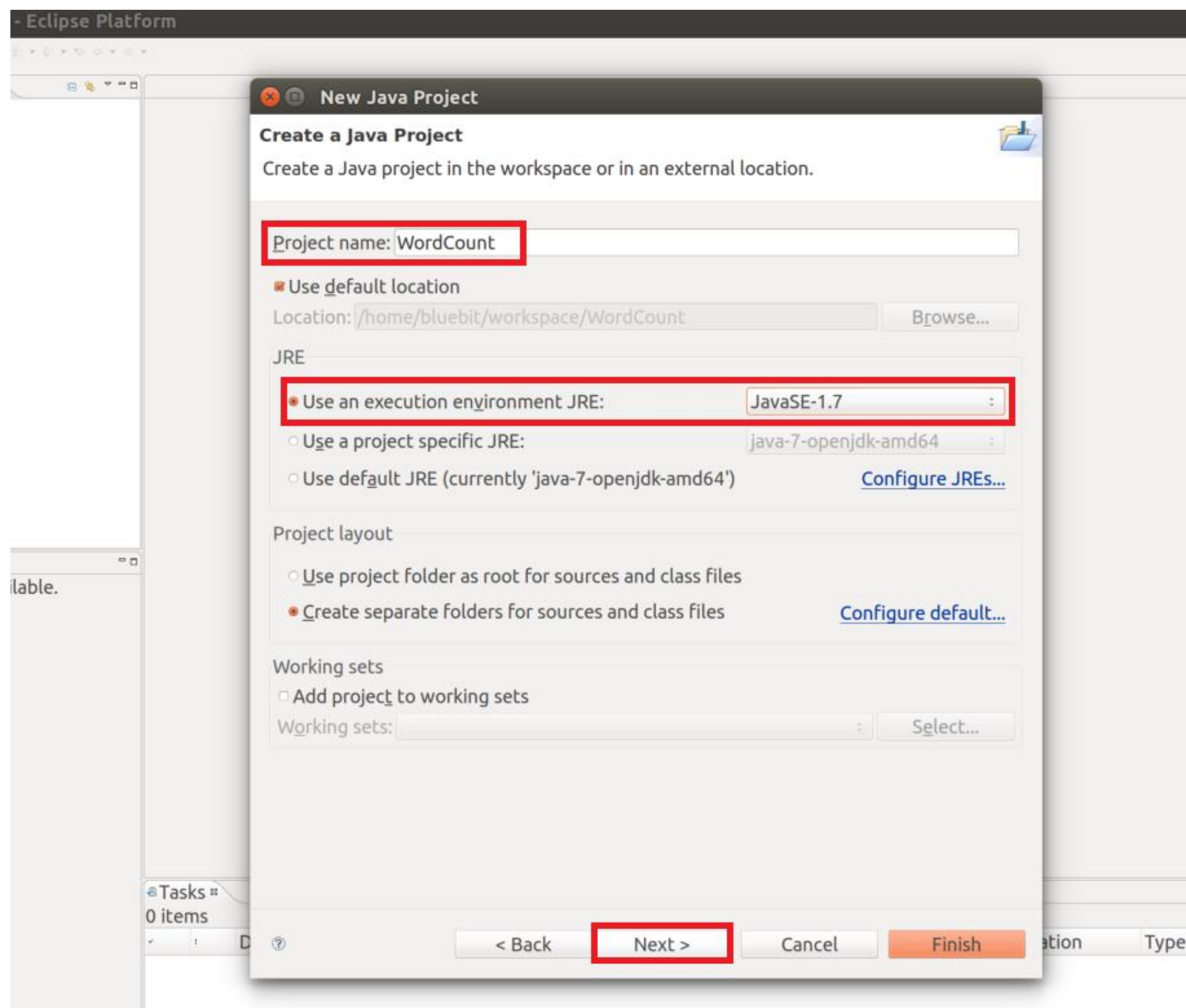
```
sudo rm /var/lib/dpkg/lock
```

- *Another way (not recommended):* `sudo killall apt apt-get`



Step 1: Create a Java Project

(1) Launch Eclipse (type `eclipse` in *cmd*), then create a New Project (`ctrl+N`) and name it as **WordCount**

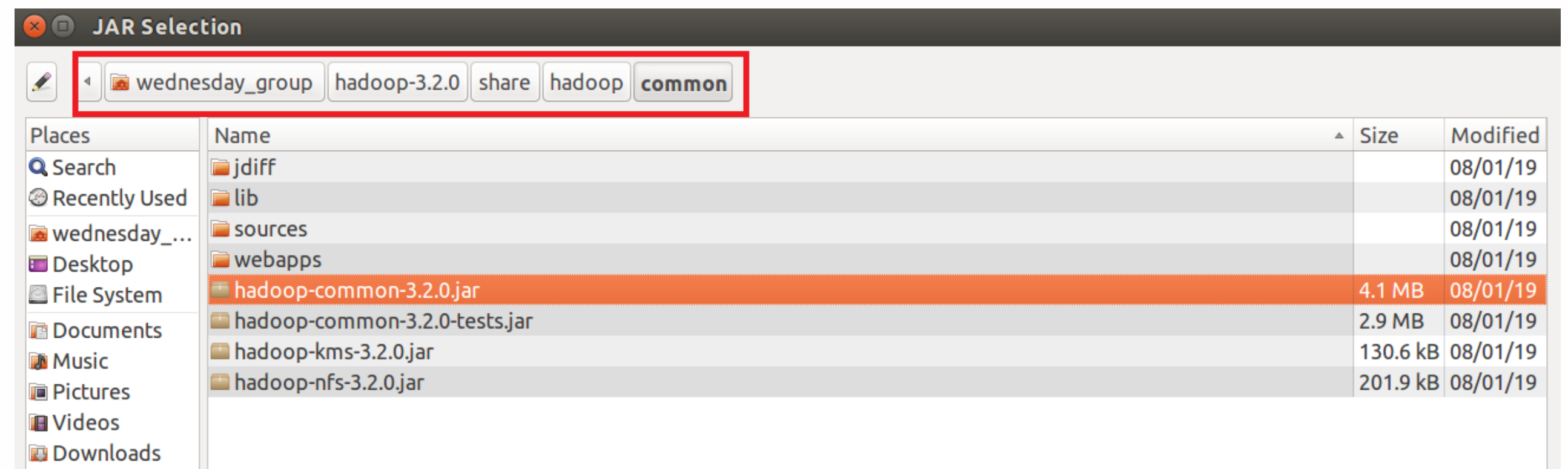
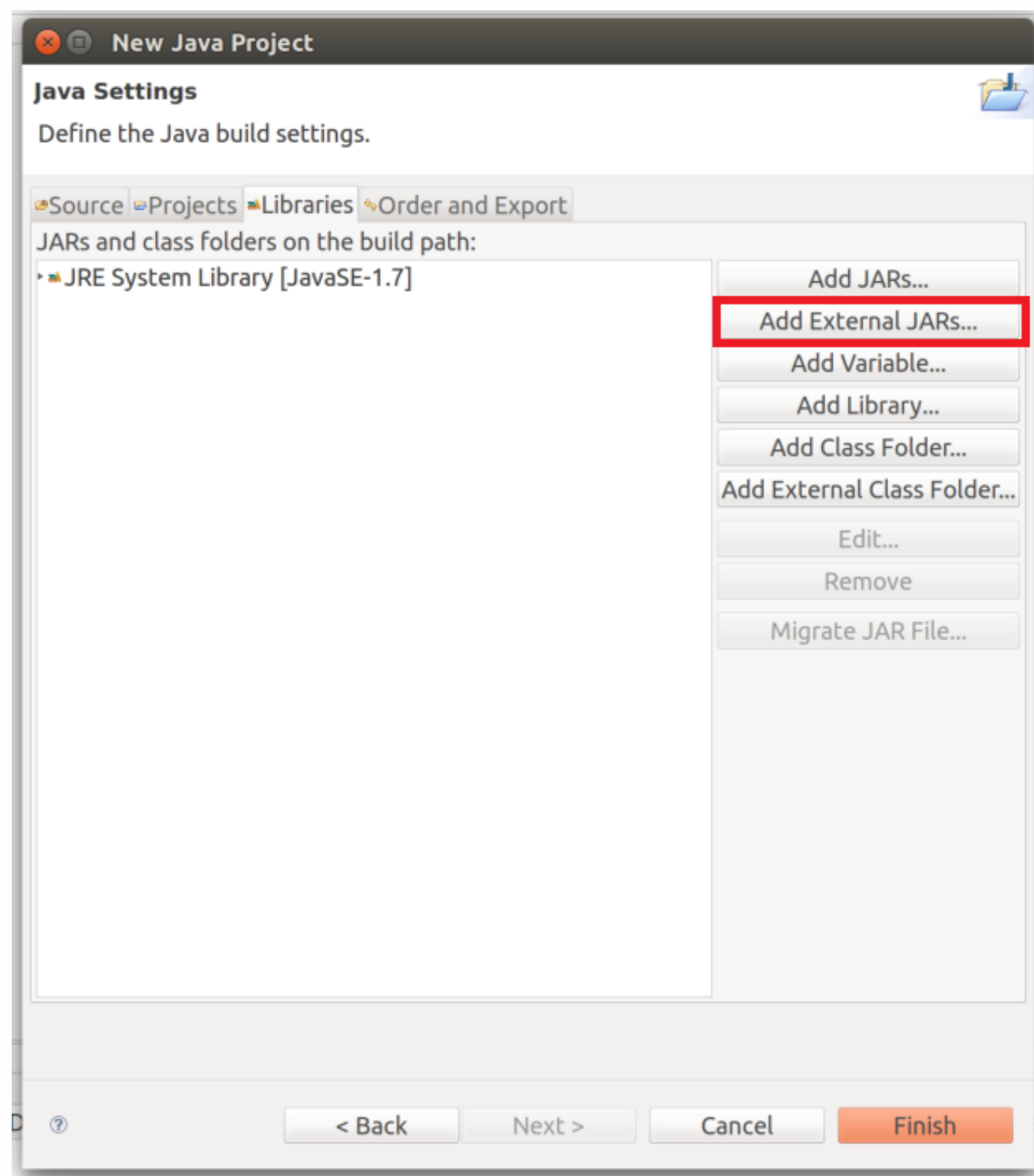


(2) Then, Press **Next**

Step 2: Add Hadoop libraries

- (3) Run **hadoop version** to find the source of jar files.
- Click on “*Add External JARs...*” from **Libraries** tab.
 - Find Hadoop common files from:

```
wednesday_group@uel-Desktop-VM: ~  
wednesday_group@uel-Desktop-VM:~$ hadoop version  
Hadoop 3.2.0  
Source code repository https://github.com/apache/hadoop.git -r e97acb3bd8f3befd27418996fa5d4b50bf2e17bf  
Compiled by sunilg on 2019-01-08T06:08Z  
Compiled with protoc 2.5.0  
From source with checksum d3f0795ed0d9dc378e2c785d3668f39  
This command was run using /home/wednesday_group/hadoop-3.2.0/share/hadoop/common/hadoop-common-3.2.0.jar  
wednesday_group@uel-Desktop-VM:~$
```

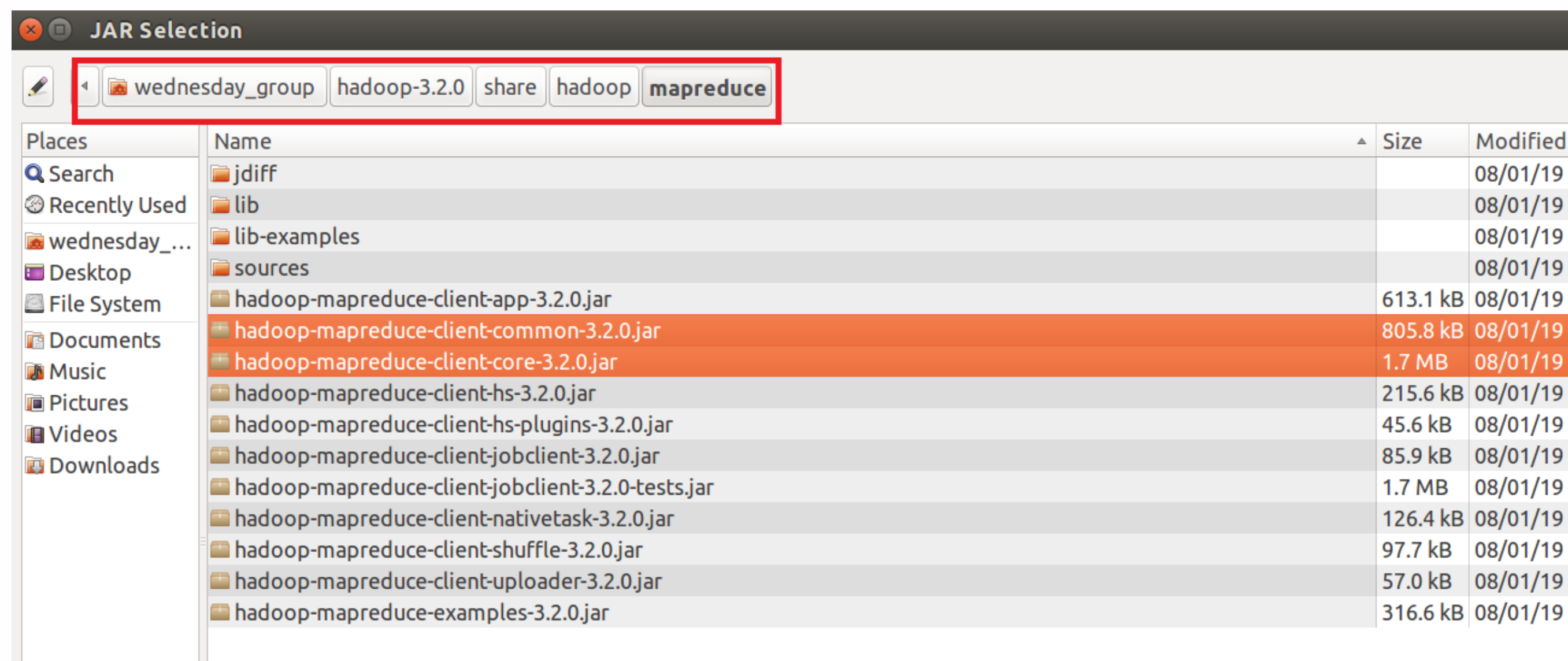


Step 3: Add JAR libraries for Hadoop

(4) Again, click on “*Add External JARs...*” from **Libraries** tab.

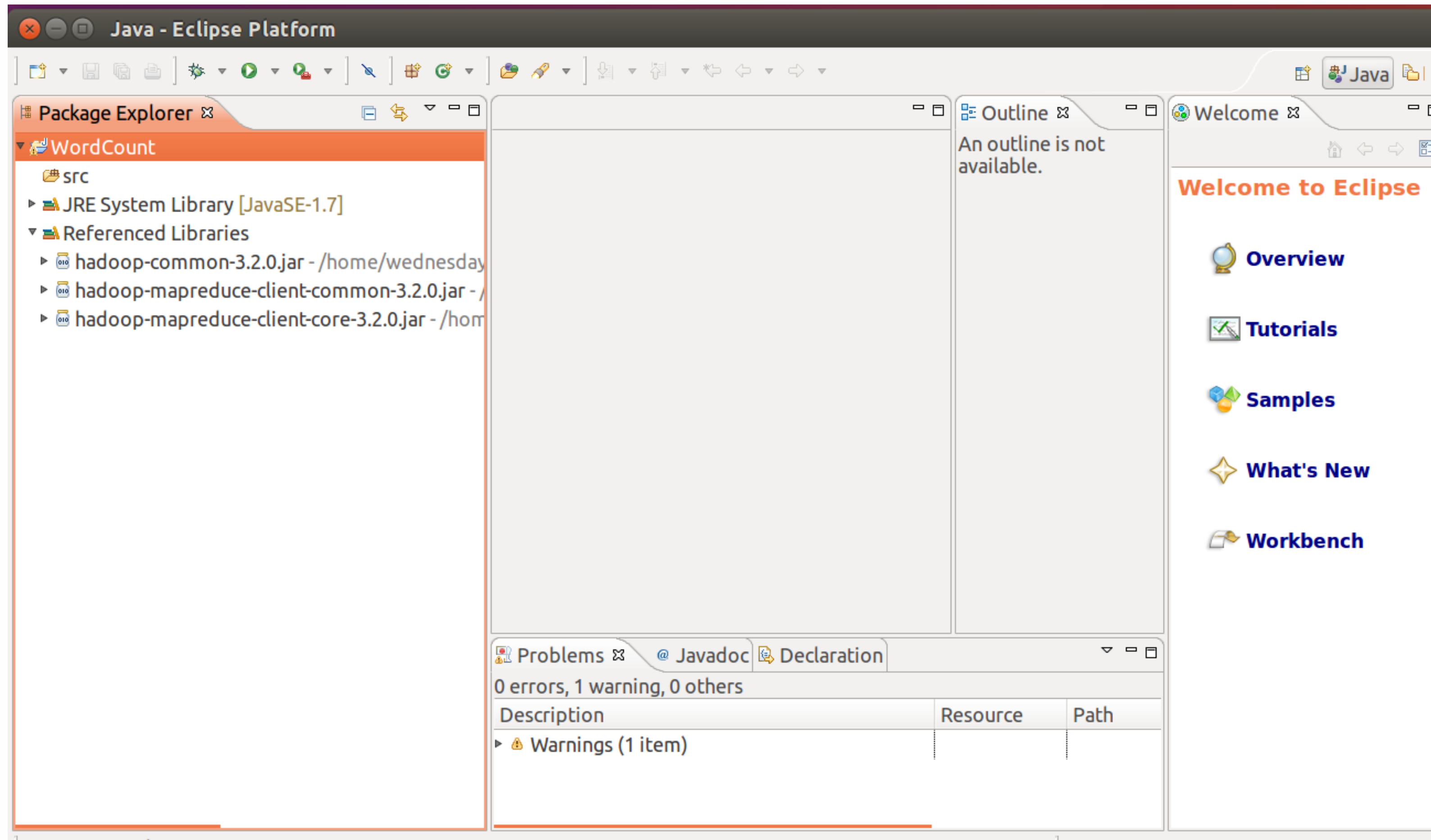
- Find JAR files from:

`$HADOOP_HOME$/share/hadoop/mapreduce`



(5) Press **Finish** button.

WordCount project in Eclipse



Step 4: Create 3 Java Classes

- Create 3 classes: Driver, Mapper, and Reducer.
- Right click on the project (**WordCount**), and select **New → Class**.

The screenshot shows the 'New Java Class' dialog box. The 'Name' field is highlighted with a red box and contains the text 'WordCount'. The 'Finish' button at the bottom right is also highlighted with a red box. Other fields include 'Source folder' (WordCount/src), 'Package' (default), 'Superclass' (java.lang.Object), and 'Which method stubs would you like to create?' with options for public static void main, constructors, and inherited abstract methods.

The screenshot shows the 'New Java Class' dialog box. The 'Name' field is highlighted with a red box and contains the text 'WordMapper'. The 'Finish' button at the bottom right is also highlighted with a red box. Other fields include 'Source folder' (WordCount/src), 'Package' (default), 'Superclass' (java.lang.Object), and 'Which method stubs would you like to create?' with options for public static void main, constructors, and inherited abstract methods.

The screenshot shows the 'New Java Class' dialog box. The 'Name' field is highlighted with a red box and contains the text 'WordReducer'. The 'Finish' button at the bottom right is also highlighted with a red box. Other fields include 'Source folder' (WordCount/src), 'Package' (default), 'Superclass' (java.lang.Object), and 'Which method stubs would you like to create?' with options for public static void main, constructors, and inherited abstract methods.

Driver Class

```
WordCount.java WordMapper.java WordReducer.java

public class WordCount extends Configured implements Tool{

    @Override
    public int run(String[] arg0) throws Exception {
        // TODO Auto-generated method stub
        if (arg0.length < 2){
            System.out.println("Please enter Input and Output paths accordingly.");
            return -1;
        }

        JobConf conf = new JobConf(WordCount.class);

        FileInputFormat.setInputPaths(conf, new Path(arg0[0]));
        FileOutputFormat.setOutputPath(conf, new Path(arg0[1]));

        conf.setMapperClass(WordMapper.class);
        conf.setReducerClass(WordReducer.class);

        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        JobClient.runJob(conf);

        return 0;
    }

    public static void main(String args[]) throws Exception{
        int exitCode = ToolRunner.run(new WordCount(), args);
        System.exit(exitCode);
    }
}
```

Configure a job

Configure Input and Output

Define Mapper and Reducer Classes

Define Key/Value types in Mapper

Define Key/Value types in Reducer

Run Configuration

Run Job

Mapper Class

For WordCout program:

Key = Word

Value = Frequency

```
WordCount.java WordMapper.java WordReducer.java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WordMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable>{

    @Override
    public void map(LongWritable Key, Text Value,
        OutputCollector<Text, IntWritable> Output, Reporter r)
        throws IOException {

        String s = Value.toString();
        final IntWritable one = new IntWritable(1);

        for (String word:s.split(" ")){
            if (word.length() > 0){
                Output.collect(new Text(word), one);
            }
        }
    }
}
```

Input
(Key, Value)

Output
(Key, Value)

Initially, Key is *LongWritable* and 0;
Text is the words in each mapper.

Reporter is responsible for reporting
Mapper code to Driver Code

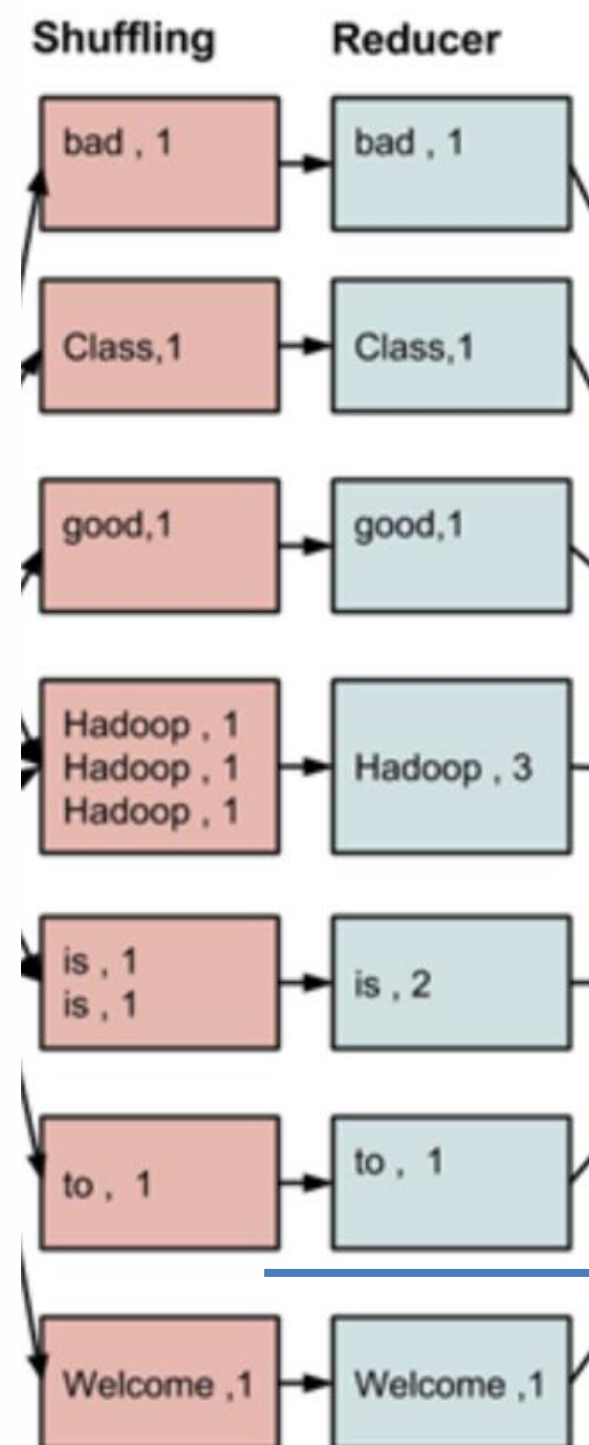
It is similar to “int one = 1” in ordinary use.

Separate each word with space.



University of
East London

Reducer Class



```
WordCount.java WordMapper.java WordReducer.java
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WordReducer extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text Key, Iterator<IntWritable> Value,
        OutputCollector<Text, IntWritable> output, Reporter r)
        throws IOException {

        int count = 0;
        IntWritable i;

        while(Value.hasNext()){
            i = Value.next();
            count += i.get();
        }
        output.collect(Key, new IntWritable(count));
    }
}
```

Input
(Key, Value)

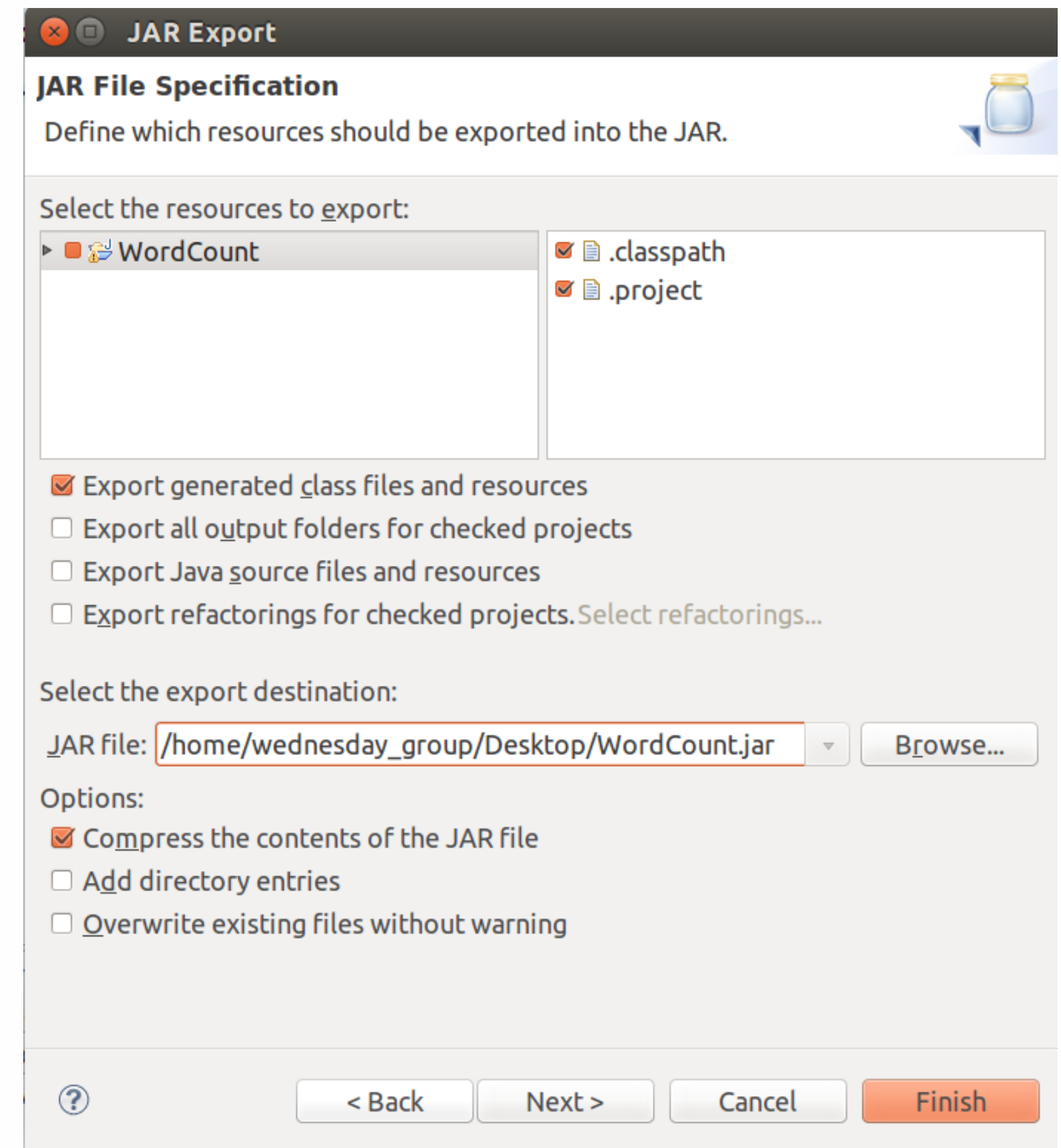
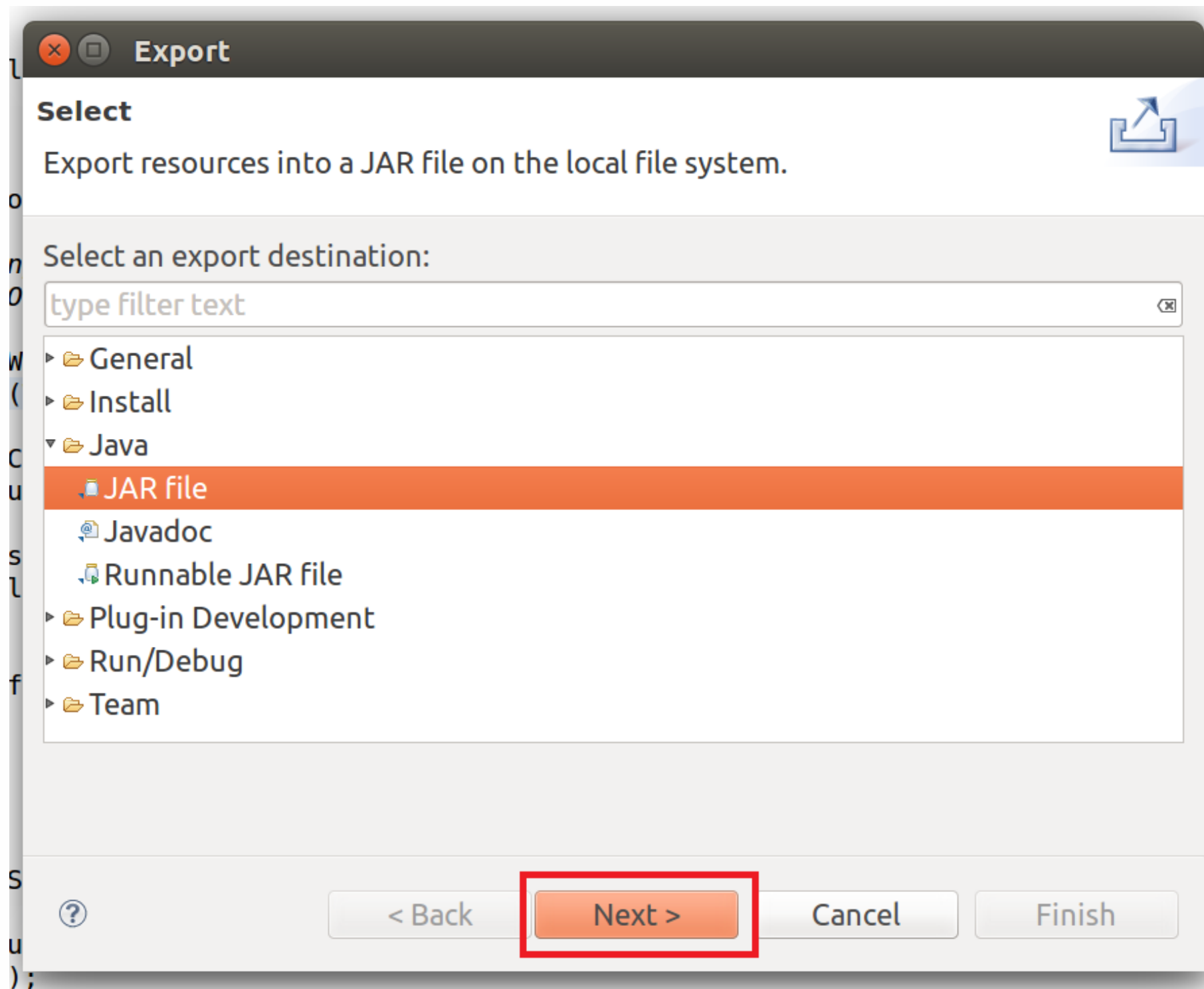
Output
(Key, Value)

Reporter is responsible for
reporting Reducer code to
Driver Code



Step 5: Create a JAR file

- Right click on the project (**WordCount**), and select **Export**.



Step 6: Send the file into HDFS

- In this case, we can use **Hamlet.txt** (in Moodle).
- Now, put it in the Hadoop:

```
start-all.sh
```

```
hdfs dfs -mkdir /WordCount
```

```
hdfs dfs -put <Hamlet.txt path> /WordCount
```

```
wednesday_group@uel-Deskop-VM: ~  
wednesday_group@uel-Deskop-VM:~$ start-all.sh  
WARNING: Attempting to start all Apache Hadoop daemons as wednesday_group in 10 seconds.  
WARNING: This is not a recommended production deployment configuration.  
WARNING: Use CTRL-C to abort.  
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes [uel-Deskop-VM]  
Starting resourcemanager  
Starting nodemanagers  
wednesday_group@uel-Deskop-VM:~$ hdfs dfs -mkdir /WordCount  
wednesday_group@uel-Deskop-VM:~$ hdfs dfs -put /home/wednesday_group/Desktop/Hamlet.txt /WordCount  
wednesday_group@uel-Deskop-VM:~$
```



Step 7: Run the WordCount.jar

Syntax:

hadoop jar <jar file path> <driver name> <HDFS input path> <HDFS output path>

Run:

```
hadoop jar /home/wednesday_group/Desktop/WordCount.jar  
WordCount /WordCount/Hamlet.txt /WordCount/HamletOutput
```

Check the output:

```
hdfs dfs -cat /WordCount/HamletOutput/part-00000
```

MapReduce Output


```
root@ubuntu:/home/bluebit# hadoop jar Documents/WordCount.jar WordCount /WordCount/Hamlet.txt Hamlet
17/10/17 06:38:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/10/17 06:38:22 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/10/17 06:38:22 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/10/17 06:38:22 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/10/17 06:38:22 INFO mapred.FileInputFormat: total input files to process : 1
17/10/17 06:38:23 INFO mapreduce.JobSubmitter: number of splits:2
17/10/17 06:38:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1508247403025_0002
17/10/17 06:38:25 INFO impl.YarnClientImpl: Submitted application application_1508247403025_0002
17/10/17 06:38:25 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1508247403025_0002/
17/10/17 06:38:25 INFO mapreduce.Job: Running job: job_1508247403025_0002
17/10/17 06:38:34 INFO mapreduce.Job: Job job_1508247403025_0002 running in uber mode : false
17/10/17 06:38:34 INFO mapreduce.Job: map 0% reduce 0%
17/10/17 06:38:41 INFO mapreduce.Job: map 100% reduce 0%
17/10/17 06:38:47 INFO mapreduce.Job: map 100% reduce 100%
17/10/17 06:38:48 INFO mapreduce.Job: Job job_1508247403025_0002 completed successfully
17/10/17 06:38:48 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=385071
    FILE: Number of bytes written=1179278
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=197223
    HDFS: Number of bytes written=83992
    HDFS: Number of read operations=9
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=2
```



The Outputs in Web Browser

Browsing HDFS x All Applications x +

localhost:8088/cluster

 **All Applications**

Cluster

- About
- Nodes
- Node Labels
- Applications
 - NEW
 - NEW_SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved
1	0	0	1	0	0 B	8 GB	0 B

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=M), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB
application_1571313334001_0001	wednesday_group	WordCount.jar	MAPREDUCE	default	0	Thu Oct 17 13:01:30 +0100 2019	Thu Oct 17 13:01:35 +0100 2019	Thu Oct 17 13:02:41 +0100 2019	FINISHED	SUCCEEDED	N/A	N/A	N/A

Showing 1 to 1 of 1 entries

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/WordCount/HamletOutput Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	wednesday_group	supergroup	0 B	Oct 17 13:02	1	128 MB	_SUCCESS
-rw-r--r--	wednesday_group	supergroup	82.02 KB	Oct 17 13:02	1	128 MB	part-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2019.

Optimization with Partitioner

- By default, we have just one Reducer slot.
- The main purpose of **partitioner** is partitioning the key/value pairs of mapper output.
- Keys are break down into different groups and send them to different reducers and create separate outputs.
- The **partitioner** phase takes place after the map phase and before the reduce phase in the MapReduce program.



Optimization with Partitioner

- *Hashpartitioner()* function is responsible for executing partitioner.
- Example: we have a file containing lowercase and uppercase words (e.g., hh EE kk bb ZZ ...). Divide your output into two separate parts, one file for lowercase words, one file for uppercase ones.
- Example: for WordCount example, generate different outputs based on **word length**. E.g., words with length 1 in 1st reducer, with length 2 in 2nd reducer, and so on.

WordCount with Partitioner

```
WordCount.java WordMapper.java WordReducer.java MyPartitioner.java x
import javax.xml.soap.Text;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Partitioner;

public class MyPartitioner implements Partitioner<Text, IntWritable>{

    @Override
    public void configure(JobConf arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public int getPartition(Text Key, IntWritable Value, int SetNumReducer) {

        String S = Key.toString();

        if (S.length() == 1)
            return 0;
        else if (S.length() == 2)
            return 1;
        else if (S.length() == 3)
            return 2;
        else if (S.length() == 4)
            return 3;
        else
            return 4;
    }
}
```

```
public class WordCount extends Configured implements Tool{

    @Override
    public int run(String[] arg0) throws Exception {
        // TODO Auto-generated method stub
        if (arg0.length < 2){
            System.out.println("Please enter Input and Output paths accordingly.");
            return -1;
        }

        JobConf conf = new JobConf(WordCount.class);

        FileInputFormat.setInputPaths(conf, new Path(arg0[0]));
        FileOutputFormat.setOutputPath(conf, new Path(arg0[1]));

        conf.setMapperClass(WordMapper.class);
        conf.setReducerClass(WordReducer.class);

        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);

        conf.setPartitionerClass(MyPartitioner.class);
        conf.setNumReduceTasks(5);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
    }
}
```

- (1) Create **MyPartitioner.java** class
- (3) **Export** it as WordCount_Part.jar

- (2) Add these 2 conf. lines in the **Driver** class



WordCount with Partitioner

Run the JAR file:

```
hadoop jar /home/wednesday_group/Desktop/WordCount_Part.jar  
WordCount /WordCount/Hamlet.txt /WordCount/HamletOutput_Partition
```

```
wednesday_group@uel-Desktop-VM:~$ hadoop jar /home/wednesday_group/Desktop/WordCount_Part.jar WordCount /WordCount/Hamlet.txt /WordCount/HamletOutput_Partition
2019-10-17 16:09:43,671 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2019-10-17 16:09:43,950 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2019-10-17 16:09:44,201 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2019-10-17 16:09:44,220 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/wednesday_group/.staging/job_1571313334001_0003
2019-10-17 16:09:44,543 INFO mapred.FileInputFormat: Total input files to process : 1
2019-10-17 16:09:44,693 INFO mapreduce.JobSubmitter: number of splits:2
2019-10-17 16:09:44,765 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2019-10-17 16:09:44,937 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1571313334001_0003
2019-10-17 16:09:44,938 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-10-17 16:09:45,209 INFO conf.Configuration: resource-types.xml not found
2019-10-17 16:09:45,210 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-10-17 16:09:45,327 INFO impl.YarnClientImpl: Submitted application application_1571313334001_0003
2019-10-17 16:09:45,406 INFO mapreduce.Job: The url to track the job: http://uel-Desktop-VM:8088/proxy/application_1571313334001_0003/
2019-10-17 16:09:45,408 INFO mapreduce.Job: Running job: job_1571313334001_0003
2019-10-17 16:09:53,815 INFO mapreduce.Job: Job job_1571313334001_0003 running in uber mode : false
2019-10-17 16:09:53,829 INFO mapreduce.Job:  map 0% reduce 0%
2019-10-17 16:10:02,325 INFO mapreduce.Job:  map 100% reduce 0%
2019-10-17 16:10:22,267 INFO mapreduce.Job:  map 100% reduce 20%
2019-10-17 16:10:23,291 INFO mapreduce.Job:  map 100% reduce 40%
2019-10-17 16:10:25,313 INFO mapreduce.Job:  map 100% reduce 60%
2019-10-17 16:10:26,325 INFO mapreduce.Job:  map 100% reduce 100%
2019-10-17 16:10:27,363 INFO mapreduce.Job: Job job_1571313334001_0003 completed successfully
```

- How to check the output? `cat` or Web Interface

Optimization with Combiner

- If the number of key/value gets more and more, the **Hadoop traffic** goes up for processing. It means that, a file with around 1GB for WordCount, might provide 1 milion keys and associated values, and the "Application Performance" goes down.
- MapReduce applications are limited by the bandwidth available in clusters. **Combiner** minimizes the data shuffled between map and reduce tasks.
- The combiner (**mini-reducer**) function runs on the output of the map phase and is used as a filtering or an aggregating step to reduce the number of intermediate keys that are being passed to the reducer.

Map → Combiner → Partitioner → Shuffle → Sort → **Reduce**

Optimization with Combiner

Output in one mapper:

```
<What,1> <do,1> <you,1> <mean,1> <by,1> <Object,1> <What,1> <do,1> <you,1>  
<know,1> <about,1> <Java,1> <What,1> <is,1> <Java,1> <Virtual,1> <Machine,1>  
<How,1> <Java,1> <enabled,1> <High,1> <Performance,1>
```

Combiner in mapper output:

```
<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1> <Object,1> <know,1> <about,1>  
<Java,1,1,1> <is,1> <Virtual,1> <Machine,1> <How,1> <enabled,1> <High,1>  
<Performance,1>
```

How to use Combiner? Add *conf.setCombinerClass()* as follows:

```
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
```

```
conf.setMapperClass(WordMapper.class);  
conf.setCombinerClass(WordReducer.class);  
conf.setReducerClass(WordReducer.class);
```



Exercise: Find the Max Salary

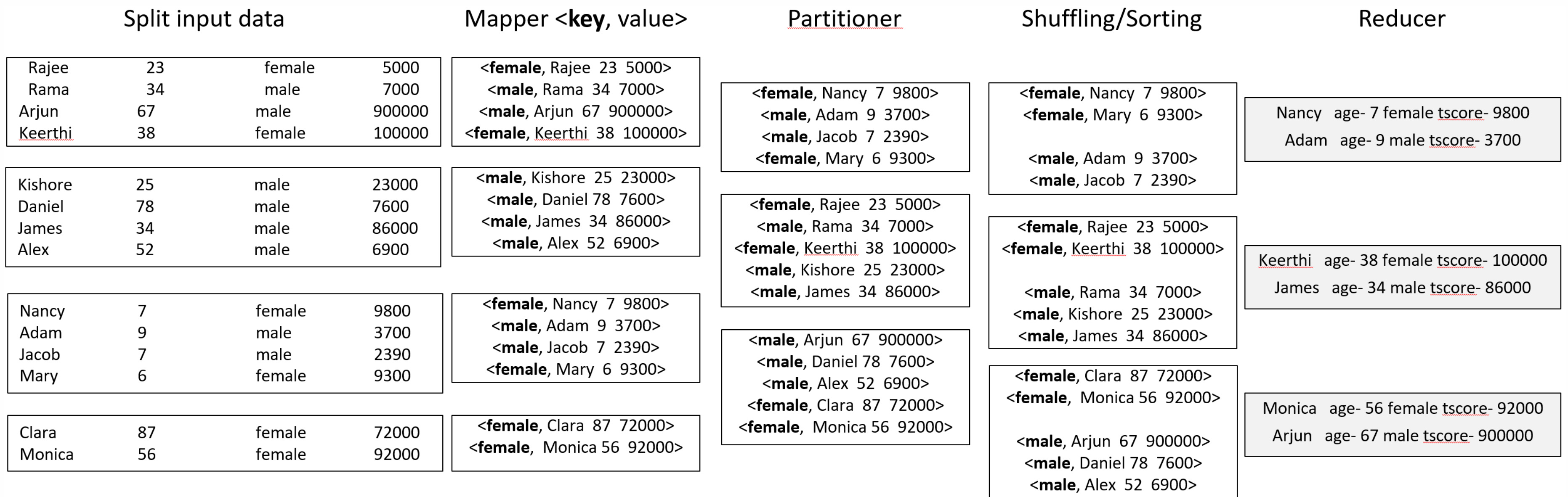
Find the maximum salary in each gender and three age categories: less than 20, 20 to 50, greater than 50.

Rajee	23	female	5000
Rama	34	male	7000
Arjun	67	male	900000
Keerthi	38	female	100000
Kishore	25	male	23000
Daniel	78	male	7600
James	34	male	86000
Alex	52	male	6900
Nancy	7	female	9800
Adam	9	male	3700
Jacob	7	male	2390
Mary	6	female	9300
Clara	87	female	72000
Monica	56	female	92000

- 1) Copy the file into Hadoop
> **hdfs dfs -put <input-file> <hdfs-path>**
- 2) Write **driver, mapper, partitioner,** and **reducer** classes
- 3) Export java program to a JAR file
- 4) Run JAR file:
> **hadoop jar <file.jar> <ClassName> <input file> <output file>**

MapReduce Solution

- `<key, value>` = **<gender, other info>**
- **Partitioner** includes 3 categories: less than 20, 20 to 50, greater than 50.
- **Max Salary** goes in Reducer phase.



Driver Class

```
MaxSalary.java MaxSalaryMapper.java MaxSalaryReducer.java MyPartitioner.java
import org.apache.hadoop.conf.Configuration;

public class MaxSalary extends Configured implements Tool{

    @Override
    public int run(String[] args) throws Exception {
        // TODO Auto-generated method stub
        if (args.length < 2){
            System.out.println("Please enter Input and Output paths accordingly.");
            return -1;
        }

        JobConf conf = new JobConf(MaxSalary.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        conf.setMapperClass(MaxSalaryMapper.class);
        conf.setReducerClass(MaxSalaryReducer.class);

        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(Text.class);

        conf.setPartitionerClass(MyPartitioner.class);
        conf.setNumReduceTasks(3);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);

        JobClient.runJob(conf);

        return 0;
    }

    public static void main(String args[]) throws Exception{
        int exitCode = ToolRunner.run(new MaxSalary(), args);
        System.exit(exitCode);
    }
}
```



Mapper Class

```
MaxSalary.java  MaxSalaryMapper.java  MaxSalaryReducer.java  MyPartitioner.java

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class MaxSalaryMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text>{

    @Override
    public void map(LongWritable Key, Text Value,
        OutputCollector<Text, Text> Output, Reporter r)
        throws IOException {

        // sample record:
        // name<tab>age<tab>gender<tab>salary

        String[] Token = Value.toString().split("\t");
        String Gender = Token[2].toString();
        String NameAgeSalary = Token[0] + "\t" + Token[1] + "\t" + Token[3];

        Output.collect(new Text(Gender), new Text(NameAgeSalary));

    }
}
```



Partitioner Class

```
MaxSalary.java  MaxSalaryMapper.java  MaxSalaryReducer.java  MyPartitioner.java x
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Partitioner;

public class MyPartitioner implements Partitioner<Text, Text>{

    @Override
    public void configure(JobConf arg0) {
        // TODO Auto-generated method stub
    }
    @Override
    public int getPartition(Text Key, Text Value, int setNumRedTasks) {

        String [] NameAgeSalary = Value.toString().split("\t");

        String age = NameAgeSalary[1];

        //this is done to avoid performing mod with 0
        int ageInt = Integer.parseInt(age);

        if(setNumRedTasks == 0)
            return 0;

        //if the age is <20, assign partition 0
        if(ageInt <=20){
            return 0;
        }
        //else if the age is between 20 and 50, assign partition 1
        if(ageInt >20 && ageInt <=50){
            return 1 % setNumRedTasks;
        }
        //otherwise assign partition 2
        else
            return 2 % setNumRedTasks;
    }
}
```



Reducer Class

```
MaxSalary.java  MaxSalaryMapper.java  MaxSalaryReducer.java  MyPartitioner.java

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MaxSalaryReducer extends MapReduceBase
    implements Reducer<Text, Text, Text, Text>{

    @Override
    public void reduce(Text Key, Iterator<Text> Value,
        OutputCollector<Text, Text> output, Reporter r)
        throws IOException {

        int maxSalary = Integer.MIN_VALUE;
        String name = "";
        String age = "";
        String gender = "";
        int salary = 0;

        while(Value.hasNext()){
            String [] valTokens = Value.next().toString().split("\t");
            salary = Integer.parseInt(valTokens[2]);

            if(salary > maxSalary){
                name = valTokens[0];
                age = valTokens[1];
                gender = Key.toString();
                maxSalary = salary;
            }
        }
        output.collect(new Text(name), new Text("age- " + age + "\t" + gender + "tscore- " + maxSalary));
    }
}
```



Output

```
root@ubuntu:/home/bluebit/Desktop# hadoop jar Salary.jar MaxSalary /Salary/Salary.txt /Salary/Output10
17/10/18 14:51:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
Using jni-in-java classes where applicable
17/10/18 14:51:24 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/10/18 14:51:24 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/10/18 14:51:24 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed.
Using the Tool interface to execute your application with ToolRunner to remedy this.
17/10/18 14:51:25 INFO mapred.FileInputFormat: Total input files to process : 1
17/10/18 14:51:25 INFO mapreduce.JobSubmitter: number of splits:2
17/10/18 14:51:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1508354491283_0025
17/10/18 14:51:25 INFO impl.YarnClientImpl: Submitted application application_1508354491283_0025
17/10/18 14:51:25 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1508354491283_0025/
17/10/18 14:51:25 INFO mapreduce.Job: Running job: job_1508354491283_0025
17/10/18 14:51:32 INFO mapreduce.Job: Job job_1508354491283_0025 running in uber mode : false
17/10/18 14:51:32 INFO mapreduce.Job: map 0% reduce 0%
17/10/18 14:51:37 INFO mapreduce.Job: map 100% reduce 0%
17/10/18 14:51:43 INFO mapreduce.Job: map 100% reduce 33%
17/10/18 14:51:44 INFO mapreduce.Job: map 100% reduce 67%
17/10/18 14:51:45 INFO mapreduce.Job: map 100% reduce 100%
17/10/18 14:51:45 INFO mapreduce.Job: Job job_1508354491283_0025 completed successfully
17/10/18 14:51:45 INFO mapreduce.Job: Counters: 49
```

```
Bytes Written=198
root@ubuntu:/home/bluebit/Desktop# hdfs dfs -cat /Salary/Output10/part-00000
17/10/18 14:54:11 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using jni-in-java classes where applicable
Nancy age- 7 femaletscore- 9800
Adam age- 9 maletscore- 3700
root@ubuntu:/home/bluebit/Desktop# hdfs dfs -cat /Salary/Output10/part-00001
17/10/18 14:54:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using jni-in-java classes where applicable
Keerthi age- 38 femaletscore- 100000
James age- 34 maletscore- 86000
root@ubuntu:/home/bluebit/Desktop# hdfs dfs -cat /Salary/Output10/part-00002
17/10/18 14:54:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using jni-in-java classes where applicable
Monica age- 56 femaletscore- 92000
Arjun age- 67 maletscore- 900000
root@ubuntu:/home/bluebit/Desktop#
```



How to execute MaxSalary project?

- Create the JAVA project and name it *MaxSalary*, similar to the *WordCount* Project.
- Then, Download the *Salary.txt* file from Moodle and put it inside the HDFS, then run the “*hadoop jar*” command:

```
tuesday_group@uel-Desktop-VM:~$ hdfs dfs -put /home/tuesday_group/Desktop/Salary.txt /
tuesday_group@uel-Desktop-VM:~$ hadoop jar /home/tuesday_group/Desktop/MaxSalary_Calculate.jar MaxSalary /Salary.txt /Salary_Output
2019-10-17 21:41:07,371 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2019-10-17 21:41:07,611 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
```

- Or, simply ignore the preparation steps in Eclipse, and download the *MaxSalary_Calculate.jar* file from Moodle and execute it by “*hadoop jar*” command.

Summary

- MapReduce Programming
- Optimizing MapReduce Performance by Partitioner and Combiner
- Practiced Driver, Mapper, Partitioner, Combiner and Reducer programming with java
- Exercised several MapReduce Programming in Java Eclipse