Apache Hadoop Developer Hands On Document

TABLE OF CONTENTS

READ ME

This training course uses a VMWare Virtual Machine running the CentOS 6 Linux distribution. This VM is loaded with all the required softwares and tools like jdk 1.7, eclipse juno, and apache hadoop 1.0.3(vanilla hadoop) in Pseudo---Distributed mode.

Pseudo---Distributed means all the 5 components/daemons of Hadoop are running on the same system. For beginners, pseudo mode is quite sufficient; the only key difference (apart from speed) in pseudo mode and cluster mode is that the block replication factor is set to 1, since there is only a single DataNode available.

Points to note

1. The VM is having a user named training whose password is also “training”. Should you need any root privileges, you can switch to root user whose password is also “training”.
2. Sample solutions for all Map-reduce exercises are always available in hd_workspace directory inside the <user home> directory (which is \home\training)
3. We will start with detailed instruction and as this document progresses, instructions will be on higher level only. Feel free to call your trainer at any step. Should you feel consulting your fellow student, please do it as quietly as possible.
4. We use ‘$’ to indicate the command prompt for brevity’s sake, although your command prompt is more verbose.
5. Terminal can be opened by clicking on Applications→Open Terminal

1. HDFS commands

Hadoop is copied/installed on your virtual machine at /usr/local/hadoop($HADOOP_HOME). Start the hadoop daemons by typing following command on terminal window

```
$ start-all.sh
```

Most of your interaction with the system will be through a command---line wrapper called *hadoop.hadoop* resides in $HADOOP_HOME/bin folder and is added in your $PATH environmental variable. If you start a terminal and run this program with no arguments, it prints a help message. To try it out, run the following command on terminal:

```
$ Hadoop
```

The hadoop command is subdivided into several subsystems. For example, there is a subsystem for working with files in HDFS and another for launching and managing MapReduce processing jobs.

1.1 Check the status of cluster

- Check the status by typing following command on terminal

```
$ hadoop dfsadmin –report
```

- To check safemode, type this command

```
$ hadoop dfsadmin –safemode get
```

1.2 Exploring HDFS

The subsystem associated with HDFS in the Hadoop wrapper program is called FsShell. This subsystem can be invoked with the command hadoop fs

- To see help type following

```
$ hadoop fs
```

You see a help message describing all the commands associated with this subsystem.

- To see contents of HDFS, type

```
$ hadoop fs  -ls
```

This might throw you exception like " No File or Directory", since HDFS is empty. There is no home directory of the training user yet created on HDFS.
Please follow the next command

- To see contents of root directory type

```
$ hadoop fs -ls /
```

- To create a directory in HDFS, type following

```
$ hadoop fs -mkdir first
```

- To see contents of your hadoop user's HOME directory

```
$ hadoop fs -ls /user/training
```

Please note that here you are using training user as a hadoop user. In actual production environments there must be a dedicated user for accessing HDFS. Also note that the directory structure in HDFS has nothing to do with the directory structure of the local filesystem; they are completely separate namespaces.

1.3 Loading data into HDFS

Besides browsing the existing filesystem, another important thing you can do with FsShell is to upload data into HDFS.

- Go to directory containing sample data:

```
$ cd /home/training/traning_materials/data
```

- If you perform a 'regular' ls command in this directory, you will see a file named as shakespeare.tar.gz. Unzip this file by typing

```
$ tar zxvf shakespeare.tar.gz
```

This creates a directory named shakespeare/ containing several files on your local filesystem.

- Push or insert this file into HDFS by typing this command

```
$ hadoop fs -put shakespeare shakespeare
```

This copies the local "shakespeare" directory and its contents into HDFS directory named "/user/training/shakespeare". Please note that "/user/training" is the home directory of training user on HDFS.
Please note that you can also use **copyFromLocal** instead of put. Please try the above command using copyFromLocal

- List the contents of your HDFS home directory now

```
$ hadoop fs -ls
```

You should see an entry for the "shakespeare" directory. Please note that if you run above command without using the directory (/user/training), then also you will get the same result because then –ls command thinks that you are in your home directory of the user who is performing the command. In our case it is **"/user/training**"

Anything specified relative will be treated relative to the user's home directory**("/user/training**" here)

If you pass any relative (non-absolute) paths to FsShell commands (or use relative paths in MapReduce programs), they are considered relative to yourhome directory. For example, you can see the contents of the uploadedshakespeare directory by running:

```
$ hadoop fs -ls shakespeare
```

- Try the same (put) with access_log.gz

Hint: Use gunzip. Create a directory by using mkdir command. And then copy file in this directory.
NOTE: We need this data in the subsequent map reduce program. Please perform put this data

1.4 Viewing and manipulating files

Let's try to view data you copied earlier.

- First check the contents in directory

```
$ hadoop fs -ls shakespeare
```

You can see the files like comedies, glossary, histories, poems, and tragedies, very similar to the regular file system contents.

- Remove glossary file, as it's not a work of Shakespeare

```
$ hadoop fs -rm shakespeare/glossary
```

NOTE: we are using relative path. Because by default it is pointing to home directory of training user on HDFS. In our case "training" user is performing this operation, so it is pointing to "/user/training" which is the home directory of training user on HDFS.

- Print first 50 lines of histories file by typing this

```
$ hadoop fs -cat shakespeare/histories | head-50
```

- Copy a file from HDFS to local file system. This is opposite to 'put' command

```
$ hadoop fs -get shakespeare/poems ~/shakepoems.txt
```

You can also use **copyToLocal** command

- Verifying whether you can set the replication factor of a file to 0 (zero) or not.

```
$hadoop fs  -setrep 0 shakespeare/poems
```

- As you know that no two replicas can reside on the same machine, since you are working on pseudo mode verify whether you can set the replication factor to more than 1 or not

```
$hadoop  fs  -setrep 10 shakespeare/poems
```

Please go to the namenode browser and check the replication factor of this file.
You might have observed it is showing you 10 replicas.
Where are the rest 9 replicas, because you cannot have more than 1 replica on the same machine.
Run the following command to check :

```
$hadoop  fsck  /user/training/shakespeare/poems  -files   -blocks  -locations
```

2. HDFS API

In this exercise you will run Java programs to access HDFS.

You need to run a java standalone program - HadoopUtility.

2.1 Get the sample code into eclipse

- Launch Eclipse. Double click on the shortcut to open Eclipse. It will ask to choose a workspace. Choose /home/training/hd_workspace for your work.
- Select “Import” from the “File” menu at the top.
- Select General ---> Existing Projects into Workspace, and click Next.
- Specify /home/training/hd_workspace/UsingHDFSAPI in the Select Root Directory field. It should show the UsingHDFSAPI sample project as selected.
- Click on Finish button. That will get the UsingHDFSAPI project in your workspace.

2.2 Launch HadoopUtility

- Expand src/main/java
- Expand com.felix.hadoop.training package
- Right-click on HadoopUtility and select “Run As”->”Java Application”

You will see some output information about the HDFS files and directories on the console. Here is the explanation of important piece of code

2.3 Getting the FileSystem

```
Configuration conf = new Configuration();

//Telling the configuration object where is the namenode running

conf.set("fs.default.name","hdfs://localhost:54310");

FileSystem fs = FileSystem.get(conf);
```

2.4 Getting home directory

```
Path homeDir = fs.getHomeDirectory();
```

2.5 Creating a directory

```
fs.mkdirs(path)
```

2.6 Creating a file and populating

```
FSDataOutputStream ostream = fs.create(inputFile);

//lets write something to this newly created file

ostream.writeBytes("Hello!! How are you");

ostream.close();//closing the file
```

2.7 Counting files and directories

```
for(FileStatus st:stat) {
                        Path p =st.getPath();
                        if (fs.isDirectory(p)) {
                                dirCount++;
                                count(fs,p);
                        } else {
                                fileCount++;
                        }
                }
```

3. Running WordCount Map-Reduce job

In this exercise you will compile Java files, create a JAR, and run Map-Reduce jobs.

In addition to manipulating files in HDFS, the wrapper program hadoop is used to launch Map-Reduce jobs. The code for a job is contained in a compiled JAR file. Hadoop loads the JAR into HDFS and distributes it to the worker nodes, where the individual tasks of the Map-Reduce job are executed. One simple example of a Map-Reduce job is to count the number of occurrences of each word in a file or set of files. In this exercise, you will compile and submit a Map-Reduce job to count the number of occurrences of every word in the works of shakespeare.

3.1 Get the sample code into eclipse

- Launch Eclipse. Double click on the shortcut to open Eclipse. It will ask to choose a workspace. Choose /home/training/hd_workspace for your work.
- Select “Import” from the “File” menu at the top.
- Select General ---> Existing Projects into Workspace, and click Next.
- Specify /home/training/hd_workspace/BasicWordCount in the Select Root Directory field. It should show the BasicWordCount sample project as selected.
- Click on Finish button. That will get the BasicWordCount project in your workspace.

3.2 Create jar file

- Right click on BasicWordCount project and click on “Export”
- Choose Java→ JAR file and click on “Next”.
- Make sure to choose only Java files in “Select the resources to export” dialog box. Unselect the .classpath and .project from right side of the window.
- Click on browse and select the /home/training/hd_workspace/BasicWordCount and provide a file name say wordcount.jar. Click on “Save”
- Then click on “Finish” button.

3.3 Submit the job to JobTracker

Submit a MapReduce job to Hadoop using your JAR file to count the occurrences of each word inshakespeare:

```
$ hadoop jar wordcount.jarcom.felix.hadoop.training.WordCountDrivershakespeare
wordcounts
```

This hadoop jar command names the JAR file to use (wordcount.jar), the class whose main method should beinvoked (WordCountDriver), and the HDFS input and output directories to use for the MapReduce job.

3.4 Miscellanous

- Try to run the above command again; it will not work because the output directory already exists.
- Check the result of your job

```
$ hadoop fs -ls wordcounts
```

You should see a file part-r-00000, along with a _SUCCESS file and a _logs directory. File part-r-00000 is the output file of a reducer (default 1)

- View the contents of output file by typing:

```
$ hadoop fs -cat wordcounts/part-r-00000 | less
```

- Delete the output files , should you look to run it again(or specify a new output file for next invocation)

```
$ hadoop fs -rmr wordcounts
```

Please ignore the warning.

4. Word Co-Occurence

In this exercise you will write your own Mapper and Reducer as well as Driver class.You will write an application that counts the number of times words appear next to each other. Please note that we are only interested in pairs of words which appear directly next to each other.

For any text input, the job output should report the occurrence of pair of words coming together. For example, for input:

```
This is a hadoop hands on exercise
This is a pdf
 This is just assistance
```

The output would be:

```
This is3
is a2
ahadoop1
hadoop hands1
hands on1
on excercise1
a pdf          1
is just           1
just assistance   1
```

The algorithm for this program is a simple one---pass MapReduce program:

4.1 MAPPER

The Mapper receives a line of text for each input value. (Please ignore the input key.) For each word in the line, emit that word and the next word as a key (Text, separated by space),and 1(IntWritable) as a value. For example, for input value:

```
This is a hadoop hands on exercise
```

Your Mapper should emit:

```
This is 1
Is a 1
a hadoop1
hadoop hands 1
hands on 1
on exercise   1
```

4.2 REDUCER

The Reducer receives the keys(adjacent words separated by a space) in sorted order, and all the values for one key appear together. (WHY?)So, for the Mapper output above, the Reducer receives this:

```
This is(1,1,1)

a hadoop(1)

a pdf(1)

hadoop hands (1)

hands on        (1)

is a            (1,1)

is just(1)

just assistance  (1)

onexercise      (1)
```

For above input, the reducer output should be:

```
This is          3
a hadoop         1
a pdf            1
hadoop hands     1
hands on        1
is a             2
is just         1
just assistance  1
onexercise      1
```

4.3 DRIVER(Using ToolRunner)

Write a Driver class and configure your Mapper and Reducer. Specify the input and output directories. Please make sure you are using a unique non-existent output directory. Should you have any doubt fire the “hadoop ls” command to check the HDFS data.

You can see the Driver class for WordCount (com.hadoop.example.wordcount.WordCountDriver) for any references.

4.4 EXECUTION

Compile, jar, and test your program. You can use the entire Shakespeare dataset for your input, or you can try it with just one of the files in the dataset, or with your own test data.

5. Average Word length

You will write an application that calculates the average word length for each character.

For any text input, the job output should report the average length of words that begin with ‘a’, ‘b’, and so forth. For simplicity, let’s keep this program case-sensitive ie ‘a’ and ‘A’ are treated as different characters.To elaborate more, let’s see an example input:

```
This is a hadoop hands on excercise
```

The output would be:

```
T       4

i       2

a       1

h       5.5

o       2

e       9
```

The algorithm for this program is a simple one---pass MapReduce program:

5.1 MAPPER

The Mapper receives a line of text for each input value. (Please ignore the input key.) For each word in the line, emit the first letter of the word as a key, and the length of the word as a value. For example, for input value:

```
Now is definitely the time
```

Your Mapper should emit:

```
N     3

i     2

d     10

t     3

t     4
```

5.2 REDUCER

The Reducer receives the keys in sorted order, and all the values for one key appear together. (WHY?) So, for the Mapper output above, the Reducer receives this:

```
N    (3)

d    (10)

i    (2)

t    (3, 4)
```

For above input, the reducer output should be:

```
N    3

d    10

i    2

t    3.5
```

5.3 DRIVER(Using ToolRunner)

Write a Driver class and configure your Mapper and Reducer. Specify the input and output directories. Please make sure you are using a unique non-existent output directory. Should you have any doubt fire the “hadoop ls” command to check the HDFS data.

You can see the Driver class for WordCount (com.hadoop.example.wordcount.WordCountDriver) for any references.

5.4 EXECUTION

Compile, jar, and test your program. You can use the entire Shakespeare dataset for your input, or you can try it with just one of the files in the dataset, or with your own test data.

6. Inverted Index

You need to write a job which outputs a word and list of files where this word occurred. For ex, if “the” occurred in a.txt and b.txt, the output of job should be

```
the a.txt, b.txt
```

6.1 Mapper

Mapper will be similar to word count mapper. It should take each line and get all the words in an array. Now, it has to emit each word as key and file name as value. You can get the filename by using below code:

```
FileSplit fileSplit = (FileSplit) context.getInputSplit();

Path path = fileSplit.getPath();

String fileName = path.getName();
```

So, if your input is like this:

```
Hi how are you
```

Your mapper should emit:

```
Hi          a.txt
how         a.txt
are         a.txt
you         a.txt
```

6.2 Reducer

Reducer will get an input of keys and list of files. Please note that for commonly occurring words like the, I, is, am etc there will be multiple outputs from mapper, hence the filenames will be repeating. For example, this can be an input:

```
Hi          (a.txt, b.txt, a.txt, a.txt, b.txt)
how         (a.txt, b.txt)
are         (a.txt, b.txt, b.txt, c.txt, d.txt, d.txt, a.txt)
you         (c.txt, c.txt)
```

Reducer output should be something like this:

```
Hi       a.txt, b.txt
how      a.txt, b.txt
```

```
are     a.txt, b.txt, c.txt, d.txt
you     c.txt
```

6.3 DRIVER(Using ToolRunner)

Write a Driver class and configure your Mapper and Reducer. Specify the input and output directories. Please make sure you are using a unique non-existent output directory

7. Searching

In this section you will learn how to search a word or a phrase.

Searching a word can be accomplished by providing the search term to the mapper. The best way to make any lookup data available to map() method of mapper/reducer is using setup() method, since it will be called only once by the framework.

One of a very good way to make anything available in setup() method is passing parameter while running job using ToolRunner. Please see "Passing Parameters" section below to understand how it's done.

8. Sorting

In this section, you are going to see how to write your own sorting logic.

As you know, framework always does sorting on keys (Why?). So, in order to achieve sorting on any field of data, you need to make that field as the key and keep the entire line as the value to the mapper. Follow the steps to understand how to create different keys from input lines.

Step1

Import the data.

Use **/home/training/training_materials/data/ReduceAndMapSideJoin/Employee_Table.csv**. Run **hadoop fs -put** command and copy this file into HDFS.

Step 2

Write a Mapper class.

Ignore the key as it's an offset of each line. Take the value which is representing the complete line.

Extract the firstname of employee by using Java String's split() method i.e

```
String fisrtName = str.split(",")[0];
```

Emit **firstName** as the Key and the **complete line** as the value from mapper.

Step 3

Write the driver code. Use IdentityReducer as the reducer. Execute the code.

All keys must implements **WritableComparable** which means sorting logic is defined within the type. If you want to define an alternate sorting logic, you have to write a class which extends **WritableComparator** and provide this new Comparator to the job. This will be demonstrated in **SecondarySorting**.

9. Combiner

You need to set a **combiner** for a job. (job.setCombiner(<yourclassname.class>))

Existing reducer can be used as a combiner if operation is **associative and commutative**. The **WordCount** job is one such kind. Inside Eclipse, create a new project by name **WordCountWithCombiner** and copy the code from **BasicWordCount** project. Set the appropriate build path.

The only change you need to make is to make combiner available to the job. Please add the following to the driver code:

```
job.setCombiner(WordCountReducer.class);
```

Now, create the jar and run your job.

NOTE: In order to see whether combiner is working or not, run your job on accesslog data. Run your job with and without combiner to verify whether you are gaining improvement or not

10. Distributed Cache

You will be using DistributedCache facility when your map reduce job require extra information. For example: Assuming you have Employee data which has "location code" as one of the field

and your job is to replace this " location code" with the actual location name. The mapping of the location code to location name is present in another file, let's say "LocationData".

Note that your map function will be called for every employee record and for every employee record you need to look up the "LocationTable". So to make this data available to every employee record you will use distributed cache facility.

Place your file on HDFS:

```
hadoop fs -put ~/training_material/data/ReduceAndMapSideJoin/Employee_Table.csv emp_table

hadoop fs -put ~/training_material/data/ReduceAndMapSideJoin/ us_states.csv location_table
```

Create your jar file and run the follow command (provide the jar name, driver name and output directory name)

```
hadoop jar<jar name><Driver Name> emp_table <output directory name>
```

11. Passing Parameters

You need to pass parameters to your job.

When driver is written using ToolRunner methodology, this is fairly simple. Just supply "–D name=value" in the job command. Please note the space between D and name.

Step 1

Decide the parameter name. Let's call it search.word. Access the parameter in Mapper by using context object like this

```
public void setup(Context context)
 {

searchWord = context.getConfiguration().get("search.word");

 }
```

Step 2

Provide the search.word from job execution command like this(we are searching “the” in all input lines)

```
hadoop jar myJar MyDriver –D search.word=“the” in out
```

Step 3

Use the Shakespeare data used in sorting above to find “the” records.

12. Counters

In this section you will write your own counters.

We will go through Shakespeare data and try to find out number of words starting with vowels and number of words starting using consonants.

Step 1

In the mapper, all you need is to find out the first character of any word(Use string API):

```
char ch = word.charAt(0);
```

Once you have the match use context object to put this counter value.

```
output.getCounter("VowelConsonantCounter", WORD_STARTING_WITH_VOWEL").increment(1);
```

Step 2

Print the value of counter in driver code after job gets completed by using following

```
job.getCounters().findCounter"VowelConsonantCounter","WORD_STARTING_WITH_VOWEL").getValue();
```

On a separate note, these counters are printed on the client console as part of summary as well as you can see these values on the job ui page at “http://localhost:50030”

13. Custom Key-Values

You need to write a custom Type. It’s a java class required when the Hadoop’s boxed types are not enough. While writing any custom type it should be a combination of hadoop’s type only so

that you can utilize the optimization benefits of these types in your custom type. Custom type must implement Writable (for Value type) and WritableComparable(for Key Type).

For hands-on on custom key, you can refer the project WordCountUsingCustomKeys in the hd_workspace folder.

13.1 Writable

Step 1

Write a class EmployeeRecord which implements Writable. Create these fields as Text type:

- firstName
- lastName
- details
- locationCode
- locationName

Step2

Provide default constructor (which should call new Text() for every field). Overload this constructor and write two more construtors – one should take String arguments, other should take Text as an argument:

- publicEmployeeRecord() {….}
- publicEmployeeRecord(String firstName, String lastName, String details, String locationCode, String locationName) {…..}
- public EmployeeRecord(Text firstName, Text lastName, Text details, Text locationCode, Text locationName) {….}

Step 3

Provide getter and setters for each field

Step 4

Override readFields() like this:

```
@Override
public void readFields(DataInput in) throws IOException {
firstName.readFields(in);
   ……..
}
```

Step 5

Override write() like this

```
  @Override
public void write(DataOutput out) throws IOException {
                firstName.write(out);
                ……
        }
```

Step 6

Override equals() and hashCode() if this type is going to be used in any collection objects. Override as well as toString() method should you require to print state of object anywhere in job.

13.2 WritableComparable

Step1

Write a class TextPair which implements WritableComparable. Create these fields as Text type:

- firstWord
- secondWord

Step 2 – Step 6

These are same as above for Writable. Provide equals() and hashCode() as must.

Step 7

Override compareTo() as your keys are going to be used as Keys and keys are sorted by default:

```
@Override
        public int compareTo(LocationCustomKey o) {
   //compare both firstWord and secondWord
……
        }
```

Use TextPair as a key in Word Co-Occurrence. You will see that result ordering has changed

14. Custom Partitioner

Here you need to write your own partitioner. Partitioner is required to didvide keys among reducers based on load handling and/or use case requirements.
Implement a parttioner which divides all the words between 2 set of files – one file should contain all the words starting with vowels(a,e,I,o,u) and the other will contain the remaining words starting with consonants.

So, output should be only 2 files and please remember we are not taking any count of words here.

14.1 Mapper

Mapper will get each line. So, it will find out all words and then emit the character of each word as key and the word itself as value.

14.2 Partitioner

You need to extend Partitioner class given by MapReduce framework. Override the method getPartition(). Return 0 for characters a,e,I,o,u and return 1 for rest.

14.3 Reducer

Reducer is much simpler, just output values from the list as key and null(NullWritable) as value.

14.4 Driver

In driver you must set the partitioner by calling

```
job.setPartitionerClass(CustomPartitioner.class);
```

Don’t forget to set number of reducers to 2 by sepecifying below while running job

```
-D mapred.reduce.tasks=2
```

Place above parameter after specifying Driver class in “hadoop jar” command. Please notice the space between D and mapred.reduce.tasks.
You may use Shakespeare data

15. Custom Input Format

Custom Input format is used when you want a better control over the key value pairs to be processed by the map function.

Upload the sample wiki file on HDFS

```
hadoop fs -put ~/training_material/data/wikisample/wiki1 wiki1
```

Create the jar file and run the following command (provide the jar name, driver name and output file

```
hadoop jar <jar name> <Driver name> -D PROJECT_NAME=en wiki1 customif_op
```

Check the output file whether proper filtering is done or not. Verify by giving different values to PROJECT_NAME property in the command line

16. Secondary Sorting

This is a complex program. You need to write your own Custom key, Partitioner, Grouping Comparator and Sort Comparator

16.1 Pseudo Steps

- Write a custom key called Employee by implementing WritableComparable. Create two fields, firstName and lastName. Provide equals() and hashCode() as well. This class is exactly same as TextPair mentioned above in “Custom Key Values” section
- Write a Comparator by extending WritableComparator. Override compare() method and compare both firstName and lastName. compare() method will look like this

```
public int compare(WritableComparable a, WritableComparable b) {
                        if (a instanceof Person && b instanceof Person) {
                                Person p1 = (Person) a;
                                Person p2 = (Person) b;
                                int comparison =
        p1.getFirstName().compareTo(p2.getFirstName());
                                if(comparison == 0) {
                        return
```

```
        p1.getLastName().compareTo(p2.getLastName());
                        }
                        return comparison;
                }
                return super.compare(a,b);
        }
```

- Write a Partitioner by extending Partitioner. Implements getPartition() and just use hashCode of firstName.
- Write a Grouping Comparator by extending WritableComparator. Override compare() method and compare only firstName.
- Write a Driver using ToolRunner by extending Comfigured and implementing Tools.
- Inside Driver class set the comparator by calling setSortComparatorClass() and set grouping comparator by calling setGroupingComparatorClass().
- Inside Driver Set partitioner by calling setPartitionerClass().
- Create a jar file by using export option in Eclipse. Don't include .classpath and .project files while creating jar file.
- Run the code by typing "hadoop jar <ssort.jar><SecondarySortDriver><input><output>

17. Map side join

This is covered in Distributed Cache section. Please refer that section.

18. Reduce side join

This is a complex program. You need to write your own Custome Key, Custom Value, Partitioner, Grouping Comparator and Sort Comparator apart from mapper and reducer.

18.1 Pseudo Steps

- Write a custom value type called EmployeeRecord by implementing Writable. Create two fields – firstName and lastName. Provide equals() and hashCode() although it's not mandatory.
- Write a custom key type called LocationCustomKey by implementing WritableComparable. Create two fields – locationCode and typeOfRecord. Provide equals() and hashCode() as well. Compare on locationCode first and then on typeOfRecord (if locationCode is equal) in compareTo() method
- Write a Comparator by extending WritableComparator. Override compare() method and compare both firstName and lastName.

- Write a Partitioner by extending Partitioner. Implements getPartition() and just use hashCode of locationCode from LocationCustomKey.
- Write a Grouping Comparator by extending WritableComparator. Override compare() method and compare only locationCode from LocationCustomKey.
- Write a mapper class. For each line split it on commas(,) and then based on length of record you can either create a EmployeeRecord for employee or for location. In either case, create a LocationCustomKey record. For employees, set type of record as "z" and for location set it to "a". Objective is to make sure the EmployeeRecord record for location is the first record in reducer followed by all the EmployeeRecord record for employees. Mapper will emit LocationCustomKey as key and EmployeeRecord as value.
- Write a reducer class. Because of secondary sorting on locationCode and typeOfRecord fields in LocationCustomKey, reducer will get a list of EmployeeRecords in which first object is for location and rest all are for employees. So, just copy the location to a temporary variable. Loop through rest of the values and emit employee details(Override toString() in EmployeeRecord) as key and location as value.
- Write a Driver using ToolRunner by extending Comfigured and implementing Tools.
- Inside Driver class set the comparator by calling setSortComparatorClass() and set grouping comparator by calling setGroupingComparatorClass().
- Inside Driver Set partitioner by calling setPartitionerClass().
- Create a jar file by using export option in Eclipse. Don't include .classpath and .project files while creating jar file.
- Run the code by typing "hadoop jar <reducejoin.jar><ReduceSideDriver><input><output>

19. Writing Map-Reduce Using Stub project

Use the StubProject in the workspace to write your own Mapper-Reducer-Driver code. Please ask your trainer for the specific use case.

Copy the project with a different name in Eclipse. Change the name of Mapper-Driver-Reducer. Set the classpath to include Hadoop dependencies.Get, Set , GO…

20. Hive Hands On

NOTE: COPY the hive and pig scripts under /home/training folder

20.1 Opening a hive shell

Before running hive start your hadoop cluster and check whether hadoop is in safemode or not.

Run the below command to start your hadoop cluster

```
$ start-all.sh
```

Run the below command to check whether your hadoop cluster is in safemode or not

```
$ hadoop    dfsadmin    -safemode    get
```

Wait till hadoop's safemode status is OFF.

Now run the following command to start the hive shell

```
$ hive
```

If there is no error, then you will enter into hive shell

```
$ hive>
```

20.2 Common Operations

20.2.1 Listing databases

```
$hive> show databases;
```

By default you will see "default" databases

20.2.2 Creating and using database

```
$hive> create database myDB;
```

You can run the "show databases" command to verify whether "myDB" has been created or not.

You can also verify that a directory by name "mydb.db" has been created under "/user/hive/warehouse" on your hdfs

NOTE: The names of the databases and tables are case insensitive

Run the following command to use this data base

```
$hive> use mydb;
```

Now all the tables will be created under “mydb”

Getting your database locations

```
$hive> DESCRIBE DATABASE EXTENDED mydb;
```

20.2.3 Changing the location of the database

```
$hive> CREATE DATABASE IF NOT EXISTS ‘sample’ LOCATION ‘/user/mydb’;

#verify whether the databases is created or not
$hive>SHOW DATABASES;

#verify the location of the database. Directory by name “mydb” will be created
$hive> DESCRIBE DATABASE  sample;
```

20.2.4 Adding descriptive comments while creating database

You can add a comment while creating a database

```
$hive>CREATE DATABASE IF NOT EXISTS  SAMPLE_1  COMMENT "My sample hive database";

#verify the comment by describing the databases;
$hive>DESCRIBE DATABASE SAMPLE_1;
```

20.2.4.1 Adding DBProperties

Providing key value pair

```
$hive>CREATE DATABASE IF NOT EXISTS  SAMPLE_2 WITH DBPROPERTIES
('creator'='training','date'='29/0/2013');

#verify the database whether you can see the properties or not
$hive>DESCRIBE DATABSE EXTENDED SAMPLE_2;
```

20.2.5 Using the data base

```
$hive>set  hive.cli.print.current.db=true;
```

```
$hive(default)> ## you are currently in default databases;
```

20.2.6 Dropping the database

```
$hive>DROP DATABASE test;
```

Alternatively you can check if the database exists, then only drop it

```
$hive> DROP DATABASE IF EXISTS test;
```

Also you can't drop the database if already tables are existing in the database. Either you have to delete all the tables first and then drop the database, else you can run the following command

```
$hive> DROP DATABASE IF EXISTS test CASCADE;
```

The above command will delete all the tables along with the databases;

20.2.7 Altering database

You can only set the DB properties. No other properties can be changed

```
$hive>CREATE DATABASE IF NOT EXISTS sample;
$hive>ALTER DATABASE  sample SET DBPROPERTIES('CREATOR'='ME');
```

Note youcannot unset a db property, but you can change the property

```
$hive>ALTER DATABASE  sample SET DBPROPERTIES('CREATOR'='you');
```

20.2.8 Creating table

```
$hive> create table docs(words string);
```

You can verify that directory by name "docs" has been created under '/user/hive/warehouse/mydb.db'

```
$hadoop  fs  -ls  /user/hive/warehouse
```

20.2.9 Listing the tables in a database

```
$hive> show tables;
```

You can also view the tables in other databases as well using “IN” keyword

```
$hive>USE default;
```

Using the above command it will point to default database

Run the below command to view the tables in “mydb” database

```
$hive> SHOW TABLES IN mydb;
```

20.2.10 Loading the data into table (Data is in local file system)

Assuming the data is in the local file system, you can run below command to load the data into your table

```
$hive > LOAD DATA LOCAL INPATH  ‘/home/training/training_material/data/shakespeare’
INTO TABLE docs;
```

BEWARE of the single quotes when you are cut copying paste the command on your console

Also verify under the docs folder on your HDFS all the files from the Shakespeare folder is copied

20.2.11 Loading the data into table (Data is on HDFS)

Assuming the data is on HDFS (/user/training/shakespeare), you can run the below command to load the data into your table

```
$hive> LOAD DATA INPATH ‘shakespeare’ INTO TABLE docs;
```

NOTE: The “shakespeare” folder will be moved from “/user/training/shakespeare” to “/user/hive/warehouse/docs” folder

20.2.12 Viewing the contents table on Console

```
$hive> select * from docs;
```

This query will not run map reduce job. Why?

20.2.13 Viewing the schema

```
$hive>DESCRIBE docs;
```

20.2.14 Viewing the size of the table and other information

```
$hive> show table extended like docs;
```

You can run the following commands to get the detailed information

```
$hive>DESCRIBE EXTENDED docs;
```

```
$hive> DESCRIBE FORMATTED docs;
```

20.3 Executing Word Count Problem

Remember you have almost written 100 lines of java code to count the frequency of each word.

```
$hive> CREATE TABLE word_count AS
SELECT word, count(*) AS count FROM
(SELECT explode(split(words, '\\W+')) AS word FROM docs) w
GROUP BY word;
```

20.4 Executing hive script

```
$hive -f   /home/training/hivescripts/wordcount.sql
```

20.5 Partitioning a hive table

```
$hive –f /home/training/hivescripts/UsingPartition.sql
```

It will create directory with a partition

20.6 Creating an External Table

```
DROP TABLE IF EXISTS location_table;

CREATE EXTERNAL TABLE location_table(

location_code STRING,
location_name STRING)
ROW FORMAT
DELIMITED FIELDS TERMINATED by ','
LINES TERMINATED by '\n'
LOCATION '/home/training/training_material/data/ReduceAndMapSideJoin/us_states.csv';
```

External tables are not managed by hive and when you drop the tables the data will not be deleted.

20.7 Difference between managed and external table

When you drop the managed table (internal table) the data associated with it will also be deleted, but when you drop the external table, the data is not deleted.

20.8 Executing data mining on Wiki data set

Put the wiki hourly data set on HDFS by name “wiki”

Extract the wiki files present under “/home/training/training_material/data/wikidata”

You can right click on the file and click on option “extract here” option. Do it for both the files

Now create a directory my name “wiki” on hdfs

```
$ hadoop   fs   -mkdir wiki
```

Put both the files on HDFS

```
$hadoop   fs  -put  ~/training_material/data/wikidata/pagecounts-20130101-000000   wiki

$hadoop fs  -put ~/training_material/data/wikidata/pagecounts-20130101-020001   wiki
```

Execute the hive script as follows:

```
$ hive  -f  /home/training/hivescripts/wiki.sql
```

21. Pig Hands On

21.1 Opening Grunt Shell

Execute the "pig" command from shell

```
$pig
```

The above command if successful, it will open up the grunt shell

```
$grunt>
```

Also note that in the console, pig is pointing to correct File system and job tracker

21.2 Accessing HDFS from grunt shell

You can perform all the file system operation from the grunt shell aswell

```
$grunt> ls
$grunt> fs  -put <input path><output path>
```

21.3 Executing Word Count Problem using Pig

Let's put the Shakespeare directory on HDFS from the grunt shell

```
$grunt>fs -put /home/training/training_material/data/shakespeare shakespeare
```

Now load the shakespeare data and let's also dump if everything looks fine

```
$grunt>A = load 'shakespeare';
$grunt> dump A;
```

Now you need to break the lines into words

```
$grunt>B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word
```

Again dump the output on console to check whether everything is fine

```
$grunt> dump B
```

You might have observed in the output that spaces are also coming, so let's filter those spaces

```
$grunt>C = filter B by word matches '\\w+';
$grunt> dump C;
```

Once it is done now let's group the words so that we can count the words

```
$grunt>D = group C by word;
$grunt> dump D;
```

Once the grouping is done, let's count the words

```
$grunt>E = foreach D generate group,COUNT(C);
$grunt> dump E;
```

21.4 Executing Pig script

Instead of running each the queries through grunt shell, you can write the queries into a file and save that file with ".pig" extension and can run that script.

```
$pig /home/training/PigScript/wordcount.pig
```

21.5 Data mining on wiki data set using pig

We are trying to find top-10 english project sites

Make sure the data is present on HDFS. Assuming "wiki" is a directory present on HDFS where your Wikipedia hourly data files are present

So first step is to load the data

```
$grunt>records = LOAD 'wiki'using PigStorage(' ') as
(projectName:chararray,pageName:chararray,pageCount:int,pageSize:int);
```

Since we are interested to find top-10 english sites, so let's filter the records

```
$grunt>filtered_records = FILTER records by projectName == 'en';
```

Now after filtering, we will group the records and sum their page count

```
$grunt>grouped_records = GROUP filtered_records by pageName;
```

```
$grunt>results = FOREACH grouped_records generate group, SUM(filtered_records.pageCount);
```

Now we have calculate the sum of the page counts, now let's sort the results in descending order

```
sorted_result = ORDER results by $1 desc;
```

Finally storing the output into a file

```
STORE sorted_result INTO  'wikiOP';
```

"wiki.pig" script file is provided to you, which you can run as follows:

```
$grunt> pig /home/training/PigScript/wiki.pig
```

22. Sqoop hands on

CAUTION: DON'T CUT COPY PASTE THE COMMANDS FROM PDF TO YOUR TERMINAL. SINGLE QUOTES, SPACES AND DOUBLE QUOTES,hypen sign WILL NOT RECOGNIZED WHICH WILL LEAD TO ERRORS

Sqoop is not having an interface like hive or pig. It is just a set of tools. Open a terminal and type following:

```
$ sqoop
```

It will say warning about HBase and then "*Try 'sqoop help' for usage*"
So, let's try tools one by one.

22.1 sqoop help or sqoop-help

```
$ sqoop help
```

It should print the general help message and all the tools available. Now, try to get help on a specific tool like import or eval

```
$ sqoop help import
```

```
$ sqoop help eval
```

22.2 sqoop import

Before importing let's check if data is present in mysql database (installed in VM). Open a new terminal and type

```
$ mysql  hadoopguide  -p
```

It will ask for a password. Password is "training". Now it will open mysql prompt.

Check the structure of table first

```
mysql> describe widgets;
```

Now, check the data by issuing

```
mysql> select * from widgets;
```

It should show 10 rows of data. (here, remember "id" is primary key)

```
| id | widget_name | price | design_date | version | design_comment       |
+----+-------------+-------+-------------+---------+----------------------+
|  1 | sprocket    |  0.25 | 2010-02-10  |       1 | Connects two gizmos  |
|  2 | gizmo       |  4.00 | 2009-11-30  |       4 | NULL                 |
|  3 | gadget      | 99.99 | 1983-08-13  |      13 | Our flagship product |
```

```
+----+-------------+-------+-------------+---------+--------------------+
10 rows in set (0.00 sec)
```

Now?

Import it by using import command

```
$ sqoop import --connect jdbc:mysql://localhost/hadoopguide\ --table widgets -m 1
```

If the above command throws the error that the training user has not enough privileges then try running the below command

```
$sqoop import –username=training  -P  jdbc:mysql://localhost/hadoopguide \  --table
widgets  -m 1
```

When you specify number of mapper as 1, then it would be sequential import. You can increase the parallelism by setting the value of m to some greater value, in which case you need to have either the primary key defined in the table or you can provide the "—split-by" option on a column name
For example:

```
$sqoop import  --username=training  -P jdbc:mysql://localhost/hadoopguide \
 --table widgets   --split-by gadget -m 1
```

Beware ! Hadoop should be up. If not, please start by using "start-all.sh"

Examine the output of import. It should create a directory "widgets" in HDFS HOME directory i.e. "/user/training". Since, we specified number of mappers as 1, it will create a single file "/user/training/widgets/part-m-00000". Check the contents by using "cat". By default, rows are stored as "comma" delimited.

Now delete the widgets directory from HDFS.

```
$ hadoopfs –rmr widgets
```

Now, let's try to selectively import

```
$ sqoop import --connect jdbc:mysql://localhost/hadoopguide \
 --table widgets --columns id,widget_name -m 1
```

Check output once again. You will see only data for 2 columns id & widget imported this time

Delete the widgets directory once again.

Now, let's run a free-form query while importing and also specified a target directory

```
$ sqoop import --connect jdbc:mysql://localhost/hadoopguide --query
'SELECT * FROM widgets WHERE widgets.id=1 AND $CONDITIONS'
--split-by widgets.id --target-dir /user/training/joinresults
```

Check output, there is only one file with one record

22.3 Importing to the Hive table

NOTE : Since both HBase and hive is installed on VM and they are using different versions of thrift jars. So sqoop will not be able to upload the data into hive table because of the different versions of thrift jars.

As a work around what you can do, change the HBASE_HOME to some other non-existent path in your .bashrc file and then do the import.

Please revert back HBASE_HOME once you have performed the import to hive table.

Also please close the hive shell if you have already opened it

You can create a hive table with the same schema and run the following command to do the import

```
$sqoopimport  --username=training -P  --connect jdbc:mysql://localhost/testDB --table
<table_in_your_sql> --hive-import --hive-table <hive_table_name> -m 1
```

Now open the hive shell and check whether the data in the table has been populated or not.

Alternatively, if you want the sqoop to create the hive table , then you can run the following command.

Before running the following command, make sure the hive shell is not opened

```
sqoop import  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --
```

```
table <table_in_your_sql>--hive-import --create-hive-table --hive-table test1 -m 1
```

Open your hive shell and verify whether the table has been created or not and verify the data.

22.4 Importing to HBase Table

Run the following command to start the "HBase" process

```
$start-hbase.sh
```

Run "jps" command to check whether a process by name "HMaster" is running. If the process is running then that means "HBase" is up and running

Now run the following command to enter into hbase shell

```
$hbase shell
```

Once you are in hbase shell, run the following command from the hbase shell to create a table

```
hbase(main):001:0> create  'test','cf'
```

Run the following command to import the data into hbase table

NOTE: Please note the option "—hbase-row-key". Here you need to specify which column you would like make as row key for HBase

```
$sqoop import  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --
table testTable  --hbase-table test --hbase-row-key vehicle_number --column-family cf -m 1
```

Open the hbase shell and run the command

```
hbase(main):001:0> scan 'test'
```

You should see the data should have been populated in hbase table.

Alternatively, if you want sqoop to create hbase table, then you can run the following command

In the below command we are creating "test1" table in hbase which is currently not present

```
$sqoop import  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --
table testTable --hbase-create-table  --hbase-table test1 --hbase-row-key vehicle_number --column-
```

```
family cf -m 1
```

Within the hbase shell, verify whether the table has been created or not

```
hbase(main):001:0> list
```

Verify whether the table 'test1' has been created or not

Then run the following command from the hbase shell to verify the contents of the table

```
hbase(main):001:0> scan 'test1'
```

22.5 Options file

Let's list all tables in mysqldatabase using options file. Create a file "import.txt" in user home directory ie /home/training/import.txt. Edit and paste following

```
$ sqooplist-tables–connectjdbc:mysql://localhost/hadoopguide
```

Save and close this file. Now run following

```
$ sqoop --options-file import.txt
```

Above will result in showing widgets table

22.6 Saved jobs/incremental import

One of the attractive features given by sqoop is saved job which is used by incremental imports

First let's check if we have any job existing

```
$sqoop  job  -list
```

Now, create a new job by typing below

```
$ sqoop job --create wjob -- import --connect
jdbc:mysql://localhost/hadoopguide--table widgets
```

```
--check-column id --incremental append --last-value 3
```

Please notice a " " space between -- and import.

Again check metastore for the saved jobs

```
$ sqoop job –list
```

Creating a job won't import it. You still need to execute. So do it

```
$ sqoop job --exec wjob
```

Check the contents of imported directory. Check the number of part files (now m is not equal to 1!) it will import rows from id=4 to id=10 in total 4 mapper files

```
$ hadoopfs –cat widgets/*
```

Execute the job again

```
$ sqoop job --exec wjob
```

This will not do anything since no change in table(Sqoop remembers the last-value parameter after each import…)

Now create another row in mysql

```
mysql> INSERT INTO widgets VALUES (NULL, 'new',145.00,
'2007-10-05',64, 'new new');
```

Again try

```
$ sqoop job --exec wjob\
```

This time it will import 11th row in new mapper file

22.7 Using CodeGen

Run the following command to generate the java class

```
$sqoopcodegen  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --table testTable --outdirsrc/main/java --class-name com.example.SqoopImport
```

Verify that a file by name “SqoopImport.java” should have been created under “src/main/java/com/example” folder

22.8 Exporting the data

Exporting the data from HDFS to RDBMS table

Assuming that data has been already present on HDFS, run the following command to export the data from HDFS to SQL table

```
$ sqoop export  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --table testTable --export-dir /user/training/testTable -m 1
```

22.8.1 Exporting the data from Hive table to RDBMS table

If you are exporting the data from the hive table to RDBMS table, you need to provide how the input fields are terminated in your hive table

Run the following commands to export the data from hive table to the sql table

```
$sqoop export  --username=training --password=training  --connect jdbc:mysql://localhost/testDB --table testTable --export-dir /user/hive/warehouse/test1 -m 1 --input-fields-terminated-by '\0001'
```

Open the mysql shell and verify whether the data has been updated in the “testable” or not

23. Flume Hands On

23.1 Single Flume Agent Deployment

The netcat source will receive an event from an RPC client and displays those events on console

Open a terminal and go to $FLUME_HOME/bin folder

```
$ cd $FLUME_HOME/bin
```

Now run the flume agent as follows:

```
$./flume-ng  agent  --conf  ../conf  -f  ../conf/sample.properties  -n sample  -
Dflume.root.logger=INFO,console
```

The above command will start the flume agent.

Now open another terminal and telnet to port 44444 and send the flume and event

```
telnet localhost 44444
```

And then send flume an event by typing some words, and then check whether those events are being printed on the first terminal or not

NOTE: if telnet is not installed, enter as a root user and install telnet. The steps are as follows:

```
$su root
```

It will ask you the password; enter the password as "training"

```
$ yum install telnet
```

Once the installation is done, you can exit from the root user

```
$exit
```

Now do the telnet as explained above

23.2 Using RPC mechanism to send the entire file to Flume avro source

Let's first run the flume agent which consist of an avro source which is listening to port 41414 , memory channel and logger sink.

```
$ cd $FLUME_HOME/bin
```

Now run the flume agent

```
./flume-ng  agent  --conf  ../conf  -f  ../conf/rpcagent.properties  -n sample  -
Dflume.root.logger=INFO,console
```

Now let's send the entire file as an event to this above agent. Open another terminal

```
$cd  $FLUME_HOME/bin
```

Now run the following command to send an entire file to the above agent

```
$ ./flume-ngavro-client -H localhost -p 41414 -F ../conf/sample.properties
```

Verify that in the first terminal the contents of the file "sample.properties" are displayed on the console

23.3 Using RPC mechanism sending an entire file to HDFS

Start your hadoop cluster

```
$start-all.sh
```

Make sure that hadoop is not in safe mode

```
$hadoop  dfsadmin  -safemode get
```

Now run the flume agent where source is avro, channel is memory and sink is HDFS

```
$ cd $FLUME_HOME/bin
```

Now run the agent

```
$ ./flume-ng   agent  --conf  ../conf   -f  ../conf/rpc_to_hdfs.properties -n sample
```

Now open another terminal

```
$ cd $FLUME_HOME/bin
```

Now using RPC you send the entire file

```
$ ./flume-ng  avro-client -H localhost -p 41414 -F ../conf/sample.properties
```

Now once the file is sent, you can check whether the file is transferred to your HDFS or not

```
$ hadoop  fs   -ls
```

Verify that a directory by name “Flume” has been created on HDFS and inside Flume, you have multiple files created which consist of the data from the file. It is due to the reason, that file is rolling after every 10 events received or 30 sec whichever is first.

Here events are nothing but the number of lines

23.4 Running Exec Source

```
$ cd $FLUME_HOME/bin
```

Now run the agent which consist of exec source, memory channel and logger sink

```
$./flume-ng   agent  --conf  ../conf   -f  ../conf/execSource.properties -n sample-
Dflume.root.logger=INFO,console
```

23.5 Running Spool Directory Source

First let’s create a spool directory, where we will keep these files

```
$ mkdir /home/training/flumespool
```

Let’s copy one file into this directory

```
$ cp $FLUME_HOME/conf/sample.properties ~/flumespool/
```

Before running this source make sure that hadoop cluster is up and running, since the terminal sink is HDFS

```
$ cd $FLUME_HOME/bin
```

```
$ /flume-ngagent  --conf  ../conf   -f  ../conf/spooldirectory.properties -n sample
```

23.6 Using Sequence Generator Source

```
$ cd $FLUME_HOME/bin
```

```
$./flume-ng   agent  --conf  ../conf   -f  ../conf/SeqGeneratorSource.properties -n sample -
```

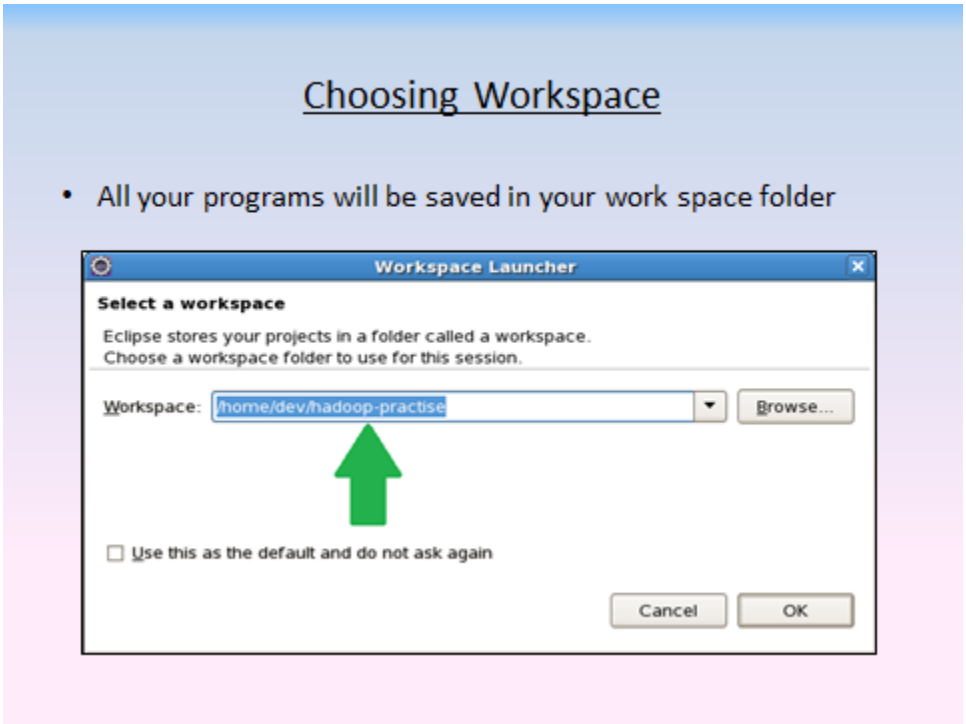
```
Dflume.root.logger=INFO,console
```

24. Appendix A

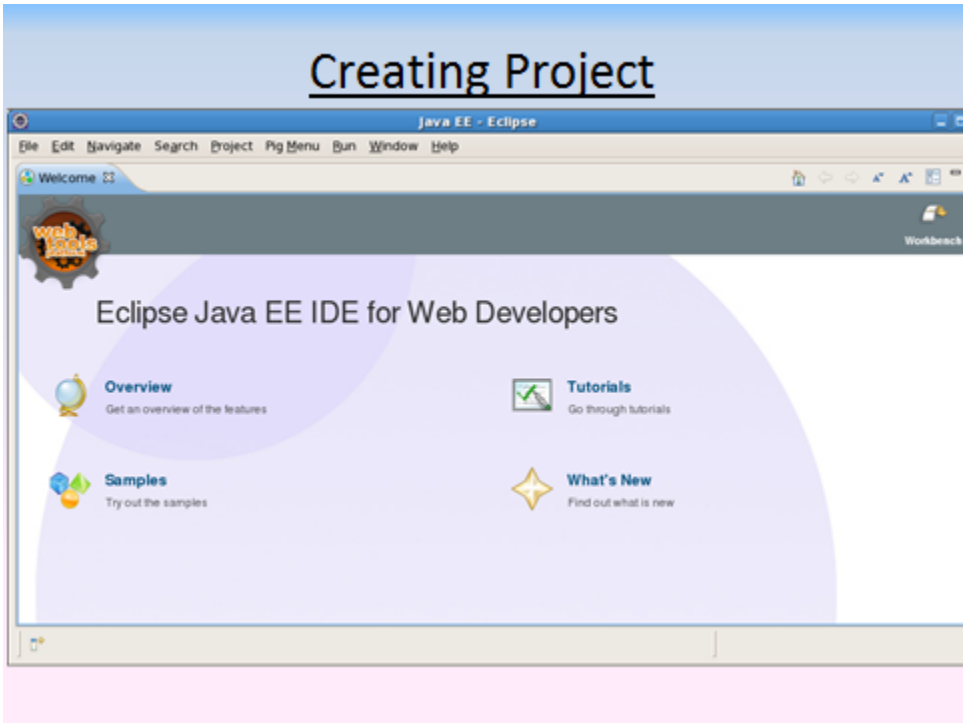
24.1 Creating java project in Eclipse and configuring build path for map reduce project

Follow the below screen shots to create a java project in eclipse and configuring the build path for your map reduce jobs

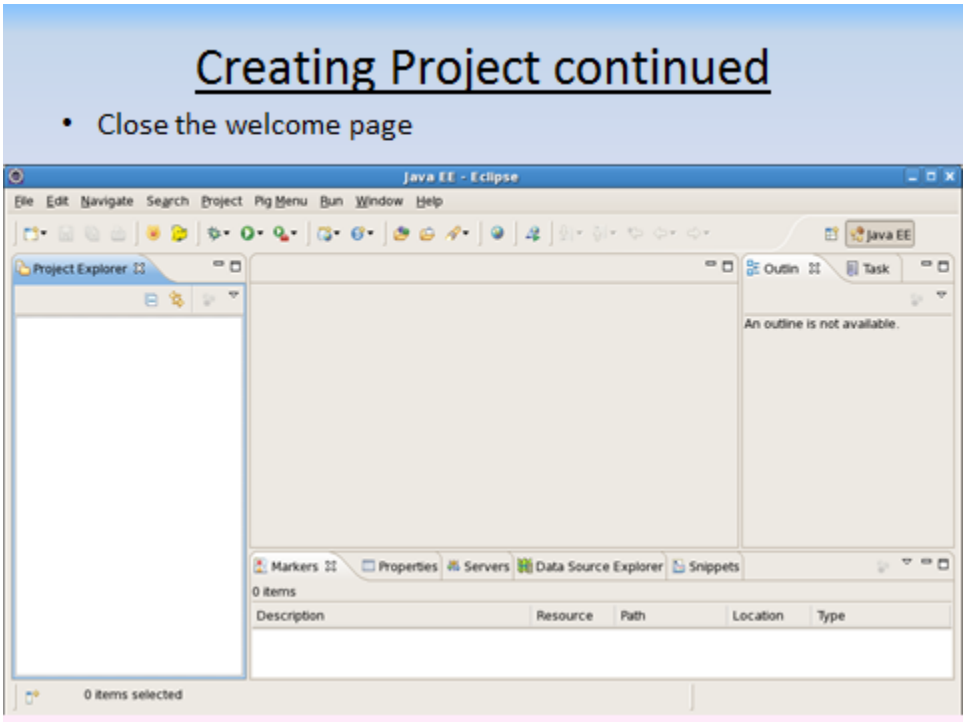
Step 1:



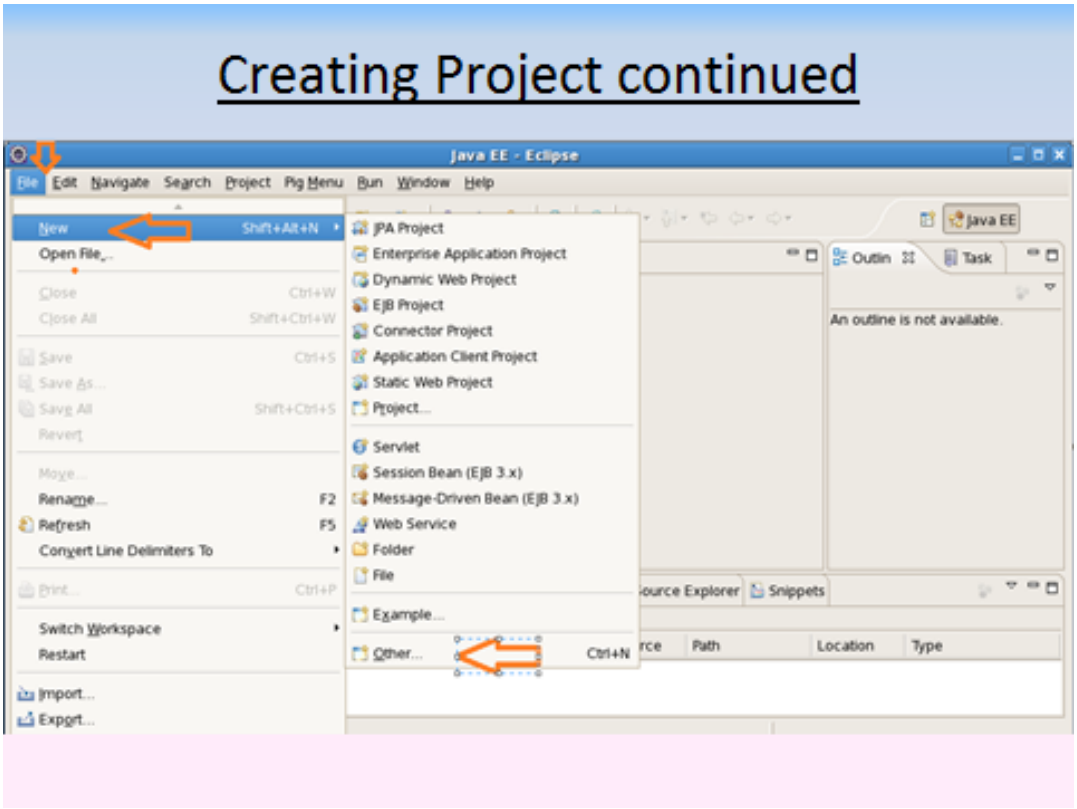
Step 2: Eclipse console when you click on the eclipse icon



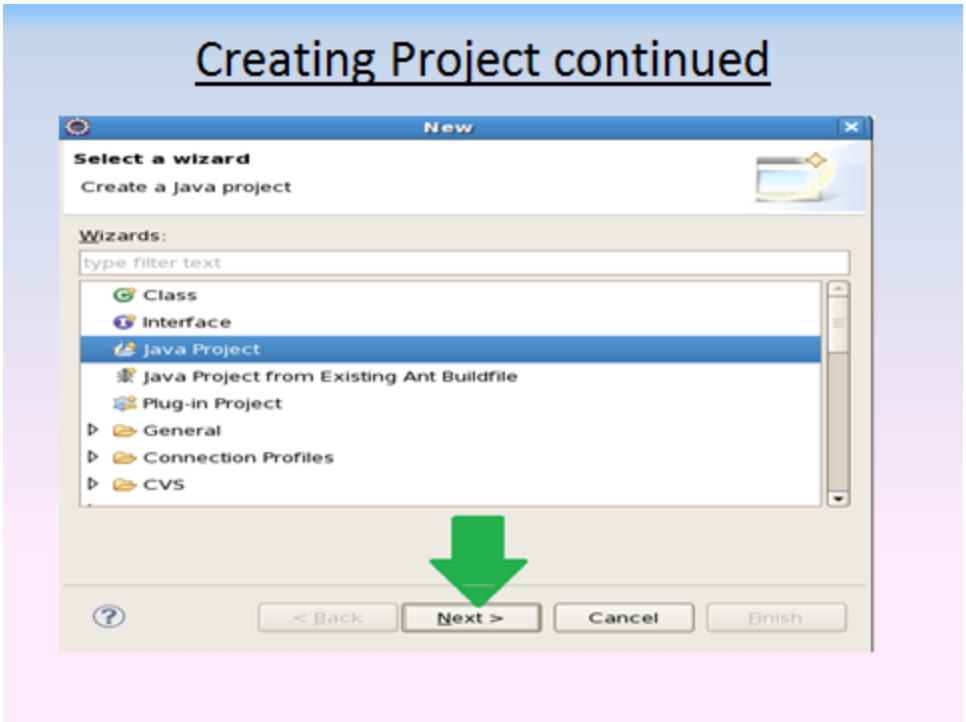
Step 3:



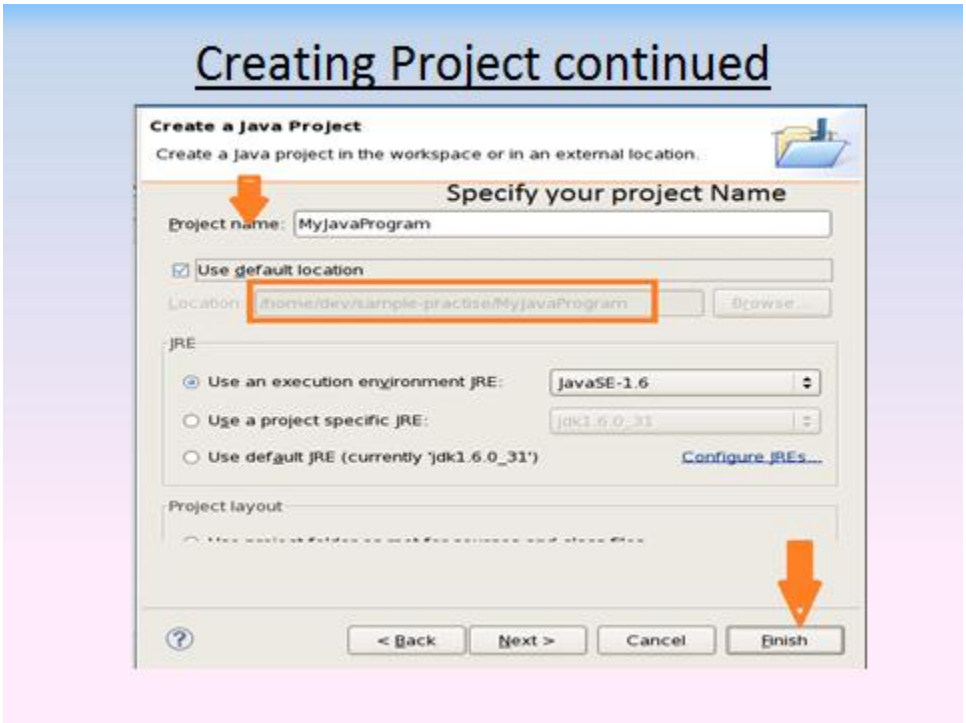
Step 4 :



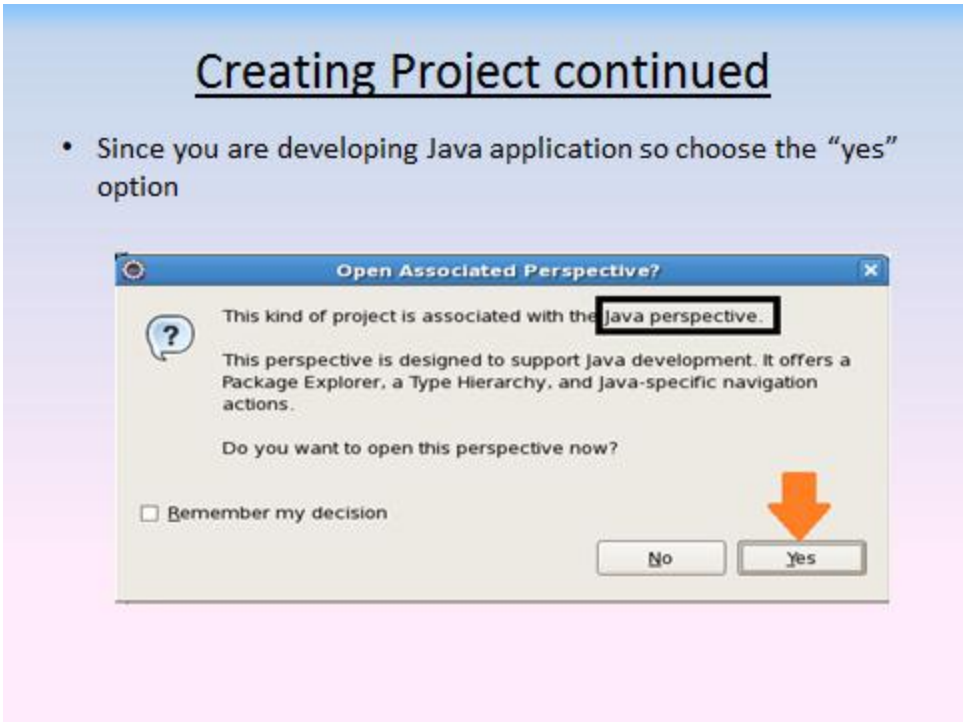
Step 5:



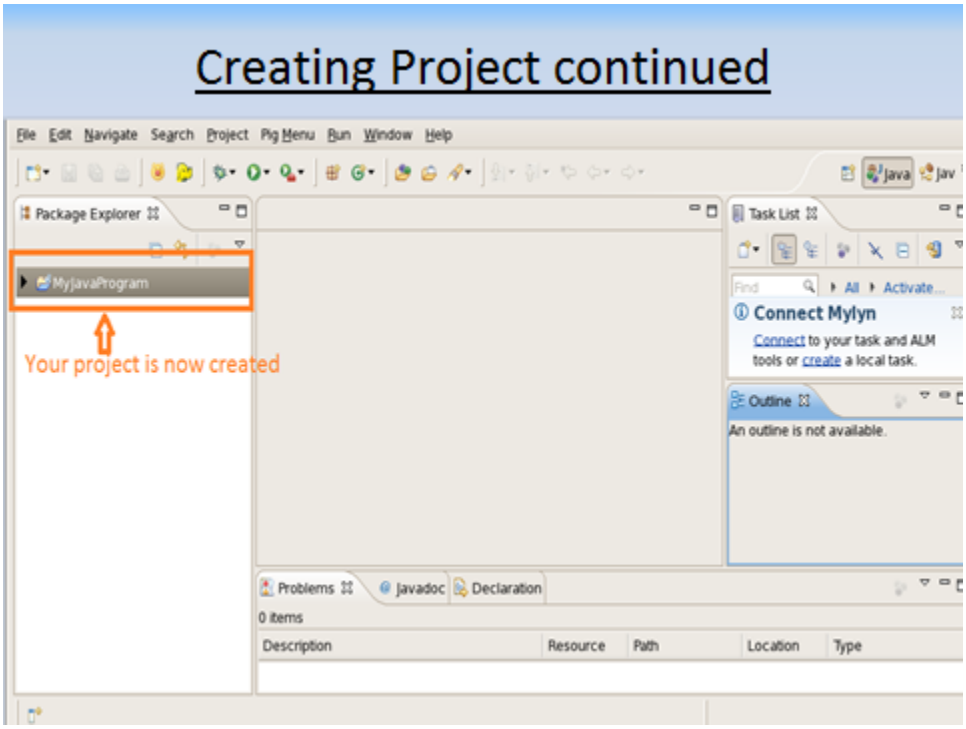
Step 6:



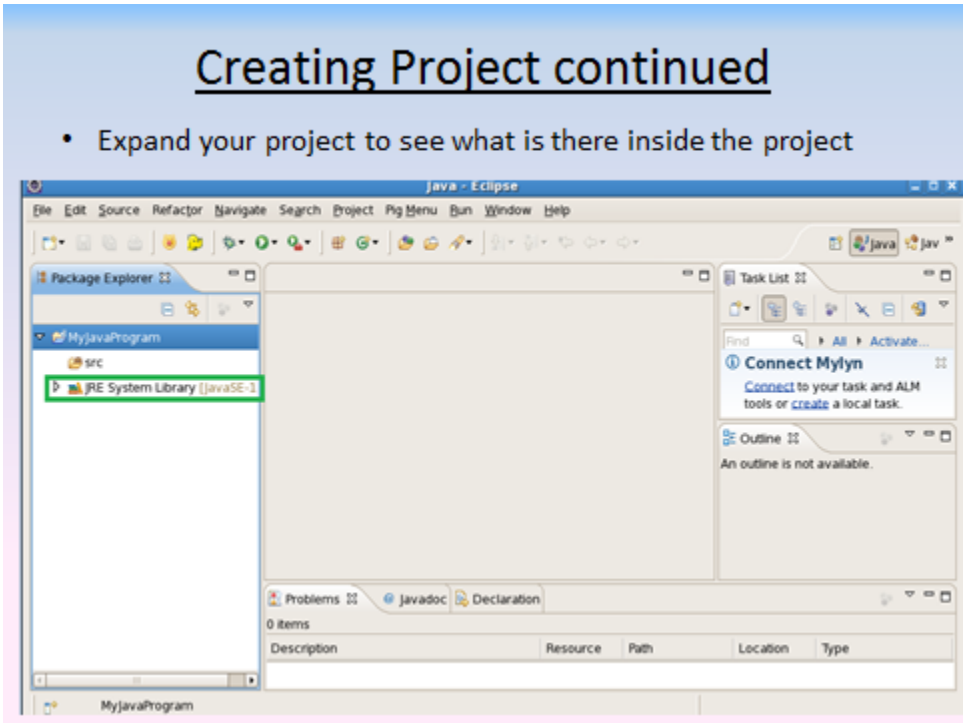
Step 7:



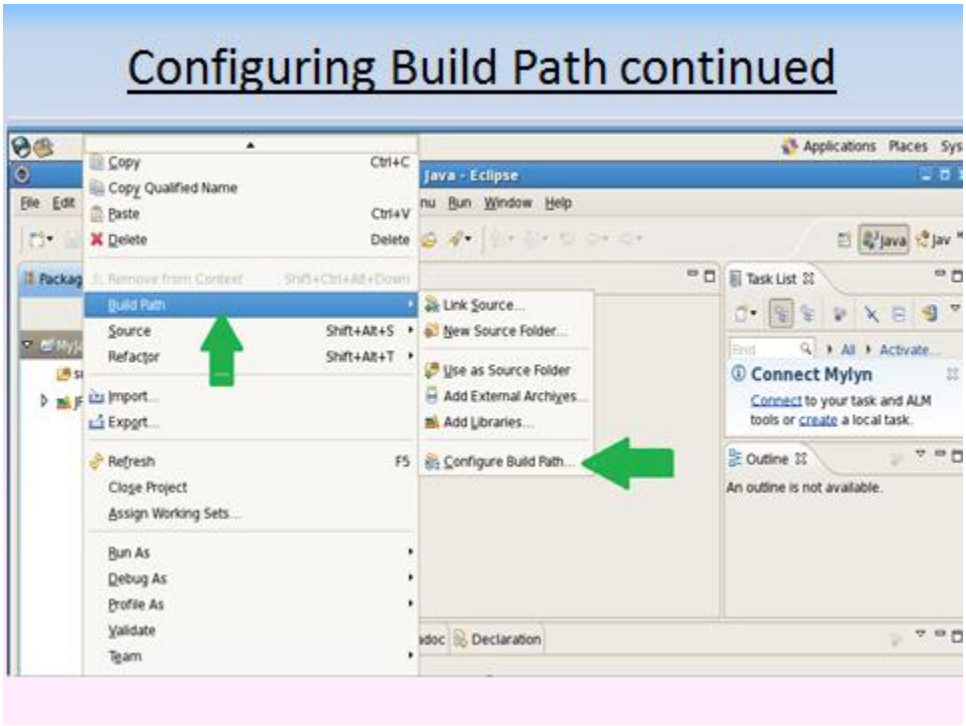
Step 8:



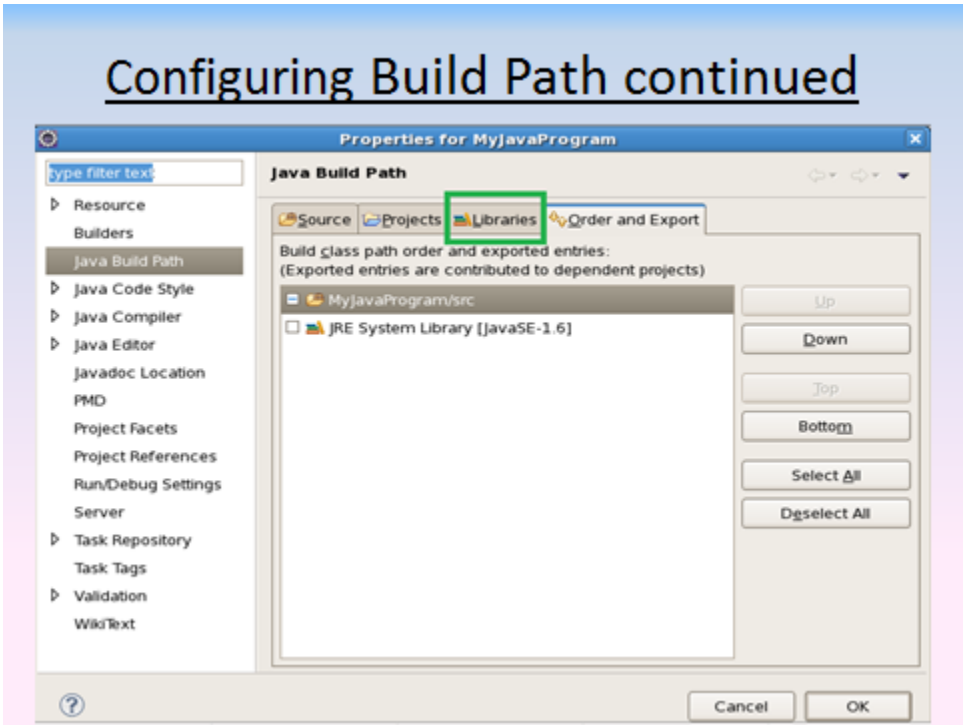
Step 9:



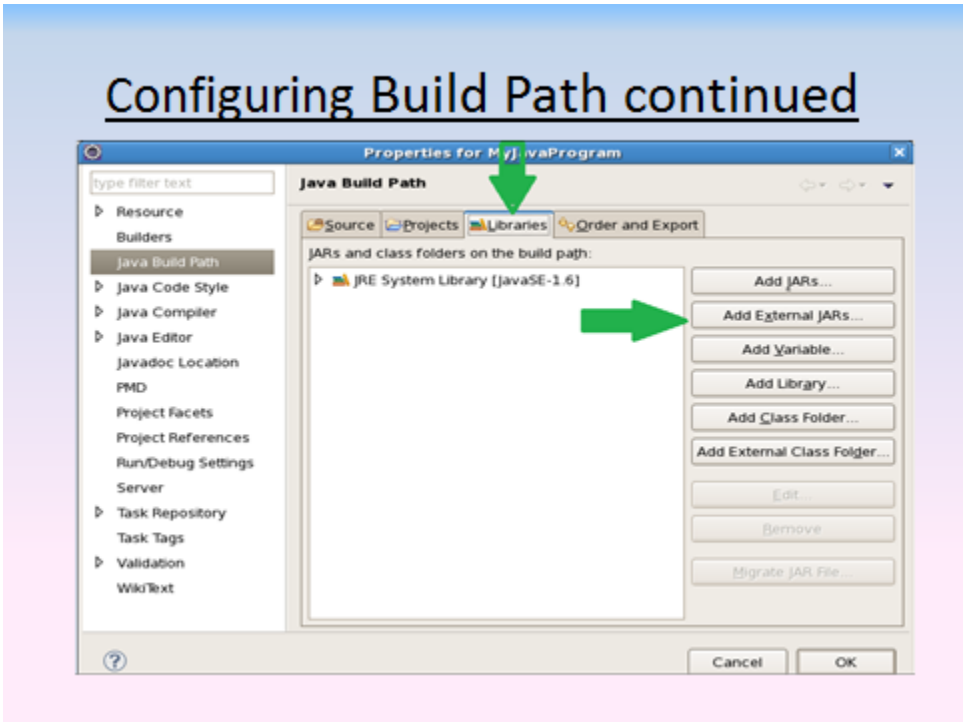
Step 10:



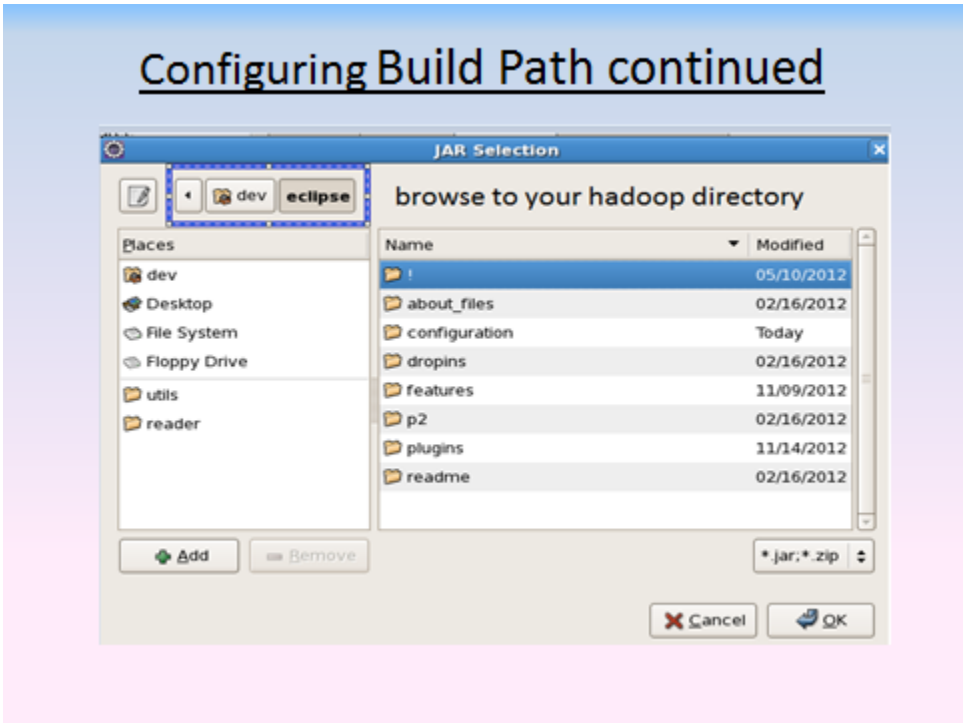
Step 11:



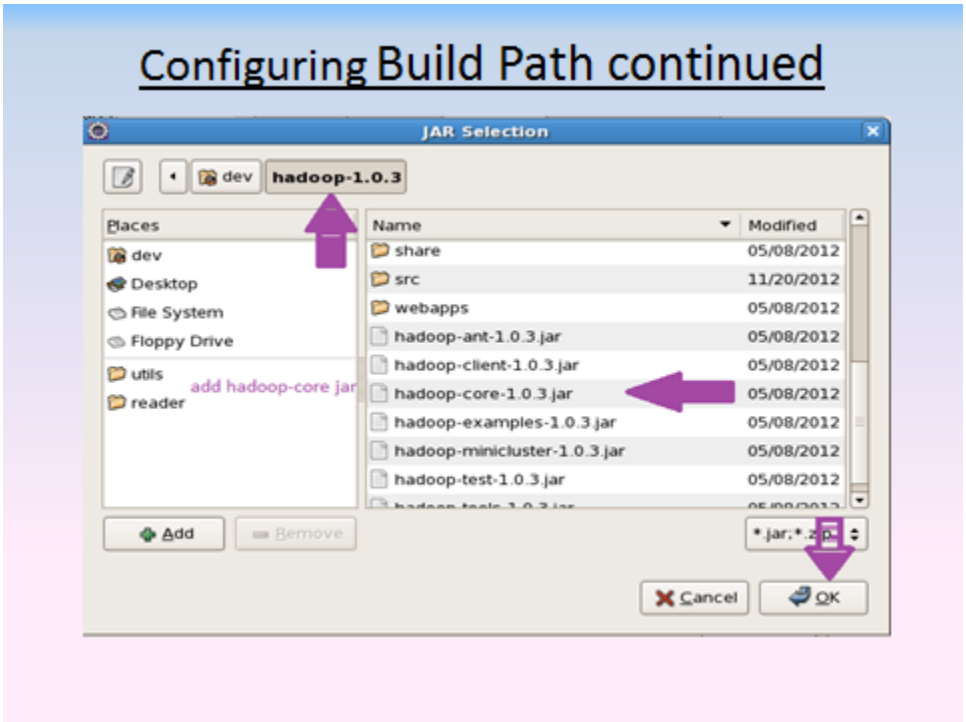
Step 12:



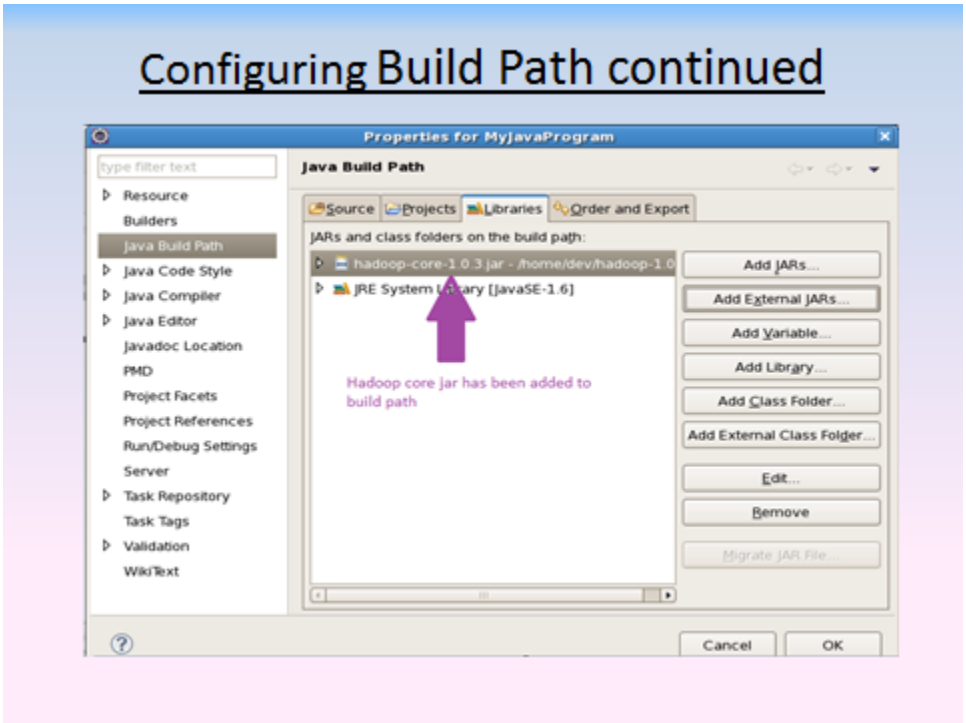
Step 13:



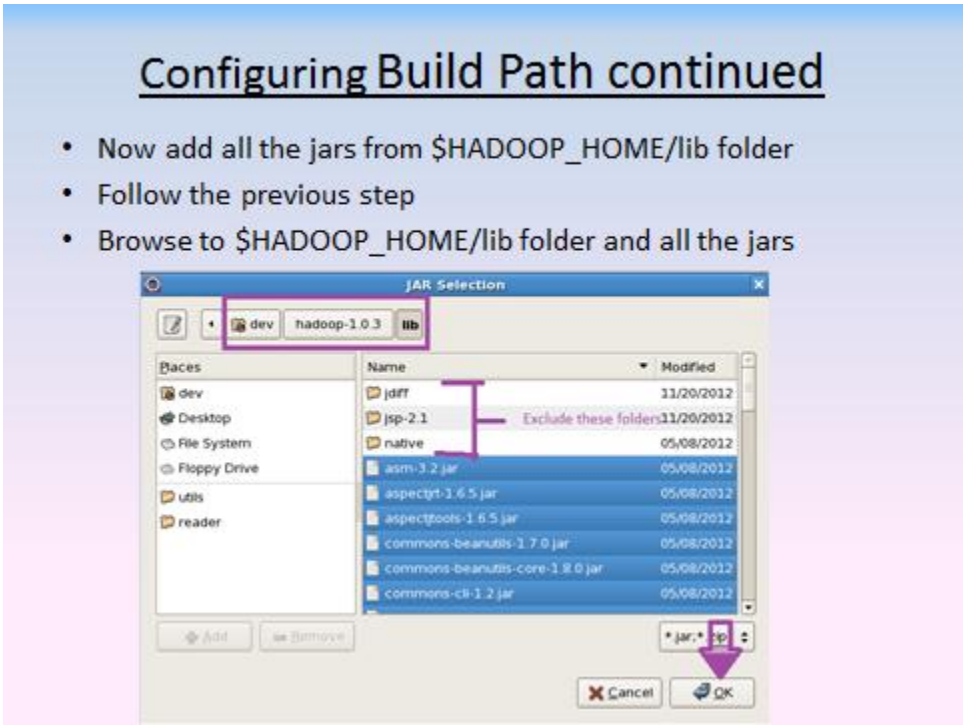
Step 14:



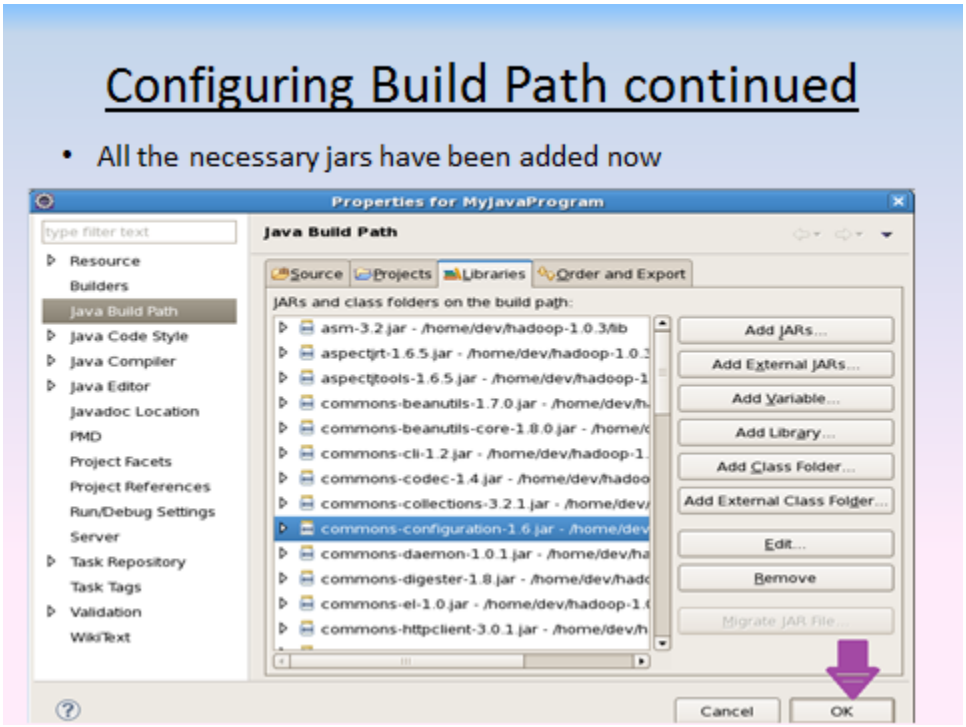
Step 15:



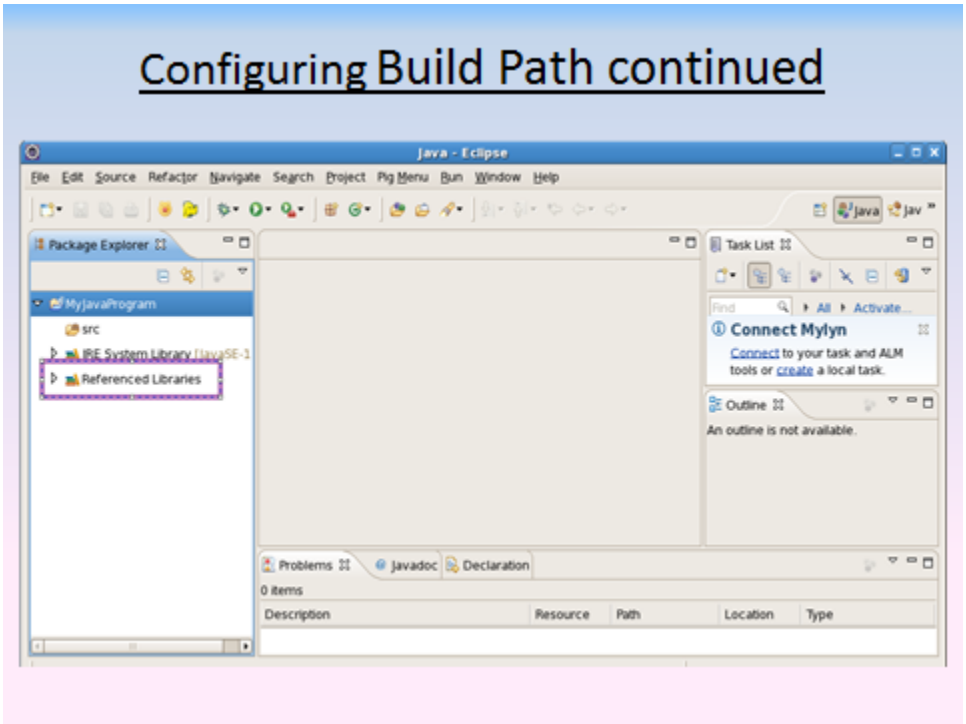
Step 16:



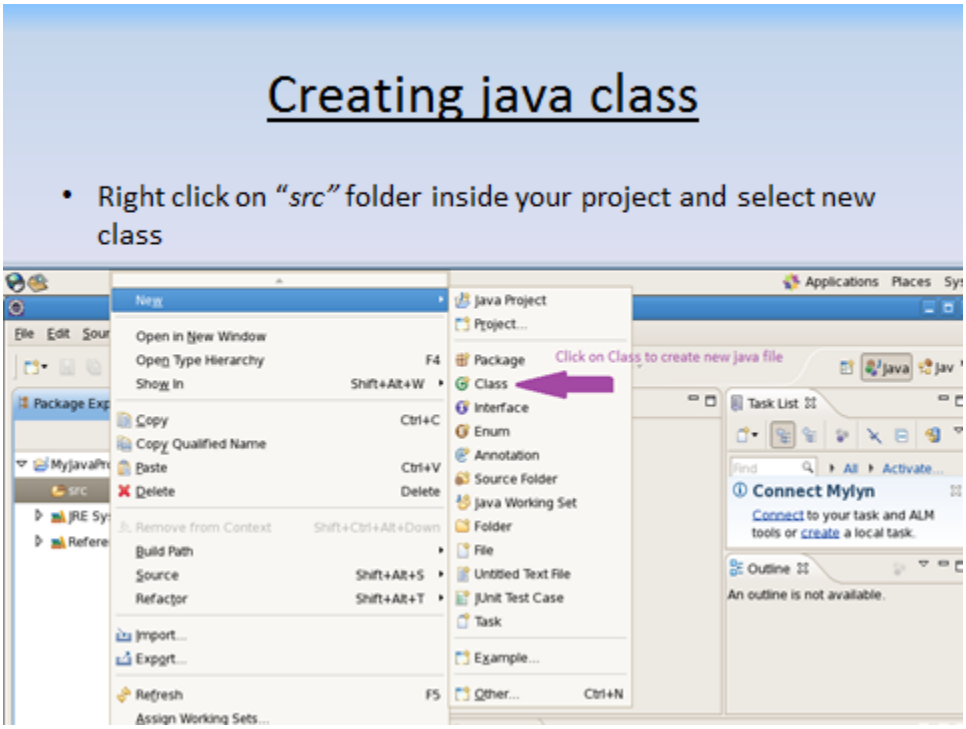
Step 17:



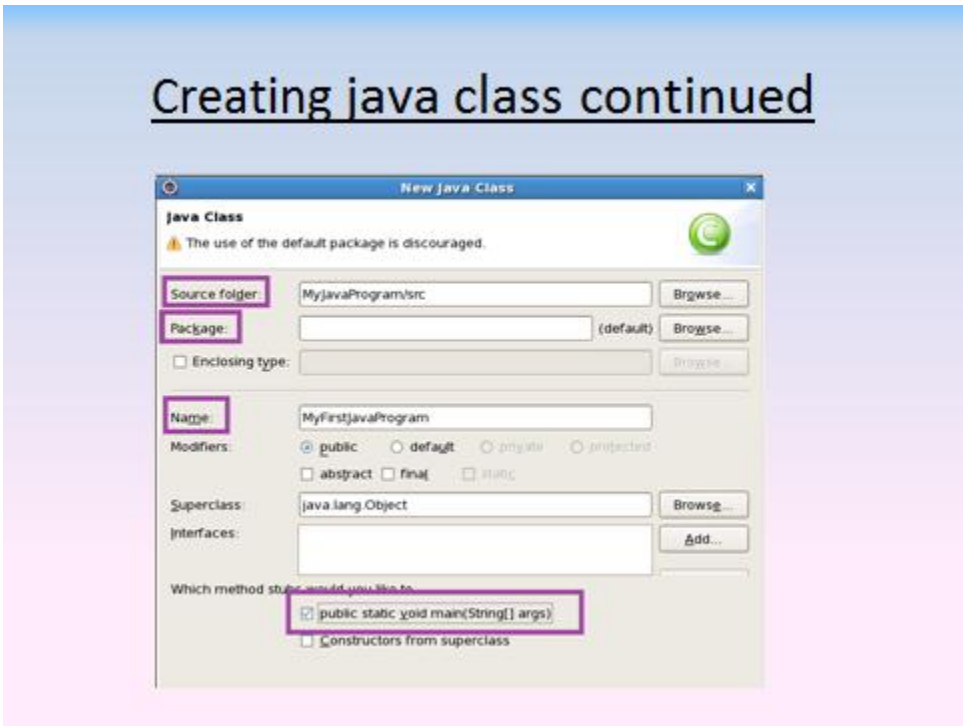
Step 18:



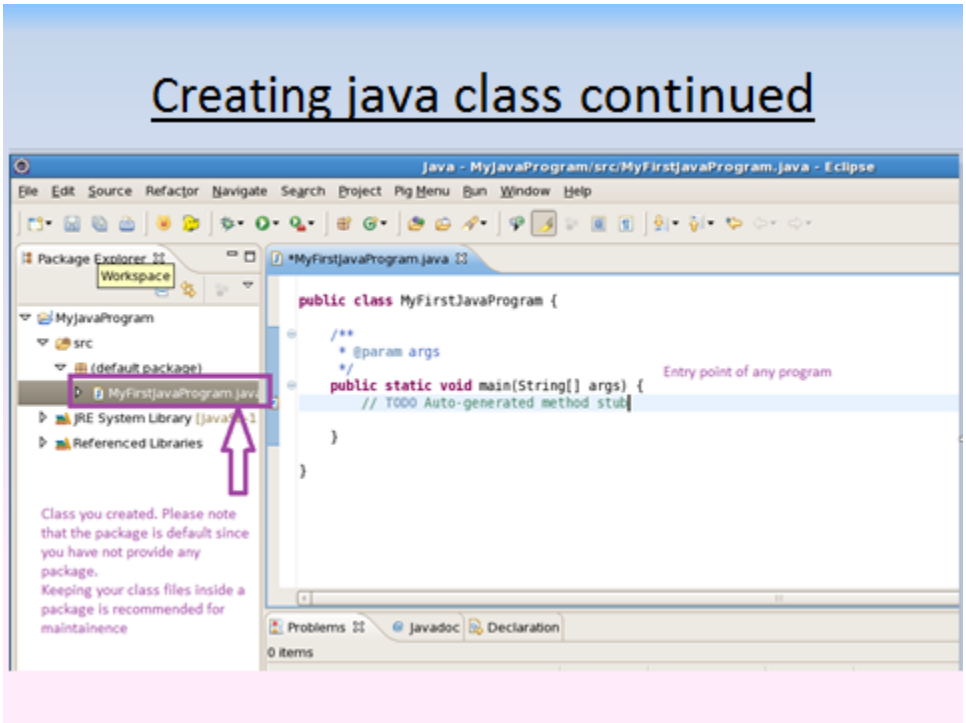
Step 19:



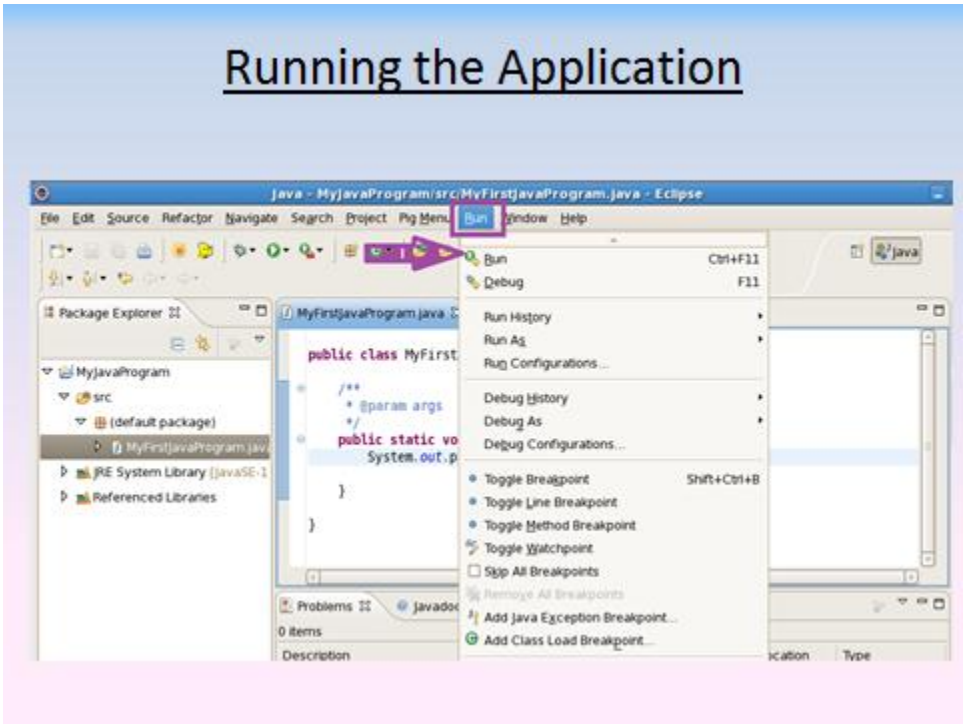
Step 20:



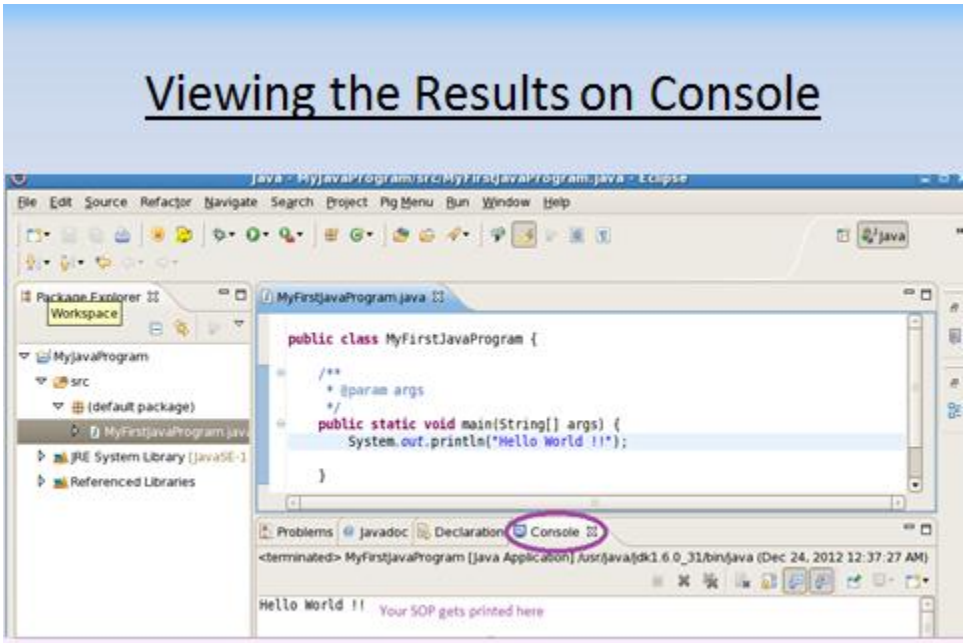
Step 21:



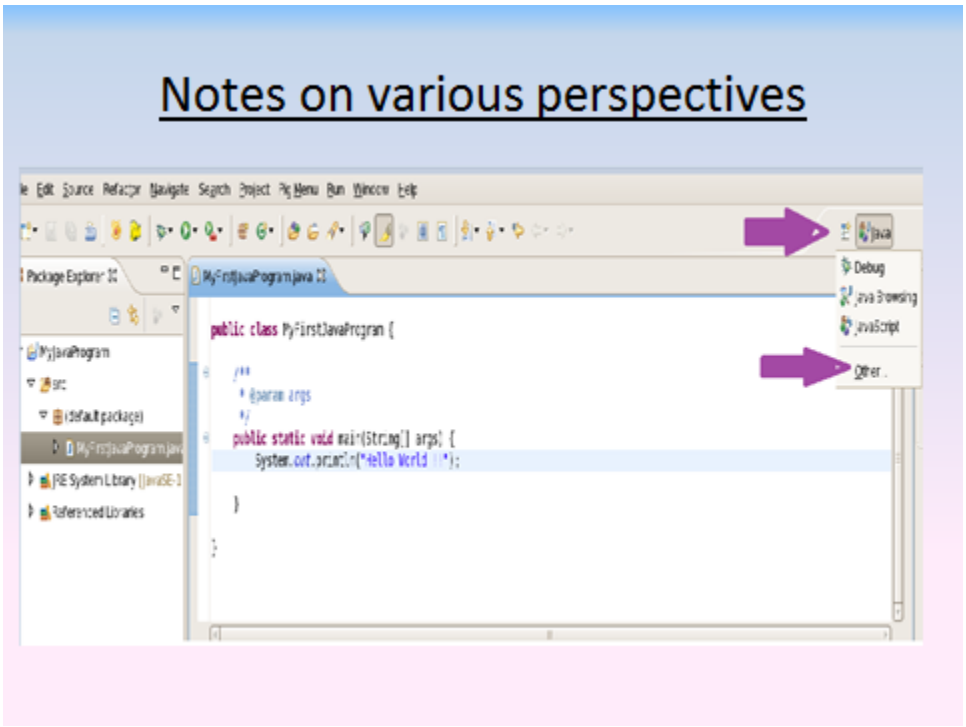
Step 22:



Step 23:

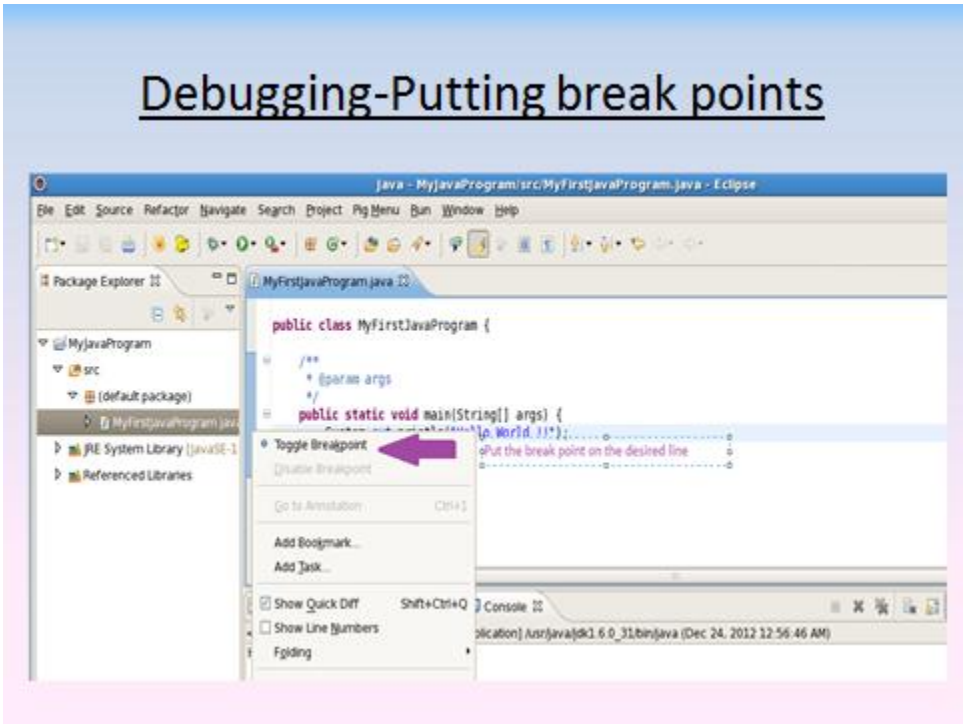


Step 24:

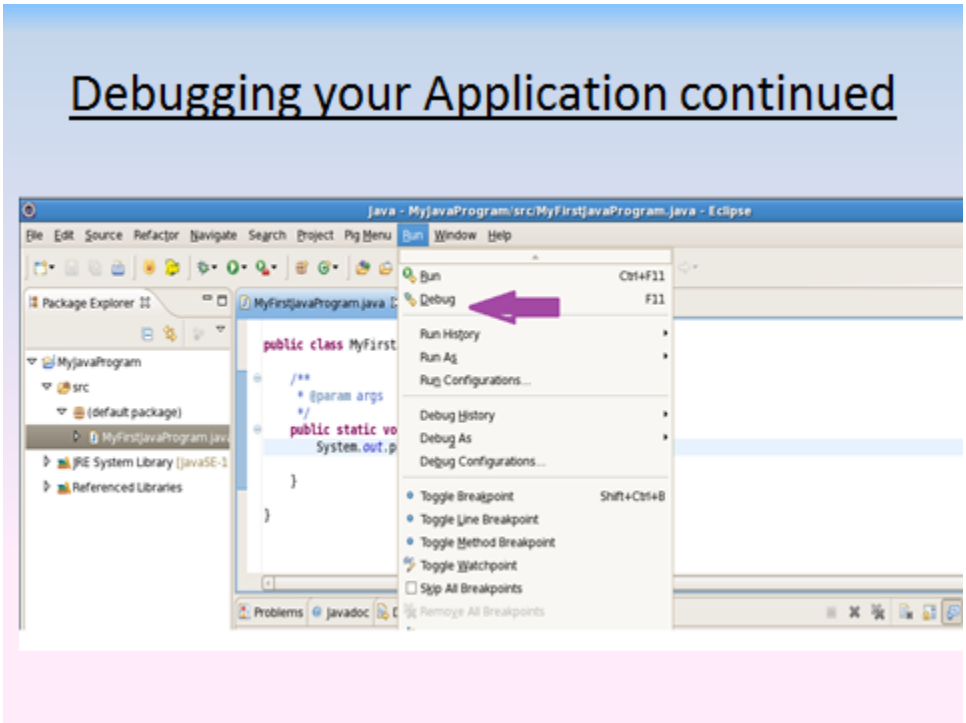


24.2 Debugging Java Project in Eclipse

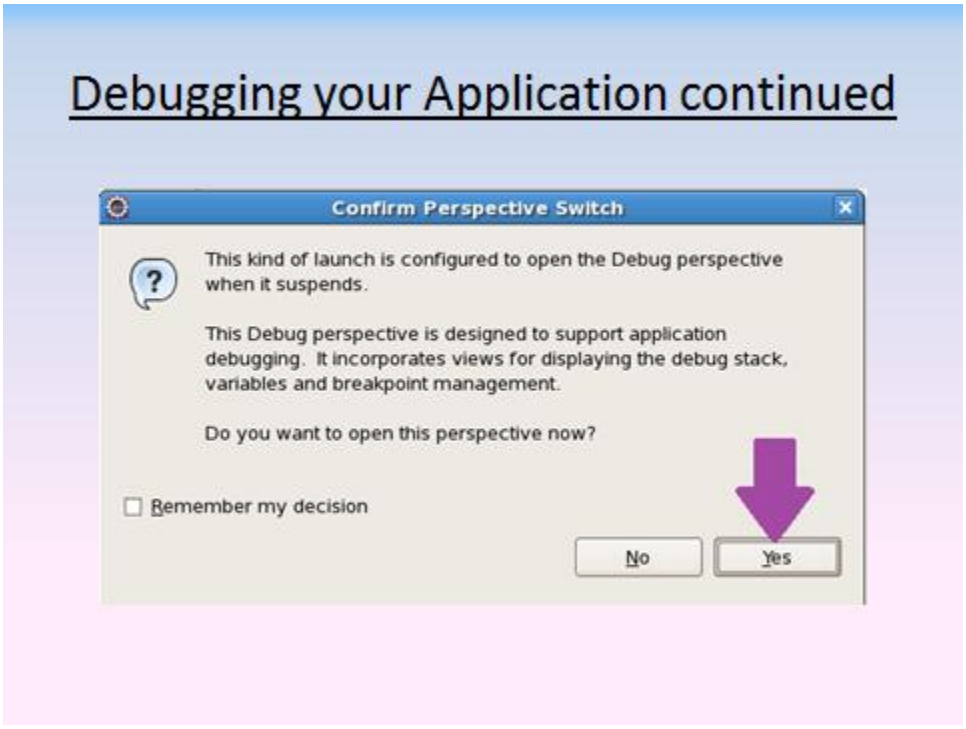
Step 1: Putting break points



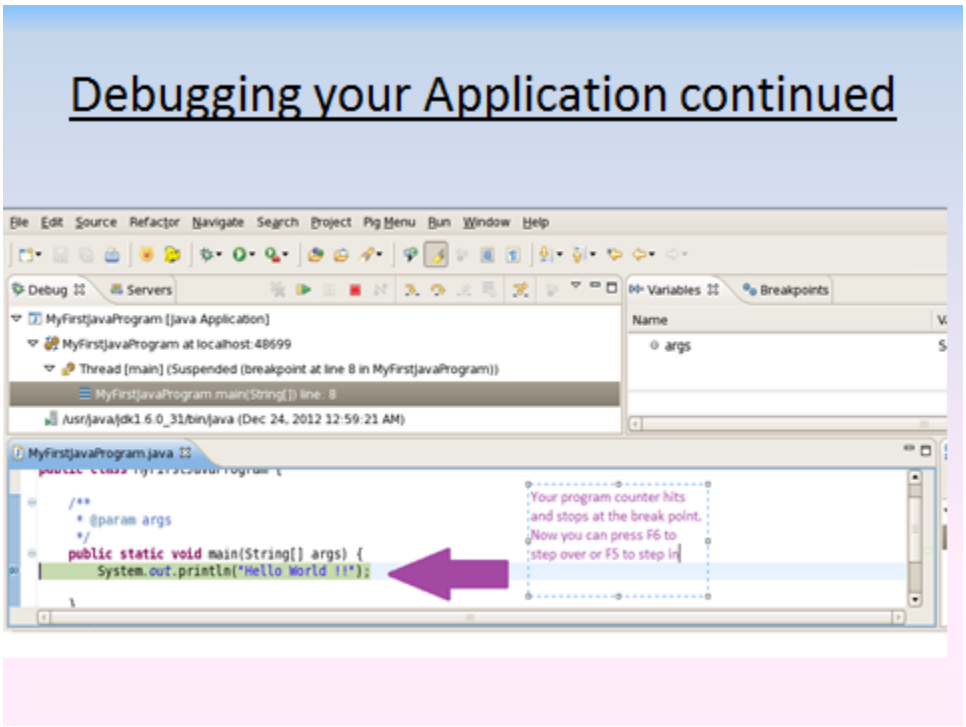
Step 2: Debug your application



Step 3:



Step 4:



25. Appendix B – Basic Linux command

25.1 Show me the current directory or present working directory

```
$ pwd
```

25.2 Listing the contents of file (Excluding hidden files)

```
$ ls
```

25.3 Listing the contents of file (Including hidden files)

```
$ ls  -a
```

25.4 Display the permissions of the files

```
$ ls  -l    OR  $ls  -ltr
```

25.5 Changing the Directory

```
$ cd <dir_name>
```

25.6 Going home directory of the user directly

```
$ cd
```

25.7 Navigating folders via home directory

```
$ cd   ~/dir1/dir2
```

The ~ sign represents the home directory of the user

25.8 Clearing the screen

```
$clear
```

25.9 Displaying the entire contents of file on screen

```
$cat <file_name>
```

25.10 Displaying top n lines of a file

```
$ head -5 <file_name>
```

The above command will display the top 5 lines of the given file name

25.11 Displaying the last n lines of a files

```
$ tail -5 <file_name>
```

The above command will display the last 5 lines of the given file name

25.12 Untarring a tar file (.tar.gz file)

```
$ tar   -zxvf  <file_name.tar.gz>
```

25.13 Unzipping a file (.gz file)

```
$ gunzip <file_name.gz>
```

25.14 Switching to root or any other user

```
$ su root
```

25.15 Installing a software from repository

```
$ yum install <software name>
```

26. Appendix C - Beginners Error while running MR Job

26.1 While running job got an exception Java.io.FileNotFoundException

If you are running a job as follows:

```
$hadoop  jar  <jar_name> <driver name> <input>  <output>
```

And if you got the error that jar file " <jar_name>" not found then it means that the jar is not present in the location from where you are running the command

First Way:

Go to the location where your jar is present and then run the command

Second Way:

Give the fully qualified path of the jar file. For example

```
$ hadooop   jar   /home/training/wordcount/wordcount.jar <driver_name>  <input_Date>  <output>
```

26.2 Class Not Found Exception

Test 1: You might have not created the jar properly. Whenever you are creating the jar from Eclipse, right click on your project (not on any java files or package) and then export.

Test2: It would be a class path problem. Try putting in CLASSPATH