# Big Data Hadoop Training

Session 3 Assignment 2 Solution:

**Task 4:**

**Q) Write a java program to list down all the files inside a directory and its sub-directories who have last modified timestamp between the start\_ts and end\_ts passed as an argument. The default value of start\_ts is 0 and for end\_ts is infinite.**

A) **Java Program**: Takes HDFS Path , start\_ts , end\_ts as input and display all the files and sub-directories in that HDFS path who have last modified timestamp between the start\_ts and end\_ts.

(**Note:** Assuming the HDFS path provided as input has Read permission for displaying the content of the directory)

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.net.URISyntaxException;

**import** java.sql.Timestamp;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileStatus;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**public** **class** BDDS3A2{

**public** **static** **void** main(String[] args) **throws** IOException, URISyntaxException

{

//1. Get the file path instance

Path filepath = **new** Path(args[0]);

//2. Get the Configuration instance

Configuration configuration = **new** Configuration();

//3. Get the instance of the HDFS - file system

FileSystem filesystem = FileSystem.*get*(filepath.toUri(), configuration);

//4. Get the metadata of the desired directory

FileStatus[] fileStatus = filesystem.listStatus(filepath);

//5. Default start and end timestamps

Timestamp start\_ts = **new** Timestamp(0);

Timestamp end\_ts = **new** Timestamp((**new** Double(Double.***POSITIVE\_INFINITY***)).longValue());

//5. Modify start and end timestamps if provided as input

**if**(args.length > 1) {

start\_ts = Timestamp.*valueOf*(args[1]);;

}

**if**(args.length > 2) {

end\_ts = Timestamp.*valueOf*(args[2]);

}

//6. Call the function which displays the content of the directory

System.***out***.print("list of all the files who have last modified timestamp between ");

System.***out***.println("Input start\_ts : "+start\_ts+" Input end\_ts: "+end\_ts);

*displayDirectoryContents*(fileStatus, start\_ts, end\_ts);

}

**public** **static** **void** displayDirectoryContents(FileStatus[] status, Timestamp fromtime, Timestamp totime) **throws** FileNotFoundException, IOException {

**for** (**int** i = 0; i < status.length; i++) {

FileStatus fileStatus = status[i];

**if** (fileStatus.isDirectory()) {

Configuration conf = **new** Configuration();

FileSystem filesystem = FileSystem.*get*(fileStatus.getPath().toUri(), conf);

FileStatus[] subStatus = filesystem.listStatus(fileStatus.getPath());

Timestamp mytimedir = **new** Timestamp(fileStatus.getModificationTime());

**if**(mytimedir.after(fromtime) && mytimedir.before(totime)) {

System.***out***.print("directory:" + fileStatus.getPath());

System.***out***.println(" last modified timestamp:" + mytimedir.toString());

}

*displayDirectoryContents*(subStatus, fromtime, totime);

} **else** {

Timestamp mytimefile = **new** Timestamp(fileStatus.getModificationTime());

**if**(mytimefile.after(fromtime) && mytimefile.before(totime)) {

System.***out***.print(" file:" + fileStatus.getPath());

System.***out***.println(" last modified timestamp is:" + mytimefile.toString());

}

}

}

}

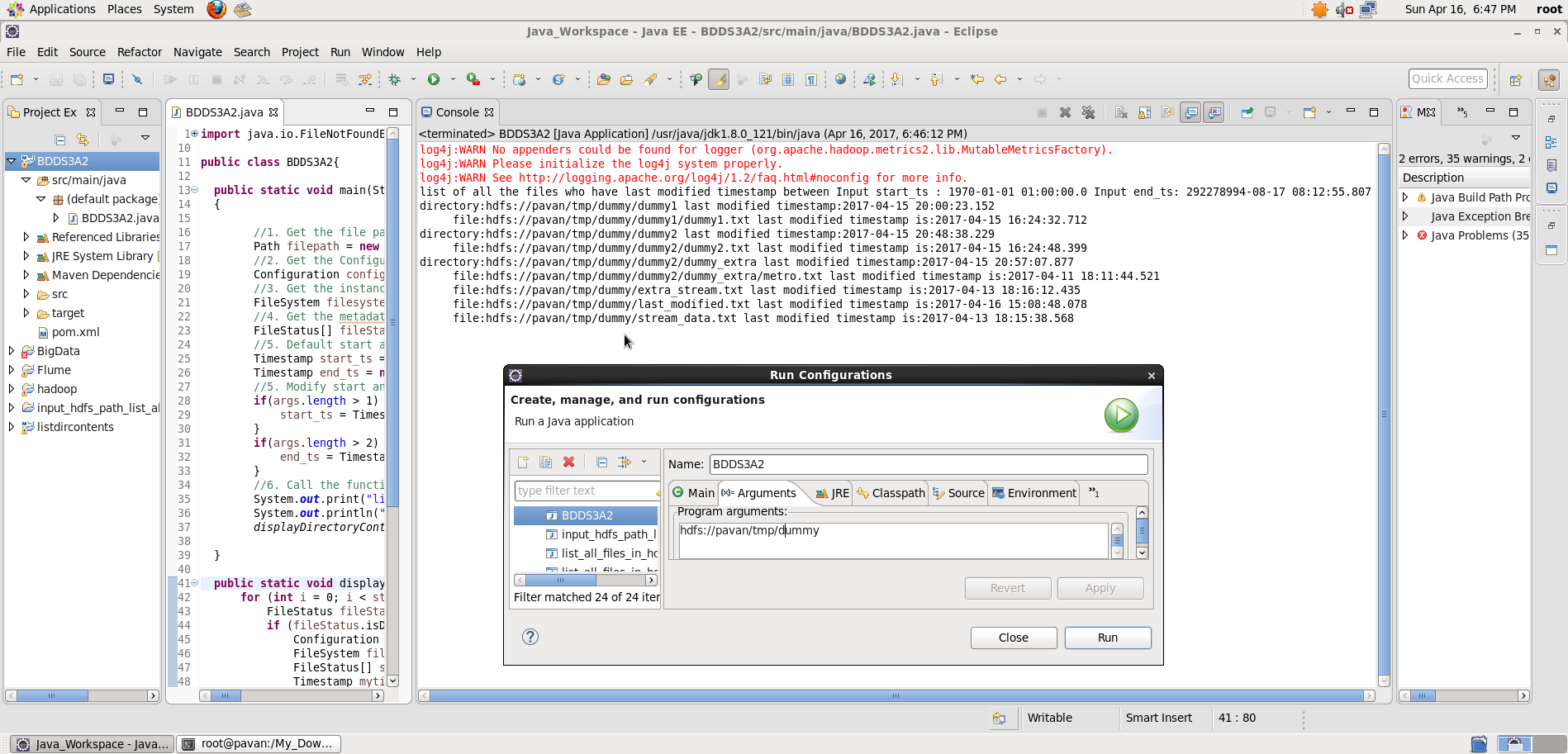
}

**Input format (No start\_ts and end\_ts provided): “**hdfs://<hostname>:<port\_no>/<HDFS\_File\_Path>**”**

**Ex: “**hdfs://pavan/tmp/dummy**” here, hostname :** pavan , **HDFS\_File\_Path :** tmp/dummy

**Running in Eclipse :**

* Input given in Run Configurations Argument Tab as : “hdfs://pavan/tmp/dummy”
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output in Console (Right side) - displays all the files and sub-directories in that HDFS path since default **start\_ts is zero and end\_ts is infinite.**



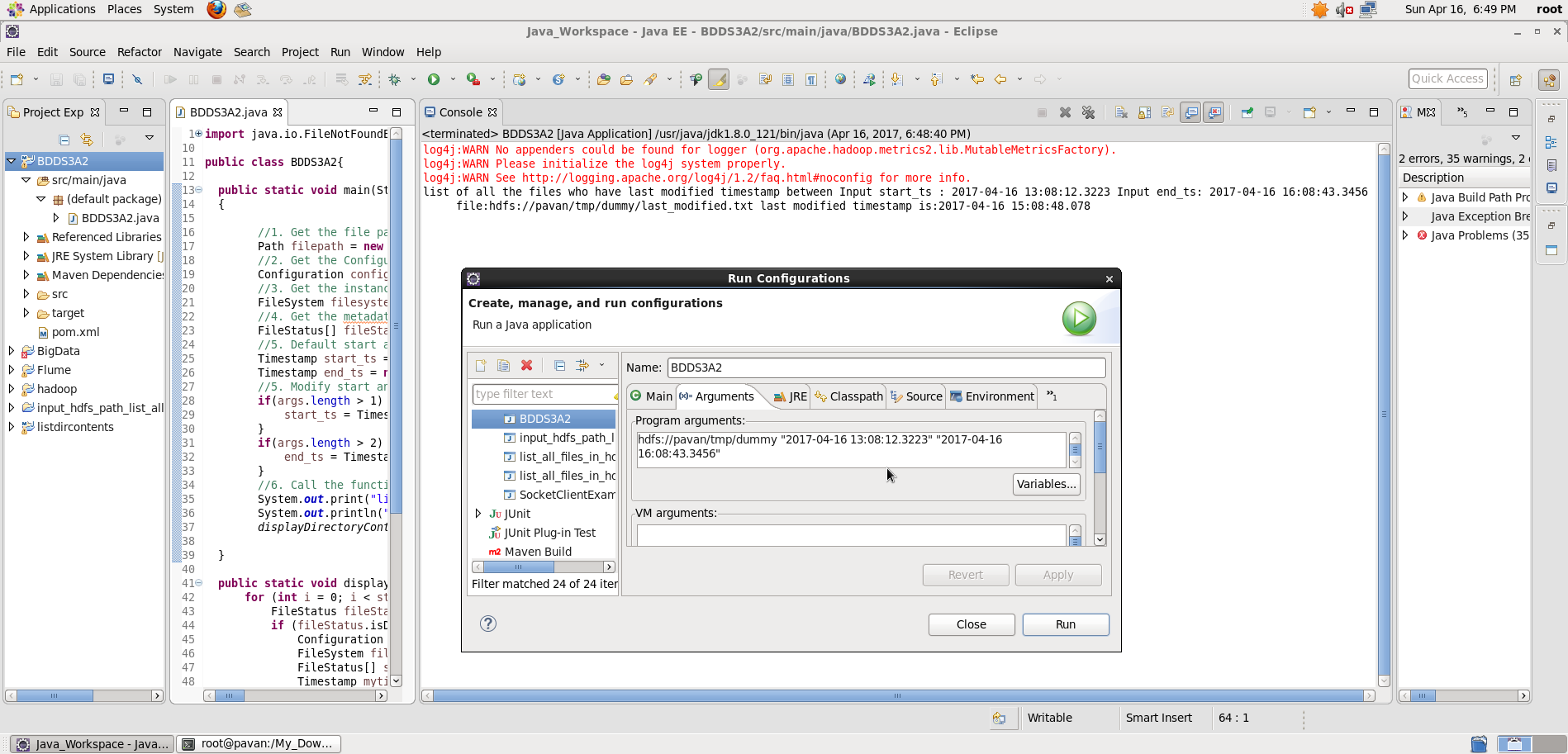
**Input format (with start\_ts and end\_ts provided): “**hdfs://<hostname>:<port\_no>/<HDFS\_File\_Path>**” “start\_ts” “end\_ts”**

**Start-ts & end\_ts input format : “yyyy-mm-dd HH:MM:SS(.fffffff)” – provided this for easier user input**

**Ex: “**hdfs://pavan/tmp/dummy**” “2017-04-16 13:08:12.3223” “2017-04-16 16:08:43.3456” here, hostname :** pavan , **HDFS\_File\_Path :** tmp/dummy  **start\_ts : 2017-04-16 13:08:12.3223 end\_ts : 2017-04-16 16:08:43.3456.**

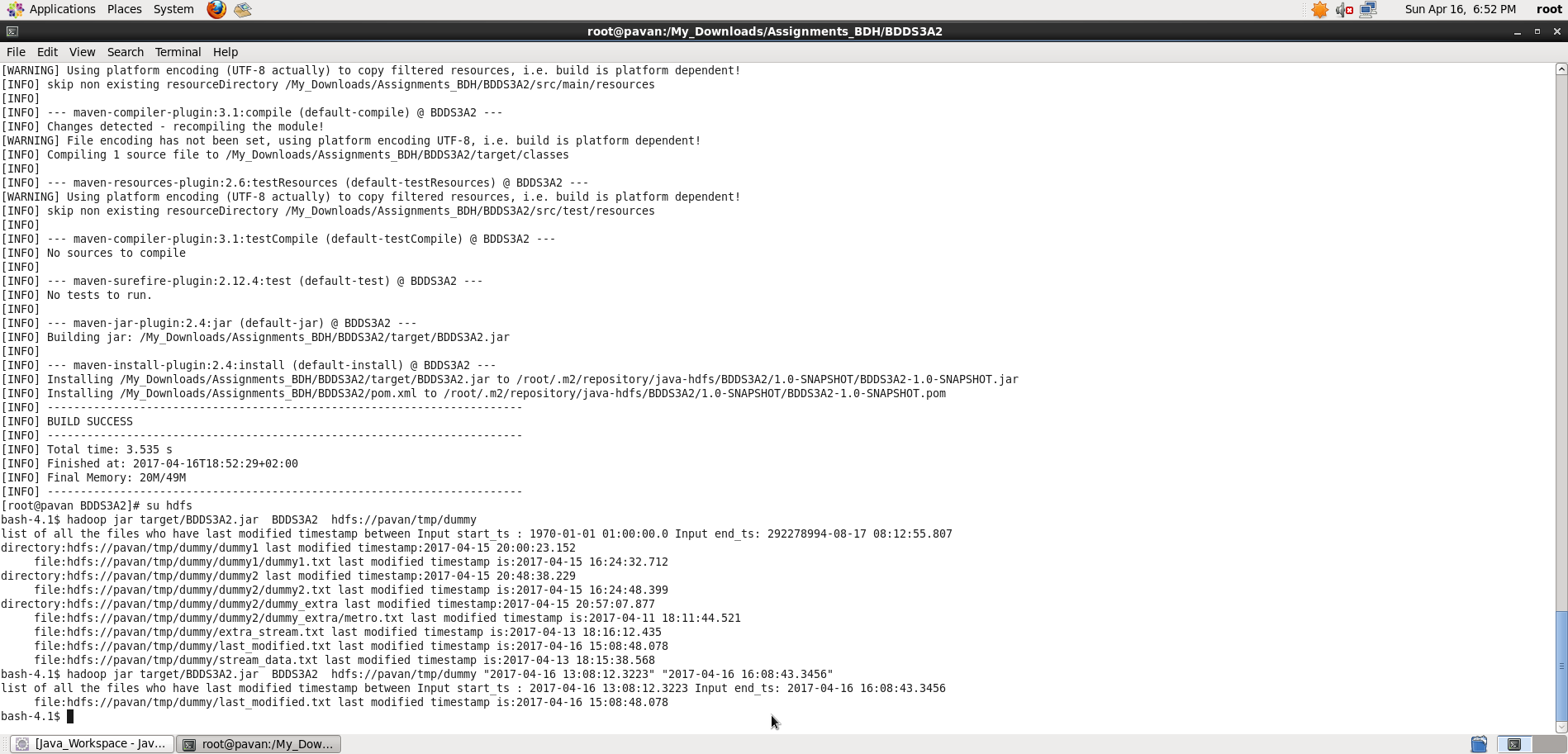
**Running in Eclipse :**

* Input given in Run Configurations Argument Tab as : **“**hdfs://pavan/tmp/dummy**” “2017-04-16 13:08:12.3223” “2017-04-16 16:08:43.3456”**
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output in Console (Right side) - displays the only file that got last modified between the start\_ts and end\_ts timestamps specified as input



**Running in Command Line:**

* Make the folder structure BDDS3A2/src/main/java/BDDS3A2.java
* Then in BDDS3A2 folder, open terminal
* Create the Jar file target/BDDS3A2.jar using Maven # mvn clean install
* Run the Hadoop Job : “hadoop jar <jar\_file\_path> <class\_name> <input\_hdfs\_file\_path>” “<start\_ts>” “<end\_ts>”
* In our case : “hadoop jar target/BDDS3A2.jar BDDS3A2 hdfs://pavan/tmp/dummy” **“2017-04-16 13:08:12.3223” “2017-04-16 16:08:43.3456”**
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output gets displayed on the screen



**Task 5:**

**Q) Write a Java program to display the content of a file in HDFS on screen.**

A) **Java Program**: Takes HDFS file Path and display the content of the file

(**Note:** Assuming the HDFS path have Read permission for displaying the content)

**import** java.io.IOException;

**import** java.net.URISyntaxException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FSDataInputStream;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IOUtils;

**public** **class** BDDS3A2{

**public** **static** **void** main(String[] args) **throws** IOException, URISyntaxException

{

//1. Get the file path instance

Path filepath = **new** Path(args[0]);

//2. Get the Configuration instance

Configuration configuration = **new** Configuration();

//3. Get the instance of the HDFS - file system

FileSystem filesystem = FileSystem.*get*(filepath.toUri(), configuration);

System.***out***.println("Contents of the Input File: "+args[0]);

//4. FSDataInputStream provides stream(channel) for reading data

FSDataInputStream in = **null**;

in = filesystem.open(filepath);

//5. IOUtils - An utility class for I/O related functionality

// copyBytes - Copies from one stream to another

IOUtils.*copyBytes*(in, System.***out***, configuration);

IOUtils.*closeStream*(in);

}

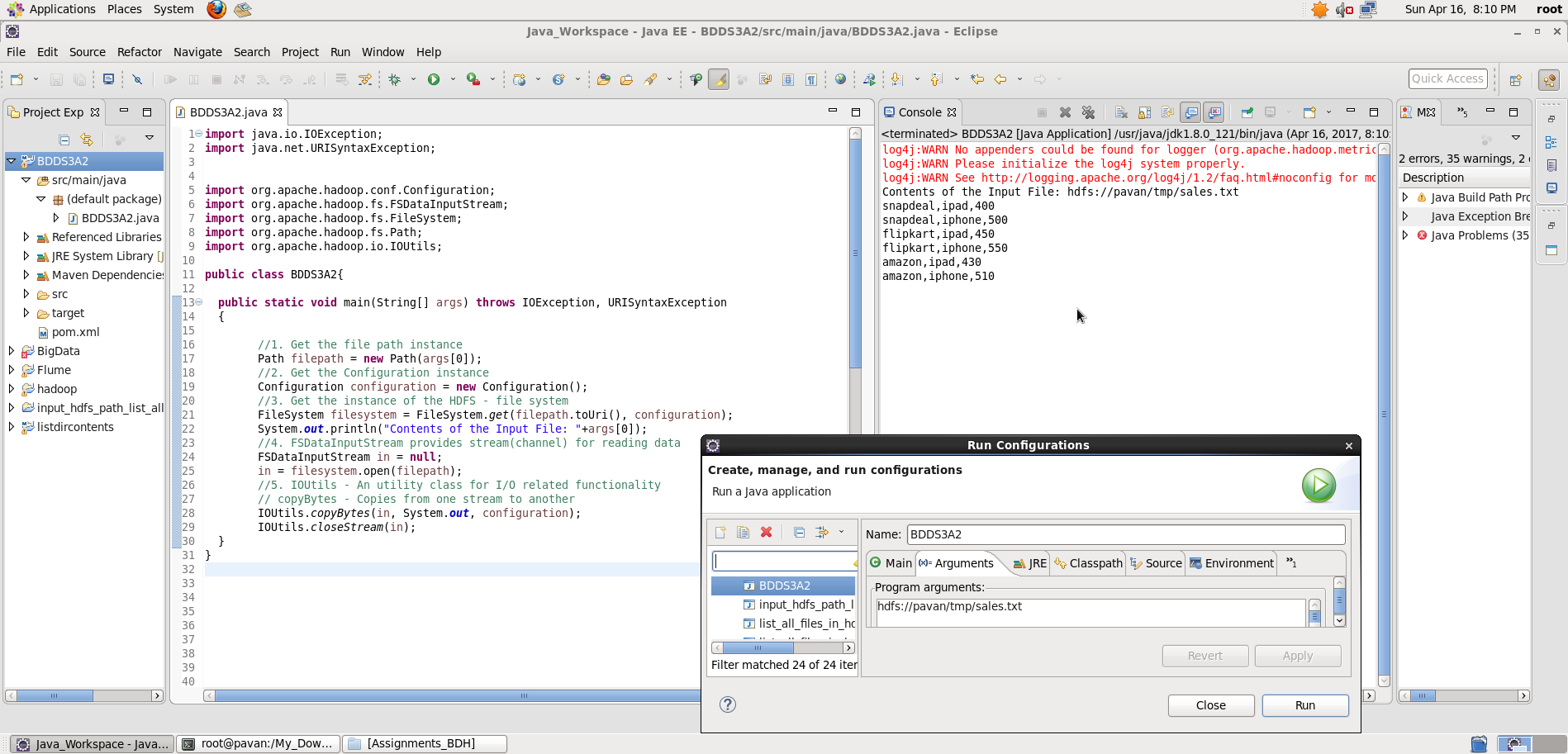
}

**Input format: “**hdfs://<hostname>:<port\_no>/<HDFS\_File\_Path>**”**

**Ex: “**hdfs://pavan/tmp/sales.txt**” here, hostname :** pavan , **HDFS\_File\_Path :** tmp/sales.txt

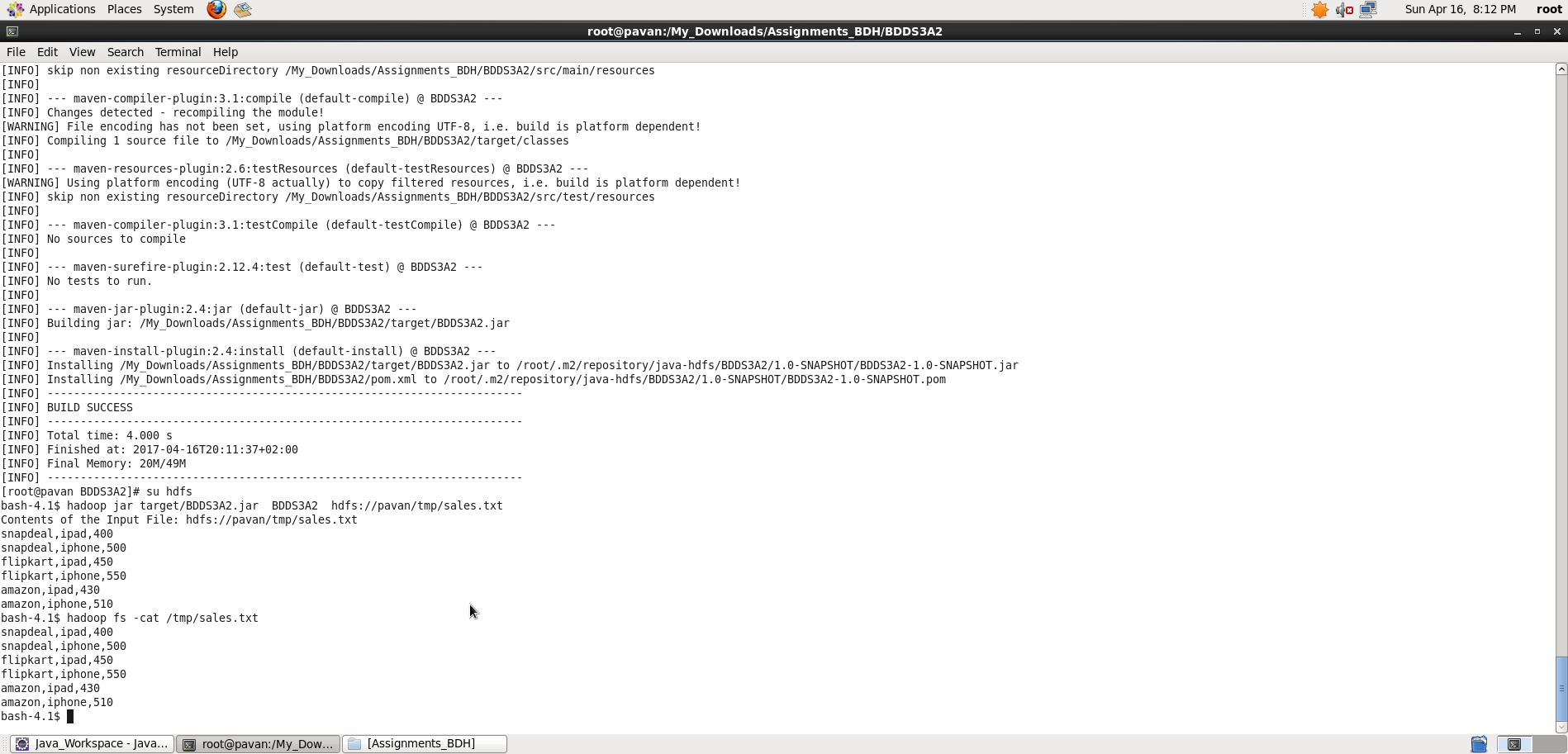
**Running in Eclipse:**

* Input given in Run Configurations Argument Tab as : “hdfs://pavan/tmp/sales.txt”
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output in Console (Right side) - displays the content of the file



**Running in Command Line:**

* Make the folder structure BDDS3A2/src/main/java/BDDS3A2.java
* Then in BDDS3A2 folder, open terminal
* Create the Jar file target/BDDS3A2.jar using Maven # mvn clean install
* Run the Hadoop Job : “hadoop jar <jar\_file\_path> <class\_name> <input\_hdfs\_file\_path>”
* In our case : “hadoop jar target/BDDS3A2.jar BDDS3A2 hdfs://pavan/tmp/sales.txt”
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* **Output gets displayed on the screen which is same as $hadoop fs –cat /tmp/sales.txt**



**Task 6:**

**Q) Write a Java program to copy a file from local filesystem to HDFS.**

A) **Java Program**: Takes local file path and HDFS filepath, copies local file into HDFS

(**Note:** Assuming the HDFS path has write permission for copying the file into the path)

**import** java.io.BufferedInputStream;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.net.URISyntaxException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FSDataInputStream;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IOUtils;

**public** **class** BDDS3A2 {

**public** **static** **void** main(String[] args) **throws** IOException, URISyntaxException {

// ARGUMENT FOR INPUT\_LOCATION AND OUTPUT\_LOCATION

Path localInputPath = **new** Path(args[0]);

Path outputPath = **new** Path(args[1]);

//2. Get the Configuration instance

Configuration conf = **new** Configuration();

//Displaying input contents of the local file

InputStream inp = **new** BufferedInputStream(**new** FileInputStream(localInputPath.toString()));

System.***out***.println("Contents of Input file : "+localInputPath.getName());

IOUtils.*copyBytes*(inp, System.***out***, conf, **false**); // Copying the dataset from input stream to print stream

// Copying a file in local file system into HDFS

FileSystem fs = FileSystem.*get*(outputPath.toUri(),conf);

fs.copyFromLocalFile(localInputPath, outputPath);

// Viewing the file contents copied into HDFS destination path

System.***out***.println("Contents of the copied file in HDFS :"+outputPath.toString());

//FileSystem fs\_out = FileSystem.get(outputPath.toUri(),conf);

FSDataInputStream in = **null**;

in = fs.open(outputPath);

//5. IOUtils - An utility class for I/O related functionality

// copyBytes - Copies from one stream to another

IOUtils.*copyBytes*(in, System.***out***, conf, **false**);

IOUtils.*closeStream*(in);

}

}

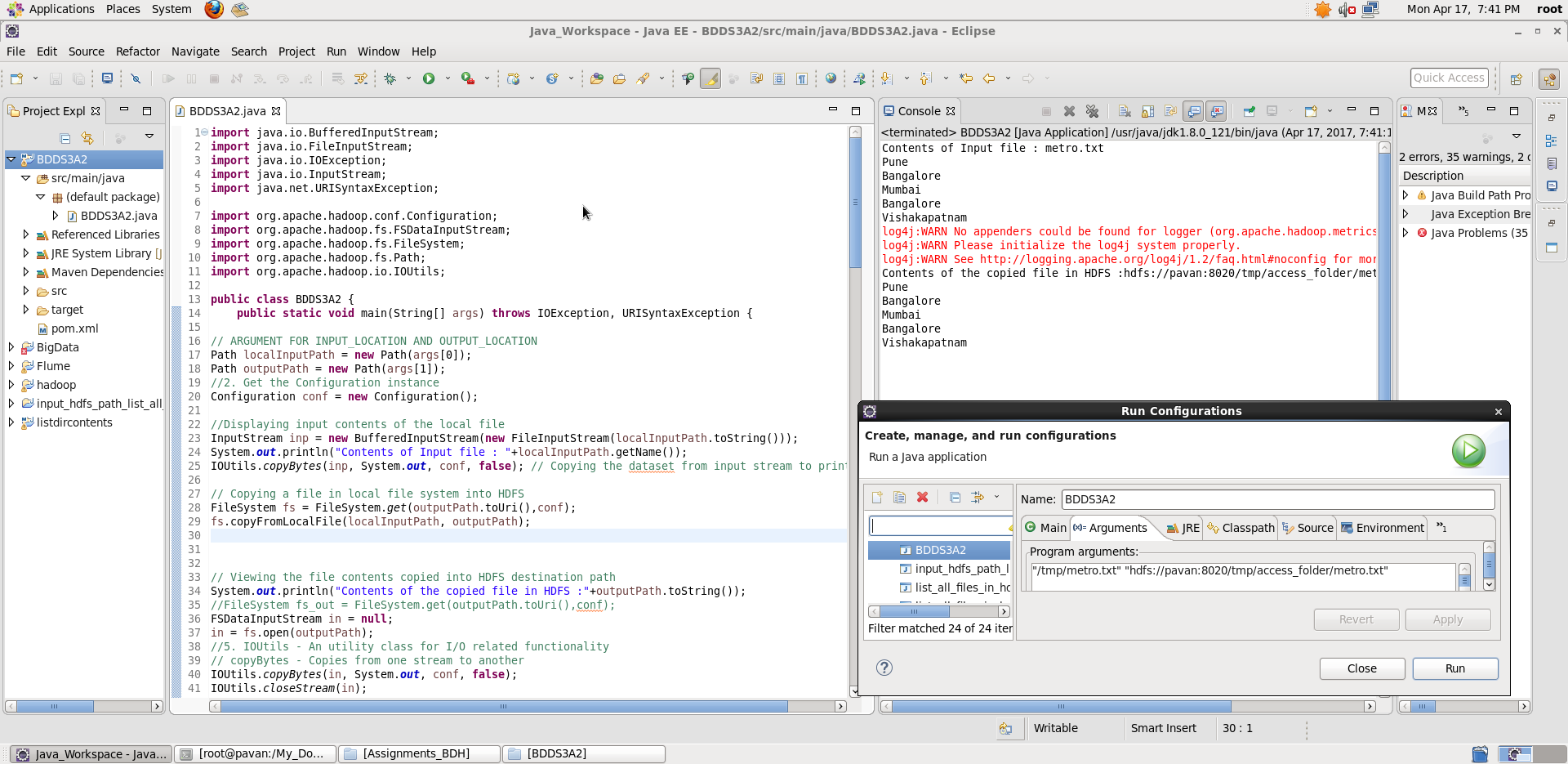
**Input format: “**local\_file\_path**” “**hdfs://<hostname>:<port\_no>/<HDFS\_File\_Path>

**Ex: “/tmp/metro.txt” “hdfs://pavan/tmp/access\_folder/metro.txt**

**here, hostname :** pavan , **Local\_File\_Path : /**tmp/metro.txt , **HDFS\_File\_Path :** tmp/access\_folder/metro.txt

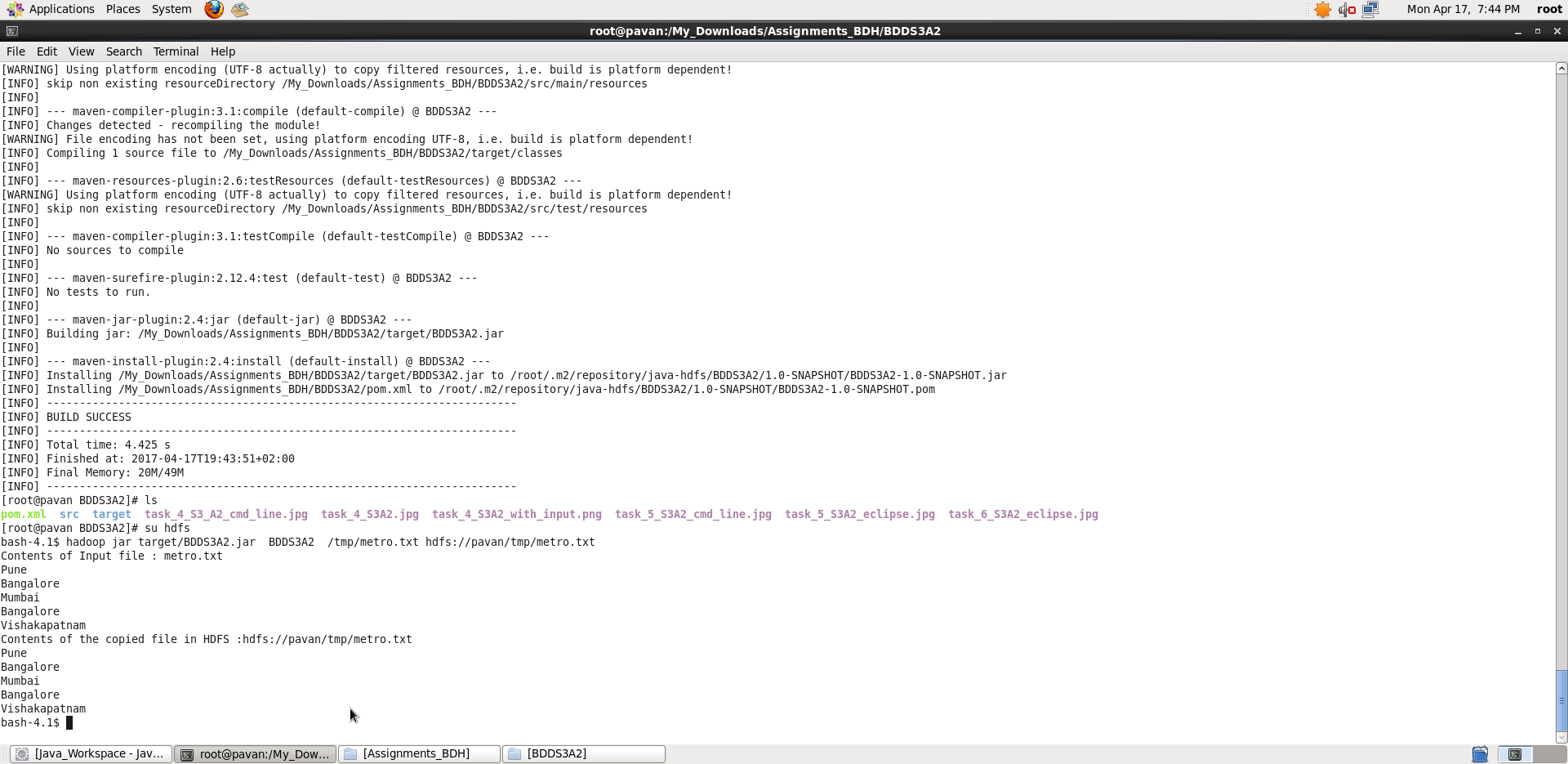
**Running in Eclipse:**

* Input given in Run Configurations Argument Tab as : **“/tmp/metro.txt” “hdfs://pavan/tmp/access\_folder/metro.txt**
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output in Console (Right side) - **displays initial content of local file , then copied content inside the HDFS file location, displays contents of the copied file in HDFS**

****

**Running in Command Line:**

* Make the folder structure BDDS3A2/src/main/java/BDDS3A2.java
* Then in BDDS3A2 folder, open terminal
* Create the Jar file target/BDDS3A2.jar using Maven # mvn clean install
* Run the Hadoop Job : “hadoop jar <jar\_file\_path> <class\_name> <input\_local\_file\_path> <input\_hdfs\_file\_path>
* In our case : “hadoop jar target/BDDS3A2.jar BDDS3A2 “/tmp/metro.txt” “hdfs://pavan/tmp/access\_folder/metro.txt”
* **Note: If port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml**
* Output on screen - displays initial content of local file , then copied content inside the HDFS file location

****

**Thus, we achieved the task of listing down all the files inside a directory and its sub-directories who have last modified timestamp between the start\_ts and end\_ts passed as argument, displayed the content of a file in HDFS on screen and copied a file from local filesystem to HDFS**

**Please Note: I didn’t mention port\_no while taking screenshots, internally if port\_no not specified, it takes the default port\_no : 8020 specified in core-site.xml.**