CPrE/SE 419: SOFTWARE TOOLS FOR LARGE-SCALE DATA ANALYSIS, SPRING 2022

Lab 1: Using the Cluster and Basic Intro to HDFS

1. Purpose

We have prepared a Virtual Machine (VM) instance as your lab environment. The first goal of this lab is to get familiar with this environment, called "Hadoop". The second goal is to get familiar with a distributed file system for storing large data sets, called "HDFS", which is short for "Hadoop Distributed File System". Specifically, at the end of this lab, you will know how to:

- Install VM instance on the local machine or connect to remote VM
- Write, compile, and execute a Hadoop program
- Use HDFS and the HDFS Programming Interface

2. Submission

Create a zip archive, "lab1_*netID*.zip", with the following and hand it in through canvas.

- A write-up answering questions for each experiment in the lab. For output, cut and paste results from your terminal and summarize when necessary.
- Commented Code for your program. Include all source files needed for compilation

3. Install Virtual Machine on the local machine (Preferred)

Please check "Handout-VM_LAB_MACHINE_ Local.pdf" at Canvas -> Modules -> LABS->Lab-Setups

4. Connect to Remote Virtual Machine

Please check "Handout-VM_LAB_MACHINE_ remote.pdf" at Canvas -> Modules -> LABS->Lab-Setups

5. HDFS Usage

HDFS is a distributed file system that is used by Hadoop to store large files. It automatically splits a file into many "blocks" of no more than 64MB and stores the blocks on separate machines. This helps the system to store huge files, much larger than any single machine can store. The HDFS also creates replicas for each block (by default 3). This helps to make the system more fault-tolerant. HDFS is an integral part of the Hadoop eco-system. HDFS is an Open Source implementation of the Google File System.

First setup VM instance following Section 3 or Section 4

To try out different HDFS utilities, try the commands:

$ hadoop fs -help

You should see something like the screenshot given below:

```
cpre419@cpre419-VirtualBox:~/hadoop/sbin$ hadoop fs -help
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst>]
        [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] [-e] <path> ...]
        [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] [-v] [-x] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
        [-head <file>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
```

You will notice that the HDFS utilities are very similar to that of the default UNIX file system utilities and hence should be easy to learn.

6. HDFS API

HDFS can also be accessed using a Java API, which allows us to read/write into the file system. For documentation, see: https://hadoop.apache.org/docs/r3.1.1/api/
Look at Package *org.apache.hadoop.fs*

The *FSDataOutputStream* and *FSDataInputStream* can be used to read/write files in HDFS, respectively. Check out their class documentation as well.

Read the source code of the program below, which would help you to better understand how the HDFS API works. Then, look at the different classes that were used and learn more about the various methods they provide. All the methods can be found in the Hadoop API documentation. Once you have browsed through the HDFS API, go onto the following experiment.

7. Experiment 1 (10 points):

First, we will manually copy a file into Hadoop file system. (1). Create a directory called "/lab1" under HDFS. (2). Create a new file called "afile.txt" with some text under "~/Downloads" folder in Ubuntu VM. (3). Finally, use *hadoop fs* command to move this file to the directory you created ("/lab1"). Screenshot the hadoop fs script as well as result and copy to your lab report

Next, we will write a program that create a file and write something to it. We use the Java programming language, as that is the default language used with Hadoop.

(Please check Section 9 for how to create a java program and compile it with Hadoop libraries)

The following program creates a new file and write something into it.

```java
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;

public class HDFSWrite  {
    public static void main ( String [] args ) throws Exception {
        // The system configuration
        // creates a configuration object to configure the Hadoop system
        Configuration conf = new Configuration();

        // Get an instance of the Filesystem using configuration
        FileSystem fs = FileSystem.get(conf);

        // The path to the file that needs to be written
        String path_name = "/lab1/newfile";

        Path path = new Path(path_name);

        // The Output Data Stream to write into
        FSDataOutputStream file = fs.create(path);

        // Write some data
        file.writeChars("The first Hadoop Program!");

        // Close the file and the file system instance
        file.close();
        fs.close();
    }
}
```

The above Java program uses the HDFS Application Programming Interface (API) provided by Hadoop to write into a HDFS file.

Paste the code into your favorite Java editor

Note that this is a HDFS path and will not be visible on the local filesystem.

We will then compile it and then run it to check the output.

(See the following Section 9 on how to compile and run program on hadoop)

8. Experiment 2 (40 points):

Write a program using the Java HDFS API that reads the contents of the HDFS file "*bigdata*" (access link: https://iastate.box.com/s/0lr732hj8g0uxcy7nnruaq6y5b29y7ue) and computes the 8-bit XOR checksum of all bytes whose offsets range from 1000000000 till 1000000999, both endpoints inclusive. Print out the 8-bit checksum.

Attach the Java code in your submission and the XOR output.

Hint: Use caret '^' to do the XOR.

Note: To download the bigdata file, I would recommend you find the web browser in Ubuntu VM and download the data file through it. If you download the data outside the Ubuntu VM and drag into it, the transfer will be very slow.

9. Using Eclipse IDE with Hadoop

To use Maven:

1. **Open Eclipse IDE**
   (which has been installed in Ubuntu VM, "/home/eclipse/java-2019-12/eclipse")
2. **Create new Maven project**
   ("File" -> "New" -> "Other", and then find "Maven Project" under "Maven")
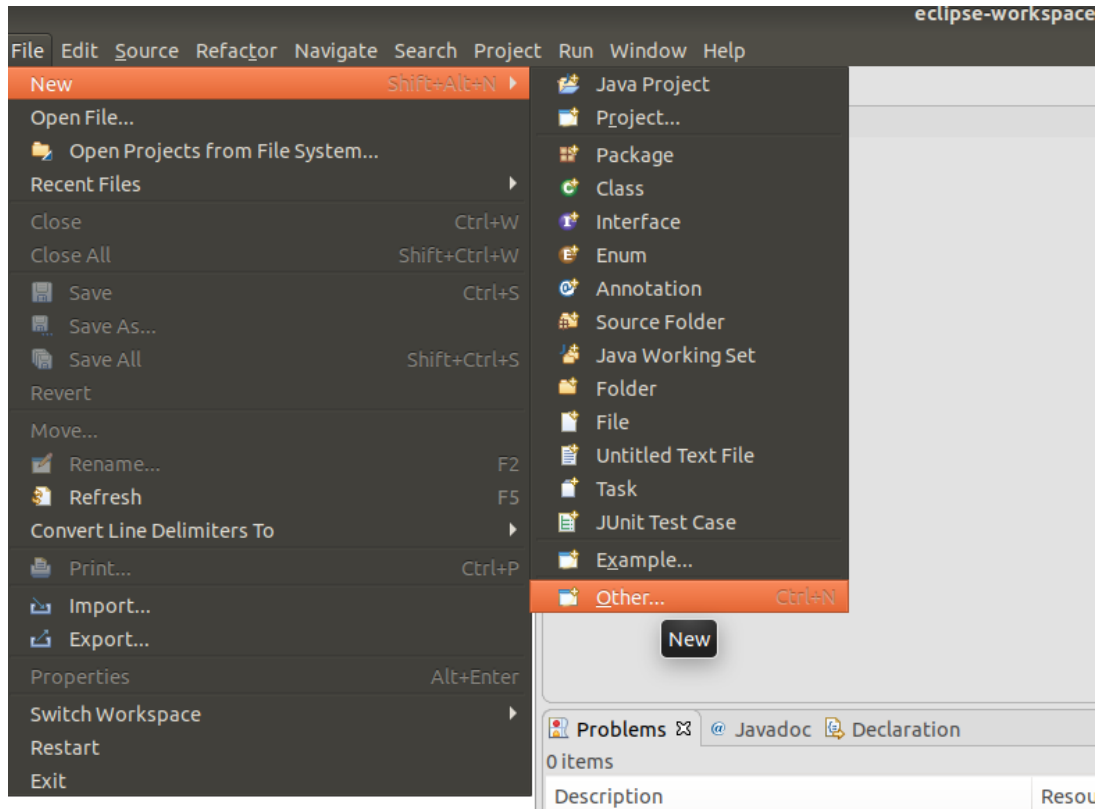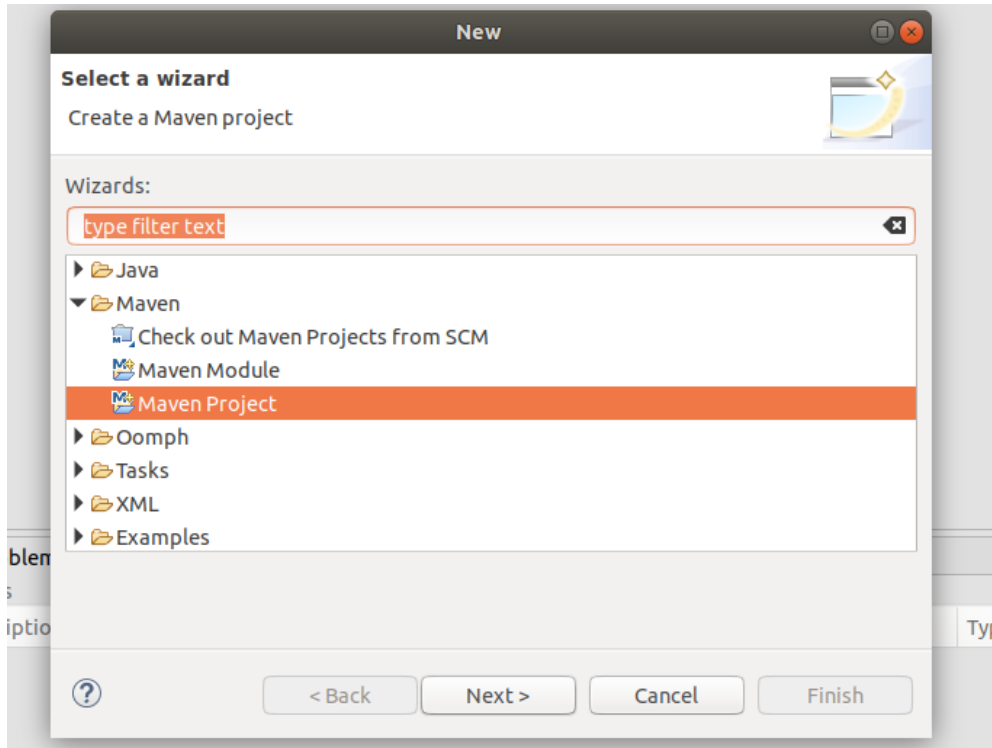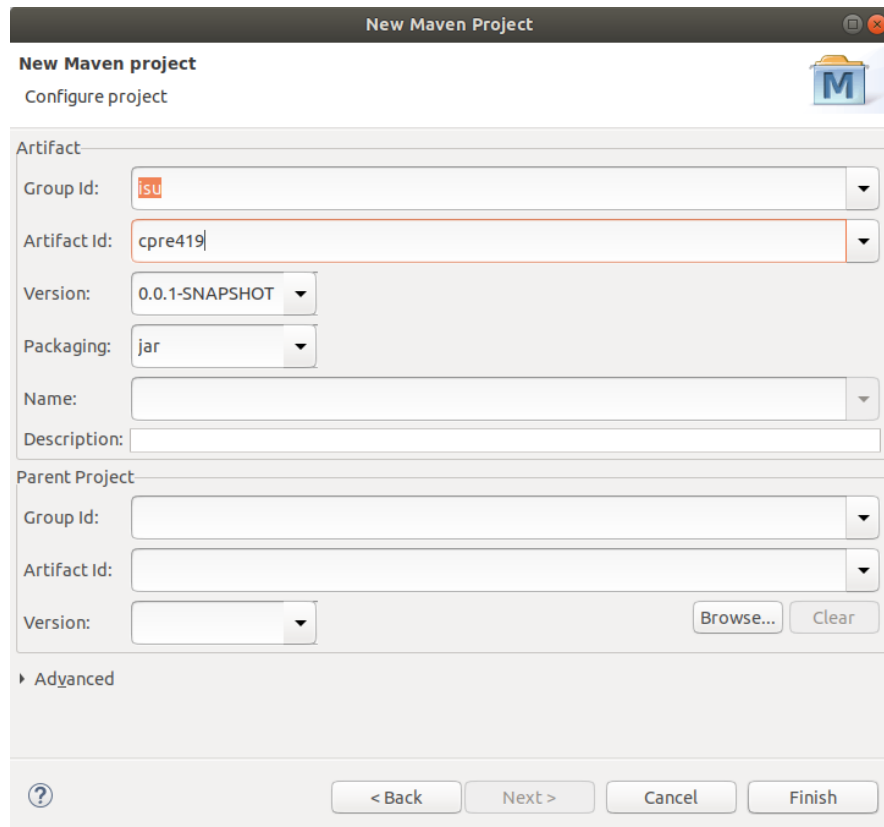
Check both "Create a simple project (…)" and "Use default Workspace location"
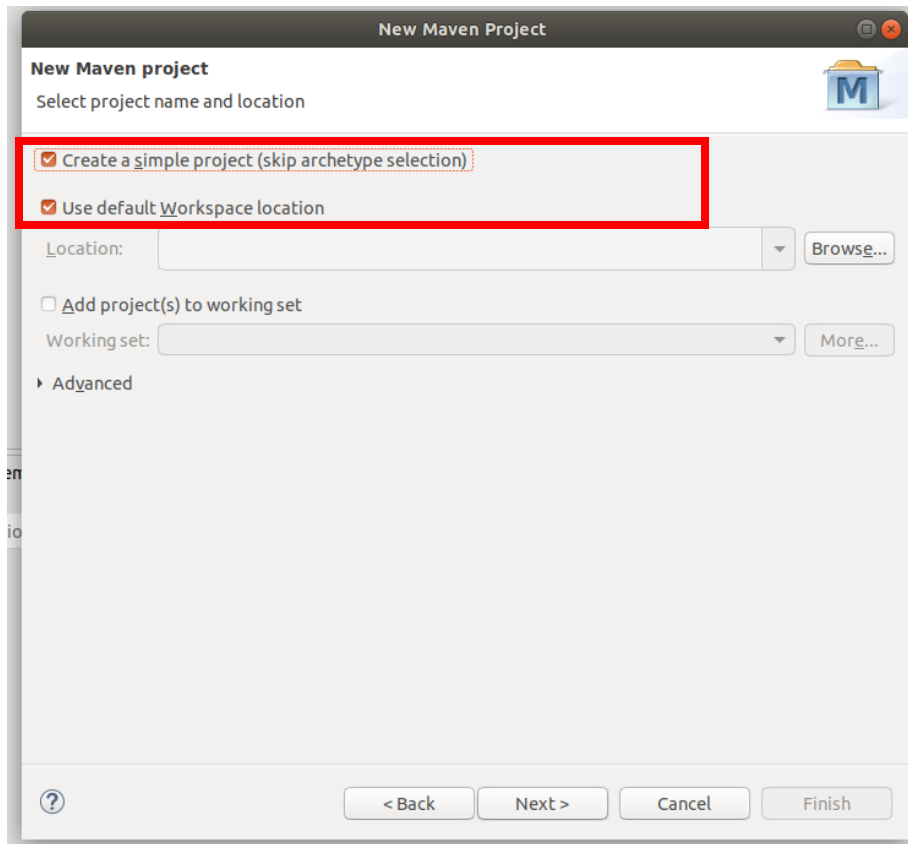
Give it a Group Id and Artifact Id. Then Finish

3. Expand the project in the left Package Explorer. Right click on "JRE System Library [J2SE-1.5]"
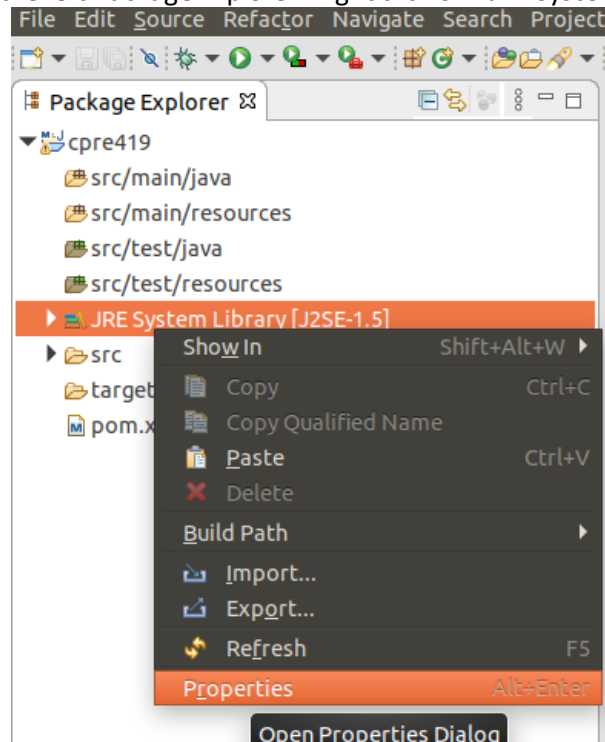
Switch it from Execution Environment 1.5 to "Workspace default JRE" which by fault should be java-8
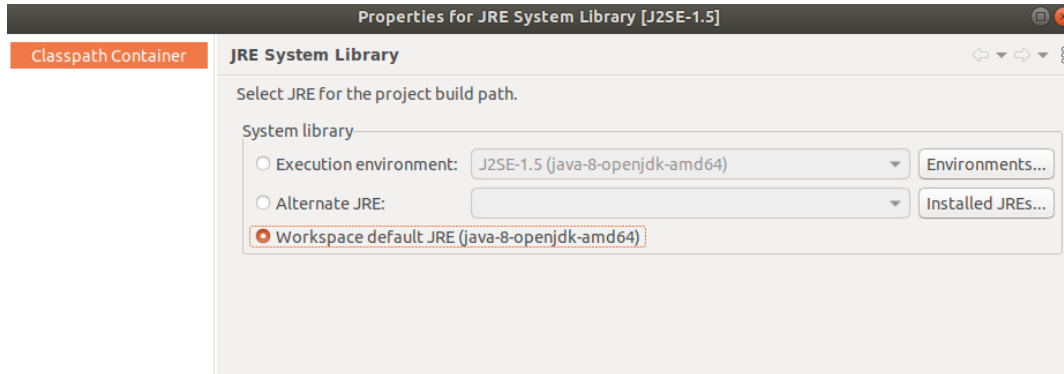
4. **Add Hadoop library to Maven pom.xml file**
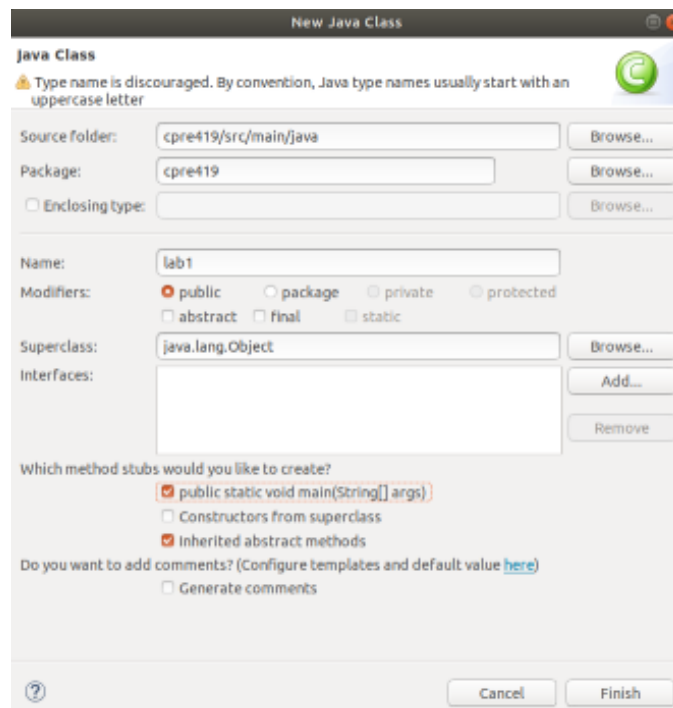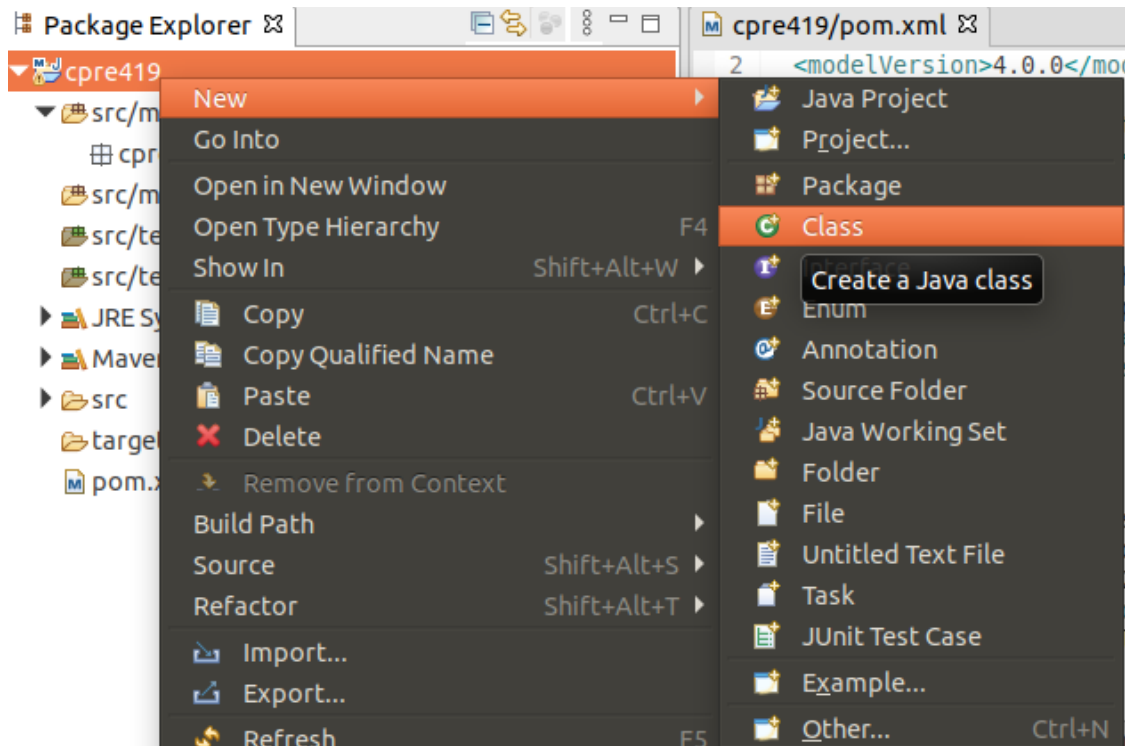   **Add the following in your pom.xml file:**

```
…
     <dependencies>
       <dependency>
          <groupId>org.apache.hadoop</groupId>
          <artifactId>hadoop-common</artifactId>
          <version>3.1.3</version>
          <scope>provided</scope>
       </dependency>


       <dependency>
          <groupId>jdk.tools</groupId>
          <artifactId>jdk.tools</artifactId>
          <version>1.8.0_242</version>
          <scope>system</scope>
          <systemPath>/usr/lib/jvm/java-8-openjdk-
amd64/lib/tools.jar</systemPath>
       </dependency>
     </dependencies>
```

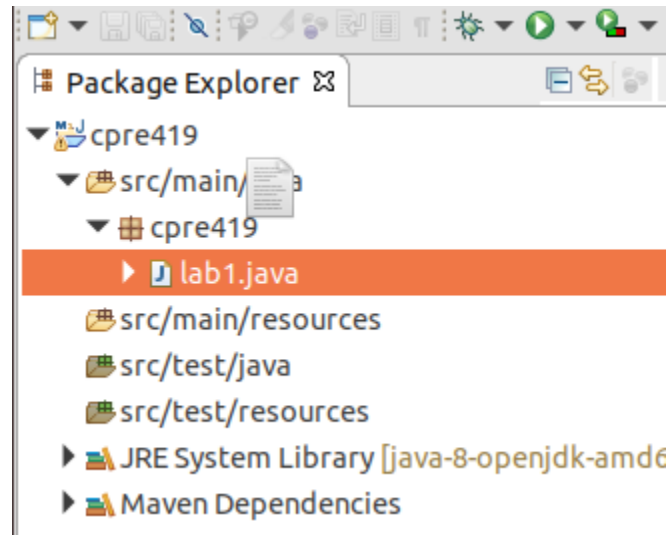**You want to have the following lines in your pom.xml file as well. They make sure that you are using java 1.8**

```
     <properties>
          <maven.compiler.source>1.8</maven.compiler.source>
          <maven.compiler.target>1.8</maven.compiler.target>
     </properties>
```

As the result, you can check "pom_hadoop.xml" under "Downloads" in Ubuntu VM

To write a Java program

Then you will have a new .java class under "src/main/java" -> "cpre419". To simplify the compiling process, move it from "cpre419" subclass to directly under "src/main/java"
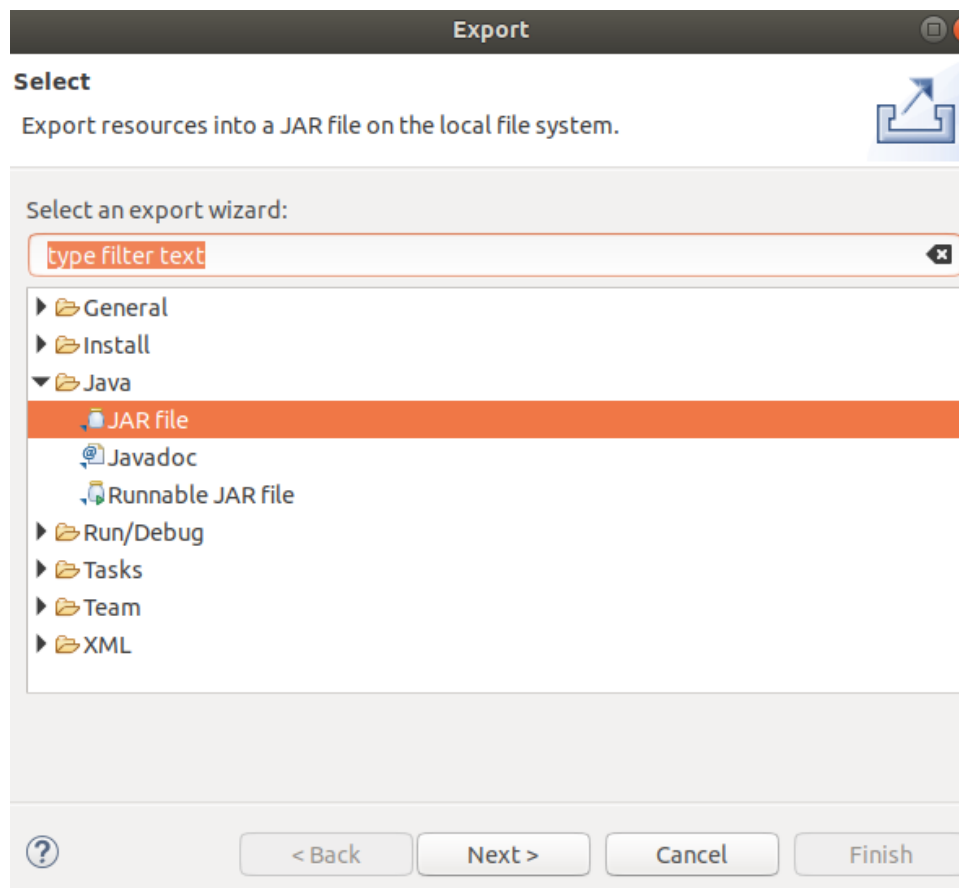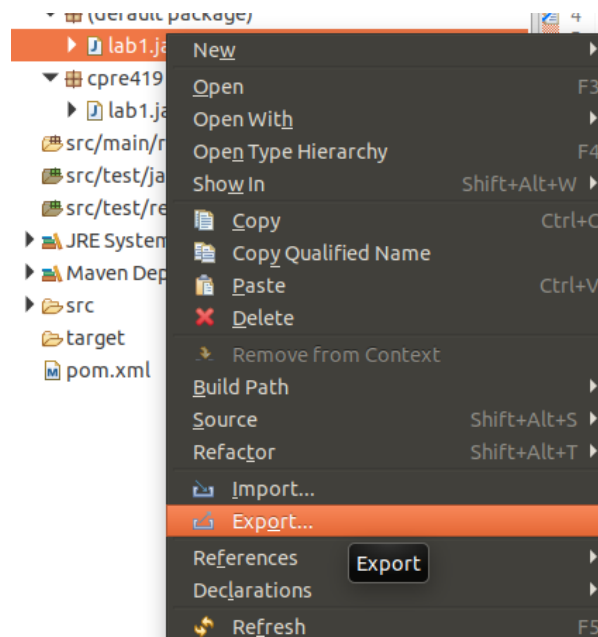
Eventually, you should be able to find a new folder "(default package)" automatically generated and your java class is copied under it. Open that java file and do your coding part
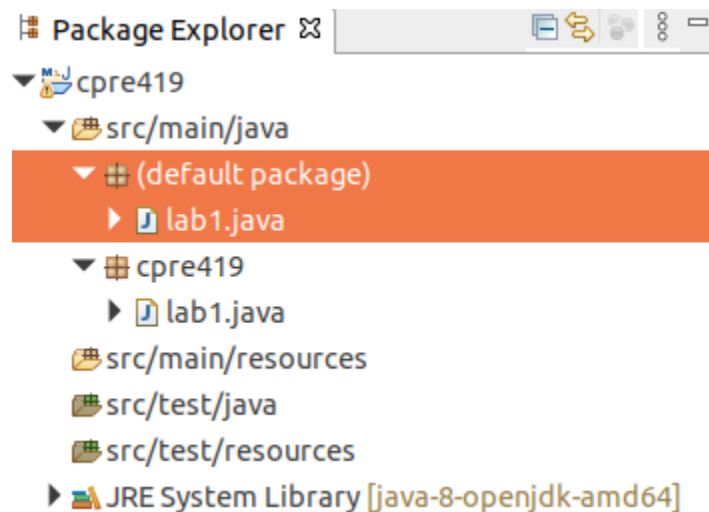

10. Exporting Jar File

1. Once your code has been completed, right click on the java class and select Export. Choose "JAR file" under "Java"

2. Check the three boxes on the right (.classpath, .project, pom.xml). You will also need to choose a location for the generated jar file to be saved. Then click the finish button to generate the jar file. You may receive a warning about compilation warnings, you may ignore these.
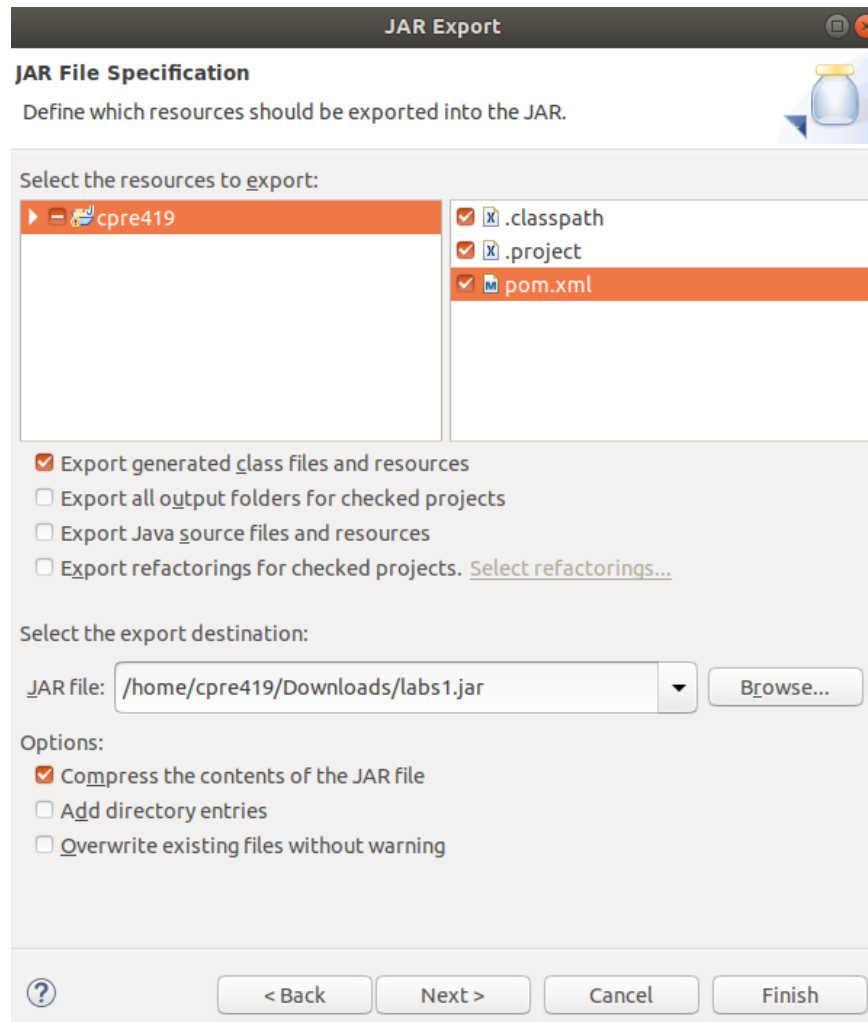
3. Run your jar file by using

hadoop jar <location_of_jar_file> <name_of_java_class>

For instance, in the above example, I should use "hadoop jar ~/Downloads/labs1.jar HDFSWrite" to run my exported jar file