MapReduce - Phân cụm Kmeans

May 3, 2022

Lập trình MapReduce cho bài toán phân cụm Kmeans

Bước 1: Tạo file chứa dữ liệu

Tạo file **data-kmeans.txt** với nội dung như sau:

```
25,79
34,51
22,53
27,78
33,59
33,74
31,73
22,57
35,69
34,75
67,51
54,32
57,40
43,47
50,53
57,36
59,35
52,58
65,59
47,50
49,25
48,20
35,14
33,12
```

```
44,20
45,5
38,29
43,27
51,8
46,7
```

Bước 2: Tạo thư mục đầu vào trong hdfs

```
hdfs dfs -mkdir /k-input
```

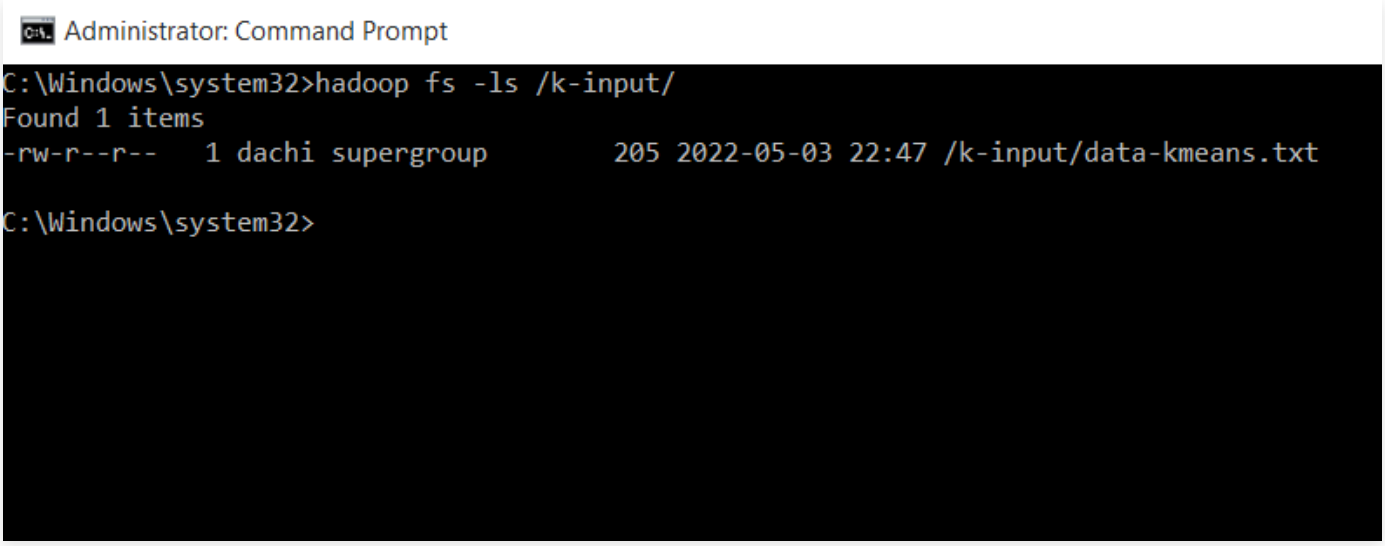
Bước 3: Đẩy file **data-kmeans.txt** vào folder **k-input** vừa tạo:

```
hadoop fs -put C:\kmeans\data-kmeans.txt /k-input
```

> *NOTE: Thay **C:\kmeans\data-kmeans.txt** bằng đường dẫn thư mục lưu file*

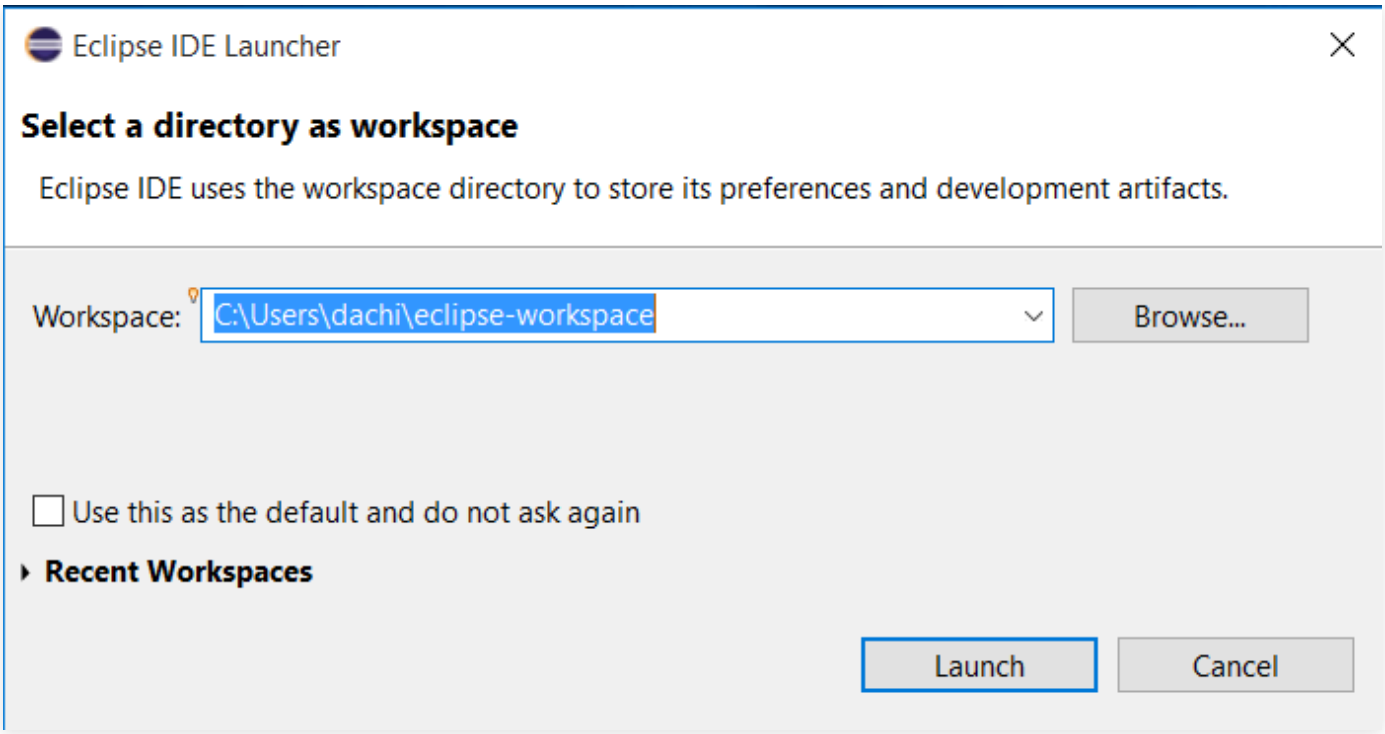
Kiểm tra xem file trong thư mục **k-input**

```
hadoop fs -ls /k-input/
```

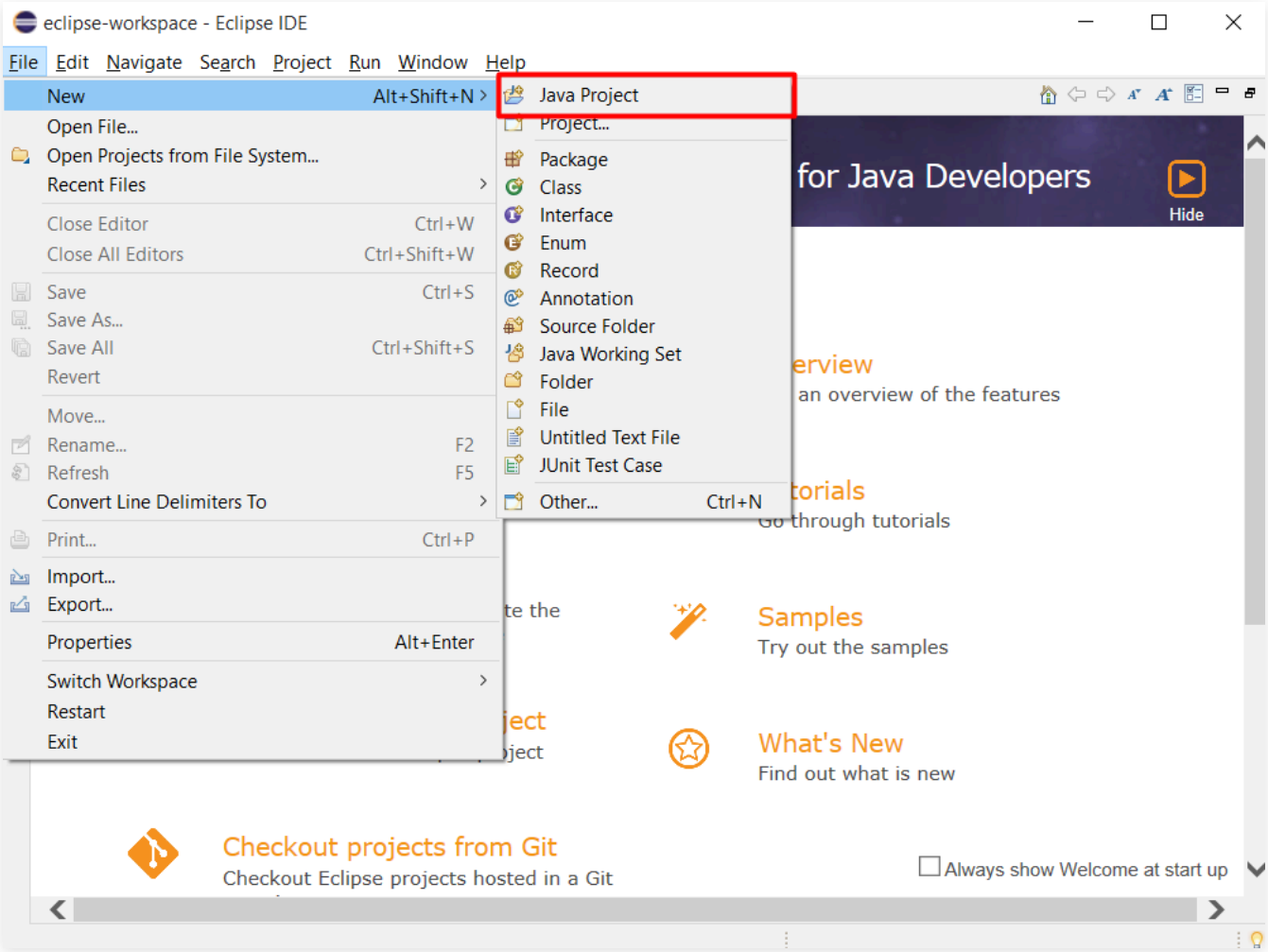


Bước 4: Tạo project

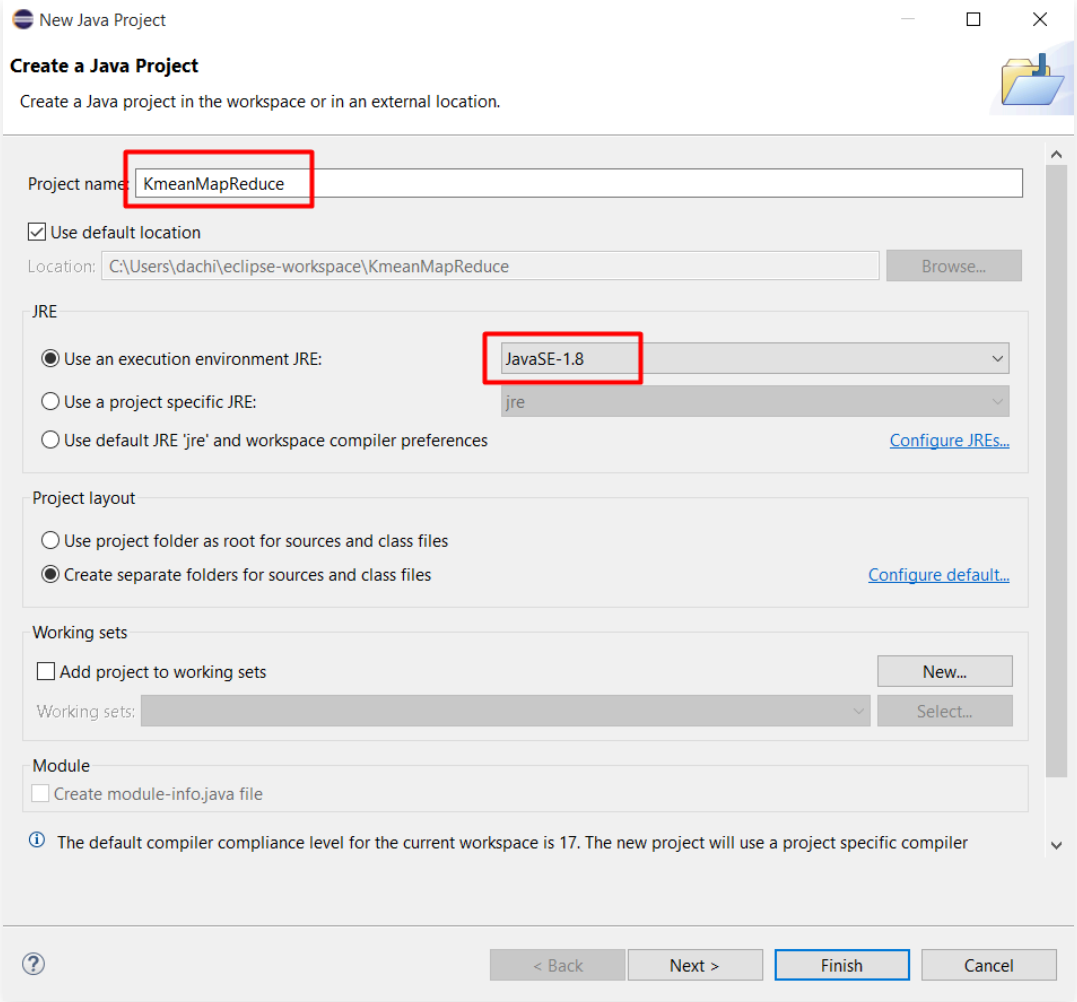
Mở chương trình Eclipse. Chọn **workspace** (nên để mặc định)



Tạo project Java, chọn **File > New > Java Project**

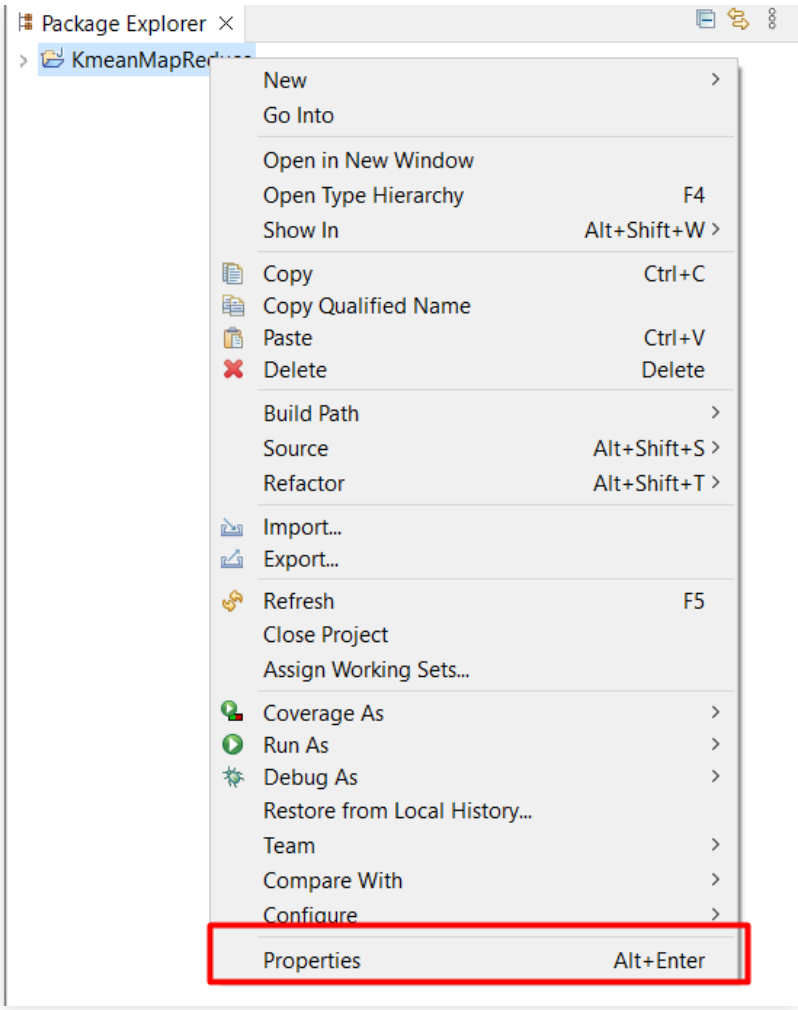


Đặt tên project là **KmeanMapReduce** và chọn môi trường là **JavaSE-1.8**. Xong ấn **Finish**.

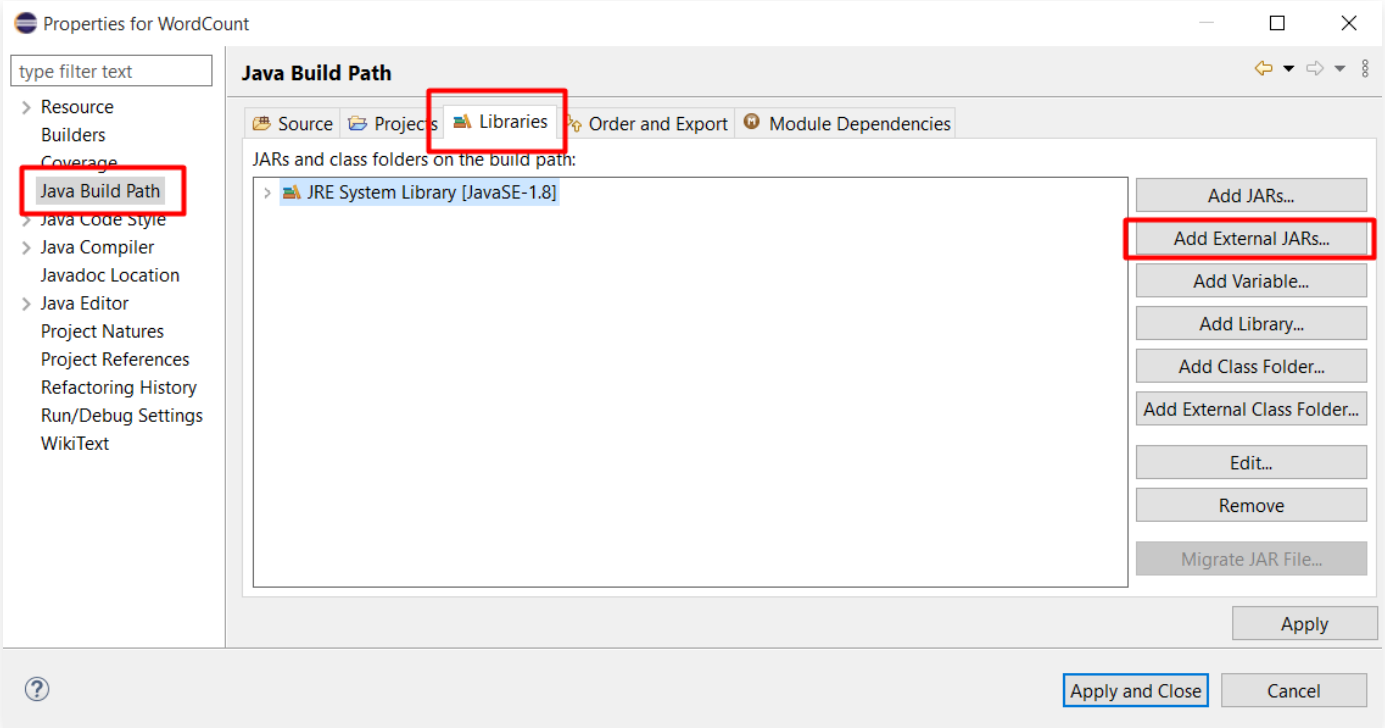


Bước 5: Thêm thư viện cần thiết để chạy MapReduce

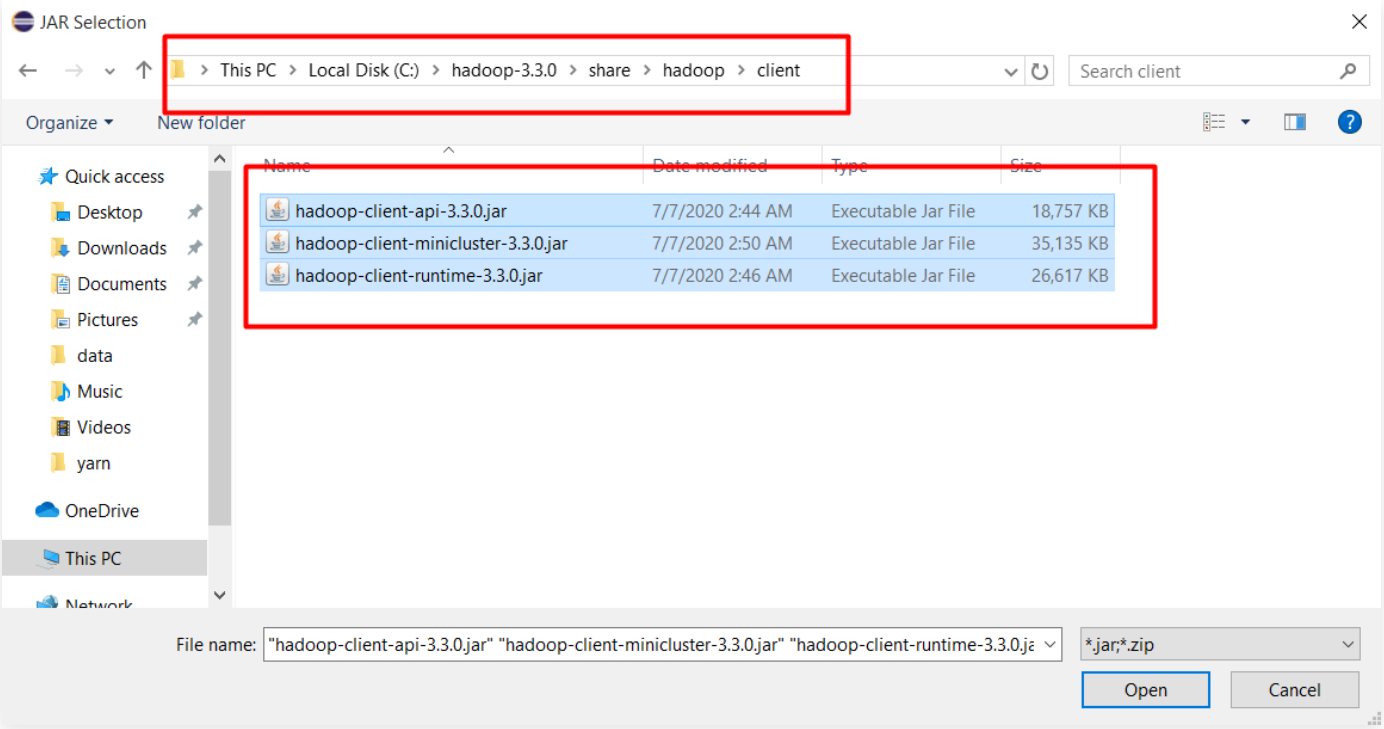
Chuột phải vào project **KmeanMapReduce** chọn **Properties**



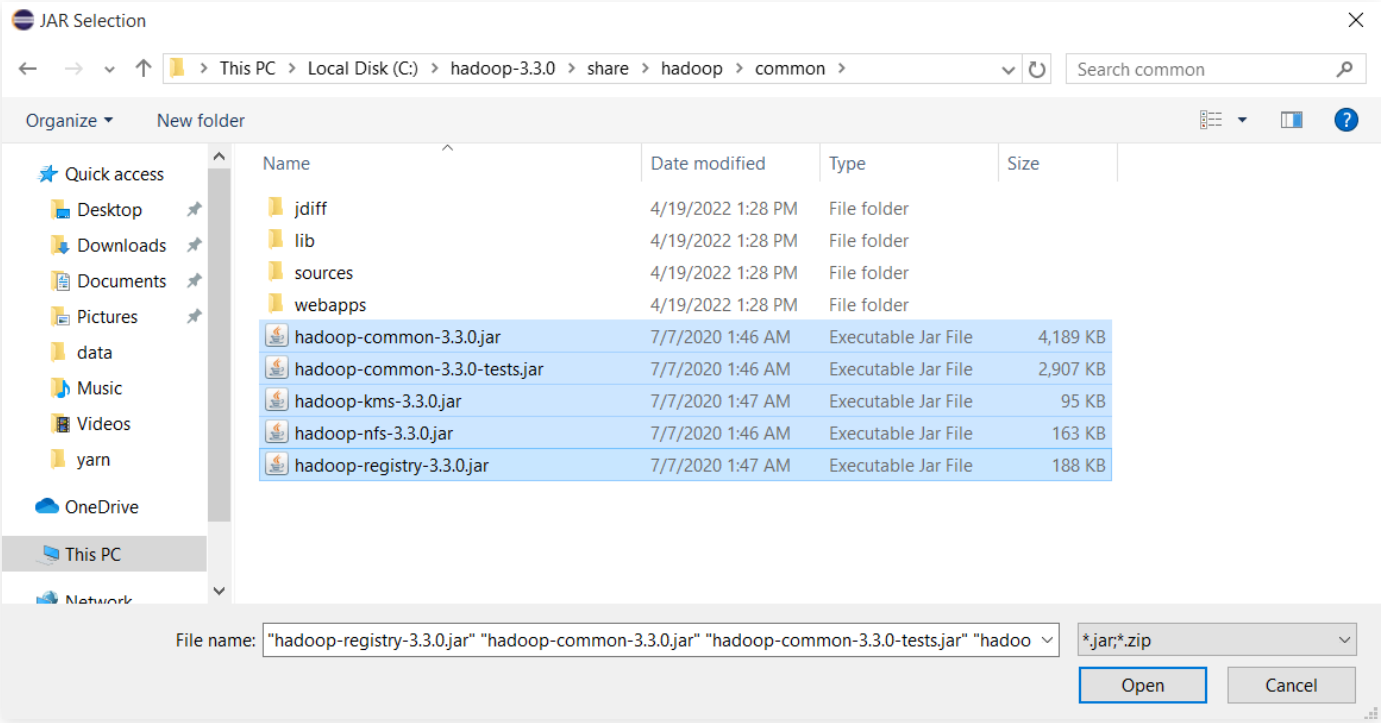
Chọn **Java Build Path**, chọn tab **Libraries** và bấm **Add External JARs**



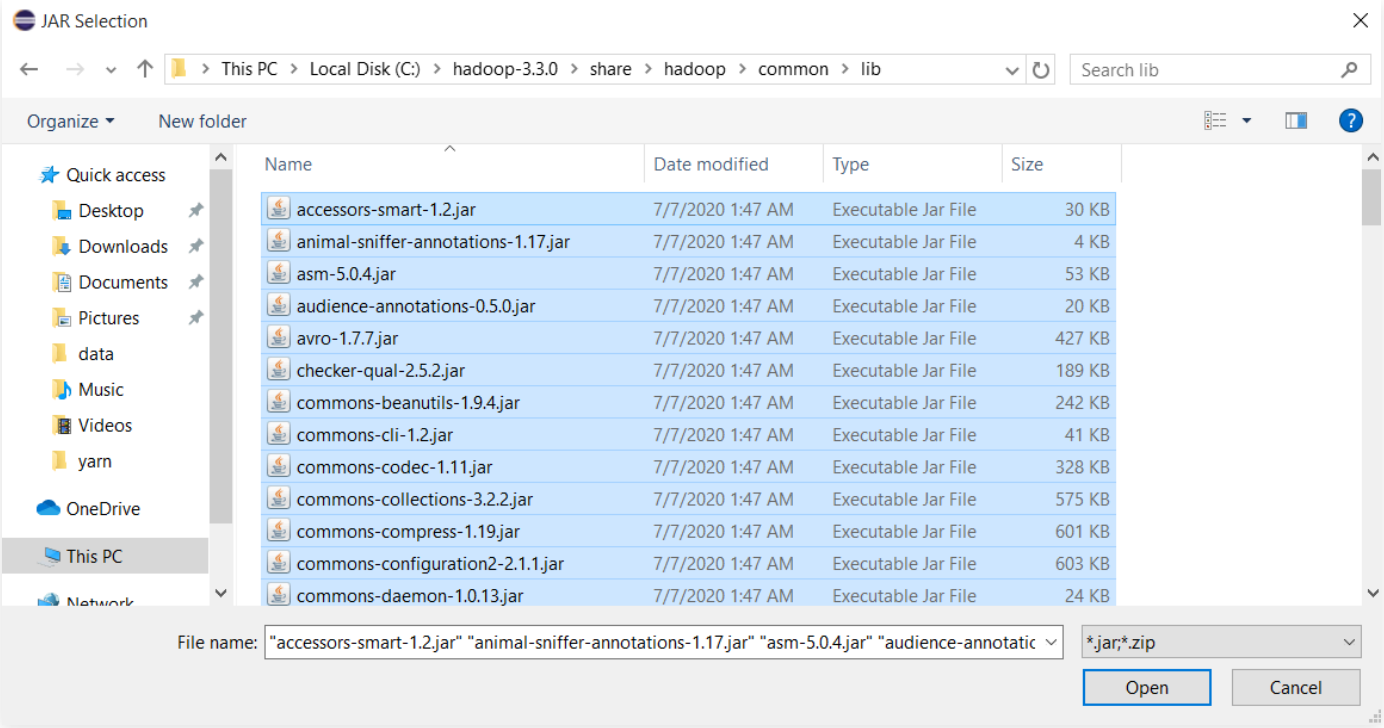
Chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\client** và ấn **Open**



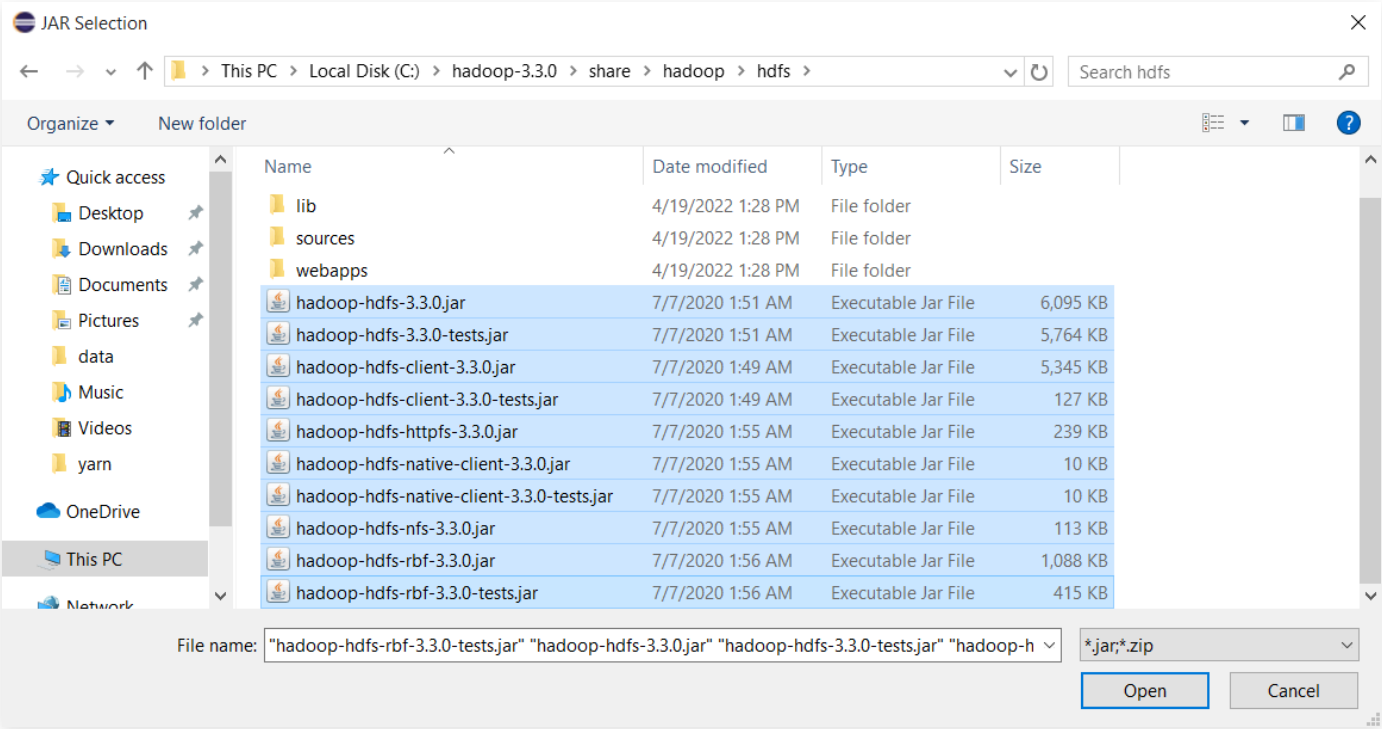
Tương tự chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\common** và ấn **Open**



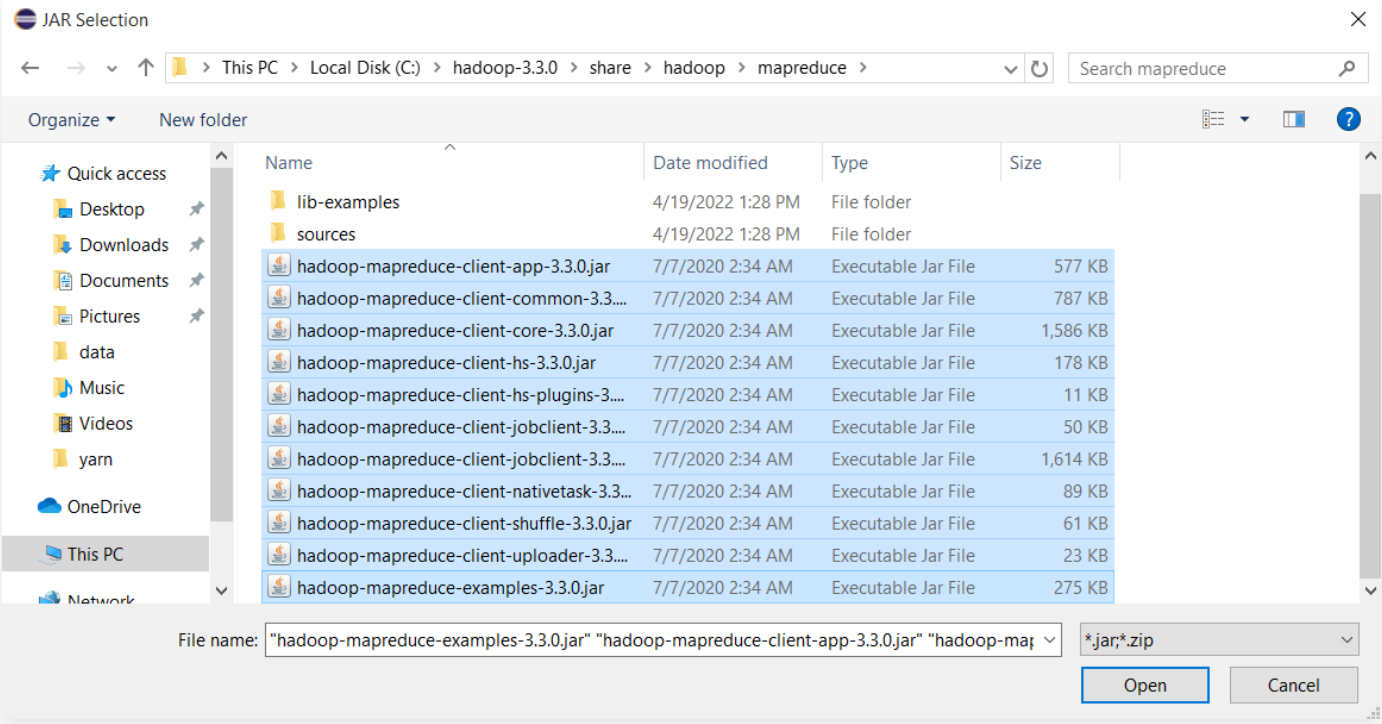
Chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\common\lib** và ấn **Open**



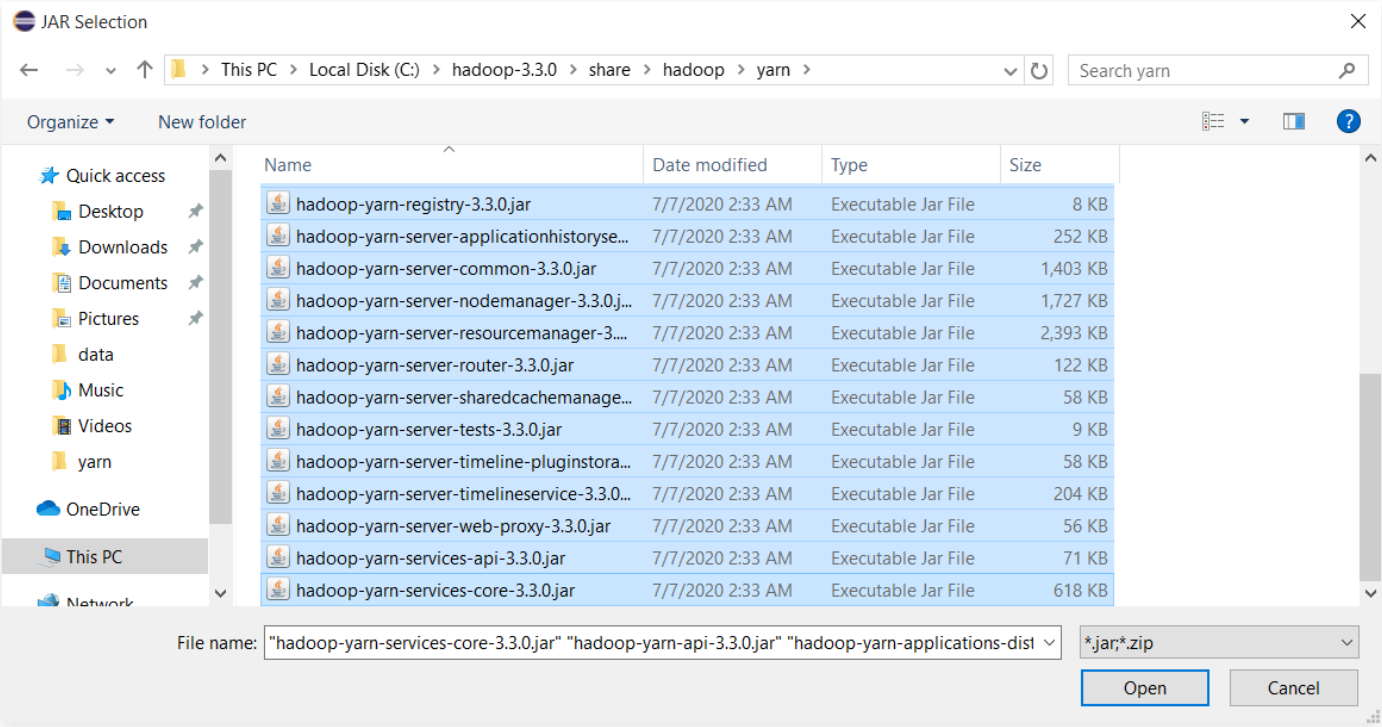
Chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\hdfs** và ấn **Open**



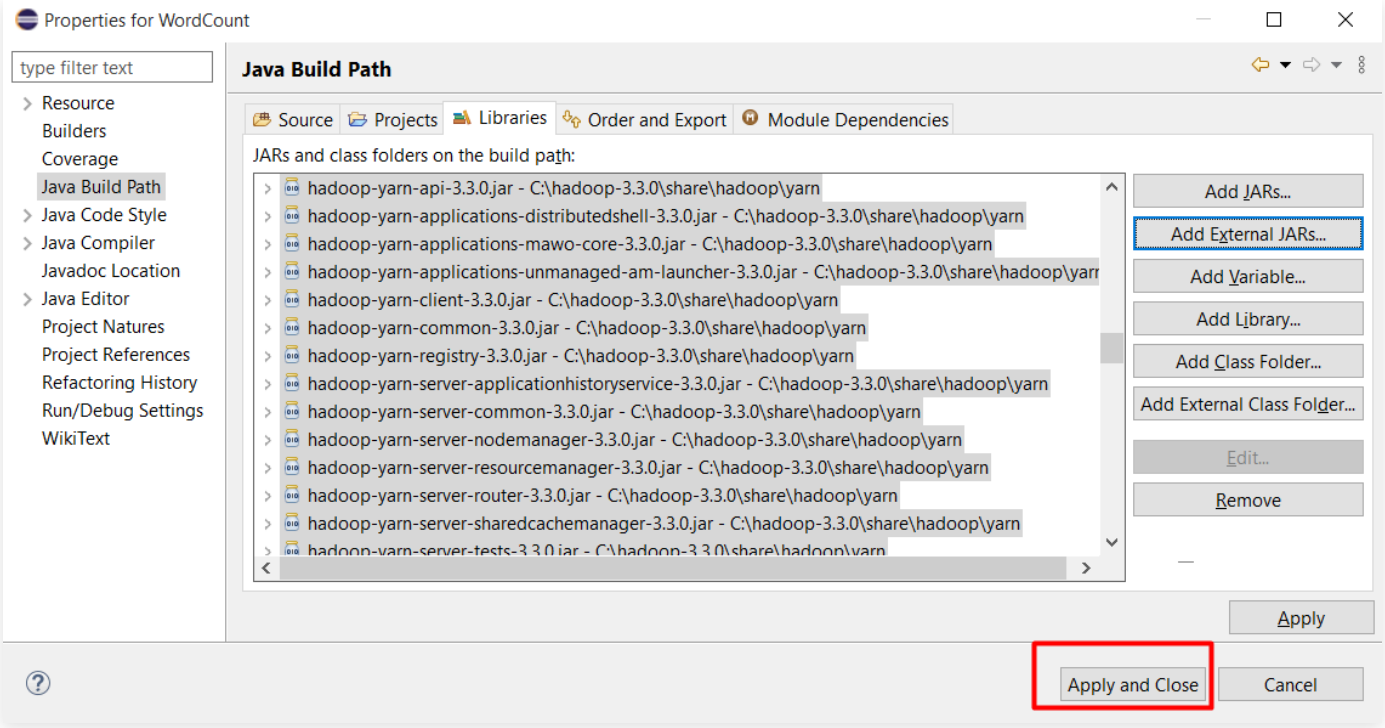
Chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\mapreduce** và ấn **Open**



Chọn tất cả file trong thư mục **C:\hadoop-3.3.0\share\hadoop\yarn** và ấn **Open**

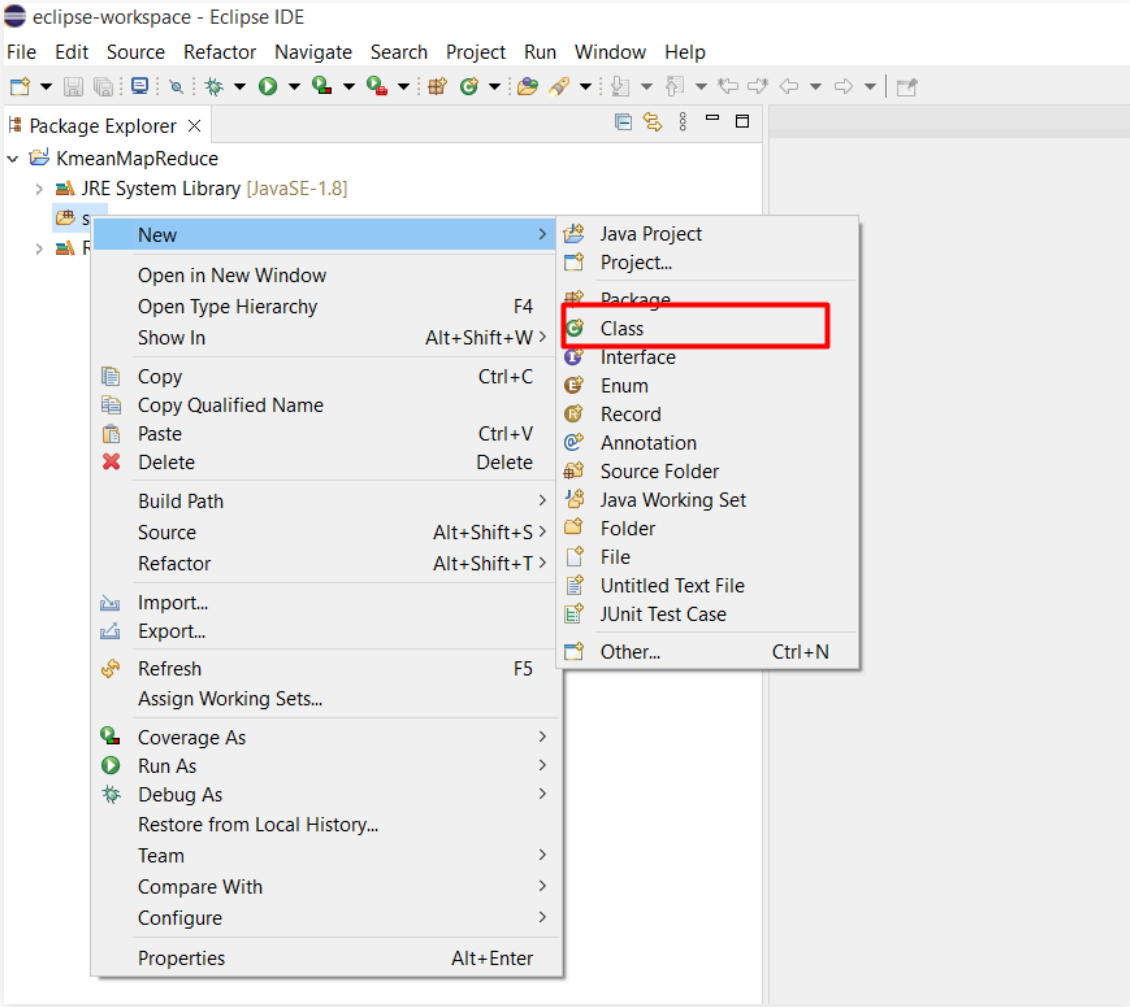


Ấn **Apply and Close**

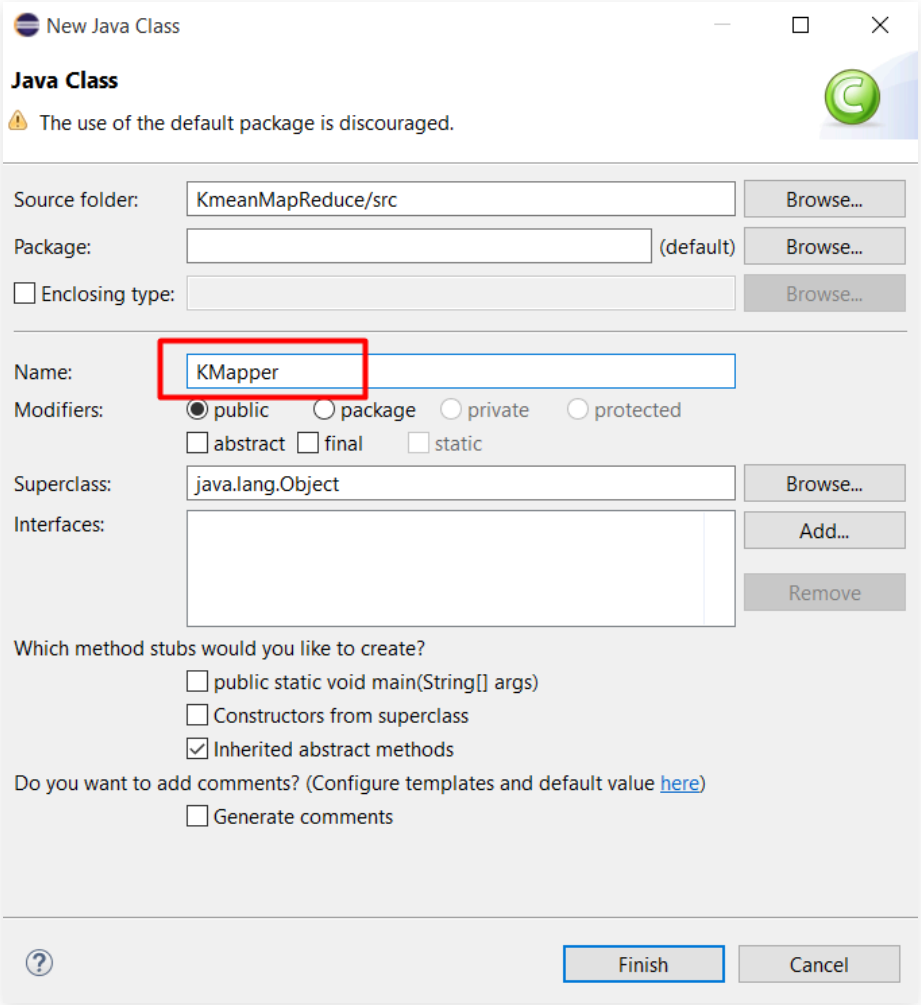


Bước 6: Tạo các class xử lý nhiệm vụ phân cụm K-means

Double click vào project **KmeanMapReduce**, chuột phải vào **src** và chọn **New > Class**



Tạo class để xử lý nhiệm vụ **Map**, đặt tên là **KMapper**



Nội dung bên trong file **KMapper.java**:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class KMapper extends Mapper<LongWritable, Text, LongWritable, PointWritable> {

        private PointWritable[] currCentroids;
        private final LongWritable centroidId = new LongWritable();
        private final PointWritable pointInput = new PointWritable();

        @Override
        public void setup(Context context) {
                int nClusters = Integer.parseInt(context.getConfiguration().get("k"));

                this.currCentroids = new PointWritable[nClusters];
```

```
                for (int i = 0; i < nClusters; i++) {
                        String[] centroid = context.getConfiguration().getStrings("C" +
                        // this.currCentroids[i] = new PointWritable(centroid[0].split(
                        this.currCentroids[i] = new PointWritable(centroid);
                }
        }

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOExcep

                String[] arrPropPoint = value.toString().split(",");
                pointInput.set(arrPropPoint);
                double minDistance = Double.MAX_VALUE;
                int centroidIdNearest = 0;
                for (int i = 0; i < currCentroids.length; i++) {
                        System.out.println("currCentroids[" + i + "]=" + currCentroids[
                        double distance = pointInput.calcDistance(currCentroids[i]);
                        if (distance < minDistance) {
                                centroidIdNearest = i;
                                minDistance = distance;
                        }
                }
                centroidId.set(centroidIdNearest);
                context.write(centroidId, pointInput);
        }
}
```

Tương tự tạo class xử lý nhiệm vụ **Combiner**, đặt tên là **KCombiner**:

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class KCombiner extends Reducer<LongWritable, PointWritable, LongWritable, Point

        public void reduce(LongWritable centroidId, Iterable<PointWritable> points, Con
                        throws IOException, InterruptedException {

                PointWritable ptSum = PointWritable.copy(points.iterator().next());
                while (points.iterator().hasNext()) {
                        ptSum.sum(points.iterator().next());
                }
```

```
                context.write(centroidId, ptSum);
        }
}
```

Tạo class xử lý nhiệm vụ **Reducer**, đặt tên là **KReducer**:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class KReducer extends Reducer<LongWritable, PointWritable, Text, Text> {

        private final Text newCentroidId = new Text();
        private final Text newCentroidValue = new Text();

        public void reduce(LongWritable centroidId, Iterable<PointWritable> partialSums,
                        throws IOException, InterruptedException {

                PointWritable ptFinalSum = PointWritable.copy(partialSums.iterator().nex
                while (partialSums.iterator().hasNext()) {
                        ptFinalSum.sum(partialSums.iterator().next());
                }

                ptFinalSum.calcAverage();

                newCentroidId.set(centroidId.toString());
                newCentroidValue.set(ptFinalSum.toString());
                context.write(newCentroidId, newCentroidValue);
        }
}
```

Tạo class **PointWritable**:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class PointWritable implements Writable {
```

```java
private float[] attributes = null;
private int dim;
private int nPoints;

public PointWritable() {
        this.dim = 0;
}

public PointWritable(final float[] c) {
        this.set(c);
}

public PointWritable(final String[] s) {
        this.set(s);
}

public static PointWritable copy(final PointWritable p) {
        PointWritable ret = new PointWritable(p.attributes);
        ret.nPoints = p.nPoints;
        return ret;
}

public void set(final float[] c) {
        this.attributes = c;
        this.dim = c.length;
        this.nPoints = 1;
}

public void set(final String[] s) {
        this.attributes = new float[s.length];
        this.dim = s.length;
        this.nPoints = 1;
        for (int i = 0; i < s.length; i++) {
                this.attributes[i] = Float.parseFloat(s[i]);
        }
}

@Override
public void readFields(final DataInput in) throws IOException {
        this.dim = in.readInt();
        this.nPoints = in.readInt();
        this.attributes = new float[this.dim];

        for (int i = 0; i < this.dim; i++) {
                this.attributes[i] = in.readFloat();
        }
```

```
    }

    @Override
    public void write(final DataOutput out) throws IOException {
            out.writeInt(this.dim);
            out.writeInt(this.nPoints);

            for (int i = 0; i < this.dim; i++) {
                    out.writeFloat(this.attributes[i]);
            }
    }

    @Override
    public String toString() {
            StringBuilder point = new StringBuilder();
            for (int i = 0; i < this.dim; i++) {
                    point.append(Float.toString(this.attributes[i]));
                    if (i != dim - 1) {
                            point.append(",");
                    }
            }
            return point.toString();
    }

    public void sum(PointWritable p) {
            for (int i = 0; i < this.dim; i++) {
                    this.attributes[i] += p.attributes[i];
            }
            this.nPoints += p.nPoints;
    }

    public double calcDistance(PointWritable p) {

            double dist = 0.0f;
            for (int i = 0; i < this.dim; i++) {
                    dist += Math.pow(Math.abs(this.attributes[i] - p.attributes[i])
            }
            dist = Math.sqrt(dist);
            return dist;

    }

    public void calcAverage() {
            for (int i = 0; i < this.dim; i++) {
                    float temp = this.attributes[i] / this.nPoints;
                    this.attributes[i] = (float) Math.round(temp * 100000) / 100000
```

```
                }
                this.nPoints = 1;
        }
}
```

Và tạo class **Main** chứa hàm **main** để khởi chạy chương trình:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Random;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Main extends Configured implements Tool {

        public static PointWritable[] initRandomCentroids(int kClusters, int nLineOfInpu
                        Configuration conf) throws IOException {
                System.out.println("Initializing random " + kClusters + " centroids...")
                PointWritable[] points = new PointWritable[kClusters];

                List<Integer> lstLinePos = new ArrayList<Integer>();
```

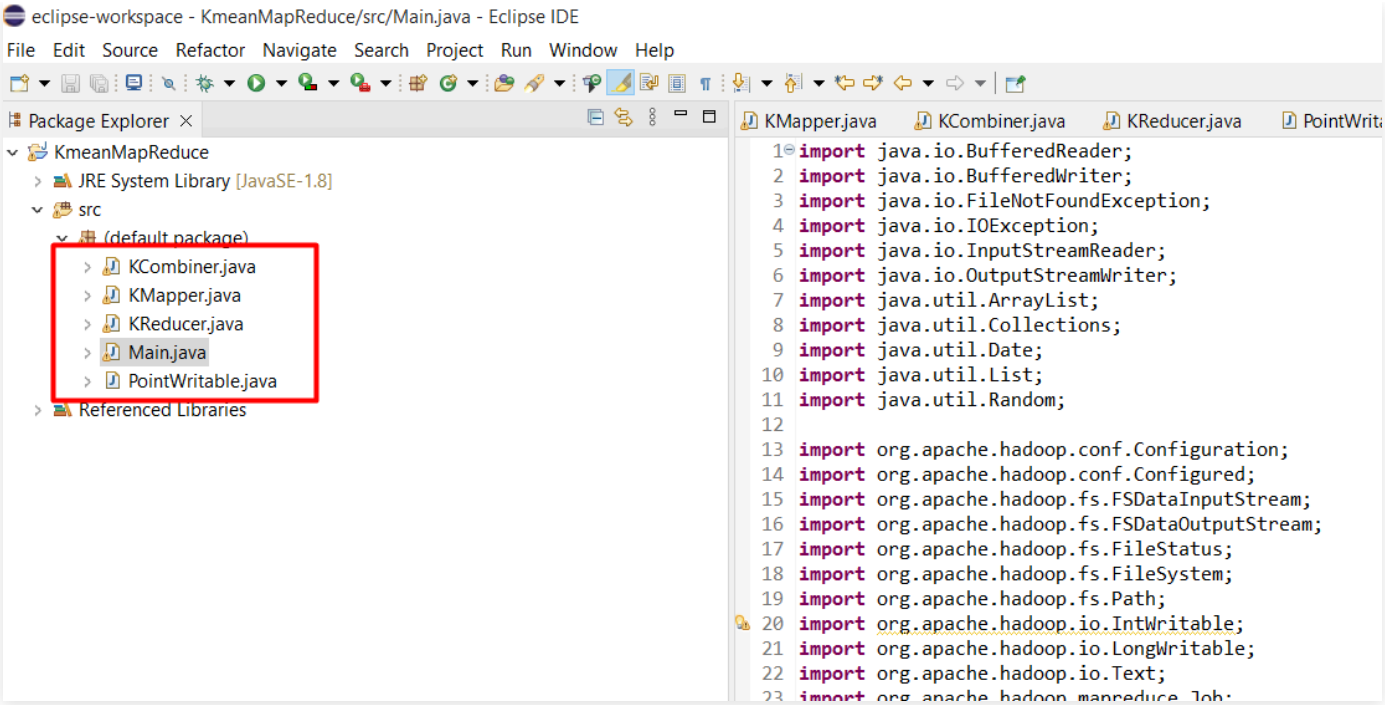
```
        Random random = new Random();
        int pos;
        while (lstLinePos.size() < kClusters) {
                pos = random.nextInt(nLineOfInputFile);
                if (!lstLinePos.contains(pos)) {
                        lstLinePos.add(pos);
                }
        }
        Collections.sort(lstLinePos);

        FileSystem hdfs = FileSystem.get(conf);
        FSDataInputStream in = hdfs.open(new Path(inputFilePath));

        BufferedReader br = new BufferedReader(new InputStreamReader(in));

        int row = 0;
        int i = 0;
        while (i < lstLinePos.size()) {
                pos = lstLinePos.get(i);
                String point = br.readLine();
                if (row == pos) {
                        points[i] = new PointWritable(point.split(","));
                        i++;
                }
                row++;
        }
        br.close();
        return points;

}

public static void saveCentroidsForShared(Configuration conf, PointWritable[] p
        for (int i = 0; i < points.length; i++) {
                String centroidName = "C" + i;
                conf.unset(centroidName);
                conf.set(centroidName, points[i].toString());
        }
}

public static PointWritable[] readCentroidsFromReducerOutput(Configuration conf,
                String folderOutputPath) throws IOException, FileNotFoundExcept
        PointWritable[] points = new PointWritable[kClusters];
        FileSystem hdfs = FileSystem.get(conf);
        FileStatus[] status = hdfs.listStatus(new Path(folderOutputPath));

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

```
                if (!status[i].getPath().toString().endsWith("_SUCCESS")) {
                        Path outFilePath = status[i].getPath();
                        System.out.println("read " + outFilePath.toString());
                        BufferedReader br = new BufferedReader(new InputStreamR
                        String line = null;// br.readLine();
                        while ((line = br.readLine()) != null) {
                                System.out.println(line);

                                String[] strCentroidInfo = line.split("\t"); //
                                int centroidId = Integer.parseInt(strCentroidIn
                                String[] attrPoint = strCentroidInfo[1].split("
                                points[centroid] = new PointWritable(attrPoin
                        }
                        br.close();
                }
        }

        hdfs.delete(new Path(folderOutputPath), true);

        return points;
}

private static boolean checkStopKMean(PointWritable[] oldCentroids, PointWritab
        boolean needStop = true;

        System.out.println("Check for stop K-Means if distance <= " + threshold
        for (int i = 0; i < oldCentroids.length; i++) {

                double dist = oldCentroids[i].calcDistance(newCentroids[i]);
                System.out.println("distance centroid[" + i + "] changed: " + d
                needStop = dist <= threshold;
                // chỉ cần 1 tâm < ngưỡng thì return false
                if (!needStop) {
                        return false;
                }
        }
        return true;
}

private static void writeFinalResult(Configuration conf, PointWritable[] centro
                PointWritable[] centroidsInit) throws IOException {
        FileSystem hdfs = FileSystem.get(conf);
        FSDataOutputStream dos = hdfs.create(new Path(outputFilePath), true);
        BufferedWriter br = new BufferedWriter(new OutputStreamWriter(dos));
```

```
        for (int i = 0; i < centroidsFound.length; i++) {
                br.write(centroidsFound[i].toString());
                br.newLine();
                System.out.println("Centroid[" + i + "]:  (" + centroidsFound[i]
        }

        br.close();
        hdfs.close();
}

public static PointWritable[] copyCentroids(PointWritable[] points) {
        PointWritable[] savedPoints = new PointWritable[points.length];
        for (int i = 0; i < savedPoints.length; i++) {
                savedPoints[i] = PointWritable.copy(points[i]);
        }
        return savedPoints;
}

public static int MAX_LOOP = 50;

public static void printCentroids(PointWritable[] points, String name) {
        System.out.println("=> CURRENT CENTROIDS:");
        for (int i = 0; i < points.length; i++)
                System.out.println("centroids(" + name + ")[" + i + "]=> :" + p
        System.out.println("----------------------------------");
}

public int run(String[] args) throws Exception {

        Configuration conf = getConf();
        String inputFilePath = conf.get("in", null);
        String outputFolderPath = conf.get("out", null);
        String outputFileName = conf.get("result", "result.txt");

        int nClusters = conf.getInt("k", 3);
        float thresholdStop = conf.getFloat("thresh", 0.001f);
        int numLineOfInputFile = conf.getInt("lines", 0);
        MAX_LOOP = conf.getInt("maxloop", 50);
        int nReduceTask = conf.getInt("NumReduceTask", 1);
        if (inputFilePath == null || outputFolderPath == null || numLineOfInput
                System.err.printf(
                                "Usage: %s -Din <input file name> -Dlines <numb
                                getClass().getSimpleName());
                ToolRunner.printGenericCommandUsage(System.err);
                return -1;
        }
```

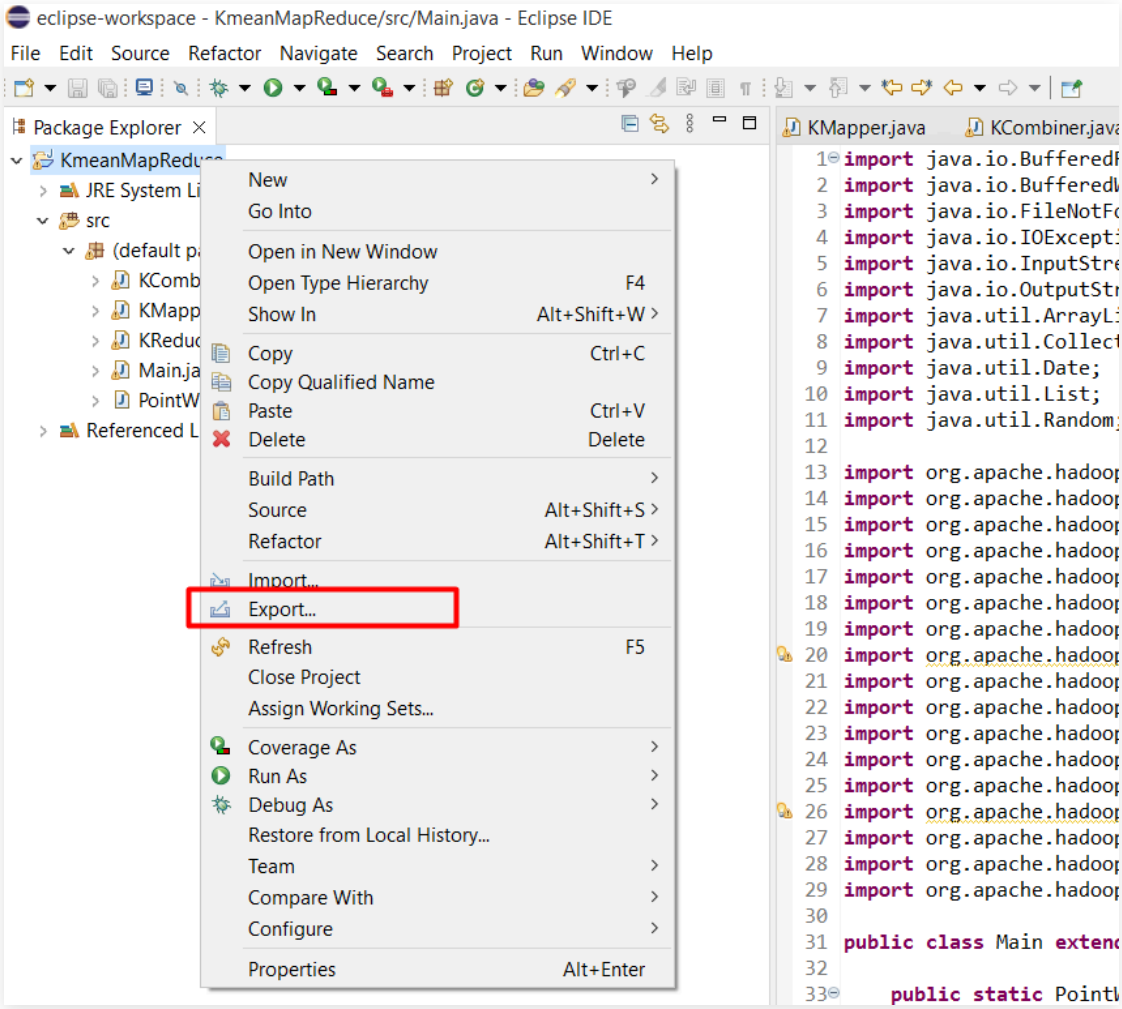
```
System.out.println("---------------INPUT PARAMETERS---------------");
System.out.println("inputFilePath:" + inputFilePath);
System.out.println("outputFolderPath:" + outputFolderPath);
System.out.println("outputFileName:" + outputFileName);
System.out.println("maxloop:" + MAX_LOOP);
System.out.println("numLineOfInputFile:" + numLineOfInputFile);
System.out.println("nClusters:" + nClusters);
System.out.println("threshold:" + thresholdStop);
System.out.println("NumReduceTask:" + nReduceTask);

System.out.println("--------------- STATR ---------------");
PointWritable[] oldCentroidPoints = initRandomCentroids(nClusters, numL:
PointWritable[] centroidsInit = copyCentroids(oldCentroidPoints);
printCentroids(oldCentroidPoints, "init");
saveCentroidsForShared(conf, oldCentroidPoints);
int nLoop = 0;

PointWritable[] newCentroidPoints = null;
long t1 = (new Date()).getTime();
while (true) {
        nLoop++;
        if (nLoop == MAX_LOOP) {
                break;
        }
        Job job = new Job(conf, "K-Mean");// Job thực hiện deepCopy con
        job.setJarByClass(Main.class);
        job.setMapperClass(KMapper.class);
        job.setCombinerClass(KCombiner.class);
        job.setReducerClass(KReducer.class);
        job.setMapOutputKeyClass(LongWritable.class);
        job.setMapOutputValueClass(PointWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(inputFilePath));

        FileOutputFormat.setOutputPath(job, new Path(outputFolderPath))
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setNumReduceTasks(nReduceTask);

        boolean ret = job.waitForCompletion(true);
        if (!ret) {
                return -1;

        }
```

```
                        newCentroidPoints = readCentroidsFromReducerOutput(conf, nClust
                        printCentroids(newCentroidPoints, "new");
                        boolean needStop = checkStopKMean(newCentroidPoints, oldCentroi

                        oldCentroidPoints = copyCentroids(newCentroidPoints);

                        if (needStop) {
                                break;
                        } else {
                                saveCentroidsForShared(conf, newCentroidPoints);
                        }

                }
                if (newCentroidPoints != null) {
                        System.out.println("------------------- FINAL RESULT ----------
                        writeFinalResult(conf, newCentroidPoints, outputFolderPath + "/
                }
                System.out.println("----------------------------------------------");
                System.out.println("K-MEANS CLUSTERING FINISHED!");
                System.out.println("Loop:" + nLoop);
                System.out.println("Time:" + ((new Date()).getTime() - t1) + "ms");

                return 1;
        }

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

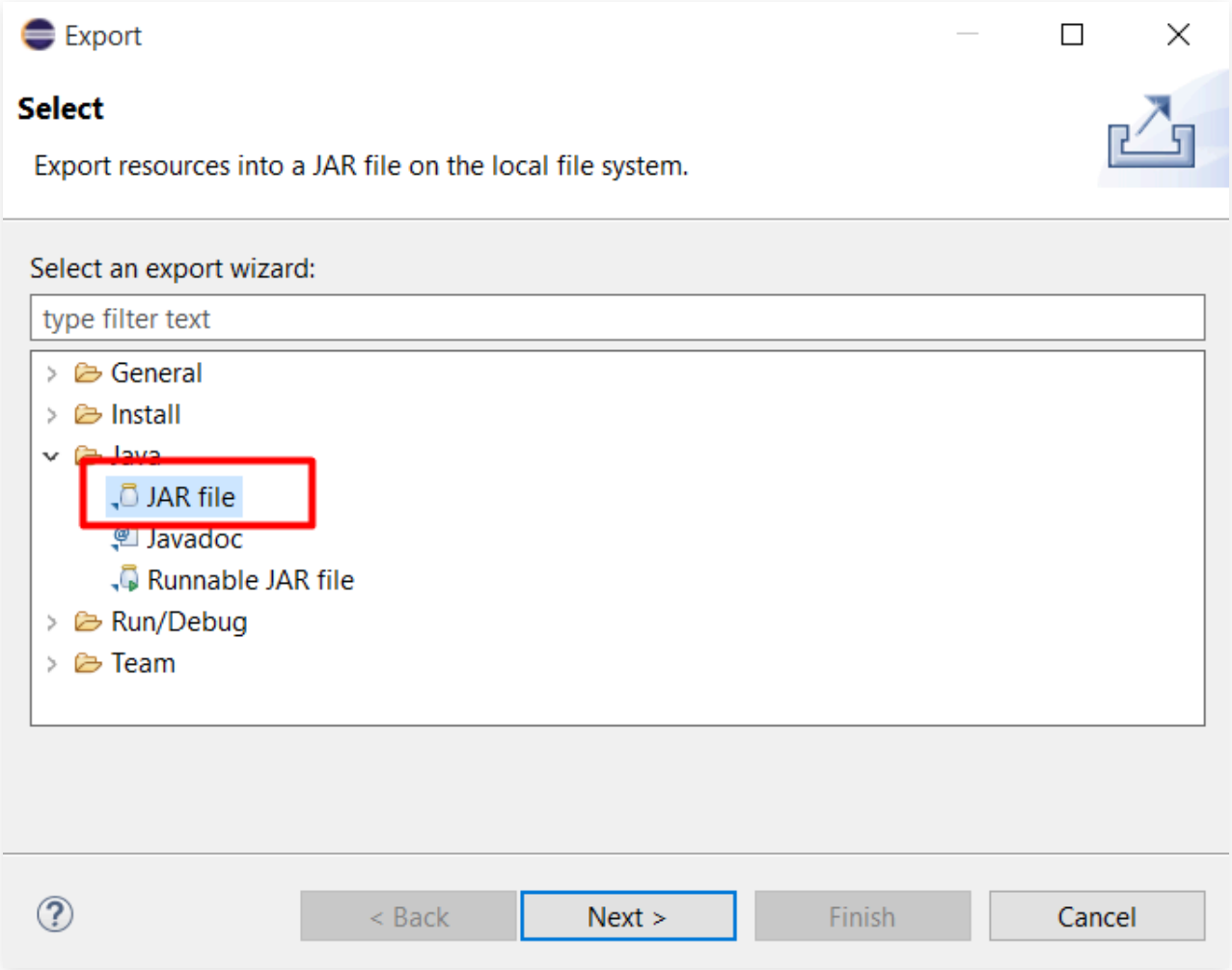


Bước 7: Tạo file JAR

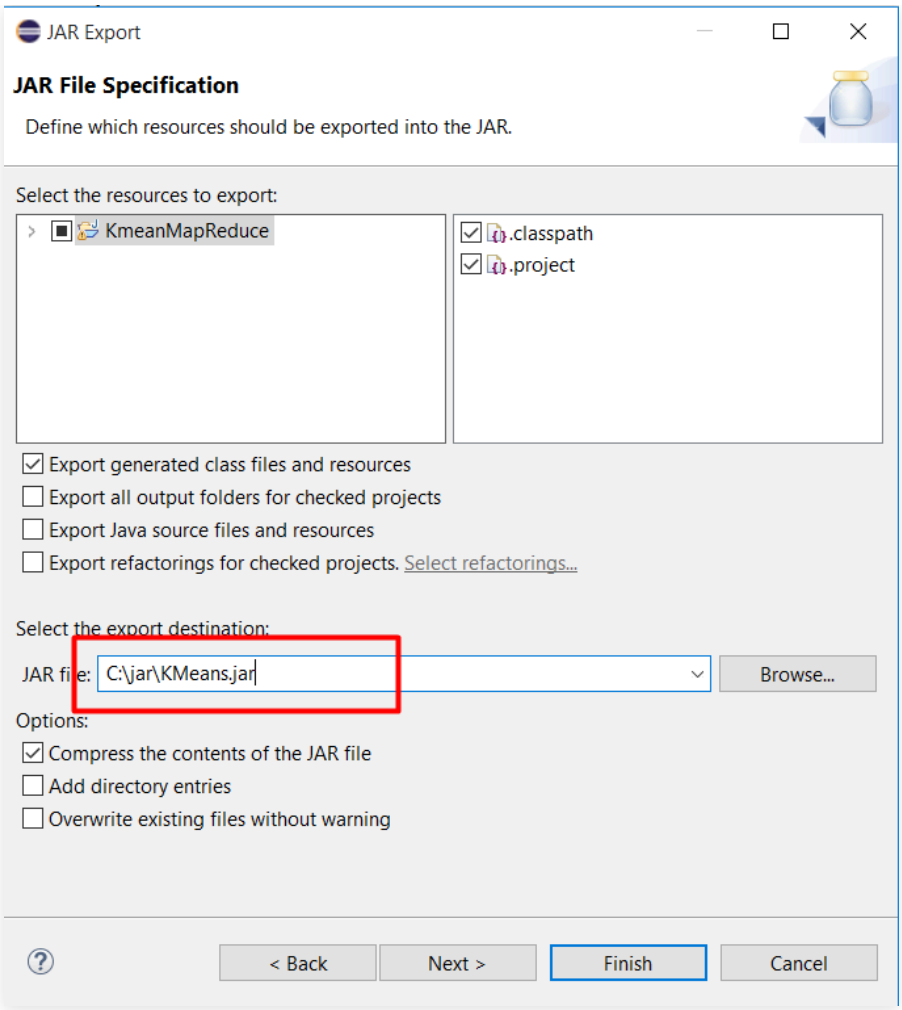
Chuột phải vào project **KmeanMapReduce** chọn **Export**



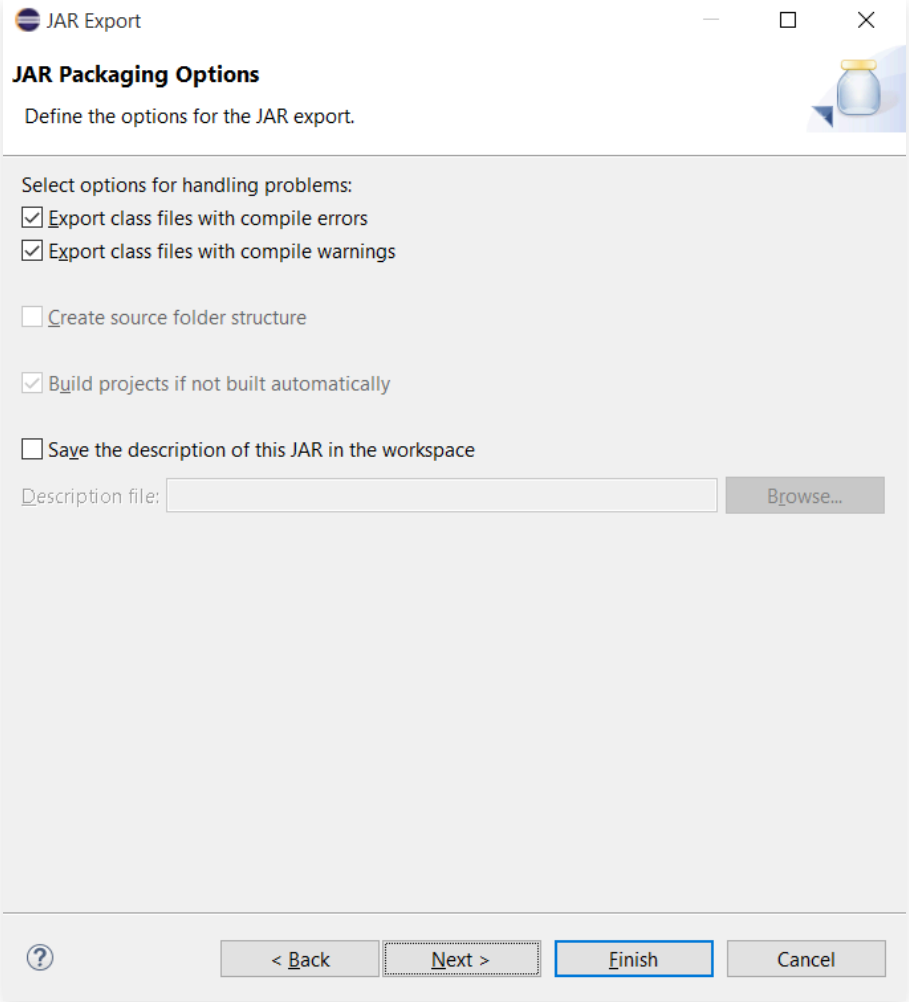
Chọn **Java > JAR File** rồi bấm **Next**



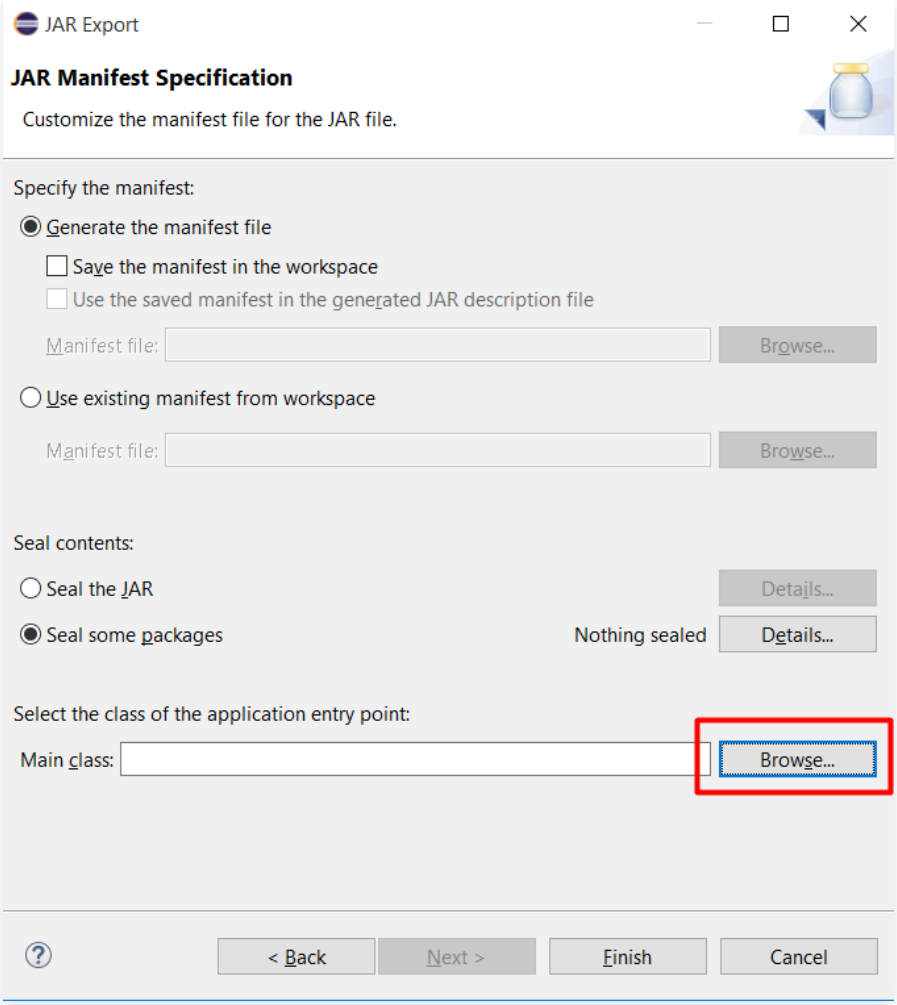
Chọn đường dẫn lưu file JAR và bấm **Next**


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
KmeanMapReduce
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: C:\jar\KMeans.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Finish
Cancel

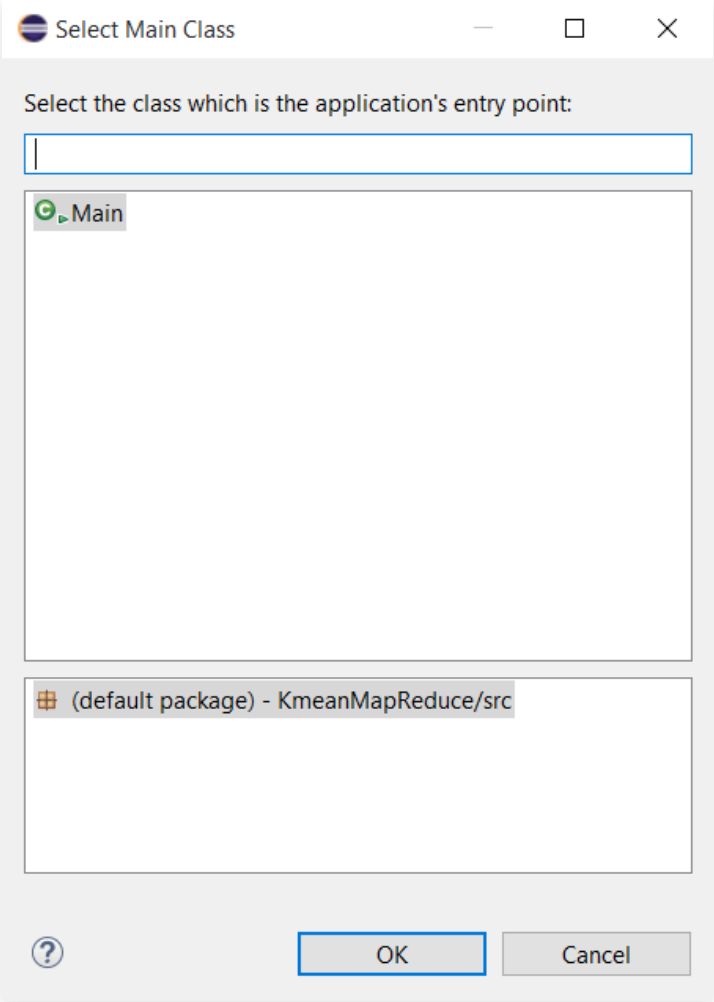
Bấm **Next**



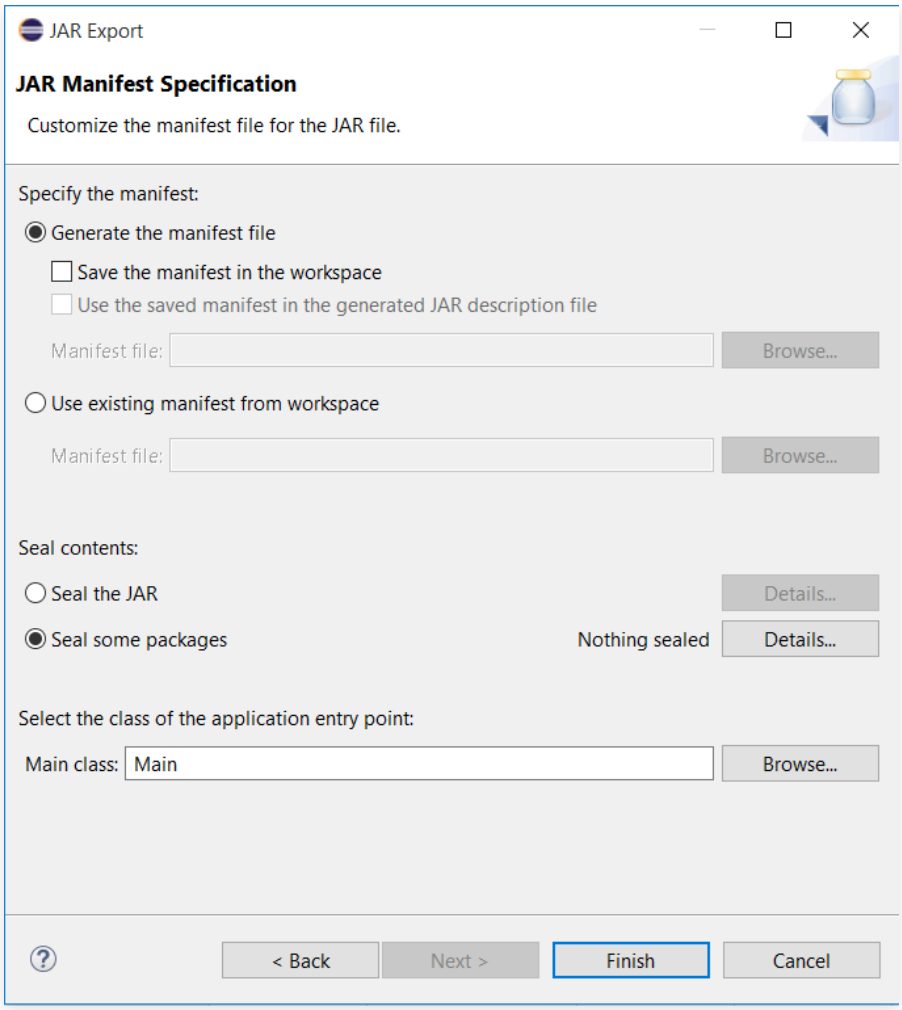
Bấm **Browser** để chọn file **main**



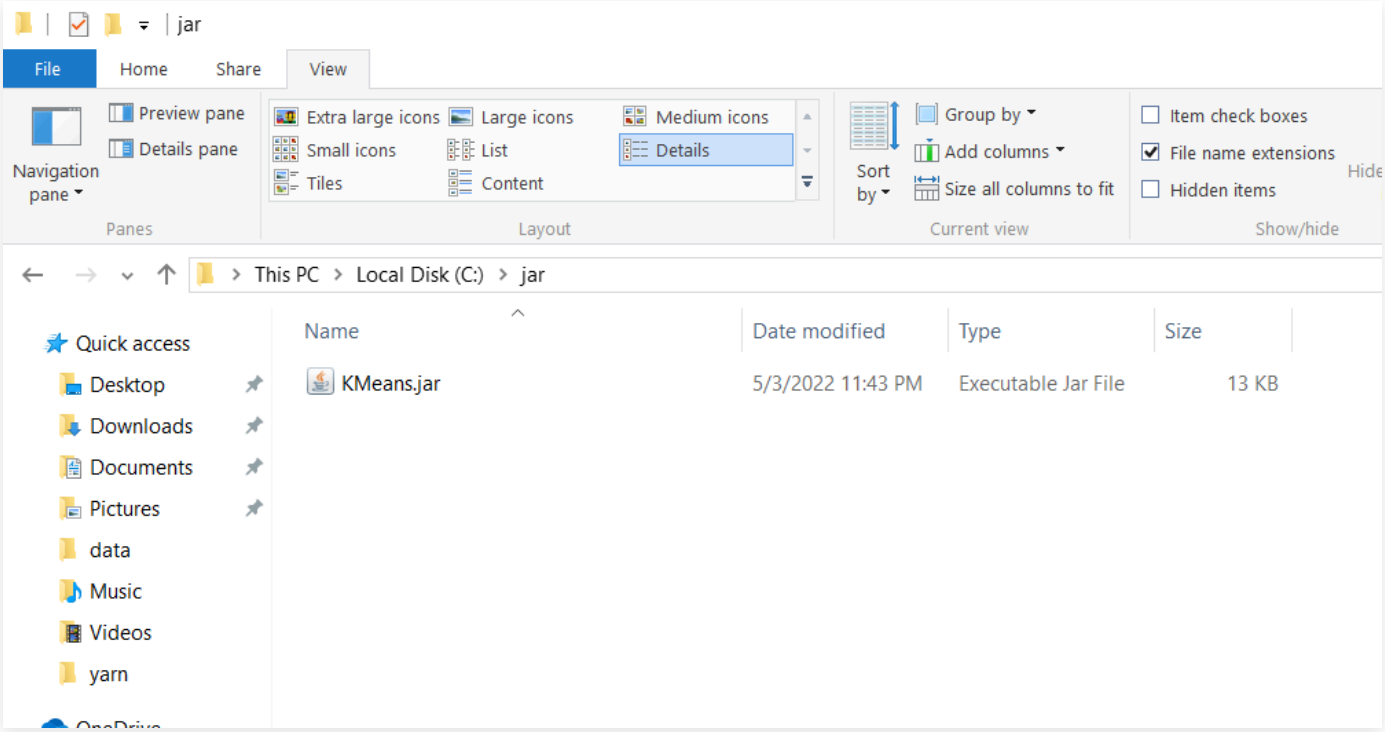
Chọn **Main** và bấm **OK**



Bấm **Finish** để thực hiện quá trình **Export**



Vào thư mục chứa lưu file JAR vừa tạo và kiểm tra kết quả



Bước 8: Thử nghiệm trên file **data-kmeans.txt**

Thử nghiệm trên file dữ liệu **data-kmeans.txt** đã tạo ở trên, và kết quả thu được lưu tại **result.txt** trong thư mục **k-output**. Chạy lệnh sau:

```
hadoop jar C:\jar\KMeans.jar -Din /k-input/data-kmeans.txt  -Dlines 30 -Dresult result.
```

> *Lưu ý: Thay "C:\jar\KMeans.jar" bằng đường dẫn chứa file JAR ở trên máy*

```
Administrator: Command Prompt - hadoop  jar C:\jar\KMeans.jar -Din /k-input/data-kmeans.txt  -Dlines 30 -Dresult result.txt -Dmaxl...
C:\Windows\system32>hadoop jar C:\jar\KMeans.jar -Din /k-input/data-kmeans.txt  -Dlines 30 -Dresult result.txt -Dmaxloop
 50 -Dk 4 -Dthresh 0.0001 -DNumReduceTask 2 -Dout /k-output
---------------INPUT PARAMETERS---------------
inputFilePath:/k-input/data-kmeans.txt
outputFolderPath:/k-output
outputFileName:result.txt
maxloop:50
numLineOfInputFile:30
nClusters:4
threshold:1.0E-4
NumReduceTask:2
--------------- STATR ---------------
Initializing random 4 centroids...
=> CURRENT CENTROIDS:
centroids(init)[0]=> :25.0,79.0
centroids(init)[1]=> :22.0,53.0
centroids(init)[2]=> :57.0,40.0
centroids(init)[3]=> :51.0,8.0
---------------------------------
2022-05-03 23:47:23,802 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2022-05-03 23:47:24,316 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
dachi/.staging/job_1651592845177_0001
2022-05-03 23:47:24,752 INFO input.FileInputFormat: Total input files to process : 1
2022-05-03 23:47:24,836 INFO mapreduce.JobSubmitter: number of splits:1
2022-05-03 23:47:24,991 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1651592845177_0001
2022-05-03 23:47:24,992 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-05-03 23:47:25,147 INFO conf.Configuration: resource-types.xml not found
2022-05-03 23:47:25,148 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-05-03 23:47:25,818 INFO impl.YarnClientImpl: Submitted application application_1651592845177_0001
2022-05-03 23:47:26,023 INFO mapreduce.Job: The url to track the job: http://DESKTOP-QBKQL72:8088/proxy/application_1651
```

Sử dụng lệnh sau để kiểm tra kết quả

```
hdfs dfs -cat /k-output/result.txt
```

```
Administrator: Command Prompt
C:\Windows\system32>hdfs dfs -cat /k-output/result.txt
30.83334,74.66667
27.75,55.0
55.1,46.1
43.2,16.7

C:\Windows\system32>_
```