Assignment 2

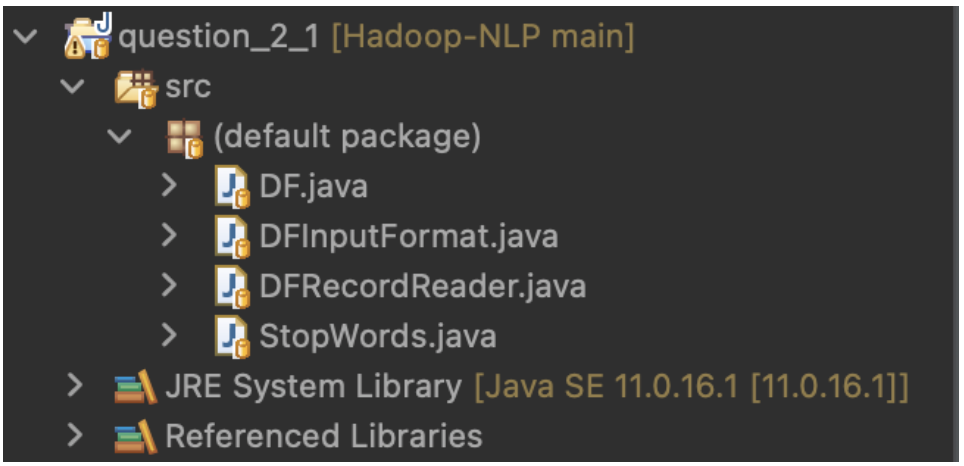
Solution 1

Solution 2

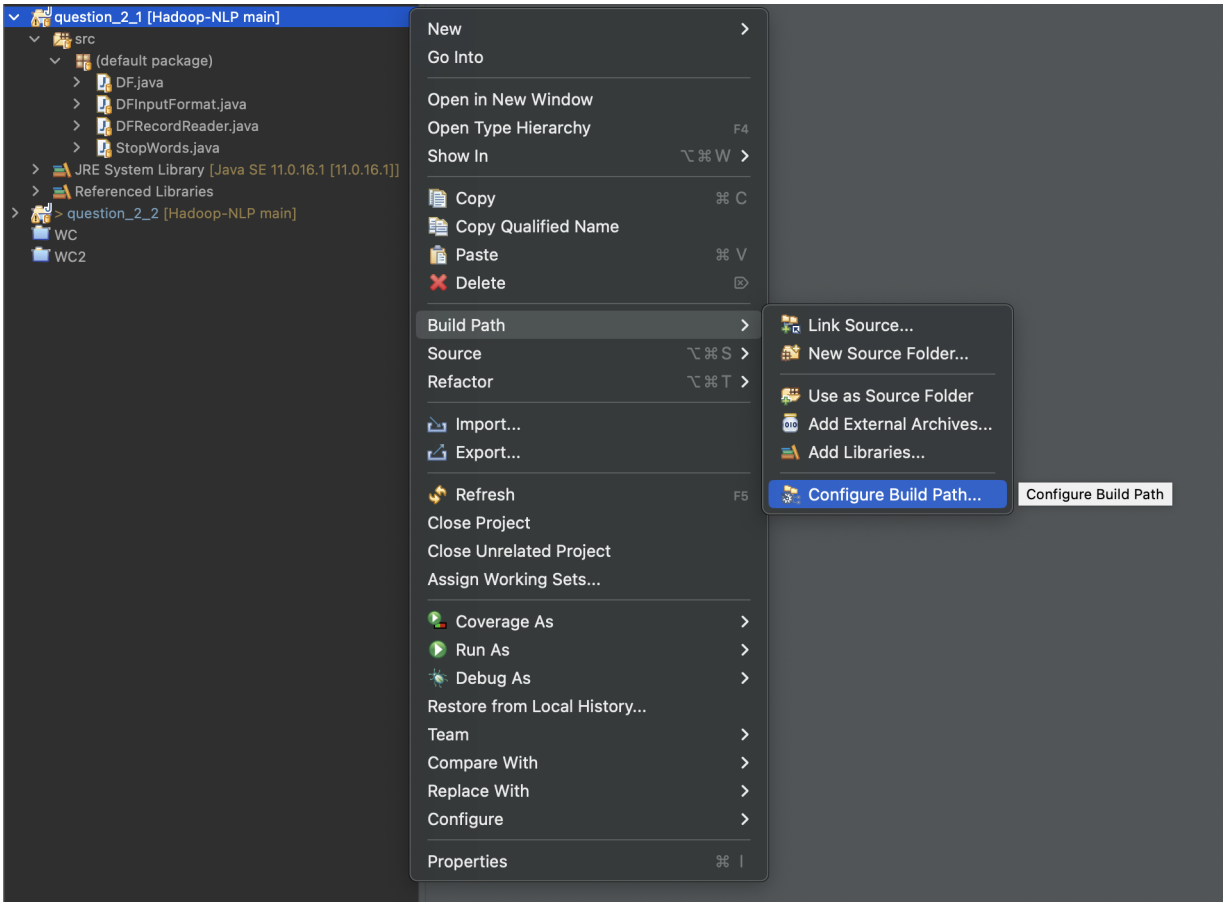
Solution for 2.1

How to Run the code:

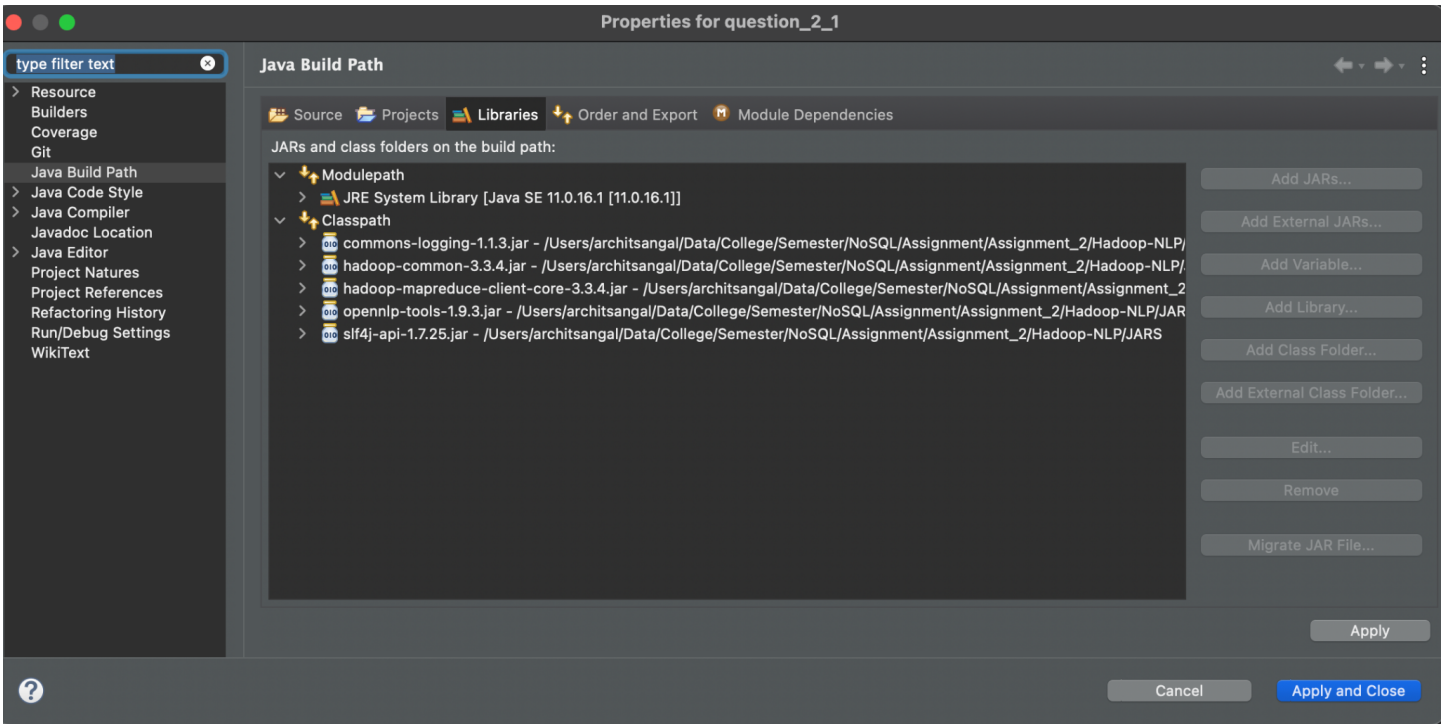
Open the project in eclipse with the file system in folder 2.1. Directory Structure will be something like below:



Right click on the project file structure and add external jar files to the project.



Select ‘Java Build Path’ from the right window. Click on the ‘Classpath’ and add all the jar files from ‘JARS’ folder.

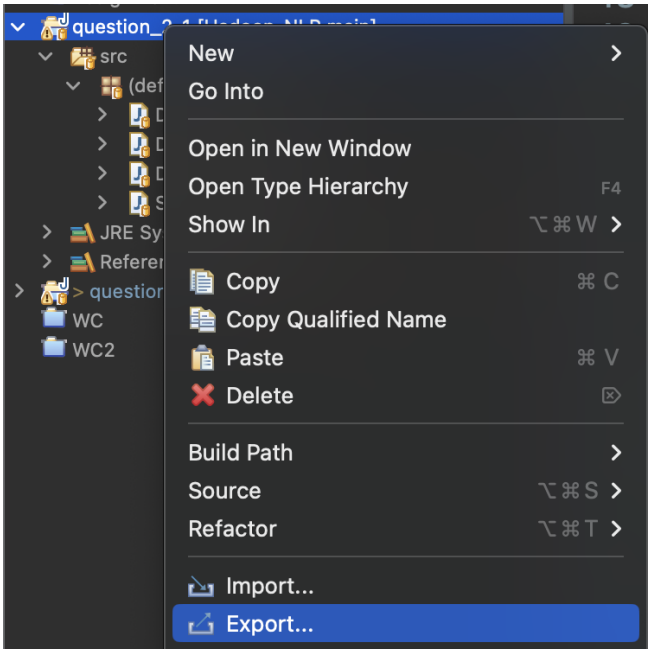


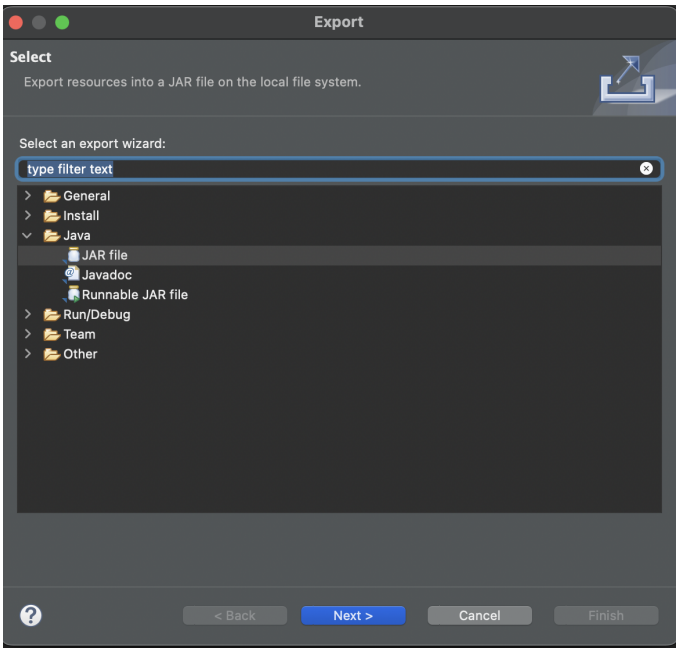
Make sure that **HADOOP_HOME** and **HADOOP_CLASSPATH.** On terminal this should return something like the following:

```
$ echo  ${HADOOP_HOME}
/opt/homebrew/Cellar/hadoop/3.3.4/libexec
```

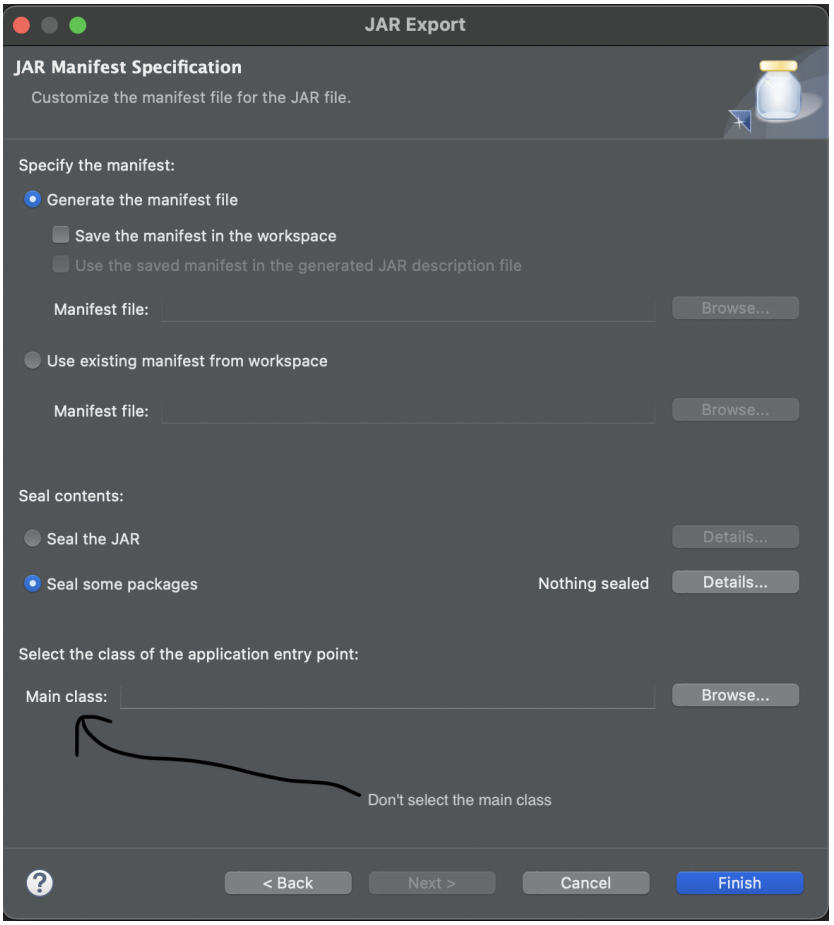
```
echo ${HADOOP_CLASSPATH}
/Users/architsangal/Data/College/Semester/NoSQL/Assignment/Assignment_2/Hadoop-NLP/JARS
/*
```

Now export the project in form of a ‘jar’ file. Save it to a folder called ‘TarFiles’.





Don’t save the ‘main class’, leave it empty.



Now move to the 'TarFiles' folder, and run the following command. Note that the output folder should not be there. It will be created in run time.

```
$ hadoop jar temp.jar DF ./../Wikipedia-articles/full/ ./output/full
```

The output can be found in the file **part-r-00000 (**at location 'TarFiles/output/full'**)**.

Code Explanation:

We have the following code in the main function:

```
public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "df");

    job.setMapperClass(TokenizerMapper.class);
    // job.setCombinerClass(IntSumReducer.class); // enable to use 'local
    // aggregation'
    job.setReducerClass(IntSumReducer.class);

    job.setInputFormatClass(DFInputFormat.class);
    // job.setInputFormatClass(DFInputFormat.class);

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Instances of configuration and jobs are created. As we are reading the entire document, and it must not get divided we need to use some custom class. They are the following:

```
@Override
protected boolean isSplitable(JobContext context, Path file) {
    return false;
}

@Override
public RecordReader<IntWritable, Text> createRecordReader(
        InputSplit split, TaskAttemptContext context) throws IOException,
        InterruptedException {
    DFRecordReader reader = new DFRecordReader();
    reader.initialize(split, context);
    return reader;
}
```

Notice isSplitable() returns false. And we return the CustomRecordReader.

Take the document as Text and store the words in the set after stemming them. And remove the unnecessary words like number, empty string, stop words, etc. pass the stemmed words in form of a (key,value) i.e. (word,1)

```
@Override
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();
    String[] tokens = line.split("[^\\w']+");

    PorterStemmer steammer = new PorterStemmer();
    for (int i = 0; i < tokens.length; i++)
        tokens[i] = steammer.stem(tokens[i]).toString();

    StopWords st = new StopWords();
    HashSet<String> words = st.words;

    HashSet<String> set = new HashSet<>();

    for (String token : tokens) {
        if (!(set.contains(token) || words.contains(token.toLowerCase()) || token.equals(""))) {
            set.add(token);
            word.set(token);
            context.write(word, one); // context is like the output
        }
    }
}
```

Below is the reducer that sums the value with the same key.

```
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Solution for 2.2

How to Run the code:

Steps remain the same, just the hadoop commands need to be modified, which is the following for test files:

```
$ hadoop jar temp_2.jar TFIDF ./../Wikipedia-articles/test/ ./output/secondtest/
./../TarFiles/output/secondtest/part-r-00000  ./output/secondsectest
```

There should be no folder in output directory with names: **secondtest** and **secondsectest**.

Run the following command for running it on the entire set.

```
$ hadoop jar temp_2.jar TFIDF ./../Wikipedia-articles/full/ ./output/secondfull/
./../TarFiles/output/secondfull/part-r-00000  ./output/secondsecfull
```

Code Explanation:

We have the following code in the main function:

```
public static void main(String[] args) throws Exception
{
    Configuration conf = new Configuration();

    Job job0 = Job.getInstance(conf, "MR TF");

    job0.addCacheFile(new URI("/Users/architsangal/Data/College/Semester/NoSQL/Assignment/Assignment_2/Hadoop-NLP/TarFiles/output/full/part-r-00000"));

    job0.setInputFormatClass(DFInputFormat.class);

    job0.setMapperClass(MapTF.class);
    job0.setReducerClass(ReduceTF.class);

    job0.setOutputKeyClass(Text.class);
    job0.setOutputValueClass(MapWritable.class);

    job0.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.setInputPaths(job0, new Path(args[0]));
    FileOutputFormat.setOutputPath(job0, new Path(args[1]));

    ////////////////////
    Configuration conf1 = new Configuration();
    Job job1 = Job.getInstance(conf1, "Stripes");

    job1.addCacheFile(new URI("/Users/architsangal/Data/College/Semester/NoSQL/Assignment/Assignment_2/Hadoop-NLP/TarFiles/output/full/part-r-00000"));

    job1.setMapperClass(Map.class);
    job1.setReducerClass(Reduce.class);

    job1.setOutputKeyClass(Text.class);
    job1.setOutputValueClass(Text.class);

    FileInputFormat.setInputPaths(job1, new Path(args[2]));
    FileOutputFormat.setOutputPath(job1, new Path(args[3]));

    ////////////////////

    if(job0.waitForCompletion(true))
        System.exit(job1.waitForCompletion(true) ? 0 : 1);


}
```

So we have two jobs here:

1. For finding TF
2. For finding the Score using DF (imported from the previous part 2.1 i.e. **TarFiles/output/full/part-r-00000**) and TF.

The 2 jobs need to be cascaded in nature. Hence, we have that last if condition.

2.2.1 Finding TF

We have a class Top which stores the top DF words using setup() of the mapper class of the TF. It takes the cached files and takes the input. It does some preprocessing on the data and choose top 100 words.

```java
@Override
public void setup(Context context) throws IOException, InterruptedException
{
    if(Top.initialized)
    {
        return;
    }
    // Load DF values from the cached file into a main-memory data structure
    URI[] cacheFiles = context.getCacheFiles();
    if (cacheFiles != null && cacheFiles.length > 0) {
        try
        {
            HashMap<Integer,ArrayList<String>> map = new HashMap<Integer,ArrayList<String>>();
            PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

            BufferedReader bufferedReader = new BufferedReader(new FileReader(cacheFiles[0].toString()));
            String line;
            while ((line = bufferedReader.readLine()) != null)
            {
                String[] words = line.trim().split("\t");
                String val = words[0];
                int DF = Integer.parseInt(words[1]);
                pq.add(DF);
                if(map.containsKey(DF))
                    map.get(DF).add(val);
                else
                {
                    map.put(DF, new ArrayList<String>());
                    map.get(DF).add(val);
                }

                  System.out.println(val + " " + DF);
            }

            Top obj = new Top();

            bufferedReader.close();

            int count = 0;
            ArrayList<String> answer = new ArrayList<String>();
            ArrayList<Integer> df = new ArrayList<Integer>();
            while(count<=100)
            {
                int DF = pq.remove();
                try
                {
                    Integer.parseInt(map.get(DF).get(0));
                }
                catch(Exception e)
                {
                    answer.add(map.get(DF).get(0));
                    System.out.print("\"" + map.get(DF).get(0) + "\",");
                    df.add(DF);
                    Top.newtop(map.get(DF).get(0));
                    Top.newfreq(DF);
                    count++;
                }
                map.get(DF).remove(0);
            }

            System.out.println();
            for(int DF : df)
            {
                System.out.print(DF + ",");
            }
        } catch (Exception e) {
            System.out.println("Error while reading cache : " + e.toString());
            System.exit(1);
        }
    }
}
```

Below is the code of the map function.It takes the words of the document (here also we use the custom class for taking the entire document as input). Takes the words from the document and stems it compare it to DF. If the word is there in top DF then put it in a map which act like a stripes.

```
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {

        String line = value.toString();
        String[] tokens = line.split("[^\\w']+");

        PorterStemmer steammer = new PorterStemmer();
        for (int i = 0; i < tokens.length; i++)
            tokens[i] = steammer.stem(tokens[i]).toString();
        int[] freq = new int[100];
        Top obj = new Top();
        String[] top = obj.top;
        for(String token : tokens)
        {
            for(int i=0;i<100;i++)
            {
                if(token.equals(top[i]))
                {
                    freq[i]++;
                }
            }
        }

        MapWritable map = new MapWritable();
        for(int i=0;i<100;i++)
        {
            map.put(new Text(top[i]),new LongWritable(freq[i]));
        }

        context.write(new Text(key.toString()), map);
    }
}
```

Just store the key and value as output.

```
 public static class ReduceTF extends Reducer<Text, MapWritable, Text, Text>
 {

     @Override
     public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException
     {

         for (MapWritable value : values)
         {
             for(MapWritable.Entry<Writable, Writable> e : value.entrySet())
             {
                 System.out.println(key.toString());
                 context.write(new Text(key.toString()+"\t"+e.getKey().toString()),new Text(e.getValue().toString()));
             }

         }
     }
 }
```

The above was a map reduce for finding the TF. Now we have a map reduce for finding the score. Each line of the output of the previous line is a key,value pair for us in the mapper. Mapper does some preprocessing and passes the key and value to a reducer.

```
public static class Map extends Mapper<Object, Text, Text, Text>
{
    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException
    {

        String line = value.toString();
        String[] tokens = line.split("\t",-1);

        context.write(new Text(tokens[0]+"\t"+tokens[1]), new Text(tokens[2]));
    }
}
```

Reducer calculates the score using the TF and DF already cached or passed from map. And store the final output.