Institut supérieur d'informatique
et de mathématiques de Monastir

Année Universitaire : 2022/2023

Niveau : L3 INFO
Enseignante : Asma KERKENI

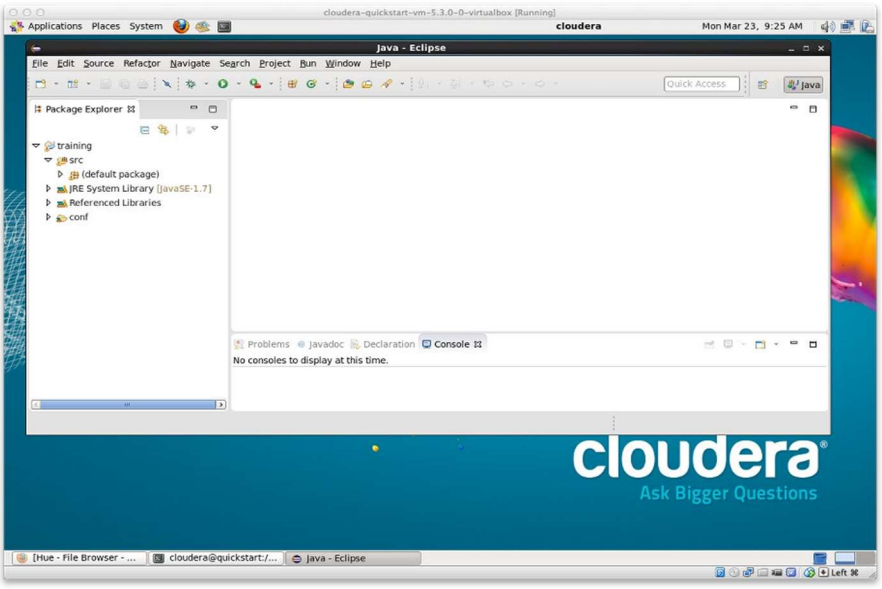
Big Data

TP2: Premier programme Hadoop Map-Reduce

Dans ce TP, nous allons créer le projet Java **WordCount** avec Eclipse pour Hadoop.
Le **WordCount** est l'équivalent du **HelloWorld** pour les applications de traitement de données. Il permet de calculer le nombre de mots dans un fichier donné, en décomposant le calcul en deux étapes :

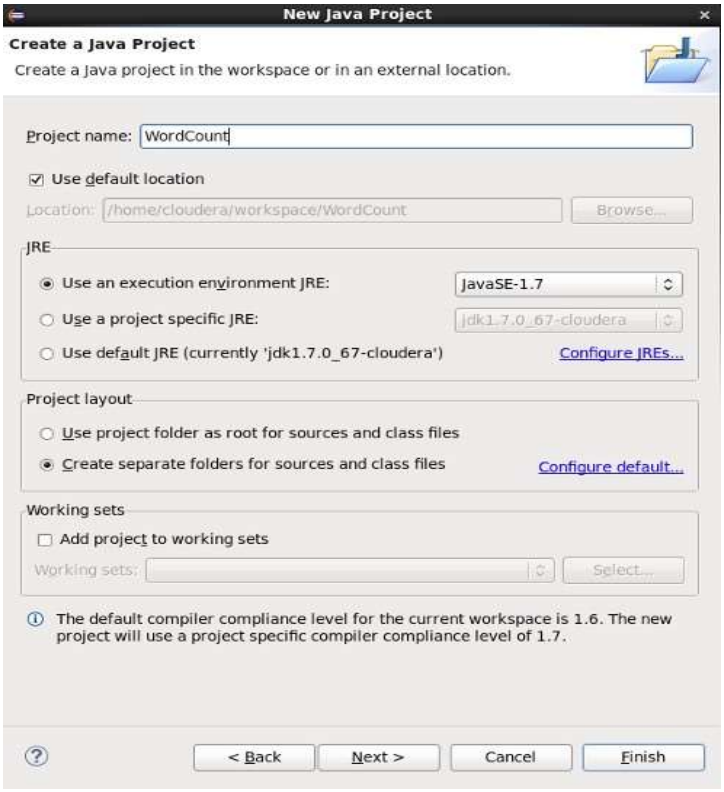
- L'étape de Mapping, qui permet de découper le texte en mots et de délivrer en sortie un flux textuel, où chaque ligne contient le mot trouvé, suivi de la valeur 1 (pour dire que le mot a été trouvé une fois).
- L'étape de Reducing, qui permet de faire la somme des 1 pour chaque mot, pour trouver le nombre total d'occurrences de ce mot dans le texte.

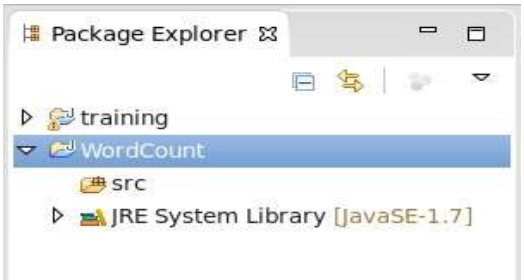
Bien que nous travaillions dans Cloudera VM, la création du projet Java peut être appliquée à n'importe quel environnement.



1. Création du projet WordCount

- Dans les menus d'Eclipse, cliquez sur « *File>New>Project> Java Project> Next* ».
- Choisir **« WordCount »** comme nom de projet et cliquer sur *« Finish »*.



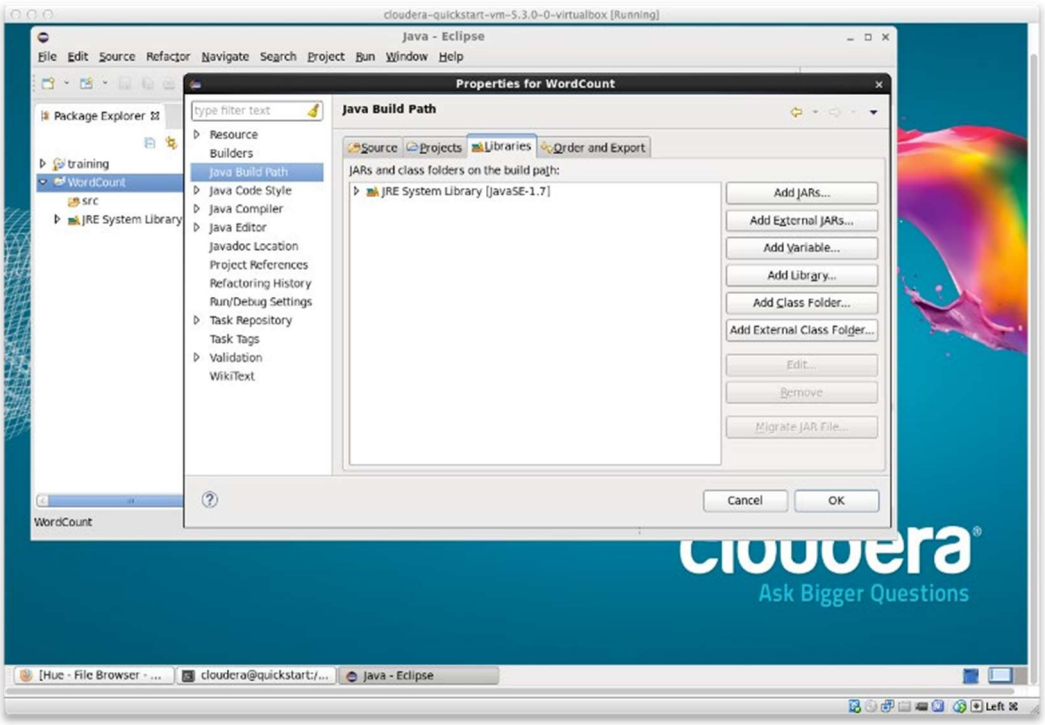


2. Ajout des dépendances au projet

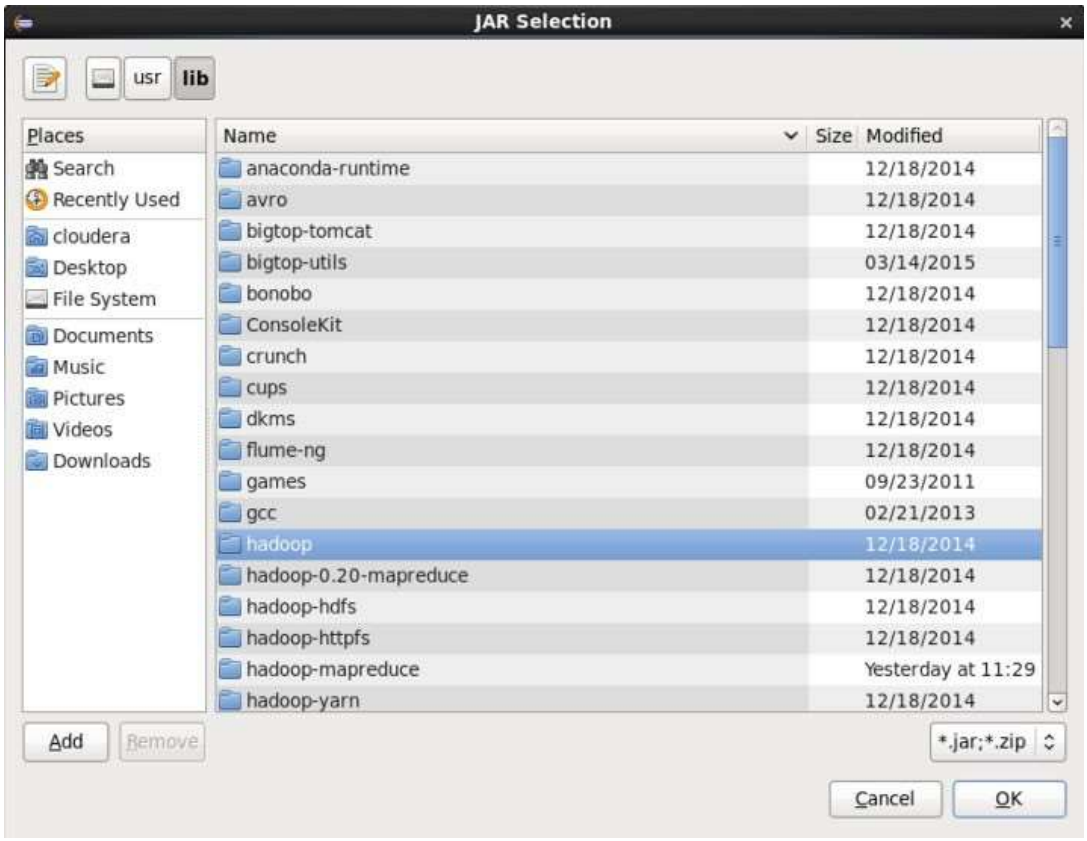
Maintenant, il faut ajouter les dépendances suivantes pour Hadoop, HDFS et Map Reduce :

- *hadoop-common,*
- *hadoop-mapreduce-client-common,*
- *hadoop-mapreduce-client-core*
- *hadoop-hdfs.*

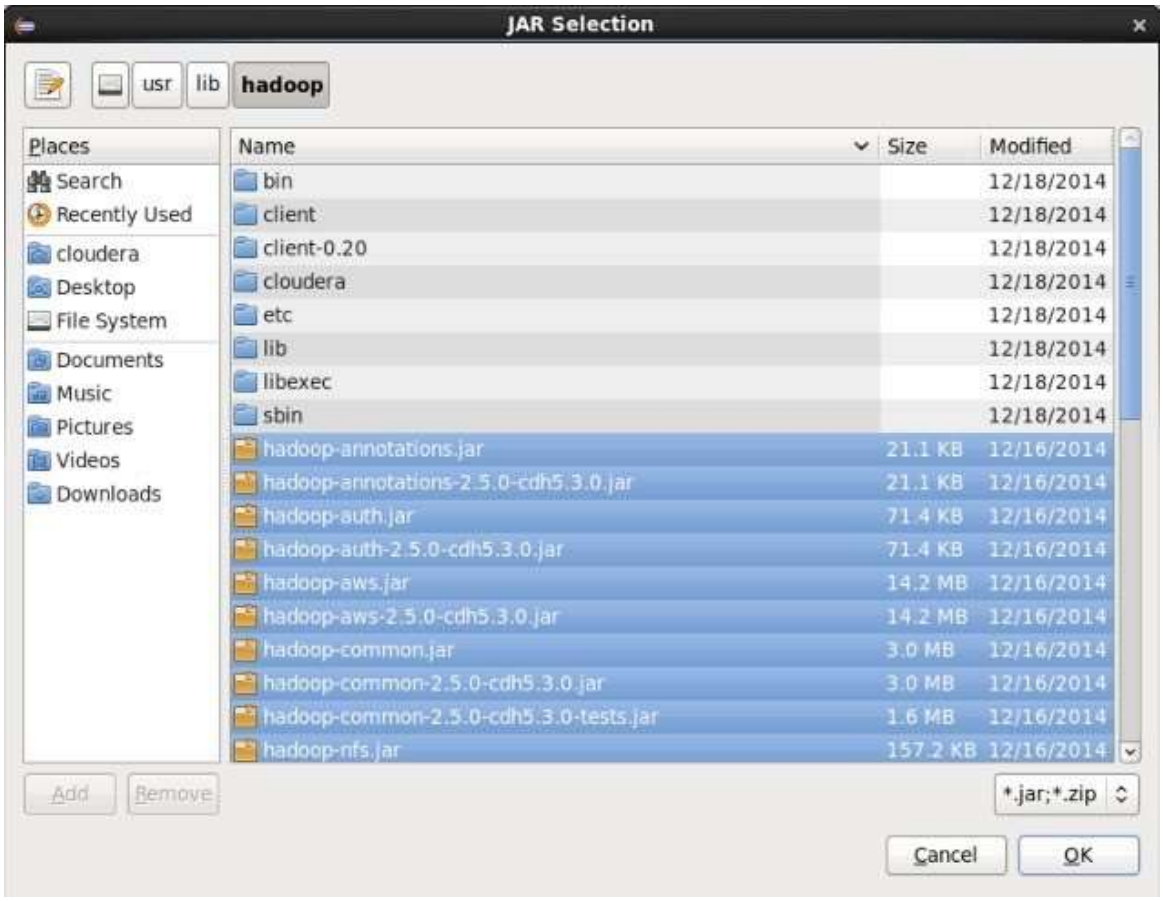
- Faites un clic droit sur le projet **WordCount** et sélectionnez *« Properties »*.
- Sélectionnez *« Java Build Path »* puis cliquez sur *« Libraries ».*

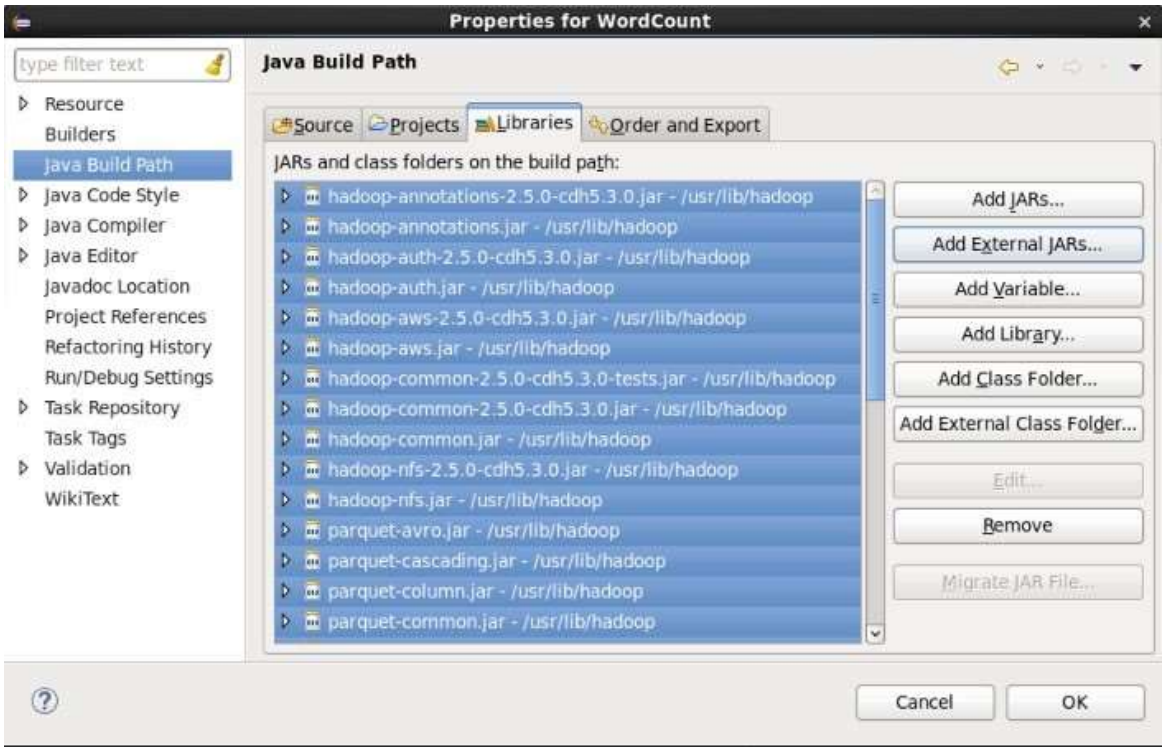


- Cliquez sur *« Add External JARs... »*, puis sur *« File System > usr > lib > hadoop »* :

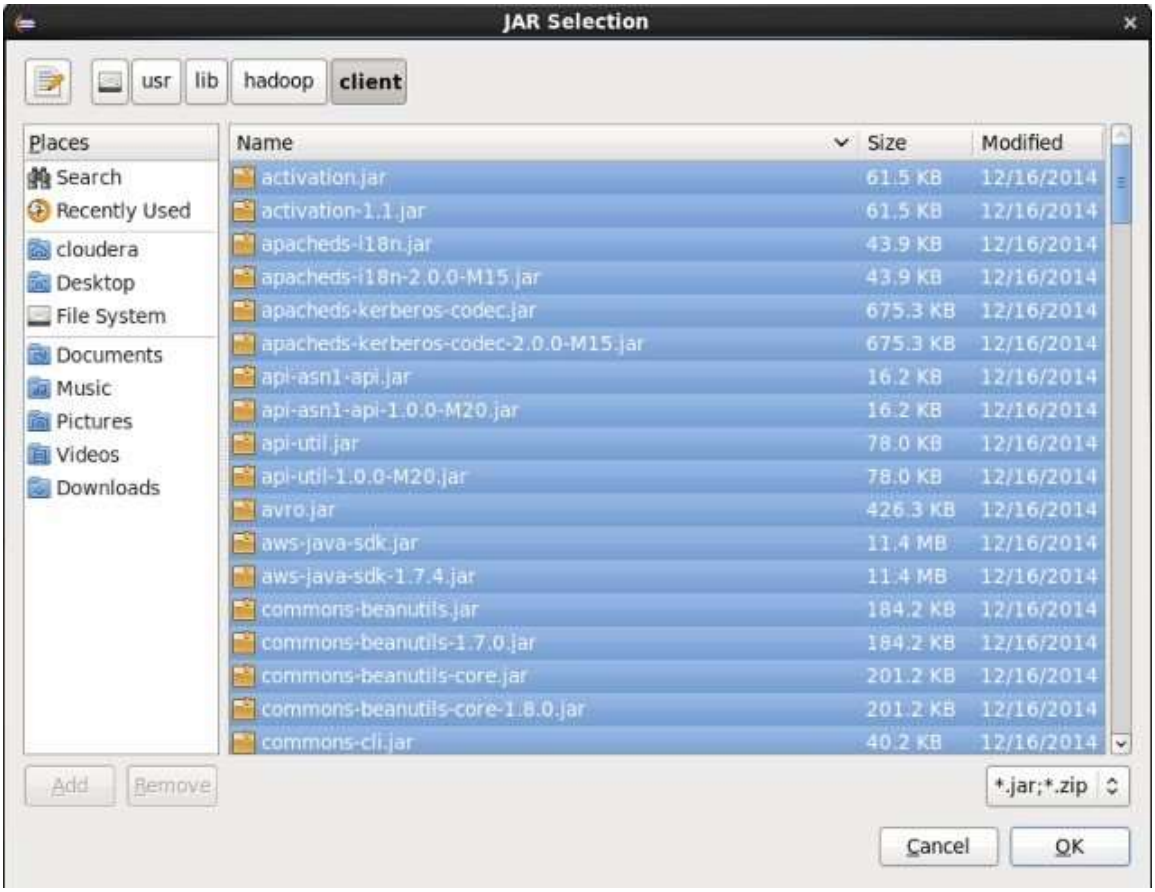


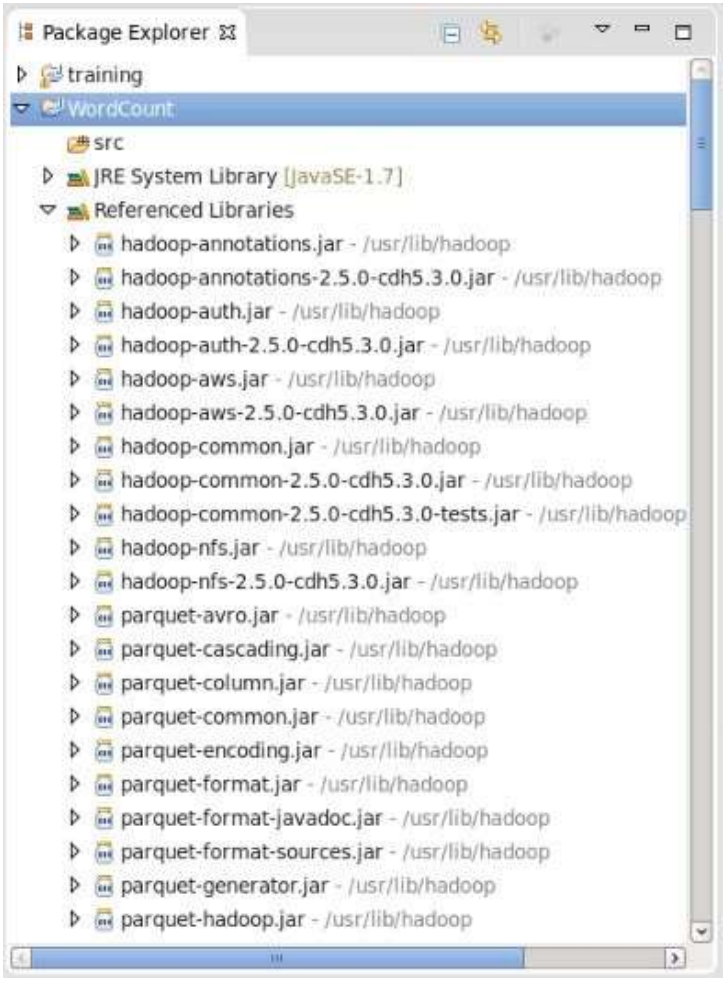
- On va sélectionner tous les jars, et cliquer sur OK.





- Nous n'avons pas encore fini. Nous devons ajouter d'autres librairies externes. Allez à *« Add External JARs... »* encore une fois, puis prenez toutes les librairies dans *«File System > usr >lib > hadoop>Client »* :
- Ensuite, cliquez sur *« OK ».*


JAR Selection
usr
lib
hadoop
client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
activation.jar 61.5 KB 12/16/2014
activation-1.1.jar 61.5 KB 12/16/2014
apacheds-i18n.jar 43.9 KB 12/16/2014
apacheds-i18n-2.0.0-M15.jar 43.9 KB 12/16/2014
apacheds-kerberos-codec.jar 675.3 KB 12/16/2014
apacheds-kerberos-codec-2.0.0-M15.jar 675.3 KB 12/16/2014
api-asn1-api.jar 16.2 KB 12/16/2014
api-asn1-api-1.0.0-M20.jar 16.2 KB 12/16/2014
api-util.jar 78.0 KB 12/16/2014
api-util-1.0.0-M20.jar 78.0 KB 12/16/2014
avro.jar 426.3 KB 12/16/2014
aws-java-sdk.jar 11.4 MB 12/16/2014
aws-java-sdk-1.7.4.jar 11.4 MB 12/16/2014
commons-beanutils.jar 184.2 KB 12/16/2014
commons-beanutils-1.7.0.jar 184.2 KB 12/16/2014
commons-beanutils-core.jar 201.2 KB 12/16/2014
commons-beanutils-core-1.8.0.jar 201.2 KB 12/16/2014
commons-cli.jar 40.2 KB 12/16/2014
Add
Remove
.jar;.zip
Cancel
OK



3. Implémentation des classes

Maintenant, il faut implémenter les classes *« TokenizerMapper », « IntSumReducer »* et *« WordCount »*. Le Mapper et Reducer peuvent être déclarés comme des classes internes. Ce n'est pas imposé par le framework, c'est même déconseillé.

- Créer la classe « *TokenizerMapper »*, contenant ce code :

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import java.util.StringTokenizer;

public class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Mapper.Context context
  ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
          context.write(word, one);
      }
```

```
    }
  }
```

- Créer la classe « *IntSumReducer* », contenant ce code :

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

public class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
    ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            System.out.println("value: "+val.get());
            sum += val.get();
        }
        System.out.println("--> Sum = "+sum);
        result.set(sum);
        context.write(key, result);
    }
}
```

- Créer la classe « *WordCount* », contenant ce code :

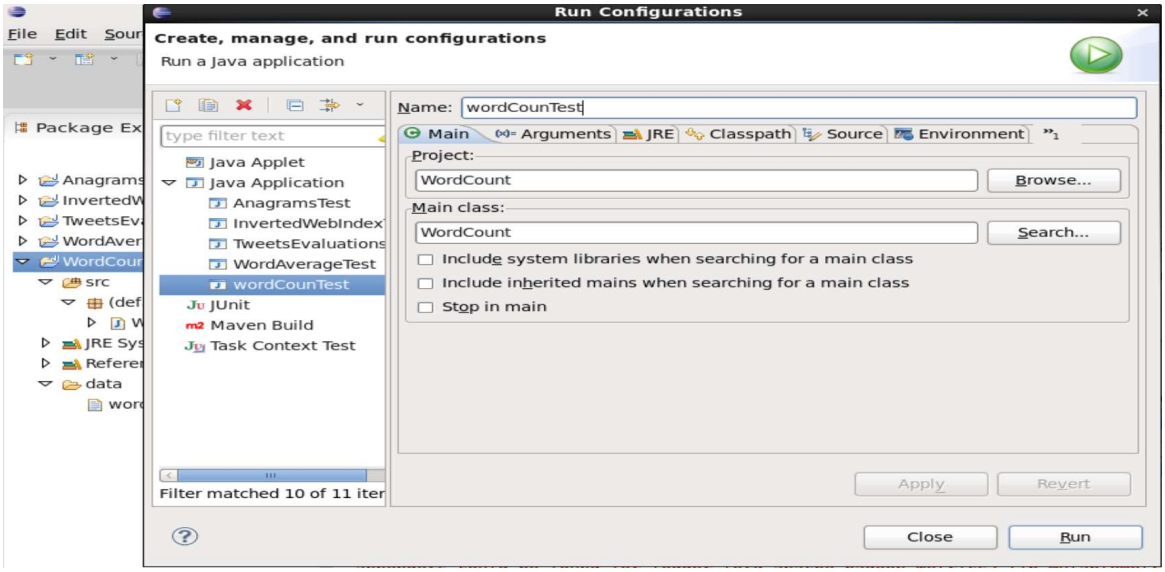
```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.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);
    }
}
```

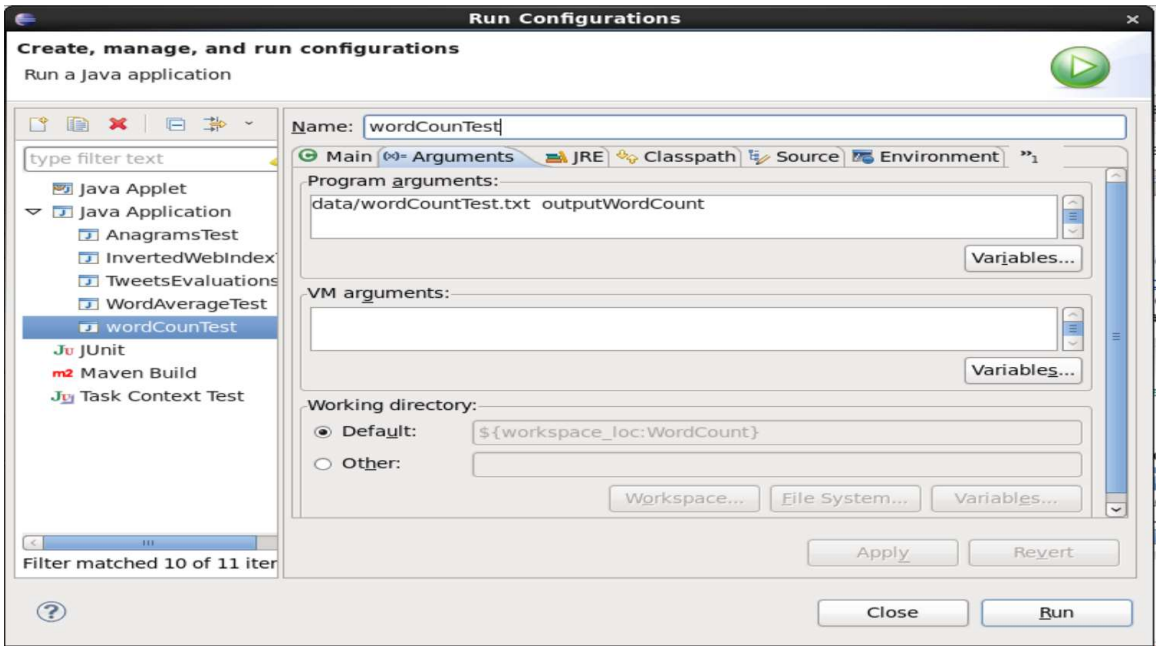
4. Exécution locale

Avant de lancer un programme MapReduce sur un cluster, on le teste toujours localement, sur un échantillon des données à traiter. Dans ce cas, le framework ne lit/écrit pas sur HDFS, mais sur votre système de fichiers local.

- Créez un programme qui compte le nombre d'occurrences de chaque mot dans un fichier texte (cf. *« WordCount »* vu en cours).
- Créez à la racine du projet un fichier texte, avec le (petit) contenu de votre choix.
- La fonction main prend en argument le chemin du fichier d'entrée et de sortie. Avant de lancer le programme avec Eclipse, il faut donc lui indiquer ces chemins (vous pouvez utiliser des chemins relatifs à la racine du projet). Sélectionnez le projet puis cliquez sur *« Run as > Run Configuration »* et indiquer le nom du projet, de la classe main.



- Ajoutez les arguments et cliquez sur *Apply*.



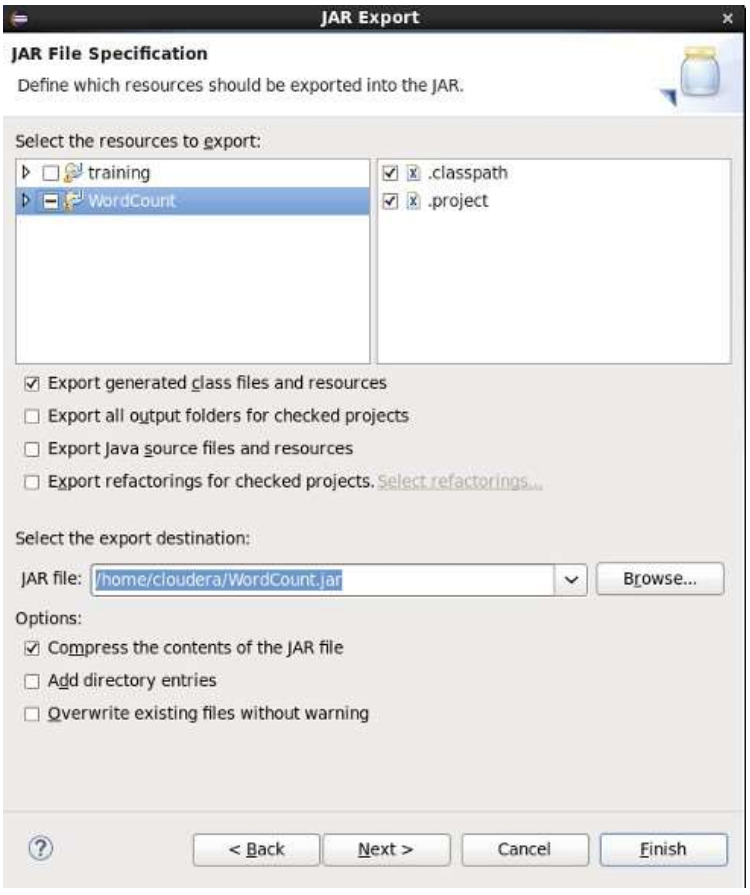
- Cliquez sur *Run* pour lancez le programme. Les traces d'exécution du framework apparaissent dans l'onglet *« Console ».*
- En cas de bug, vous y verrez l'exception à l'origine du plantage. Ouvrez le fichier généré et vérifiez vos résultats.
- Quand le programme marche, un répertoire « *outputWordCount* » sera créé contenant notamment un fichier part-r-00000 contenant le résultat.

5. Exécution sur le cluster

Le programme écrit, s'il fonctionnait localement, devrait fonctionner sur le cluster sans modifications. Mais il faut d'abord en faire un paquet compilé.

- Dans le navigateur d'Eclipse, faites un clic droit sur le répertoire « *src* ». Cliquez sur *« Export ».*
- Ouvrez la catégorie *« Java »,* puis double-cliquez sur *« JAR file ».*

- Parmi les ressources à exporter, vérifiez que le répertoire lib n'est pas sélectionné.
- Entrez le nom du paquet, par exemple *« WordCount.jar »*.



- Le paquet généré sera créé dans la destination indiquée.

- Nous allons maintenant compter le nombre d'occurrences d'un fichier texte *« worcount.txt »*. Ce fichier doit être placé sur HDFS.
- Maintenant on va exécuter le job MapReduce en tapant :

```
hadoop jar <chemin>/WordCount.jar WordCount wordcount.txt output
```

- Pour visualiser le résultat obtenu, tapez :

```
hadoop fs -ls

output

hadoop fs -cat output/part-00000
```