Hadoop first exercise[1]

For this exercise you have to prepare a report including:

- Your answers discussing the questions proposed along the exercises.
- Your source codes: a file per problem (e.g. : Question3_1.java for question 3.1). In each file precise in a comment the compiling and execution commands.
- Compress everything and sent a .zip or a .tar to genoveva.vargas@gmail.com

1.1 WordCount

1.1.1 Preparing the exercise

Using the file `WordCount.java` that contains the `mapper`, the `reducer` and the `main` program of Word – Count

- Compile the file and prepare a `jar`
- Prepare some text documents and copy them to HDFS
- Execute the program `MapReduce` on your data and make sure that results are correct.
- Now test your program giving as input the 5 tomes of Victor Hugo's *Les Misérables* located at `TutoHadoop/AllData`

1.1.2 Observing the counters

Question 1: Observe the counters of your job on the web interface of `JobTracker`:

1. What dooes *Reduce input groups* mean? What can we do for changing this value?
2. 2Waht does *Map input records* mean? and, *Map output records*?
3. In your opinion how are *Map input records* and *Map output records* related?

Question 2: Add a `combiner` to your example and execute again the code on the 5 tomes of Victor Hugo's *Les Misérables*.

1. Which are the counters that let verify that the `combiner` worked?

2. How can we estimate the advantage of using a `combiner`? Give its values when using and not using the combiner and justify your answer.

1.2 In-mapper combining

The book « *Data-Intensive Text Processing with MapReduce* » page 43 presents the technique *in-mapper combining*. The principle is to avoid using a combiner and introduce its work within the function map. Therefore, do not call the `combiner`, exploit instead the fact that in the class java `Mapper`, MapReduce:

- Calls `public void setup(Context)`
- Then calls `public void map(...)`
- Once map has processed all data, the public method `void cleanup(Context)` is called.

---

[1] This exercise was conceived and proposed by Dr. Alexandre Termier, University Joseph Fourier, France.

Question 3:

1. Do the necessary modifications in the `class Mapper`, using the following instructions and the pseudo – code given in the book.
2. Make sure that the new implementation works correctly. Which are the counters that let easily verify that everything worked fine? Which are the expected values?

1.3 Analysing public data

The data of the English city Bedford-Borrough are provided in `TutoHadoop/Data`. The meaning of columns is:

```
column 1: category ; column 2: provider ; column 3: transaction ; column 4: amount
```

Question 4: Write a `MapReduce` computing the sum of payments done by the city for each category.

Question 5: Modify the previous code for producing, besides the previous results the total sum of all payments despite their category.

Warning! Your code must work even if there are multiple reducers. Recall that reducers do not communicate between each other.

Test your code with 2 reducers (see the `javadoc` of the class `Job`), give the results of each reducer.

Question 6: We want to compute the average of the payments done by category.

3. Modify your code so that it computes the payment average by category.

4. Is it possible to use your `reducer` as `combiner`? Justify your answer.

5. Refer to the book *Data-Intensive Text Processing with MapReduce* for writing a combiner.

N.B. If you need to store value couples, you have to program a class that implements the interface Writable. You can find an example in the `javadoc` of this interface and in the book *Hadoop: the definitve Guide*, p96.

Question 7: We want to know the total of the payments done by the city per month. Modify your code for computing this value.

Question 8: Finally, we want to compute the average of the payments done per category per month. Modify your code for programming this.

N.B. If you need a complex key, follow the approach of question 6 but this time implement the interface `WritableComparable`.