Contents

1 rapport TP2

1.1 the API

1.1.1 org.apache.hadoop.conf.Configuration

class responssible to provide access to configuration parameters from the configuration directory

1.1.2 org.apache.hadoop.conf.Configured

Base class for things that may be configured with a Configuration.

1.1.3 org.apache.hadoop.fs.FileSystem

filesystem object to use the hadoop distributed file system

1.1.4 org.apache.hadoop.fs.Path

names a file or directoy in a FileSystem.

1.1.5 org.apache.hadoop.io.IOUtils

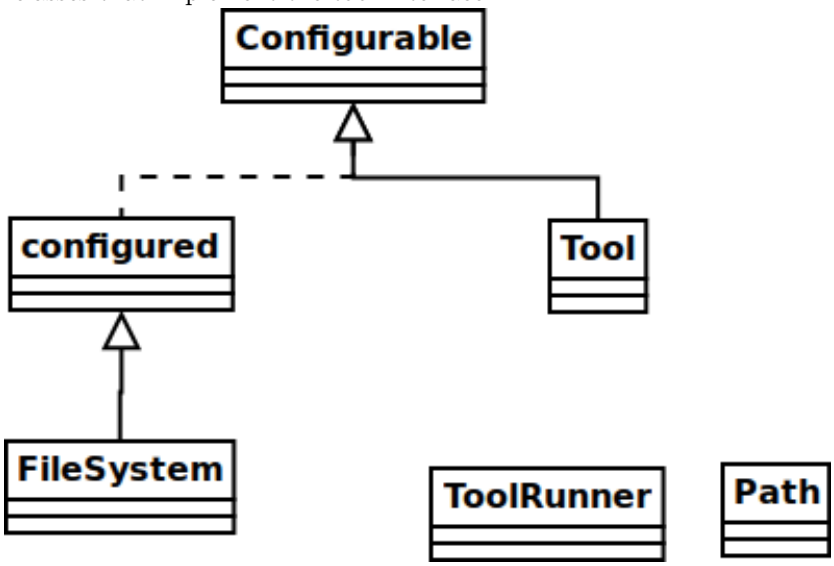
utility class responsible for the I/O related functionality

1.1.6 org.apache.hadoop.util.Tool

an interface to implement the handling of generic command-line options

1.1.7 org.apache.hadoop.util.ToolRunner

run classes that implement the tool interface



1.2 first and second program

as the implementation of this program was given in class and the implementation is really similar we take as best practice for the exercise explain the functionality of each step on the program

1.2.1 the base

the first step is to create a tool and implement the method run which will called on main in conjunction with ToolRunner

1.2.2 the run method

on the run method is where the program lives the following is the steps followed to archive the functionality of the first program

1. create a URI object with the output path (file on the HDFS)
2. normalize the URI (means remove any "." ".." and etc)
3. create a Path object from the normalized URI
4. create a Configuration object and load the actual configuration on it via the command getConf
5. create a FileSystem object specifying the path, the configuration, and the user
6. create the outPutstream on the filesystem
7. create the inputstream usding the local file

8. copy the bytes from the inputstream to the outPutstream
9. close the streams

for the second program the steps are really similar with the diference that on the step 7 and foward has a litle change

1. create a loop for each local file
2. create the inputstream on the local file
3. copy the bytes from the inputstream to outPutstream
4. close the inputstream

and after the loop close the outPutstream

1.3 generation of words

the implementation of this last exercise didn't differ much from the previous ones to archive the goal 3 functions have being created :

```
public static char getRandom(char[] array) {
    int rnd = new Random().nextInt(array.length);
    return array[rnd];
}

public static String randomSyllable(){
    char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
    return  new StringBuilder().append(getRandom(alphabet)).append(getRandom(alphabet)).to
}

public static String randomWord(int size){
    String exampleString = "";
    for(int i=0 ; i < size; i++){
        exampleString += randomSyllable();
    }
    return exampleString + " ";
 }
```

them using the same code from the last exercise and changing the loop to iterate a number of times passed by argument and changing how the inputstream is created :

```
InputStream is = new ByteArrayInputStream(randomWord(10).getBytes(StandardCharsets.UTF_8))
```

that way reading the bytes direct from the string and not a file

1.4 exercise 7 execution time

at this execise we can see the diference in execution time compared to the script made on the exercise 4 the java version of the program has a noticeable faster. that is because the script version stays opening and closing each file descriptor for each file what is a expensive operation but on the java version it keeps the file descriptor opened on the hadoop side until all files are concatenated into this one. what reliefs the cost of the operation.