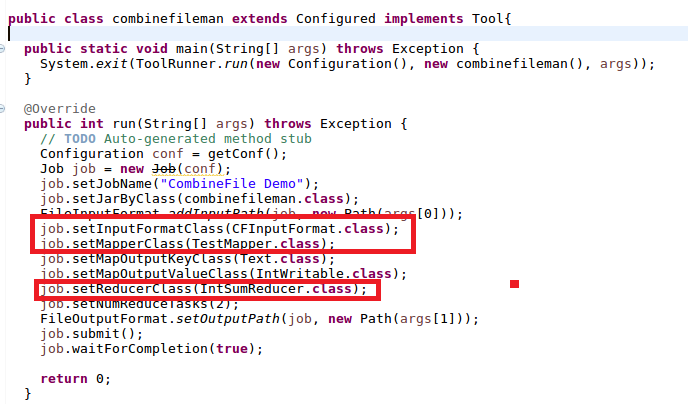
**Write a mapreduce program to use CombineFileInputFormat. Pass a directory containing multiple small files. The program should be able to pick multiple files in a single mapper**

In order to process multiple files using a single mapper we must define a custom combine file Input class which extends a combinefile Input format class where a custom record reader will be used for reading the file

**Driver**

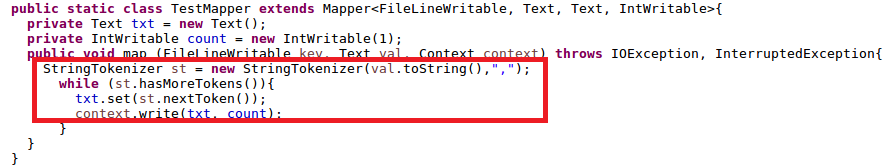
**In driver we set Input format as CFInput format class which is custom class that extends combine file Input format class**

**Mapper class is set and IntNumReducer class is used which performs the count operation**



**Mapper Class**

In Mapper split the input data by comma and sent the word as key and a count of 1 which will added in IntNumReducer class

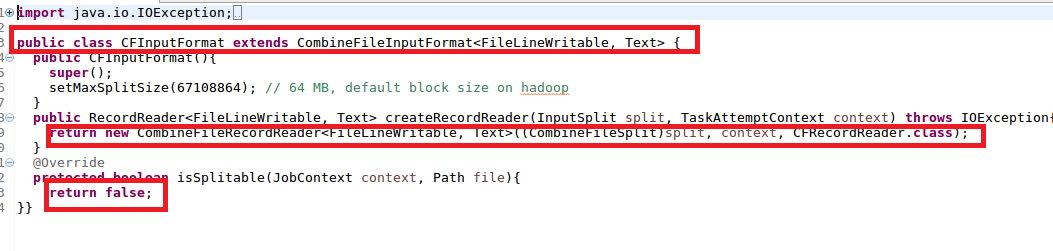


CFInputFormatclass

The CFInputFormat.java doesn’t do much. You implement createRecordReader to pass in the record reader that does the combine file logic, call setMaxSplitSize in the initializer to control the size of each chunk of files;

if you don’t want to split files into half, remember to return false in isSplitable method, which defaults to true.

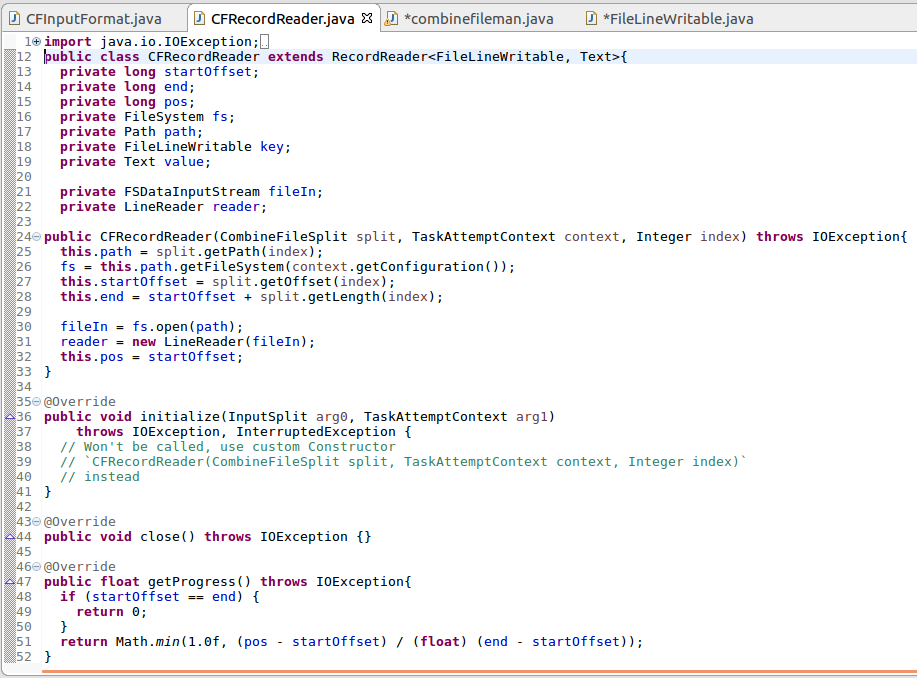
**CF**

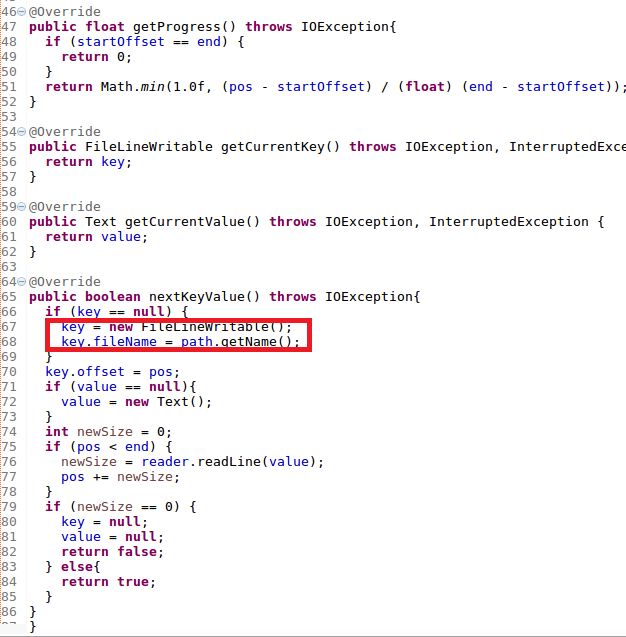
****

**CFRecordReader**

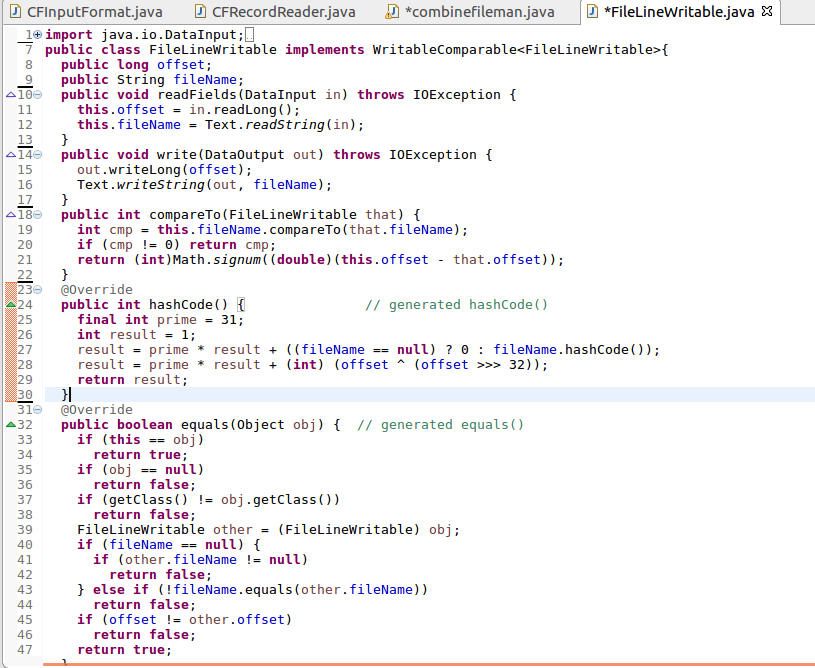
CFRecordReader is a delegate class of CombineFileRecordReader, a built in class that pass each split (typically a whole file in this case) to our class CFRecordReader. When the hadoop job starts, CombineFileRecordReader reads all the file sizes in HDFS that we want it to process, and decides how many splits base on the MaxSplitSize we defined in CFInputFormat. For every split (must be a file, because we set isSplitabe to false), CombineFileRecordReader creates a CFRecrodReader instance via a custom constructor, and pass in CombineFileSplit, context, and index for CFRecordReader to locate the file to process with.

When processing the file, the CFRecordReader creates a FileLineWritable as the key for hadoop mapper class. With each line a FileLineWritable consists the file name and the offset length of that line. The difference between FileLineWritable and the normally used LongWritable in mapper is LongWritable only denote the offset of a line in a file, while FileLineWritable adds the file information into the key.



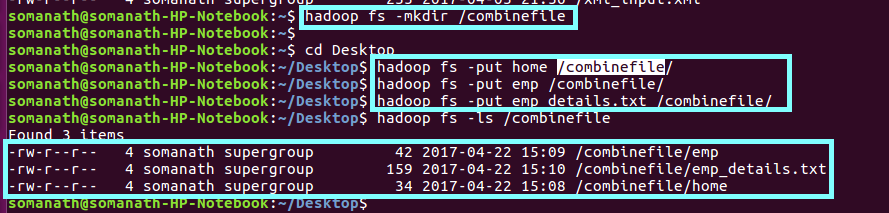


**FileLineWritable Class**

****

**INPUT:**

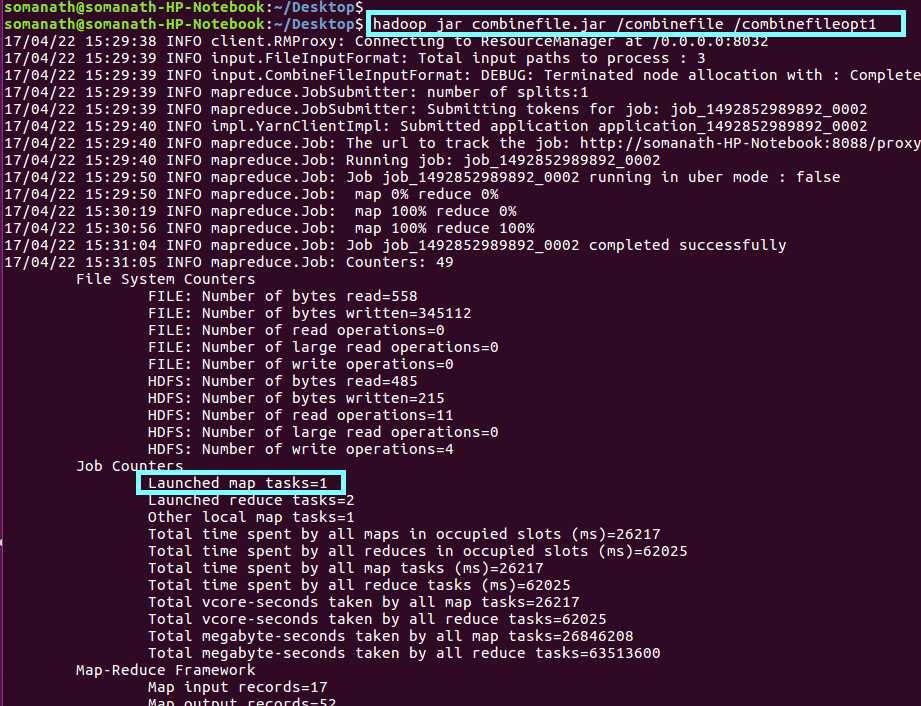
**I created a directory combinefile and added 3 small files as shown**

****

**Running Jar**

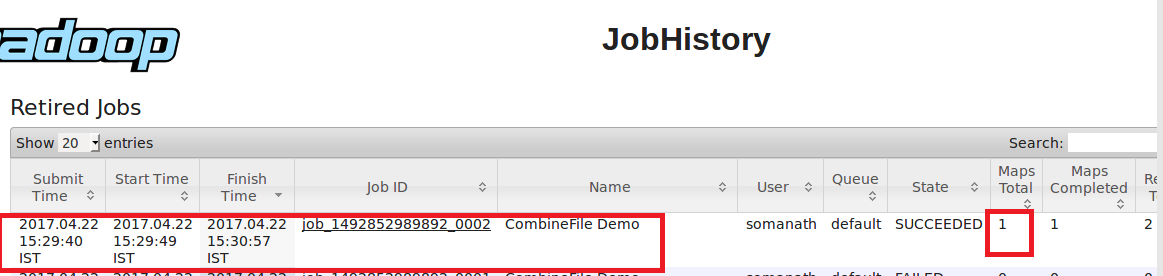
**Running the jar where the directory is given as Input**

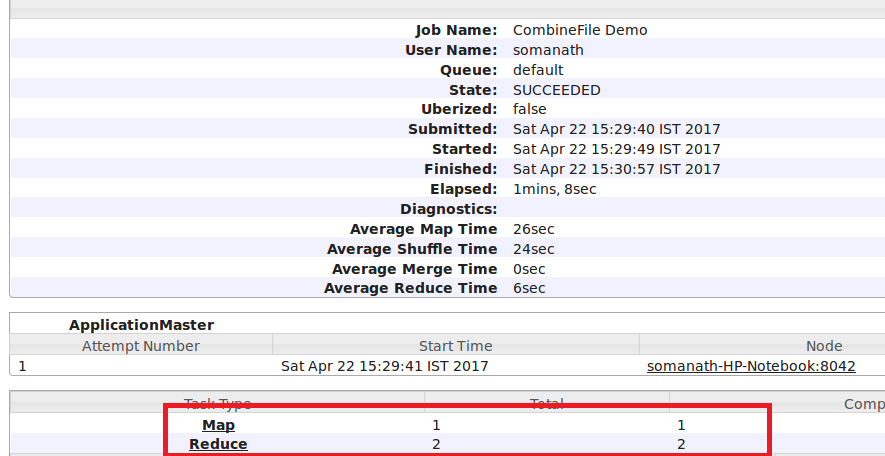
**We can see that the launched map tasks =1 as highlighted**

****

**Output:**

**From job history also we conformed total maps-1 eventhough 3 files are present**

****

****