**Counter:**

1. save as CounterChk.java
2. hdfs dfs -put /volcheck /volcheck

* Uploads the local directory /volcheck to HDFS under the path /volcheck. This directory contains the input data for the MapReduce job.

1. mkdir -m 755 counterchk

* Creates a directory named counterchk with permissions 755 (read, write, execute for the owner; read and execute for others). This directory will store the compiled .class files.

1. javac -classpath $(hadoop classpath) -d counterchk CounterChk.java

* Compiles the CounterChk.java file with Hadoop classpath, outputting .class files into the counterchk directory.

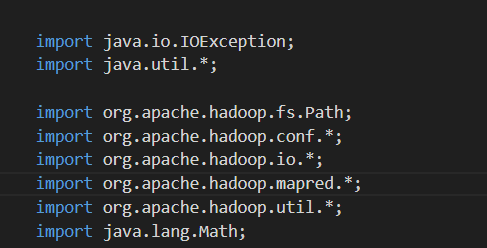
1. jar -cvf ${HOME}/scripts/counterchk.jar -C counterchk/ .

* Creates a JAR file (counterchk.jar) containing the compiled classes from the counterchk directory. This JAR file is needed to run the MapReduce job on Hadoop

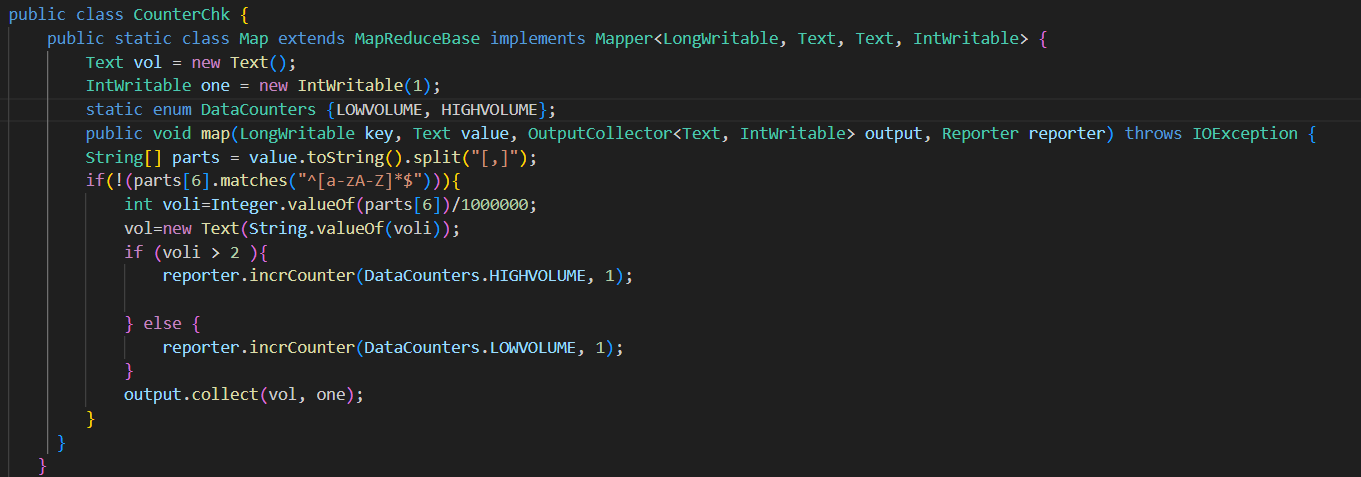
1. hadoop jar ${HOME}/scripts/counterchk.jar CounterChk /volcheck /cntchk.res666

* Executes the MapReduce job using the JAR file created. CounterChk is the class name, /volcheck is the HDFS input path, and /cntchk.res666 is the HDFS output path where results will be stored.

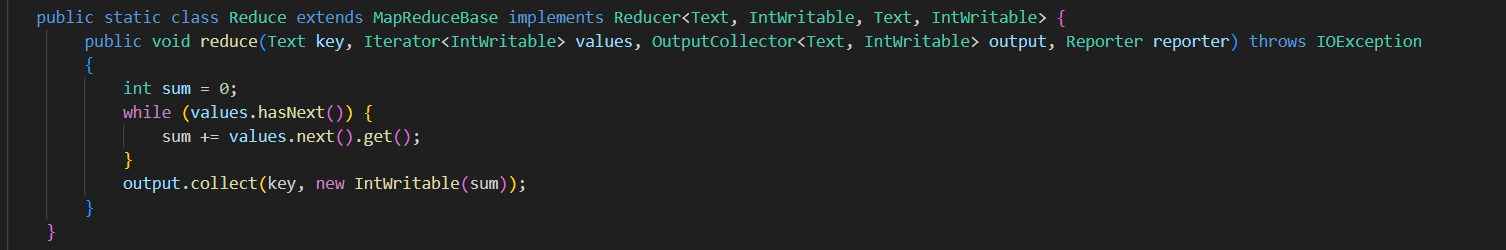
Save as CounterChk.java



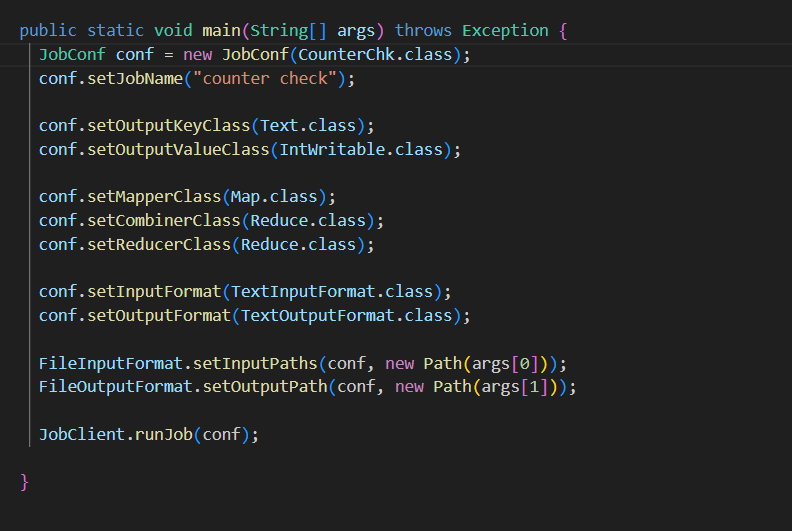
These imports bring in necessary Hadoop classes for MapReduce operations, file handling, and configuration.



* An enum named MyCounters defines custom counters to track occurrences of specific conditions during the MapReduce job.
* Increments TOTAL\_LINES for every processed line. If the line contains "keyword1" or "keyword2", corresponding counters are incremented.
* Output: Emits key-value pairs where the key is the keyword and the value is 1
* user defined variables as enumerators to be used as counters you can have anything you want increment the counter named TOTAL whenever the 7th column in input contains a number more than 2 million
* Increment the enumerator Pass the updated value of enumerator to the Reporter
* Reporter uses it to report



* Aggregates the counts of each keyword received from the mapper.
* Outputs the total count for each keyword



* Sets the job name and specifies the mapper and reducer classes.
* Defines input and output paths.
* Specifies the data types for keys and values for both the map and reduce phases.
* Uses TextInputFormat and TextOutputFormat for reading and writing text files.
* Executes the run method and exits with the appropriate status code
* Defines a MapReduce job to count occurrences of specific keywords using counters and outputs the results.
* Uploads data to HDFS, compiles the code, packages it into a JAR, and runs the MapReduce job on Hadoop.