```
// Adaptación del ejemplo original en http://wiki.apache.org/hadoop/WordCount
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

// Este ejemplo cuenta el número de veces que aparece cada palabra en el archivo de entrada usando MapReduce.
// El código tiene 3 partes: mapper, reduce y programa principal.
public class WordCount {


    // Clase principal con método main que iniciará la ejecución de la tarea
    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf();
        Job job = Job.getInstance(conf);
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);
        // Utilizar en el caso de que exista combinador
        job.setCombinerClass(Clase_del_combinador.class);

        // Declaración de tipos de salida para el mapper
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // Declaración de tipos de salida para el reducer
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Archivos de entrada y directorio de salida
        FileInputFormat.addInputPath(job, new Path( "../texto.txt" ));
        FileOutputFormat.setOutputPath(job, new Path( "salida" ));

        // Aquí podemos elegir el numero de nodos Reduce. Cada reducer genera un fichero  'part-r-XXXXX'
        job.setNumReduceTasks(2);

        // Ejecuta la tarea y espera a que termine.
        // El argumento boolean es para indicar si se quiere información sobre de progreso (verbosity)
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

```
    // El mapper extiende de la interfaz org.apache.hadoop.mapreduce.Mapper. Cuando se ejecuta Hadoop, el mapper
    // recibe cada linea del archivo de entrada como argumento. La función "map" parte cada línea y para
    // cada palabra emite la pareja (word,1) como salida.
    public static 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, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()){
                String word_aux = itr.nextToken();
                word.set( word_aux );
                context.write(word, one);
            }
        }
    }


    // La función "reduce" recibe los valores (apariciones) asociados a la misma clave (palabra) como entrada y
    // produce una pareja con la palabra y el número total de apariciones. como las parejas generadas por la función
    // "map" siempre tienen como valor 1 se podría evitar la suma y devolver directamente el número de elementos.
    public static 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) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

}
```