```
package it.polito.bigdata.hadoop.exercise1;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * Driver class.
 */
public class DriverBigData extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {

    Path inputPath;
    Path outputDir;
    int numberOfReducers;
      int exitCode;

      // Parse the parameters
    numberOfReducers = Integer.parseInt(args[0]);
    inputPath = new Path(args[1]);
    outputDir = new Path(args[2]);

    Configuration conf = this.getConf();

    // The fist job is a word count job
    // It counts the number of occurrences of each movie in watchedmovies.txt

    // Define a new job
    Job job = Job.getInstance(conf);

    // Assign a name to the job
    job.setJobName("Exam 1 - Exercise 1 - count");

    // Set path of the input file/folder (if it is a folder, the job reads all the
files in the specified folder) for this job
    FileInputFormat.addInputPath(job, inputPath);

    // Set path of the output folder for this job
    // A temporary folder
    FileOutputFormat.setOutputPath(job, new Path("temp/"));

    // Specify the class of the Driver for this job
    job.setJarByClass(DriverBigData.class);

    // Set input format
    job.setInputFormatClass(TextInputFormat.class);

    // Set job output format
```

```
    job.setOutputFormatClass(TextOutputFormat.class);

    // Set map class
    job.setMapperClass(Mapper1BigData.class);

    // Set map output key and value classes
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    // Set reduce class
    job.setReducerClass(Reducer1BigData.class);

    // Set reduce output key and value classes
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    // Set number of reducers
    job.setNumReduceTasks(numberOfReducers);

    // Execute the job and wait for completion
    if (job.waitForCompletion(true)==true)
    {
        // The second job select the top 1 movie in terms of number of occurrences
      // the movies in watchedmovies.txt

        // Define a new job
        Job job2 = Job.getInstance(conf);

        // Assign a name to the job
        job2.setJobName("Exam 1 - Exercise 1 - top 1");

        // Set path of the input file/folder (if it is a folder, the job reads all
the files in the specified folder) for this job
        FileInputFormat.addInputPath(job2, new Path("temp/"));

        // Set path of the output folder for this job
        // A temporary folder
        FileOutputFormat.setOutputPath(job2, outputDir);

        // Specify the class of the Driver for this job
        job2.setJarByClass(DriverBigData.class);

        // Set input format
        job2.setInputFormatClass(KeyValueTextInputFormat.class);

        // Set job output format
        job2.setOutputFormatClass(TextOutputFormat.class);

        // Set map class
        job2.setMapperClass(Mapper2BigData.class);

        // Set map output key and value classes
        job2.setMapOutputKeyClass(NullWritable.class);
        job2.setMapOutputValueClass(Text.class);

        // Set reduce class
        job2.setReducerClass(Reducer2BigData.class);

        // Set reduce output key and value classes
```

```
        job2.setOutputKeyClass(NullWritable.class);
        job2.setOutputValueClass(Text.class);

        // Set number of reducers
        // The top-k pattern is characterized by num. reducers = 1
        job2.setNumReduceTasks(1);

        // Execute the job and wait for completion
        if (job2.waitForCompletion(true)==true)
            exitCode=0;
        else
            exitCode=1;
    }
    else
      exitCode=1;

    return exitCode;
  }

  /** Main of the driver
   */

  public static void main(String args[]) throws Exception {
      // Exploit the ToolRunner class to "configure" and run the Hadoop application
    int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);

    System.exit(res);
  }
}
```

```
package it.polito.bigdata.hadoop.exercise1;

import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * Mapper
 */
class Mapper1BigData extends Mapper<
                              LongWritable , // Input key type
                    Text,         // Input value type
                    Text,         // Output key type
                    IntWritable> {// Output value type

    protected void map(
            LongWritable  key,   // Input key type
            Text value,         // Input value type
            Context context) throws IOException, InterruptedException {

            String[] fields=value.toString().split(",");

            // fields[1] = movieid
            // fields[2] = startTime
            // fields[3] = endTime

            // Consider this line if and only if the duration is > 10
            if (BigDataTime.computeDuration(fields[2], fields[3])>10) {
                  // Emit a pair (movieid,1)
                  context.write(new Text(fields[1]), new IntWritable(1));
            }
    }

}
```

```
package it.polito.bigdata.hadoop.exercise1;

import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * Reducer
 */
class Reducer1BigData extends Reducer<
                Text,           // Input key type
                IntWritable,  // Input value type
                Text,           // Output key type
                IntWritable> {  // Output value type



    @Override
    protected void reduce(
        Text key, // Input key type
        Iterable<IntWritable> values, // Input value type
        Context context) throws IOException, InterruptedException {

      // key = movieid
      // values = list of ones associated with the movieid
      // if a combiner is used values can contain also values greater than 1

      // Count the total number of occurrences of the current movie
        // Iterate over the set of values and sum them
      int sum=0;

        for (IntWritable value : values) {
            sum=sum+value.get();
        }

        // Emit pair (movieid, total number of occurrences)
        context.write(key, new IntWritable(sum));
    }
}
```

```
package it.polito.bigdata.hadoop.exercise1;

import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * Mapper
 */
class Mapper2BigData extends Mapper<
                    Text, // Input key type
                    Text,         // Input value type
                    NullWritable, // Output key type
                    Text> {// Output value type

      // Store the record (movied, num occurrences) of the most "frequent" movie
      MovieOccurrences top1;

    protected void setup(Context context)
    {
      top1=null;
    }

    protected void map(
            Text key,   // Input key type
            Text value,         // Input value type
            Context context) throws IOException, InterruptedException {

            // key = movieid
            // value = num occurrencies of movieid

            int occurrences=Integer.parseInt(value.toString());

            // Check if the occurrences of this movie is greater than the current
top
            if (top1==null || top1.occurrences<occurrences)
            {
                  top1=new MovieOccurrences();
                  top1.movieid=new String(key.toString());
                  top1.occurrences=occurrences;
            }
    }

    protected void cleanup(Context context) throws IOException,
InterruptedException
    {
            // Emit the key associated with the top movie
            context.write(NullWritable.get(), new
Text(top1.movieid+"_"+top1.occurrences));
    }

}
```

```
package it.polito.bigdata.hadoop.exercise1;

import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * Reducer
 */
class Reducer2BigData extends Reducer<
                NullWritable,           // Input key type
                Text,  // Input value type
                NullWritable,           // Output key type
                Text> {  // Output value type



      // The reduce method is called only once in this approach
      // All the key-value pairs emitted by the mappers as the
      // same key (NullWritable.get())
    @Override
    protected void reduce(
        NullWritable key, // Input key type
        Iterable<Text> values, // Input value type
        Context context) throws IOException, InterruptedException {

      String movieid;
      int occurrences;

      MovieOccurrences top1;

      top1=null;

        // Iterate over the set of values and select the top 1
        for (Text value : values) {

            String[] record=value.toString().split("_");


            // record[0] = movieid
            // record[1] = num occurrencies of movieid

            movieid=record[0];
            occurrences=Integer.parseInt(record[1]);

            // Check if the occurrences of this movie is greater than the current
top
            if (top1==null || top1.occurrences<occurrences)
            {
                  top1=new MovieOccurrences();
                  top1.movieid=new String(movieid);
                  top1.occurrences=occurrences;
            }
        }

            // Emit the key associated with the top movie
      context.write(NullWritable.get(), new Text(top1.movieid));
    }
```

```
}
```

```
package it.polito.bigdata.hadoop.exercise1;

public class MovieOccurrences {

	String movieid;
	int occurrences;

}
```

```
package it.polito.bigdata.hadoop.exercise1;

public class BigDataTime {

	static public int computeDuration(String startTime, String endTime) {

		//20160606_18:00,20160606_18:34
		String[] fields=startTime.split("_");

		int dates=Integer.parseInt(fields[0]);

		String[] times=fields[1].split(":");

		int hours=Integer.parseInt(times[0]);
		int mins=Integer.parseInt(times[1]);

		String[] fields2=endTime.split("_");

		int datee=Integer.parseInt(fields2[0]);

		String[] timee=fields2[1].split(":");

		int houre=Integer.parseInt(timee[0]);
		int mine=Integer.parseInt(timee[1]);


		int diffMins=(datee-dates)*24*60+(houre-hours)*60+(mine-mins);

		return diffMins;

	}
}
```