```java
import java.io.*;
import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.regex.Pattern;

public class YAAM_test {

    private static List<String> readConfigLines(String filename) throws IOException {
        FileReader fileReader = new FileReader(filename);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        List<String> lines = new LinkedList<>();
        String line = null;
        while ((line = bufferedReader.readLine()) ≠ null) {
            if (¬line.startsWith("#")) {
                lines.add(line);
            }
        }
        bufferedReader.close();
        return lines;
    }

    public static void main(String[] args) throws InterruptedException, IOException {

        // ---------------------- INIT INPUT AND VARIABLES FROM CONFIG ----------------------

        Scanner sc = new Scanner(System.in);
        /*File apacheLogs = new File("test_log.txt");
        Scanner sc = new Scanner(apacheLogs);*/

        String config_general = readConfigLines("config_general").get(0);
        Logger.currentLogLevel = Logger.intToLogLevel(Integer.parseInt(config_general.split(",")[0]));
        int PARSER_POOL_SIZE = Integer.parseInt(config_general.split(",")[1]);

        List<String> config_statistics = readConfigLines("config_statistics");

        List<String> config_statistics_viewer = readConfigLines("config_statistics_viewer");
        String config_statistics_viewer_line1 = config_statistics_viewer.remove(0);
        int statisticsViewWaitMilliseconds = Integer.parseInt(config_statistics_viewer_line1.split(",")[0]);

        List<String> config_loggers = readConfigLines("config_loggers");

        List<String> config_rankings = readConfigLines("config_rankings");
        String config_rankings_line1 = config_rankings.remove(0);
        int rankingTempFileMaxLines = Integer.parseInt(config_rankings_line1.split(",")[0]);
        int rankingTempNumThreads = Integer.parseInt(config_rankings_line1.split(",")[1]);
        int rankingMergerSleepMilliseconds = Integer.parseInt(config_rankings_line1.split(",")[2]);
        int maxErrorsToShow = Integer.parseInt(config_rankings_line1.split(",")[3]);

        // ---------------------- LOAD STATISTICS AND START STATISTICS UPDATER THREADS ----------------------

        List<String> statisticsNames = new LinkedList<>();
        List<Pattern> statisticsRegex = new LinkedList<>();
        HashMap<String, Statistic> statistics = new HashMap<>();
        List<LinkedBlockingQueue> statisticUpdatersQueues = new LinkedList<>();
        List<StatisticUpdater> statisticUpdaters = new LinkedList<>();
        List<Thread> statisticUpdaterThreads = new LinkedList<>();

        /*String[] config_statistics = {"requests,^(.+ .+ \\[.+\\] \".+ .+ .+\" [0-9]{3} .+)$",
                "clients,^(.+) .+ \\[.+\\] \".+ .+ .+\" [0-9]{3} .+$",
                "errors,^.+ .+ \\[(error)\\] \".+ .+ .+\" [0-9]{3} .+$",
                "resources,^.+ .+ \\[.+\\] \".+ (.+) .+\" [0-9]{3} .+$"};*/


        for (int i = 0; i < config_statistics.size(); ++i) {
            String[] splitLine = config_statistics.get(i).split(",");
            statisticsNames.add(splitLine[0]);
            statisticsRegex.add(Pattern.compile(splitLine[1]));

            statistics.put(statisticsNames.get(i), new Statistic(statisticsNames.get(i)));
            statisticUpdatersQueues.add(new LinkedBlockingQueue());
            statisticUpdaters.add(new StatisticUpdater(statisticUpdatersQueues.get(i),
                    statistics.get(statisticsNames.get(i))));
            statisticUpdaterThreads.add(new Thread(statisticUpdaters.get(i)));
            statisticUpdaterThreads.get(i).start();
        }

        // ---------------------- START STATISTICS VIEWER THREAD ----------------------

        StatisticViewer statisticViewer = new StatisticViewer(statistics, statisticsViewWaitMilliseconds);
        Thread statisticsViewerThread = new Thread(statisticViewer);
        statisticsViewerThread.start();

        // ---------------------- START LOGGING THREADS ----------------------

        List<String> loggerNames = new LinkedList<>();
        List<Pattern> loggerRegex = new LinkedList<>();
        List<LinkedBlockingQueue> fileLoggersQueues = new LinkedList<>();
        List<FileLogger> fileLoggers = new LinkedList<>();
        List<Thread> fileLoggersThreads = new LinkedList<>();

        /*String[] config_loggers = {"full_log,^(.+ .+ \\[.+\\] \".+ .+ .+\" [0-9]{3} .+)$",
                                   "error_log,^.+ (.+) \\[error\\] \".+ .+ .+\" [0-9]{3} (.+)$"};*/

        for (int i = 0; i < config_loggers.size(); ++i) {
            String[] splitLine = config_loggers.get(i).split(",");

            loggerNames.add(splitLine[0]);
            loggerRegex.add(Pattern.compile(splitLine[1]));
            fileLoggersQueues.add(new LinkedBlockingQueue());
            fileLoggers.add(new FileLogger(loggerNames.get(i), fileLoggersQueues.get(i)));
            fileLoggersThreads.add(new Thread(fileLoggers.get(i)));
            fileLoggersThreads.get(i).start();
        }

        // ---------------------- START RANKING THREADS ----------------------

        List<String> rankingNames = new LinkedList<>();
        List<Pattern> rankingRegex = new LinkedList<>();
        List<LinkedBlockingQueue> rankingQueues = new LinkedList<>();
        List<List<RankingLogger>> rankingLoggers = new LinkedList<>();
        List<List<Thread>> rankingThreads = new LinkedList<>();
        List<String> finishedLogFiles = new LinkedList<>();

        /*String[] config_rankings = {"errors_ranking,^.+ .+ \\[error\\] \".+ .+ .+\" [0-9]{3} (.+)$"};*/

        for (int i = 0; i < config_rankings.size(); ++i) {
            String[] splitLine = config_rankings.get(i).split(",");

            rankingNames.add(splitLine[0]);
            rankingRegex.add(Pattern.compile(splitLine[1]));
            rankingQueues.add(new LinkedBlockingQueue());
            rankingLoggers.add(new LinkedList<>());
            rankingThreads.add(new LinkedList<>());
            for (int j = 0; j < rankingTempNumThreads; ++j) {
                RankingLogger rankingLogger = new RankingLogger(rankingNames.get
```

```
    (i), rankingQueues.get(i),
126                     finishedLogFiles, rankingTempFileMaxLines);
127                 rankingLoggers.get(i).add(rankingLogger);
128                 Thread rankingLoggerThread = new Thread(rankingLogger);
129                 rankingThreads.get(i).add(rankingLoggerThread);
130                 rankingLoggerThread.start();
131             }
132         }
133
134         // ---------------------- START RANKING MERGE THREADS ------------------
    ----
135
136         List<RankingLoggerMerge> rankingMergers = new LinkedList<>();
137         List<Thread> rankingMergerThreads = new LinkedList<>();
138
139         for (int i = 0; i < config_rankings.size(); ++i) {
140             String[] splitLine = config_rankings.get(i).split(",");
141
142             rankingMergers.add(new RankingLoggerMerge(splitLine[0], finishedLogF
    iles,
143                     rankingMergerSleepMilliseconds, maxErrorsToShow));
144             rankingMergerThreads.add(new Thread(rankingMergers.get(i)));
145             rankingMergerThreads.get(i).start();
146         }
147
148         // ---------------------- START PARSER THREAD POOL ---------------------
    -
149
150         LinkedBlockingQueue analyzerPoolQueue = new LinkedBlockingQueue();
151
152         Parser[] analyzers = new Parser[PARSER_POOL_SIZE];
153         Thread[] analyzerThreads = new Thread[PARSER_POOL_SIZE];
154
155         List<LinkedBlockingQueue> allQueues = new LinkedList<>();
156         allQueues.addAll(statisticUpdatersQueues);
157         allQueues.addAll(fileLoggersQueues);
158         allQueues.addAll(rankingQueues);
159         List<Pattern> allRegex = new LinkedList<>();
160         allRegex.addAll(statisticsRegex);
161         allRegex.addAll(loggerRegex);
162         allRegex.addAll(rankingRegex);
163
164         for (int i = 0; i < PARSER_POOL_SIZE; ++i) {
165             analyzers[i] = new Parser(analyzerPoolQueue, allRegex, allQueues);
166             analyzerThreads[i] = new Thread(analyzers[i]);
167             analyzerThreads[i].start();
168         }
169
170
171         // ---------------------- SHUTDOWN HOOK ----------------------
172
173         Runtime.getRuntime ().addShutdownHook ( new Thread () {
174             @Override
175             public void run () {
176                 System.out.println ( "Shutdown hook" );
177                 Logger.log("main", "Closing everything", Logger.logLevel.INFO);
178
179                 try {
180                     // ---------------------- CLOSE STATISTIC UPADTER THREADS --
    --------------------
181
182                     for (int i = 0; i < statisticUpdaterThreads.size(); ++i) {
183                         statisticUpdaters.get(i).stopKeepAlive();
184                         statisticUpdaterThreads.get(i).interrupt();
185                         statisticUpdaterThreads.get(i).join();
186                     }
187
188                     // ---------------------- CLOSE STATISTIC VIEWER THREAD ----
    ------------------
189
190                     statisticViewer.stopKeepAlive();
191                     statisticsViewerThread.interrupt();
192                     statisticsViewerThread.join();
```



```
193
194                     // ---------------------- CLOSE LOGGER THREADS -------------
    ---------
195
196                     for (int i = 0; i < fileLoggersThreads.size(); ++i) {
197                         fileLoggers.get(i).stopKeepAlive();
198                         fileLoggersThreads.get(i).interrupt();
199                         fileLoggersThreads.get(i).join();
200                     }
201
202                     // ---------------------- CLOSE RANKING THREADS ------------
    ----------
203
204                     for (int i = 0; i < rankingThreads.size(); ++i) {
205                         List<RankingLogger> oneRankLoggers = rankingLoggers.get(
    i);
206                         List<Thread> oneRankLoggerThreads = rankingThreads.get(i
    );
207                         for (int j = 0; j < oneRankLoggers.size(); ++j) {
208                             oneRankLoggers.get(j).stopKeepAlive();
209                             oneRankLoggerThreads.get(j).interrupt();
210                             oneRankLoggerThreads.get(j).join();
211                         }
212                     }
213
214                     // ---------------------- CLOSE RANKING MERGER THREADS -----
    ------------------
215
216                     for (int i = 0; i < rankingMergerThreads.size(); ++i) {
217                         rankingMergers.get(i).stopKeepAlive();
218                         rankingMergerThreads.get(i).interrupt();
219                         rankingMergerThreads.get(i).join();
220                     }
221
222                     // ---------------------- CLOSE PARSER THREADS -------------
    ---------
223
224                     for (int i = 0; i < PARSER_POOL_SIZE; ++i) {
225                         analyzers[i].stopKeepAlive();
226                         analyzerThreads[i].interrupt();
227                         analyzerThreads[i].join();
228                     }
229
230                     Logger.log("main", "All done", Logger.logLevel.INFO);
231                 } catch (InterruptedException e) {
232                     Logger.log("main", "Shutdown hook interrupted", Logger.logLevel.INF
    O);
233                 }
234             }
235         } );
236
237         // ---------------------- MAIN LOOP ----------------------
238
239         boolean endSignal = false;
240         while(¬endSignal ∧ sc.hasNextLine()) {
241             String logLine = sc.nextLine();
242             Logger.log("main", "Recibi de la cola: " + logLine, Logger.logLevel.INFO);
243
244             if (logLine.equals("end")) {
245                 endSignal = true;
246             } else {
247                 analyzerPoolQueue.put(logLine);
248             }
249         }
250     }
251
252 }
```

Mar 24, 18 18:47 **StatisticViewer.java** Page 1/1

```java
import java.util.*;

public class StatisticViewer extends GracefulRunnable {

    private Map<String, Statistic> statistics;
    private int sleepMilliseconds;

    public StatisticViewer(Map<String, Statistic> statistics, int sleepMilliseconds) {
        super("StatisticsViewer");

        this.statistics = statistics;
        this.sleepMilliseconds = sleepMilliseconds;
    }

    @Override
    public void doWork() {

        try {
            Thread.sleep(sleepMilliseconds);

            System.out.println("\n");

            // requests por segundo
            int totalRequests = 0;
            Map<String, Integer> requestStatistics = statistics.get("requests").getStatistic();
            for (String key : requestStatistics.keySet()) {
                totalRequests += requestStatistics.getOrDefault(key, 0);
            }
            float requestsPerSecond = (float)totalRequests / (sleepMilliseconds / 1000);
            System.out.println("[VIEWER] requests per second: " + requestsPerSecond);

            // requests por cliente
            int totalDistinctClients = statistics.get("clients").getStatistic().keySet().size();
            float requestsPerClient = totalDistinctClients > 0 ? (float)totalRequests / totalDistinctClients : 0;
            System.out.println("[VIEWER] requests per client: " + requestsPerClient);

            // cantidad de errores
            int totalErrors = statistics.get("errors").getStatistic().getOrDefault("error", 0);
            System.out.println("[VIEWER] total errors: " + totalErrors);

            // 10 recursos mas pedidos
            LimitedSortedSet<String> topResources = new LimitedSortedSet<>(10, new CountCommaNameComparator());
            Map<String, Integer> resources = statistics.get("resources").getStatistic();
            for (String key : resources.keySet()) {
                topResources.add(resources.get(key) + "," + key);
            }
            System.out.println("[VIEWER] 10 most requested resources: ");
            for (String resource : topResources) {
                System.out.println("\t" + resource);
            }

            System.out.println("\n");

        } catch (InterruptedException e) {
            Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
        }
    }
}
```

Mar 24, 18 18:47 **StatisticUpdater.java** Page 1/1

```java
import java.util.concurrent.LinkedBlockingQueue;

public class StatisticUpdater extends GracefulRunnable {

    private LinkedBlockingQueue inputQueue;
    private Statistic myStatistic;

    public StatisticUpdater(LinkedBlockingQueue queue, Statistic stat) {
        super("StatisticUpdater " + stat.name);

        this.inputQueue = queue;
        this.myStatistic = stat;
    }

    @Override
    protected void doWork() {

        Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);

        try {
            String value = inputQueue.take().toString();
            Logger.log(logName, "Recibi de la cola: " + value, Logger.logLevel.INFO);

            myStatistic.updateStatistic(value);

        } catch (InterruptedException e) {
            Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
        }
    }
}
```

Mar 22, 18 1:20 **Statistic.java** Page 1/1

```java
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Pattern;

public class Statistic {

    //  ---------------------- CLASS VARIABLES ----------------------

    public String name;
    private Map<String, Integer> statisticValues = new HashMap<>();
    private Object statisticLock = new Object();

    public Statistic(String name) {
        this.name = name;
    }


    //  ---------------------- CLASS METHODS ----------------------

    public void updateStatistic(String key) {

        synchronized (statisticLock) {
            statisticValues.merge(key, 1, Integer::sum);
        }

    }

    public Map<String, Integer> getStatistic() {

        synchronized (statisticLock) {
            Map<String, Integer> temp = new HashMap<>(statisticValues);
            statisticValues.clear();
            return temp;
        }

    }
}
```

Mar 26, 18 2:48 **RankingLoggerMerge.java** Page 1/4

```java
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.*;

public class RankingLoggerMerge extends GracefulRunnable {

    private List<String> finishedLogfiles;
    private int sleepMilliseconds;
    private int numErrorsToList;
    private String folderName;

    public RankingLoggerMerge(String name, List<String> finishedLogfiles, int sleepMilliseconds, int numErrorsToList) {
        super("RankingLoggerMerge " + name);

        this.finishedLogfiles = finishedLogfiles;
        this.sleepMilliseconds = sleepMilliseconds;
        this.numErrorsToList = numErrorsToList;
        this.folderName = name;
    }

    // en vez de hacer un merge de todos los archivos generados por los RankingLoggers
    // se usa el ranking anterior (si hay) y los archivos generados luego de ese
    private String getOldRankingFilename() {

        File dir = new File(folderName + "/temp/");
        File[] files = dir.listFiles((d, name) -> name.startsWith("ranking"));
        if (files.length ≤ 0) {
            return "";
        }

        String newest = files[0].getName();
        for (int i = 1; i < files.length; ++i) {
            if (newest.compareTo(files[i].getName()) < 0) {
                newest = files[i].getName();
            }
        }

        Logger.log(logName, "Using old ranking: " + folderName + "/temp/" + newest, Logger.logLevel.INFO);
        return folderName + "/temp/" + newest;
    }

    @Override
    protected void initWork() {
        try {
            if (¬Files.exists(Paths.get(folderName))) {
                Files.createDirectory(Paths.get(folderName));
            }
        } catch (IOException e) {
            Logger.log(logName, "Couldn't create folder: " + folderName, Logger.logLevel.ERROR);
        }

        try {
            if (¬Files.exists(Paths.get(folderName + "/temp"))) {
                Files.createDirectory(Paths.get(folderName + "/temp"));
            }
        } catch (IOException e) {
            Logger.log(logName, "Couldn't create folder: " + folderName + "/temp", Logger.logLevel.ERROR);
        }

        Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
    }

    @Override
    protected void doWork() {

        try {
```

```
69
70              Logger.log(logName, "Going to sleep for " + (sleepMilliseconds / 1000) + "
    seconds",
71                      Logger.logLevel.INFO);
72              Thread.sleep(sleepMilliseconds);
73              Logger.log(logName, "Waking up", Logger.logLevel.INFO);
74
75              // los archivos a mergear se sacan de la lista y despues se borra
76              List<String> currentFinishedLogFiles;
77              synchronized (finishedLogfiles) {
78                  currentFinishedLogFiles = new LinkedList<>(finishedLogfiles);
79                  finishedLogfiles.clear();
80              }
81
82              if (currentFinishedLogFiles.size() > 0) {
83
84                  Logger.log(logName, "Doing work", Logger.logLevel.INFO);
85
86                  // get newest old ranking file
87                  String oldRanking = getOldRankingFilename();
88                  if (oldRanking ≠ "") {
89                      currentFinishedLogFiles.add(oldRanking);
90                  }
91
92                  // output file
93                  String timestamp = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss_SSS
    ").format(new Date());
94                  String outFilename = folderName + "/temp/ranking_" + timestamp;
95                  PrintWriter fileWriter;
96                  try {
97                      fileWriter = new PrintWriter(new FileWriter(outFilename, tru
    e));
98                  } catch (IOException e) {
99                      Logger.log(logName, "Error opening output file: " + outFilename, Logg
    er.logLevel.ERROR);
100                     return;
101                 }
102
103                 // se inician los FileReaders y se lee la primera linea de cada
    archivo
104                 List<BufferedReader> fileReaders = new LinkedList<>();
105                 List<String> lines = new LinkedList<>();
106                 for (int i = 0; i < currentFinishedLogFiles.size(); ++i) {
107                     String filename = currentFinishedLogFiles.get(i);
108                     try {
109                         fileReaders.add(new BufferedReader(new FileReader(filena
    me)));
110                         try {
111                             lines.add(fileReaders.get(i).readLine());
112                         } catch (IOException e) {
113                             Logger.log(logName, "Error reading from file: " + filename,
    Logger.logLevel.ERROR);
114                             fileReaders.remove(i);
115                         }
116                     } catch (FileNotFoundException e) {
117                         Logger.log(logName, "Error opening file: " + filename, Logger.l
    ogLevel.ERROR);
118                     }
119                 }
120
121                 LimitedSortedSet<String> mostFrequentErrors = new LimitedSortedS
    et<>(numErrorsToList,
122                         new CountCommaNameComparator());
123
124                 while (fileReaders.size() > 0 ∧ lines.size() > 0) {
125
126                     // se busca el siguiente error con mas apariciones
127                     // no es un merge comun, porque hay registros de la forma "1
    ,error1", "2,error1"
128                     // en cada archivo no puede aparecer dos veces un error
129                     // entonces se hace un merge acumulando las apariciones de l
    a linea actual en cada archivo
130                     List<Integer> smallestIndexes = new LinkedList<>();
```



```
131                     smallestIndexes.add(0);
132
133                     String errMsg = lines.get(0).split(",")[0];
134                     int errMsgCount = Integer.parseInt(lines.get(0).split(",")[1
    ]);
135                     for (int i = 1; i < lines.size(); ++i) {
136                         int compVal = lines.get(i).split(",")[0].compareTo(errMs
    g);
137                         if (compVal < 0) {
138                             smallestIndexes.clear();
139                             smallestIndexes.add(i);
140
141                             String[] line = lines.get(i).split(",");
142                             errMsg = line[0];
143                             errMsgCount = Integer.parseInt(line[1]);
144                         } else if (compVal ≡ 0) {
145                             smallestIndexes.add(i);
146                             errMsgCount += Integer.parseInt(lines.get(i).split("
    ,")[1]);
147                         }
148                     }
149
150                     fileWriter.println(errMsg + "," + errMsgCount);
151                     mostFrequentErrors.add(errMsgCount + "," + errMsg);
152
153                     // advance used files
154                     for (int i = 0; i < smallestIndexes.size(); ++i) {
155                         int smallestIndex = smallestIndexes.get(i);
156
157                         try {
158                             lines.set(smallestIndex, fileReaders.get(smallestInd
    ex).readLine());
159                         } catch (IOException e) {
160                             Logger.log(logName, "Error reading from file: " + smallestIn
    dex,
161                                     Logger.logLevel.ERROR);
162                             //lines.remove(smallestIndex);
163                             //fileReaders.remove(smallestIndex);
164                         }
165                     }
166
167                     // clear empty files
168                     Iterator<String> itLines = lines.iterator();
169                     Iterator<BufferedReader> itFilesReaders = fileReaders.iterat
    or();
170                     while(itLines.hasNext() ∧ itFilesReaders.hasNext()) {
171                         itFilesReaders.next();
172                         if (itLines.next() ≡ null) {
173                             itLines.remove();
174                             itFilesReaders.remove();
175                         }
176                     }
177                 }
178
179                 fileWriter.close();
180
181                 // most frequent errors output
182                 try {
183                     File ranking = new File(folderName + "/ranking");
184                     if (ranking.exists()) {
185                         ranking.delete();
186                     }
187                     PrintWriter freqErrorsFileWriter = new PrintWriter(
188                             new FileWriter(folderName + "/ranking"));
189                     for (String line : mostFrequentErrors) {
190                         freqErrorsFileWriter.println(line);
191                     }
192                     freqErrorsFileWriter.close();
193                 } catch (IOException e) {
194                     Logger.log(logName, "Error opening output file: " + folderName + "/rank
    ing",
195                             Logger.logLevel.ERROR);
196                 }
```

Mar 26, 18 2:48 **RankingLoggerMerge.java** Page 4/4

```
197
198                 }
199
200             } catch (InterruptedException e) {
201                 Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
202             }
203         }
204     }
```

Mar 26, 18 2:47 **RankingLogger.java** Page 1/2

```
1   import java.io.FileWriter;
2   import java.io.IOException;
3   import java.io.PrintWriter;
4   import java.nio.file.Files;
5   import java.nio.file.Paths;
6   import java.text.Collator;
7   import java.text.SimpleDateFormat;
8   import java.util.*;
9   import java.util.concurrent.LinkedBlockingQueue;
10
11  public class RankingLogger extends GracefulRunnable {
12
13      private LinkedBlockingQueue inputQueue;
14
15      private List<String> finishedLogfiles;
16      private Map<String, Integer> lines = new HashMap<>();
17      private int maxLines;
18      private String folderName;
19
20      public RankingLogger(String name, LinkedBlockingQueue queue, List<String> fi
    nishedLogfiles, int maxLines) {
21
22          super("RankingLogger " + name);
23
24          this.inputQueue = queue;
25          this.maxLines = maxLines;
26          this.finishedLogfiles = finishedLogfiles;
27          this.folderName = name;
28      }
29
30      // se acumulan en memoria hasta cierto numero de errores con su cantidad de
    apariciones
31      // si se pasa ese numero maximo, se hace un dump de los errores ordenados a
    un archivo
32      private void saveLinesToFile() {
33
34          try {
35
36              List<String> stringLines = new LinkedList<>();
37              for (String key : lines.keySet()) {
38                  String line = key + "," + lines.get(key);
39                  stringLines.add(line);
40              }
41              stringLines.sort(new Comparator<String>() {
42                  @Override
43                  public int compare(String o1, String o2) {
44                      return Collator.getInstance().compare(o1, o2);
45                  }
46              });
47
48              String timestamp = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss_SSS").f
    ormat(new Date());
49              String filename = logName.split(" ")[1] + "/temp/" + Thread.currentThr
    ead().getName() +
50                      "_" + timestamp;
51
52              PrintWriter fileWriter = new PrintWriter(new FileWriter(filename, tr
    ue));
53              for (String line : stringLines) {
54                  fileWriter.println(line);
55              }
56              fileWriter.close();
57              lines.clear();
58
59              synchronized (finishedLogfiles) {
60                  finishedLogfiles.add(filename);
61              }
62          } catch (IOException e) {
63              Logger.log("RankingLogger " + logName, e.getMessage(), Logger.logLevel.
    ERROR);
64          }
65      }
66
```

```
67      @Override
68      protected void initWork() {
69          try {
70              if (¬Files.exists(Paths.get(folderName))) {
71                  Files.createDirectory(Paths.get(folderName));
72              }
73          } catch (IOException e) {
74              Logger.log(logName, "Couldn't create folder: " + folderName, Logger.logLevel
    .ERROR);
75          }
76
77          try {
78              if (¬Files.exists(Paths.get(folderName + "/temp"))) {
79                  Files.createDirectory(Paths.get(folderName + "/temp"));
80              }
81          } catch (IOException e) {
82              Logger.log(logName, "Couldn't create folder: " + folderName + "/temp", Logger
    .logLevel.ERROR);
83          }
84
85          Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
86      }
87
88      @Override
89      public void doWork() {
90
91          Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
92
93          try {
94              String line = inputQueue.take().toString();
95              Logger.log(logName, "Recibi de la cola: " + line, Logger.logLevel.INFO);
96
97              if (lines.size() ≡ maxLines ∧ ¬lines.containsKey(line)) {
98                  Logger.log(logName, "Dumping lines to file", Logger.logLevel.INFO);
99                  saveLinesToFile();
100             }
101             lines.merge(line, 1, Integer::sum);
102
103         } catch (InterruptedException e) {
104             Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
105         }
106     }
107 }
```



```
1   import java.util.List;
2   import java.util.concurrent.LinkedBlockingQueue;
3   import java.util.regex.Matcher;
4   import java.util.regex.Pattern;
5
6   public class Parser extends GracefulRunnable {
7
8       private LinkedBlockingQueue inputQueue;
9
10      private List<Pattern> patterns;
11      private List<LinkedBlockingQueue> queues;
12
13      public Parser(LinkedBlockingQueue queue, List<Pattern> patterns, List<Linked
    BlockingQueue> queues) {
14          super("Parser");
15
16          this.inputQueue = queue;
17
18          this.patterns = patterns;
19          this.queues = queues;
20
21          Logger.log("Parser", "Creating object", Logger.logLevel.INFO);
22      }
23
24      @Override
25      protected void doWork() {
26
27          Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
28
29          try {
30              String line = inputQueue.take().toString();
31              Logger.log(logName, "Recibi de la cola: " + line, Logger.logLevel.INFO);
32
33              // pass to queues
34              for (int i = 0; i < patterns.size(); ++i) {
35                  Matcher matchResult = patterns.get(i).matcher(line);
36                  if (matchResult.matches()) {
37                      String resultString = matchResult.group(1);
38                      for (int  j = 2; j ≤ matchResult.groupCount(); ++j) {
39                          resultString += " " + matchResult.group(j);
40                      }
41                      queues.get(i).put(resultString);
42                  }
43              }
44
45          } catch (InterruptedException e) {
46              Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
47          }
48      }
49  }
```

Mar 26, 18 2:47 **Logger.java** Page 1/1

```java
public class Logger {

    public static logLevel currentLogLevel = logLevel.INFO;

    public enum logLevel {
        ERROR,
        WARNING,
        INFO
    }

    public static logLevel intToLogLevel(int i) {
        switch (i) {
            case 0:
                return logLevel.ERROR;
            case 1:
                return logLevel.WARNING;
            case 2:
                return logLevel.INFO;
            default:
                return logLevel.INFO;
        }
    }

    private static String logLevelToString(logLevel level) {
        switch(level) {
            case ERROR:
                return "[ERROR]";
            case WARNING:
                return "[WARNING]";
            case INFO:
                return "[INFO]";
            default:
                return "[INVALID LOGLEVEL]";
        }
    }

    public static void log(String name, String message, logLevel level) {
        if (currentLogLevel.ordinal() ≥ level.ordinal()) {
            System.out.println(Thread.currentThread().getName() + "\t" +
                    logLevelToString(level) + "\t" + name + ":" + message);
        }
    }

}
```

Mar 26, 18 2:10 **LimitedSortedSet.java** Page 1/1

```java
import java.util.Collection;
import java.util.Comparator;
import java.util.TreeSet;

// un sorted set que automaticamente borra elementos de si mismo si se pasa del maximo
class LimitedSortedSet<E> extends TreeSet<E> {

    private int maxSize;

    LimitedSortedSet( int maxSize ) {
        this.maxSize = maxSize;
    }

    LimitedSortedSet( int maxSize, Comparator<? super E> comparator ) {
        super(comparator);
        this.maxSize = maxSize;
    }

    @Override
    public boolean addAll( Collection<? extends E> c ) {
        boolean added = super.addAll( c );
        if( size() > maxSize ) {
            E firstToRemove = (E)toArray( )[maxSize];
            removeAll( tailSet( firstToRemove ) );
        }
        return added;
    }

    @Override
    public boolean add( E o ) {
        boolean added =  super.add( o );
        /*if( size() > maxSize ) {
            E firstToRemove = (E)toArray( )[maxSize];
            removeAll( tailSet( firstToRemove ) );
        }*/
        while (size() > maxSize) {
            remove(last());
        }
        return added;
    }
}
```

Mar 24, 18 18:47 **GracefulRunnable.java** Page 1/1

```java
public abstract class GracefulRunnable implements Runnable {

    private volatile boolean keepAlive = true;
    protected String logName;

    public GracefulRunnable(String name) {
        this.logName = name;

        Logger.log(name, "Creating object", Logger.logLevel.INFO);
    }

    public void stopKeepAlive() {
        keepAlive = false;
    }

    @Override
    public void run() {

        initWork();
        while (keepAlive) {
            doWork();
        }
        endWork();
    }

    protected void initWork() {
        Logger.log(logName,"Starting RUN", Logger.logLevel.INFO);
    }

    protected abstract void doWork();

    protected void endWork() {
        Logger.log(logName,"Ending RUN", Logger.logLevel.INFO);
    }
}
```

Mar 25, 18 3:46 **FileLogger.java** Page 1/1

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.concurrent.LinkedBlockingQueue;

public class FileLogger extends GracefulRunnable {

    private LinkedBlockingQueue inputQueue;

    private String filename;
    private PrintWriter fileWriter;

    public FileLogger(String name, LinkedBlockingQueue queue) {
        super("FileLogger " + name);
        this.inputQueue = queue;
        this.filename = name;
    }

    @Override
    protected void initWork() {

        try {
            if (¬Files.exists(Paths.get(filename))) {
                Files.createDirectory(Paths.get(filename));
            }
            fileWriter = new PrintWriter(new FileWriter(filename + "/" + filenam
e, true));
        } catch (IOException e) {
            Logger.log(logName, "Couldn't create file: " + filename + "/" + filename, Lo
gger.logLevel.ERROR);
            fileWriter = null;
        }

        Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
    }

    @Override
    protected void doWork() {

        Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);

        try {
            String line = inputQueue.take().toString();
            Logger.log(logName, "Recibi de la cola: " + line, Logger.logLevel.INFO);

            fileWriter.println(line);

        } catch (InterruptedException e) {
            Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
        }
    }

    @Override
    protected void endWork(){
        Logger.log(logName, "Ending RUN", Logger.logLevel.INFO);

        fileWriter.close();
    }
}
```

Mar 26, 18 2:10 **CountCommaNameComparator.java** Page 1/1

```java
import java.util.Comparator;


// es un comparador para string de la forma "numero,string" por ejemplo "3,hola"
public class CountCommaNameComparator implements Comparator<String> {
    @Override
    public int compare(String o1, String o2) {

        int o1_int = Integer.parseInt(o1.split(",")[0]);
        String o1_str = o1.split(",")[1];
        int o2_int = Integer.parseInt(o2.split(",")[0]);
        String o2_str = o2.split(",")[1];

        if (o2_int < o1_int) {
            return -1;
        }
        if (o2_int > o1_int) {
            return 1;
        }

        return o2.compareTo(o1);
    }
}
```

Mar 26, 18 2:51 **Table of Content** Page 1/1

Table of Contents

1 *YAAM_test.java*..... sheets 1 to 2 (2) pages 1- 4 253 lines
2 *StatisticViewer.java* sheets 3 to 3 (1) pages 5- 5 59 lines
3 *StatisticUpdater.java* sheets 3 to 3 (1) pages 6- 6 31 lines
4 *Statistic.java*..... sheets 4 to 4 (1) pages 7- 7 38 lines
5 *RankingLoggerMerge.java* sheets 4 to 6 (3) pages 8- 11 205 lines
6 *RankingLogger.java*.. sheets 6 to 7 (2) pages 12- 13 108 lines
7 *Parser.java*..... sheets 7 to 7 (1) pages 14- 14 50 lines
8 *Logger.java*..... sheets 8 to 8 (1) pages 15- 15 45 lines
9 *LimitedSortedSet.java* sheets 8 to 8 (1) pages 16- 16 42 lines
10 *GracefulRunnable.java* sheets 9 to 9 (1) pages 17- 17 36 lines
11 *FileLogger.java*..... sheets 9 to 9 (1) pages 18- 18 60 lines
12 *CountCommaNameComparator.java* sheets 10 to 10 (1) pages 19- 19 24 lines