```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 ----------------------

        Logger.init("YAAM_log.txt");

        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<>();
        Object finishedLogFilesLock = new Object();

        /*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<>());
```

Mar 30, 18 23:29 **YAAM_test.java** Page 3/4

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

Mar 30, 18 23:29 **YAAM_test.java** Page 4/4

```
194                     statisticsViewerThread.interrupt();
195                     statisticsViewerThread.join();
196
197                     // ---------------------- CLOSE LOGGER THREADS -------------
    ---------
198
199                     for (int i = 0; i < fileLoggersThreads.size(); ++i) {
200                         fileLoggers.get(i).stopKeepAlive();
201                         fileLoggersThreads.get(i).interrupt();
202                         fileLoggersThreads.get(i).join();
203                     }
204
205                     // ---------------------- CLOSE RANKING THREADS ------------
    ----------
206
207                     for (int i = 0; i < rankingThreads.size(); ++i) {
208                         List<RankingLogger> oneRankLoggers = rankingLoggers.get(
    i);
209                         List<Thread> oneRankLoggerThreads = rankingThreads.get(i
    );
210                         for (int j = 0; j < oneRankLoggers.size(); ++j) {
211                             oneRankLoggers.get(j).stopKeepAlive();
212                             oneRankLoggerThreads.get(j).interrupt();
213                             oneRankLoggerThreads.get(j).join();
214                         }
215                     }
216
217                     // ---------------------- CLOSE RANKING MERGER THREADS -----
    ------------------
218
219                     for (int i = 0; i < rankingMergerThreads.size(); ++i) {
220                         rankingMergers.get(i).stopKeepAlive();
221                         rankingMergerThreads.get(i).interrupt();
222                         rankingMergerThreads.get(i).join();
223                     }
224
225                     // ---------------------- CLOSE PARSER THREADS -------------
    ---------
226
227                     for (int i = 0; i < PARSER_POOL_SIZE; ++i) {
228                         analyzers[i].stopKeepAlive();
229                         analyzerThreads[i].interrupt();
230                         analyzerThreads[i].join();
231                     }
232
233                     Logger.log("main", "All done", Logger.logLevel.INFO);
234                 } catch (InterruptedException e) {
235                     Logger.log("main", "Shutdown hook interrupted", Logger.logLevel.INF
    O);
236                 }
237
238                 Logger.close();
239             }
240         } );
241
242         // ---------------------- MAIN LOOP ----------------------
243
244         boolean endSignal = false;
245         while(¬endSignal ∧ sc.hasNextLine()) {
246             String logLine = sc.nextLine();
247             Logger.log("main", "Recibi de la cola: " + logLine, Logger.logLevel.INFO);
248
249             if (logLine.equals("end")) {
250                 endSignal = true;
251             } else {
252                 analyzerPoolQueue.put(logLine);
253             }
254         }
255     }
256
257 }
```

Mar 31, 18 2:50 **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);

            Logger.output("\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);
            Logger.output("[VIEWER] requests per second: " + requestsPerSecond);

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

            // cantidad de errores
            int totalErrors = statistics.get("errors").getStatistic().getOrDefault("error", 0);
            Logger.output("[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()) {
                if (shouldStop()) {
                    return;
                }
                topResources.add(resources.get(key) + "," + key);
            }
            Logger.output("[VIEWER] 10 most requested resources: ");
            for (String resource : topResources) {
                Logger.output("\t" + resource);
            }

            Logger.output("\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 31, 18 2:11 **RankingLoggerMerge.java** Page 1/5

```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 String finishedLogfiles;
    private Object finishedLogfilesLock;
    private int sleepMilliseconds;
    private int numErrorsToList;
    private String folderName;

    public RankingLoggerMerge(String name, Object finishedLogfilesLock,
                              int sleepMilliseconds, int numErrorsToList) {
        super("RankingLoggerMerge " + name);

        this.folderName = name;
        this.finishedLogfiles = folderName + "/temp/" +
                "_finished_logfilenames";
        this.finishedLogfilesLock = finishedLogfilesLock;
        this.sleepMilliseconds = sleepMilliseconds;
        this.numErrorsToList = numErrorsToList;
    }

    // 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
```

```
74      protected void doWork() {
75
76          try {
77
78              Logger.log(logName, "Going to sleep for " +
79                              (sleepMilliseconds / 1000) + " seconds",
80                              Logger.logLevel.INFO);
81              Thread.sleep(sleepMilliseconds);
82              Logger.log(logName, "Waking up", Logger.logLevel.INFO);
83
84              // los archivos a mergear se sacan de la lista y despues se borra
85              List<String> currentFinishedLogFiles = new LinkedList<>();
86              synchronized (finishedLogfilesLock) {
87                  FileReader fileReader = null;
88                  try {
89                      fileReader = new FileReader(finishedLogfiles);
90                  BufferedReader bufferedReader = new BufferedReader(fileReader);
91                  String line = null;
92                  while ((line = bufferedReader.readLine()) ≠ null) {
93                      if (¬line.startsWith("#")) {
94                          currentFinishedLogFiles.add(line);
95                      }
96                  }
97                  bufferedReader.close();
98
99                  File deleteFile = new File(finishedLogfiles);
100                 deleteFile.delete();
101                 } catch (FileNotFoundException e) {
102                     Logger.log(logName, "Error loading temp rank " +
103                             "filenames: " + e.getMessage(), Logger.logLevel
104                             .ERROR);
105                     currentFinishedLogFiles.clear();
106                 } catch (IOException e) {
107                     Logger.log(logName, "Error loading temp rank " +
108                             "filenames: " + e.getMessage(), Logger.logLevel
109                             .ERROR);
110                     currentFinishedLogFiles.clear();
111                 }
112             }
113
114             if (currentFinishedLogFiles.size() > 0) {
115
116                 //Logger.log(logName, "Doing work", Logger.logLevel.INFO);
117                 Logger.output("[RANKING LOGGER MERGE " + folderName
118                         + "] Going to work on merging files");
119
120                 // get newest old ranking file
121                 String oldRanking = getOldRankingFilename();
122                 if (oldRanking ≠ "") {
123                     currentFinishedLogFiles.add(oldRanking);
124                 }
125
126                 // output file
127                 String timestamp = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss_SSS
    ").format(new Date());
128                 String outFilename = folderName + "/temp/ranking_" + timestamp;
129                 PrintWriter fileWriter;
130                 try {
131                     fileWriter = new PrintWriter(new FileWriter(outFilename, tru
    e));
132                 } catch (IOException e) {
133                     Logger.log(logName, "Error opening output file: " + outFilename, Logg
    er.logLevel.ERROR);
134                     return;
135                 }
136
137                 // se inician los FileReaders y se lee la primera linea de cada
    archivo
138                 List<BufferedReader> fileReaders = new LinkedList<>();
139                 List<String> lines = new LinkedList<>();
140                 for (int i = 0; i < currentFinishedLogFiles.size(); ++i) {
141                     String filename = currentFinishedLogFiles.get(i);
142                     try {
```



```
143                         fileReaders.add(new BufferedReader(new FileReader(filena
    me)));
144                         try {
145                             lines.add(fileReaders.get(i).readLine());
146                         } catch (IOException e) {
147                             Logger.log(logName, "Error reading from file: " + filename,
    Logger.logLevel.ERROR);
148                             fileReaders.remove(i);
149                         }
150                     } catch (FileNotFoundException e) {
151                         Logger.log(logName, "Error opening file: " + filename, Logger.l
    ogLevel.ERROR);
152                     }
153                 }
154
155                 LimitedSortedSet<String> mostFrequentErrors = new LimitedSortedS
    et<>(numErrorsToList,
156                         new CountCommaNameComparator());
157
158                 while (fileReaders.size() > 0 ∧ lines.size() > 0) {
159
160                     if (shouldStop()) {
161
162                         // close file readers
163                         for (int i = 0; i < fileReaders.size(); ++i) {
164                             try {
165                                 fileReaders.get(i).close();
166                             } catch (IOException e) {
167                                 Logger.log(logName,
168                                         "Error closing file",
169                                         Logger.logLevel.ERROR);
170                             }
171                         }
172
173                         // delete temp output
174                         fileWriter.close();
175                         File deleteOutFile = new File(outFilename);
176                         deleteOutFile.delete();
177
178                         // save filenames back to file for future processing
179                         if (oldRanking ≠ "") {
180                             currentFinishedLogFiles.remove
181                                     (currentFinishedLogFiles.size() - 1);
182                         }
183                         PrintWriter tempFilenamesWriter = null;
184                         try {
185                             tempFilenamesWriter = new PrintWriter(new
186                                     FileWriter(finishedLogfiles, true));
187                         } catch (IOException e) {
188                             Logger.log(logName,
189                                     "Error opening file after" +
190                                             "interrupt",
191                                     Logger.logLevel.ERROR);
192                         }
193                         for (int i = 0; i < currentFinishedLogFiles.size();
194                              ++i) {
195                             tempFilenamesWriter.println
196                                     (currentFinishedLogFiles.get(i));
197                         }
198                         tempFilenamesWriter.close();
199
200                         return;
201                     }
202
203                     // se busca el siguiente error con mas apariciones
204                     // no es un merge comun, porque hay registros de la forma "1
    ,error1", "2,error1"
205                     // en cada archivo no puede aparecer dos veces un error
206                     // entonces se hace un merge acumulando las apariciones de l
    a linea actual en cada archivo
207                     List<Integer> smallestIndexes = new LinkedList<>();
208                     smallestIndexes.add(0);
209
```

```
                        String errMsg = lines.get(0).split(",")[0];
                        int errMsgCount = Integer.parseInt(lines.get(0).split(",")[1
    ]);
                        for (int i = 1; i < lines.size(); ++i) {
                            int compVal = lines.get(i).split(",")[0].compareTo(errMs
    g);
                            if (compVal < 0) {
                                smallestIndexes.clear();
                                smallestIndexes.add(i);

                                String[] line = lines.get(i).split(",");
                                errMsg = line[0];
                                errMsgCount = Integer.parseInt(line[1]);
                            } else if (compVal == 0) {
                                smallestIndexes.add(i);
                                errMsgCount += Integer.parseInt(lines.get(i).split("
    ,")[1]);
                            }
                        }

                        fileWriter.println(errMsg + "," + errMsgCount);
                        mostFrequentErrors.add(errMsgCount + "," + errMsg);

                        // advance used files
                        for (int i = 0; i < smallestIndexes.size(); ++i) {
                            int smallestIndex = smallestIndexes.get(i);

                            try {
                                lines.set(smallestIndex, fileReaders.get(smallestInd
    ex).readLine());
                            } catch (IOException e) {
                                Logger.log(logName, "Error reading from file: " + smallestIn
    dex,
                                        Logger.logLevel.ERROR);
                                //lines.remove(smallestIndex);
                                //fileReaders.remove(smallestIndex);
                            }
                        }

                        // clear empty files
                        Iterator<String> itLines = lines.iterator();
                        Iterator<BufferedReader> itFilesReaders = fileReaders.iterat
    or();
                        while(itLines.hasNext() ∧ itFilesReaders.hasNext()) {
                            BufferedReader fr = itFilesReaders.next();
                            String str = itLines.next();
                            if (str == null) {
                                itLines.remove();
                                try {
                                    fr.close();
                                } catch (IOException e) {
                                    Logger.log(logName,
                                            "Error closing file",
                                            Logger.logLevel.ERROR);
                                }
                                itFilesReaders.remove();
                            }
                        }
                    }

                    fileWriter.close();

                    // most frequent errors output
                    try {
                        File ranking = new File(folderName + "/ranking");
                        if (ranking.exists()) {
                            ranking.delete();
                        }
                        PrintWriter freqErrorsFileWriter = new PrintWriter(
                                new FileWriter(folderName + "/ranking"));
                        for (String line : mostFrequentErrors) {
                            freqErrorsFileWriter.println(line);
                        }
```



```
                        freqErrorsFileWriter.close();
                    } catch (IOException e) {
                        Logger.log(logName, "Error opening output file: "
                                + folderName + "/ranking",
                                Logger.logLevel.ERROR);
                    }

                }

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

Mar 31, 18 2:06 **RankingLogger.java** Page 1/2

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.Collator;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class RankingLogger extends GracefulRunnable {

    private LinkedBlockingQueue inputQueue;

    private Object finishedLogfilesLock;
    private Map<String, Integer> lines = new HashMap<>();
    private int maxLines;
    private String folderName;
    private String finishedLogfiles;

    public RankingLogger(String name, LinkedBlockingQueue queue, Object
            finishedLogfilesLock, int maxLines) {

        super("RankingLogger " + name);

        this.folderName = name;
        this.inputQueue = queue;
        this.maxLines = maxLines;
        this.finishedLogfiles = folderName + "/temp/" +
                "_finished_logfilenames";
        this.finishedLogfilesLock = finishedLogfilesLock;
    }

    // se acumulan en memoria hasta cierto numero de errores con su cantidad de apariciones
    // si se pasa ese numero maximo, se hace un dump de los errores ordenados a un archivo
    private void saveLinesToFile() {

        Logger.log(logName, "Dumping temp rank file", Logger.logLevel.INFO);

        try {

            List<String> stringLines = new LinkedList<>();
            for (String key : lines.keySet()) {
                String line = key + "," + lines.get(key);
                stringLines.add(line);
            }
            stringLines.sort(new Comparator<String>() {
                @Override
                public int compare(String o1, String o2) {
                    return Collator.getInstance().compare(o1, o2);
                }
            });

            String timestamp = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss_SSS").format(new Date());
            String filename = logName.split(" ")[1] + "/temp/" + Thread.currentThread().getName() +
                    "_" + timestamp + "-";
            int sufix = 0;
            File f = new File(filename + sufix);
            while (f.exists()) {
                sufix++;
                f = new File(filename + sufix);
            }
            filename += sufix;

            PrintWriter fileWriter = new PrintWriter(new FileWriter(filename, true));
            for (String line : stringLines) {
                fileWriter.println(line);
            }
            fileWriter.close();
            lines.clear();

            synchronized (finishedLogfilesLock) {
                PrintWriter finishedLogfilesWriter = new PrintWriter(new
                        FileWriter
                        (finishedLogfiles, true));
                finishedLogfilesWriter.println(filename);
                finishedLogfilesWriter.close();
            }
        } catch (IOException e) {
            Logger.log(logName, e.getMessage(), Logger.logLevel.ERROR);
        }
    }

    @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
    public 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);

            if (lines.size() ≡ maxLines ∧ ¬lines.containsKey(line)) {
                Logger.log(logName, "Dumping lines to file", Logger.logLevel.INFO);
                saveLinesToFile();
            }
            lines.merge(line, 1, Integer::sum);

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

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

```java
import java.util.List;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Parser extends GracefulRunnable {

    private LinkedBlockingQueue inputQueue;

    private List<Pattern> patterns;
    private List<LinkedBlockingQueue> queues;

    public Parser(LinkedBlockingQueue queue, List<Pattern> patterns, List<LinkedBlockingQueue> queues) {
        super("Parser");

        this.inputQueue = queue;

        this.patterns = patterns;
        this.queues = queues;

        Logger.log("Parser", "Creating object", 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);

            // pass to queues
            for (int i = 0; i < patterns.size(); ++i) {
                Matcher matchResult = patterns.get(i).matcher(line);
                if (matchResult.matches()) {
                    String resultString = matchResult.group(1);
                    for (int  j = 2; j ≤ matchResult.groupCount(); ++j) {
                        resultString += " " + matchResult.group(j);
                    }
                    queues.get(i).put(resultString);
                }
            }

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

Mar 28, 18 13:18 **Logger.java** Page 1/2

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Logger {

    public static logLevel currentLogLevel = logLevel.INFO;
    private static PrintWriter logWriter = null;

    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 init(String filename) {
        try {
            logWriter = new PrintWriter(new FileWriter(filename));

            String timeStamp = new SimpleDateFormat("yyyy/MM/dd/ HH:mm:ss").format(new Date());
            logWriter.println("***************" + timeStamp + "***************");
        } catch (IOException e) {
            log("Logger", "Couldn't open logfile for writing", logLevel.ERROR);
        }
    }

    public static void close() {
        if (logWriter ≠ null) {
            logWriter.close();
        }
    }

    public static void log(String name, String message, logLevel level) {

        String logLine = Thread.currentThread().getName() + "\t" +
                logLevelToString(level) + "\t" + name + ": " + message;

        // output to screen
        if (currentLogLevel.ordinal() ≥ level.ordinal()) {
            System.out.println(logLine);
        }
        // output to logfile
        if (logWriter ≠ null) {
            logWriter.println(logLine);
```

Mar 28, 18 13:18 **Logger.java** Page 2/2

```
73          }
74      }
75
76      public static void output(String outString) {
77          System.out.println(outString);
78          logWriter.println(outString);
79      }
80
81  }
```

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

```
1   import java.util.Collection;
2   import java.util.Comparator;
3   import java.util.TreeSet;
4
5   // un sorted set que automaticamente borra elementos de si mismo si se pasa del
    maximo
6   class LimitedSortedSet<E> extends TreeSet<E> {
7
8       private int maxSize;
9
10      LimitedSortedSet( int maxSize ) {
11          this.maxSize = maxSize;
12      }
13
14      LimitedSortedSet( int maxSize, Comparator<? super E> comparator ) {
15          super(comparator);
16          this.maxSize = maxSize;
17      }
18
19      @Override
20      public boolean addAll( Collection<? extends E> c ) {
21          boolean added = super.addAll( c );
22          if( size() > maxSize ) {
23              E firstToRemove = (E)toArray( )[maxSize];
24              removeAll( tailSet( firstToRemove ) );
25          }
26          return added;
27      }
28
29      @Override
30      public boolean add( E o ) {
31          boolean added =  super.add( o );
32          /*if( size() > maxSize ) {
33              E firstToRemove = (E)toArray( )[maxSize];
34              removeAll( tailSet( firstToRemove ) );
35          }*/
36          while (size() > maxSize) {
37              remove(last());
38          }
39          return added;
40      }
41  }
```

Mar 28, 18 13:16 **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(); // if doWork is time consuming, call shouldStop periodical
ly
        }
        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);
    }

    protected boolean shouldStop() { return ¬keepAlive; }
}
```

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 31, 18 13:52 **Table of Content** Page 1/1