Apr 08, 18 14:23 **YAAM.java** Page 1/1

```java

import java.io.*;
import java.util.*;

public class YAAM {

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

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

        Logger.init("YAAM_log.txt");

        Scanner sc = new Scanner(System.in);

        // ----------- MONITOR -----------
        Monitor monitor = new Monitor();
        if (¬monitor.loadConfig("config")) {
            return;
        }

        // ----------- STATISTICS -----------
        monitor.createStatistics();

        // ----------- LOGGERS -----------
        monitor.createLoggers();

        // ----------- RANKINGS -----------
        monitor.createRankings();

        // ----------- PARSERS -----------
        monitor.createParsers();


        // ----------- SHUTDOWN HOOK -----------

        Runtime.getRuntime ().addShutdownHook ( new Thread () {
            @Override
            public void run () {
                Logger.output ( "Shutdown hook" );
                Logger.log("main", "Closing everything",
                        Logger.logLevel.INFO);
                try {
                    monitor.stopAll();
                    Logger.log("main", "All done",
                            Logger.logLevel.INFO);
                } catch (InterruptedException e) {
                    Logger.log("main", "Shutdown hook interrupted",
                            Logger.logLevel.INFO);
                }
                Logger.close();
            }
        } );

        // ----------- MAIN LOOP -----------

        monitor.startAll();

        while(sc.hasNextLine()) {
            String logLine = sc.nextLine();
            Logger.log("main", "Recibi de la cola: " + logLine,
                    Logger.logLevel.INFO);
            monitor.processLog(logLine);
        }
    }

}
```

Apr 08, 18 14:03 **StatisticViewer.java** Page 1/1

```java
import java.util.*;

public class StatisticViewer extends GracefulRunnableTask {

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

    public StatisticViewer(List<Statistic> statistics, int period) {
        super("StatisticsViewer", period, period);

        this.statistics = new HashMap<>();
        for (Statistic s : statistics) {
            this.statistics.put(s.name, s);
        }
        this.timeWindow = period;
    }

    @Override
    public void doWork() throws InterruptedException {

        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 / (timeWindow / 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");
    }
}
```

Apr 08, 18 12:25 **StatisticUpdater.java** Page 1/1

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

public class StatisticUpdater extends GracefulRunnableThread {

    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() throws InterruptedException {

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

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

        myStatistic.updateStatistic(value);
    }
}
```

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;
        }

    }
}
```

Apr 08, 18 14:31 **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.*;
import java.util.concurrent.ScheduledExecutorService;

public class RankingLoggerMerge extends GracefulRunnableTask {

    // config variables
    private String finishedLogfiles;
    private Object finishedLogfilesLock;
    //private int timeWindow;
    private int numErrorsToList;
    private String folderName;
    private String tempFolderName;
    private String tempOutputFilenamePrefix;
    private String outputFilename;

    // loop variables
    private String outFilename;
    private PrintWriter outputFileWriter;
    private String oldRanking;
    private List<String> currentFinishedLogFiles;
    private List<BufferedReader> fileReaders;
    private List<String> lines;
    private LimitedSortedSet<String> mostFrequentErrors;

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

        this.folderName = name;
        this.tempFolderName = folderName + "/temp/";
        this.finishedLogfiles = tempFolderName + "_finished_logfilenames";
        this.tempOutputFilenamePrefix = folderName + "/temp/ranking_";
        this.outputFilename = folderName + "/ranking";

        this.finishedLogfilesLock = finishedLogfilesLock;
        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(tempFolderName);
        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: " + tempFolderName
                + newest, Logger.logLevel.INFO);
        return tempFolderName + newest;
    }

    @Override
    protected void initWork() {

        // create work folder
        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.WARNING);
        }

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

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

    @Override
    protected void doWork() throws InterruptedException {

        /*Logger.log(logName, "Going to sleep for " +
                        (timeWindow / 1000) + " seconds",
                        Logger.logLevel.INFO);
        Thread.sleep(timeWindow);*/
        Logger.log(logName, "Waking up", Logger.logLevel.INFO);

        // los archivos a mergear se sacan de la lista y despues se limpia
        currentFinishedLogFiles = finishedLogfileNames();
        if (currentFinishedLogFiles.size() ≤ 0) {
            return;
        }

        Logger.output("[RANKING LOGGER MERGE " + folderName
                + "] Going to work on merging files");

        // get newest old ranking file
        oldRanking = getOldRankingFilename();
        if (oldRanking ≠ "") {
            // si hay un ranking anterior lo voy a usar en el merge
            currentFinishedLogFiles.add(oldRanking);
        }

        // output file
        String timestamp = new SimpleDateFormat(
                "yyyy_MM_dd_HH_mm_ss_SSS").format(new Date());
        outFilename = tempOutputFilenamePrefix + timestamp;
        try {
            outputFileWriter =
                    new PrintWriter(new FileWriter(outFilename, true));
        } catch (IOException e) {
            Logger.log(logName, "Error opening output file: "
                    + outFilename, Logger.logLevel.ERROR);
            return;
        }

        // se inician los FileReaders y se lee
        // la primera linea de cada archivo
        fileReaders = new LinkedList<>();
        lines = new LinkedList<>();
        initFileReadersForMerge();

        // el resultado final del merge es un archivo con todos los errores
        // ordenados alfabeticamente, y otro mas reducido con los N con mas
        // apariciones, ordenado en memoria
        mostFrequentErrors = new LimitedSortedSet<>(numErrorsToList,
                        new CountCommaNameComparator());

        while (fileReaders.size() > 0 ∧ lines.size() > 0) {

            if (shouldStop()) {
                interruptMerge();
                return;
```

```
147             }
148
149             String[] errMsgAndCount = getNextHighestCountError();
150             String errMsg = errMsgAndCount[0];
151             String errCount = errMsgAndCount[1];
152             outputFileWriter.println(errMsg + "," + errCount);
153             mostFrequentErrors.add(errCount + "," + errMsg);
154
155             clearEmptyFiles();
156         }
157
158         // merge alfabetico completado
159         outputFileWriter.close();
160
161         // falta ordenar y guardar a archivo los N con mas apariciones
162         writeReducedRankingToFile();
163     }
164
165     private List<String> finishedLogfileNames() {
166         List<String> currentFinishedLogFiles = new LinkedList<>();
167         synchronized (finishedLogfilesLock) {
168             FileReader fileReader = null;
169             try {
170                 fileReader = new FileReader(finishedLogfiles);
171                 BufferedReader bufferedReader = new BufferedReader(fileReader);
172                 String line = null;
173                 while ((line = bufferedReader.readLine()) ≠ null) {
174                     currentFinishedLogFiles.add(line);
175                 }
176                 bufferedReader.close();
177
178                 File deleteFile = new File(finishedLogfiles);
179                 deleteFile.delete();
180             } catch (FileNotFoundException e) {
181                 Logger.log(logName, "No new temp logfiles ",
182                         Logger.logLevel.INFO);
183                 currentFinishedLogFiles.clear();
184             } catch (IOException e) {
185                 Logger.log(logName, "Error loading temp rank " +
186                         "filenames: " + e.getMessage(), Logger.logLevel
187                         .ERROR);
188                 currentFinishedLogFiles.clear();
189             }
190         }
191         return currentFinishedLogFiles;
192     }
193
194     private void initFileReadersForMerge() {
195         for (int i = 0; i < currentFinishedLogFiles.size(); ++i) {
196             String filename = currentFinishedLogFiles.get(i);
197             try {
198                 fileReaders.add(new BufferedReader(new FileReader(filename)));
199                 try {
200                     lines.add(fileReaders.get(i).readLine());
201                 } catch (IOException e) {
202                     Logger.log(logName, "Error reading from file: " +
203                             filename,  Logger.logLevel.ERROR);
204                     fileReaders.remove(i);
205                 }
206             } catch (FileNotFoundException e) {
207                 Logger.log(logName, "Error opening file: " + filename,
208                         Logger.logLevel.ERROR);
209             }
210         }
211     }
212
213     private void interruptMerge() {
214         // close file readers
215         for (int i = 0; i < fileReaders.size(); ++i) {
216             try {
217                 fileReaders.get(i).close();
218             } catch (IOException e) {
219                 Logger.log(logName,
```



```
220                         "Error closing file",
221                         Logger.logLevel.ERROR);
222             }
223         }
224
225         // delete temp output
226         outputFileWriter.close();
227         File deleteOutFile = new File(outFilename);
228         deleteOutFile.delete();
229
230         // save filenames back to file for future processing
231         if (oldRanking ≠ "") {
232             currentFinishedLogFiles.remove
233                     (currentFinishedLogFiles.size() - 1);
234         }
235         PrintWriter tempFilenamesWriter = null;
236         try {
237             tempFilenamesWriter = new PrintWriter(new
238                     FileWriter(finishedLogfiles, true));
239         } catch (IOException e) {
240             Logger.log(logName,
241                     "Error opening file after" +
242                             "interrupt",
243                     Logger.logLevel.ERROR);
244         }
245         for (int i = 0; i < currentFinishedLogFiles.size();
246              ++i) {
247             tempFilenamesWriter.println
248                     (currentFinishedLogFiles.get(i));
249         }
250         tempFilenamesWriter.close();
251
252         Logger.log(logName, "Finished InterruptMerge", Logger.logLevel.INFO);
253     }
254
255     private String[] getNextHighestCountError() {
256         // se busca el siguiente error con mas apariciones. no es un merge
257         // comun, porque hay registros de la forma "1,error1", "2,error1"
258         // en cada archivo no puede aparecer dos veces un error entonces
259         // se hace un merge acumulando las apariciones de la linea actual
260         // en cada archivo
261         List<Integer> smallestIndexes = new LinkedList<>();
262         smallestIndexes.add(0);
263
264         String errMsg = lines.get(0).split(",")[0];
265         int errMsgCount = Integer.parseInt(lines.get(0).split(",")[1]);
266         for (int i = 1; i < lines.size(); ++i) {
267             int compVal = lines.get(i).split(",")[0].compareTo(errMsg);
268             if (compVal < 0) {
269                 smallestIndexes.clear();
270                 smallestIndexes.add(i);
271
272                 String[] line = lines.get(i).split(",");
273                 errMsg = line[0];
274                 errMsgCount = Integer.parseInt(line[1]);
275             } else if (compVal ≡ 0) {
276                 smallestIndexes.add(i);
277                 errMsgCount += Integer.parseInt(lines.get(i).split(",")[1]);
278             }
279         }
280
281         // advance used files
282         for (int i = 0; i < smallestIndexes.size(); ++i) {
283             int smallestIndex = smallestIndexes.get(i);
284
285             try {
286                 lines.set(smallestIndex,
287                         fileReaders.get(smallestIndex).readLine());
288             } catch (IOException e) {
289                 Logger.log(logName, "Error reading from file: "
290                         + smallestIndex, Logger.logLevel.ERROR);
291             }
292         }
```

Apr 08, 18 14:31 **RankingLoggerMerge.java** Page 5/5

```
293
294         return new String[] {errMsg, Integer.toString(errMsgCount)};
295     }
296
297     private void clearEmptyFiles() {
298         // clear empty files
299         Iterator<String> itLines = lines.iterator();
300         Iterator<BufferedReader> itFilesReaders = fileReaders.iterator();
301         while(itLines.hasNext() ∧ itFilesReaders.hasNext()) {
302             BufferedReader fr = itFilesReaders.next();
303             String str = itLines.next();
304             if (str ≡ null) {
305                 itLines.remove();
306                 try {
307                     fr.close();
308                 } catch (IOException e) {
309                     Logger.log(logName,
310                             "Error closing file",
311                             Logger.logLevel.ERROR);
312                 }
313                 itFilesReaders.remove();
314             }
315         }
316     }
317
318     private void writeReducedRankingToFile() {
319         try {
320             File ranking = new File(outputFilename);
321             if (ranking.exists()) {
322                 ranking.delete();
323             }
324             PrintWriter freqErrorsFileWriter = new PrintWriter(
325                     new FileWriter(outputFilename));
326             for (String line : mostFrequentErrors) {
327                 freqErrorsFileWriter.println(line);
328             }
329             freqErrorsFileWriter.close();
330         } catch (IOException e) {
331             Logger.log(logName, "Error opening output file: "
332                     + outputFilename, Logger.logLevel.ERROR);
333         }
334     }
335 }
```

Apr 08, 18 14:18 **RankingLogger.java** Page 1/3

```
1  import java.io.File;
2  import java.io.FileWriter;
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import java.nio.file.Files;
6  import java.nio.file.Paths;
7  import java.text.Collator;
8  import java.text.SimpleDateFormat;
9  import java.util.*;
10 import java.util.concurrent.LinkedBlockingQueue;
11
12 public class RankingLogger extends GracefulRunnableThread {
13
14     private LinkedBlockingQueue inputQueue;
15
16     private Object finishedLogfilesLock;
17     private Map<String, Integer> lines = new HashMap<>();
18     private int maxLines;
19     private int maxOcurrances;
20     private int currentOcurrances;
21
22     private String folderName;
23     private String folderNameTemp;
24     private String finishedLogfilesList;
25
26     public RankingLogger(String name, LinkedBlockingQueue queue, Object
27             finishedLogfilesLock, int maxLines, int maxOcurrances) {
28
29         super("RankingLogger " + name);
30         this.inputQueue = queue;
31         this.maxLines = maxLines;
32         this.maxOcurrances = maxOcurrances;
33         this.currentOcurrances = 0;
34
35         this.folderName = name;
36         this.folderNameTemp = this.folderName + "/temp/";
37         this.finishedLogfilesList = folderNameTemp + "_finished_logfilenames";
38         this.finishedLogfilesLock = finishedLogfilesLock;
39     }
40
41     // se acumulan en memoria hasta cierto numero de errores con su cantidad
42     // de apariciones si se pasa ese numero maximo, se hace un dump de los
43     // errores ordenados a un archivo
44     private void saveLinesToFile() {
45
46         Logger.log(logName, "Dumping temp rank file",
47                 Logger.logLevel.INFO);
48
49         try {
50
51             List<String> stringLines = sortedLines();
52
53             // write dump file
54             String filename = logDumpFilename();
55             PrintWriter fileWriter = new PrintWriter(new FileWriter(filename,
56                     true));
57             for (String line : stringLines) {
58                 fileWriter.println(line);
59             }
60             fileWriter.close();
61             lines.clear();
62             currentOcurrances = 0;
63
64             // add dump file name to list for processing
65             synchronized (finishedLogfilesLock) {
66                 PrintWriter finishedLogfilesWriter = new PrintWriter(new
67                         FileWriter
68                         (finishedLogfilesList, true));
69                 finishedLogfilesWriter.println(filename);
70                 finishedLogfilesWriter.close();
71             }
72         } catch (IOException e) {
73             Logger.log(logName, e.getMessage(), Logger.logLevel.ERROR);
```

```
74          }
75      }
76
77      private List<String> sortedLines() {
78          List<String> stringLines = new LinkedList<>();
79          for (String key : lines.keySet()) {
80              String line = key + "," + lines.get(key);
81              stringLines.add(line);
82          }
83          stringLines.sort(new Comparator<String>() {
84              @Override
85              public int compare(String o1, String o2) {
86                  return Collator.getInstance().compare(o1, o2);
87              }
88          });
89          return stringLines;
90      }
91
92      private String logDumpFilename() {
93          String timestampPattern = "yyyy_MM_dd_HH_mm_ss_SSS";
94          String timestamp = new SimpleDateFormat(timestampPattern)
95                  .format(new Date());
96          String filename = folderNameTemp + Thread.currentThread().getName()
97                  + "_" + timestamp + "-";
98          int sufix = 0;
99          File f = new File(filename + sufix);
100         while (f.exists()) {
101             sufix++;
102             f = new File(filename + sufix);
103         }
104         filename += sufix;
105         return filename;
106     }
107
108     @Override
109     protected void initWork() {
110         try {
111             if (¬Files.exists(Paths.get(folderName))) {
112                 Files.createDirectory(Paths.get(folderName));
113             }
114         } catch (IOException e) {
115             Logger.log(logName, "Couldn't create folder: " +
116                     folderName, Logger.logLevel.WARNING);
117         }
118
119         try {
120             if (¬Files.exists(Paths.get(folderNameTemp))) {
121                 Files.createDirectory(Paths.get(folderNameTemp));
122             }
123         } catch (IOException e) {
124             Logger.log(logName, "Couldn't create folder: " +
125                     folderNameTemp, Logger.logLevel.WARNING);
126         }
127
128         Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
129     }
130
131     @Override
132     public void doWork() throws InterruptedException {
133
134         Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
135
136         String line = inputQueue.take().toString();
137         Logger.log(logName, "Recibi de la cola: " + line,
138                 Logger.logLevel.INFO);
139
140         if ((lines.size() ≡ maxLines ∧ ¬lines.containsKey(line)) ∨
141                 currentOcurrances ≡ maxOcurrances) {
142             Logger.log(logName, "Dumping lines to file",
143                     Logger.logLevel.INFO);
144             saveLinesToFile();
145         }
146         lines.merge(line, 1, Integer::sum);
```



```
147             currentOcurrances++;
148         }
149     }
```

Apr 08, 18 14:04 **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 GracefulRunnableThread {

    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;
    }

    @Override
    protected void doWork() throws InterruptedException {

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

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

        // por cada patron registrado se matcha la linea de log
        for (int i = 0; i < patterns.size(); ++i) {
            Matcher matchResult = patterns.get(i).matcher(line);
            if (matchResult.matches()) {
                String resultString = matchResult.group(1);
                // en caso de match j = 1 es la linea completa, asi que
                // se ignora. el resto de los campos capturados se concatenan
                // y se envian a traves de la cola correspondiente
                for (int  j = 2; j ≤ matchResult.groupCount(); ++j) {
                    resultString += " " + matchResult.group(j);
                }
                // se envian los campos capturados a la cola correspondiente
                queues.get(i).put(resultString);
            }
        }
    }
}
```

Apr 08, 18 14:39 **Monitor.java** Page 1/3

```java
import com.google.gson.Gson;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.regex.Pattern;

public class Monitor {

    private Config config = null;

    private List<Pattern> patterns = new LinkedList<>();
    private List<LinkedBlockingQueue> queues = new LinkedList<>();
    private LinkedBlockingQueue parsersQueue;

    private List<GracefulRunnable> runnables = new LinkedList<>();

    public boolean loadConfig(String configFilename) {
        try {
            // read json config
            BufferedReader br = new BufferedReader(
                    new FileReader( configFilename));
            String jsonString = "";
            String s;
            while ((s = br.readLine()) ≠ null) {
                jsonString += s;
            }

            // load to object
            config = new Gson().fromJson(jsonString, Config.class);

            // general configs
            Logger.currentLogLevel = Logger.intToLogLevel(config.logLevel);
            GracefulRunnableTask.setPoolSize(config.maxTaskPoolSize);

            return true;
        }
        catch (FileNotFoundException e) {
            Logger.log("Monitor", "config file: " + configFilename +
                    " not found", Logger.logLevel.ERROR);
        } catch (IOException e) {
            Logger.log("Monitor", "config file: " + configFilename +
                    " could not be read", Logger.logLevel.ERROR);
        }

        return false;
    }

    public void createStatistics() {

        // statistic updater threads
        List<Statistic> statistics = new LinkedList<>();
        for (ConfigNameRegex conf : config.config_statistics) {

            String statisticName = conf.name;
            String pattern = conf.regex;

            Statistic statistic = new Statistic(statisticName);
            statistics.add(statistic);
            LinkedBlockingQueue statisticQueue = new LinkedBlockingQueue();
            StatisticUpdater statisticUpdater =
                    new StatisticUpdater(statisticQueue, statistic);

            patterns.add(Pattern.compile(pattern));
            queues.add(statisticQueue);
            runnables.add(statisticUpdater);
        }

        // statistics viewer
```

Apr 08, 18 14:39 **Monitor.java** Page 2/3

```
74          int delay = config.config_statistics_viewer.millisecondsWindow;
75          StatisticViewer viewer = new StatisticViewer(statistics, delay);
76
77          runnables.add(viewer);
78      }
79
80      public void createLoggers() {
81
82          for (ConfigNameRegex conf : config.config_loggers) {
83
84              String name = conf.name;
85              String pattern = conf.regex;
86
87              LinkedBlockingQueue loggerQueue = new LinkedBlockingQueue();
88              FileLogger fileLogger = new FileLogger(name, loggerQueue);
89
90              patterns.add(Pattern.compile(pattern));
91              queues.add(loggerQueue);
92              runnables.add(fileLogger);
93          }
94      }
95
96      public void createRankings() {
97
98          for (ConfigNameRegex conf : config.config_rankings.rankings) {
99
100             String name = conf.name;
101             String pattern = conf.regex;
102
103             // temp threads
104             Object finishedLogFilesLock = new Object();
105             LinkedBlockingQueue rankingLoggerQueue = new LinkedBlockingQueue();
106             for (int j = 0; j < config.config_rankings.numthreadsPerRankingDump;
107                  ++j) {
108                 RankingLogger rankingLogger = new RankingLogger(name,
109                         rankingLoggerQueue, finishedLogFilesLock,
110                         config.config_rankings.linesPerTempFile,
111                         config.config_rankings.ocurrancesPerTempFile);
112
113                 runnables.add(rankingLogger);
114             }
115             patterns.add(Pattern.compile(pattern));
116             queues.add(rankingLoggerQueue);
117
118             // merge thread
119             RankingLoggerMerge rankingMerger = new RankingLoggerMerge(name,
120                     finishedLogFilesLock,
121                     config.config_rankings.rankingMergeSleepMilliseconds,
122                     config.config_rankings.rankingDisplayNum);
123
124             runnables.add(rankingMerger);
125         }
126     }
127
128     // el orden de llamadas es importante, este tiene que llamarse despues de
129     // los demas porque necesita sus colas
130     public void createParsers() {
131
132         parsersQueue = new LinkedBlockingQueue();
133
134         for (int i = 0; i < config.parserNumThreads; ++i) {
135             Parser parser = new Parser(parsersQueue, patterns, queues);
136
137             runnables.add(parser);
138         }
139     }
140
141     public void startAll() {
142         for (GracefulRunnable runnable : runnables) {
143             runnable.start();
144         }
145     }
146
```

Apr 08, 18 14:39 **Monitor.java** Page 3/3

```
147     public void stopAll() throws InterruptedException {
148         for (GracefulRunnable runnable : runnables) {
149             runnable.stop();
150         }
151         for (GracefulRunnable runnable : runnables) {
152             runnable.join();
153         }
154     }
155
156     public void processLog(String log) throws InterruptedException {
157         parsersQueue.put(log);
158     }
159 }
```

Apr 07, 18 11:39 **Logger.java** Page 1/2

```
1  import java.io.FileWriter;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  public class Logger {
8
9      public static logLevel currentLogLevel = logLevel.INFO;
10     private static PrintWriter logWriter = null;
11
12     public enum logLevel {
13         ERROR,
14         WARNING,
15         INFO
16     }
17
18     public static logLevel intToLogLevel(int i) {
19         switch (i) {
20             case 0:
21                 return logLevel.ERROR;
22             case 1:
23                 return logLevel.WARNING;
24             case 2:
25                 return logLevel.INFO;
26             default:
27                 return logLevel.INFO;
28         }
29     }
30
31     private static String logLevelToString(logLevel level) {
32         switch(level) {
33             case ERROR:
34                 return "[ERROR]";
35             case WARNING:
36                 return "[WARNING]";
37             case INFO:
38                 return "[INFO]";
39             default:
40                 return "[INVALID LOGLEVEL]";
41         }
42     }
43
44     public static void init(String filename) {
45         try {
46             logWriter = new PrintWriter(new FileWriter(filename));
47
48             String timeStamp = new SimpleDateFormat(
49                     "yyyy/MM/dd/ HH:mm:ss").format(new Date());
50             logWriter.println("***************" +
51                     timeStamp + "***************");
52         } catch (IOException e) {
53             log("Logger", "Couldn't open logfile for writing",
54                     logLevel.ERROR);
55         }
56     }
57
58     public static void close() {
59         if (logWriter ≠ null) {
60             logWriter.close();
61         }
62     }
63
64     public static void log(String name, String message, logLevel level) {
65
66         String logLine = Thread.currentThread().getName() + "\t" +
67                 logLevelToString(level) + "\t" + name + ": " + message;
68
69         // output to screen
70         if (currentLogLevel.ordinal() ≥ level.ordinal()) {
71             System.out.println(logLine);
72         }
73         // output to logfile
```

Apr 07, 18 11:39 **Logger.java** Page 2/2

```
74         if (logWriter ≠ null) {
75             logWriter.println(logLine);
76         }
77     }
78
79     public static void output(String outString) {
80         System.out.println(outString);
81         logWriter.println(outString);
82     }
83
84 }
```

Apr 07, 18 18:52 **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 );
        while (size() > maxSize) {
            remove(last());
        }
        return added;
    }
}
```

Apr 08, 18 14:39 **GracefulRunnableThread.java** Page 1/1

```java
public abstract class GracefulRunnableThread extends GracefulRunnable {

    Thread thread = new Thread(this);

    public GracefulRunnableThread(String name) {
        super(name);
    }

    @Override
    public void run() {

        initWork();
        while (¬shouldStop()) {
            // if doWork is time consuming, call shouldStop periodically
            // inside doWork
            try {
                doWork();
            } catch (InterruptedException e) {
                Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
            }
        }
        endWork();
    }

    @Override
    public void start() {
        thread.start();
    }

    @Override
    public void stop() {
        stopKeepAlive();
        thread.interrupt();
    }

    @Override
    public void join() throws InterruptedException {
        thread.join();
    }
}
```

Apr 08, 18 14:38 **GracefulRunnableTask.java** Page 1/1

```java
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public abstract class GracefulRunnableTask extends GracefulRunnable {

    private int initialDelay;
    private int period;
    private static ScheduledExecutorService executor = null;

    public GracefulRunnableTask(String name, int initialDelay, int period) {
        super(name);
        this.initialDelay = initialDelay;
        this.period = period;
    }

    public static void setPoolSize(int poolSize) {
        if (executor ≡ null) {
            executor = Executors.newScheduledThreadPool(poolSize);
        } else {
            Logger.log("GracefulRunnableTask", "Task pool size already " +
                    "set", Logger.logLevel.WARNING);
        }
    }

    @Override
    public void run() {

        initWork();
        try {
            // if doWork is time consuming, call shouldStop periodically
            doWork();
        } catch (InterruptedException e) {
            Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
        }
        endWork();
    }

    @Override
    public void start() {

        if (executor ≡ null) {
            executor = Executors.newScheduledThreadPool(1);
            Logger.log("GracefulRunnableTask", "Task started before pool " +
                    "size was set. Size 1 used", Logger.logLevel.WARNING);
        }

        executor.scheduleAtFixedRate(this, initialDelay, period,
                TimeUnit.MILLISECONDS);
    }

    @Override
    public void stop() {
        stopKeepAlive();
        executor.shutdown();
    }

    @Override
    public void join() throws InterruptedException {
        if (¬executor.awaitTermination(10, TimeUnit.SECONDS)) {
            Logger.log(logName, "ScheduledExecutor did not " +
                    "shut down in time", Logger.logLevel.WARNING);
        }
    }
}
```

Apr 08, 18 14:38 **GracefulRunnable.java** Page 1/1

```java
public abstract class GracefulRunnable implements Runnable {

    private volatile boolean keepAlive;
    protected String logName;

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

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

    protected boolean shouldStop() {
        return ¬keepAlive;
    }

    protected void stopKeepAlive() {
        keepAlive = false;
    }

    public abstract void start();

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

    protected abstract void doWork() throws InterruptedException;

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

    public abstract void stop();

    public abstract void join() throws InterruptedException;
}
```

Apr 08, 18 12:25 **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 GracefulRunnableThread {

    private LinkedBlockingQueue inputQueue;

    private String fileName;
    private String folderName;
    private PrintWriter fileWriter;

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

    @Override
    protected void initWork() {

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

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

    @Override
    protected void doWork() throws InterruptedException {

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

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

        fileWriter.println(line);
    }

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

        fileWriter.close();
    }
}
```

Apr 07, 18 11:40 **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);
    }
}
```

Apr 08, 18 2:29 **ConfigStatisticsViewer.java** Page 1/1

```java
import java.util.List;

public class ConfigStatisticsViewer {

    public int millisecondsWindow;

    List<ConfigNameRegex> statistics;
}
```

Apr 08, 18 14:11 **ConfigRankings.java** Page 1/1

```java
import java.util.List;

public class ConfigRankings {

    public int linesPerTempFile;
    public int ocurrancesPerTempFile;
    public int numthreadsPerRankingDump;
    public int rankingMergeSleepMilliseconds;
    public int rankingDisplayNum;

    public List<ConfigNameRegex> rankings;
}
```

Apr 08, 18 2:29 **ConfigNameRegex.java** Page 1/1

```
1  public class ConfigNameRegex {
2
3      public String name;
4      public String regex;
5  }
```

Apr 08, 18 2:35 **ConfigNameOperation.java** Page 1/1

```
1  public class ConfigNameOperation {
2
3      public String name;
4      public String operation;
5  }
```

Apr 08, 18 13:53 **Config.java** Page 1/1

```java
import java.util.List;

public class Config {

    public int logLevel;
    public int parserNumThreads;
    public int maxTaskPoolSize;

    public List<ConfigNameRegex> config_statistics;
    public ConfigStatisticsViewer config_statistics_viewer;
    public List<ConfigNameRegex> config_loggers;
    public ConfigRankings config_rankings;
}
```

Apr 08, 18 2:25 **ConfigGeneral.java** Page 1/1

```java
public class ConfigGeneral {

    public int logLevel;
    public int parserNumThreads;

}
```

Apr 08, 18 14:43 **Table of Content** Page 1/1

Table of Contents