

Apr 08, 18 14:23

YAAM.java

Page 1/1

```

1
2 import java.io.*;
3 import java.util.*;
4
5 public class YAAM {
6
7     public static void main(String[] args) throws InterruptedException {
8
9         // ----- INIT INPUT AND VARIABLES FROM CONFIG -----
10
11         Logger.init("YAAM_log.txt");
12
13         Scanner sc = new Scanner(System.in);
14
15         // ----- MONITOR -----
16         Monitor monitor = new Monitor();
17         if (!monitor.loadConfig("config")) {
18             return;
19         }
20
21         // ----- STATISTICS -----
22         monitor.createStatistics();
23
24         // ----- LOGGERS -----
25         monitor.createLoggers();
26
27         // ----- RANKINGS -----
28         monitor.createRankings();
29
30         // ----- PARSERS -----
31         monitor.createParsers();
32
33
34         // ----- SHUTDOWN HOOK -----
35
36         Runtime.getRuntime().addShutdownHook ( new Thread () {
37             @Override
38             public void run () {
39                 Logger.output ( "Shutdown hook" );
40                 Logger.log("main", "Closing everything",
41                     Logger.logLevel.INFO);
42                 try {
43                     monitor.stopAll();
44                     Logger.log("main", "All done",
45                         Logger.logLevel.INFO);
46                 } catch (InterruptedException e) {
47                     Logger.log("main", "Shutdown hook interrupted",
48                         Logger.logLevel.INFO);
49                 }
50                 Logger.close();
51             }
52         } );
53
54         // ----- MAIN LOOP -----
55
56         monitor.startAll();
57
58         while(sc.hasNextLine()) {
59             String logLine = sc.nextLine();
60             Logger.log("main", "Recibi de la cola: " + logLine,
61                 Logger.logLevel.INFO);
62             monitor.processLog(logLine);
63         }
64     }
65
66 }

```

Apr 08, 18 14:03

StatisticViewer.java

Page 1/1

```

1 import java.util.*;
2
3 public class StatisticViewer extends GracefulRunnableTask {
4
5     private Map<String, Statistic> statistics;
6     private int timeWindow;
7
8     public StatisticViewer(List<Statistic> statistics, int period) {
9         super("StatisticsViewer", period, period);
10
11         this.statistics = new HashMap<>();
12         for (Statistic s : statistics) {
13             this.statistics.put(s.name, s);
14         }
15         this.timeWindow = period;
16     }
17
18     @Override
19     public void doWork() throws InterruptedException {
20
21         Logger.output("\n");
22
23         // requests por segundo
24         int totalRequests = 0;
25         Map<String, Integer> requestStatistics = statistics.get("requests")
26             .getStatistic();
27         for (String key : requestStatistics.keySet()) {
28             totalRequests += requestStatistics.getOrDefault(key, 0);
29         }
30         float requestsPerSecond =
31             (float)totalRequests / (timeWindow / 1000);
32         Logger.output("[VIEWER] requests per second: " +
33             requestsPerSecond);
34
35         // requests por cliente
36         int totalDistinctClients = statistics.get("clients").getStatistic()
37             .keySet().size();
38         float requestsPerClient = totalDistinctClients > 0 ?
39             (float)totalRequests / totalDistinctClients : 0;
40         Logger.output("[VIEWER] requests per client: " +
41             requestsPerClient);
42
43         // cantidad de errores
44         int totalErrors = statistics.get("errors").getStatistic()
45             .getOrDefault("error", 0);
46         Logger.output("[VIEWER] total errors: " + totalErrors);
47
48         // 10 recursos mas pedidos
49         LimitedSortedSet<String> topResources =
50             new LimitedSortedSet<>(10, new CountCommaNameComparator());
51         Map<String, Integer> resources = statistics.get("resources")
52             .getStatistic();
53         for (String key : resources.keySet()) {
54             if (shouldStop()) {
55                 return;
56             }
57             topResources.add(resources.get(key) + "," + key);
58         }
59         Logger.output("[VIEWER] 10 most requested resources: ");
60         for (String resource : topResources) {
61             Logger.output("\t" + resource);
62         }
63
64         Logger.output("\n");
65     }
66 }

```

Apr 08, 18 12:25

StatisticUpdater.java

Page 1/1

```

1  import java.util.concurrent.LinkedBlockingQueue;
2
3  public class StatisticUpdater extends GracefulRunnableThread {
4
5      private LinkedBlockingQueue inputQueue;
6      private Statistic myStatistic;
7
8      public StatisticUpdater(LinkedBlockingQueue queue, Statistic stat) {
9          super("StatisticUpdater " + stat.name);
10
11          this.inputQueue = queue;
12          this.myStatistic = stat;
13      }
14
15      @Override
16      protected void doWork() throws InterruptedException {
17
18          Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
19
20          String value = inputQueue.take().toString();
21          Logger.log(logName, "Recibi de la cola: " + value,
22                  Logger.logLevel.INFO);
23
24          myStatistic.updateStatistic(value);
25      }
26  }

```

Mar 22, 18 1:20

Statistic.java

Page 1/1

```

1  import java.util.HashMap;
2  import java.util.Map;
3  import java.util.regex.Pattern;
4
5  public class Statistic {
6
7      // ----- CLASS VARIABLES -----
8
9      public String name;
10     private Map<String, Integer> statisticValues = new HashMap<>();
11     private Object statisticLock = new Object();
12
13     public Statistic(String name) {
14         this.name = name;
15     }
16
17
18     // ----- CLASS METHODS -----
19
20     public void updateStatistic(String key) {
21
22         synchronized (statisticLock) {
23             statisticValues.merge(key, 1, Integer::sum);
24         }
25     }
26
27     public Map<String, Integer> getStatistic() {
28
29         synchronized (statisticLock) {
30             Map<String, Integer> temp = new HashMap<>(statisticValues);
31             statisticValues.clear();
32             return temp;
33         }
34     }
35
36 }
37

```

Apr 08, 18 14:31

RankingLoggerMerge.java

Page 1/5

```

1  import java.io.*;
2  import java.nio.file.Files;
3  import java.nio.file.Paths;
4  import java.text.SimpleDateFormat;
5  import java.util.*;
6  import java.util.concurrent.ScheduledExecutorService;
7
8  public class RankingLoggerMerge extends GracefulRunnableTask {
9
10     // config variables
11     private String finishedLogfiles;
12     private Object finishedLogfilesLock;
13     //private int timeWindow;
14     private int numErrorsToList;
15     private String folderName;
16     private String tempFolderName;
17     private String tempOutputFilenamePrefix;
18     private String outputFilename;
19
20     // loop variables
21     private String outFilename;
22     private PrintWriter outputFileWriter;
23     private String oldRanking;
24     private List<String> currentFinishedLogFiles;
25     private List<BufferedReader> fileReaders;
26     private List<String> lines;
27     private LimitedSortedSet<String> mostFrequentErrors;
28
29     public RankingLoggerMerge(String name, Object finishedLogfilesLock,
30                             int period, int numErrorsToList) {
31         super("RankingLoggerMerge " + name, period, period);
32
33         this.folderName = name;
34         this.tempFolderName = folderName + "/temp/";
35         this.finishedLogfiles = tempFolderName + "_finished_logfilenames";
36         this.tempOutputFilenamePrefix = folderName + "/temp/ranking_";
37         this.outputFilename = folderName + "/ranking";
38
39         this.finishedLogfilesLock = finishedLogfilesLock;
40         this.numErrorsToList = numErrorsToList;
41     }
42
43     // en vez de hacer un merge de todos los archivos generados por los
44     // RankingLoggers se usa el ranking anterior (si hay) y los archivos
45     // generados luego de ese
46     private String getOldRankingFilename() {
47
48         File dir = new File(tempFolderName);
49         File[] files = dir.listFiles((d, name) → name.startsWith("ranking"));
50         if (files.length ≤ 0) {
51             return "";
52         }
53
54         String newest = files[0].getName();
55         for (int i = 1; i < files.length; ++i) {
56             if (newest.compareTo(files[i].getName()) < 0) {
57                 newest = files[i].getName();
58             }
59         }
60
61         Logger.log(logName, "Using old ranking: " + tempFolderName
62                 + newest, Logger.logLevel.INFO);
63         return tempFolderName + newest;
64     }
65
66     @Override
67     protected void initWork() {
68
69         // create work folder
70         try {
71             if (¬Files.exists(Paths.get(folderName))) {
72                 Files.createDirectory(Paths.get(folderName));
73             }

```

Apr 08, 18 14:31

RankingLoggerMerge.java

Page 2/5

```

74     } catch (IOException e) {
75         Logger.log(logName, "Couldn't create folder: " +
76                 folderName, Logger.logLevel.WARNING);
77     }
78
79     // create temp folder inside work folder
80     try {
81         if (¬Files.exists(Paths.get(tempFolderName))) {
82             Files.createDirectory(Paths.get(tempFolderName));
83         }
84     } catch (IOException e) {
85         Logger.log(logName, "Couldn't create folder: " +
86                 tempFolderName, Logger.logLevel.WARNING);
87     }
88
89     Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
90 }
91
92 @Override
93 protected void doWork() throws InterruptedException {
94
95     /*Logger.log(logName, "Going to sleep for " +
96             (timeWindow / 1000) + " seconds",
97             Logger.logLevel.INFO);
98     Thread.sleep(timeWindow);*/
99     Logger.log(logName, "Waking up", Logger.logLevel.INFO);
100
101     // los archivos a mergear se sacan de la lista y despues se limpia
102     currentFinishedLogFiles = finishedLogfileNames();
103     if (currentFinishedLogFiles.size() ≤ 0) {
104         return;
105     }
106
107     Logger.output("[RANKING LOGGER MERGE " + folderName
108             + "] Going to work on merging files");
109
110     // get newest old ranking file
111     oldRanking = getOldRankingFilename();
112     if (oldRanking ≠ "") {
113         // si hay un ranking anterior lo voy a usar en el merge
114         currentFinishedLogFiles.add(oldRanking);
115     }
116
117     // output file
118     String timestamp = new SimpleDateFormat(
119             "yyyy_MM_dd_HH_mm_ss_SSS").format(new Date());
120     outFilename = tempOutputFilenamePrefix + timestamp;
121     try {
122         outputFileWriter =
123             new PrintWriter(new FileWriter(outFilename, true));
124     } catch (IOException e) {
125         Logger.log(logName, "Error opening output file: "
126                 + outFilename, Logger.logLevel.ERROR);
127         return;
128     }
129
130     // se inician los FileReaders y se lee
131     // la primera linea de cada archivo
132     fileReaders = new LinkedList<>();
133     lines = new LinkedList<>();
134     initFileReadersForMerge();
135
136     // el resultado final del merge es un archivo con todos los errores
137     // ordenados alfabeticamente, y otro mas reducido con los N con mas
138     // apariciones, ordenado en memoria
139     mostFrequentErrors = new LimitedSortedSet<>(numErrorsToList,
140             new CountCommaNameComparator());
141
142     while (fileReaders.size() > 0 ^ lines.size() > 0) {
143
144         if (shouldStop()) {
145             interruptMerge();
146             return;

```

Apr 08, 18 14:31

RankingLoggerMerge.java

Page 3/5

```

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,

```

Apr 08, 18 14:31

RankingLoggerMerge.java

Page 4/5

```

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         return new String[] {errMsg, Integer.toString(errMsgCount)};
294     }
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 maxOccurrences;
20     private int currentOccurrences;
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 maxOccurrences) {
28
29         super("RankingLogger " + name);
30         this.inputQueue = queue;
31         this.maxLines = maxLines;
32         this.maxOccurrences = maxOccurrences;
33         this.currentOccurrences = 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             currentOccurrences = 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);

```

Apr 08, 18 14:18

RankingLogger.java

Page 2/3

```

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);

```

Apr 08, 18 14:18

RankingLogger.java

Page 3/3

```

147         currentOcurrances++;
148     }
149 }

```

Apr 08, 18 14:04

Parser.java

Page 1/1

```

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 GracefulRunnableThread {
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,
14                 List<LinkedBlockingQueue> queues) {
15        super("Parser");
16
17        this.inputQueue = queue;
18
19        this.patterns = patterns;
20        this.queues = queues;
21    }
22
23    @Override
24    protected void doWork() throws InterruptedException {
25
26        Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
27
28        String line = inputQueue.take().toString();
29        Logger.log(logName, "Recibi de la cola: " + line,
30                 Logger.logLevel.INFO);
31
32        // por cada patron registrado se matcha la linea de log
33        for (int i = 0; i < patterns.size(); ++i) {
34            Matcher matchResult = patterns.get(i).matcher(line);
35            if (matchResult.matches()) {
36                String resultString = matchResult.group(1);
37                // en caso de match j = 1 es la linea completa, asi que
38                // se ignora. el resto de los campos capturados se concatenan
39                // y se envian a traves de la cola correspondiente
40                for (int j = 2; j ≤ matchResult.groupCount(); ++j) {
41                    resultString += " " + matchResult.group(j);
42                }
43                // se envian los campos capturados a la cola correspondiente
44                queues.get(i).put(resultString);
45            }
46        }
47    }
48 }

```

Apr 08, 18 14:39

Monitor.java

Page 1/3

```

1 import com.google.gson.Gson;
2
3 import java.io.BufferedReader;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.LinkedList;
8 import java.util.List;
9 import java.util.concurrent.LinkedBlockingQueue;
10 import java.util.regex.Pattern;
11
12 public class Monitor {
13
14     private Config config = null;
15
16     private List<Pattern> patterns = new LinkedList<>();
17     private List<LinkedBlockingQueue> queues = new LinkedList<>();
18     private LinkedBlockingQueue parsersQueue;
19
20     private List<GracefulRunnable> runnables = new LinkedList<>();
21
22     public boolean loadConfig(String configFilename) {
23         try {
24             // read json config
25             BufferedReader br = new BufferedReader(
26                 new FileReader(configFilename));
27             String jsonString = "";
28             String s;
29             while ((s = br.readLine()) ≠ null) {
30                 jsonString += s;
31             }
32
33             // load to object
34             config = new Gson().fromJson(jsonString, Config.class);
35
36             // general configs
37             Logger.currentLogLevel = Logger.intToLogLevel(config.logLevel);
38             GracefulRunnableTask.setPoolSize(config.maxTaskPoolSize);
39
40             return true;
41         }
42         catch (FileNotFoundException e) {
43             Logger.log("Monitor", "config file: " + configFilename +
44                 " not found", Logger.logLevel.ERROR);
45         }
46         catch (IOException e) {
47             Logger.log("Monitor", "config file: " + configFilename +
48                 " could not be read", Logger.logLevel.ERROR);
49         }
50
51         return false;
52     }
53
54     public void createStatistics() {
55
56         // statistic updater threads
57         List<Statistic> statistics = new LinkedList<>();
58         for (ConfigNameRegex conf : config.config_statistics) {
59
60             String statisticName = conf.name;
61             String pattern = conf.regex;
62
63             Statistic statistic = new Statistic(statisticName);
64             statistics.add(statistic);
65             LinkedBlockingQueue statisticQueue = new LinkedBlockingQueue();
66             StatisticUpdater statisticUpdater =
67                 new StatisticUpdater(statisticQueue, statistic);
68
69             patterns.add(Pattern.compile(pattern));
70             queues.add(statisticQueue);
71             runnables.add(statisticUpdater);
72         }
73
74         // 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.ocurrencesPerTempFile);
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

```

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

```

Apr 08, 18 14:39

GracefulRunnableThread.java

Page 1/1

```

1 public abstract class GracefulRunnableThread extends GracefulRunnable {
2
3     Thread thread = new Thread(this);
4
5     public GracefulRunnableThread(String name) {
6         super(name);
7     }
8
9     @Override
10    public void run() {
11
12        initWork();
13        while (!shouldStop()) {
14            // if doWork is time consuming, call shouldStop periodically
15            // inside doWork
16            try {
17                doWork();
18            } catch (InterruptedException e) {
19                Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
20            }
21        }
22        endWork();
23    }
24
25    @Override
26    public void start() {
27        thread.start();
28    }
29
30    @Override
31    public void stop() {
32        stopKeepAlive();
33        thread.interrupt();
34    }
35
36    @Override
37    public void join() throws InterruptedException {
38        thread.join();
39    }
40 }

```

Apr 08, 18 14:38

GracefulRunnableTask.java

Page 1/1

```

1 import java.util.concurrent.Executors;
2 import java.util.concurrent.ScheduledExecutorService;
3 import java.util.concurrent.TimeUnit;
4
5 public abstract class GracefulRunnableTask extends GracefulRunnable {
6
7     private int initialDelay;
8     private int period;
9     private static ScheduledExecutorService executor = null;
10
11     public GracefulRunnableTask(String name, int initialDelay, int period) {
12         super(name);
13         this.initialDelay = initialDelay;
14         this.period = period;
15     }
16
17     public static void setPoolSize(int poolSize) {
18         if (executor == null) {
19             executor = Executors.newScheduledThreadPool(poolSize);
20         } else {
21             Logger.log("GracefulRunnableTask", "Task pool size already " +
22                 "set", Logger.logLevel.WARNING);
23         }
24     }
25
26     @Override
27     public void run() {
28         initWork();
29         try {
30             // if doWork is time consuming, call shouldStop periodically
31             doWork();
32         } catch (InterruptedException e) {
33             Logger.log(logName, "I was interrupted", Logger.logLevel.INFO);
34         }
35         endWork();
36     }
37
38     @Override
39     public void start() {
40
41         if (executor == null) {
42             executor = Executors.newScheduledThreadPool(1);
43             Logger.log("GracefulRunnableTask", "Task started before pool " +
44                 "size was set. Size 1 used", Logger.logLevel.WARNING);
45         }
46
47         executor.scheduleAtFixedRate(this, initialDelay, period,
48             TimeUnit.MILLISECONDS);
49     }
50
51     @Override
52     public void stop() {
53         stopKeepAlive();
54         executor.shutdown();
55     }
56
57     @Override
58     public void join() throws InterruptedException {
59         if (!executor.awaitTermination(10, TimeUnit.SECONDS)) {
60             Logger.log(logName, "ScheduledExecutor did not " +
61                 "shut down in time", Logger.logLevel.WARNING);
62         }
63     }
64 }
65

```

Apr 08, 18 14:38

GracefulRunnable.java

Page 1/1

```

1 public abstract class GracefulRunnable implements Runnable {
2
3     private volatile boolean keepAlive;
4     protected String logName;
5
6     public GracefulRunnable(String name) {
7         this.logName = name;
8         this.keepAlive = true;
9
10         Logger.log(name, "Creating object", Logger.logLevel.INFO);
11     }
12
13     protected boolean shouldStop() {
14         return !keepAlive;
15     }
16
17     protected void stopKeepAlive() {
18         keepAlive = false;
19     }
20
21     public abstract void start();
22
23     protected void initWork() {
24         Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
25     }
26
27     protected abstract void doWork() throws InterruptedException;
28
29     protected void endWork() {
30         Logger.log(logName, "Ending RUN", Logger.logLevel.INFO);
31     }
32
33     public abstract void stop();
34
35     public abstract void join() throws InterruptedException;
36 }

```

Apr 08, 18 12:25

FileLogger.java

Page 1/1

```

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.util.concurrent.LinkedBlockingQueue;
7
8 public class FileLogger extends GracefulRunnableThread {
9
10     private LinkedBlockingQueue inputQueue;
11
12     private String fileName;
13     private String folderName;
14     private PrintWriter fileWriter;
15
16     public FileLogger(String name, LinkedBlockingQueue queue) {
17         super("FileLogger " + name);
18         this.inputQueue = queue;
19         this.folderName = name;
20         this.fileName = this.folderName + "/" + name;
21     }
22
23     @Override
24     protected void initWork() {
25
26         try {
27             if (!Files.exists(Paths.get(folderName))) {
28                 Files.createDirectory(Paths.get(folderName));
29             }
30             fileWriter = new PrintWriter(new FileWriter(fileName,
31                 true));
32         } catch (IOException e) {
33             Logger.log(logName, "Couldn't create file: " + fileName,
34                 Logger.logLevel.ERROR);
35             fileWriter = null;
36         }
37
38         Logger.log(logName, "Starting RUN", Logger.logLevel.INFO);
39     }
40
41     @Override
42     protected void doWork() throws InterruptedException {
43
44         Logger.log(logName, "Waiting for input", Logger.logLevel.INFO);
45
46         String line = inputQueue.take().toString();
47         Logger.log(logName, "Recibi de la cola: " + line,
48             Logger.logLevel.INFO);
49
50         fileWriter.println(line);
51     }
52
53     @Override
54     protected void endWork() {
55         Logger.log(logName, "Ending RUN", Logger.logLevel.INFO);
56
57         fileWriter.close();
58     }
59 }

```

Apr 07, 18 11:40

CountCommaNameComparator.java

Page 1/1

```

1 import java.util.Comparator;
2
3
4 // es un comparador para string de la forma "numero,string"
5 // por ejemplo "3,hola"
6 public class CountCommaNameComparator implements Comparator<String> {
7     @Override
8     public int compare(String o1, String o2) {
9
10         int o1_int = Integer.parseInt(o1.split(",")[0]);
11         String o1_str = o1.split(",")[1];
12         int o2_int = Integer.parseInt(o2.split(",")[0]);
13         String o2_str = o2.split(",")[1];
14
15         if (o2_int < o1_int) {
16             return -1;
17         }
18         if (o2_int > o1_int) {
19             return 1;
20         }
21
22         return o2.compareTo(o1);
23     }
24 }

```

Apr 08, 18 2:29

ConfigStatisticsViewer.java

Page 1/1

```
1 import java.util.List;
2
3 public class ConfigStatisticsViewer {
4
5     public int millisecondsWindow;
6
7     List<ConfigNameRegex> statistics;
8 }
```

Apr 08, 18 14:11

ConfigRankings.java

Page 1/1

```
1 import java.util.List;
2
3 public class ConfigRankings {
4
5     public int linesPerTempFile;
6     public int occurrencesPerTempFile;
7     public int numthreadsPerRankingDump;
8     public int rankingMergeSleepMilliseconds;
9     public int rankingDisplayNum;
10
11     public List<ConfigNameRegex> rankings;
12 }
```

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

```
1 import java.util.List;
2
3 public class Config {
4
5     public int logLevel;
6     public int parserNumThreads;
7     public int maxTaskPoolSize;
8
9     public List<ConfigNameRegex> config_statistics;
10    public ConfigStatisticsViewer config_statistics_viewer;
11    public List<ConfigNameRegex> config_loggers;
12    public ConfigRankings config_rankings;
13 }
```

Apr 08, 18 2:25

ConfigGeneral.java

Page 1/1

```
1 public class ConfigGeneral {
2
3     public int logLevel;
4     public int parserNumThreads;
5
6 }
```

Apr 08, 18 14:53

Table of Content

Page 1/1

1	Table of Contents			
2	1 YAAM.java.....	sheets 1 to 1 (1)	pages 1- 1	67 lines
3	2 StatisticViewer.java	sheets 1 to 1 (1)	pages 2- 2	67 lines
4	3 StatisticUpdater.java	sheets 2 to 2 (1)	pages 3- 3	27 lines
5	4 Statistic.java.....	sheets 2 to 2 (1)	pages 4- 4	38 lines
6	5 RankingLoggerMerge.java	sheets 3 to 5 (3)	pages 5- 9	336 lines
7	6 RankingLogger.java..	sheets 5 to 6 (2)	pages 10- 12	150 lines
8	7 Parser.java.....	sheets 7 to 7 (1)	pages 13- 13	49 lines
9	8 Monitor.java.....	sheets 7 to 8 (2)	pages 14- 16	160 lines
10	9 Logger.java.....	sheets 9 to 9 (1)	pages 17- 18	85 lines
11	10 LimitedSortedSet.java	sheets 10 to 10 (1)	pages 19- 19	39 lines
12	11 GracefulRunnableThread.java	sheets 10 to 10 (1)	pages 20- 20	41 lines
13	12 GracefulRunnableTask.java	sheets 11 to 11 (1)	pages 21- 21	66 lines
14	13 GracefulRunnable.java	sheets 11 to 11 (1)	pages 22- 22	37 lines
15	14 FileLogger.java.....	sheets 12 to 12 (1)	pages 23- 23	60 lines
16	15 CountCommaNameComparator.java	sheets 12 to 12 (1)	pages 24- 24	25 lines
17	16 ConfigStatisticsViewer.java	sheets 13 to 13 (1)	pages 25- 25	9 lines
18	17 ConfigRankings.java.	sheets 13 to 13 (1)	pages 26- 26	13 lines
19	18 ConfigNameRegex.java	sheets 14 to 14 (1)	pages 27- 27	6 lines
20	19 ConfigNameOperation.java	sheets 14 to 14 (1)	pages 28- 28	6 lines
21	20 Config.java.....	sheets 15 to 15 (1)	pages 29- 29	14 lines
22	21 ConfigGeneral.java..	sheets 15 to 15 (1)	pages 30- 30	7 lines