abr 09, 18 3:41 **ApacheLogEntry.java** Page 1/2

```java
package Wrapper;

import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class ApacheLogEntry {
    private HashMap<String, String> lineParts;

    private static String getAccessLogRegex() {
        String rClientAddr = "^([\\d.\\S]+)"; // Client address
        String rSep1 = " (\\S+)"; // -
        String rSep2 = " (\\S+)"; // -
        String rDate = " \\[([\\w:/]+\\s[+\\-]\\d{4})\\]"; // Date
        String rMethodAndURL = " \"([A-Z]+) ([^ ]+?) (.+?)\""; // request method, url, http version
        String rHTTPCODE = " (\\d{3})"; // HTTP code
        String rBytes = " (\\d+|(.+?))"; // Number of bytes
        String rReferer = "( \"([^\"]+|(.+?))\")?"; // Referer
        String rAgent = "( \"([^\"]+|(.+?))\")?"; // Agent

        return rClientAddr + rSep1 + rSep2 + rDate + rMethodAndURL + rHTTPCODE + rBytes + rReferer + rAgent;
    }

    private static String getErrorLogRegex() {
        String rDate = "^(\\[[^\\]]+\\])"; // Date
        String rType = "( \\[(\\S+)\\])?"; // Type
        String rClient = "( \\[(client (\\S+))\\])?"; //Client
        String rErrorMessage = " (.*)?$"; //Error message
        return rDate + rType + rClient + rErrorMessage;
    }

    private static final Pattern accessLogPattern = Pattern.compile(
            getAccessLogRegex(),Pattern.CASE_INSENSITIVE | Pattern.DOTALL
    );

    private static final Pattern errorLogPattern = Pattern.compile(
            getErrorLogRegex(),Pattern.CASE_INSENSITIVE | Pattern.DOTALL
    );

    private ApacheLogEntry(String line) {
        lineParts = new HashMap<>();
        lineParts.put("_line",line);
        parseLine();
    }

    public ApacheLogEntry(ApacheLogEntry apacheLogEntry) {
        this.lineParts = new HashMap<>(apacheLogEntry.lineParts);
    }

    public static ApacheLogEntry from(String line) {
        return isApacheLogEntry(line) ? new ApacheLogEntry(line) : null;
    }

    private static boolean isApacheLogEntry(String entry) {
        return entry.matches(getAccessLogRegex()) ∨ entry.matches(getErrorLogRegex());
    }

    private void parseLine() {
        String logLine = lineParts.get("_line");
        if (logLine.matches(getAccessLogRegex())) {
            parseAccessLogLine();
        } else if (logLine.matches(getErrorLogRegex())) {
```

abr 09, 18 3:41 **ApacheLogEntry.java** Page 2/2

```java
            parseErrorLogLine();
        }
    }

    private void parseErrorLogLine() {
        Matcher errorLogMatcher = errorLogPattern.matcher( lineParts.get("_line") );
        if (errorLogMatcher.matches()) {
            lineParts.put("date", errorLogMatcher.group(1));
            lineParts.put("error_type", errorLogMatcher.group(3));
            lineParts.put("error_client", errorLogMatcher.group(6));
            lineParts.put("error_msg", errorLogMatcher.group(7));
        }
    }

    private void parseAccessLogLine() {
        Matcher accessLogMatcher = accessLogPattern.matcher( lineParts.get("_line") );
        if (accessLogMatcher.matches()) {
            lineParts.put("client", accessLogMatcher.group(1));
            lineParts.put("date", accessLogMatcher.group(4));
            lineParts.put("http_method", accessLogMatcher.group(5));
            lineParts.put("resource", accessLogMatcher.group(6));
            lineParts.put("http_version", accessLogMatcher.group(7));
            lineParts.put("http_code", accessLogMatcher.group(8));
            lineParts.put("bytes", accessLogMatcher.group(9));
            lineParts.put("referer", accessLogMatcher.group(15));
        }
    }

    public boolean isError() {
        return lineParts.containsKey("error_type")
                ∧ lineParts.get("error_type") ≠ null
                ∧ lineParts.get("error_type").equals("error");
    }

    public String getError() {
        return lineParts.get("error_msg");
    }

    public boolean hasClient() {
        return lineParts.containsKey("client") ∧ lineParts.get("client") ≠ null;
    }

    public String getClient() {
        return lineParts.get("client");
    }

    public boolean hasResource() {
        return lineParts.containsKey("resource") ∧ lineParts.get("resource") ≠ null;
    }

    public String getResource() {
        return lineParts.get("resource");
    }

    public String getRawLine() {
        return lineParts.get("_line");
    }
}
```

abr 10, 18 23:35 **WorkExecutor.java** Page 1/1

```java
import java.util.ArrayList;
import java.util.concurrent.Callable;
import java.util.concurrent.Executor;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.apache.log4j.*;

public class WorkExecutor {
    private volatile ArrayList<Thread> workers;
    private volatile boolean active = false;
    WorkExecutor() {
        workers = new ArrayList<>();
    }

    public synchronized boolean isActive() {
        return active;
    }

    public synchronized void startWork() {
        active = true;
        workers.forEach(Thread::start);
    }

    public synchronized void end() {
        active = false;
        workers.forEach(Thread::interrupt);
    }

    public void addWorker(Runnable runnableWorker) {
        workers.add(new Thread(runnableWorker));
    }

    public void addWorker(Callable<Runnable> runneableCreator, int clons) {
        for (int i = 0; i < clons; i++) {
            try {
                Runnable runnable = runneableCreator.call();
                addWorker(runnable);
            } catch (Exception e) {
                Logger.getLogger(WorkExecutor.class).warn("Cannot create Runnable to app
end in work executor. Ignoring it.");
            }

        }
    }

    public void join() {
        workers.forEach(t -> {
            try {
                t.join();
            } catch (InterruptedException e) {
                Logger.getLogger(WorkExecutor.class).warn("InterruptedException on join "
 + t.getName());
            }
        });
    }
}
```

abr 10, 18 11:31 **ThreadActivity.java** Page 1/1

```java
import org.apache.log4j.*;

public abstract class ThreadActivity implements Runnable {
    protected Settings settings;
    protected Logger logger = Logger.getLogger(this.getClass().getName());
    protected final WorkExecutor workExecutor;
    ThreadActivity(Settings settings, WorkExecutor workExecutor) {
        this.workExecutor = workExecutor;
        this.settings = settings;
    }

    abstract boolean cycle() throws InterruptedException;

    abstract void onStop();


    @Override
    public void run() {
        logger = Logger.getLogger(this.getClass().getName() + "-" + Thread.curre
ntThread().getId());
        Boolean active = true;
        while (active) {
            try {
                active = cycle() ∧ workExecutor.isActive();
            } catch (InterruptedException e) {
                logger.info("InterruptSignal. Stopping work");
                active = false;
            }
        }
        onStop();
        logger.info("OK. Stopped");
    }

}
```

abr 10, 18 11:32 **StatViewer.java** Page 1/1

```java
import java.util.Set;
import org.apache.log4j.*;

public class StatViewer extends ThreadActivity {
    private final Stats stats;
    private final int pollSeconds;
    private final int topResources;
    StatViewer(Settings settings, Stats stats, WorkExecutor workExecutor) {
        super(settings, workExecutor);
        this.stats = stats;
        pollSeconds = settings.statsDumperFrequency();
        topResources = settings.statsTopMostRequestResources();
    }

    @Override
    boolean cycle() throws InterruptedException {
        /*
         * 1. Read Shared memory who has stats
         * 2. Show info in stdout
         * 3. Sleep 60 seconds.
         */
        StatsSummary statsSummary = stats.getSummary(topResources);
        stats.reset();

        StringBuilder msg = new StringBuilder()
                .append("\n\t- Request per seconds ").append((float) statsSummary.requests / (float) pollSeconds)
                .append("\n\t- Request per client ").append(statsSummary.requestPerClient)
                .append("\n\t- Errors ").append(statsSummary.errors)
                .append("\n\t- Most request resource");
        statsSummary.topResource.forEach(s → msg.append("\n\t\t").append(s));
        logger.info("STATS SUMMARY:" + msg.toString());

        Thread.sleep(1000 * pollSeconds);
        return true;
    }

    @Override
    void onStop() {
    }
}
```

abr 09, 18 2:01 **StatsSummary.java** Page 1/1

```java
import java.util.List;

public class StatsSummary {
    protected final int requests;
    protected final int errors;
    protected final float requestPerClient;
    protected final List<String> topResource;
    StatsSummary(int request, int errors, float requestPerClient, List<String> topResource) {
        this.requests = request;
        this.errors = errors;
        this.requestPerClient = requestPerClient;
        this.topResource = topResource;
    };
}
```

abr 10, 18 11:32 **StatsRegister.java** Page 1/1

```java
import Wrapper.ApacheLogEntry;
import org.apache.log4j.Logger;

import java.util.concurrent.ArrayBlockingQueue;

public class StatsRegister extends ThreadActivity {
    private final ArrayBlockingQueue<ApacheLogEntry> parserQueue;
    private final Stats stats;
    StatsRegister(ArrayBlockingQueue<ApacheLogEntry> parserQueue, Settings settings,
                  Stats stats, WorkExecutor workExecutor) {
        super(settings, workExecutor);
        this.parserQueue = parserQueue;
        this.stats = stats;
    }

    @Override
    boolean cycle() throws InterruptedException {
        /*
            * 1. Read Parser queue
            * 2. Store stat in shared memory
         */
        ApacheLogEntry logLine = parserQueue.take();
        stats.add(logLine);
        return true;
    }

    @Override
    void onStop() {
    }
}
```

abr 10, 18 12:56 **Stats.java** Page 1/2

```java
import Wrapper.ApacheLogEntry;
import org.apache.log4j.Logger;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;

public class Stats {
    private int requests;
    private int errors;
    private HashSet<String> clients;
    private HashMap<String, Integer> resourceCounter;

    public Stats() {
        clients = new HashSet<>();
        resourceCounter = new HashMap<>();
    }

    synchronized void reset() {
        requests = 0;
        errors = 0;
        clients.clear();
        resourceCounter.clear();
    }

    private int getRequestCount() {
        return requests;
    }

    private float getRequestPerClient() {
        return (clients.size() > 0) ? (float)requests / (float)clients.size() : 0;
    }

    private int getErrorsCount() {
        return errors;
    }

    synchronized void add(ApacheLogEntry log) {
        if (log.hasClient()) {
            requests += 1;
            clients.add(log.getClient());
        }
        if (log.isError()) {
            Logger.getLogger(StatsRegister.class).debug("Register error");
            errors+=1;
        }
        if (log.hasResource()) {
            String resourcer = log.getResource();
            Integer previousValue = resourceCounter.get(resourcer);
            resourceCounter.put(resourcer, previousValue == null ? 1 : previousValue + 1);
        }
    }

    synchronized StatsSummary getSummary( int countTopResources ) {
        return new StatsSummary(
          getRequestCount(),
          getErrorsCount(),
          getRequestPerClient(),
          getMostRequestResource(countTopResources)
        );
    }

```

abr 10, 18 12:56 **Stats.java** Page 2/2

```
65      private List<String> getMostRequestResource(int n) {
66          ArrayList<String> topResource = new ArrayList<>();
67          resourceCounter.entrySet().stream()
68                  .sorted((k1, k2) → -k1.getValue().compareTo(k2.getValue()))
69                  .forEach(k → topResource.add( "[" + k.getValue() + "]" + k.getK
    ey()));
70          return (topResource.size() ≤ n) ? topResource : topResource.subList(0, n
    );
71      }
72
73
74  }
```

abr 10, 18 17:05 **Settings.java** Page 1/2

```
1   import org.apache.log4j.Logger;
2
3   import java.io.*;
4   import java.nio.file.Files;
5   import java.util.Properties;
6
7   public class Settings {
8       private final Properties properties;
9       private Settings() {
10          properties = new Properties();
11      }
12
13      public static Settings fromProperties(String propertiesFile) {
14          InputStream input = null;
15          Settings settings = new Settings();
16          try {
17              input = new FileInputStream(propertiesFile);
18              settings.properties.load(input);
19          } catch (IOException ex) {
20              System.out.println("[ERROR] Cannot load " + propertiesFile + " using default p
    roperties");
21          } finally {
22              if (input ≠ null) {
23                  try {
24                      input.close();
25                  } catch (IOException e) {
26                      System.out.println("[WARNING] IOException when attemp to close " + pr
    opertiesFile);
27                  }
28              }
29          }
30          return settings;
31      }
32
33      public int statsDumperFrequency() {
34          return Integer.parseInt(properties.getProperty("STATS_FREQUENCY_SEC", "
    20"));
35      }
36
37      public int statsTopMostRequestResources() {
38          return Integer.parseInt(properties.getProperty("STATS_MOST_REQUEST_RESO
    URCE_TOP", "10"));
39      }
40
41      public String errorFrequencyWorkDir() {
42          return properties.getProperty("ERROR_HANDLER_WORK_DIR", "err/");
43      }
44
45      public int errorFrequencyPoll() {
46          return Integer.parseInt(properties.getProperty("ERROR_COLLECTOR_FREQUE
    NCY_SEC", "20"));
47      }
48
49      public String errorFrequencyFile() {
50          return properties.getProperty("ERROR_FREQUENCY_FILE", "error_frequency.log")
    ;
51      }
52
53      public int queueSize() {
54          return Integer.parseInt(properties.getProperty("QUEUE_SIZE", "1024"));
55      }
56
57      public String errorLogFile() {
58          return properties.getProperty("ERROR_LOG", "apache_error.log");
59      }
60
```

abr 10, 18 17:05 **Settings.java** Page 2/2

```
61      public String dumperLogFile() {
62          return properties.getProperty("DUMPER_LOG", "apache_dump.log");
63      }
64
65      public int errorFrequencyMaxFiles() {
66          return Integer.parseInt(properties.getProperty("ERROR_FREQUENCY_MAX_FI
    LES", "500"));
67      }
68
69      public InputStream getInputReader() {
70          String inputStream = properties.getProperty("READER_INPUT","STDIN");
71          if (inputStream.equals("STDIN")) {
72              return System.in;
73          }
74          try {
75              return new FileInputStream(new File(inputStream));
76          } catch (FileNotFoundException e) {
77              Logger logger = Logger.getLogger(Settings.class);
78              logger.warn("Cannot open file " + inputStream);
79              logger.debug(e);
80              logger.info("Using STDIN for Reader input");
81              return System.in;
82          }
83      }
84
85      public int numberParserWorkers() {
86          return Integer.parseInt( properties.getProperty("PARSER_WORKERS","1")
    );
87      }
88
89      public int numberErrorFrequencyWorkers() {
90          return Integer.parseInt( properties.getProperty("ERROR_FREQUENCY_WORK
    ERS","1") );
91      }
92
93      public int numberStatsRegisterWorkers() {
94          return Integer.parseInt( properties.getProperty("STATS_REGISTER_WORKERS
    ","1") );
95      }
96  }
```

abr 10, 18 12:52 **Reader.java** Page 1/1

```
1   import Wrapper.ApacheLogEntry;
2
3   import java.util.Random;
4   import java.util.Scanner;
5   import java.util.concurrent.ArrayBlockingQueue;
6
7   public class Reader extends ThreadActivity {
8       private final ArrayBlockingQueue<ApacheLogEntry> parserQueue;
9       private final ArrayBlockingQueue<ApacheLogEntry> dumperQueue;
10      private final Scanner stdinScanner;
11      Reader(ArrayBlockingQueue<ApacheLogEntry> parserQueue, ArrayBlockingQueue<Ap
    acheLogEntry> dumperQueue,
12              Settings settings, WorkExecutor workExecutor) {
13          super(settings,workExecutor);
14          this.parserQueue = parserQueue;
15          this.dumperQueue = dumperQueue;
16          this.stdinScanner = new Scanner(settings.getInputReader());
17      }
18
19      @Override
20      boolean cycle() throws InterruptedException {
21          /*
22          1. Read STDIN
23          1.2 detect if EOF -> gracefull quit
24          2. Detect if is Apache Log
25              2.1. Send to Parser
26              2.2. Send to Logger clon
27          */
28          if (¬stdinScanner.hasNextLine()) {
29              logger.info("EOF Detected. Closing...");
30              workExecutor.end();
31              return false;
32          }
33          String log = stdinScanner.nextLine();
34          ApacheLogEntry logEntry = ApacheLogEntry.from(log);
35          if (logEntry ≠ null) {
36              parserQueue.put(logEntry);
37              dumperQueue.put(new ApacheLogEntry(logEntry));
38          } else {
39              logger.info("Ignoring line\n'" + log + "'");
40          }
41          //NOTE: To Debug
42          if (logger.isDebugEnabled()) {
43              //logger.debug("Sleep Reader to DEBUG System");
44              Thread.sleep(100 + (new Random()).nextInt(300));
45          }
46          return true;
47      }
48
49      @Override
50      void onStop() {
51          stdinScanner.close();
52      }
53  }
```

abr 10, 18 11:31 **Parser.java** Page 1/1

```java
import Wrapper.ApacheLogEntry;
import org.apache.log4j.Logger;

import java.util.concurrent.ArrayBlockingQueue;

class Parser extends ThreadActivity {
    private final ArrayBlockingQueue<ApacheLogEntry> readerQueue;
    private final ArrayBlockingQueue<ApacheLogEntry> statsQueue;
    private final ArrayBlockingQueue<ApacheLogEntry> errorHandlerQueue;

    Parser(ArrayBlockingQueue<ApacheLogEntry> readerQueue, ArrayBlockingQueue<ApacheLogEntry> statsQueue,
           ArrayBlockingQueue<ApacheLogEntry> errorHandlerQueue, Settings settings, WorkExecutor workExecutor) {
        super(settings, workExecutor);
        this.readerQueue = readerQueue;
        this.statsQueue = statsQueue;
        this.errorHandlerQueue = errorHandlerQueue;
    }

    @Override
    boolean cycle() throws InterruptedException {
        ApacheLogEntry apacheLogEntry = new ApacheLogEntry(readerQueue.take());
        statsQueue.put(apacheLogEntry);
        if (apacheLogEntry.isError()) {
            errorHandlerQueue.put(apacheLogEntry);
        }
        return true;
    }

    @Override
    void onStop() {
    }
}
```

abr 10, 18 11:34 **Main.java** Page 1/2

```java
import Wrapper.ApacheLogEntry;
import sun.misc.Signal;

import java.io.IOException;
import java.util.concurrent.ArrayBlockingQueue;

import org.apache.log4j.*;


public class Main {
    public static void main(String[] args)  {
        Settings settings = Settings.fromProperties("config.properties");
        WorkExecutor workExecutor = new WorkExecutor();

        PropertyConfigurator.configure("log4j.properties");
        Logger logger = Logger.getLogger(Main.class);

        // 0. Signal handler to detect CTRL-C and do graceful quit
        Signal.handle(new Signal("INT"), sig -> {
            logger.info("CTRL+C Detected. Closing...");
            workExecutor.end();
        });

        // 1. Create Queues
        int queueCapacity = settings.queueSize();
        ArrayBlockingQueue<ApacheLogEntry> parserQueue           = new ArrayBlockingQueue<>(queueCapacity);
        ArrayBlockingQueue<ApacheLogEntry> dumperQueue           = new ArrayBlockingQueue<>(queueCapacity);
        ArrayBlockingQueue<ApacheLogEntry> errorHandlerQueue     = new ArrayBlockingQueue<>(queueCapacity);
        ArrayBlockingQueue<ApacheLogEntry> statsQueue            = new ArrayBlockingQueue<>(queueCapacity);
        ArrayBlockingQueue<ApacheLogEntry> errorFreqQueue        = new ArrayBlockingQueue<>(queueCapacity);

        // 2. Create workers in threads
        workExecutor.addWorker(
                new Reader(parserQueue, dumperQueue, settings, workExecutor)
        );
        workExecutor.addWorker(
                () -> new Parser(parserQueue, statsQueue, errorHandlerQueue, settings, workExecutor),
                settings.numberParserWorkers()
        );
        try {
            workExecutor.addWorker(
                    new Dumper(dumperQueue, settings, workExecutor)
            );
            workExecutor.addWorker(
                    new ErrorHandler(errorHandlerQueue, errorFreqQueue, settings, workExecutor)
            );
        } catch (IOException e) {
            logger.fatal("Error on create ErrorHandler and Dumper file. Aborting");
            System.exit(1);
        }
        ErrorFrequencyFileManager fileManager = new ErrorFrequencyFileManager(
                settings.errorFrequencyWorkDir(), settings.errorFrequencyMaxFiles()
        );
        workExecutor.addWorker(
                () -> new ErrorFrequency(errorFreqQueue, fileManager, settings, workExecutor),
                settings.numberErrorFrequencyWorkers()
        );
```

abr 10, 18 11:34 **Main.java** Page 2/2

```
58          workExecutor.addWorker(
59                  new ErrorFrequencyCollector(fileManager, settings, workExecutor)
60          );
61          Stats stats = new Stats();
62          workExecutor.addWorker(
63                  () → new StatsRegister(statsQueue, settings, stats, workExecuto
    r),
64                  settings.numberStatsRegisterWorkers()
65          );
66          workExecutor.addWorker(
67                  new StatViewer(settings, stats, workExecutor)
68          );
69
70          // 3. Run Threads
71          workExecutor.startWork();
72
73          // 4. Join Threads
74          workExecutor.join();
75          logger.info("Bye :)");
76      }
77  }
```

abr 10, 18 11:31 **ErrorHandler.java** Page 1/1

```
1   import Wrapper.ApacheLogEntry;
2
3   import java.io.FileWriter;
4   import java.io.IOException;
5   import java.io.PrintWriter;
6   import java.util.concurrent.ArrayBlockingQueue;
7   import org.apache.log4j.*;
8
9   public class ErrorHandler extends ThreadActivity {
10      private final ArrayBlockingQueue<ApacheLogEntry> parserQueue;
11      private final ArrayBlockingQueue<ApacheLogEntry> errorFrequencyQueue;
12      private final PrintWriter errorWriter;
13
14      ErrorHandler(ArrayBlockingQueue<ApacheLogEntry> parserQueue, ArrayBlockingQu
    eue<ApacheLogEntry> errorFreqQueue,
15                   Settings settings, WorkExecutor workExecutor) throws IOExceptio
    n {
16          super(settings, workExecutor);
17          this.parserQueue = parserQueue;
18          this.errorFrequencyQueue = errorFreqQueue;
19          errorWriter = new PrintWriter(new FileWriter(super.settings.errorLogFile
    ()));
20      }
21
22      private void writeError(String errorLineLog) {
23          errorWriter.write(errorLineLog);
24          errorWriter.write("\n");
25          if (errorWriter.checkError()) {
26              logger.warn("Error on write line in apache_error.log");
27          }
28      }
29
30      @Override
31      boolean cycle() throws InterruptedException {
32          /*
33              1. Read Parser Queue
34              2. Write error in error_log
35              3. Send Error to ErrorFrequency
36          */
37          ApacheLogEntry errorLineLog = parserQueue.take();
38          writeError(errorLineLog.getRawLine());
39          errorFrequencyQueue.put(errorLineLog);
40          return true;
41      }
42
43      @Override
44      void onStop() {
45          errorWriter.close();
46      }
47  }
```

abr 10, 18 12:59 **ErrorFrequency.java** Page 1/2

```java
import Wrapper.ApacheLogEntry;

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.HashMap;
import java.util.concurrent.ArrayBlockingQueue;

public class ErrorFrequency extends ThreadActivity {
    private final ArrayBlockingQueue<ApacheLogEntry> errorHandlerQueue;
    private final ErrorFrequencyFileManager fileManager;
    ErrorFrequency(ArrayBlockingQueue<ApacheLogEntry> errorHandlerQueue, ErrorFrequencyFileManager fileManager,
                   Settings settings, WorkExecutor workExecutor) {
        super(settings, workExecutor);
        this.fileManager = fileManager;
        this.errorHandlerQueue = errorHandlerQueue;
    }

    @Override
    boolean cycle() throws InterruptedException {
        /*
            1. Read Error handler Queue
            2. Write error file (name is hashed)
                ej. File: co.err (al errors who start with co)
                2.1. If file not exist. Create
                2.2. If file exist update (if exist in file) or append error.
        */
        registerLog(errorHandlerQueue.take());
        return true;
    }

    private HashMap<String,Integer> readFileError(String fileName) {
        HashMap<String,Integer> errors = new HashMap<>();
        try {
            Files.readAllLines(Paths.get(fileName)).forEach(s -> {
                String[] parts = s.split("==", 2);
                errors.put(parts[1].trim(), Integer.parseInt(parts[0].trim()));
            });
        } catch (IOException e) {
            logger.debug(e);
            logger.warn(fileName + " not exist. It will be created");
        }
        return errors;
    }

    private boolean writeErrorCountInFile(HashMap<String, Integer> errorCount, String fileName) {
        try {
            PrintWriter writer = new PrintWriter(new FileWriter(fileName));
            errorCount.forEach((key, value) -> writer.write(value + "==" + key + "\n"));
            writer.close();
            logger.debug("Updated errors in " + fileName);
        } catch (IOException e) {
            logger.warn("Cannot write in file [" + fileName + "]. Ignoring error log entry");
            logger.debug(e);
            return false;
        }
        return true;
    }

    private void registerLog(ApacheLogEntry apacheLog) {
        String error = apacheLog.getError().trim();
```

abr 10, 18 12:59 **ErrorFrequency.java** Page 2/2

```java
        HashMap<String,Integer> errorCount = new HashMap<>();
        logger.debug("Before take lock");
        fileManager.workOverFile(error, f -> {
            logger.debug("Take lock " + f);
            errorCount.putAll(readFileError(f));
            Integer actualErrorCount = errorCount.get(error);
            errorCount.put(error, actualErrorCount == null ? 1 : actualErrorCount + 1);
            boolean result = writeErrorCountInFile(errorCount, f);
            logger.debug("Free " + f);
            return result;
        });
        logger.debug("After work over the file (end error register)");
    }

    @Override
    void onStop() {
    }
}
```

abr 10, 18 8:12 **ErrorFrequencyFileManager.java** Page 1/1

```java
import java.io.File;
import java.util.ArrayList;
import java.util.concurrent.Callable;
import java.util.function.Function;

import org.apache.log4j.*;

public class ErrorFrequencyFileManager {
    private volatile ArrayList<Object> filesMutex;
    private final String workingDir;
    ErrorFrequencyFileManager(String workingDir, int maxFiles) {
        Logger logger = Logger.getLogger(ErrorFrequencyFileManager.class);
        this.workingDir = workingDir;
        if (createPath()) {
            logger.info("Created working directory (" + workingDir + ")");
        } else {
            logger.fatal("Cannot craeate working directory (" + workingDir + ").");
            System.exit(1);
        }
        filesMutex = new ArrayList<>();
        for (int i = 0; i < maxFiles; i++) {
            filesMutex.add( new Object() );
        }
    }

    private boolean createPath() {
        File workingDirectory = new File(workingDir);
        return workingDirectory.exists() ∨ workingDirectory.mkdir();
    }

    private synchronized Object getFileMutex(int fileId) {
        return filesMutex.get(fileId);
    }

    private synchronized int getNumberOfFiles() {
        return filesMutex.size();
    }

    private void workOverFile(int fileId, Function<String, Boolean> work) {
        String fileName = String.valueOf(fileId) + ".err";
        synchronized (getFileMutex(fileId)) {
            try {
                Boolean result = work.apply(workingDir + fileName);
                if (¬result) {
                    Logger.getLogger(ErrorFrequencyFileManager.class).debug("Work returned ‘False‘");
                }
            } catch (Exception e) {
                Logger.getLogger(ErrorFrequencyFileManager.class).warn("Exception when working over file",e);
            }
        }
    }

    public void workOverFile(String error, Function<String, Boolean> work) {
        int fileId = Math.abs(error.hashCode()) % getNumberOfFiles();
        workOverFile(fileId, work);
    }

    public void workOverFiles(Function<String, Boolean> work) {
        for (int i = 0; i < getNumberOfFiles(); i++) {
            workOverFile(i, work);
        }
    }
}
```

abr 10, 18 11:30 **ErrorFrequencyCollector.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.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.apache.log4j.*;

public class ErrorFrequencyCollector extends ThreadActivity {
    private final ErrorFrequencyFileManager fileManager;
    private final String workingDir;
    private final String outputLog;
    private final int pollSeconds;
    ErrorFrequencyCollector(ErrorFrequencyFileManager fileManager, Settings settings, WorkExecutor workExecutor) {
        super(settings, workExecutor);
        this.fileManager = fileManager;
        this.workingDir = settings.errorFrequencyWorkDir();
        this.outputLog = settings.errorFrequencyFile();
        this.pollSeconds = settings.errorFrequencyPoll();
    }

    @Override
    boolean cycle() throws InterruptedException {
        /*
            1. Sleep 3 minutes
            2. Collect all errors in hashmap
            3. Store errors in file 'error_frequency.log'
         */
        ArrayList<String> errorFrequencyDump = new ArrayList<>();
        collectErrors().entrySet().stream()
                .sorted((e1,e2) → -e1.getValue().compareTo(e2.getValue()))
                .forEach(e → errorFrequencyDump.add(e.getValue() + "|" + e.getKey()));
        generateTopErrorFile(errorFrequencyDump);
        Thread.sleep(1000 * pollSeconds);
        return true;
    }

    private void generateTopErrorFile(List<String> errors) {
        try {
            PrintWriter writer = new PrintWriter(new FileWriter(outputLog));
            for (String line : errors) {
                writer.write(line + "\n");
            }
            writer.close();
            logger.info("Generated " + outputLog);
        } catch (IOException e) {

            logger.warn("Cannot write in file [" + outputLog + "]");
            logger.debug(e);
        }
    }

    private List<String> readFile(String path) {
        try {
            if (new File(path).exists()) {
                return Files.readAllLines(Paths.get(path));
            }
        } catch (IOException e) {
            logger.warn("Cannot read file [" + path + "] to collect frequencies");
        }
        return new ArrayList<>();
```

abr 10, 18 11:30 **ErrorFrequencyCollector.java** Page 2/2

```
65      }
66
67      private HashMap<String,Integer> collectErrors() {
68          HashMap<String, Integer> errorCount = new HashMap<>();
69          fileManager.workOverFiles( fileName → {
70              logger.debug("Take lock on " + fileName);
71              readFile(fileName).forEach(s → {
72                  String[] parts = s.split("==", 2);
73                  errorCount.put(parts[1], Integer.parseInt(parts[0].trim()));
74              });
75              logger.debug("Free " + fileName);
76              return true;
77          });
78          return errorCount;
79
80      }
81
82      @Override
83      void onStop() {
84      }
85  }
```

abr 10, 18 11:30 **Dumper.java** Page 1/1

```
1   import Wrapper.ApacheLogEntry;
2
3   import java.io.FileWriter;
4   import java.io.IOException;
5   import java.io.PrintWriter;
6   import java.util.concurrent.ArrayBlockingQueue;
7   import org.apache.log4j.*;
8
9   public class Dumper extends ThreadActivity {
10      private final ArrayBlockingQueue<ApacheLogEntry> readerQueue;
11      private final PrintWriter dumperFile;
12
13      Dumper(ArrayBlockingQueue<ApacheLogEntry> readerQueue, Settings settings, Wo
    rkExecutor workExecutor) throws IOException {
14          super(settings, workExecutor);
15          this.readerQueue = readerQueue;
16          dumperFile = new PrintWriter(new FileWriter(super.settings.dumperLogFile
    ()));
17      }
18
19      private void writeLine(String line) {
20          dumperFile.write(line);
21          dumperFile.write("\n");
22          if (dumperFile.checkError()) {
23              logger.warn("Error on write line");
24          }
25      }
26
27      @Override
28      boolean cycle() throws InterruptedException {
29          writeLine(readerQueue.take().getRawLine());
30          return true;
31      }
32
33      @Override
34      void onStop() {
35          dumperFile.close();
36      }
37  }
```

Simple **Java** application that includes a class with `main()` method

abr 16, 18 17:56 **Table of Content** Page 1/1

Table of Contents