abr 08, 17 18:24 **AppTest.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class AppTest
    extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    public void testApp()
    {
        assertTrue( true );
    }
}
```

abr 08, 17 18:24 **TerminateSignal.java** Page 1/1

```java
package ar.fiuba.taller.utils;

public class TerminateSignal {

  private boolean terminate = false;

  public boolean hasTerminate() {
    return this.terminate;
  }

  public void terminate() {
    this.terminate = true;
  }
}
```

abr 08, 17 18:24 **Pair.java** Page 1/1

```java
package ar.fiuba.taller.utils;

public class Pair<A, B> {
  private A first;
  private B second;

  public Pair(A first, B second) {
    super();
    this.first = first;
    this.second = second;
  }

  public int hashCode() {
    int hashFirst = first ≠ null ? first.hashCode() : 0;
    int hashSecond = second ≠ null ? second.hashCode() : 0;

    return (hashFirst + hashSecond) * hashSecond + hashFirst;
  }

  public boolean equals(Object other) {
    if (other instanceof Pair) {
      Pair otherPair = (Pair) other;
      return ((this.first ≡ otherPair.first
          ∨ (this.first ≠ null ∧ otherPair.first ≠ null
              ∧ this.first.equals(otherPair.first)))
          ∧ (this.second ≡ otherPair.second
              ∨ (this.second ≠ null ∧ otherPair.second ≠ null
                  ∧ this.second.equals(otherPair.second))));
    }

    return false;
  }

  public String toString() {
    return "(" + first + "," + second + ")";
  }

  public A getFirst() {
    return first;
  }

  public void setFirst(A first) {
    this.first = first;
  }

  public B getSecond() {
    return second;
  }

  public void setSecond(B second) {
    this.second = second;
  }
}
```

abr 08, 17 18:24 **HttpRequester.java** Page 1/3

```java
package ar.fiuba.taller.utils;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Map;

import javax.net.ssl.HttpsURLConnection;

public class HttpRequester {

  public HttpRequester() {
    // TODO Auto-generated constructor stub
  }

  public String doHttpRequest(String method, String url,
      Map<String, String> headers, String data) throws Exception {
    String result = null;
    if (method.toLowerCase().equals("get")) {
      result = doGet(url, headers, data);
    } else if (method.toLowerCase().equals("post")) {
      result = doPost(url, headers, data);
    } else if (method.toLowerCase().equals("put")) {
      result = doPut(url, headers, data);
    }
    return result;

  }

  // HTTP GET request
  private String doGet(String url, Map<String, String> headers, String data)
      throws Exception {
    // Armo la conexion
    if (data.length() > 0)
      url += "?" + data;
    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    // optional default is GET
    con.setRequestMethod("GET");

    // add request header
    if (¬headers.isEmpty()) {
      for (Map.Entry<String, String> entry : headers.entrySet()) {
        con.setRequestProperty(entry.getKey(), entry.getValue());
      }
    }

    BufferedReader in = new BufferedReader(
        new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) ≠ null) {
      response.append(inputLine);
    }
    in.close();

    // print result
    return response.toString();
  }

  // HTTP POST request
  private String doPost(String url, Map<String, String> headers, String data)
```

```
67        throws Exception {
68      URL obj = new URL(url);
69      HttpURLConnection con = (HttpURLConnection) obj.openConnection();
70
71      // add reuqest method
72      con.setRequestMethod("POST");
73
74      // add reuqest header
75      if (¬headers.isEmpty()) {
76        for (Map.Entry<String, String> entry : headers.entrySet()) {
77          con.setRequestProperty(entry.getKey(), entry.getValue());
78        }
79      }
80
81      // Send post request
82      con.setDoOutput(true);
83      DataOutputStream wr = new DataOutputStream(con.getOutputStream());
84      wr.writeBytes(data);
85      wr.flush();
86      wr.close();
87
88      BufferedReader in = new BufferedReader(
89          new InputStreamReader(con.getInputStream()));
90      String inputLine;
91      StringBuffer response = new StringBuffer();
92
93      while ((inputLine = in.readLine()) ≠ null) {
94        response.append(inputLine);
95      }
96      in.close();
97
98      // print result
99      return response.toString();
100
101   }
102
103   // HTTP PUT request
104   private String doPut(String url, Map<String, String> headers, String data)
105       throws Exception {
106     URL obj = new URL(url);
107     HttpURLConnection con = (HttpURLConnection) obj.openConnection();
108
109     // add reuqest method
110     con.setRequestMethod("PUT");
111
112     // add reuqest header
113     if (¬headers.isEmpty()) {
114       for (Map.Entry<String, String> entry : headers.entrySet()) {
115         con.setRequestProperty(entry.getKey(), entry.getValue());
116       }
117     }
118
119     // Send post request
120     con.setDoOutput(true);
121     DataOutputStream wr = new DataOutputStream(con.getOutputStream());
122     wr.writeBytes(data);
123     wr.flush();
124     wr.close();
125
126
127     BufferedReader in = new BufferedReader(
128         new InputStreamReader(con.getInputStream()));
129     String inputLine;
130     StringBuffer response = new StringBuffer();
131
132     while ((inputLine = in.readLine()) ≠ null) {
```



```
133       response.append(inputLine);
134     }
135     in.close();
136
137     // print result
138     return response.toString();
139   }
140
141 }
```

abr 08, 17 18:24 **UserTask.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;

public class UserTask extends Task {

  public UserTask(Integer id, TASK_STATUS status) {
    super(id, status);
    // TODO Auto-generated constructor stub
  }

}
```

abr 08, 17 18:24 **UserPattern.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

public abstract class UserPattern {

  private Integer numberOfUsers;
  private Integer upperBound;

  public UserPattern(Integer numberOfUseres, Integer upperBound) {
    this.setNumberOfUsers(numberOfUseres);
    this.setUpperBound(upperBound);
  }

  public abstract Integer getUsers(Integer tick);

  public Integer getNumberOfUsers() {
    return numberOfUsers;
  }

  public void setNumberOfUsers(Integer numberOfUsers) {
    this.numberOfUsers = numberOfUsers;
  }

  public Integer getUpperBound() {
    return upperBound;
  }

  public void setUpperBound(Integer upperBound) {
    this.upperBound = upperBound;
  }

}
```

```
1  package ar.fiuba.taller.loadTestConsole;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.net.URISyntaxException;
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.HashMap;
9  import java.util.Iterator;
10 import java.util.List;
11 import java.util.Map;
12 import java.util.concurrent.ArrayBlockingQueue;
13 import java.util.concurrent.ExecutorService;
14 import java.util.concurrent.Executors;
15
16 import javax.swing.text.BadLocationException;
17 import javax.xml.parsers.DocumentBuilder;
18 import javax.xml.parsers.DocumentBuilderFactory;
19 import javax.xml.parsers.ParserConfigurationException;
20
21 import org.apache.log4j.Logger;
22 import org.jsoup.Jsoup;
23 import org.jsoup.nodes.Element;
24 import org.jsoup.select.Elements;
25 import org.w3c.dom.Node;
26 import org.w3c.dom.NodeList;
27 import org.xml.sax.SAXException;
28
29 import ar.fiuba.taller.utils.HttpRequester;
30
31 public class User implements Runnable {
32   private ArrayBlockingQueue<UserTask> userTaskPendingQueue;
33   private ArrayBlockingQueue<UserTask> userTaskFinishedQueue;
34   private ArrayBlockingQueue<SummaryTask> summaryQueue;
35   private ArrayBlockingQueue<ReportTask> reportQueue;
36   private ArrayBlockingQueue<DownloaderTask> downloaderTaskPendingQueue;
37   private ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue;
38   private ExecutorService downloadersThreadPool;
39   private Integer downloaders;
40   private DownloaderTask finishedTask;
41   private DownloaderTask downloaderTask;
42   private UserTask userTask;
43   private Boolean gracefullQuit;
44   private Map<String, String> map;
45   private HttpRequester httpRequester;
46   private String method;
47   private String url = "";
48   private String data = "";
49   private NodeList nList;
50   private Integer taskId;
51   List<String> requestList;
52   String html;
53   final static Logger logger = Logger.getLogger(User.class);
54
55   public User(ArrayBlockingQueue<UserTask> userTaskPendingQueue,
56       ArrayBlockingQueue<UserTask> userTaskFinishedQueue,
57       ArrayBlockingQueue<SummaryTask> summaryQueue,
58       ArrayBlockingQueue<ReportTask> reportQueue) {
59     this.userTaskPendingQueue = userTaskPendingQueue;
60     this.userTaskFinishedQueue = userTaskFinishedQueue;
61     this.summaryQueue = summaryQueue;
62     this.reportQueue = reportQueue;
63     this.downloaders = 0;
64     this.gracefullQuit = false;
65     this.downloaderTaskPendingQueue = null;
66     this.downloaderTaskFinishedQueue = null;
```



```
67     this.userTask = null;
68     this.finishedTask = null;
69     this.downloaderTask = null;
70     this.httpRequester = new HttpRequester();
71     this.method = "";
72     this.url = "";
73     this.data = "";
74     this.requestList = null;
75     this.html = "";
76   }
77
78   public void run() {
79     logger.info("Iniciando el usuario");
80     downloaders = ConfigLoader.getInstance()
81         .getMaxSizeDownloadersPoolThread();
82     long time_elapsed, time_end, time_start;
83     Integer bytesDownloaded;
84
85     // Script
86
87
88     logger.info("Obteniendo la lista de pasos");
89     nList = getStepList();
90
91     logger.info("Cargando el usuario");
92     initUser();
93
94     while (¬gracefullQuit) {
95       try {
96         userTask = userTaskPendingQueue.take();
97         // Informo al monitor que arranco el usuario
98         reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
99             Constants.TASK_STATUS.SUBMITTED, true, null));
100        if (userTask.getId() ≡ Constants.DISCONNECT_ID) {
101          terminateUser();
102        } else {
103          logger.info("Se inicia un nuevo pulso de usuario");
104          logger.info("Leo el script");
105          try {
106            logger.info("Leo el script");
107            for (int temp = 0; temp < nList.getLength(); temp++) {
108              map = new HashMap<String, String>();
109              method = "";
110              url = "";
111              data = "";
112              taskId = 0;
113
114              loadStep(temp);
115
116              logger.info("Siguiente paso a realizar: " + method
117                  + " " + url);
118              logger.info("Obteniendo recurso ...");
119              html = httpRequester
120                  .doHttpRequest(method, url, map, data)
121                  .toLowerCase();
122              time_start = System.currentTimeMillis();
123              bytesDownloaded = html.length();
124              time_end = System.currentTimeMillis();
125              time_elapsed = time_end − time_start;
126
127              logger.info(
128                  "Bytes descargados: " + bytesDownloaded);
129              logger.info("Tiempo inicial: " + time_start
130                  + " nanosgundos");
131              logger.info("Tiempo final: " + time_end
132                  + " nanosgundos");
```

```
133                 logger.info("Tiempo transcurrido: " + time_elapsed
134                     + " nanosgundos");
135
136                 // Enviando estadistica de descarga a la cola de
137                 // estadisticas
138                 summaryQueue
139                     .put(new SummaryTask(Constants.DEFAULT_ID,
140                         Constants.TASK_STATUS.SUBMITTED, 0,
141                         true, time_elapsed));
142
143                 analyzeResource();
144
145                 // Informo al monitor que analice una url
146                 reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
147                     Constants.TASK_STATUS.EXECUTING, true,
148                     null));
149                 waitDownloaders();
150               }
151             } catch (IOException e) {
152               //e.printStackTrace();
153             } catch (Exception e) {
154               // Informo al summary que fallo la descarga
155               summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID,
156                   Constants.TASK_STATUS.SUBMITTED, 0, false, 0));
157               // Informo al monitor que fallo la descarga
158               reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
159                   Constants.TASK_STATUS.FAILED, true, null));
160               //e.printStackTrace();
161             }
162
163           }
164           // Informo que el user termino
165           logger.info("Informando al monitor que el usuario termino");
166           reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
167               Constants.TASK_STATUS.FINISHED, true, null));
168         } catch (InterruptedException e1) {
169           // TODO Auto-generated catch block
170           e1.printStackTrace();
171         }
172       }
173     }
174
175     private List<String> getLinks(String page, String tag)
176         throws URISyntaxException, IOException, BadLocationException {
177       List<String> requestsList = new ArrayList<String>();
178       String currentAttribute = null;
179
180       if (tag.equals(Constants.LINK_TAG)) {
181         currentAttribute = "href";
182       } else if (tag.equals(Constants.SCRIPT_TAG)) {
183         currentAttribute = "src";
184       } else if (tag.equals(Constants.IMG_TAG)) {
185         currentAttribute = "src";
186       }
187       org.jsoup.nodes.Document doc = Jsoup.parse(page);
188       Elements resource = doc.select(tag);
189       Iterator<Element> it = resource.iterator();
190
191       while (it.hasNext()) {
192         requestsList.add(it.next().attr(currentAttribute));
193       }
194
195       return requestsList;
196     }
197
198     private NodeList getStepList() {
```



```
199       File inputFile = new File(Constants.SCRIPT_FILE);
200       DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
201       DocumentBuilder dBuilder = null;
202       try {
203         dBuilder = dbFactory.newDocumentBuilder();
204       } catch (ParserConfigurationException e3) {
205         // TODO Auto-generated catch block
206         e3.printStackTrace();
207       }
208       org.w3c.dom.Document doc = null;
209       try {
210         doc = dBuilder.parse(inputFile);
211       } catch (SAXException e3) {
212         // TODO Auto-generated catch block
213         e3.printStackTrace();
214       } catch (IOException e3) {
215         // TODO Auto-generated catch block
216         e3.printStackTrace();
217       }
218       doc.getDocumentElement().normalize();
219       NodeList nList = doc.getElementsByTagName("request");
220       return nList;
221     }
222
223     private void initUser() {
224       logger.info("Creo la cola de tareas");
225       downloaderTaskPendingQueue =
226           new ArrayBlockingQueue<DownloaderTask>(
227           ConfigLoader.getInstance().getTasksQueueSize());
228
229       logger.info(
230           "Creo la cola para recibir los mensajes de "
231           + "terminado de los downloaders");
232       downloaderTaskFinishedQueue =
233           new ArrayBlockingQueue<DownloaderTask>(
234           ConfigLoader.getInstance().getTasksQueueSize());
235
236       logger.info("Creo el pool de threads de downloaders");
237       downloadersThreadPool = Executors
238           .newFixedThreadPool(downloaders);
239
240       logger.info("Lanzo " + downloaders
241           + " downloaders y les paso las dos colas");
242       for (int i = 0; i < downloaders; i++) {
243         logger.info("Lanzando el downloader: " + i);
244         downloadersThreadPool.submit(new Downloader(
245             downloaderTaskPendingQueue, downloaderTaskFinishedQueue,
246             summaryQueue, reportQueue));
247       }
248     }
249
250     private void terminateUser() throws InterruptedException {
251       logger.info(
252           "Se ha recibido un mensaje de desconexion. Se "
253           + "envÃ-a mensaje de desconexion a los downloaders");
254       for (int i = 0; i < downloaders; ++i) {
255         downloaderTaskPendingQueue
256             .put(new DownloaderTask(Constants.DISCONNECT_ID,
257                 null, null, null, null));
258       }
259       logger.info("Esperando a que los downloaders finalicen");
260       /*while (downloaders > 0) {
261         downloaderTask = downloaderTaskFinishedQueue.take();
262         downloaders--;
263       }*/
264       logger.info("Downloaders finalizados");
```

```
265       logger.info("Finalizando usuario");
266       logger.info(
267           "Avisando al Control principal que el usuario "
268           + "termino");
269       userTaskFinishedQueue.put(userTask);
270       gracefullQuit = true;
271       logger.info("Usuario finalizado");
272     }
273
274     private void loadStep(Integer item) {
275       // Obtengo la url
276       org.w3c.dom.Node nNode = nList.item(item);
277       org.w3c.dom.Element eElement =
278           (org.w3c.dom.Element) nNode;
279       method = eElement.getElementsByTagName("method")
280           .item(0).getTextContent();
281       url = eElement.getElementsByTagName("url").item(0)
282           .getTextContent();
283       data = eElement.getElementsByTagName("data").item(0)
284           .getTextContent();
285       org.w3c.dom.Element e2 = (org.w3c.dom.Element) eElement
286           .getElementsByTagName("headers").item(0);
287       org.w3c.dom.NodeList n2 = e2.getChildNodes();
288       for (int i = 0; i < n2.getLength(); i++) {
289         Node n = n2.item(i);
290         org.w3c.dom.NodeList n3 = n.getChildNodes();
291         if (n3.getLength() > 0) {
292           if (¬(n3.item(1).getTextContent().trim()
293               .equals(""))) {
294             map.put(n3.item(1).getTextContent(),
295                 n3.item(3).getTextContent());
296           }
297         }
298       }
299     }
300
301     private void waitDownloaders() throws InterruptedException {
302       logger.info(
303           "Esperando a que terminen los downloaders");
304       //
305       while (taskId > 0) {
306         finishedTask = downloaderTaskFinishedQueue
307             .take();
308         taskId--;
309         logger.info("Task finalizada:\nTaskId: "
310             + finishedTask.getId() + "\nStatus: "
311             + finishedTask.getStatus());
312       }
313     }
314
315     private void analyzeResource() throws InterruptedException, URISyntaxException
    , IOException, BadLocationException {
316       logger.info(
317           "Rescato los tags LINK, IMG y SCRIPT e "
318           + "inserto las tasks en la cola");
319       for (String tag : Arrays.asList(Constants.IMG_TAG,
320           Constants.SCRIPT_TAG, Constants.LINK_TAG)) {
321         logger.debug("Analizando el tag " + tag
322             + "sobre el documento: html");
323         requestList = getLinks(html, tag);
324         logger.debug(
325             "Recursos encontrados en la pagina"
326             + " analizada: "
327                 + requestList.size());
328         for (String request : requestList) {
329           logger.info(
```



```
330               "Enviando una nueva task con los siguiente "
331               + "parametros:\nTaskId: "
332                   + taskId + "\nMetodo: "
333                   + Constants.GET_METHOD
334                   + "\nRequest: " + request
335                   + "\nEstado: "
336                   + Constants.TASK_STATUS.SUBMITTED);
337           downloaderTaskPendingQueue
338               .put(new DownloaderTask(taskId,
339                   Constants.GET_METHOD,
340                   request,
341                   Constants.TASK_STATUS.SUBMITTED,
342                   tag));
343           ++taskId;
344         }
345       }
346     }
347   }
```

abr 08, 17 18:24 **Terminator.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import org.apache.log4j.Logger;

import ar.fiuba.taller.utils.TerminateSignal;

public class Terminator extends Thread {
  private TerminateSignal terminateSignal;
  final static Logger logger = Logger.getLogger(Terminator.class);

  public Terminator(TerminateSignal terminateSignal) {
    this.terminateSignal = terminateSignal;
  }

  public void run() {
    try {
      TerminateSignal dotSignal = new TerminateSignal();
      Thread dotPrinterThread = new Thread(new DotPrinter(dotSignal));
      logger.info("Finalizando la simulacion...");
      System.out.print("Finalizando la simulacion");
      dotPrinterThread.start();
      terminateSignal.terminate();
      dotSignal.terminate();
      dotPrinterThread.join();
      logger.info("Simulacion Finalizada");
      System.out.println("");
      System.out.println("Simulacion Finalizada");
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

}
```

abr 08, 17 18:24 **Task.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

public abstract class Task {

  private Integer id;
  private Constants.TASK_STATUS status;

  public Task(Integer id, Constants.TASK_STATUS status) {
    this.id = id;
    this.status = status;
  }

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

  public Constants.TASK_STATUS getStatus() {
    return status;
  }

  public void setStatus(Constants.TASK_STATUS status) {
    this.status = status;
  }

}
```

abr 08, 17 18:24 **SummaryTask.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;

public class SummaryTask extends Task {
  private Integer usersAmount;
  private Boolean successfullRequest;
  private long timeElapsed;

  public Integer getUsersAmount() {
    return usersAmount;
  }

  public void setUsersAmount(Integer usersAmount) {
    this.usersAmount = usersAmount;
  }

  public long getTimeElapsed() {
    return timeElapsed;
  }

  public void setTimeElapsed(long timeElapsed) {
    this.timeElapsed = timeElapsed;
  }

  public SummaryTask(Integer id, TASK_STATUS status, Integer usersAmount,
      Boolean successfullRequest, long timeElapsed) {
    super(id, status);
    this.usersAmount = usersAmount;
    this.successfullRequest = successfullRequest;
    this.timeElapsed = timeElapsed;
  }

  public Boolean getSuccessfullRequest() {
    return successfullRequest;
  }

  public void setSuccessfullRequest(Boolean successfullRequest) {
    this.successfullRequest = successfullRequest;
  }

}
```

abr 08, 17 18:24 **SummaryPrinter.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import org.apache.log4j.Logger;

import ar.fiuba.taller.utils.TerminateSignal;

public class SummaryPrinter implements Runnable {

  Summary summary;
  TerminateSignal terminateSignal;
  TerminateSignal globalTerminateSignal;

  final static Logger logger = Logger.getLogger(App.class);

  public SummaryPrinter(Summary summary, TerminateSignal terminateSignal,
      TerminateSignal globalTerminateSignal) {
    super();
    this.summary = summary;
    this.terminateSignal = terminateSignal;
    this.globalTerminateSignal = globalTerminateSignal;
  }

  public void run() {
    logger.info("Iniciando SummaryPrinter");
    final String ANSI_CLS = "\u001b[2J";
    final String ANSI_HOME = "\u001b[H";
    while (¬terminateSignal.hasTerminate()) {
      if (¬globalTerminateSignal.hasTerminate()) {
        // Limpio la pantalla
        System.out.print(ANSI_CLS + ANSI_HOME);
        System.out.flush();

        // Imprimo el resumen
        System.out.println("Load Test Console: Resumen de ejecucion");
        System.out.println("––––––––––––––––––––––––––––––––––––––––");
        System.out.println("Tiempo de descarga promedio...: "
            + summary.getAverageTime() + " ms");
        System.out.println("Requests exitosos.....: "
            + summary.getSuccessfullrequest() + "/"
            + summary.getTotalRequests());
        System.out.println("Requests fallidos.....: "
            + summary.getFailedrequest() + "/"
            + summary.getTotalRequests());
        System.out.println("Cantidad de usuarios.....: "
            + summary.getUsers());
        System.out.println("");
        System.out.println("Presione ^C para terminar...");
      }
      try {
        Thread.sleep(Constants.SUMMARY_PRINTER_TIMEOUT);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }
    logger.info("Finalizando SummaryPrinter");
  }
}
```

abr 08, 17 18:24 **Summary.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

public class Summary {
  private long totalTime;
  private Integer successfullrequest;
  private Integer failedrequest;
  private Integer users;

  public Summary() {
    super();
    totalTime = 0;
    successfullrequest = 0;
    failedrequest = 0;
    users = 0;
  }

  public synchronized Integer getSuccessfullrequest() {
    return successfullrequest;
  }

  public synchronized void setSuccessfullrequest(Integer successfullrequest) {
    this.successfullrequest = successfullrequest;
  }

  public synchronized Integer getFailedrequest() {
    return failedrequest;
  }

  public synchronized void setFailedrequest(Integer failedrequest) {
    this.failedrequest = failedrequest;
  }

  public synchronized Integer getUsers() {
    return users;
  }

  public synchronized void setUsers(Integer users) {
    this.users = users;
  }

  public synchronized long getAverageTime() {
    if (successfullrequest > 0) {
      return totalTime / successfullrequest;
    }
    return 0;
  }

  public synchronized void addTime(long time) {
    totalTime += time;
  }

  public synchronized Integer getTotalRequests() {
    return successfullrequest + failedrequest;
  }
}
```

abr 08, 17 18:24 **SummaryController.java** Page 1/2

```java
package ar.fiuba.taller.loadTestConsole;

import java.util.concurrent.ArrayBlockingQueue;

import org.apache.log4j.Logger;

public class SummaryController implements Runnable {
  ArrayBlockingQueue<SummaryTask> pendingSummaryQueue;
  ArrayBlockingQueue<SummaryTask> finishedSummaryQueue;
  Summary summary;
  final static Logger logger = Logger.getLogger(App.class);

  public SummaryController(
      ArrayBlockingQueue<SummaryTask> pendingSummaryQueue,
      ArrayBlockingQueue<SummaryTask> finishedSummaryQueue,
      Summary summary) {
    super();
    this.pendingSummaryQueue = pendingSummaryQueue;
    this.finishedSummaryQueue = finishedSummaryQueue;
    this.summary = summary;
  }

  public void run() {
    logger.info("Se inicia el monitor");
    SummaryTask summaryTask = null;
    //
    try {
      do {
        summaryTask = pendingSummaryQueue.take();
        logger.info("Estadistica recibida:\n" + "Id: "
            + summaryTask.getId() + "\nStatus: "
            + summaryTask.getStatus() + "\nTime elapsed: "
            + summaryTask.getTimeElapsed() + "\nAmount of users: "
            + summaryTask.getUsersAmount()
            + "\nSuccessfull request: "
            + summaryTask.getSuccessfullRequest());
        if (summaryTask.getId() ≠ Constants.DISCONNECT_ID) {
          // Actualizo las estadisticas
          // Actualizo la cantidad de usuarios
          if (summaryTask.getUsersAmount() ≠ 0) {
            summary.setUsers(summaryTask.getUsersAmount());
          }
          // Actualizo los requests
          if (summaryTask.getSuccessfullRequest() ≠ null) {
            if (summaryTask.getSuccessfullRequest()) {
              summary.setSuccessfullrequest(
                  summary.getSuccessfullrequest() + 1);
            } else {
              summary.setFailedrequest(
                  summary.getFailedrequest() + 1);
            }
          }
          // Actualizo el tiempo promedio
          summary.addTime(summaryTask.getTimeElapsed());
          logger.info("Estadisticas actualizadas");
        }
      } while (summaryTask.getId() ≠ Constants.DISCONNECT_ID);
      logger.info(
          "Finalizando el monitor. Enviando mensje de finalizacion"
          + " al control principal.");
      finishedSummaryQueue.put(summaryTask);
    } catch (InterruptedException e) {
      logger.warn("No se ha podido tomar la tarea de la pendingSummaryQueue");
      e.printStackTrace();
    }
    logger.info("Monitor finalizado.");
```

abr 08, 17 18:24 **SummaryController.java** Page 2/2

```
67    }
68  }
```

abr 08, 17 18:24 **StairsUserPattern.java** Page 1/1

```
1   package ar.fiuba.taller.loadTestConsole;
2
3   import java.util.List;
4
5   public class StairsUserPattern extends UserPattern {
6
7     private Integer stepLength;
8
9     private Integer ticksLeft;
10
11    private Integer heightOfStep;
12
13    public StairsUserPattern(List<Integer> paramList, Integer upperBound) {
14      super(paramList.get(0), upperBound);
15      this.stepLength = paramList.get(1);
16      this.ticksLeft = paramList.get(1);
17      this.heightOfStep = paramList.get(0);
18    }
19
20    @Override
21    public Integer getUsers(Integer tick) {
22      if (ticksLeft ≡ 0) {
23        ticksLeft = stepLength;
24        setNumberOfUsers(getNumberOfUsers() + heightOfStep);
25      }
26      ticksLeft−−;
27      return getNumberOfUsers() ≤ getUpperBound() ? getNumberOfUsers()
28          : getUpperBound();
29    }
30  }
```

abr 08, 17 18:24 **ReportTask.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;

public class ReportTask extends Task {

  private Boolean analyzer;
  private String resource;

  public Boolean getAnalyzer() {
    return analyzer;
  }

  public void setAnalyzer(Boolean analyzer) {
    this.analyzer = analyzer;
  }

  public String getResource() {
    return resource;
  }

  public void setResource(String resource) {
    this.resource = resource;
  }

  public ReportTask(Integer id, TASK_STATUS status, Boolean analyzer,
      String resource) {
    super(id, status);
    this.analyzer = analyzer;
    this.resource = resource;
  }
}
```

abr 08, 17 18:24 **Report.java** Page 1/2

```java
package ar.fiuba.taller.loadTestConsole;

public class Report {

  private Integer analyzedUrl;
  private Integer downloadedScripts;
  private Integer downloadedLinks;
  private Integer downloadedImages;
  private Integer executionScriptThreads;
  private Integer downloadResourceThreads;

  public synchronized Integer getAnalyzedUrl() {
    return analyzedUrl;
  }

  public synchronized void incAnalyzedUrl() {
    analyzedUrl++;
  }

  public synchronized void decAnalyzedUrl() {
    analyzedUrl--;
  }

  public synchronized Integer getDownloadedScripts() {
    return downloadedScripts;
  }

  public synchronized void incDownloadedScripts() {
    downloadedScripts++;
  }

  public synchronized void decDownloadedScripts() {
    downloadedScripts--;
  }

  public synchronized Integer getDownloadedLinks() {
    return downloadedLinks;
  }

  public synchronized void incDownloadedLinks() {
    downloadedLinks++;
  }

  public synchronized void decDownloadedLinks() {
    downloadedLinks--;
  }

  public synchronized Integer getDownloadedImages() {
    return downloadedImages;
  }

  public synchronized void incDownloadedImages() {
    downloadedImages++;
  }

  public synchronized void decDownloadedImages() {
    downloadedImages--;
  }

  public synchronized Integer getExecutionScriptThreads() {
    return executionScriptThreads;
  }

  public synchronized void incExecutionScriptThreads() {
    executionScriptThreads++;
  }
```

abr 08, 17 18:24 **Report.java** Page 2/2

```
67
68    public synchronized void decExecutionScriptThreads() {
69      executionScriptThreads--;
70    }
71
72    public synchronized Integer getDownloadResourceThreads() {
73      return downloadResourceThreads;
74    }
75
76    public synchronized void incDownloadResourceThreads() {
77      downloadResourceThreads++;
78    }
79
80    public synchronized void decDownloadResourceThreads() {
81      downloadResourceThreads--;
82    }
83
84    public Report() {
85      super();
86      analyzedUrl = 0;
87      downloadedScripts = 0;
88      downloadedLinks = 0;
89      downloadedImages = 0;
90      executionScriptThreads = 0;
91      downloadResourceThreads = 0;
92    }
93
94  }
```

abr 08, 17 18:24 **ReportController.java** Page 1/2

```
1   package ar.fiuba.taller.loadTestConsole;
2
3   import java.util.concurrent.ArrayBlockingQueue;
4
5   import org.apache.log4j.Logger;
6
7   public class ReportController implements Runnable {
8
9     private ArrayBlockingQueue<ReportTask> pendingReportQueue;
10    private ArrayBlockingQueue<ReportTask> finishedReportQueue;
11    private Report report;
12    final static Logger logger = Logger.getLogger(App.class);
13
14    public ReportController(ArrayBlockingQueue<ReportTask> pendingReportQueue,
15        ArrayBlockingQueue<ReportTask> finishedReportQueue, Report report) {
16      super();
17      this.pendingReportQueue = pendingReportQueue;
18      this.finishedReportQueue = finishedReportQueue;
19      this.report = report;
20    }
21
22    public void run() {
23      logger.info("Se inicia el report controller");
24      ReportTask reportTask = null;
25      //
26      try {
27        do {
28          reportTask = pendingReportQueue.take();
29          logger.info(
30              "Estadistica recibida:\n" + "Id: " + reportTask.getId()
31                  + "\nStatus: " + reportTask.getStatus()
32                  + "\nAnalyzer: " + reportTask.getAnalyzer()
33                  + "\nResource: " + reportTask.getResource());
34          if (reportTask.getId() ≠ Constants.DISCONNECT_ID) {
35            // Actualizo el reporte
36            if (reportTask.getAnalyzer()) { // Es una task enviada por
37                                            // un user
38              logger.debug("VINE POR ACA");
39              if (reportTask
40                  .getStatus() ≡ Constants.TASK_STATUS.SUBMITTED) {
41                logger.debug("INCREMENTO THREAD");
42                report.incExecutionScriptThreads();
43              } else if (reportTask
44                  .getStatus() ≡ Constants.TASK_STATUS.EXECUTING) {
45                logger.debug("INCREMENTO URL");
46                report.incAnalyzedUrl();
47              } else {
48                logger.debug("DECREMENTO THREAD");
49                report.decExecutionScriptThreads();
50              }
51            } else { // Es una task enviada por un downloader
52              if (reportTask
53                  .getStatus() ≡ Constants.TASK_STATUS.SUBMITTED) {
54                report.incDownloadResourceThreads();
55              } else if (reportTask
56                  .getStatus() ≡ Constants.TASK_STATUS.EXECUTING) {
57                if (reportTask.getResource()
58                    .equals(Constants.SCRIPT_TAG)) {
59                  report.incDownloadedScripts();
60                } else if (reportTask.getResource()
61                    .equals(Constants.LINK_TAG)) {
62                  report.incDownloadedLinks();
63                } else if (reportTask.getResource()
64                    .equals(Constants.IMG_TAG)) {
65                  report.incDownloadedImages();
66                }
```

abr 08, 17 18:24 **ReportController.java** Page 2/2

```
67              } else {
68                report.decDownloadResourceThreads();
69              }
70            }
71            logger.info("Reporte actualizado");
72          }
73        } while (reportTask.getId() ≠ Constants.DISCONNECT_ID);
74        logger.info(
75            "Finalizando el controller. "
76            + "Enviando mensje de finalizacion al control principal.");
77        finishedReportQueue.put(reportTask);
78      } catch (InterruptedException e) {
79        // TODO Auto-generated catch block
80        e.printStackTrace();
81      }
82      logger.info("Controller finalizado.");
83    }
84
85  }
```

abr 08, 17 18:24 **RampUserPattern.java** Page 1/1

```
1   package ar.fiuba.taller.loadTestConsole;
2
3   import java.util.List;
4
5   public class RampUserPattern extends UserPattern {
6
7     public RampUserPattern(List<Integer> paramList, Integer upperBound) {
8       super(paramList.get(0), upperBound);
9     }
10
11    @Override
12    public Integer getUsers(Integer tick) {
13      return getNumberOfUsers() * tick ≤ getUpperBound()
14          ? getNumberOfUsers() * tick : getUpperBound();
15    }
16
17  }
```

abr 08, 17 18:24 **Monitor.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import java.io.IOException;
import java.io.PrintWriter;

import org.apache.log4j.Logger;

import ar.fiuba.taller.utils.TerminateSignal;

public class Monitor implements Runnable {

  Report report;
  TerminateSignal terminateSignal;

  final static Logger logger = Logger.getLogger(App.class);

  public Monitor(Report report, TerminateSignal terminateSignal) {
    super();
    this.report = report;
    this.terminateSignal = terminateSignal;
  }

  public void run() {
    logger.info("Iniciando Monitor");
    while (¬terminateSignal.hasTerminate()) {

      try {
        PrintWriter writer = new PrintWriter(Constants.REPORT_FILE,
            "UTF-8");
        writer.println("Load Test Console: Monitor de reportes");
        writer.println("----------------------------------------");
        writer.println("URLs analizadas.....: "
            + report.getAnalyzedUrl());
        writer.println("SCRIPTS descargados.....: "
            + report.getDownloadedScripts());
        writer.println("LINKS descargados.....: "
            + report.getDownloadedLinks());
        writer.println("IMGs descargadas.....: "
            + report.getDownloadedImages());
        writer.println("Hilos ejecutando script.....: "
            + report.getExecutionScriptThreads());
        writer.println("Hilos descargando recurso.....: "
            + report.getDownloadResourceThreads());
        writer.close();
      } catch (IOException e) {
        // do something
      }

      try {
        Thread.sleep(Constants.MONITOR_TIMEOUT);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        logger.warn("Fallo el sleep del monitor");
      }
    }
    logger.info("Finalizando Monitor");
  }

}
```

abr 08, 17 18:24 **LoadTestConsole.java** Page 1/4

```java
package ar.fiuba.taller.loadTestConsole;

import ar.fiuba.taller.utils.*;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import org.apache.log4j.Logger;

public class LoadTestConsole {

  private Integer maxSizeUserPoolThread;
  private String function;
  private List<Integer> functionParamList;
  private UserPattern userPattern;
  private List<Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>>> usersQueu
esList;
  private TerminateSignal terminateSignal;
  private ArrayBlockingQueue<UserTask> userTaskPendingQueue;
  private ArrayBlockingQueue<UserTask> userTaskFinishedQueue;
  private ArrayBlockingQueue<SummaryTask> summaryPendingQueue;
  private ArrayBlockingQueue<SummaryTask> summaryFinishedQueue;
  private ArrayBlockingQueue<ReportTask> reportPendingQueue;
  private ArrayBlockingQueue<ReportTask> reportFinishedQueue;
  private Summary summary;
  private Report report;
  private TerminateSignal summaryTerminateSignal;
  private TerminateSignal reportTerminateSignal;
  private Thread summaryControllerThread;
  private Thread summaryPrinterThread;
  private Thread reportControllerThread;
  private Thread monitorThread;

  final static Logger logger = Logger.getLogger(App.class);

  public LoadTestConsole(TerminateSignal terminateSignal) throws IOException {
    this.terminateSignal = terminateSignal;
    ConfigLoader.getInstance().init(Constants.PROPERTIES_FILE);
    usersQueuesList = new ArrayList<Pair<BlockingQueue<UserTask>, BlockingQueue<
UserTask>>>();
    function = ConfigLoader.getInstance().getFunction();
    functionParamList = ConfigLoader.getInstance()
        .getFunctionPatternParam();
    maxSizeUserPoolThread = ConfigLoader.getInstance()
        .getMaxsizeUserPoolThread();

    // TODO: Arreglar esto
    if (function.equals("ConstantUserPattern")) {
      userPattern = new ConstantUserPattern(functionParamList);
    } else if (function.equals("StairsUserPattern")) {
      userPattern = new StairsUserPattern(functionParamList,
          ConfigLoader.getInstance().getMaxsizeUserPoolThread());
    } else if (function.equals("RampUserPattern")) {
      userPattern = new RampUserPattern(functionParamList,
          ConfigLoader.getInstance().getMaxsizeUserPoolThread());
    }

  }

  public void start() {
    Integer currentUsers = 0, deltaUseres = 0, tick = 0, userNumber = 0;
    UserTask userTask;
```

```
65      logger.info("Se inicia una nueva instancia de LoadTestConsole");
66      initConsole();
67      logger.info("Creando el pool de threads de usuarios");
68      ExecutorService usersThreadPool = Executors
69          .newFixedThreadPool(maxSizeUserPoolThread);
70      initReportThreads();
71
72      try {
73        while (¬terminateSignal.hasTerminate()) {
74          tick++;
75          deltaUseres = userPattern.getUsers(tick) − currentUsers;
76          logger.info("Se inicia el pulso: " + tick);
77          logger.info("Cantidad de usuarios actualmente corriendo: "
78              + currentUsers);
79          logger.info(
80              "Cantidad de usuarios que deben correr en este pulso: "
81                  + userPattern.getUsers(tick));
82          logger.info("Cantidad de usuarios que deben ingresar: "
83              + deltaUseres);
84
85          for (int i = 0; i < deltaUseres; i++) {
86            currentUsers++;
87            logger.info("Creando el usuario: " + currentUsers);
88            // Creo las colas para el nuevo usuario
89            userTaskPendingQueue = new ArrayBlockingQueue<UserTask>(
90                ConfigLoader.getInstance().getTasksQueueSize());
91            userTaskFinishedQueue = new ArrayBlockingQueue<UserTask>(
92                ConfigLoader.getInstance().getTasksQueueSize());
93
94            // Me guardo las dos colas en la lista de colas de usuarios
95            usersQueuesList
96                .add(new Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>>(
97                    userTaskPendingQueue,
98                    userTaskFinishedQueue));
99
100           // Creo el usuario y les paso las colas para que puedan
101           // insertar tareas
102           usersThreadPool.submit(new User(userTaskPendingQueue,
103               userTaskFinishedQueue, summaryPendingQueue,
104               reportPendingQueue));
105           logger.info("Usuario creado");
106         }
107         logger.info("Usuarios creados");
108
109         // Actualizo la cola de estadisticas con la cantidad de usuarios
110         // actuales
111         summaryPendingQueue.put(new SummaryTask(Constants.DEFAULT_ID,
112             Constants.TASK_STATUS.SUBMITTED, currentUsers, null,
113             0));
114
115         userNumber = 0;
116
117         // Despierto a los users enviandoles un mensaje para que se
118         // pongan a trabajar
119         logger.info("Despertando los usuarios para un nuevo pulso");
120         for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : users
    QueuesList) {
121           logger.info("Despertando al usuario: " + userNumber);
122           pair.getFirst().put(new UserTask(Constants.DEFAULT_ID,
123               Constants.TASK_STATUS.SUBMITTED));
124           ++userNumber;
125         }
126         Thread.sleep(Constants.CONSOLE_TIMEOUT);
127       }
128       // Termino la simulacion. Tengo que parar a los usuarios
129
```



```
130       userNumber = 0;
131       // Despierto a los users enviandoles un mensaje de desconeccion
132       logger.info("Despertando los usuarios para desconectar");
133       for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : usersQu
    euesList) {
134         logger.info("Desconectando al usuario: " + userNumber);
135         pair.getFirst().put(new UserTask(Constants.DISCONNECT_ID,
136             Constants.TASK_STATUS.SUBMITTED));
137         ++userNumber;
138       }
139
140       userNumber = 0;
141       logger.info("Me quedo esperando a que los usuarios se desconecten");
142       for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : usersQu
    euesList) {
143         logger.info("Esperando al usuario: " + userNumber);
144         do {
145           userTask = pair.getSecond().take();
146         } while (userTask.getId() ≠ Constants.DISCONNECT_ID);
147         logger.info("Usuario: " + userNumber + " desconectado");
148         ++userNumber;
149       }
150
151       terminateReportThreads();
152
153     } catch (InterruptedException e) {
154       // TODO Auto-generated catch block
155       e.printStackTrace();
156     }
157
158   }
159
160   private void initReportThreads() {
161     // Reportes
162     summaryControllerThread.start();
163     summaryPrinterThread.start();
164     reportControllerThread.start();
165     monitorThread.start();
166   }
167
168   private void terminateReportThreads() throws InterruptedException {
169     logger.info("Parando el summary printer");
170     summaryTerminateSignal.terminate();
171     summaryPrinterThread.join();
172     logger.info("Summary printer finalizado");
173
174     logger.info("Parando el monitor");
175     reportTerminateSignal.terminate();
176     monitorThread.join();
177     logger.info("Monitor finalizado");
178
179     logger.info("Parando el Controlador de reportes");
180     reportPendingQueue.put(new ReportTask(Constants.DISCONNECT_ID,
181         Constants.TASK_STATUS.SUBMITTED, null, null));
182     reportFinishedQueue.take();
183     reportControllerThread.join();
184     logger.info("Controlador de reportes finalizado");
185
186     logger.info("Parando el Controlador de resumen");
187     summaryPendingQueue.put(new SummaryTask(Constants.DISCONNECT_ID,
188         Constants.TASK_STATUS.SUBMITTED, 0, false, 0));
189     summaryFinishedQueue.take();
190     summaryControllerThread.join();
191     logger.info("Controlador de resumen finalizado");
192   }
193
```

abr 08, 17 18:24 **LoadTestConsole.java** Page 4/4

```
194   private void initConsole() {
195     summaryPendingQueue = new ArrayBlockingQueue<SummaryTask>(
196         ConfigLoader.getInstance().getTasksQueueSize());
197     summaryFinishedQueue = new ArrayBlockingQueue<SummaryTask>(
198         ConfigLoader.getInstance().getTasksQueueSize());
199     reportPendingQueue = new ArrayBlockingQueue<ReportTask>(
200         ConfigLoader.getInstance().getTasksQueueSize());
201     reportFinishedQueue = new ArrayBlockingQueue<ReportTask>(
202         ConfigLoader.getInstance().getTasksQueueSize());
203
204     summary = new Summary();
205     report = new Report();
206     summaryTerminateSignal = new TerminateSignal();
207     reportTerminateSignal = new TerminateSignal();
208     summaryControllerThread = new Thread(new SummaryController(
209         summaryPendingQueue, summaryFinishedQueue, summary));
210     summaryPrinterThread = new Thread(new SummaryPrinter(summary,
211         summaryTerminateSignal, terminateSignal));
212     reportControllerThread = new Thread(new ReportController(
213         reportPendingQueue, reportFinishedQueue, report));
214     monitorThread = new Thread(
215         new Monitor(report, reportTerminateSignal));
216   }
217
218 }
```

abr 08, 17 18:24 **DownloaderTask.java** Page 1/1

```
1  package ar.fiuba.taller.loadTestConsole;
2
3  public class DownloaderTask extends Task {
4
5    private String method;
6    private String uri;
7    private String resourceType;
8
9    public DownloaderTask(Integer id, String method, String uri,
10       Constants.TASK_STATUS status, String resourceType) {
11     super(id, status);
12     this.method = method;
13     this.uri = uri;
14     this.setResourceType(resourceType);
15   }
16
17   public String getMethod() {
18     return method;
19   }
20
21   public void setMethod(String method) {
22     this.method = method;
23   }
24
25   public String getUri() {
26     return uri;
27   }
28
29   public void setUri(String uri) {
30     this.uri = uri;
31   }
32
33   public String getResourceType() {
34     return resourceType;
35   }
36
37   public void setResourceType(String resourceType) {
38     this.resourceType = resourceType;
39   }
40
41 }
```

```java
package ar.fiuba.taller.loadTestConsole;

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

import org.apache.log4j.Logger;

import ar.fiuba.taller.utils.HttpRequester;

public class Downloader implements Runnable {

  private ArrayBlockingQueue<DownloaderTask> downloaderTaskPendigQueue;
  private ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue;
  private ArrayBlockingQueue<SummaryTask> summaryQueue;
  private ArrayBlockingQueue<ReportTask> reportQueue;

  final static Logger logger = Logger.getLogger(App.class);

  public Downloader(
      ArrayBlockingQueue<DownloaderTask> downloaderTaskPendigQueue,
      ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue,
      ArrayBlockingQueue<SummaryTask> summaryQueue,
      ArrayBlockingQueue<ReportTask> reportQueue) {
    this.downloaderTaskPendigQueue = downloaderTaskPendigQueue;
    this.downloaderTaskFinishedQueue = downloaderTaskFinishedQueue;
    this.summaryQueue = summaryQueue;
    this.reportQueue = reportQueue;
  }

  public void run() {
    DownloaderTask task = null;
    Boolean gracefullQuit = false;
    Integer bytesDownloaded;
    long time_start, time_end, time_elapsed;
    HttpRequester httpRequester = new HttpRequester();

    logger.info("Iniciando un nuevo downloader");
    logger.info("Obteniendo una nueva task");
    try {
      while (¬gracefullQuit) {
        task = downloaderTaskPendigQueue.take();
        // Informo al monitor que arranco el downloader
        reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
            Constants.TASK_STATUS.SUBMITTED, false, null));
        logger.info("Task obtenida con los siguientes parametros:\n"
            + "Id: " + task.getId() + "\nStatus: "
            + task.getStatus() + "\nMethod: " + task.getMethod()
            + "\nUri: " + task.getUri());
        if (task.getId() ≡ Constants.DISCONNECT_ID) {
          logger.info(
              "Mensaje de desconexion. Se finaliza el thread.");
          logger.info("Confirmando finalizacion al user");
          downloaderTaskFinishedQueue.put(task);
          logger.info("Downloader finalizado");
          gracefullQuit = true;
        } else {
          logger.info("Nueva tarea recibida");
          logger.info("Descargando recurso...");
          task.setStatus(Constants.TASK_STATUS.EXECUTING);
          try {
            reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
                Constants.TASK_STATUS.EXECUTING, false,
                task.getResourceType()));
            time_start = System.currentTimeMillis();
            bytesDownloaded = httpRequester
                .doHttpRequest(task.getMethod(), task.getUri(),
                    new HashMap<String, String>(), "")
                .length();// download(task.getMethod(),
                      // task.getUri());
            time_end = System.currentTimeMillis();
            time_elapsed = time_end − time_start;
            logger.info("Bytes descargados: " + bytesDownloaded);
            logger.info("Tiempo inicial: " + time_start
                + " nanosgundos");
            logger.info(
                "Tiempo final: " + time_end + " nanosgundos");
            logger.info("Tiempo transcurrido: " + time_elapsed
                + " nanosegundos");
            summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID,
                Constants.TASK_STATUS.SUBMITTED, 0, true,
                time_elapsed));
            // Informo al monitor que arranco el usuario
            reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
                Constants.TASK_STATUS.FINISHED, false, null));
            task.setStatus(Constants.TASK_STATUS.FINISHED);
          } catch (Exception e) {
            task.setStatus(Constants.TASK_STATUS.FAILED);
            reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
                Constants.TASK_STATUS.FAILED, false, null));
            summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID,
                Constants.TASK_STATUS.SUBMITTED, 0, false, 0));
          } finally {
            downloaderTaskFinishedQueue.put(task);
          }
        }
      }
    } catch (InterruptedException e) {
      logger.warn("No se ha podido pushear tareas en las colas");
      e.printStackTrace();
    }
  }

}
```

abr 08, 17 18:24 **DotPrinter.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import ar.fiuba.taller.utils.TerminateSignal;

public class DotPrinter implements Runnable {
  private TerminateSignal signal;

  public DotPrinter(TerminateSignal signal) {
    this.signal = signal;
  }

  public void run() {
    while (¬signal.hasTerminate()) {
      System.out.print(".");
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }
  }
}
```

abr 08, 17 18:24 **ConstantUserPattern.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import java.util.List;

public class ConstantUserPattern extends UserPattern {

  public ConstantUserPattern(List<Integer> paramList) {

    super(paramList.get(0), paramList.get(0));
  }

  @Override
  public Integer getUsers(Integer tick) {
    return getNumberOfUsers() ≤ getUpperBound() ? getNumberOfUsers()
        : getUpperBound();
  }

}
```

abr 08, 17 18:24 **Constants.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

public final class Constants {
  public static final String PROPERTIES_FILE = "src/main/resources/configuration.properties";
  public static final String SCRIPT_FILE = "src/main/resources/script.xml";
  public static final String REPORT_FILE = "log/Report.txt";
  public static final String SIMULATION_TIME_PROPERTY = "simulationTime";
  public static final String USER_LOGIN_TIME_INTERVAL_PROPERTY = "userLoginTimeInterval";
  public static final String FUNCTION_PROPERTY = "function";
  public static final String NUMBER_OF_USERS_PROPERTY = "numberOfUsers";
  public static final String STEP_LENGTH_PROPERTY = "stepLength";
  public static final String USERES_QUEUE_SIZE = "usersQueueSize";
  public static final String MAX_SIZE_USER_POOL_THREAD = "maxSizeUserPoolThread";
  public static final String MAX_SIZE_DOWNLOADERS_POOL_THREAD = "maxSizeDownloadersPoolThread";
  public static final String TASKS_QUEUE_SIZE = "tasksQueueSize";
  public static final String GET_METHOD = "get";
  public static final String PUT_METHOD = "put";
  public static final String POST_METHOD = "post";
  public static final String SCRIPT_TAG = "script";
  public static final String LINK_TAG = "link";
  public static final String IMG_TAG = "img";
  public static final Integer DISCONNECT_ID = -1;
  public static final Integer DEFAULT_ID = 0;
  public static final Integer SUMMARY_PRINTER_TIMEOUT = 10000;
  public static final Integer MONITOR_TIMEOUT = 1000;
  public static final Integer CONSOLE_TIMEOUT = 1000;

  public static enum TASK_STATUS {
    SUBMITTED, EXECUTING, FINISHED, FAILED
  };
}
```

abr 08, 17 18:24 **ConfigLoader.java** Page 1/3

```java
package ar.fiuba.taller.loadTestConsole;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Properties;

import org.apache.log4j.Logger;

public class ConfigLoader {

  private Integer simulationTime;
  private Integer userLoginTimeInterval;
  private String function;
  private Integer usersQueueSize;
  private Integer maxsizeUserPoolThread;
  private Integer maxSizeDownloadersPoolThread;
  private Integer tasksQueueSize;
  // Array para guardar los parametros de la funcion
  private ArrayList<Integer> functionPatternParam;

  private static ConfigLoader instance = null;

  final static Logger logger = Logger.getLogger(App.class);

  protected ConfigLoader() {
    // TODO Auto-generated constructor stub
  }

  public static ConfigLoader getInstance() {
    if (instance == null) {
      instance = new ConfigLoader();
    }
    return instance;
  }

  public void init(String configFile) throws IOException {
    logger.info("Cargando configuracion del sistema");
    functionPatternParam = new ArrayList<Integer>();
    Properties properties = new Properties();
    FileInputStream input = new FileInputStream(
        Constants.PROPERTIES_FILE);

    // cargamos el archivo de propiedades
    properties.load(input);

    // obtenemos las propiedades
    function = properties.getProperty(Constants.FUNCTION_PROPERTY);
    usersQueueSize = Integer.parseInt(
        properties.getProperty(Constants.USERES_QUEUE_SIZE));
    maxsizeUserPoolThread = Integer.parseInt(properties
        .getProperty(Constants.MAX_SIZE_USER_POOL_THREAD));
    maxSizeDownloadersPoolThread = Integer.parseInt(properties
        .getProperty(Constants.MAX_SIZE_DOWNLOADERS_POOL_THREAD));
    tasksQueueSize = Integer.parseInt(
        properties.getProperty(Constants.TASKS_QUEUE_SIZE));
    functionPatternParam.add(Integer.parseInt(properties
        .getProperty(Constants.NUMBER_OF_USERS_PROPERTY)));

    if (function.equals("StairsUserPattern")) {
      functionPatternParam.add(Integer.parseInt(properties
          .getProperty(Constants.STEP_LENGTH_PROPERTY)));

      logger.info("Parametros cargados exitosamente:");
      logger.info("simulationTime: " + simulationTime);
```

```
67        logger.info("userLoginTimeInterval: " + userLoginTimeInterval);
68        logger.info("function: " + function);
69        logger.info("Parametros de la funcion " + function);
70        for (Integer param : functionPatternParam) {
71          logger.info(param);
72        }
73      }
74    }
75
76    public Integer getSimulationTime() {
77      return simulationTime;
78    }
79
80    public void setSimulationTime(Integer simulationTime) {
81      this.simulationTime = simulationTime;
82    }
83
84    public Integer getUserLoginTimeInterval() {
85      return userLoginTimeInterval;
86    }
87
88    public void setUserLoginTimeInterval(Integer userLoginTimeInterval) {
89      this.userLoginTimeInterval = userLoginTimeInterval;
90    }
91
92    public String getFunction() {
93      return function;
94    }
95
96    public void setFunction(String function) {
97      this.function = function;
98    }
99
100   public ArrayList<Integer> getFunctionPatternParam() {
101     return functionPatternParam;
102   }
103
104   public Integer getUsersQueueSize() {
105     return usersQueueSize;
106   }
107
108   public void setUsersQueueSize(Integer usersQueueSize) {
109     this.usersQueueSize = usersQueueSize;
110   }
111
112   public Integer getMaxsizeUserPoolThread() {
113     return maxsizeUserPoolThread;
114   }
115
116   public void setMaxsizeUserPoolThread(Integer maxsizeUserPoolThread) {
117     this.maxsizeUserPoolThread = maxsizeUserPoolThread;
118   }
119
120   public Integer getMaxSizeDownloadersPoolThread() {
121     return maxSizeDownloadersPoolThread;
122   }
123
124   public void setMaxSizeDownloadersPoolThread(
125       Integer maxSizeDownloadersPoolThread) {
126     this.maxSizeDownloadersPoolThread = maxSizeDownloadersPoolThread;
127   }
128
129   public Integer getTasksQueueSize() {
130     return tasksQueueSize;
131   }
132
```



```
133   public void setTasksQueueSize(Integer tasksQueuesListSize) {
134     this.tasksQueueSize = tasksQueuesListSize;
135   }
136
137 }
```

abr 08, 17 18:24 **App.java** Page 1/1

```java
package ar.fiuba.taller.loadTestConsole;

import java.io.IOException;

import org.apache.log4j.Logger;

import ar.fiuba.taller.utils.TerminateSignal;

public class App {
  final static Logger logger = Logger.getLogger(App.class);

  public static void main(String[] args)
      throws ClassNotFoundException, InstantiationException,
      IllegalAccessException, InterruptedException, IOException {
    TerminateSignal terminateSignal = new TerminateSignal();
    LoadTestConsole loadTestConsole = new LoadTestConsole(terminateSignal);

    logger.info("[*] Se inicia una nueva instancia de LoadTestConsole");
    Runtime.getRuntime().addShutdownHook(
        new Terminator(terminateSignal));

    loadTestConsole.start();

    logger.info("[*] Finaliza la instancia de LoadTestConsole");

  }
}
```

abr 08, 17 19:00 **Table of Content** Page 1/1

Table of Contents