

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;
4:
5: public class UserTask extends Task {
6:
7:     public UserTask(Integer id, TASK_STATUS status) {
8:         super(id, status);
9:         // TODO Auto-generated constructor stub
10:    }
11:
12: }
```

1

```

68:
69:
70:
71:
72:
nce();
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
FinishedQueue,
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
Se "
127:
nloaders");
128:
129:
130:

```

```

Map<String, String> map;
HttpRequester httpRequester = new HttpRequester();
String method = "", url = "", data = "";
File inputFile = new File(Constants.SCRIPT_FILE);
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = null;
try {
    dBuilder = dbFactory.newDocumentBuilder();
} catch (ParserConfigurationException e3) {
    // TODO Auto-generated catch block
    e3.printStackTrace();
}
org.w3c.dom.Document doc = null;
try {
    doc = dBuilder.parse(inputFile);
} catch (SAXException e3) {
    // TODO Auto-generated catch block
    e3.printStackTrace();
} catch (IOException e3) {
    // TODO Auto-generated catch block
    e3.printStackTrace();
}
doc.getDocumentElement().normalize();
NodeList nList = doc.getElementsByTagName("request");

logger.info("Creo la cola de tareas");
ArrayBlockingQueue<DownloaderTask> downloaderTaskPendingQueue =
    new ArrayBlockingQueue<DownloaderTask>(
        ConfigLoader.getInstance().getTasksQueueSize());

logger.info(
    "Creo la cola para recibir los mensajes de "
    + "terminado de los downloaders");
ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue =
    new ArrayBlockingQueue<DownloaderTask>(
        ConfigLoader.getInstance().getTasksQueueSize());

logger.info("Creo el pool de threads de downloaders");
ExecutorService downloadersThreadPool = Executors
    .newFixedThreadPool(downloaders);

logger.info("Lanzo " + downloaders
    + " downloaders y les paso las dos colas");
for (int i = 0; i < downloaders; i++) {
    logger.info("Lanzando el downloader: " + i);
    downloadersThreadPool.submit(new Downloader(
        downloaderTaskPendingQueue, downloaderTask
        summaryQueue, reportQueue));
}

while (!gracefullQuit) {
    try {
        userTask = userTaskPendingQueue.take();
        // Informo al monitor que arranco el usuario
        reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
            Constants.TASK_STATUS.SUBMITTED, true, null));
        if (userTask.getId() == Constants.DISCONNECT_ID) {
            logger.info(
                "Se ha recibido un mensaje de desconexion.
                + "enviando un mensaje de desconexion a los dow

        for (int i = 0; i < downloaders; ++i) {
            downloaderTaskPendingQueue
                .put(new DownloaderTask(Constants.

```

```

DISCONNECT_ID,
131:                                     null, null, null,
null));
132:                                     }
133:                                     logger.info("Esperando a que los downloaders finalicen");
134:                                     while (downloaders > 0) {
135:                                         downloaderTask = downloaderTaskFinishedQueue.take(
);
136:                                         if (downloaderTask.getId() == Constants.DISCONNECT
_ID) {
137:                                             downloaders--;
138:                                         }
139:                                     }
140:                                     logger.info("Downloaders finalizados");
141:                                     logger.info("Finalizando usuario");
142:                                     logger.info(
143:                                         "Avisando al Control principal que el usua
rio "
144:                                         + "termino");
145:                                     userTaskFinishedQueue.put(userTask);
146:                                     gracefullQuit = true;
147:                                     logger.info("Usuario finalizado");
148:                                 } else {
149:                                     logger.info("Se inicia un nuevo pulso de usuario");
150:                                     logger.info("Leo el script");
151:                                     try {
152:
153:                                     logger.info("Leo el script");
154:                                     for (int temp = 0; temp < nList.getLength(); temp++) {
155:                                         map = new HashMap<String, String>();
156:                                         method = "";
157:                                         url = "";
158:                                         data = "";
159:                                         taskId = 0;
160:
161:                                         // Obtengo la url
162:                                         org.w3c.dom.Node nNode = nList.item(temp);
163:                                         org.w3c.dom.Element eElement =
164:                                             (org.w3c.dom.Element) nNode;
165:                                         method = eElement.getElementsByTagName("method")
166:                                             .item(0).gettextContent();
167:                                         url = eElement.getElementsByTagName("url").item(0)
168:                                             .gettextContent();
169:                                         data = eElement.getElementsByTagName("data").item(0)
170:                                             .gettextContent();
171:                                         org.w3c.dom.Element e2 = (org.w3c.dom.Element) eElement
172:                                             .getElementsByTagName("headers").item(0);
173:                                         org.w3c.dom.NodeList n2 = e2.getChildNodes();
174:                                         for (int i = 0; i < n2.getLength(); i++) {
175:                                             Node n = n2.item(i);
176:                                             org.w3c.dom.NodeList n3 = n.getChildNodes();
177:                                             if (n3.getLength() > 0) {
178:                                                 if (!(n3.item(1).gettextContent().trim()
179:                                                     .equals(""))) {
180:                                                     map.put(n3.item(1).gettextContent(
),
181:                                                         n3.item(3).getText
Content());
182:                                                     }
183:                                                 }
184:                                             }
185:                                         }
186:                                     logger.info("Siguiete paso a realizar: " + method
187:                                         + " " + url);
188:                                     logger.info("Obteniendo recurso ...");
189:                                     html = httpRequester
190:                                         .doHttpRequest(method, url, map, data)

```

```

191:
192:                                     .toLowerCase();
193:                                     time_start = System.currentTimeMillis();
194:                                     bytesDownloaded = html.length();
195:                                     time_end = System.currentTimeMillis();
196:                                     time_elapsed = time_end - time_start;
197:
198:                                     logger.info(
199:                                         "Bytes descargados: " + bytesDownloaded);
200:                                     logger.info("Tiempo inicial: " + time_start
201:                                         + " nanosgundos");
202:                                     logger.info("Tiempo final: " + time_end
203:                                         + " nanosgundos");
204:                                     logger.info("Tiempo transcurrido: " + time_elapsed
205:                                         + " nanosgundos");
206:
207:                                     // Enviando estadistica de descarga a la cola de
208:                                     // estadisticas
209:                                     summaryQueue
210:                                         .put(new SummaryTask(Constants.DEFAULT_ID,
211:                                             Constants.TASK_STATUS.SUBM
212:                                                 ited, 0,
213:                                                 true, time_elapsed));
214:
215:                                     logger.info(
216:                                         "Rescato los tags LINK, IMG y SCRIPT e "
217:                                         + "inserto las tasks en la cola");
218:                                     for (String tag : Arrays.asList(Constants.IMG_TAG,
219:                                         Constants.SCRIPT_TAG, Constants.LINK_TAG))
220:
221:                                         logger.debug("Analizando el tag " + tag
222:                                             + "sobre el documento: html");
223:                                         requestList = getLinks(html, tag);
224:                                         logger.debug(
225:                                             "Recursos encontrados en la pagina
226:                                             + " analizada: "
227:                                             + requestList.size
228:
229:                                         for (String request : requestList) {
230:                                             logger.info(
231:                                                 "Enviando una nueva task c
232:                                                 + "parametros:\ntaskId: "
233:                                                 + taskId +
234:                                                 + Constant
235:                                                 + "\nReque
236:                                                 + "\nEstad
237:                                                 + Constant
238:
239:                                             downloaderTaskPendingQueue
240:                                                 .put(new DownloaderTask(ta
241:                                                     skId,
242:                                                     Constants.
243:                                                     request,
244:                                                     Constants.
245:                                                     tag));
246:
247:                                             ++taskId;
248:                                         }
249:                                     }
250:                                     // Informo al monitor que analice una url
251:                                     reportQueue.put(new ReportTask(Constants.DEFAULT_ID,

```

```
245: Constants.TASK_STATUS.EXECUTING, true, 306: throws URISyntaxException, IOException, BadLocationExcepti
246: null)); on {
247: 307: List<String> requestsList = new ArrayList<String>();
248: logger.info( 308: String currentAttribute = null;
249: "Esperando a que terminen los downloaders" 309:
310: // 311: if (tag.equals(Constants.LINK_TAG)) {
312: while (taskId > 0) { 313: currentAttribute = "href";
314: finishedTask = downloaderTaskFinishedQueue 315: } else if (tag.equals(Constants.SCRIPT_TAG)) {
316: .take(); 317: currentAttribute = "src";
318: taskId--; 319: } else if (tag.equals(Constants.IMG_TAG)) {
320: logger.info("Task finalizada:\nTaskId: " 321: currentAttribute = "src";
322: + finishedTask.getId() + "\nStatus 323: }
324: + finishedTask.getStatus()); 325: org.jsoup.nodes.Document doc = Jsoup.parse(page);
326: 327: Elements resource = doc.select(tag);
328: Iterator<Element> it = resource.iterator();
329:
330: while (it.hasNext()) {
331: requestsList.add(it.next().attr(currentAttribute));
332: }
333:
334: return requestsList;
335: }
336: }
337: }
338: }
339: }
340: }
341: }
342: }
343: }
344: }
345: }
346: }
347: }
348: }
349: }
350: }
351: }
352: }
353: }
354: }
355: }
356: }
357: }
358: }
359: }
360: }
361: }
362: }
363: }
364: }
365: }
366: }
367: }
368: }
369: }
370: }
371: }
372: }
373: }
374: }
375: }
376: }
377: }
378: }
379: }
380: }
381: }
382: }
383: }
384: }
385: }
386: }
387: }
388: }
389: }
390: }
391: }
392: }
393: }
394: }
395: }
396: }
397: }
398: }
399: }
400: }
401: }
402: }
403: }
404: }
405: }
406: }
407: }
408: }
409: }
410: }
411: }
412: }
413: }
414: }
415: }
416: }
417: }
418: }
419: }
420: }
421: }
422: }
423: }
424: }
425: }
426: }
427: }
428: }
429: }
430: }
431: }
432: }
433: }
434: }
435: }
436: }
437: }
438: }
439: }
440: }
441: }
442: }
443: }
444: }
445: }
446: }
447: }
448: }
449: }
450: }
451: }
452: }
453: }
454: }
455: }
456: }
457: }
458: }
459: }
460: }
461: }
462: }
463: }
464: }
465: }
466: }
467: }
468: }
469: }
470: }
471: }
472: }
473: }
474: }
475: }
476: }
477: }
478: }
479: }
480: }
481: }
482: }
483: }
484: }
485: }
486: }
487: }
488: }
489: }
490: }
491: }
492: }
493: }
494: }
495: }
496: }
497: }
498: }
499: }
500: }
501: }
502: }
503: }
504: }
505: }
506: }
507: }
508: }
509: }
510: }
511: }
512: }
513: }
514: }
515: }
516: }
517: }
518: }
519: }
520: }
521: }
522: }
523: }
524: }
525: }
526: }
527: }
528: }
529: }
530: }
531: }
532: }
533: }
534: }
535: }
536: }
537: }
538: }
539: }
540: }
541: }
542: }
543: }
544: }
545: }
546: }
547: }
548: }
549: }
550: }
551: }
552: }
553: }
554: }
555: }
556: }
557: }
558: }
559: }
560: }
561: }
562: }
563: }
564: }
565: }
566: }
567: }
568: }
569: }
570: }
571: }
572: }
573: }
574: }
575: }
576: }
577: }
578: }
579: }
580: }
581: }
582: }
583: }
584: }
585: }
586: }
587: }
588: }
589: }
590: }
591: }
592: }
593: }
594: }
595: }
596: }
597: }
598: }
599: }
600: }
601: }
602: }
603: }
604: }
605: }
606: }
607: }
608: }
609: }
610: }
611: }
612: }
613: }
614: }
615: }
616: }
617: }
618: }
619: }
620: }
621: }
622: }
623: }
624: }
625: }
626: }
627: }
628: }
629: }
630: }
631: }
632: }
633: }
634: }
635: }
636: }
637: }
638: }
639: }
640: }
641: }
642: }
643: }
644: }
645: }
646: }
647: }
648: }
649: }
650: }
651: }
652: }
653: }
654: }
655: }
656: }
657: }
658: }
659: }
660: }
661: }
662: }
663: }
664: }
665: }
666: }
667: }
668: }
669: }
670: }
671: }
672: }
673: }
674: }
675: }
676: }
677: }
678: }
679: }
680: }
681: }
682: }
683: }
684: }
685: }
686: }
687: }
688: }
689: }
690: }
691: }
692: }
693: }
694: }
695: }
696: }
697: }
698: }
699: }
700: }
701: }
702: }
703: }
704: }
705: }
706: }
707: }
708: }
709: }
710: }
711: }
712: }
713: }
714: }
715: }
716: }
717: }
718: }
719: }
720: }
721: }
722: }
723: }
724: }
725: }
726: }
727: }
728: }
729: }
730: }
731: }
732: }
733: }
734: }
735: }
736: }
737: }
738: }
739: }
740: }
741: }
742: }
743: }
744: }
745: }
746: }
747: }
748: }
749: }
750: }
751: }
752: }
753: }
754: }
755: }
756: }
757: }
758: }
759: }
760: }
761: }
762: }
763: }
764: }
765: }
766: }
767: }
768: }
769: }
770: }
771: }
772: }
773: }
774: }
775: }
776: }
777: }
778: }
779: }
780: }
781: }
782: }
783: }
784: }
785: }
786: }
787: }
788: }
789: }
790: }
791: }
792: }
793: }
794: }
795: }
796: }
797: }
798: }
799: }
800: }
801: }
802: }
803: }
804: }
805: }
806: }
807: }
808: }
809: }
810: }
811: }
812: }
813: }
814: }
815: }
816: }
817: }
818: }
819: }
820: }
821: }
822: }
823: }
824: }
825: }
826: }
827: }
828: }
829: }
830: }
831: }
832: }
833: }
834: }
835: }
836: }
837: }
838: }
839: }
840: }
841: }
842: }
843: }
844: }
845: }
846: }
847: }
848: }
849: }
850: }
851: }
852: }
853: }
854: }
855: }
856: }
857: }
858: }
859: }
860: }
861: }
862: }
863: }
864: }
865: }
866: }
867: }
868: }
869: }
870: }
871: }
872: }
873: }
874: }
875: }
876: }
877: }
878: }
879: }
880: }
881: }
882: }
883: }
884: }
885: }
886: }
887: }
888: }
889: }
890: }
891: }
892: }
893: }
894: }
895: }
896: }
897: }
898: }
899: }
900: }
901: }
902: }
903: }
904: }
905: }
906: }
907: }
908: }
909: }
910: }
911: }
912: }
913: }
914: }
915: }
916: }
917: }
918: }
919: }
920: }
921: }
922: }
923: }
924: }
925: }
926: }
927: }
928: }
929: }
930: }
931: }
932: }
933: }
934: }
935: }
936: }
937: }
938: }
939: }
940: }
941: }
942: }
943: }
944: }
945: }
946: }
947: }
948: }
949: }
950: }
951: }
952: }
953: }
954: }
955: }
956: }
957: }
958: }
959: }
960: }
961: }
962: }
963: }
964: }
965: }
966: }
967: }
968: }
969: }
970: }
971: }
972: }
973: }
974: }
975: }
976: }
977: }
978: }
979: }
980: }
981: }
982: }
983: }
984: }
985: }
986: }
987: }
988: }
989: }
990: }
991: }
992: }
993: }
994: }
995: }
996: }
997: }
998: }
999: }
1000: }
```

```
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: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.io.IOException;
4: import java.util.Arrays;
5:
6: import org.apache.log4j.Logger;
7:
8: import ar.fiuba.taller.utils.TerminateSignal;
9:
10: public class SummaryPrinter implements Runnable {
11:
12:     Summary summary;
13:     TerminateSignal terminateSignal;
14:     TerminateSignal globalTerminateSignal;
15:
16:     final static Logger logger = Logger.getLogger(App.class);
17:
18:     public SummaryPrinter(Summary summary, TerminateSignal terminateSignal,
19:         TerminateSignal globalTerminateSignal) {
20:         super();
21:         this.summary = summary;
22:         this.terminateSignal = terminateSignal;
23:         this.globalTerminateSignal = globalTerminateSignal;
24:     }
25:
26:     public void run() {
27:         logger.info("Iniciando SummaryPrinter");
28:         final String ANSI_CLS = "\u001b[2J";
29:         final String ANSI_HOME = "\u001b[H";
30:         while (!terminateSignal.hasTerminate()) {
31:             if (!globalTerminateSignal.hasTerminate()) {
32:                 // Limpio la pantalla
33:                 System.out.print(ANSI_CLS + ANSI_HOME);
34:                 System.out.flush();
35:
36:                 // Imprimo el resumen
37:                 System.out.println("Load Test Console: Resumen de
ejecucion");
38:                 System.out.println("-----");
39:                 System.out.println("Tiempo de descarga promedio...
40:                                     + summary.getAverageTime() + " ms"
41: );
42:                 System.out.println("Requests exitosos.....
43:                                     + summary.getSuccessfullrequest()
44:                                     + summary.getTotalRequests());
45:                 System.out.println("Requests fallidos.....
46:                                     + summary.getFailedrequest() + "/"
47:                                     + summary.getTotalRequests());
48:                 System.out.println("Cantidad de usuarios.....
49:                                     + summary.getUsers());
50:                 System.out.println("");
51:                 System.out.println("Presione ^C para terminar...")
52: ;
53:             }
54:             try {
55:                 Thread.sleep(10000);
56:             } catch (InterruptedException e) {
57:                 // TODO Auto-generated catch block
58:                 e.printStackTrace();
59:             }
60:         }
61:     }
62: }
```

```
logger.info("Finalizando SummaryPrinter");
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.io.IOException;
4: import java.io.PrintWriter;
5:
6: import org.apache.log4j.Logger;
7:
8: import ar.fiuba.taller.utils.TerminateSignal;
9:
10: public class Monitor implements Runnable {
11:
12:     Report report;
13:     TerminateSignal terminateSignal;
14:
15:     final static Logger logger = Logger.getLogger(App.class);
16:
17:     public Monitor(Report report, TerminateSignal terminateSignal) {
18:         super();
19:         this.report = report;
20:         this.terminateSignal = terminateSignal;
21:     }
22:
23:     public void run() {
24:         logger.info("Iniciando Monitor");
25:         while (!terminateSignal.hasTerminate()) {
26:
27:             try {
28:                 PrintWriter writer = new PrintWriter(Constants.REP
ORT_FILE,
29:                 "UTF-8");
30:                 writer.println("Load Test Console: Monitor de repo
rtes");
31:                 writer.println("-----");
32:                 writer.println("URLs analizadas.....");
33:                 + report.getAnalyzedUrl());
34:                 writer.println("SCRIPTS descargados.....");
35:                 + report.getDownloadedScripts());
36:                 writer.println("LINKS descargados.....");
37:                 + report.getDownloadedLinks());
38:                 writer.println("IMGs descargadas.....");
39:                 + report.getDownloadedImages());
40:                 writer.println("Hilos ejecutando script.....");
41:                 + report.getExecutionScriptThreads
());
42:                 writer.println("Hilos descargando recurso.....");
43:                 + report.getDownloadResourceThread
s());
44:                 writer.close();
45:             } catch (IOException e) {
46:                 // do something
47:             }
48:
49:             try {
50:                 Thread.sleep(1000);
51:             } catch (InterruptedException e) {
52:                 // TODO Auto-generated catch block
53:                 e.printStackTrace();
54:             }
55:         }
56:         logger.info("Finalizando Monitor");
57:     }
58:
59: }
```

```

1: package ar.fiuba.taller.loadTestConsole;
2:
3: import ar.fiuba.taller.utils.*;
4: import java.util.ArrayList;
5: import java.util.List;
6: import java.util.concurrent.ArrayBlockingQueue;
7: import java.util.concurrent.BlockingQueue;
8: import java.util.concurrent.ExecutorService;
9: import java.util.concurrent.Executors;
10: import java.util.concurrent.TimeUnit;
11:
12: import org.apache.log4j.Logger;
13:
14: public class LoadTestConsole implements Runnable {
15:
16:     private Integer maxSizeUserPoolThread;
17:     private String function;
18:     private List<Integer> functionParamList;
19:     private UserPattern userPattern;
20:     private List<Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>>> users
QueuesList;
21:     private TerminateSignal terminateSignal;
22:
23:     final static Logger logger = Logger.getLogger(App.class);
24:
25:     public LoadTestConsole(TerminateSignal terminateSignal) {
26:         this.terminateSignal = terminateSignal;
27:         ConfigLoader.getInstance().init(Constants.PROPERTIES_FILE);
28:         usersQueuesList = new ArrayList<Pair<BlockingQueue<UserTask>, Bloc
kingQueue<UserTask>>>();
29:         function = ConfigLoader.getInstance().getFunction();
30:         functionParamList = ConfigLoader.getInstance().
31:             .getFunctionPatternParam();
32:         maxSizeUserPoolThread = ConfigLoader.getInstance().
33:             .getMaxsizeUserPoolThread();
34:
35:         // TODO: Arreglar esto
36:         if (function.equals("ConstantUserPattern")) {
37:             userPattern = new ConstantUserPattern(functionParamList);
38:         } else if (function.equals("StairsUserPattern")) {
39:             userPattern = new StairsUserPattern(functionParamList,
40:                 ConfigLoader.getInstance().getMaxsizeUserP
oolThread());
41:         } else if (function.equals("RampUserPattern")) {
42:             userPattern = new RampUserPattern(functionParamList,
43:                 ConfigLoader.getInstance().getMaxsizeUserP
oolThread());
44:         }
45:
46:     }
47:
48:     public void run() {
49:         ArrayBlockingQueue<UserTask> userTaskPendingQueue;
50:         ArrayBlockingQueue<UserTask> userTaskFinishedQueue;
51:         ArrayBlockingQueue<SummaryTask> summaryPendingQueue = new ArrayBlo
ckingQueue<SummaryTask>(
52:             ConfigLoader.getInstance().getTasksQueueSize());
53:         ArrayBlockingQueue<SummaryTask> summaryFinishedQueue = new ArrayBl
ockingQueue<SummaryTask>(
54:             ConfigLoader.getInstance().getTasksQueueSize());
55:         ArrayBlockingQueue<ReportTask> reportPendingQueue = new ArrayBlock
ingQueue<ReportTask>(
56:             ConfigLoader.getInstance().getTasksQueueSize());
57:         ArrayBlockingQueue<ReportTask> reportFinishedQueue = new ArrayBloc
kingQueue<ReportTask>(
58:             ConfigLoader.getInstance().getTasksQueueSize());
59:
60:         Summary summary = new Summary();
61:         Report report = new Report();
62:         TerminateSignal summaryTerminateSignal = new TerminateSignal();
63:         TerminateSignal reportTerminateSignal = new TerminateSignal();
64:         Thread summaryControllerThread = new Thread(new SummaryController(
65:             summaryPendingQueue, summaryFinishedQueue, summary
));
66:         Thread summaryPrinterThread = new Thread(new SummaryPrinter(summar
y,
67:             summaryTerminateSignal, terminateSignal));
68:         Thread reportControllerThread = new Thread(new ReportController(
69:             reportPendingQueue, reportFinishedQueue, report));
70:         Thread monitorThread = new Thread(
71:             new Monitor(report, reportTerminateSignal));
72:
73:         Integer currentUsers = 0, deltaUseres = 0, tick = 0, userNumber =
0;
74:         UserTask userTask;
75:         SummaryTask summaryTask;
76:         ReportTask reportTask;
77:
78:         logger.info("Se inicia una nueva instancia de LoadTestConsole");
79:         logger.info("Creando el pool de threads de usuarios");
80:         ExecutorService usersThreadPool = Executors
81:             .newFixedThreadPool(maxSizeUserPoolThread);
82:
83:         // Reportes
84:         summaryControllerThread.start();
85:         summaryPrinterThread.start();
86:         reportControllerThread.start();
87:         monitorThread.start();
88:
89:         try {
90:             while (!terminateSignal.hasTerminate()) {
91:                 tick++;
92:                 deltaUseres = userPattern.getUsers(tick) - current
Users;
93:                 logger.info("Se inicia el pulso: " + tick);
94:                 logger.info("Cantidad de usuarios actualmente corr
iendo: "
95:                     + currentUsers);
96:                 logger.info("
97:                     "Cantidad de usuarios que deben co
rrer en este pulso: "
98:                     + userPattern.getU
sers(tick));
99:                 logger.info("Cantidad de usuarios que deben ingres
ar: "
100:                     + deltaUseres);
101:
102:                 for (int i = 0; i < deltaUseres; i++) {
103:                     currentUsers++;
104:                     logger.info("Creando el usuario: " + curre
ntUsers);
105:                     // Creo las colas para el nuevo usuario
106:                     userTaskPendingQueue = new ArrayBlockingQu
eue<UserTask>(
107:                         ConfigLoader.getInstance().
108:                             getTasksQueueSize());
109:                     userTaskFinishedQueue = new ArrayBlockingQu
eue<UserTask>(
110:                         ConfigLoader.getInstance().
111:                             getTasksQueueSize());
112:                     // Me guardo las dos colas en la lista de
113:                     usersQueuesList

```



```

113:                                     .add(new Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> (
114:                                     userTaskPendingQueue, userTaskFinishedQueue));
115:                                     // Creo el usuario y les paso las colas para que puedan
116:                                     // insertar tareas
117:                                     usersThreadPool.submit(new User(userTaskPendingQueue, userTaskFinishedQueue, summaryPendingQueue, DEFAULT_ID, currentUsers, null, userNumber = 0;
118:                                     // Despierto a los users enviandoles un mensaje para que se
119:                                     // pongan a trabajar
120:                                     logger.info("Despertando los usuarios para un nuevo pulso");
121:                                     for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : usersQueuesList) {
122:                                     logger.info("Despertando al usuario: " + userNumber);
123:                                     pair.getFirst().put(new UserTask(Constants.DEFAULT_ID, Constants.TASK_STATUS.SUBMITTED, null, null, reportTask, summaryTask));
124:                                     ++userNumber;
125:                                     userNumber = 0;
126:                                     logger.info("Me quedo esperando a que los usuarios terminen sus tareas");
127:                                     for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : usersQueuesList) {
128:                                     logger.info("Esperando al usuario: " + userNumber);
129:                                     userTask = pair.getSecond().take();
130:                                     logger.info("Usuario: " + userNumber + " finalizado");
131:                                     ++userNumber;
132:                                     }
133:                                     // Termino la simulacion. Tengo que parar a los usuarios
134:                                     userNumber = 0;
135:                                     // Despierto a los users enviandoles un mensaje de desconectar
136:                                     logger.info("Despertando los usuarios para desconectar");
137:                                     for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>> pair : usersQueuesList) {
138:                                     logger.info("Esperando al usuario: " + userNumber);
139:                                     userTask = pair.getSecond().take();
140:                                     logger.info("Usuario: " + userNumber + " desconectado");
141:                                     ++userNumber;
142:                                     }
143:                                     // Reportes
144:                                     logger.info("Parando el summary printer");
145:                                     summaryTerminateSignal.terminate();
146:                                     summaryPrinterThread.join();
147:                                     logger.info("Summary printer finalizado");
148:                                     logger.info("Parando el monitor");
149:                                     reportTerminateSignal.terminate();
150:                                     monitorThread.join();
151:                                     logger.info("Monitor finalizado");
152:                                     logger.info("Parando el Controlador de reportes");
153:                                     reportPendingQueue.put(new ReportTask(Constants.DISCONNECT_ID, Constants.TASK_STATUS.SUBMITTED, null, null, reportTask));
154:                                     reportTask = reportFinishedQueue.take();
155:                                     reportControllerThread.join();
156:                                     logger.info("Controlador de reportes finalizado");
157:                                     logger.info("Parando el Controlador de resumen");
158:                                     summaryPendingQueue.put(new SummaryTask(Constants.DISCONNECT_ID, Constants.TASK_STATUS.SUBMITTED, 0, false, summaryTask));
159:                                     summaryTask = summaryFinishedQueue.take();
160:                                     summaryControllerThread.join();
161:                                     logger.info("Controlador de resumen finalizado");
162:                                     } catch (InterruptedException e) {
163:                                     // TODO Auto-generated catch block
164:                                     e.printStackTrace();
165:                                     }
166:                                     public String getFunction() {
167:                                     return function;
168:                                     }

```

```
215:
216:     public void setFunction(String function) {
217:         this.function = function;
218:     }
219:
220:     public List<Integer> getFunctionParamList() {
221:         return functionParamList;
222:     }
223:
224:     public void setFunctionParamList(ArrayList<Integer> functionParamList) {
225:         this.functionParamList = functionParamList;
226:     }
227:
228: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: public class Report {
4:
5:     private Integer analyzedUrl;
6:     private Integer downloadedScripts;
7:     private Integer downloadedLinks;
8:     private Integer downloadedImages;
9:     private Integer executionScriptThreads;
10:    private Integer downloadResourceThreads;
11:
12:    public synchronized Integer getAnalyzedUrl() {
13:        return analyzedUrl;
14:    }
15:
16:    public synchronized void incAnalyzedUrl() {
17:        analyzedUrl++;
18:    }
19:
20:    public synchronized void decAnalyzedUrl() {
21:        analyzedUrl--;
22:    }
23:
24:    public synchronized Integer getDownloadedScripts() {
25:        return downloadedScripts;
26:    }
27:
28:    public synchronized void incDownloadedScripts() {
29:        downloadedScripts++;
30:    }
31:
32:    public synchronized void decDownloadedScripts() {
33:        downloadedScripts--;
34:    }
35:
36:    public synchronized Integer getDownloadedLinks() {
37:        return downloadedLinks;
38:    }
39:
40:    public synchronized void incDownloadedLinks() {
41:        downloadedLinks++;
42:    }
43:
44:    public synchronized void decDownloadedLinks() {
45:        downloadedLinks--;
46:    }
47:
48:    public synchronized Integer getDownloadedImages() {
49:        return downloadedImages;
50:    }
51:
52:    public synchronized void incDownloadedImages() {
53:        downloadedImages++;
54:    }
55:
56:    public synchronized void decDownloadedImages() {
57:        downloadedImages--;
58:    }
59:
60:    public synchronized Integer getExecutionScriptThreads() {
61:        return executionScriptThreads;
62:    }
63:
64:    public synchronized void incExecutionScriptThreads() {
65:        executionScriptThreads++;
66:    }
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: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import org.apache.log4j.Logger;
4:
5: import ar.fiuba.taller.utils.TerminateSignal;
6:
7: public class App {
8:     final static Logger logger = Logger.getLogger(App.class);
9:
10:    public static void main(String[] args)
11:        throws ClassNotFoundException, InstantiationException,
12:        IllegalAccessException, InterruptedException {
13:        TerminateSignal terminateSignal = new TerminateSignal();
14:        Thread loadTestConsoleThread = new Thread(
15:            new LoadTestConsole(terminateSignal));
16:
17:        logger.info("[*] Se inicia una nueva instancia de LoadTestConsole"
);
18:        Runtime.getRuntime().addShutdownHook(
19:            new Terminator(loadTestConsoleThread, terminateSig
nal));
20:        //
21:        loadTestConsoleThread.start();
22:        // Thread.sleep(1200000);
23:        // terminateSignal.terminate();
24:        //
25:        // System.out.println("Presione ^c para terminar.");
26:
27:        logger.info("[*] Finaliza la instancia de LoadTestConsole");
28:
29:    }
30: }
```

```

1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.io.FileInputStream;
4: import java.io.IOException;
5: import java.util.ArrayList;
6: import java.util.Properties;
7:
8: import org.apache.log4j.Logger;
9:
10: public class ConfigLoader {
11:
12:     private Integer simulationTime;
13:     private Integer userLoginTimeInterval;
14:     private String function;
15:     private Integer usersQueueSize;
16:     private Integer maxSizeThreadPool;
17:     private Integer maxSizeDownloadersThreadPool;
18:     private Integer tasksQueueSize;
19:     // Array para guardar los parametros de la funcion
20:     private ArrayList<Integer> functionPatternParam;
21:
22:     private static ConfigLoader instance = null;
23:
24:     final static Logger logger = Logger.getLogger(App.class);
25:
26:     protected ConfigLoader() {
27:         // TODO Auto-generated constructor stub
28:     }
29:
30:     public static ConfigLoader getInstance() {
31:         if (instance == null) {
32:             instance = new ConfigLoader();
33:         }
34:         return instance;
35:     }
36:
37:     public void init(String configFile) {
38:         try {
39:             logger.info("Cargando configuracion del sistema");
40:             functionPatternParam = new ArrayList<Integer>();
41:             Properties properties = new Properties();
42:             FileInputStream input = new FileInputStream(
43:                 Constants.PROPERTIES_FILE);
44:
45:             // cargamos el archivo de propiedades
46:             properties.load(input);
47:
48:             // obtenemos las propiedades
49:             function = properties.getProperty(Constants.FUNCTION_PROPE
RTY);
50:             usersQueueSize = Integer.parseInt(
51:                 properties.getProperty(Constants.USERES_QUEUE_SIZE));
52:             maxSizeThreadPool = Integer.parseInt(properties
53:                 .getProperty(Constants.MAX_SIZE_USER_POOL_
THREAD));
54:             maxSizeDownloadersThreadPool = Integer.parseInt(properties
55:                 .getProperty(Constants.MAX_SIZE_DOWNLOADER
S_POOL_THREAD));
56:             tasksQueueSize = Integer.parseInt(
57:                 properties.getProperty(Constants.TASKS_QUEUE_SIZE));
58:             functionPatternParam.add(Integer.parseInt(properties
59:                 .getProperty(Constants.NUMBER_OF_USERS_PROPERT
Y)));
60:
61:             if (function.equals("StairsUserPattern")) {

```

```

62:                                     functionPatternParam.add(Integer.parseInt(property
es
63:                                     ).getProperty(Constants.STEP_LENGTH
_PROPERTY));
64:                                     }
65:                                 } catch (IOException ex) {
66:                                     logger.error(ex.toString());
67:                                     ex.printStackTrace();
68:                                 }
69:
70:                                     logger.info("Parametros cargados exitosamente.");
71:                                     logger.info("simulationTime: " + simulationTime);
72:                                     logger.info("userLoginTimeInterval: " + userLoginTimeInterval);
73:                                     logger.info("function: " + function);
74:                                     logger.info("Parametros de la funcion " + function);
75:                                     for (Integer param : functionPatternParam) {
76:                                         logger.info(param);
77:                                     }
78:
79:                     }
80:
81:     public Integer getSimulationTime() {
82:         return simulationTime;
83:     }
84:
85:     public void setSimulationTime(Integer simulationTime) {
86:         this.simulationTime = simulationTime;
87:     }
88:
89:     public Integer getUserLoginTimeInterval() {
90:         return userLoginTimeInterval;
91:     }
92:
93:     public void setUserLoginTimeInterval(Integer userLoginTimeInterval) {
94:         this.userLoginTimeInterval = userLoginTimeInterval;
95:     }
96:
97:     public String getFunction() {
98:         return function;
99:     }
100:
101:     public void setFunction(String function) {
102:         this.function = function;
103:     }
104:
105:     public ArrayList<Integer> getFunctionPatternParam() {
106:         return functionPatternParam;
107:     }
108:
109:     public Integer getUsersQueueSize() {
110:         return usersQueueSize;
111:     }
112:
113:     public void setUsersQueueSize(Integer usersQueueSize) {
114:         this.usersQueueSize = usersQueueSize;
115:     }
116:
117:     public Integer getMaxsizeUserPoolThread() {
118:         return maxsizeUserPoolThread;
119:     }
120:
121:     public void setMaxsizeUserPoolThread(Integer maxsizeUserPoolThread) {
122:         this.maxsizeUserPoolThread = maxsizeUserPoolThread;
123:     }
124:
125:     public Integer getMaxSizeDownloadersPoolThread() {
126:         return maxSizeDownloadersPoolThread;

```

```
127:     }
128:
129:     public void setMaxSizeDownloadersPoolThread(
130:         Integer maxSizeDownloadersPoolThread) {
131:         this.maxSizeDownloadersPoolThread = maxSizeDownloadersPoolThread;
132:     }
133:
134:     public Integer getTasksQueueSize() {
135:         return tasksQueueSize;
136:     }
137:
138:     public void setTasksQueueSize(Integer tasksQueuesListSize) {
139:         this.tasksQueueSize = tasksQueuesListSize;
140:     }
141:
142: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import org.apache.log4j.Logger;
4:
5: import ar.fiuba.taller.utils.TerminateSignal;
6:
7: public class Terminator extends Thread {
8:     private Thread loadTestConsoleThread;
9:     private TerminateSignal terminateSignal;
10:     final static Logger logger = Logger.getLogger(App.class);
11:
12:     public Terminator(Thread loadTestConsoleThread,
13:         TerminateSignal terminateSignal) {
14:         this.loadTestConsoleThread = loadTestConsoleThread;
15:         this.terminateSignal = terminateSignal;
16:     }
17:
18:     public void run() {
19:         try {
20:             TerminateSignal dotSignal = new TerminateSignal();
21:             Thread dotPrinterThread = new Thread(new DotPrinter(dotSig
nal));
22:             logger.info("Finalizando la simulacion...");
23:             System.out.print("Finalizando la simulacion");
24:             dotPrinterThread.start();
25:             terminateSignal.terminate();
26:             loadTestConsoleThread.join();
27:             dotSignal.terminate();
28:             dotPrinterThread.join();
29:             logger.info("Simulacion Finalizada");
30:             System.out.println("");
31:             System.out.println("Simulacion Finalizada");
32:         } catch (InterruptedException e) {
33:             // TODO Auto-generated catch block
34:             e.printStackTrace();
35:         }
36:     }
37:
38: }
```

```

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:     ArrayBlockingQueue<ReportTask> pendingReportQueue;
10:    ArrayBlockingQueue<ReportTask> finishedReportQueue;
11:    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: " + r
eportTask.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() == Co
nstants.TASK_STATUS.SUBMITTED) {
41:                            logger.debug("INCREMENTO T
HREAD");
42:                            report.incExecutionScriptT
hread();
43:                        } else if (reportTask
44:                            .getStatus() == Co
nstants.TASK_STATUS.EXECUTING) {
45:                            logger.debug("INCREMENTO U
RL");
46:                            report.incAnalyzedUrl();
47:                        } else {
48:                            logger.debug("DECREMENTO T
HREAD");
49:                            report.decExecutionScriptT
hread();
50:                        }
51:                    } else { // Es una task enviada por un dow
nloader
52:
53:                        if (reportTask
54:                            .getStatus() == Co
nstants.TASK_STATUS.SUBMITTED) {
55:                            report.incDownloadResource
Threads();
56:                        } else if (reportTask
57:                            .getStatus() == Co
nstants.TASK_STATUS.EXECUTING) {
58:                            if (reportTask.getResource
59:                                .equals(Co
nstants.SCRIPT_TAG)) {
60:                                report.incDownload
Scripts();
61:                            } else if (reportTask.getR
62:                                .equals(Co
nstants.LINK_TAG)) {
63:                                report.incDownload
edLinks();
64:                            } else if (reportTask.getR
65:                                .equals(Co
nstants.IMG_TAG)) {
66:                                report.incDownload
edImages();
67:                            } else {
68:                                report.decDownloadResource
Threads();
69:                            }
70:                        }
71:                        logger.info("Reporte actualizado");
72:                    } while (reportTask.getId() != Constants.DISCONNECT_ID);
73:                    logger.info(
74:                        "Finalizando el controller. "
75:                        + "Enviando mensaje de finalizacion al cont
rol principal.");
76:
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:        }

```



```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: public abstract class UserPattern {
4:
5:     private Integer numberOfUsers;
6:     private Integer upperBound;
7:
8:     public UserPattern(Integer numberOfUsers, Integer upperBound) {
9:         this.setNumberOfUsers(numberOfUsers);
10:        this.setUpperBound(upperBound);
11:    }
12:
13:    public abstract Integer getUsers(Integer tick);
14:
15:    public Integer getNumberOfUsers() {
16:        return numberOfUsers;
17:    }
18:
19:    public void setNumberOfUsers(Integer numberOfUsers) {
20:        this.numberOfUsers = numberOfUsers;
21:    }
22:
23:    public Integer getUpperBound() {
24:        return upperBound;
25:    }
26:
27:    public void setUpperBound(Integer upperBound) {
28:        this.upperBound = upperBound;
29:    }
30:
31: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;
4:
5: public class SummaryTask extends Task {
6:     private Integer usersAmount;
7:     private Boolean successfullRequest;
8:     private long timeElapsed;
9:
10:    public Integer getUsersAmount() {
11:        return usersAmount;
12:    }
13:
14:    public void setUsersAmount(Integer usersAmount) {
15:        this.usersAmount = usersAmount;
16:    }
17:
18:    public long getTimeElapsed() {
19:        return timeElapsed;
20:    }
21:
22:    public void setTimeElapsed(long timeElapsed) {
23:        this.timeElapsed = timeElapsed;
24:    }
25:
26:    public SummaryTask(Integer id, TASK_STATUS status, Integer usersAmount,
27:        Boolean successfullRequest, long timeElapsed) {
28:        super(id, status);
29:        this.usersAmount = usersAmount;
30:        this.successfullRequest = successfullRequest;
31:        this.timeElapsed = timeElapsed;
32:    }
33:
34:    public Boolean getSuccessfullRequest() {
35:        return successfullRequest;
36:    }
37:
38:    public void setSuccessfullRequest(Boolean successfullRequest) {
39:        this.successfullRequest = successfullRequest;
40:    }
41:
42: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: public final class Constants {
4:     public static final String PROPERTIES_FILE = "src/main/resources/configura
tion.properties";
5:     public static final String SCRIPT_FILE = "src/main/resources/script.xml";
6:     public static final String REPORT_FILE = "log/Report.txt";
7:     public static final String SIMULATION_TIME_PROPERTY = "simulationTime";
8:     public static final String USER_LOGIN_TIME_INTERVAL_PROPERTY = "userLoginT
imeInterval";
9:     public static final String FUNCTION_PROPERTY = "function";
10:    public static final String NUMBER_OF_USERS_PROPERTY = "numberOfUsers";
11:    public static final String STEP_LENGTH_PROPERTY = "stepLength";
12:    public static final String USERES_QUEUE_SIZE = "usersQueueSize";
13:    public static final String MAX_SIZE_USER_POOL_THREAD = "maxSizeUserPoolThr
ead";
14:    public static final String MAX_SIZE_DOWNLOADERS_POOL_THREAD = "maxSizeDown
loadersPoolThread";
15:    public static final String TASKS_QUEUE_SIZE = "tasksQueueSize";
16:    public static final String GET_METHOD = "get";
17:    public static final String PUT_METHOD = "put";
18:    public static final String POST_METHOD = "post";
19:    public static final String SCRIPT_TAG = "script";
20:    public static final String LINK_TAG = "link";
21:    public static final String IMG_TAG = "img";
22:    public static final Integer DISCONNECT_ID = -1;
23:    public static final Integer DEFAULT_ID = 0;
24:
25:    public static enum TASK_STATUS {
26:        SUBMITTED, EXECUTING, FINISHED, FAILED
27:    };
28: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import ar.fiuba.taller.utils.TerminateSignal;
4:
5: public class DotPrinter implements Runnable {
6:     private TerminateSignal signal;
7:
8:     public DotPrinter(TerminateSignal signal) {
9:         this.signal = signal;
10:    }
11:
12:    public void run() {
13:        while (!signal.hasTerminate()) {
14:            System.out.print(".");
15:            try {
16:                Thread.sleep(1000);
17:            } catch (InterruptedException e) {
18:                // TODO Auto-generated catch block
19:                e.printStackTrace();
20:            }
21:        }
22:    }
23: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.util.concurrent.BlockingQueue;
4:
5: import org.apache.log4j.Logger;
6:
7: public class UsersControl implements Runnable {
8:
9:     private BlockingQueue useresQueue;
10:    private Integer simulationTime;
11:    final static Logger logger = Logger.getLogger(App.class);
12:
13:    public UsersControl(Integer simulationTime) {
14:        this.setSimulationTime(simulationTime);
15:    }
16:
17:    public void run() {
18:        // TODO Auto-generated method stub
19:    }
20:
21:
22:    public Integer getSimulationTime() {
23:        return simulationTime;
24:    }
25:
26:    public void setSimulationTime(Integer simulationTime) {
27:        this.simulationTime = simulationTime;
28:    }
29:
30: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.util.List;
4:
5: public class ConstantUserPattern extends UserPattern {
6:
7:     public ConstantUserPattern(List<Integer> paramList) {
8:
9:         super(paramList.get(0), paramList.get(0));
10:    }
11:
12:    @Override
13:    public Integer getUsers(Integer tick) {
14:        return getNumberOfUsers() <= getUpperBound() ? getNumberOfUsers()
15:            : getUpperBound();
16:    }
17:
18: }
```

```

1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.io.BufferedReader;
4: import java.io.InputStreamReader;
5: import java.net.HttpURLConnection;
6: import java.net.URL;
7: import java.util.HashMap;
8: import java.util.concurrent.ArrayBlockingQueue;
9:
10: import org.apache.log4j.Logger;
11:
12: import ar.fiuba.taller.utils.HttpRequester;
13:
14: public class Downloader implements Runnable {
15:
16:     private ArrayBlockingQueue<DownloaderTask> downloaderTaskPendigQueue;
17:     private ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue;
18:     private ArrayBlockingQueue<SummaryTask> summaryQueue;
19:     private ArrayBlockingQueue<ReportTask> reportQueue;
20:
21:     final static Logger logger = Logger.getLogger(App.class);
22:
23:     public Downloader(
24:         ArrayBlockingQueue<DownloaderTask> downloaderTaskPendigQueue,
25:         ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQueue,
26:         ArrayBlockingQueue<SummaryTask> summaryQueue,
27:         ArrayBlockingQueue<ReportTask> reportQueue) {
28:         this.downloaderTaskPendigQueue = downloaderTaskPendigQueue;
29:         this.downloaderTaskFinishedQueue = downloaderTaskFinishedQueue;
30:         this.summaryQueue = summaryQueue;
31:         this.reportQueue = reportQueue;
32:     }
33:
34:     public void run() {
35:         DownloaderTask task = null;
36:         Boolean gracefullQuit = false;
37:         Integer bytesDownloaded;
38:         long time_start, time_end, time_elapsed;
39:         HttpRequester httpRequester = new HttpRequester();
40:
41:         logger.info("Iniciando un nuevo downloader");
42:         logger.info("Obteniendo una nueva task");
43:         try {
44:             while (!gracefullQuit) {
45:                 task = downloaderTaskPendigQueue.take();
46:                 // Informo al monitor que arranco el downloader
47:                 reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
48:                     Constants.TASK_STATUS.SUBMITTED, false, null));
49:                 logger.info("Task obtenida con los siguientes para
50:                     metros:\n"
51:                     + "Id: " + task.getId() + "\nStatus: "
52:                     + task.getStatus() + "\nMethod: "
53:                     + task.getMethod() + "\nUri: " + task.getUri());
54:                 if (task.getId() == Constants.DISCONNECT_ID) {
55:                     logger.info("Mensaje de desconexion. Se finaliza el thread.");
56:                 }
57:                 downloaderTaskFinishedQueue.put(task);
58:                 logger.info("Downloader finalizado");
59:
60:                 gracefullQuit = true;
61:             } else {
62:                 logger.info("Nueva tarea recibida");
63:                 logger.info("Descargando recurso...");
64:                 task.setStatus(Constants.TASK_STATUS.EXECUTING);
65:                 try {
66:                     reportQueue.put(new ReportTask(Constants.TASK_STATUS.EXECUTING, false, task.getResourceType(),
67:                         time_start = System.currentTimeMillis(), bytesDownloaded = httpRequester.doHttpRequest(task.getUri(),
68:                             new HashMap<String, String>(), "")));
69:                     time_end = System.currentTimeMillis();
70:                     time_elapsed = time_end - time_start;
71:                     logger.info("Bytes descargados: " + bytesDownloaded);
72:                     logger.info("Tiempo inicial: " + time_start + " nanosgundos");
73:                     logger.info("Tiempo final: " + time_end + " nanosgundos");
74:                     logger.info("Tiempo transcurrido: " + time_elapsed + " nanosegundos");
75:                     summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID, Constants.TASK_STATUS.FINISHED, 0, true, time_elapsed));
76:                     // Informo al monitor que arranco
77:                     reportQueue.put(new ReportTask(Constants.TASK_STATUS.FINISHED, false, null));
78:                     task.setStatus(Constants.TASK_STATUS.FAILED);
79:                     reportQueue.put(new ReportTask(Constants.TASK_STATUS.FAILED, false, null));
80:                     summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID, Constants.TASK_STATUS.FAILED, 0, false, 0));
81:                 } catch (Exception e) {
82:                     task.setStatus(Constants.TASK_STATUS.FAILED);
83:                     reportQueue.put(new ReportTask(Constants.TASK_STATUS.FAILED, false, null));
84:                     summaryQueue.put(new SummaryTask(Constants.DEFAULT_ID, Constants.TASK_STATUS.FAILED, 0, false, 0));
85:                 } finally {
86:                     downloaderTaskFinishedQueue.put(task);
87:                 }
88:             }
89:         }
90:     }
91: }

```

```
98:         }
99:     }
100: }
101: } catch (InterruptedException e) {
102:     // TODO Auto-generated catch block
103:     e.printStackTrace();
104: }
105: }
106:
107: }
```



```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: public abstract class Task {
4:
5:     private Integer id;
6:     private Constants.TASK_STATUS status;
7:
8:     public Task(Integer id, Constants.TASK_STATUS status) {
9:         this.id = id;
10:        this.status = status;
11:    }
12:
13:    public Integer getId() {
14:        return id;
15:    }
16:
17:    public void setId(Integer id) {
18:        this.id = id;
19:    }
20:
21:    public Constants.TASK_STATUS getStatus() {
22:        return status;
23:    }
24:
25:    public void setStatus(Constants.TASK_STATUS status) {
26:        this.status = status;
27:    }
28:
29: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: public class Summary {
4:     private long totalTime;
5:     private Integer successfullrequest;
6:     private Integer failedrequest;
7:     private Integer users;
8:
9:     public Summary() {
10:         super();
11:         totalTime = 0;
12:         successfullrequest = 0;
13:         failedrequest = 0;
14:         users = 0;
15:     }
16:
17:     public synchronized Integer getSuccessfullrequest() {
18:         return successfullrequest;
19:     }
20:
21:     public synchronized void setSuccessfullrequest(Integer successfullrequest)
{
22:         this.successfullrequest = successfullrequest;
23:     }
24:
25:     public synchronized Integer getFailedrequest() {
26:         return failedrequest;
27:     }
28:
29:     public synchronized void setFailedrequest(Integer failedrequest) {
30:         this.failedrequest = failedrequest;
31:     }
32:
33:     public synchronized Integer getUsers() {
34:         return users;
35:     }
36:
37:     public synchronized void setUsers(Integer users) {
38:         this.users = users;
39:     }
40:
41:     public synchronized long getAverageTime() {
42:         if (successfullrequest > 0) {
43:             return totalTime / successfullrequest;
44:         }
45:         return 0;
46:     }
47:
48:     public synchronized void addTime(long time) {
49:         totalTime += time;
50:     }
51:
52:     public synchronized Integer getTotalRequests() {
53:         return successfullrequest + failedrequest;
54:     }
55: }
```

```

1: package ar.fiuba.taller.loadTestConsole;
2:
3: import java.util.concurrent.ArrayBlockingQueue;
4:
5: import org.apache.log4j.Logger;
6:
7: public class SummaryController implements Runnable {
8:     ArrayBlockingQueue<SummaryTask> pendingSummaryQueue;
9:     ArrayBlockingQueue<SummaryTask> finishedSummaryQueue;
10:     Summary summary;
11:     final static Logger logger = Logger.getLogger(App.class);
12:
13:     public SummaryController(
14:         ArrayBlockingQueue<SummaryTask> pendingSummaryQueue,
15:         ArrayBlockingQueue<SummaryTask> finishedSummaryQueue,
16:         Summary summary) {
17:         super();
18:         this.pendingSummaryQueue = pendingSummaryQueue;
19:         this.finishedSummaryQueue = finishedSummaryQueue;
20:         this.summary = summary;
21:     }
22:
23:     public void run() {
24:         logger.info("Se inicia el monitor");
25:         SummaryTask summaryTask = null;
26:         //
27:         try {
28:             do {
29:                 summaryTask = pendingSummaryQueue.take();
30:                 logger.info("Estadística recibida:\n" + "Id: "
31:                     + summaryTask.getId() + "\nStatus: "
32:                     + summaryTask.getStatus() + "\nTime elapsed: "
33:                     + summaryTask.getTimeElapsed() + "
34:                     + summaryTask.getUsersAmount()
35:                     + "\nSuccessful request: "
36:                     + summaryTask.getSuccessfulRequests());
37:                 if (summaryTask.getId() != Constants.DISCONNECT_ID)
38:                 {
39:                     // Actualizo las estadísticas
40:                     // Actualizo la cantidad de usuarios
41:                     if (summaryTask.getUsersAmount() != 0) {
42:                         summary.setUsers(summaryTask.getUsersAmount());
43:                     }
44:                     // Actualizo los requests
45:                     if (summaryTask.getSuccessfulRequests() != 0) {
46:                         summary.setSuccessfulRequests(summaryTask.getSuccessfulRequests());
47:                     }
48:                     else {
49:                         summary.setFailedRequests(summaryTask.getFailedRequests());
50:                     }
51:                 }
52:                 // Actualizo el tiempo promedio
53:                 summary.addTime(summaryTask.getTimeElapsed());
54:                 logger.info("Estadísticas actualizadas");
55:
56:             }
57:         } while (summaryTask.getId() != Constants.DISCONNECT_ID);
58:         logger.info(
59:             "Finalizando el monitor. Enviando mensaje de finalización"
60:             + " al control principal.");
61:         finishedSummaryQueue.put(summaryTask);
62:     } catch (InterruptedException e) {
63:         // TODO Auto-generated catch block
64:         e.printStackTrace();
65:     }
66:     logger.info("Monitor finalizado.");
67: }
68: }

```

```
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: }
```

```
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.setStepLength(paramList.get(1));
16:         this.setTicksLeft(paramList.get(1));
17:         this.setHeightOfStep(paramList.get(0));
18:     }
19:
20:     @Override
21:     public Integer getUsers(Integer tick) {
22:         if (getTicksLeft() == 0) {
23:             setTicksLeft(stepLength);
24:             setNumberOfUsers(getNumberOfUsers() + getHeightOfStep());
25:         }
26:         setTicksLeft(getTicksLeft() - 1);
27:         return getNumberOfUsers() <= getUpperBound() ? getNumberOfUsers()
28:             : getUpperBound();
29:     }
30:
31:     public Integer getStepLength() {
32:         return stepLength;
33:     }
34:
35:     public void setStepLength(Integer stepLength) {
36:         this.stepLength = stepLength;
37:     }
38:
39:     public Integer getTicksLeft() {
40:         return ticksLeft;
41:     }
42:
43:     public void setTicksLeft(Integer ticksLeft) {
44:         this.ticksLeft = ticksLeft;
45:     }
46:
47:     public Integer getHeightOfStep() {
48:         return heightOfStep;
49:     }
50:
51:     public void setHeightOfStep(Integer heightOfStep) {
52:         this.heightOfStep = heightOfStep;
53:     }
54: }
```

```
1: package ar.fiuba.taller.loadTestConsole;
2:
3: import ar.fiuba.taller.loadTestConsole.Constants.TASK_STATUS;
4:
5: public class ReportTask extends Task {
6:
7:     private Boolean analyzer;
8:     private String resource;
9:
10:    public Boolean getAnalyzer() {
11:        return analyzer;
12:    }
13:
14:    public void setAnalyzer(Boolean analyzer) {
15:        this.analyzer = analyzer;
16:    }
17:
18:    public String getResource() {
19:        return resource;
20:    }
21:
22:    public void setResource(String resource) {
23:        this.resource = resource;
24:    }
25:
26:    public ReportTask(Integer id, TASK_STATUS status, Boolean analyzer,
27:        String resource) {
28:        super(id, status);
29:        this.analyzer = analyzer;
30:        this.resource = resource;
31:    }
32: }
```

```

1: package ar.fiuba.taller.utils;
2:
3: import java.io.BufferedReader;
4: import java.io.DataOutputStream;
5: import java.io.InputStreamReader;
6: import java.net.HttpURLConnection;
7: import java.net.URL;
8: import java.util.Map;
9:
10: import javax.net.ssl.HttpsURLConnection;
11:
12: public class HttpRequester {
13:
14:     public HttpRequester() {
15:         // TODO Auto-generated constructor stub
16:     }
17:
18:     public String doHttpRequest(String method, String url,
19:         Map<String, String> headers, String data) throws Exception
20:     {
21:         String result = null;
22:         if (method.toLowerCase().equals("get")) {
23:             result = doGet(url, headers, data);
24:         } else if (method.toLowerCase().equals("post")) {
25:             result = doPost(url, headers, data);
26:         } else if (method.toLowerCase().equals("put")) {
27:             result = doPut(url, headers, data);
28:         }
29:         return result;
30:     }
31:
32:     // HTTP GET request
33:     private String doGet(String url, Map<String, String> headers, String data)
34:         throws Exception {
35:         // Armo la conexi3n
36:         if (data.length() > 0)
37:             url += "?" + data;
38:         URL obj = new URL(url);
39:         HttpURLConnection con = (HttpURLConnection) obj.openConnection();
40:
41:         // optional default is GET
42:         con.setRequestMethod("GET");
43:
44:         // add request header
45:         if (!headers.isEmpty()) {
46:             for (Map.Entry<String, String> entry : headers.entrySet())
47:                 con.setRequestProperty(entry.getKey(), entry.getValue());
48:         }
49:
50:         int responseCode = con.getResponseCode();
51:         // System.out.println("\nSending 'GET' request to URL : " + url);
52:         // System.out.println("Response Code : " + responseCode);
53:
54:         BufferedReader in = new BufferedReader(
55:             new InputStreamReader(con.getInputStream()));
56:         String inputLine;
57:         StringBuffer response = new StringBuffer();
58:
59:         while ((inputLine = in.readLine()) != null) {
60:             response.append(inputLine);
61:         }
62:         in.close();
63:
64:

```

```

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

```

```
127:
128:         // Send post request
129:         con.setDoOutput(true);
130:         DataOutputStream wr = new DataOutputStream(con.getOutputStream());
131:         wr.writeBytes(data);
132:         wr.flush();
133:         wr.close();
134:
135:         int responseCode = con.getResponseCode();
136:         // System.out.println("\nSending 'PUT' request to URL : " + url);
137:         // System.out.println("Post parameters : " + data);
138:         // System.out.println("Response Code : " + responseCode);
139:
140:         BufferedReader in = new BufferedReader(
141:             new InputStreamReader(con.getInputStream()));
142:         String inputLine;
143:         StringBuffer response = new StringBuffer();
144:
145:         while ((inputLine = in.readLine()) != null) {
146:             response.append(inputLine);
147:         }
148:         in.close();
149:
150:         // print result
151:         return response.toString();
152:     }
153:
154: }
```



```
1: package ar.fiuba.taller.utils;
2:
3: public class TerminateSignal {
4:
5:     private boolean terminate = false;
6:
7:     public synchronized boolean hasTerminate() {
8:         return this.terminate;
9:     }
10:
11:     public synchronized void terminate() {
12:         this.terminate = true;
13:     }
14: }
```

```
1: package ar.fiuba.taller.utils;
2:
3: public class Pair<A, B> {
4:     private A first;
5:     private B second;
6:
7:     public Pair(A first, B second) {
8:         super();
9:         this.first = first;
10:        this.second = second;
11:    }
12:
13:    public int hashCode() {
14:        int hashFirst = first != null ? first.hashCode() : 0;
15:        int hashSecond = second != null ? second.hashCode() : 0;
16:
17:        return (hashFirst + hashSecond) * hashSecond + hashFirst;
18:    }
19:
20:    public boolean equals(Object other) {
21:        if (other instanceof Pair) {
22:            Pair otherPair = (Pair) other;
23:            return ((this.first == otherPair.first
24:                || (this.first != null && otherPair.first
25:                    != null
26:                    && this.first.equals(other
27:                        Pair.first)))
28:                && (this.second == otherPair.second
29:                    || (this.second != null &&
30:                        otherPair.second != null
31:                        && this.se
32:                            cond.equals(otherPair.second)))));
33:        }
34:        return false;
35:    }
36:
37:    public String toString() {
38:        return "(" + first + ", " + second + ")";
39:    }
40:
41:    public A getFirst() {
42:        return first;
43:    }
44:
45:    public void setFirst(A first) {
46:        this.first = first;
47:    }
48:
49:    public B getSecond() {
50:        return second;
51:    }
52:
53:    public void setSecond(B second) {
54:        this.second = second;
55:    }
56: }
```