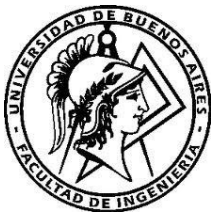


Universidad de Buenos Aires

Facultad de Ingeniería

75.61 – Taller de Programacion III

Trabajo Práctico

Load Test Console

1º Cuat. - 2017

Docentes :

- Andres Veiga
- Pablo Roca

Alumno:

- Pablo Méndez 88908 pablo.guillermo.mendez@gmail.com

Índice

1. Introducción

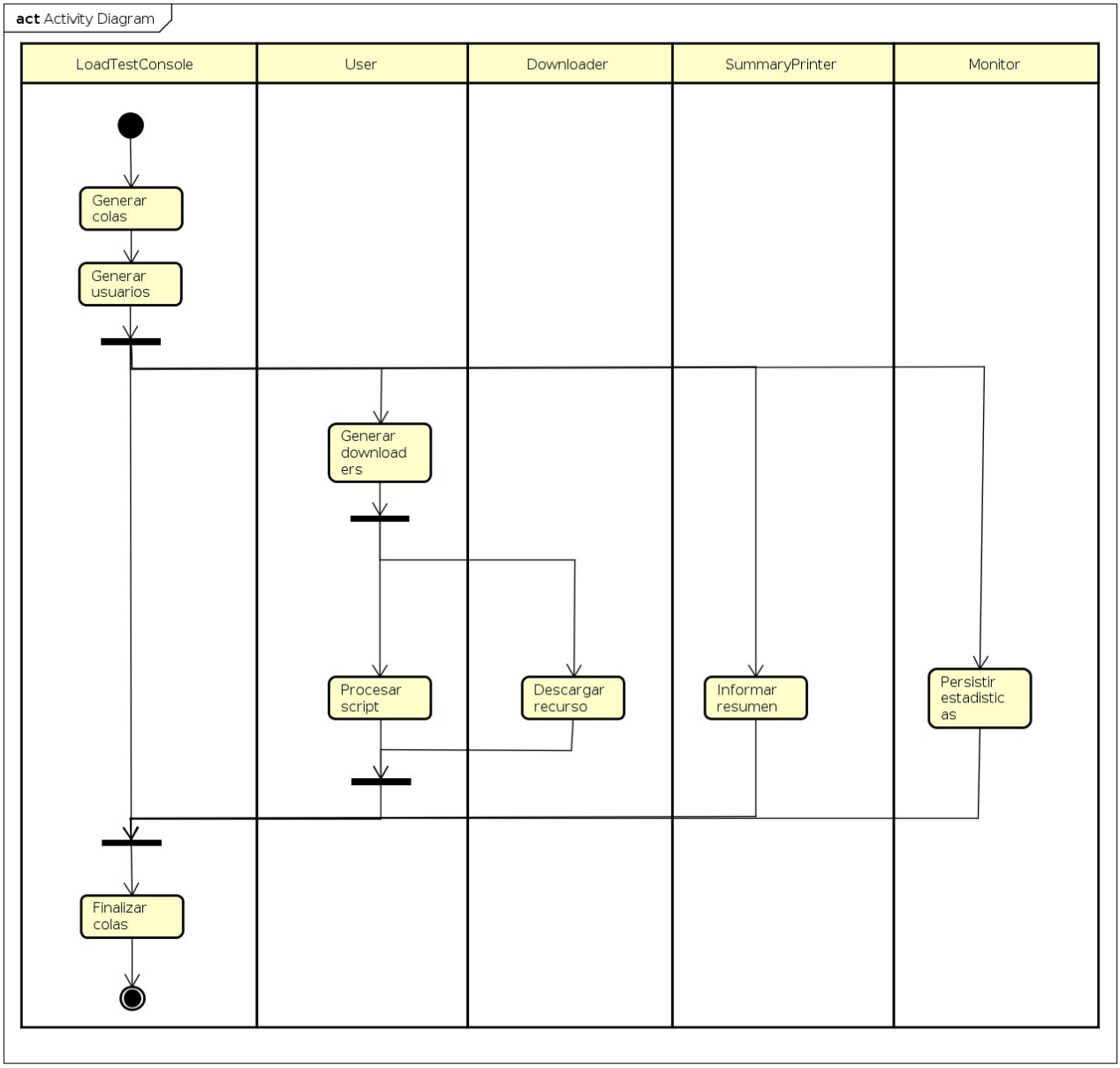
El presente trabajo consta en desarrollar una aplicación para simular tests de carga mediante la ejecución concurrende de requests http (GET, PUT y POST) y la descarga de algunos de los regusrsos asociados a las páginas obtenidas a estos métodos (SCRIPT, IMG y LINK). Para esto se desarrolló un programa java que hace uso de muchos conceptos acerca del manejo multithreading aprendidos en materias anteriores como Técnicas de Programación Concurrentes; de manera de, poder hacer un manejo adecuado y controlado de los hilos mediante la utilización de pool de threads y su correspondiente sincronización a través de colas de mensajes y locks.

2. Diagramas

2.1. Diagrama de actividades

A continuación se presenta el diagrama de actividades; el cual, compone la Vista de procesos del modelo 4+1 y mediante el cual se prsenta la forma en la que se comunican los procesos (hilos) del sistema.

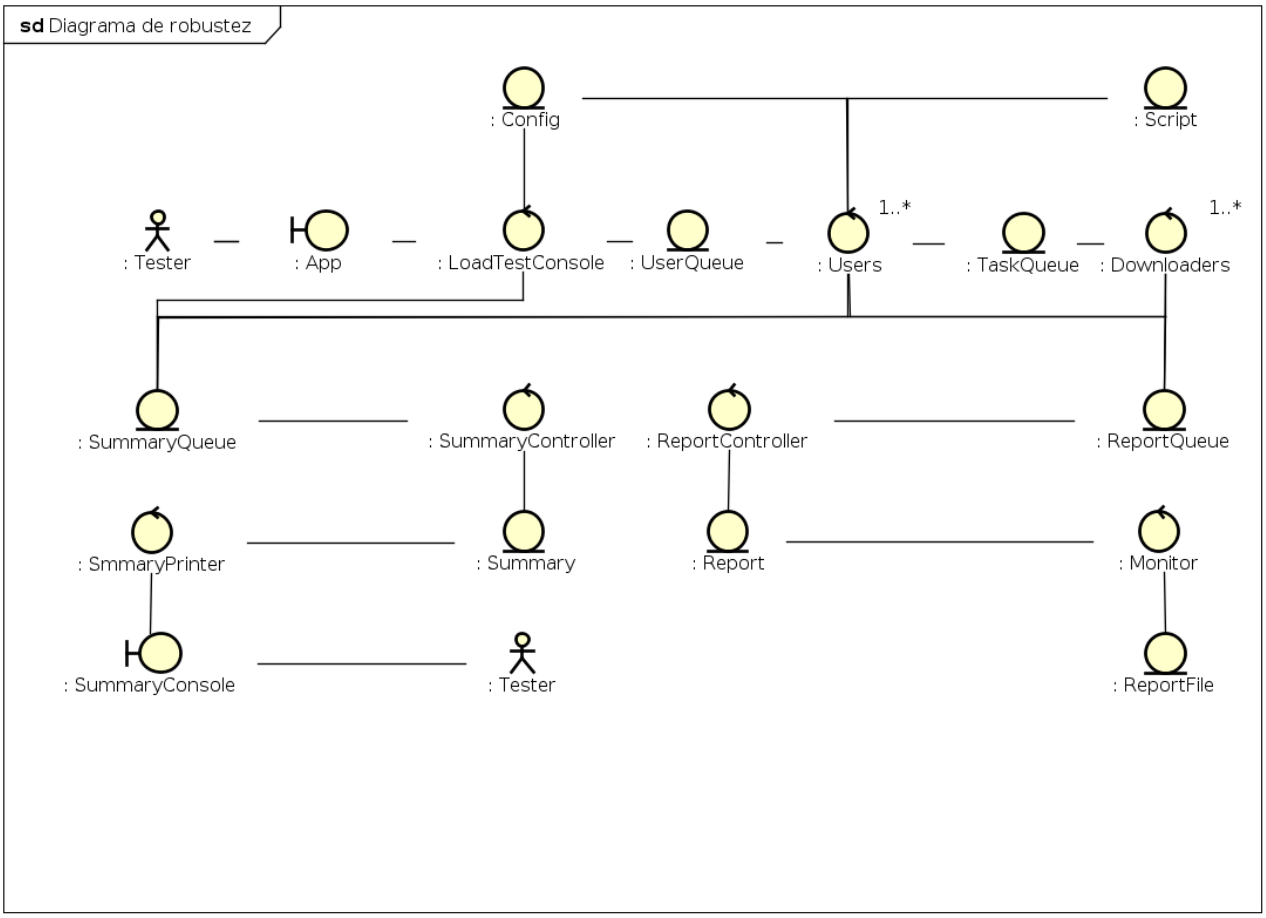
En este caso se presentan los procesos fundamentales del mismo, los cuales se encuentran en un escenario en el que el proceso principal genera al monitor, la impresora de resúmenes y a los usuarios. Estos últimos as us vez leen el script de url's y luego de un análisis de las páginas que se obtienen de ellos, generan tareas para que los downloaders descarguen los recursos asociados a esta última; esto es, imagenes, links y scripts.


act Activity Diagram
LoadTestConsole
User
Downloader
SummaryPrinter
Monitor
Generar colas
Generar usuarios
Generar downloaders
Procesar script
Descargar recurso
Informar resumen
Persistir estadisticas
Finalizar colas
powered by Astah

2.2. Diagrama de robustez

El diagrama de robustéz presentado a continuación pertenece a la vista lógica del modelo 4+1 y tiene por objetivo presentar cual es la funcionalidad del sistema, así como tambien las entidades y los controladores que lo conformarán paa poder llevarlas a cabo.

En este caso se muestra el flujo de información desde que el sistema es disparado por un usuario y como los distintos controladores van interactuando entre si a través de distintas entidades del mismo; esto es, colas de mensajes y entidades del dominio del problema protegidas con locks.



3. Código fuente

A continuación se presenta el código fuente de la solución desarrollada para este problema.

```
 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: }
```

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

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.log4j.Logger;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import ar.fiuba.taller.utils.HttpRequester;

public class User implements Runnable {
	private ArrayBlockingQueue<UserTask> userTaskPendingQueue;
	private ArrayBlockingQueue<UserTask> userTaskFinishedQueue;
	private ArrayBlockingQueue<SummaryTask> summaryQueue;
	private ArrayBlockingQueue<ReportTask> reportQueue;
	final static Logger logger = Logger.getLogger(App.class);

	public User(ArrayBlockingQueue<UserTask> userTaskPendingQueue,
			ArrayBlockingQueue<UserTask> userTaskFinishedQueue,
			ArrayBlockingQueue<SummaryTask> summaryQueue,
			ArrayBlockingQueue<ReportTask> reportQueue) {
		this.userTaskPendingQueue = userTaskPendingQueue;
		this.userTaskFinishedQueue = userTaskFinishedQueue;
		this.summaryQueue = summaryQueue;
		this.reportQueue = reportQueue;
	}

	public void run() {
		logger.info("Iniciando el usuario");
		String html;
		List<String> requestList;
		Integer downloaders = ConfigLoader.getInstance()
				.getMaxSizeDownloadersPoolThread();
		Integer taskId;
		DownloaderTask finishedTask;
		Boolean gracefullQuit = false;
		UserTask userTask;
		DownloaderTask downloaderTask;
		long time_elapsed, time_end, time_start;
		Integer bytesDownloaded;

		// Script
		Map<String, String> map;
		HttpRequester httpRequester = new HttpRequester();
		String method = "", url = "", data = "";
		File inputFile = new File(Constants.SCRIPT_FILE);
		DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInsta
nce();
		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
FinishedQueue,
					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.
 Se "
					+ "envÃa mensaje de desconexion a los dow
nloaders");
			for (int i = 0; i < downloaders; ++i) {
				downloaderTaskPendingQueue
						.put(new DownloaderTask(Constants.
```

```
DISCONNECT_ID,
                                                        null, null, null,
null));
                }
                logger.info("Esperando a que los downloaders finalicen");
                while (downloaders > 0) {
                        downloaderTask = downloaderTaskFinishedQueue.take(
);
                        if (downloaderTask.getId() == Constants.DISCONNECT
_ID) {
                                downloaders--;
                        }
                }
                logger.info("Downloaders finalizados");
                logger.info("Finalizando usuario");
                logger.info(
                                "Avisando al Control principal que el usua
rio "
                                + "termino");
                userTaskFinishedQueue.put(userTask);
                gracefullQuit = true;
                logger.info("Usuario finalizado");
        } else {
        logger.info("Se inicia un nuevo pulso de usuario");
        logger.info("Leo el script");
        try {

        logger.info("Leo el script");
        for (int temp = 0; temp < nList.getLength(); temp++) {
                map = new HashMap<String, String>();
                method = "";
                url = "";
                data = "";
                taskId = 0;

                // Obtengo la url
                org.w3c.dom.Node nNode = nList.item(temp);
                org.w3c.dom.Element eElement =
                                (org.w3c.dom.Element) nNode;
                method = eElement.getElementsByTagName("method")
                                .item(0).getTextContent();
                url = eElement.getElementsByTagName("url").item(0)
                                .getTextContent();
                data = eElement.getElementsByTagName("data").item(0)
                                .getTextContent();
                org.w3c.dom.Element e2 = (org.w3c.dom.Element) eElement
                                .getElementsByTagName("headers").item(0);
                org.w3c.dom.NodeList n2 = e2.getChildNodes();
                for (int i = 0; i < n2.getLength(); i++) {
                        Node n = n2.item(i);
                        org.w3c.dom.NodeList n3 = n.getChildNodes();
                        if (n3.getLength() > 0) {
                                if (!(n3.item(1).getTextContent().trim()
                                                .equals(""))) {
                                        map.put(n3.item(1).getTextContent(
),
                                                        n3.item(3).getText
Content());
                                }
                        }
                }

                logger.info("Siguiente paso a realizar: " + method
                                + " " + url);
                logger.info("Obteniendo recurso ...");
                html = httpRequester
                                .doHttpRequest(method, url, map, data)
                                .toLowerCase();
        time_start = System.currentTimeMillis();
        bytesDownloaded = html.length();
        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
                        + " nanosgundos");

        // Enviando estadistica de descarga a la cola de
        // estadisticas
        summaryQueue
                        .put(new SummaryTask(Constants.DEFAULT_ID,
                                        Constants.TASK_STATUS.SUBM
ITTED, 0,
                                        true, time_elapsed));

        logger.info(
                        "Rescato los tags LINK, IMG y SCRIPT e "
                        + "inserto las tasks en la cola");
        for (String tag : Arrays.asList(Constants.IMG_TAG,
                        Constants.SCRIPT_TAG, Constants.LINK_TAG))
 {
                logger.debug("Analizando el tag " + tag
                                + "sobre el documento: html");
                requestList = getLinks(html, tag);
                logger.debug(
                                "Recursos encontrados en la pagina
"
                                + " analizada: "
                                                + requestList.size
());

                for (String request : requestList) {
                        logger.info(
                                        "Enviando una nueva task c
on los siguiente "
                                        + "parametros:\nTaskId: "
                                                        + taskId +
 "\nMetodo: "
                                                        + Constant
s.GET_METHOD
                                                        + "\nReque
st: " + request
                                                        + "\nEstad
o: "
                                                        + Constant
s.TASK_STATUS.SUBMITTED);
                        downloaderTaskPendingQueue
                                        .put(new DownloaderTask(ta
skId,
                                                        Constants.
GET_METHOD,
                                                        request,
                                                        Constants.
TASK_STATUS.SUBMITTED,
                                                        tag));
                        ++taskId;
                }
        }
        // Informo al monitor que analice una url
        reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
```

```
245:                                    Constants.TASK_STATUS.EXECUTING, true,
246:                                    null));
247:
248:                    logger.info(
249:                                    "Esperando a que terminen los downloaders"
);
250:                    //
251:                    while (taskId > 0) {
252:                            finishedTask = downloaderTaskFinishedQueue
253:                                            .take();
254:                            taskId--;
255:                            logger.info("Task finalizada:\nTaskId: "
256:                                            + finishedTask.getId() + "\nStatus
: "
257:                                            + finishedTask.getStatus());
258:                    }
259:                    }
260:                    } catch (IOException e) {
261:                            e.printStackTrace();
262:                    } catch (Exception e) {
263:                            // Informo al summary que fallo la descarga
264:                            summaryQueue.put(new SummaryTask(Constants.DEFAULT
_ID,
265:                                            Constants.TASK_STATUS.SUBMITTED, 0
, false, 0));
266:                            // Informo al monitor que fallo la descarga
267:                            reportQueue.put(new ReportTask(Constants.DEFAULT_I
D,
268:                                            Constants.TASK_STATUS.FAILED, true
, null));
269:                            e.printStackTrace();
270:                    }
271:
272:                }
273:                // Informo que el user termino
274:                logger.info("Informando al monitor que el usuario termino");
275:                reportQueue.put(new ReportTask(Constants.DEFAULT_ID,
276:                                Constants.TASK_STATUS.FINISHED, true, null));
277:                logger.info(
278:                                "Informando al loadtestconsolo que el usuario "
279:                                + "termino de ejecutar el script");
280:                userTaskFinishedQueue.put(new UserTask(Constants.DEFAULT_ID,
281:                                Constants.TASK_STATUS.FINISHED));
282:
283:                } catch (InterruptedException e1) {
284:                        // TODO Auto-generated catch block
285:                        e1.printStackTrace();
286:                }
287:                }
288:        }
289:
290:        private String getPage(String urlToRead, String method) throws Exception {
291:                StringBuilder result = new StringBuilder();
292:                URL url = new URL(urlToRead);
293:                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
294:                conn.setRequestMethod(method);
295:                BufferedReader rd = new BufferedReader(
296:                                new InputStreamReader(conn.getInputStream()));
297:                String line;
298:                while ((line = rd.readLine()) != null) {
299:                        result.append(line);
300:                }
301:                rd.close();
302:                return result.toString();
303:        }
304:
305:        private List<String> getLinks(String page, String tag)
306:                        throws URISyntaxException, IOException, BadLocationExcepti
on {
307:                List<String> requestsList = new ArrayList<String>();
308:                String currentAttribute = null;
309:
310:                if (tag.equals(Constants.LINK_TAG)) {
311:                        currentAttribute = "href";
312:                } else if (tag.equals(Constants.SCRIPT_TAG)) {
313:                        currentAttribute = "src";
314:                } else if (tag.equals(Constants.IMG_TAG)) {
315:                        currentAttribute = "src";
316:                }
317:                org.jsoup.nodes.Document doc = Jsoup.parse(page);
318:                Elements resource = doc.select(tag);
319:                Iterator<Element> it = resource.iterator();
320:
321:                while (it.hasNext()) {
322:                        requestsList.add(it.next().attr(currentAttribute));
323:                }
324:
325:                return requestsList;
326:        }
327: }
```

```
 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:                                 System.out.println("Requests exitosos.............
: "
   42:                                                 + summary.getSuccessfullrequest()
+ "/"
   43:                                                 + summary.getTotalRequests());
   44:                                 System.out.println("Requests fallidos.............
: "
   45:                                                 + summary.getFailedrequest() + "/"
   46:                                                 + summary.getTotalRequests());
   47:                                 System.out.println("Cantidad de usuarios..........
: "
   48:                                                 + summary.getUsers());
   49:                                 System.out.println("");
   50:                                 System.out.println("Presione ^C para terminar...")
;
   51:                         }
   52:                         try {
   53:                                 Thread.sleep(10000);
   54:                         } catch (InterruptedException e) {
   55:                                 // TODO Auto-generated catch block
   56:                                 e.printStackTrace();
   57:                         }
   58:                 }
   59:                 logger.info("Finalizando SummaryPrinter");
   60:         }
   61: }
```

```
    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: }
```

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

import ar.fiuba.taller.utils.*;
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 java.util.concurrent.TimeUnit;

import org.apache.log4j.Logger;

public class LoadTestConsole implements Runnable {

	private Integer maxSizeUserPoolThread;
	private String function;
	private List<Integer> functionParamList;
	private UserPattern userPattern;
	private List<Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>>> usersQueuesList;
	private TerminateSignal terminateSignal;

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

	public LoadTestConsole(TerminateSignal terminateSignal) {
		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 run() {
		ArrayBlockingQueue<UserTask> userTaskPendingQueue;
		ArrayBlockingQueue<UserTask> userTaskFinishedQueue;
		ArrayBlockingQueue<SummaryTask> summaryPendingQueue = new ArrayBlockingQueue<SummaryTask>(
				ConfigLoader.getInstance().getTasksQueueSize());
		ArrayBlockingQueue<SummaryTask> summaryFinishedQueue = new ArrayBlockingQueue<SummaryTask>(
				ConfigLoader.getInstance().getTasksQueueSize());
		ArrayBlockingQueue<ReportTask> reportPendingQueue = new ArrayBlockingQueue<ReportTask>(
				ConfigLoader.getInstance().getTasksQueueSize());
		ArrayBlockingQueue<ReportTask> reportFinishedQueue = new ArrayBlockingQueue<ReportTask>(
				ConfigLoader.getInstance().getTasksQueueSize());

		Summary summary = new Summary();
		Report report = new Report();
		TerminateSignal summaryTerminateSignal = new TerminateSignal();
		TerminateSignal reportTerminateSignal = new TerminateSignal();
		Thread summaryControllerThread = new Thread(new SummaryController(
				summaryPendingQueue, summaryFinishedQueue, summary));

		Thread summaryPrinterThread = new Thread(new SummaryPrinter(summary,
				summaryTerminateSignal, terminateSignal));
		Thread reportControllerThread = new Thread(new ReportController(
				reportPendingQueue, reportFinishedQueue, report));
		Thread monitorThread = new Thread(
				new Monitor(report, reportTerminateSignal));

		Integer currentUsers = 0, deltaUseres = 0, tick = 0, userNumber = 0;

		UserTask userTask;
		SummaryTask summaryTask;
		ReportTask reportTask;

		logger.info("Se inicia una nueva instancia de LoadTestConsole");
		logger.info("Creando el pool de threads de usuarios");
		ExecutorService usersThreadPool = Executors
				.newFixedThreadPool(maxSizeUserPoolThread);

		// Reportes
		summaryControllerThread.start();
		summaryPrinterThread.start();
		reportControllerThread.start();
		monitorThread.start();

		try {
			while (!terminateSignal.hasTerminate()) {
				tick++;
				deltaUseres = userPattern.getUsers(tick) - currentUsers;

				logger.info("Se inicia el pulso: " + tick);
				logger.info("Cantidad de usuarios actualmente corriendo: "
						+ currentUsers);
				logger.info(
						"Cantidad de usuarios que deben correr en este pulso: "
								+ userPattern.getUsers(tick));
				logger.info("Cantidad de usuarios que deben ingresar: "
						+ deltaUseres);

				for (int i = 0; i < deltaUseres; i++) {
					currentUsers++;
					logger.info("Creando el usuario: " + currentUsers);

					// Creo las colas para el nuevo usuario
					userTaskPendingQueue = new ArrayBlockingQueue<UserTask>(
							ConfigLoader.getInstance()
.getTasksQueueSize());
					userTaskFinishedQueue = new ArrayBlockingQueue<UserTask>(
							ConfigLoader.getInstance()
.getTasksQueueSize());

					// Me guardo las dos colas en la lista de colas de usuarios
					usersQueuesList
```

```
  113:                                                    .add(new Pair<BlockingQueu
e<UserTask>, BlockingQueue<UserTask>>(
  114:                                                                    userTaskPe
ndingQueue,
  115:                                                                    userTaskFi
nishedQueue));
  116:
  117:                                    // Creo el usuario y les paso las colas pa
ra que puedan
  118:                                    // insertar tareas
  119:                                    usersThreadPool.submit(new User(userTaskPe
ndingQueue,
  120:                                                    userTaskFinishedQueue, sum
maryPendingQueue,
  121:                                                    reportPendingQueue));
  122:                                    logger.info("Usuario creado");
  123:                            }
  124:                            logger.info("Usuarios creados");
  125:
  126:                            // Actualizo la cola de estadisticas con la cantid
ad de usuarios
  127:                            // actuales
  128:                            summaryPendingQueue.put(new SummaryTask(Constants.
DEFAULT_ID,
  129:                                            Constants.TASK_STATUS.SUBMITTED, c
urrentUsers, null,
  130:                                            0));
  131:
  132:                            userNumber = 0;
  133:
  134:                            // Despierto a los users enviandoles un mensaje pa
ra que se
  135:                            // pongan a trabajar
  136:                            logger.info("Despertando los usuarios para un nuev
o pulso");
  137:                            for (Pair<BlockingQueue<UserTask>, BlockingQueue<U
serTask>> pair : usersQueuesList) {
  138:                                    logger.info("Despertando al usuario: " + u
serNumber);
  139:                                    pair.getFirst().put(new UserTask(Constants
.DEFAULT_ID,
  140:                                                    Constants.TASK_STATUS.SUBM
ITTED));
  141:
  142:                                    ++userNumber;
  143:                            }
  144:
  145:                            userNumber = 0;
  146:
  147:                            logger.info(
  148:                                            "Me quedo esperando a que los usua
rios terminen sus tareas");
  149:                            for (Pair<BlockingQueue<UserTask>, BlockingQueue<U
serTask>> pair : usersQueuesList) {
  150:                                    logger.info("Esperando al usuario: " + use
rNumber);
  151:                                    userTask = pair.getSecond().take();
  152:                                    logger.info("Usuario: " + userNumber + " f
inalizado");
  153:                                    ++userNumber;
  154:                            }
  155:
  156:                    }
  157:                    // Termino la simulacion. Tengo que parar a los usuarios
  158:
  159:                    userNumber = 0;
  160:                    // Despierto a los users enviandoles un mensaje de descone
ccion
  161:                    logger.info("Despertando los usuarios para desconectar");
  162:                    for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>
> pair : usersQueuesList) {
  163:                            logger.info("Desconectando al usuario: " + userNum
ber);
  164:                            pair.getFirst().put(new UserTask(Constants.DISCONN
ECT_ID,
  165:                                            Constants.TASK_STATUS.SUBMITTED));
  166:                            ++userNumber;
  167:                    }
  168:
  169:                    userNumber = 0;
  170:                    logger.info("Me quedo esperando a que los usuarios se desc
onecten");
  171:                    for (Pair<BlockingQueue<UserTask>, BlockingQueue<UserTask>
> pair : usersQueuesList) {
  172:                            logger.info("Esperando al usuario: " + userNumber)
;
  173:                            do {
  174:                                    userTask = pair.getSecond().take();
  175:                            } while (userTask.getId() != Constants.DISCONNECT_
ID);
  176:                            logger.info("Usuario: " + userNumber + " desconect
ado");
  177:                            ++userNumber;
  178:                    }
  179:
  180:                    // Reportes
  181:                    logger.info("Parando el summary printer");
  182:                    summaryTerminateSignal.terminate();
  183:                    summaryPrinterThread.join();
  184:                    logger.info("Summary printer finalizado");
  185:
  186:                    logger.info("Parando el monitor");
  187:                    reportTerminateSignal.terminate();
  188:                    monitorThread.join();
  189:                    logger.info("Monitor finalizado");
  190:
  191:                    logger.info("Parando el Controlador de reportes");
  192:                    reportPendingQueue.put(new ReportTask(Constants.DISCONNECT
_ID,
  193:                                    Constants.TASK_STATUS.SUBMITTED, null, nul
l));
  194:                    reportTask = reportFinishedQueue.take();
  195:                    reportControllerThread.join();
  196:                    logger.info("Controlador de reportes finalizado");
  197:
  198:                    logger.info("Parando el Controlador de resumen");
  199:                    summaryPendingQueue.put(new SummaryTask(Constants.DISCONNE
CT_ID,
  200:                                    Constants.TASK_STATUS.SUBMITTED, 0, false,
 0));
  201:                    summaryTask = summaryFinishedQueue.take();
  202:                    summaryControllerThread.join();
  203:                    logger.info("Controlador de resumen finalizado");
  204:
  205:            } catch (InterruptedException e) {
  206:                    // TODO Auto-generated catch block
  207:                    e.printStackTrace();
  208:            }
  209:
  210:    }
  211:
  212:    public String getFunction() {
  213:            return function;
  214:    }
```

```
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: }
```

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

import java.io.FileInputStream;
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) {
		try {
			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)));
			}
		} catch (IOException ex) {
			logger.error(ex.toString());
			ex.printStackTrace();
		}

		logger.info("Parametros cargados exitosamente:");
		logger.info("simulationTime: " + simulationTime);
		logger.info("userLoginTimeInterval: " + userLoginTimeInterval);
		logger.info("function: " + function);
		logger.info("Parametros de la funcion " + function);
		for (Integer param : functionPatternParam) {
			logger.info(param);
		}

	}

	public Integer getSimulationTime() {
		return simulationTime;
	}

	public void setSimulationTime(Integer simulationTime) {
		this.simulationTime = simulationTime;
	}

	public Integer getUserLoginTimeInterval() {
		return userLoginTimeInterval;
	}

	public void setUserLoginTimeInterval(Integer userLoginTimeInterval) {
		this.userLoginTimeInterval = userLoginTimeInterval;
	}

	public String getFunction() {
		return function;
	}

	public void setFunction(String function) {
		this.function = function;
	}

	public ArrayList<Integer> getFunctionPatternParam() {
		return functionPatternParam;
	}

	public Integer getUsersQueueSize() {
		return usersQueueSize;
	}

	public void setUsersQueueSize(Integer usersQueueSize) {
		this.usersQueueSize = usersQueueSize;
	}

	public Integer getMaxsizeUserPoolThread() {
		return maxsizeUserPoolThread;
	}

	public void setMaxsizeUserPoolThread(Integer maxsizeUserPoolThread) {
		this.maxsizeUserPoolThread = maxsizeUserPoolThread;
	}

	public Integer getMaxSizeDownloadersPoolThread() {
		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
hreads();
   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
hreads();
   50:                                                 }
   51:                                         } else { // Es una task enviada por un dow
nloader
   52:                                                 if (reportTask
   53:                                                                 .getStatus() == Co
nstants.TASK_STATUS.SUBMITTED) {
   54:                                                         report.incDownloadResource
Threads();
   55:                                                 } else if (reportTask
   56:                                                                 .getStatus() == Co
nstants.TASK_STATUS.EXECUTING) {
   57:                                                         if (reportTask.getResource
()
   58:                                                                         .equals(Co
nstants.SCRIPT_TAG)) {
   59:                                                                 report.incDownload
edScripts();
   60:                                                         } else if (reportTask.getR
esource()
   61:                                                                         .equals(Co
nstants.LINK_TAG)) {
   62:                                                                 report.incDownload
edLinks();
   63:                                                         } else if (reportTask.getR
esource()
   64:                                                                         .equals(Co
nstants.IMG_TAG)) {
   65:                                                                 report.incDownload
edImages();
   66:                                                         }
   67:                                                 } else {
   68:                                                         report.decDownloadResource
Threads();
   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 cont
rol 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: }
```

```
 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 numberOfUseres, Integer upperBound) {
 9:                 this.setNumberOfUsers(numberOfUseres);
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: }
```

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

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
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> downloaderTaskPendigQue
ue,
			ArrayBlockingQueue<DownloaderTask> downloaderTaskFinishedQ
ueue,
			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_I
D,
						Constants.TASK_STATUS.SUBMITTED, f
alse, null));
				logger.info("Task obtenida con los siguientes para
metros:\n"
						+ "Id: " + task.getId() + "\nStatu
s: "
						+ task.getStatus() + "\nMethod: "
+ task.getMethod()
						+ "\nUri: " + task.getUri());
				if (task.getId() == Constants.DISCONNECT_ID) {
					logger.info(
							"Mensaje de desconexion. S
e finaliza el thread.");
					logger.info("Confirmando finalizacion al u
ser");
					downloaderTaskFinishedQueue.put(task);
					logger.info("Downloader finalizado");
					gracefullQuit = true;
				} else {
					logger.info("Nueva tarea recibida");
					logger.info("Descargando recurso...");
					task.setStatus(Constants.TASK_STATUS.EXECU
TING);

					try {
						reportQueue.put(new ReportTask(Con
stants.DEFAULT_ID,
								Constants.TASK_STA
TUS.EXECUTING, false,
								task.getResourceTy
pe()));
						time_start = System.currentTimeMil
lis();
						bytesDownloaded = httpRequester
								.doHttpRequest(tas
k.getMethod(), task.getUri(),
ne
w HashMap<String, String>(), "")
								.length();// downl
oad(task.getMethod(),
// task.getUri());
						time_end = System.currentTimeMilli
s();
						time_elapsed = time_end - time_sta
rt;
						logger.info("Bytes descargados: "
+ bytesDownloaded);
						logger.info("Tiempo inicial: " + t
ime_start
								+ " nanosgundos");
						logger.info(
								"Tiempo final: " +
 time_end + " nanosgundos");
						logger.info("Tiempo transcurrido:
" + time_elapsed
								+ " nanosegundos")
;
						summaryQueue.put(new SummaryTask(C
onstants.DEFAULT_ID,
								Constants.TASK_STA
TUS.SUBMITTED, 0, true,
								time_elapsed));
						// Informo al monitor que arranco
el usuario
						reportQueue.put(new ReportTask(Con
stants.DEFAULT_ID,
								Constants.TASK_STA
TUS.FINISHED, false, null));
						task.setStatus(Constants.TASK_STAT
US.FINISHED);
					} catch (Exception e) {
						task.setStatus(Constants.TASK_STAT
US.FAILED);
						reportQueue.put(new ReportTask(Con
stants.DEFAULT_ID,
								Constants.TASK_STA
TUS.FAILED, false, null));
						summaryQueue.put(new SummaryTask(C
onstants.DEFAULT_ID,
								Constants.TASK_STA
TUS.SUBMITTED, 0, false, 0));
					} finally {
						downloaderTaskFinishedQueue.put(ta
sk);
```

```
                                        }
                                }
                        }
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

}
```

```
 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("Estadistica recibida:\n" + "Id: "
   31:                                                 + summaryTask.getId() + "\nStatus:
 "
   32:                                                 + summaryTask.getStatus() + "\nTim
e elapsed: "
   33:                                                 + summaryTask.getTimeElapsed() + "
\nAmount of users: "
   34:                                                 + summaryTask.getUsersAmount()
   35:                                                 + "\nSuccessfull request: "
   36:                                                 + summaryTask.getSuccessfullReques
t());
   37:                                 if (summaryTask.getId() != Constants.DISCONNECT_ID
) {
   38:                                         // Actualizo las estadisticas
   39:                                         // Actualizo la cantidad de usuarios
   40:                                         if (summaryTask.getUsersAmount() != 0) {
   41:                                                 summary.setUsers(summaryTask.getUs
ersAmount());
   42:                                         }
   43:                                         // Actualizo los requests
   44:                                         if (summaryTask.getSuccessfullRequest() !=
 null) {
   45:                                                 if (summaryTask.getSuccessfullRequ
est()) {
   46:                                                         summary.setSuccessfullrequ
est(
   47:                                                                         summary.ge
tSuccessfullrequest() + 1);
   48:                                                 } else {
   49:                                                         summary.setFailedrequest(
   50:                                                                         summary.ge
tFailedrequest() + 1);
   51:                                                 }
   52:                                         }
   53:                                         // Actualizo el tiempo promedio
   54:                                         summary.addTime(summaryTask.getTimeElapsed
());
   55:                                         logger.info("Estadisticas actualizadas");
   56:                                 }
   57:                         } while (summaryTask.getId() != Constants.DISCONNECT_ID);
   58:                         logger.info(
   59:                                         "Finalizando el monitor. Enviando mensje d
e finalizacion"
   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: }
```

```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());
			}
		}

		int responseCode = con.getResponseCode();
		// System.out.println("\nSending 'GET' request to URL : " + url);
		// System.out.println("Response Code : " + responseCode);

		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)
			throws Exception {
		URL obj = new URL(url);
		HttpURLConnection con = (HttpURLConnection) obj.openConnection();

		// add reuqest method
		con.setRequestMethod("POST");

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

		// Send post request
		con.setDoOutput(true);
		DataOutputStream wr = new DataOutputStream(con.getOutputStream());
		wr.writeBytes(data);
		wr.flush();
		wr.close();

		int responseCode = con.getResponseCode();
		// System.out.println("\nSending 'POST' request to URL : " + url);
		// System.out.println("Post parameters : " + data);
		// System.out.println("Response Code : " + responseCode);

		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 PUT request
	private String doPut(String url, Map<String, String> headers, String data)
			throws Exception {
		URL obj = new URL(url);
		HttpURLConnection con = (HttpURLConnection) obj.openConnection();

		// add reuqest method
		con.setRequestMethod("PUT");

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

```
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
!= null
   25:                                                         && this.first.equals(other
Pair.first)))
   26:                                         && (this.second == otherPair.second
   27:                                                         || (this.second != null &&
 otherPair.second != null
   28:                                                                         && this.se
cond.equals(otherPair.second))));
   29:                 }
   30:
   31:                 return false;
   32:         }
   33:
   34:         public String toString() {
   35:                 return "(" + first + ", " + second + ")";
   36:         }
   37:
   38:         public A getFirst() {
   39:                 return first;
   40:         }
   41:
   42:         public void setFirst(A first) {
   43:                 this.first = first;
   44:         }
   45:
   46:         public B getSecond() {
   47:                 return second;
   48:         }
   49:
   50:         public void setSecond(B second) {
   51:                 this.second = second;
   52:         }
   53: }
```