ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени федеральное государственное

Бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра информатики

Лабораторная работа №8

По дисциплине: «Технология разработки программного обеспечения»

Тема: «Модифицированный вэб сканнер»

Выполнил студент:

Группы: БФИ1902

Шацкий Е. И.

Проверила:

Мосева. М.С.

Москва, 2020 г.

**Оглавление**

[**1 Задачи на лабораторную работу.** 3](#_Toc59452550)

[**2 Ход решения лабораторной работы.** 4](#_Toc59452551)

[**2.1 Листинги программы** 4](#_Toc59452552)

[**2.2 Результаты работы программы.** 16](#_Toc59452553)

[**Список используемых источников** 17](#_Toc59452554)

# **1 Задачи на лабораторную работу.**

Для выполнения лабораторной работы необходимо:

а) Реализовать класс с именем URLPool, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов (также известный как "глубина поиска"). Первый URL-адрес, который нужно будет найти, будет иметь глубину поиска равную 0, URLадреса, найденные на этой странице, будут иметь глубину поиска равную 1 и т.д. Необходимо сохранить URL-адреса и их глубину поиска вместе, как экземпляры класса с именем URLDepthPair, как это было сделано в прошлой лабораторной работе. У пользователя класса URLPool должен быть способ получения пары URL-глубина из пула и удаления этой пары из списка одновременно. Должен также быть способ добавления пары URL-глубина к пулу

б) Чтобы выполнить веб-сканирование в нескольких потоках, необходимо создать класс CrawlerTask, который реализует интерфейс Runnable. Каждый экземпляр CrawlerTask должна иметь ссылку на один экземпляр класса URLPool, который был описан выше.

в) Так как веб-сканер будет порождать некоторое количество потоков, измените программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера.

г) Синхронизируйте объект пула URL-адресов во всех критических точках, так как теперь код должен быть ориентирован на многопоточность.

д) Веб-сканер не должен постоянно опрашивать пул URL-адресов в случае, если он пуст. Вместо этого пусть они ожидают в случае, когда нет доступных URL-адресов. Реализовать это, используя метод wait() внутри «get URL» в случае, если ни один URL-адрес в настоящее время недоступен. Соответственно, метод "add URL" пула URL-адресов должен использовать функцию notify() в случае, когда новый URL-адрес добавлен к пулу.

е) Обновить пару URL-Depth для использования класса java.net.URL и произведите соответствующие изменения в веб-сканере для того, чтобы он соответствовал и относительным URL-адресам, и абсолютным.

ж) Реализовать список URL-адресов, которые были просмотрены, и избегайте возврата к ним. Используйте один из классов коллекций java. Какойто набор, который поддер

з) Добавить другой дополнительный параметр командной строки для того, чтобы определить, сколько времени поток веб-сканера должен ждать сервера для возврата требуемой веб-страницы.

и) Также воздать класс, CrawlerHelper, который будет обрабатывать исключения и некорректность ввода дополнительно от класса Crawler, тем самым обеспечивая большую производительность.

# **2 Ход решения лабораторной работы.**

## **2.1 Листинги программы**

Ниже представлены листинги классов программы, которые обеспечивают решение поставленных задач.

Листинг 1 – класс Crawler.java

import java.lang.Exception;  
import java.util.\*;  
import java.net.MalformedURLException;  
import java.net.\*;  
import java.io.\*;  
  
public class Crawler {  
  
 public static final int *HTTP\_PORT* = 80;  
 public static final String *HOOK\_REF* = "<a href=\"";  
 public static final String *BAD\_REQUEST\_LINE* = "HTTP/1.1 400 Bad Request";   
 public static final int *NUM\_OF\_DEFAULT\_THREADS* = 4;  
   
  
 public static final String *testURL* = "http://users.cms.caltech.edu/~donnie/cs11/java/";  
 public static final int *testDepth* = 1;  
  
  
 public int depth;  
  
 public int numOfThreads;  
  
 public Crawler() {  
   
 }  
  
  
 // ’®зЄ  ўе®¤   
 public static void main (String[] args) {  
  
 Crawler crawler = new Crawler();  
  
 crawler.numOfThreads = Crawler.*NUM\_OF\_DEFAULT\_THREADS*;  
  
 URLDepthPair firstRezAndSetDepth = crawler.getFirstURLDepthPair(args);  
 crawler.numOfThreads = CrawlerHelper.*getNumOfThreads*(args);  
   
 URLPool pool = new URLPool(crawler.depth);  
 pool.put(firstRezAndSetDepth);  
   
 int totalThreads = 0;  
 int initialActive = Thread.*activeCount*();  
   
 while (pool.getWaitThreads() != crawler.numOfThreads) {  
  
   
 if (Thread.*activeCount*() - initialActive < crawler.numOfThreads) {  
 CrawlerTask crawlerTask = new CrawlerTask(pool);  
 new Thread(crawlerTask).start();  
 }  
 else {  
 try {  
 Thread.*sleep*(100); // 0.1 second  
 }  
 catch (InterruptedException ie) {  
 System.*out*.println("Caught: unexpected InterruptedException");  
 }  
  
 }  
 }  
  
   
 LinkedList<URLDepthPair> list = pool.getWatchedList();  
 int count = 1;  
 for (URLDepthPair page : list) {  
 System.*out*.println(count + " | " + page.toString());  
 count += 1;  
 }  
   
 list = pool.getBlockedList();  
 count = 1;  
 for (URLDepthPair page : list) {  
 System.*out*.println(count + " | " + page.toString());  
 count += 1;  
 }  
  
 System.*exit*(0);  
   
 }  
  
 public static void createURlDepthPairObject(String url, int depth, LinkedList<URLDepthPair> listOfUrl) {  
 URLDepthPair newURL = null;  
 try{  
 newURL = new URLDepthPair(url, depth);  
 } catch (MalformedURLException e) {  
 e.printStackTrace();  
 }  
 listOfUrl.addLast(newURL);  
 }  
   
  
 public URLDepthPair getFirstURLDepthPair(String[] args) {  
 CrawlerHelper help = new CrawlerHelper();  
  
 URLDepthPair urlDepth = help.getURLDepthPairFromArgs(args);  
 if (urlDepth == null) {  
 System.*out*.println("Args are empty or have exception.\n");  
  
 urlDepth = help.getURLDepthPairFromInput();  
 }  
  
 this.depth = urlDepth.getDepth();  
 urlDepth.setDepth(0);  
  
   
 return(urlDepth);  
  
 }  
  
  
 public static LinkedList<URLDepthPair> parsePage(URLDepthPair element) {  
 LinkedList<URLDepthPair> listOfUrl = new LinkedList<URLDepthPair>();  
   
 Socket socket = null;  
   
 try {  
 socket = new Socket(element.getHostName(), *HTTP\_PORT*);  
  
 try {  
 socket.setSoTimeout(5000);  
 }  
 catch (SocketException exc) {  
 System.*err*.println("SocketException: " + exc.getMessage());  
 return null;  
 }  
  
  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
  
 out.println("GET " + element.getPagePath() + " HTTP/1.1");  
 out.println("Host: " + element.getHostName());  
 out.println("Connection: close");  
 out.println("");  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
  
 String line = in.readLine();  
  
 if (line.startsWith(*BAD\_REQUEST\_LINE*)) {  
 System.*out*.println(line + "\n");  
 return null;  
 }  
;  
  
 int strCount = 0;  
 while(line != null) {  
 try {  
  
 line = in.readLine();  
 strCount += 1;  
 String url = CrawlerHelper.*getURLFromHTMLTag*(line);  
 if (url == null) continue;  
  
 if (url.startsWith("https://")) {  
 continue;  
 }  
  
 if (url.startsWith("../")) {   
 String newUrl = CrawlerHelper.*urlFromBackRef*(element.getURL(), url);  
 Crawler.*createURlDepthPairObject*(newUrl, element.getDepth() + 1, listOfUrl);  
 }   
  
 else if (url.startsWith("http://")) {  
 String newUrl = CrawlerHelper.*cutTrashAfterFormat*(url);  
 Crawler.*createURlDepthPairObject*(newUrl, element.getDepth() + 1, listOfUrl);  
 }   
  
 else {   
 String newUrl;  
 newUrl = CrawlerHelper.*cutURLEndFormat*(element.getURL()) + url;  
 Crawler.*createURlDepthPairObject*(newUrl, element.getDepth() + 1, listOfUrl);  
 }  
  
 }  
 catch (Exception e) {  
 break;  
 }  
 }  
   
 if (strCount == 1) {  
 System.*out*.println("No http !");  
 return null;  
 }  
  
   
 }  
 catch (UnknownHostException e) {  
 System.*out*.println("UnknownHostException catched");  
 }  
 catch (IOException e) {  
 e.printStackTrace();  
 }  
   
 return listOfUrl;  
 }  
  
  
 public static void showResults(URLDepthPair element, LinkedList<URLDepthPair> listOfUrl) {  
 System.*out*.println("Origin page: " + element.getURL());  
  
 int count = 1;  
 for (URLDepthPair pair : listOfUrl) {  
 System.*out*.println(count + " | " + pair.toString());  
 count += 1;  
 }  
 }  
   
}

Листинг 2 – класс URLDeathPair.java

import java.net.MalformedURLException;  
import java.net.URL;  
  
public class URLDepthPair {  
   
 private String url;  
 private int depth;  
 public static final String *URL\_PREFIX* = "http://";  
 public static final int *MAX\_DEPTH\_VALUE* = 100;  
   
  
 public URLDepthPair(String url, int depth) throws MalformedURLException {  
  
   
 if (depth < 0 || depth > *MAX\_DEPTH\_VALUE*) {  
 throw new IllegalArgumentException("Error limits of depth");  
 }   
   
 if (!URLDepthPair.*isHttpPrefixInURL*(url)) {  
 MalformedURLException ex = new MalformedURLException("Error of url prefix");  
 throw ex;  
 }  
   
 this.url = url;  
 this.depth = depth;  
 }  
  
 public static boolean isHttpPrefixInURL(String url) {  
 if (!url.startsWith(*URL\_PREFIX*)) return false;  
 return true;  
 }  
   
  
 public static boolean isDepthAboveLimit(int depth) {  
 if (depth > *MAX\_DEPTH\_VALUE*) return true;  
 return false;  
 }  
   
  
 public String toString() {  
 return new String("[ " + this.url + ", " + this.depth + " ]");  
 }  
  
 public String getHostName() {  
 try {  
 URL url = new URL(this.url);  
 return url.getHost();  
 }  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
   
 public String getPagePath() {  
 try {  
 URL url = new URL(this.url);  
 return url.getPath();  
 }  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
   
 public String getURL() {  
 return this.url;  
 }  
   
 public int getDepth() {  
 return this.depth;  
 }  
   
 public void setDepth(int depth) {  
 if (depth < 0 || depth > *MAX\_DEPTH\_VALUE*) {  
 throw new IllegalArgumentException("Error limits of depth");  
 }   
 this.depth = depth;  
 }  
   
 public static String fullUrlStringToHostName(String in) {  
 try {  
 URL url = new URL(in);  
 return url.getHost();  
 }  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
   
 public static URL getUrlObjectFromUrlString(String urlStr) {  
 try {  
 URL url = new URL(urlStr);  
 return url;  
 }  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
}

Листинг 3 – класс CrawlerHelper.java

import java.lang.Exception;  
import java.util.\*;  
import java.net.MalformedURLException;  
import java.net.\*;  
import java.io.\*;  
  
public class CrawlerHelper {  
  
 public static String[] *formats* = {".html", ".pdf", ".java", ".xml", "txt", ".css", ".doc", ".c"};   
   
  
 public URLDepthPair getURLDepthPairFromArgs(String[] args) {  
 if (args.length > 2) System.*out*.println("Warning more than 2 parameters from command line!\n");  
 if (args.length < 2) {  
 System.*out*.println("Warning less than 2 parameters from command line!\n");  
 return null;  
 }  
  
 int depth;  
 try {  
 depth = Integer.*parseInt*(args[1]);  
 } catch (Exception e) {  
 System.*out*.println("Error depth parameter!");  
 return null;  
 }  
   
 URLDepthPair urlDepth;  
  
 try {  
 urlDepth = new URLDepthPair(args[0], depth);  
 }   
 catch (MalformedURLException ex) {  
 System.*out*.println(ex.getMessage() + "\n");  
 return null;  
 }   
 catch (IllegalArgumentException e) {  
 System.*out*.println(e.getMessage() + "\n");  
 return null;  
 }  
   
 return urlDepth;  
 }  
   
  
 public URLDepthPair getURLDepthPairFromInput() {  
  
 String url;  
 int depth;  
  
 String[] args;  
  
 URLDepthPair urlDepth = null;  
  
 Scanner in = new Scanner(System.*in*);  
  
 while (urlDepth == null) {  
  
 System.*out*.println("Enter URL and depth of parsing (in a line with a space between):");  
 String input = in.nextLine();  
  
 args = input.split(" ", 2);  
  
 urlDepth = this.getURLDepthPairFromArgs(args);  
 if (urlDepth == null) System.*out*.println("Try again!\n");  
 }  
 return urlDepth;   
 }  
  
 public static int getNumOfThreads(String[] args) {  
 if (args == null || args.length < 3) {  
 Scanner in = new Scanner(System.*in*);  
   
 boolean nice = false;  
 int input = Crawler.*NUM\_OF\_DEFAULT\_THREADS*;  
   
 while (!nice) {  
 try{  
 System.*out*.println("\nEnter amount of threads which you want to do parsing:");  
 input = in.nextInt();  
 if (input > 0 && input < 100) nice = true;  
 }  
 catch (Exception e) {  
 in.nextLine();  
 }  
 }  
 System.*out*.println("");  
 return input;  
 }   
 else {  
 int threads;  
 try {  
 threads = Integer.*parseInt*(args[2]);  
 if (threads > 0) return threads;  
 } catch (Exception e) {  
 System.*out*.println("Error threads-parameter in arguments. Using default amount!");  
 }  
 }  
 return Crawler.*NUM\_OF\_DEFAULT\_THREADS*;  
 }  
  
 public static String[] getInfoAboutUrl(URL url, boolean needToOut) {  
   
 String[] info = new String[10];  
 info[0] = url.toString();  
 info[1] = url.getHost();  
 try {  
 info[2] = url.getContent().toString();  
 } catch (IOException e) {  
 System.*out*.println("Cannot get content-type, this may be https protocol page");  
 info[2] = "";  
 }  
 info[3] = url.getProtocol();  
 info[4] = url.getPath();  
 info[5] = url.getUserInfo();  
 info[6] = url.getFile();  
 info[7] = url.getRef();  
 try {  
 info[8] = url.toURI().toString();  
 } catch (URISyntaxException e) {  
 System.*out*.println("Cannot get URI, this may be https protocol page");  
 info[8] = "";  
 }  
 info[9] = String.*valueOf*(url.getPort());  
   
 if (needToOut) {  
 System.*out*.println("\n---Info about this url---");  
 System.*out*.println("Full url: " + info[0]);  
 System.*out*.println("Host name of url: " + info[1]);  
 System.*out*.println("Content of url: " + info[2]);  
 System.*out*.println("Protocol of url: " + info[3]);  
 System.*out*.println("Path of url: " + info[4]);  
 System.*out*.println("UserInfo of url: " + info[5]);  
 System.*out*.println("Files on url: " + info[6]);  
 System.*out*.println("Ref of url: " + info[7]);  
 System.*out*.println("URI of url: " + info[8]);  
 System.*out*.println("Port of url: " + info[9]);  
 System.*out*.println("---------------------------\n");  
 }   
 return info;  
 }  
   
 public static String[] getInfoAboutUrl(String urlStr, boolean needToOut) {  
 URL url = null;  
 try {  
 url = new URL(urlStr);  
 }  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
   
 String[] info = *getInfoAboutUrl*(url, needToOut);   
 return info;  
   
 }  
  
 public static String getURLFromHTMLTag(String line) {  
 if (line.indexOf(Crawler.*HOOK\_REF*) == -1) return null;  
   
 int indexStart = line.indexOf(Crawler.*HOOK\_REF*) + Crawler.*HOOK\_REF*.length();  
 int indexEnd = line.indexOf("\"", indexStart);  
 // Если получится так, что кавычек больше нет  
 if (indexEnd == -1) return null;  
  
   
 return line.substring(indexStart, indexEnd);  
 }  
  
 public static String cutURLEndFormat(String url) {  
 url = CrawlerHelper.*cutTrashAfterFormat*(url);  
   
 for (String format : *formats*) {  
 if (url.endsWith(format)) {  
 int lastCatalog = url.lastIndexOf("/");  
 return url.substring(0, lastCatalog + 1);  
 }  
 }  
 return url;  
 }  
  
 public static String urlFromBackRef(String url, String backRef) {  
 int count = 2;  
 int index = url.length();  
  
 char[] urlSequnce = url.toCharArray();  
 while (count > 0 && index > 0) {  
 index -= 1;  
 if (urlSequnce[index] == '/') count -= 1;  
 }  
   
 if (index == 0) return null;  
   
 String cutURL = url.substring(0, index + 1);  
 String cutBackRef = backRef.substring(3, backRef.length());  
   
 return (cutURL + cutBackRef);  
 }  
  
 public static String cutTrashAfterFormat(String url) {  
 int index = url.lastIndexOf("#");  
 if (index == -1) return url;  
 return url.substring(0, index);  
   
 }  
}

Листинг 4 – класс CrawlerTask.java

import java.util.\*;  
  
public class CrawlerTask implements Runnable {  
   
  
 private URLDepthPair element;  
  
 private URLPool myPool;  
  
 public CrawlerTask(URLPool pool) {  
 this.myPool = pool;  
 }  
  
 public void run() {  
  
 element = myPool.get();  
  
 int myDepth = element.getDepth();  
  
 LinkedList<URLDepthPair> linksList = new LinkedList<URLDepthPair>();  
 linksList = Crawler.*parsePage*(element);  
   
 Crawler.*showResults*(element, linksList);  
   
 for (URLDepthPair pair: linksList) {  
 myPool.put(pair);  
 }  
   
 }  
}

Листинг 5 – класс URLPool.java

import java.util.\*;  
  
public class URLPool {  
   
 private LinkedList<URLDepthPair> watchedList;  
   
 private LinkedList<URLDepthPair> notWatchedList;  
   
 private LinkedList<URLDepthPair> blockedList;  
  
 public int waitingThreads;  
  
 private int depth;  
   
   
 public URLPool(int depth) {  
 waitingThreads = 0;  
 watchedList = new LinkedList<URLDepthPair>();  
 notWatchedList = new LinkedList<URLDepthPair>();  
 blockedList = new LinkedList<URLDepthPair>();  
 this.depth = depth;  
 }  
   
 public synchronized int getWaitThreads() {  
 return waitingThreads;  
 }  
  
 public synchronized boolean put(URLDepthPair depthPair) {  
   
 boolean added = false;  
  
 if (depthPair.getDepth() < this.depth) {  
 notWatchedList.addLast(depthPair);  
 added = true;  
  
 if (waitingThreads > 0) waitingThreads--;  
 this.notify();  
 }  
 else {  
 blockedList.add(depthPair);  
 }  
   
 return added;  
 }  
  
 public synchronized URLDepthPair get() {  
   
 URLDepthPair myDepthPair = null;  
  
 if (notWatchedList.size() == 0) {  
 waitingThreads++;  
 try {  
 this.wait();  
 }  
 catch (InterruptedException e) {  
 System.*err*.println("MalformedURLException: " + e.getMessage());  
 return null;  
 }  
 }  
 myDepthPair = notWatchedList.removeFirst();  
 watchedList.add(myDepthPair);  
   
 return myDepthPair;  
 }  
   
   
 public LinkedList<URLDepthPair> getWatchedList() {  
 return this.watchedList;  
 }  
   
 public LinkedList<URLDepthPair> getNotWatchedList() {  
 return this.notWatchedList;  
 }  
   
 public LinkedList<URLDepthPair> getBlockedList() {  
 return this.blockedList;  
 }  
   
}

## **2.2 Результаты работы программы.**

На рисунке 1 представлены операция компиляции файлов, а также запуск программы. На рисунке 2 представлен результат работы программы.

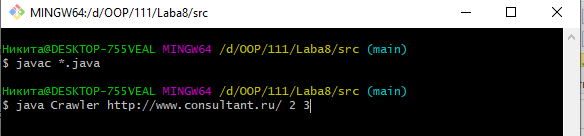


Рисунок 1 – Компиляция файлов и запуск программы.

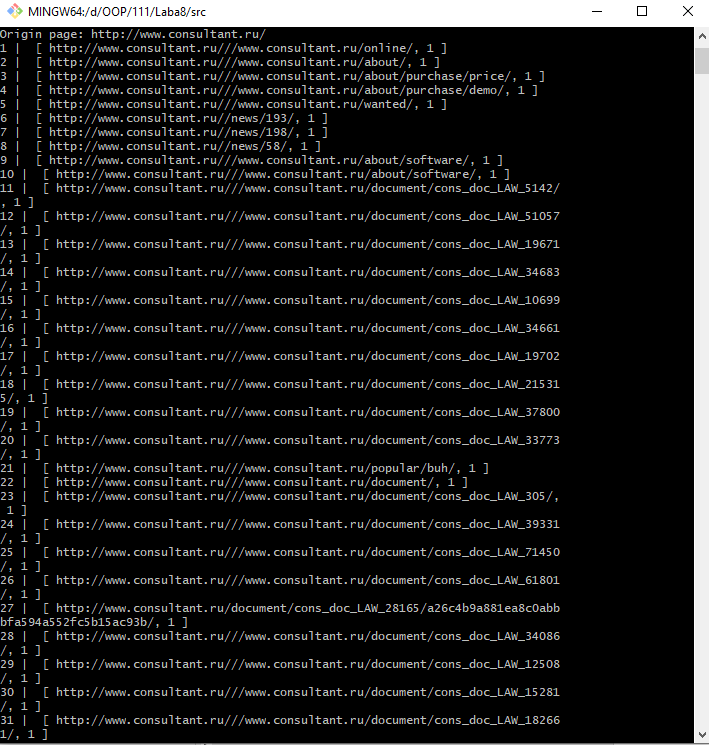


Рисунок 2 – Результат работы программы.

# **Список используемых источников**

1 ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления.

2 ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления.