**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Информатики»

**Отчет по лабораторной работе №8**

Выполнил: студент группы БВТ1901

Кускова А. Е.

Руководитель:

Мосева М. С.

Москва 2020

**Цель работы:**

Расширить сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно.

**Задание:**

1. Реализовать класс с именем URLPool, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов.
2. Создать класс CrawlerTask, который реализует интерфейс Runnable.
3. Изменить программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера.
4. Реализуйте метод wait() внутри «get URL» в случае, если ни один URL-адрес в настоящее время недоступен.

**Решение:**

1. Код для класса Crawler:

import java.util.\*;  
import java.io.\*;  
import java.net.\*;  
  
public class Crawler {  
 static int *numThreads* = 10;  
 static int *timeOut* = 1000;  
 public static void main(String[] args) {  
  
 UrlDepthPair currentDepthPair = new UrlDepthPair("http://ntsk.ru/");  
 ArrayList<String> seenURLs = new ArrayList<String>();  
 seenURLs.add(currentDepthPair.getURL());  
  
 UrlPool pool = new UrlPool(2);  
 pool.put(currentDepthPair);  
  
 int activeAmount = Thread.*activeCount*();  
  
 while (pool.getWaitThreads() != *numThreads*) {  
  
 if (Thread.*activeCount*() - activeAmount < *numThreads*) {  
 CrawlerTask crawler = new CrawlerTask(pool);  
 new Thread(crawler).start();  
 }  
 else {  
 try {  
 Thread.*sleep*(100);  
 }  
 catch (InterruptedException ie) {  
 System.*out*.println("Caught unexpected: InterruptedException, ignoring...");  
 }  
  
 }  
 }  
 for(String s : pool.seenURLs) {  
 System.*out*.println(s);  
 }  
  
 System.*out*.println(pool.seenURLs.size());  
 System.*exit*(0);  
  
 }  
  
 public static LinkedList<String> getAllLinks(UrlDepthPair myDepthPair) {  
  
 LinkedList<String> URLs = new LinkedList<String>();  
 Socket sock;  
  
 try {  
 sock = new Socket(myDepthPair.getWebHost(), 80);  
 }  
 catch (UnknownHostException e) {  
 System.*err*.println("UnknownHostException: " + e.getMessage());  
 return URLs;  
 }  
 catch (IOException ex) {  
 return URLs;  
 }  
  
 try {  
 sock.setSoTimeout(*timeOut*);  
 }  
 catch (SocketException exc) {  
 System.*err*.println("SocketException: " + exc.getMessage());  
 return URLs;  
 }  
  
 String docPath = myDepthPair.getDocPath();  
 String webHost = myDepthPair.getWebHost();  
  
 OutputStream outStream;  
  
 try {  
 outStream = sock.getOutputStream();  
 }  
 catch (IOException e) {  
 return URLs;  
 }  
  
 PrintWriter printWriter = new PrintWriter(outStream, true);  
 printWriter.println("GET " + docPath + " HTTP/1.1");  
 printWriter.println("Host: " + webHost);  
 printWriter.println("Connection: close");  
 printWriter.println();  
  
 InputStream inStream;  
 try {  
 inStream = sock.getInputStream();  
 }  
 catch (IOException ioException){  
 System.*err*.println("IOException: " + ioException.getMessage());  
 return URLs;  
 }  
 InputStreamReader inStreamReader = new InputStreamReader(inStream);  
 BufferedReader BuffReader = new BufferedReader(inStreamReader);  
  
 while (true) {  
 String line;  
 try {  
 line = BuffReader.readLine();  
 }  
 catch (IOException e) {  
 return URLs;  
 }  
 if (line == null)  
 break;  
 int beginIndex = 0;  
 int endIndex = 0;  
 int index = 0;  
  
 while (true) {  
 String START\_URL = "a href=\"";  
 String END\_URL = "\"";  
  
 index = line.indexOf(START\_URL, index);  
 if (index == -1)  
 break;  
 index += START\_URL.length();  
 beginIndex = index;  
  
 endIndex = line.indexOf(END\_URL, index);  
 index = endIndex;  
  
 try {  
 String newLink = line.substring(beginIndex, endIndex);  
 if(URLs.contains(newLink))  
 continue;  
  
 if(newLink.startsWith("http")) {  
 URLs.add(newLink);  
 }else if(!newLink.startsWith("tel")) {  
 if(newLink.startsWith("/"))  
 URLs.add("http://"+webHost+""+newLink);  
 else  
 URLs.add("http://"+webHost+"/"+newLink);  
 }  
  
 }catch(Exception exception) {  
 break;  
 }  
  
 }  
  
 }  
 return URLs;  
 }  
}

1. Код для класса UrlDepthPair:

import java.net.\*;  
public class UrlDepthPair {  
 private int depth;  
 private String url = "";  
  
 UrlDepthPair(String u, int d){  
 url = u;  
 depth = d;  
 }  
 UrlDepthPair(String u){  
 url = u;  
 depth = 0;  
 }  
  
 public String getURL() {  
 return url;  
 }  
 public int getDepth() {  
 return depth;  
 }  
 public String toString() {  
 return url + " : " + depth;  
 }  
  
 public String getDocPath() {  
 try {  
 URL tempURL = new URL(url);  
 return tempURL.getPath();  
 }  
 catch (MalformedURLException malformedURLException) {  
 System.*err*.println("MalformedURLException in getDocPath(): " + malformedURLException.getMessage());  
 return null;  
 }  
 }  
  
 public String getWebHost() {  
 try {  
 URL tempURL = new URL(url);  
 return tempURL.getHost();  
 }  
 catch (MalformedURLException malformedURLException) {  
 System.*err*.println("MalformedURLException in getWebHost: " + malformedURLException.getMessage());  
 return null;  
 }  
 }  
}

1. Код для класса CrawlerTask:

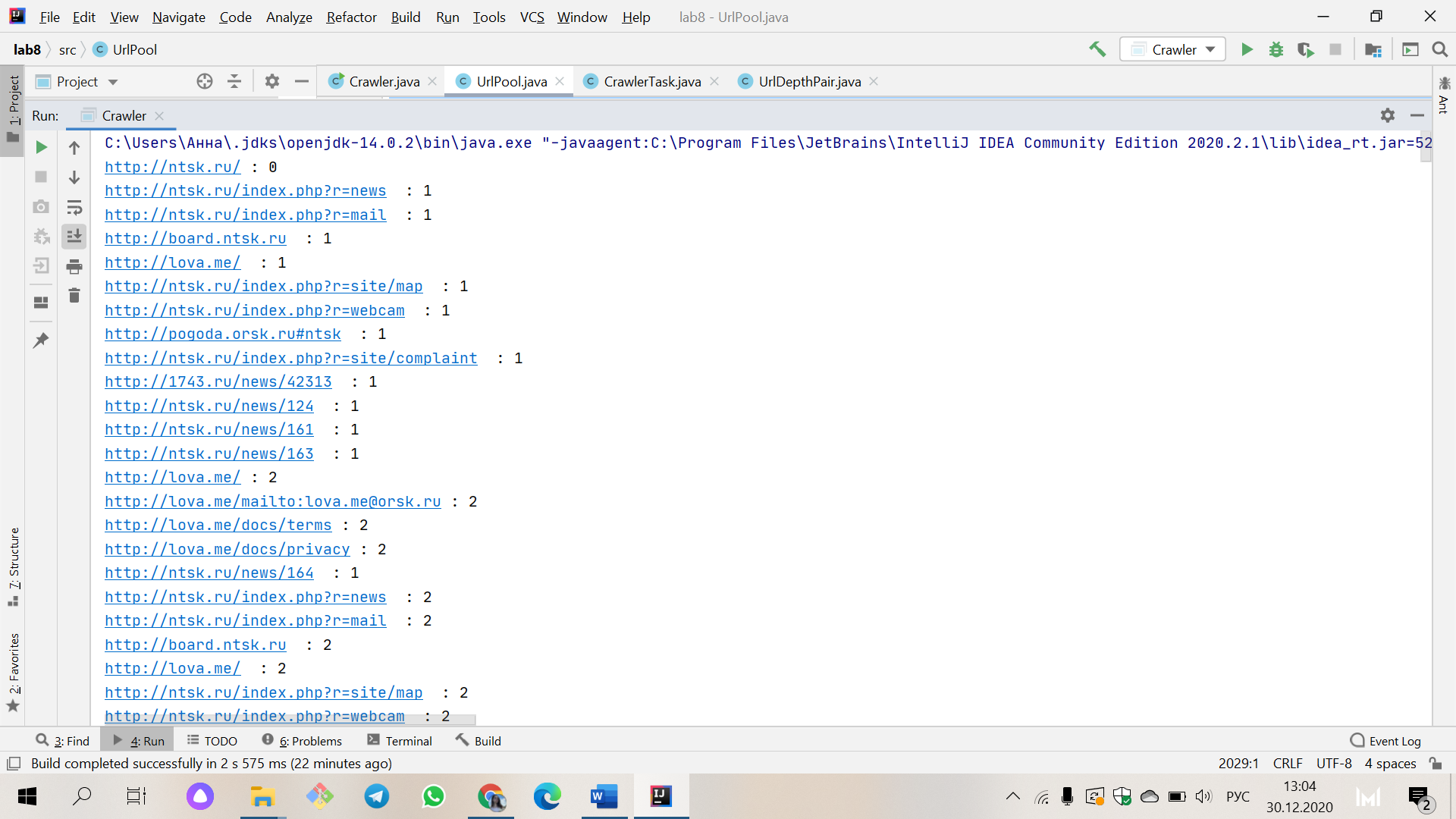
import java.util.\*;  
  
public class CrawlerTask implements Runnable {  
 public UrlDepthPair depthPair;  
  
 public UrlPool myPool;  
  
 public CrawlerTask(UrlPool pool) {  
 myPool = pool;  
 }  
  
 public void run() {  
 depthPair = myPool.get();  
  
 int myDepth = depthPair.getDepth();  
  
 LinkedList<String> linksList = new LinkedList<String>();  
 linksList = Crawler.*getAllLinks*(depthPair);  
  
 for (String newURL : linksList) {  
 UrlDepthPair newDepthPair = new UrlDepthPair(newURL, myDepth + 1);  
 myPool.put(newDepthPair);  
 }  
 }  
}

1. Код для класса UrlPool:

import java.util.\*;  
public class UrlPool {  
 public static int *max\_depth*;  
 private LinkedList<UrlDepthPair> pendingURLs;  
 public LinkedList<UrlDepthPair> processedURLs;  
 public ArrayList<String> seenURLs = new ArrayList<String>();  
  
 public int waitingThreads;  
  
 public UrlPool(int d) {  
 waitingThreads = 0;  
 pendingURLs = new LinkedList<UrlDepthPair>();  
 processedURLs = new LinkedList<UrlDepthPair>();  
 *max\_depth* = d;  
 }  
  
 public synchronized int getWaitThreads() {  
 return waitingThreads;  
 }  
  
 public synchronized int size() {  
 return pendingURLs.size();  
 }  
  
 public synchronized void decrimentWaitingThreads() {  
 waitingThreads--;  
 }  
  
 public synchronized boolean put(UrlDepthPair depthPair) {  
 boolean isAdded = false;  
  
 if (depthPair.getDepth() < *max\_depth* && !processedURLs.contains(depthPair.getURL())) {  
 pendingURLs.addLast(depthPair);  
 isAdded = true;  
 this.notify();  
 } else {  
 addSeenURL(depthPair);  
 }  
  
 return isAdded;  
 }  
  
 public synchronized UrlDepthPair get() {  
  
 UrlDepthPair depthPair = null;  
 if (pendingURLs.size() == 0) {  
 waitingThreads++;  
 try {  
 this.wait();  
 }  
 catch (InterruptedException interruptedException) {  
 System.*err*.println("InterruptedException: " + interruptedException.getMessage());  
 return null;  
 }  
 }  
 if(waitingThreads>0)waitingThreads--;  
 depthPair = pendingURLs.pop();  
 addSeenURL(depthPair);  
 processedURLs.add(depthPair);  
 return depthPair;  
 }  
  
 private synchronized void addSeenURL(UrlDepthPair dp) {  
 if(!seenURLs.contains(dp.toString()))  
 seenURLs.add(dp.toString());  
 }  
  
 public synchronized ArrayList<String> getSeenList() {  
 return seenURLs;  
 }  
}

**Результат работы программы**

На рисунке 1 изображен результат работы сканера:



**Рисунок 1** – Работа сканера.

**Вывод:**

Расширили веб-сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно.