Лабораторная работа №7

по дисциплине

“Объектно-ориентированное программирование”

Выполнил студент

группы БФИ1901

Бардюк Д.В.

Москва 2020

**Цель работы:** реализовать элементарный веб-сканер.

**Задание на лабораторную работу:**

1. Программа должна принимать в командной строке два параметра:

1) Строку, которая представляет собой URL-адрес, с которого можно начать просмотр страницы.

2) Положительное целое число, которое является максимальной глубиной поиска (см. ниже) Если указаны некорректные аргументы, программа должна немедленно остановиться и выдать сообщение об используемых аргументах, например: usage: java Crawler

2. Программа должна хранить URL-адрес в виде строки вместе с его глубиной (которая для начала будет равна 0). Вам будет необходимо создать класс для представления пар [URL, depth].

3. Программа должна подключиться к указанному сайту в URL-адресе на порт 80 с использованием сокета (см. ниже) и запросить указанную вебстраницу.

4. Программа должна проанализировать возвращаемый текст, построчно для любых подстрок, имеющих формат: Найденные URL-адреса должны быть сохранены в паре с новым значением глубины в LinkedList (URL, depth) (подробнее о LinkedLists см. ниже). Новое значение глубины должно быть больше, чем значение глубины URL-адреса, соответствующего анализируемой странице.

5. Далее программа должна закрыть соединение сокета с хостом.

6. Программа должна повторять шаги с 3 по 6 для каждого нового URL-адреса, если глубина, соответствующая URL-адресу, меньше максимальной. Обратите внимание, что при извлечении и поиске определенного URL-адреса глубина поиска увеличивается на 1. Если глубина URL-адреса достигает максимальной глубины (или больше), не извлекайте и не ищите эту веб-страницу.

7. Наконец, программа должна вывести все посещенные URLстраницы вместе с их глубиной поиска.

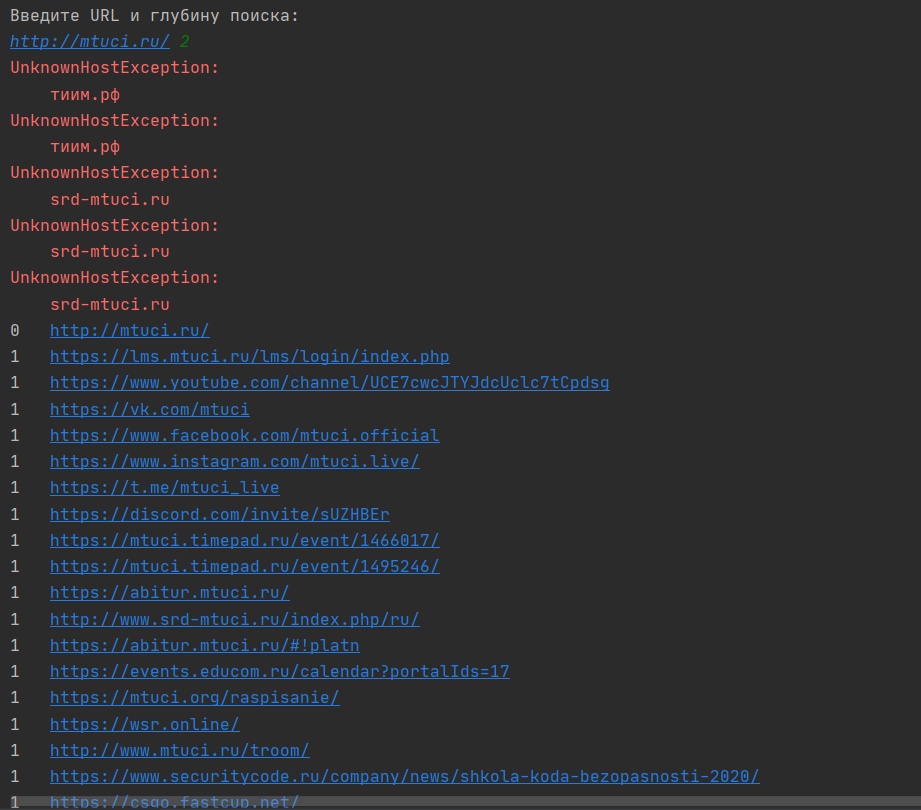
**Результат выполнения работы:**

import java.net. \*;  
public class UrlDepthPair {  
  
 /\* Поля для представления текущего URL и текущей глубины \*/  
 private final int currentDepth;  
 private final String currentURL;  
  
 /\* Конструктор, который устанавливает для ввода текущий URL-адрес и глубину\*/  
 public UrlDepthPair(String URL, int depth) {  
 currentDepth = depth;  
 currentURL = URL;  
 }  
  
 /\* Метод, возвращающий текущий URL \*/  
 public String getURL() {return currentURL;}  
  
 /\* Метод, возвращающий текущую глубину \*/  
 public int getDepth() {return currentDepth;}  
  
 /\* Метод, отображающий текущий URL и текущую глубину в строковом формате \*/  
 public String toString() {  
 String stringDepth = Integer.*toString*(currentDepth);  
 return stringDepth + '\t' + currentURL;  
 }  
  
 /\* Метод, который возвращает docPath текущего URL \*/  
 public String getDocPath() {  
  
 try {  
 URL url = new URL(currentURL);  
 return url.getPath();  
 }  
  
 catch (MalformedURLException e) {  
 System.*err*.println("MalformedURLException:\n\t" + e.getMessage());//если адрес не начинается с http://, выдается исключение  
 return null;  
 }  
  
 }  
  
 /\* Метод, который возвращает webHost текущего URL \*/  
 public String getWebHost() {  
  
 try {  
 URL url = new URL(currentURL);  
 return url.getHost();  
 }  
  
 catch (MalformedURLException exception) {  
 System.*err*.println("MalformedURLException:\n\t" + exception.getMessage());  
 return null;  
 }  
  
 }

import java.net.\*; //позволяет использование сокетов с помощью InputStream и OutputStream  
import java.util.\*; //для использования списков  
import java.io.\*; //для использования потоков  
  
public class Crawlers {  
 public static void main(String[] args) {  
  
 /\* Текущая глубина \*/  
 int depth = 0;  
 /\* Проверка правильности ввода \*/  
 if (args.length != 2) {  
 System.*out*.println("Введите данные в формате: <URL> <глубина поиска>");  
 System.*exit*(1);  
 }  
 else {  
 try {  
 depth = Integer.*parseInt*(args[1]);  
 }  
 catch (NumberFormatException nfe) {  
 System.*out*.println("Введите данные в формате: <URL> <глубина поиска>");  
 System.*exit*(1);  
 }  
 }  
 LinkedList<UrlDepthPair> pendingURLs = new LinkedList<>(); //для хранения пар  
 LinkedList<UrlDepthPair> processedURLs = new LinkedList<>();  
 /\* URLDepthPair, начать с него. Нулевая глубина и входная ссылка\*/  
 UrlDepthPair currentDepthPair = new UrlDepthPair(args[0], 0);  
 pendingURLs.add(currentDepthPair);  
 ArrayList<String> seenURLs = new ArrayList<>();  
 seenURLs.add(currentDepthPair.getURL());  
 /\* Перебор ожидающих URL-адресов, посещение каждого веб-сайта и получение всех ссылок со страницы \*/  
 while (pendingURLs.size() != 0) {  
 /\* Получите ожидающий URL-адрес, добавьте его в список обрабатываемых URL-адресов и сохраните его значение глубины\*/  
 UrlDepthPair depthPair = pendingURLs.pop();  
 processedURLs.add(depthPair);  
 int pendingURLDepth = depthPair.getDepth();  
 /\* Получить все ссылки со страницы и сохранить их как linkedList\*/  
 LinkedList<String> listOfLinks;  
 listOfLinks = Crawlers.*getAllLinks*(depthPair);  
 /\* Если не максимальная глубина, получить все ссылки, которые не были видны ранее, в списки \*/  
 if (pendingURLDepth < depth) {  
 for (String newURL : listOfLinks) {  
 if (!seenURLs.contains(newURL)) {  
 UrlDepthPair newDepthPair = new UrlDepthPair(newURL, pendingURLDepth + 1);  
 pendingURLs.add(newDepthPair);  
 seenURLs.add(newURL);  
 }  
 }  
 }  
 }  
 processedURLs.stream().distinct().forEach(System.*out*::println);  
 }  
 /\* Метод, который принимает URLDepthPair и возвращает <String> LinkedList  
 \* Берет ссылку из URLDepthPair, подключается к сайту, находит все ссылки  
 \* на сайте и добавляет их в новый возвращенный LinkedList  
 \*/  
 private static LinkedList<String> getAllLinks(UrlDepthPair myDepthPair) {  
 LinkedList<String> URLs = new LinkedList<>();  
 Socket socket;  
 try {  
 socket = new Socket(myDepthPair.getWebHost(), 80); //создание сокета из полученной строки с именем хоста  
 //и из целого числа с номером прорта - подключение к сайту на порт 80  
 }  
 catch (UnknownHostException e1) {  
 System.*err*.println("UnknownHostException:\n\t" + e1.getMessage());  
 return URLs;  
 }  
 catch (IOException e2) {  
 System.*err*.println("IOException:\n\t" + e2.getMessage());  
 return URLs;  
 }  
 /\* время ожидания сокета 3 секунды\*/  
 try {  
 socket.setSoTimeout(3000); //чтобы знал, сколько нужно ждать передачи данных с другой стороны  
 }  
 catch (SocketException e3) {  
 System.*err*.println("SocketException:\n\t" + e3.getMessage());  
 return URLs;  
 }  
 /\* Строки представляют собой URL-адрес docPath, а веб-хост \*/  
 String docPath = myDepthPair.getDocPath();  
 String webHost = myDepthPair.getWebHost();  
 OutputStream outputStream; //отправление данных на другую сторону соединения  
 try {  
 outputStream = socket.getOutputStream();  
 }  
 catch (IOException e4) {  
 System.*err*.println("IOException:\n\t" + e4.getMessage());  
 return URLs;  
 }  
 /\* PrintWriter очищается после каждого вывода\*/  
 //создание экземпляра PrintWriter из объекта outputStream  
 PrintWriter printWriter = new PrintWriter(outputStream, true); //autoFlush - очищение буфера вывода после каждого вызова метода println  
 //вызов метода println для отправки строки текста на другой конец соединения  
 printWriter.println("GET " + docPath + " HTTP/1.1");  
 printWriter.println("Host: " + webHost);  
 printWriter.println("Connection: close");  
 printWriter.println();  
 InputStream inputStream; //позволяет сокету получать данные с др. стороны соединения  
 try {  
 inputStream = socket.getInputStream();  
 }  
 catch (IOException e5){  
 System.*err*.println("IOException:\n\t" + e5.getMessage());  
 return URLs;  
 }  
 /\* InputStreamReader и BufferedReader для чтения строк из входного потока\*/  
 InputStreamReader inputStreamReader = new InputStreamReader(inputStream); //чтение символов из сокета  
 BufferedReader bufferedReader = new BufferedReader(inputStreamReader); //чтение целой строки  
 while (true) {  
 String line;  
 try {  
 line = bufferedReader.readLine();  
 }  
 catch (IOException e6) {  
 System.*err*.println("IOException:\n\t" + e6.getMessage());  
 return URLs;  
 }  
 if (line == null)  
 break;  
 int beginIndex;  
 int endIndex;  
 int index = 0;  
 while (true) {  
 /\* Константа для строки, указывающей ссылку \*/  
 String URL\_INDICATOR = "<a href=\"";  
 /\* Константа для строки, указывающей конец веб-хоста и начало пути к документации. \*/  
 String END\_URL = "\"";  
 index = line.indexOf(URL\_INDICATOR, index);  
 if (index == -1)  
 break;  
 index += URL\_INDICATOR.length();  
 beginIndex = index;  
 /\* Поиск символа конца в текущей строке и установление значения endIndex \*/  
 endIndex = line.indexOf(END\_URL, index);  
 index = endIndex;  
 if (index == -1){  
 break;  
 }  
 String newLink = line.substring(beginIndex, endIndex);  
 if (newLink.startsWith("http")){  
 URLs.add(newLink);  
 }  
 }  
  
 }  
 return URLs;  
 }  
  
}

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.println("Введите URL и глубину поиска: ");  
 String[] arg = in.nextLine().split(" ");  
 Crawlers.*main*(arg);  
 }  
}

**Результат выполнения программы**

****