**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №7: Веб-Сканер

по дисциплине «Кроссплатформенные технологии программирования»

В этой лабораторной работе вам необходимо будет реализовать элементарный веб-сканер.

Выполнил: студент

группы БСТ1803

Шевцов Никита

Вариант №24

Москва 2020

**Содержание**

[1 Задание на разработку программы 3](#_Toc53412181)

[2 Разработанные функции и классы 6](#_Toc53412182)

[2.1 Класс Crawler подкласс Runnable 6](#_Toc53412183)

[2.1.1 Метод private void set 6](#_Toc53412184)

[2.1.2 Метод private void grabAndParse 6](#_Toc53412185)

[2.1.3 Метод private double coefficient 6](#_Toc53412186)

[2.1.4 Метод void run 6](#_Toc53412187)

[2.2 Класс CrawlerWatcher 6](#_Toc53412188)

[2.2.1 Метод public void start 7](#_Toc53412189)

[2.3 Класс URLManager 7](#_Toc53412190)

[2.3.1 Метод public synchronized void addVisited 7](#_Toc53412191)

[2.3.2 Метод public synchronized void addLinks 7](#_Toc53412192)

[2.3.3 Метод public synchronized URLContainer getUnvisited 7](#_Toc53412193)

[2.3.4 Метод public synchronized boolean isEmpty 7](#_Toc53412194)

[2.4 Класс URLContainer 8](#_Toc53412195)

[2.4.1 Метод String toString 8](#_Toc53412196)

[2.4.2 Метод int getDepth 8](#_Toc53412197)

[2.4.3 Метод void setDepth 8](#_Toc53412198)

[2.4.4 Метод int getScore 8](#_Toc53412199)

[2.4.5 Метод void setScore 8](#_Toc53412200)

[2.4.6 Метод String getDomain 8](#_Toc53412201)

[2.4.7 Метод String getLocation 9](#_Toc53412202)

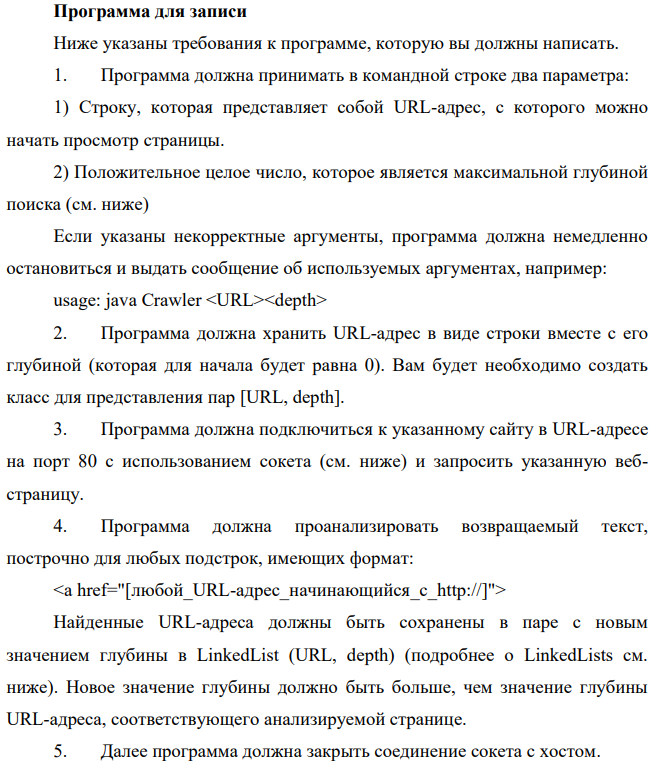
[2.5 Класс webScannerMain 9](#_Toc53412203)

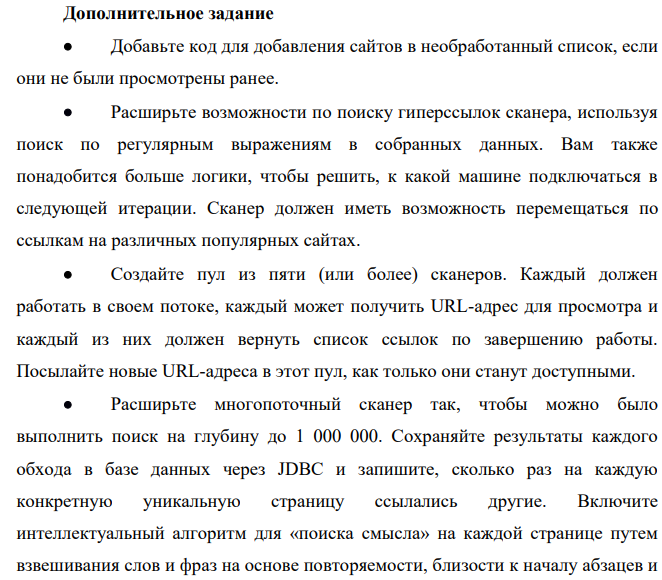
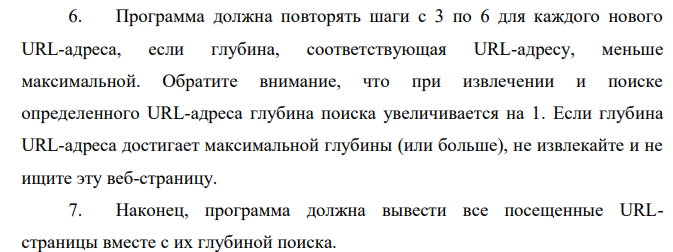
[2.5.1 Метод public static void main(String[] args) 9](#_Toc53412204)

[3 Код программы 9](#_Toc53412205)

[4 Результат работы программы 17](#_Toc53412206)

# Задание на разработку программы





# Разработанные функции и классы

## Класс Crawler подкласс Runnable

Класс реализующий сканер веб-страниц. Запускается в отдельном потоке.

* + 1. Метод private void set

Метод, устанавливающий URL для сканирования.

* + 1. Метод private void grabAndParse

Метод, осуществляющий подключение к странице, сбор данных с этой страницы и ее парсинг для присваивания счета.

* + 1. Метод private double coefficient

Метод, вычисляющий коэффициенты при присваивании счета.

* + 1. Метод void run

Метод, запускающий сканирование в потоке, пока для этого есть доступные адреса.

## Класс CrawlerWatcher

Класс, создающий нужное количество Crawler’ов, а также осуществляющий их первоначальную настройку и запуск.

* + 1. Метод public void start

Метод, запускающий сканирование.

## Класс URLManager

Класс, осуществляющий обработку результатов работы Crawler’ов

* + 1. Метод public synchronized void addVisited

Метод, выполняющий запись обработанного URL в базу данных

* + 1. Метод public synchronized void addLinks

Метод, добавляющий найденные Crawler’ом ссылки в пул не просмотренных адресов.

* + 1. Метод public synchronized URLContainer getUnvisited

Метод, возвращающий первый элемент из пула не просмотренных адресов.

* + 1. Метод public synchronized boolean isEmpty

Метод, проверяющий является ли пул адресов пустым.

* + 1. **Метод public synchronized int numURLs**

Метод, возвращающий количество адресов в пуле.

## Класс URLContainer

Класс, хранящий информацию о URL

* + 1. Метод String toString

Возвращает полный URL, записанный в контейнере

* + 1. Метод int getDepth

Возвращает глубину данного URL

* + 1. Метод void setDepth

Устанавливает глубину данного URL

* + 1. Метод int getScore

Возвращает счет данного URL

* + 1. Метод void setScore

Устанавливает счет данного URL

* + 1. Метод String getDomain

Возвращает только доменное имя URL

* + 1. Метод String getLocation

Возвращает ссылку без доменного имени

## Класс webScannerMain

Класс запускающий сканер Веб-Страниц

* + 1. Метод public static void main(String[] args)

Запускает сканирование по введенному адресу до введенной глубины поиска

# Код программы

package webScanner;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.MalformedURLException;

import java.net.Socket;

import java.util.LinkedList;

import java.util.regex.Matcher;

public class Crawler implements Runnable {

/\*\* Контейнер текущего URL \*/

private URLContainer current;

/\*\* Доменное имя сайта для подключения \*/

private String domain;

/\*\* Страница на сайте для забора данных \*/

private String location;

/\*\* Текущая глубина поиска \*/

private int currentDepth;

/\*\* Ссылки найденные на странице \*/

private LinkedList<URLContainer> foundURLs;

/\*\* Сокет для подключения к сайту \*/

private Socket io;

/\*\* Поле URLManager'а, к которому привязан Crawler \*/

private URLManager manager;

/\*\*

\* Конструктор класса - инициализирует необходимые поля и привязывает

\* Crawler к URLManager'у

\* @param mng - URLManager для привязки

\*/

public Crawler(URLManager mng) {

this.manager = mng;

this.foundURLs = new LinkedList<URLContainer>();

}

/\*\*

\* Подготавливает Crawler к работе над URL.

\* @param URL - Контейнер с информацией о URL.

\*/

private void set(URLContainer URL) {

this.current = URL;

this.domain = URL.getDomain();

this.location = URL.getLocation();

this.currentDepth = URL.getDepth();

this.foundURLs.clear();

}

@Override

public void run() {

this.set(manager.getUnvisited());

grabAndParse();

manager.addVisited(current);

manager.addLinks(foundURLs);

while (!manager.isEmpty()) {

this.set(manager.getUnvisited());

grabAndParse();

manager.addVisited(current);

manager.addLinks(foundURLs);

}

}

/\*\*

\* Подключается к URL, указанному в поле current, собирает с него ссылки и присваевает ему счет.

\*/

private void grabAndParse() {

try {

// Установить соединение с domainName по http

this.io = new Socket(this.domain,CrawlerWatcher.PORT);

this.io.setSoTimeout(CrawlerWatcher.TIMEOUT\_MS);

BufferedReader input = new BufferedReader(new InputStreamReader(io.getInputStream()));

PrintWriter output = new PrintWriter(io.getOutputStream(),true);

// Запросить с domainName страницу URL

output.println("GET " + this.location + " HTTP/1.1");;

output.println("Host: " + this.domain);

output.println("");

StringBuffer html = new StringBuffer();

String ln;

// Продолжить только в случае, если подключение успешно

if (input.readLine().endsWith("OK")) {

// Собрать весь документ в строку

while (!(ln = input.readLine()).equals("</body>")) {

html.append(ln + "\n");

}

int links = 0;

int images = 0;

Matcher tag = CrawlerWatcher.A\_TAG.matcher(html);

Matcher img = CrawlerWatcher.IMAGE\_TAG.matcher(html);

URLContainer found;

while (tag.find()) {

links += 1;

// Флаг нахождения ссылки в поле домена

boolean flag = false;

// Выбрать из совпадения группу с доменом

String link = tag.group(URLContainer.DOMAIN\_NAME\_GROUP);

// Обработка относительных ссылок

if (link == null) {

link = this.domain;

}

// Проверка ошибки в нахождении домена

flag = (link.endsWith(".html")|link.endsWith(".php")|link.endsWith(".htm"));

if (!flag) {

// Выбрать из совпадения группу с url

String link2 = tag.group(URLContainer.LOCATION\_GROUP);

// Сборка ссылки

if (!link2.startsWith("/")) {

if (!this.location.endsWith("/")) {

link2 = this.location + "/" + link2;

}

link2 = this.location + link2;

}

link = link + link2;

} else {

// Сборка ссылки

if (!link.startsWith("/")) {

if (!this.location.endsWith("/")) {

link = this.location + "/" + link;

}

link = this.location + link;

}

link = this.domain + link;

}

try {

// Упаковка ссылки в контейнер и добавление к листу найденных ссылок

found = new URLContainer(link);

found.setDepth(currentDepth + 1);

this.foundURLs.addLast(found);

} catch (MalformedURLException e) {}

}

// Подсчет картинок для определения счета страницы

while (img.find()) {

images += 1;

}

int pagesize = html.length();

int imgpart = (int) (((double)pagesize/4)\*coeficent((double)pagesize,(double)images,(double)CrawlerWatcher.CHAR\_PER\_IMG));

int lnkpart = (int) (((double)pagesize/4)\*coeficent((double)pagesize,(double)links,(double)CrawlerWatcher.CHAR\_PER\_LINK));

pagesize = pagesize/2 + imgpart + lnkpart;

this.current.setScore(pagesize);

}

io.close();

} catch (IOException e) {

}

}

/\*\*

\* Функция для получения счета страницы

\* @param size - размер страницы

\* @param param - количество элементов на странице

\* @param baseline - нужное кол-во символов страницы на один элемент

\* @return Коэффициент выполнения условия, от 1 до -0.5

\*/

private double coeficent(double size, double param, double baseline) {

double result = -Math.pow(param/(size/baseline), 2) + param/(size/(2\*baseline));

if (result < -0.5) {

result = -0.5;

}

return result;

}

}

package webScanner;

import java.util.LinkedList;

import java.util.regex.Pattern;

public class CrawlerWatcher {

/\*\* Строка для компиляции regex'а {@link #A\_TAG} \*/

public static final String A\_TAG\_STRING = "(?:<a.\*(?:href=\"" + URLContainer.DOMAIN\_NAME\_CHECK\_STRING + "?" + URLContainer.LOCATION\_CHECK\_STRING + "))";

/\*\* Для поиска в html документе тега с правильной ссылкой \*/

public static final Pattern A\_TAG = Pattern.compile(A\_TAG\_STRING);

/\*\* Для поиска картинок в документе \*/

public static final Pattern IMAGE\_TAG = Pattern.compile("(<img)");

/\*\* Кол-во символов на странице на одну картинку \*/

public static final int CHAR\_PER\_IMG = 20000;

/\*\* Кол-во символов на странице на одну ссылку \*/

public static final int CHAR\_PER\_LINK = 2000;

/\*\* Порт HTML \*/

public static final int PORT = 80;

/\*\* Время ожидания ответа \*/

public static final int TIMEOUT\_MS = 1000;

/\*\* Поле URLManager'а \*/

private URLManager manager;

/\*\* Поле кол-ва потоков \*/

private int threadNum;

/\*\* Лист всех Crawler'ов \*/

private LinkedList<Crawler> crawlers = new LinkedList<Crawler>();

/\*\* Лист всех потоков \*/

private LinkedList<Thread> threads = new LinkedList<Thread>();

/\*\*

\* Конструктор класса

\* @param start - URL, с которого начнется поиск

\* @param maxDepth - Максимальная глубина поиска

\* @param threads - Кол-во потоков для поиска

\* @throws IllegalArgumentException

\*/

public CrawlerWatcher(URLContainer start, int maxDepth, int threads) throws IllegalArgumentException {

if (threads < 1) {

throw new IllegalArgumentException("Кол-во потоков должно быть больше 0.");

}

this.threadNum = threads;

this.manager = new URLManager(maxDepth, start);

for (int i = 0; i < threads; i++) {

Crawler crawler = new Crawler(this.manager);

this.crawlers.add(crawler);

Thread thread = new Thread(crawler);

this.threads.add(thread);

}

}

/\*\*

\* Запуск поиска

\*/

public void start() {

for (Thread thread : threads) {

thread.start();

while (manager.numURLs() < this.threadNum) {}

}

}

}

package webScanner;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.HashSet;

import java.util.LinkedList;

public class URLManager {

public static final String DB = "jdbc:sqlite:" + System.getProperty("user.dir") + "\\saved\\results.db";

public static final String DRIVER = "org.sqlite.JDBC";

private int maxDepth;

private HashSet<String> visitedURLs;

private HashSet<String> visitedDomains;

private LinkedList<URLContainer> unvisited;

private Statement statement;

private Connection database;

public URLManager(int maxDepth, URLContainer start) {

try {

Class.forName(DRIVER);

} catch (ClassNotFoundException e1) {

System.out.print(e1.getMessage());

e1.printStackTrace();

}

this.maxDepth = maxDepth;

this.visitedURLs = new HashSet<String>();

this.visitedDomains = new HashSet<String>();

this.unvisited = new LinkedList<URLContainer>();

this.unvisited.addFirst(start);

try {

database = DriverManager.getConnection(DB);

this.statement = database.createStatement();

statement.executeUpdate("CREATE TABLE IF NOT EXISTS Results "

+ "(ID INTEGER PRIMARY KEY AUTOINCREMENT, "

+ "URL VARCHAR(350), "

+ "DEPTH INTEGER, "

+ "SCORE INTEGER);");

} catch (SQLException e) {

System.out.print(e.getMessage());

}

}

public synchronized void addVisited(URLContainer container) {

try {

this.statement.executeUpdate("INSERT INTO Results(URL, DEPTH, SCORE) VALUES('"

+ container.toString() + "',"

+ container.getDepth() + ","

+ container.getScore() + ");");

} catch (SQLException e) {

System.out.print(e.getMessage());

}

}

public synchronized void addLinks(LinkedList<URLContainer> list) {

for (URLContainer temp : list) {

if (temp.getDepth() < this.maxDepth) {

if (!this.visitedURLs.contains(temp.toString())) {

if (this.visitedDomains.contains(temp.getDomain())) {

this.visitedURLs.add(temp.toString());

this.unvisited.addLast(temp);

} else {

this.visitedURLs.add(temp.toString());

this.visitedDomains.add(temp.getDomain());

this.unvisited.addFirst(temp);

}

}

}

}

}

public synchronized URLContainer getUnvisited() {

return this.unvisited.pollFirst();

}

public synchronized boolean isEmpty() {

return this.unvisited.isEmpty();

}

public synchronized int numURLs() {

return this.unvisited.size();

}

}

package webScanner;

import java.net.MalformedURLException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class URLContainer {

/\*\* Строка для сборки Pattern'а поиска ссылки \*/

public static final String LOCATION\_CHECK\_STRING = "((?:\\/?[\\w-?=&%]+)\*(?:(?:\\.html?)|(?:\\.php))?)";

/\*\* Строка для сборки Pattern'a поиска доменного имени \*/

public static final String DOMAIN\_NAME\_CHECK\_STRING = "(?:(?:http:\\/\\/)?((?:[A-Za-zА-Яа-я0-9-.]+)(?:\\.[a-zа-я]{2,14})))";

/\*\* Pattern поиска ссылки \*/

public static final Pattern DOMAIN\_CHECK\_PATTERN = Pattern.compile("^(?!https:\\/\\/)" + DOMAIN\_NAME\_CHECK\_STRING + LOCATION\_CHECK\_STRING);

/\*\* Pattern поиска доменного имени \*/

public static final Pattern LOCATION\_CHECK\_PATTERN = Pattern.compile(LOCATION\_CHECK\_STRING);

/\*\* Группа домена в regex \*/

public static final int DOMAIN\_NAME\_GROUP = 1;

/\*\* Группа ссылки на домене в regex \*/

public static final int LOCATION\_GROUP = 2;

/\*\* Поле домена \*/

private String domain;

/\*\* Поле ссылки на домене \*/

private String location;

/\*\* Поле глубины URL \*/

private int depth;

/\*\* Поле счета URL \*/

private int score;

public URLContainer(String URL) throws MalformedURLException {

Matcher matchD = DOMAIN\_CHECK\_PATTERN.matcher(URL);

if (!matchD.find()) throw new MalformedURLException("Неправильный формат URL");

this.domain = matchD.group(DOMAIN\_NAME\_GROUP);

if (matchD.group(LOCATION\_GROUP).length() == 0) {

this.location = "/";

} else {

this.location = matchD.group(LOCATION\_GROUP);

}

}

public String toString() {

return this.domain + this.location;

}

public int getDepth() {

return this.depth;

}

public void setDepth(int depth) {

this.depth = depth;

}

public int getScore() {

return this.score;

}

public void setScore(int score) {

this.score = score;

}

public String getDomain() {

return this.domain;

}

public String getLocation() {

return this.location;

}

}

**package** webScanner;

**public** **class** webScannerMain {

**private** **int** maxDepth;

**private** URLContainer start;

webScannerMain(String URL, **int** MAX\_DEPTH) {

**try** {

**this**.start = **new** URLContainer(URL);

**this**.start.setDepth(0);

**this**.maxDepth = MAX\_DEPTH;

**if** (**this**.maxDepth < 1) **throw** **new** Exception("Глубина поиска дожна быть больше 0");

} **catch** (Exception e) {

System.***out***.print(e.getMessage());

}

}

**public** **static** **void** main(String[] args) {

webScannerMain scanner = **new** webScannerMain(args[0],Integer.*valueOf*(args[1]));

CrawlerWatcher watcher = **new** CrawlerWatcher(scanner.start,scanner.maxDepth,Integer.*valueOf*(args[2]));

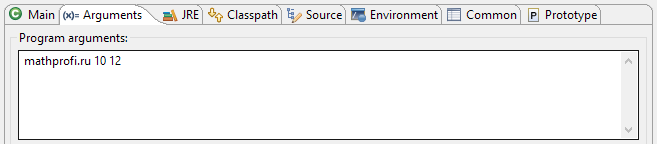
watcher.start();

}

}

# Результат работы программы

Введенные как входные данные аргументы



Некоторая часть записей в базу данных

